KLS

Vishwanathrao Deshpande Institute of Technology

Haliyal – 581329

ARTIFICIAL INTELLIGENCE [BAD402]

for

IV Semester B.E.

As prescribed by

VISVESVARAYA TECHNOLOGICAL UNIVERSITY,

BELAGAVI – 590014

(From Academic Year: 2025-26)

Prepared by

Prof. Narasimha Dixit

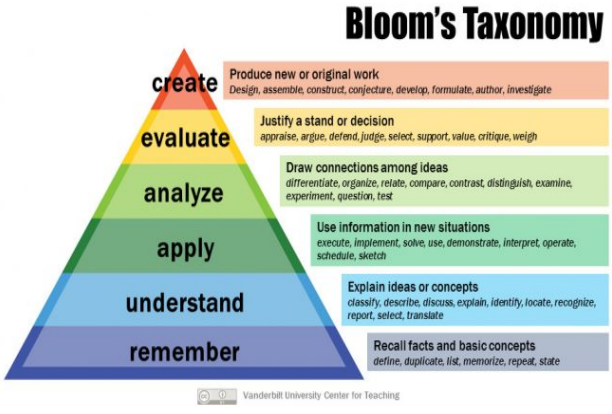
Department of Computer Science & Engineering (AIML)

Index

PROGRAM OUTCOMES(POs)

Program Outcomes as defined by NBA (PO) Engineering Graduates will be able to:

1. Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.



Vision (College)
To nurture talent and enrich society through excellence in technical education, research and innovation.
Mission (College)
1. To augment innovative Pedagogy, kindle quest for interdisciplinary learning & to enhance conceptual understanding. 2. To build competence, professional ethics & develop entrepreneurial thinking. 3. To strengthen Industry Institute Partnership & explore global collaborations. 4. To inculcate culture of socially responsible citizenship. 5. To focus on Holistic & Sustainable development.
Vision (Dept)
To lead the way in the creation and application of cutting-edge Computer science and engineering (AIML) technologies, advancing the frontiers of knowledge, and empowering future generations to drive innovation and transformation on a global scale.
Mission (Dept)
To train students with a strong conceptual understanding using innovative pedagogies, empowering them to excel in the dynamic fields of Artificial Intelligence and Machine Learning. To imbibe professional, research, and entrepreneurial skills with a commitment to the nation's development at large. To strengthen the industry-institute Interaction. To promote life–long learning with a sense of societal & ethical responsibilities.

Program Educational Objectives (PEO)	
PEO1	To develop an ability to identify and analyze the requirements of Computer Science and Engineering in design and providing novel engineering solutions.
PEO2	To develop abilities to work in team on multidisciplinary projects with effective communication skills, ethical qualities and leadership roles.
PEO3	To develop abilities for successful Computer Science Engineer and achieve higher career goals.
Program Specific Outcomes (PSO)	
PSO 1	To develop abilities to model real world problems using appropriate algorithms, computational theories and programming languages in the area of AI and ML..
PSO 2	To develop software applications and products in specialized areas of Artificial Intelligence and Machine Learning.

Mapping of Experiments with CO, PO and PSO

S.No.	Experiment Details	CO	PO	PSO
	PART A			
1	Implement and Demonstrate Depth First Search Algorithm on Water Jug Problem	CO2	2	1
2	Implement and Demonstrate Best First Search Algorithm on Missionaries-Cannibals Problems using Python	CO2	2	1
3	Implement A* Search algorithm	CO2	1	1
4	Implement AO* Search algorithm	CO2	2	1
5	Solve 8-Queens Problem with suitable assumptions	CO2	2	1
6	Implementation of TSP using heuristic approach	CO2	2	1
7	Implementation of the problem solving strategies: either using Forward Chaining or Backward Chaining	C03	1	2
8	Implement resolution principle on FOPL related problems	C04	2	2
9	Implement Tic-Tac-Toe game using Python	C04	2	2
10	Build a bot which provides all the information related to text in search box	CO5	2	1
11	Implement any Game and demonstrate the Game playing strategies	CO5	2	1

1. Implement and Demonstrate Depth First Search Algorithm on Water Jug Problem

```
from collections import deque

def is_valid_state(state, jug1_capacity, jug2_capacity):
    """Checks if the state is within valid jug capacities."""
    jug1, jug2 = state
    return 0 <= jug1 <= jug1_capacity and 0 <= jug2 <= jug2_capacity

def get_next_states(state, jug1_capacity, jug2_capacity):
    """Generates all possible states from the current state."""
    jug1, jug2 = state
    possible_states = set()

    # Fill Jug1
    possible_states.add((jug1_capacity, jug2))
    # Fill Jug2
    possible_states.add((jug1, jug2_capacity))
    # Empty Jug1
    possible_states.add((0, jug2))
    # Empty Jug2
    possible_states.add((jug1, 0))

    # Pour Jug1 -> Jug2
    pour = min(jug1, jug2_capacity - jug2)
    possible_states.add((jug1 - pour, jug2 + pour))

    # Pour Jug2 -> Jug1
    pour = min(jug2, jug1_capacity - jug1)
    possible_states.add((jug1 + pour, jug2 - pour))

    return [state for state in possible_states if is_valid_state(state, jug1_capacity, jug2_capacity)]

def dfs(jug1_capacity, jug2_capacity, target):
    """Performs DFS to find a sequence of moves to reach the target volume."""
    stack = deque()
    stack.append(((0, 0), []))  # (state, path)
    visited = set()

    while stack:
        state, path = stack.pop()

        if state in visited:
            continue

        visited.add(state)
        path = path + [state]
```

```
        jug1, jug2 = state
        if jug1 == target or jug2 == target:
            return path

        for next_state in get_next_states(state, jug1_capacity, jug2_capacity):
            if next_state not in visited:
                stack.append((next_state, path))

    return None  # No solution found

# Example Usage
jug1_capacity = 4
jug2_capacity = 3
target = 2
solution = dfs(jug1_capacity, jug2_capacity, target)

if solution:
    print("Solution Path:")
    for state in solution:
        print(state)
else:
    print("No solution found.")
```

Output

Solution Path:
(0, 0)
(0, 3)
(4, 3)
(4, 0)
(1, 3)
(1, 0)
(0, 1)
(4, 1)
(2, 3)

2. Implement and Demonstrate Best First Search Algorithm on Missionaries-Cannibals Problems using Python

```
from queue import PriorityQueue

class State:
    def __init__(self, missionaries, cannibals, boat, depth=0):
        self.missionaries = missionaries
        self.cannibals = cannibals
        self.boat = boat
        self.depth = depth  # Track depth for search

    def __repr__(self):
        return f"({self.missionaries}, {self.cannibals}, {self.boat})"

    def is_valid(self):
        """Check if the state is valid (no rule violation)."""
        if (self.missionaries < self.cannibals and self.missionaries > 0) or \
           (3 - self.missionaries < 3 - self.cannibals and 3 - self.missionaries > 0):
            return False
        return 0 <= self.missionaries <= 3 and 0 <= self.cannibals <= 3

    def is_goal(self):
        """Check if this is the goal state."""
        return self.missionaries == 0 and self.cannibals == 0 and self.boat == 0

    def heuristic(self):
        """Heuristic function: Number of people left on the original side."""
        return self.missionaries + self.cannibals

    def generate_successors(self):
        """Generate valid successor states."""
        successors = []
        moves = [(1, 0), (2, 0), (0, 1), (0, 2), (1, 1)]  # Possible moves
        direction = -1 if self.boat else 1  # Determine movement direction

        for m, c in moves:
            new_state = State(self.missionaries + direction * m,
                              self.cannibals + direction * c,
                              self.boat + direction * 1,
                              self.depth + 1)
            if new_state.is_valid():
                successors.append(new_state)
        return successors

def best_first_search():
    """Implement Best First Search for Missionaries and Cannibals."""
    initial_state = State(3, 3, 1)  # Start state: 3 missionaries, 3 cannibals, boat present
```

```
    pq = PriorityQueue()
    pq.put((initial_state.heuristic(), 0, initial_state))  # (heuristic, depth, state)
    visited = set()

    while not pq.empty():
        _, _, current_state = pq.get()
        if current_state.is_goal():
            print("\nSolution Found!")
            return current_state

        if (current_state.missionaries, current_state.cannibals, current_state.boat) in visited:
            continue

        visited.add((current_state.missionaries, current_state.cannibals, current_state.boat))

        print(f"Expanding: {current_state}")

        for successor in current_state.generate_successors():
            if (successor.missionaries, successor.cannibals, successor.boat) not in visited:
                pq.put((successor.heuristic(), successor.depth, successor))

    print("No solution found.")
    return None

# Run the search algorithm
best_first_search()
```

Output:

```
Expanding: (3, 3, 1)
Expanding: (2, 2, 0)
Expanding: (3, 2, 1)
Expanding: (1, 1, 0)
Expanding: (2, 1, 1)
Expanding: (0, 0, 0)

Solution Found!
```

3. Python program for A* Search Algorithm

```
import math
import heapq

# Define the Cell class


class Cell:
    def __init__(self):
      # Parent cell's row index
        self.parent_i = 0
    # Parent cell's column index
        self.parent_j = 0
 # Total cost of the cell (g + h)
        self.f = float('inf')
    # Cost from start to this cell
        self.g = float('inf')
    # Heuristic cost from this cell to destination
        self.h = 0


# Define the size of the grid
ROW = 9
COL = 10

# Check if a cell is valid (within the grid)


def is_valid(row, col):
    return (row >= 0) and (row < ROW) and (col >= 0) and (col < COL)

# Check if a cell is unblocked


def is_unblocked(grid, row, col):
    return grid[row][col] == 1

# Check if a cell is the destination


def is_destination(row, col, dest):
    return row == dest[0] and col == dest[1]

# Calculate the heuristic value of a cell (Euclidean distance to destination)


def calculate_h_value(row, col, dest):
    return ((row - dest[0]) ** 2 + (col - dest[1]) ** 2) ** 0.5
```

```
# Trace the path from source to destination


def trace_path(cell_details, dest):
    print("The Path is ")
    path = []
    row = dest[0]
    col = dest[1]

    # Trace the path from destination to source using parent cells
    while not (cell_details[row][col].parent_i == row and cell_details[row][col].parent_j == col):
        path.append((row, col))
        temp_row = cell_details[row][col].parent_i
        temp_col = cell_details[row][col].parent_j
        row = temp_row
        col = temp_col

    # Add the source cell to the path
    path.append((row, col))
    # Reverse the path to get the path from source to destination
    path.reverse()

    # Print the path
    for i in path:
        print("->", i, end=" ")
    print()

# Implement the A* search algorithm


def a_star_search(grid, src, dest):
    # Check if the source and destination are valid
    if not is_valid(src[0], src[1]) or not is_valid(dest[0], dest[1]):
        print("Source or destination is invalid")
        return

    # Check if the source and destination are unblocked
    if not is_unblocked(grid, src[0], src[1]) or not is_unblocked(grid, dest[0], dest[1]):
        print("Source or the destination is blocked")
        return

    # Check if we are already at the destination
    if is_destination(src[0], src[1], dest):
        print("We are already at the destination")
        return

    # Initialize the closed list (visited cells)
```

```
    closed_list = [[False for _ in range(COL)] for _ in range(ROW)]
    # Initialize the details of each cell
    cell_details = [[Cell() for _ in range(COL)] for _ in range(ROW)]

    # Initialize the start cell details
    i = src[0]
    j = src[1]
    cell_details[i][j].f = 0
    cell_details[i][j].g = 0
    cell_details[i][j].h = 0
    cell_details[i][j].parent_i = i
    cell_details[i][j].parent_j = j

    # Initialize the open list (cells to be visited) with the start cell
    open_list = []
    heapq.heappush(open_list, (0.0, i, j))

    # Initialize the flag for whether destination is found
    found_dest = False

    # Main loop of A* search algorithm
    while len(open_list) > 0:
        # Pop the cell with the smallest f value from the open list
        p = heapq.heappop(open_list)

        # Mark the cell as visited
        i = p[1]
        j = p[2]
        closed_list[i][j] = True

        # For each direction, check the successors
        directions = [(0, 1), (0, -1), (1, 0), (-1, 0),
                      (1, 1), (1, -1), (-1, 1), (-1, -1)]
        for dir in directions:
            new_i = i + dir[0]
            new_j = j + dir[1]

            # If the successor is valid, unblocked, and not visited
            if is_valid(new_i, new_j) and is_unblocked(grid, new_i, new_j) and not
closed_list[new_i][new_j]:
                # If the successor is the destination
                if is_destination(new_i, new_j, dest):
                    # Set the parent of the destination cell
                    cell_details[new_i][new_j].parent_i = i
                    cell_details[new_i][new_j].parent_j = j
                    print("The destination cell is found")
                    # Trace and print the path from source to destination
                    trace_path(cell_details, dest)
```

```
                found_dest = True
                return
            else:
                # Calculate the new f, g, and h values
                g_new = cell_details[i][j].g + 1.0
                h_new = calculate_h_value(new_i, new_j, dest)
                f_new = g_new + h_new

                # If the cell is not in the open list or the new f value is smaller
                if cell_details[new_i][new_j].f == float('inf') or cell_details[new_i][new_j].f > f_new:
                    # Add the cell to the open list
                    heapq.heappush(open_list, (f_new, new_i, new_j))
                    # Update the cell details
                    cell_details[new_i][new_j].f = f_new
                    cell_details[new_i][new_j].g = g_new
                    cell_details[new_i][new_j].h = h_new
                    cell_details[new_i][new_j].parent_i = i
                    cell_details[new_i][new_j].parent_j = j

    # If the destination is not found after visiting all cells
    if not found_dest:
        print("Failed to find the destination cell")

# Driver Code


def main():
    # Define the grid (1 for unblocked, 0 for blocked)
    grid = [
        [1, 0, 1, 1, 1, 1, 0, 1, 1, 1],
        [1, 1, 1, 0, 1, 1, 1, 0, 1, 1],
        [1, 1, 1, 0, 1, 1, 0, 1, 0, 1],
        [0, 0, 1, 0, 1, 0, 0, 0, 0, 1],
        [1, 1, 1, 0, 1, 1, 1, 0, 1, 0],
        [1, 0, 1, 1, 1, 1, 0, 1, 0, 0],
        [1, 0, 0, 0, 0, 1, 0, 0, 0, 1],
        [1, 0, 1, 1, 1, 1, 0, 1, 1, 1],
        [1, 1, 1, 0, 0, 0, 1, 0, 0, 1]
    ]

    # Define the source and destination
    src = [8, 0]
    dest = [0, 0]

    # Run the A* search algorithm
    a_star_search(grid, src, dest)
```

```
if __name__ == "__main__":
    main()
```

4. Implement AO* Search algorithm

```
import heapq
class Node:
    def __init__(self, name, is_and_node, heuristic=0):
        self.name = name
        self.is_and_node = is_and_node  # True if AND node, False if OR node
        self.heuristic = heuristic
        self.children = []
        self.parent = None
        self.g = float('inf')  # Cost from start node
        self.f = float('inf')  # Total estimated cost (g + heuristic)

    def __repr__(self):
        return f"Node({self.name})"

    # Add comparison methods for Node objects based on their 'f' value
    def __lt__(self, other):
        return self.f < other.f

    def __eq__(self, other):
        return self.f == other.f

class AOStarSearch:
    def __init__(self, start_node):
        self.start_node = start_node

    def compute_path(self, node):
        if node.parent is None:
            return [node.name]
        return self.compute_path(node.parent) + [node.name]

    def a_star_search(self):
        # Priority queue to manage the open list
        open_list = []
        self.start_node.g = 0
        self.start_node.f = self.start_node.heuristic
        heapq.heappush(open_list, (self.start_node.f, self.start_node))

        while open_list:
            # Get the node with the lowest f score
            _, current_node = heapq.heappop(open_list)

            # Check if we have reached a solution (no children left to expand)
            if not current_node.children:
                print(f"Solution found: {self.compute_path(current_node)} with cost {current_node.g}")
                return
```

```
            # If it is an AND node, we must solve all its children
            if current_node.is_and_node:
                current_node.g = 0  # Reset the cost for AND node
                for child in current_node.children:
                    child.g = float('inf')  # Reset cost for children
                    child.parent = current_node

                    # For AND nodes, calculate the cost recursively
                    child.f = child.g + child.heuristic
                    heapq.heappush(open_list, (child.f, child))

            # If it's an OR node, we choose the best solution from its children
            elif not current_node.is_and_node:
                best_child = None
                best_f = float('inf')
                for child in current_node.children:
                    child.g = float('inf')  # Reset cost for each child
                    child.parent = current_node
                    child.f = child.g + child.heuristic
                    heapq.heappush(open_list, (child.f, child))

        print("No solution found.")

# Example of usage:
if __name__ == "__main__":
    # Create nodes
    start = Node('Start', False, heuristic=5)
    goal = Node('Goal', True, heuristic=0)
    node1 = Node('Node1', False, heuristic=3)
    node2 = Node('Node2', False, heuristic=1)
    node3 = Node('Node3', True, heuristic=2)
    node4 = Node('Node4', False, heuristic=1)

    # Set up connections (children)
    start.children = [node1, node2]
    node1.children = [node3]
    node2.children = [node4]
    node3.children = []  # Goal node has no children
    node4.children = []  # Goal node has no children

    # Perform AO* search
    ao_star_search = AOStarSearch(start)
    ao_star_search.a_star_search()
```

output

```
Solution found: ['Start', 'Node1', 'Node3'] with cost inf
```

5. Solve 8-Queens Problem with suitable assumptions

```
def is_safe(board, row, col):
    # Check the column for any other queen
    for i in range(row):
        if board[i] == col or \
           board[i] - i == col - row or \
           board[i] + i == col + row:
            return False
    return True

def solve_8_queens(board, row):
    # Base case: If all queens are placed, return True
    if row == 8:
        return True

    # Try placing a queen in all columns for this row
    for col in range(8):
        if is_safe(board, row, col):
            board[row] = col  # Place the queen
            if solve_8_queens(board, row + 1):  # Recur for the next row
                return True
            # Backtrack if placing queen doesn't lead to a solution
            board[row] = -1

    # If no column works, return False
    return False

def print_solution(board):
    # Print the board in a human-readable form
    for row in range(8):
        line = ['Q' if board[row] == col else '.' for col in range(8)]
        print(' '.join(line))

def solve():
    # Initialize the board (all entries are -1, meaning no queens are placed)
    board = [-1] * 8

    # Start solving from the first row
    if solve_8_queens(board, 0):
        print_solution(board)
    else:
        print("No solution exists")

# Run the solver
solve()
```

output

```
Q . . . . . . .
. . . . Q . . .
. . . . . . . Q
. . . . . Q . .
. . Q . . . . .
. . . . . . Q .
. Q . . . . . .
. . . Q . . . .
```

6. Implementation of TSP using heuristic approach

```
import numpy as np

# Function to calculate the distance between two cities
def calculate_distance(city1, city2):
    return np.linalg.norm(np.array(city1) - np.array(city2))

# Nearest Neighbor Heuristic for solving TSP
def nearest_neighbor_heuristic(cities):
    n = len(cities)

    # List to keep track of visited cities
    visited = [False] * n
    # Start from the first city (arbitrary choice)
    current_city = 0
    visited[current_city] = True
    tour = [current_city]  # Start the tour with the first city
    total_distance = 0

    # Visit all cities
    for _ in range(n - 1):
        nearest_city = None
        nearest_distance = float('inf')

        # Find the nearest unvisited city
        for i in range(n):
            if not visited[i]:
                distance = calculate_distance(cities[current_city], cities[i])
                if distance < nearest_distance:
                    nearest_distance = distance
                    nearest_city = i

        # Move to the nearest city
        tour.append(nearest_city)
        total_distance += nearest_distance
        visited[nearest_city] = True
        current_city = nearest_city

    # Add the distance to return to the starting city
    total_distance += calculate_distance(cities[current_city], cities[tour[0]])

    return tour, total_distance

# Function to print the tour and its distance
def print_tour(tour, total_distance, cities):
    print("Tour:", " -> ".join(str(city) for city in tour))
    print(f"Total Distance: {total_distance:.2f}")
```

```
    # Print the coordinates of the cities in the tour
    print("Coordinates of the cities in the tour:")
    for city in tour:
        print(f"City {city}: {cities[city]}")

# Example usage of the TSP heuristic approach
if __name__ == "__main__":
    # Example: Set of cities represented by (x, y) coordinates
    cities = [
        (0, 0),   # City 0
        (1, 2),   # City 1
        (2, 4),   # City 2
        (3, 1),   # City 3
        (5, 3),   # City 4
        (6, 5),   # City 5
    ]

    # Solve TSP using the Nearest Neighbor Heuristic
    tour, total_distance = nearest_neighbor_heuristic(cities)

    # Print the tour and its total distance
    print_tour(tour, total_distance, cities)
```

output:

```
Tour: 0 -> 1 -> 2 -> 3 -> 4 -> 5
Total Distance: 20.51
Coordinates of the cities in the tour:
City 0: (0, 0)
City 1: (1, 2)
City 2: (2, 4)
City 3: (3, 1)
City 4: (5, 3)
City 5: (6, 5)
```

7. Implementation of the problem solving strategies: either using Forward Chaining or Backward Chaining

Forward Chaining

```
class ForwardChaining:
    def __init__(self, rules, facts):
        self.rules = rules  # List of rules (conditions → conclusion)
        self.facts = set(facts)  # Known facts

    def apply_rules(self):
        new_facts = True
        while new_facts:
            new_facts = False
            for rule in self.rules:
                conditions, conclusion = rule
                if conditions.issubset(self.facts) and conclusion not in self.facts:
                    print(f"Applying rule: {conditions} → {conclusion}")
                    self.facts.add(conclusion)
                    new_facts = True

    def check_goal(self, goal):
        return goal in self.facts

# Define rules (IF conditions THEN conclusion)
rules = [
    ({"Has_Job", "Good_Credit"}, "Eligible_For_Loan"),
    ({"No_Debt"}, "Good_Credit"),
    ({"High_Salary"}, "Has_Job"),
]

# Initial known facts
facts = {"No_Debt", "High_Salary"}

# Goal to achieve
goal = "Eligible_For_Loan"

# Create Forward Chaining System
fc_system = ForwardChaining(rules, facts)

# Apply rules iteratively
fc_system.apply_rules()

# Check if goal is achieved
if fc_system.check_goal(goal):
    print(f"Goal '{goal}' is achieved!")
else:
    print(f"Goal '{goal}' is NOT achieved.")
```

Output:

Applying rule: {'No_Debt'} → Good_Credit
Applying rule: {'High_Salary'} → Has_Job
Applying rule: {'Has_Job', 'Good_Credit'} → Eligible_For_Loan
Goal 'Eligible_For_Loan' is achieved!

Backward Chaining

```
class BackwardChaining:
    def __init__(self, rules, facts):
        self.rules = rules  # List of rules (conditions → conclusion)
        self.facts = set(facts)  # Known facts

    def infer(self, goal):
        # If goal is already in known facts, return True
        if goal in self.facts:
            return True

        # Check if any rule leads to the goal
        for conditions, conclusion in self.rules:
            if conclusion == goal:
                print(f"Checking rule: {conditions} → {goal}")
                if all(self.infer(condition) for condition in conditions):
                    self.facts.add(goal)  # Store inferred fact
                    return True

        return False  # Goal not achieved

# Define rules (IF conditions THEN conclusion)
rules = [
    ({"Has_Job", "Good_Credit"}, "Eligible_For_Loan"),
    ({"No_Debt"}, "Good_Credit"),
    ({"High_Salary"}, "Has_Job"),
]

# Initial known facts
facts = {"No_Debt", "High_Salary"}

# Goal to achieve
goal = "Eligible_For_Loan"

# Create Backward Chaining System
bc_system = BackwardChaining(rules, facts)

# Start backward reasoning
if bc_system.infer(goal):
    print(f"Goal '{goal}' is achieved!")
else:
    print(f"Goal '{goal}' is NOT achieved.")
```

Output:

Checking rule: {'Has_Job', 'Good_Credit'} → Eligible_For_Loan
Checking rule: {'High_Salary'} → Has_Job
Checking rule: {'No_Debt'} → Good_Credit
Goal 'Eligible_For_Loan' is achieved!

8. Implement resolution principle on FOPL related problems

```
from itertools import combinations

# Helper function: Check if two literals are negations of each other
def is_complementary(lit1, lit2):
    return lit1.startswith("~") and lit1[1:] == lit2 or lit2.startswith("~") and lit2[1:] == lit1

# Helper function: Resolve two clauses
def resolve(clause1, clause2):
    new_clauses = []

    for lit1 in clause1:
        for lit2 in clause2:
            if is_complementary(lit1, lit2):
                new_clause = set(clause1) | set(clause2)  # Merge clauses
                new_clause.discard(lit1)  # Remove resolved literals
                new_clause.discard(lit2)
                new_clauses.append(frozenset(new_clause))  # Store new clause
    return new_clauses

# Resolution Algorithm
def resolution(clauses):
    new = set()

    while True:
        pairs = list(combinations(clauses, 2))  # Select pairs of clauses
        for (clause1, clause2) in pairs:
            resolvents = resolve(clause1, clause2)
            if frozenset() in resolvents:  # If empty clause is found
                print("Contradiction found! The given FOPL statements are unsatisfiable.")
                return True
            new.update(resolvents)

        if new.issubset(clauses):  # If no new clauses are generated, stop
            print("No contradiction found. The statements are satisfiable.")
            return False

        clauses.update(new)  # Add new clauses for the next iteration

# Example: Set of clauses (in CNF)
clauses = {
    frozenset({"P", "~Q"}),  # P ∨ ¬Q
    frozenset({"Q"}),        # Q
    frozenset({"~P"})        # ¬P
}

# Run the resolution principle
```

resolution(clauses)

output:

Contradiction found! The given FOPL statements are unsatisfiable.
True

9. Implement Tic-Tac-Toe game using Python

```
import random

def print_board(board):
    """Function to print the Tic-Tac-Toe board."""
    for row in board:
        print(" | ".join(row))
        print("-" * 9)

def check_winner(board, player):
    """Function to check if a player has won."""
    for i in range(3):
        if all(board[i][j] == player for j in range(3)):  # Rows
            return True
        if all(board[j][i] == player for j in range(3)):  # Columns
            return True
    if all(board[i][i] == player for i in range(3)):  # Main diagonal
        return True
    if all(board[i][2 - i] == player for i in range(3)):  # Anti-diagonal
        return True
    return False

def is_draw(board):
    """Function to check if the game is a draw."""
    return all(board[i][j] != " " for i in range(3) for j in range(3))

def computer_move(board):
    """Function to generate a random move for the computer."""
    empty_cells = [(i, j) for i in range(3) for j in range(3) if board[i][j] == " "]
    return random.choice(empty_cells) if empty_cells else None

def play_tic_tac_toe():
    """Main function to play Tic-Tac-Toe."""
    board = [[" " for _ in range(3)] for _ in range(3)]
    human = "X"
    computer = "O"

    print("Welcome to Tic-Tac-Toe!")
    print("You are 'X' and the Computer is 'O'.")
    print_board(board)

    for turn in range(9):  # Max 9 turns
        if turn % 2 == 0:  # Human Player's Turn
            while True:
                try:
                    row, col = map(int, input("Your move (row and column: 1-3 1-3): ").split())
                    row -= 1  # Convert to 0-based index
```

```
                col -= 1
                if board[row][col] != " ":
                    print("Cell is already occupied! Try again.")
                    continue
                board[row][col] = human
                break
            except (ValueError, IndexError):
                print("Invalid input! Enter two numbers between 1-3.")

        else:  # Computer's Turn
            row, col = computer_move(board)
            print(f"Computer moves at ({row+1}, {col+1})")
            board[row][col] = computer

        print_board(board)

        # Check for winner
        if check_winner(board, human):
            print("Congratulations! You win! 🎉")
            return
        elif check_winner(board, computer):
            print("Computer wins! 🤖")
            return

        # Check for draw
        if is_draw(board):
            print("It's a draw! 🤝")
            return

# Run the game
play_tic_tac_toe()
```

output:

```
Welcome to Tic-Tac-Toe!
You are 'X' and the Computer is 'O'.
  |   |
---------
  |   |
---------
  |   |
---------
Your move (row and column: 1-3 1-3): 1 1
X |   |
---------
  |   |
---------
  |   |
---------
Computer moves at (2, 3)
X |   |
---------
  |   | O
---------
  |   |
---------
Your move (row and column: 1-3 1-3): 2 1
X |   |
---------
X |   | O
---------
  |   |
---------
Computer moves at (1, 2)
X | O |
---------
X |   | O
---------
  |   |
---------
Your move (row and column: 1-3 1-3): 3 1
X | O |
---------
X |   | O
---------
X |   |
---------
Congratulations! You win!
```

10. Build a bot which provides all the information related to text in search box

```
import wikipedia
import pyttsx3  # For optional voice output

def search_info(query):
    """Fetches a summarized result from Wikipedia"""
    try:
        summary = wikipedia.summary(query, sentences=3)  # Get first 3 sentences
        return summary
    except wikipedia.exceptions.DisambiguationError as e:
        return f"Multiple results found: {e.options[:5]}... Please be more specific."
    except wikipedia.exceptions.PageError:
        return "Sorry, I couldn't find relevant information. Try another search term."

def chatbot():
    """Chatbot interface for searching information"""
    print("Text Info Bot: Type 'exit' to stop.")

    while True:
        query = input("\n🔎 Enter a search term: ").strip()
        if query.lower() == "exit":
            print(" Thank you!")
            break

        result = search_info(query)
        print(f"\n Result:\n{result}")

        # Optional voice output
        engine = pyttsx3.init()
        engine.say(result)
        engine.runAndWait()

# Run the bot
if __name__ == "__main__":
    chatbot()
```

Install the required libraries:
pip install wikipedia pyttsx3

Text Info Bot:
Type 'exit' to stop.
Enter a search term: Python programming
Result: Python is a high-level, interpreted, general-purpose programming language. Its design philosophy emphasizes code readability with the use of significant indentation. Python supports multiple programming paradigms, including structured, object-oriented, and functional programming.

11. Implement any Game and demonstrate the Game playing strategies

```
import chess
import chess.engine

# Initialize the chess board
board = chess.Board()

# Load the Stockfish chess engine (Replace path if needed)
engine = chess.engine.SimpleEngine.popen_uci("stockfish")

# Play until the game ends
while not board.is_game_over():
   # AI selects the best move
   result = engine.play(board, chess.engine.Limit(time=0.5))  # AI thinks for 0.5 seconds
   board.push(result.move)  # Apply the move

   # Print the current board
   print(board)
   print("\n")

# Print game outcome
print("Game Over!")
print("Result:", board.result())  # Prints 1-0 (White Wins), 0-1 (Black Wins), or ½-½ (Draw)

# Close engine
engine.quit()
```

output

```
r n b q k b n r
p p p p p p p p
. . . . . . . .
. . . . . . . .
. . . . . . . .
. . . . . . . .
P P P P P P P P
R N B Q K B N R

Game Over!
Result: 1-0  (White Wins)
```

Virtual Lab

1. Manual Speech Signal-to-Symbol Transformation

The main objective of this experiment is to make the student understand and appreciate the difficulties in automating the task of speech signal-to-symbol transformation. The students are expected to understand terms such as phones, phonemes, syllables, transliteration, pitch or fundamental frequency, quasi-periodicity. The student should gain a good understanding of the speech signal by visual inspection. Given a speech waveform one should be able to discriminate between silence and speech, voiced speech and unvoiced speech. The student should be able to draw the speech waveform for a given word with appropriate time durations and relative amplitudes of the constituent phones.

Write down a sentence in any Indian language, preferably your native language, on a piece of paper or your note book. Also, write down the sentence in English the way you would probably write it if you were writing a mail to your friend in your language

Eg: Let us consider a sentence in Hindi:

Utt #1: भारत हमारा देश है

Transliteration in English: Bharat hamara desh hai

Part-1: Learn by pre-segmented examples

Step1 : Select an utterance of your choice from the list of examples provided.

Step2 : Note down the utterance in the native script of the language, and the word-level transliteration of the Indian language utterance using English alphabets.

Step3 : Study the subword-level (syllable-level and phoneme-level) transliterations by changing the subword unit.

Step4 : Now choose the subword unit as word, and note the list of words and their boundaries given in the table by the side of the speech waveform.

Part-2: Segmentation with feedback

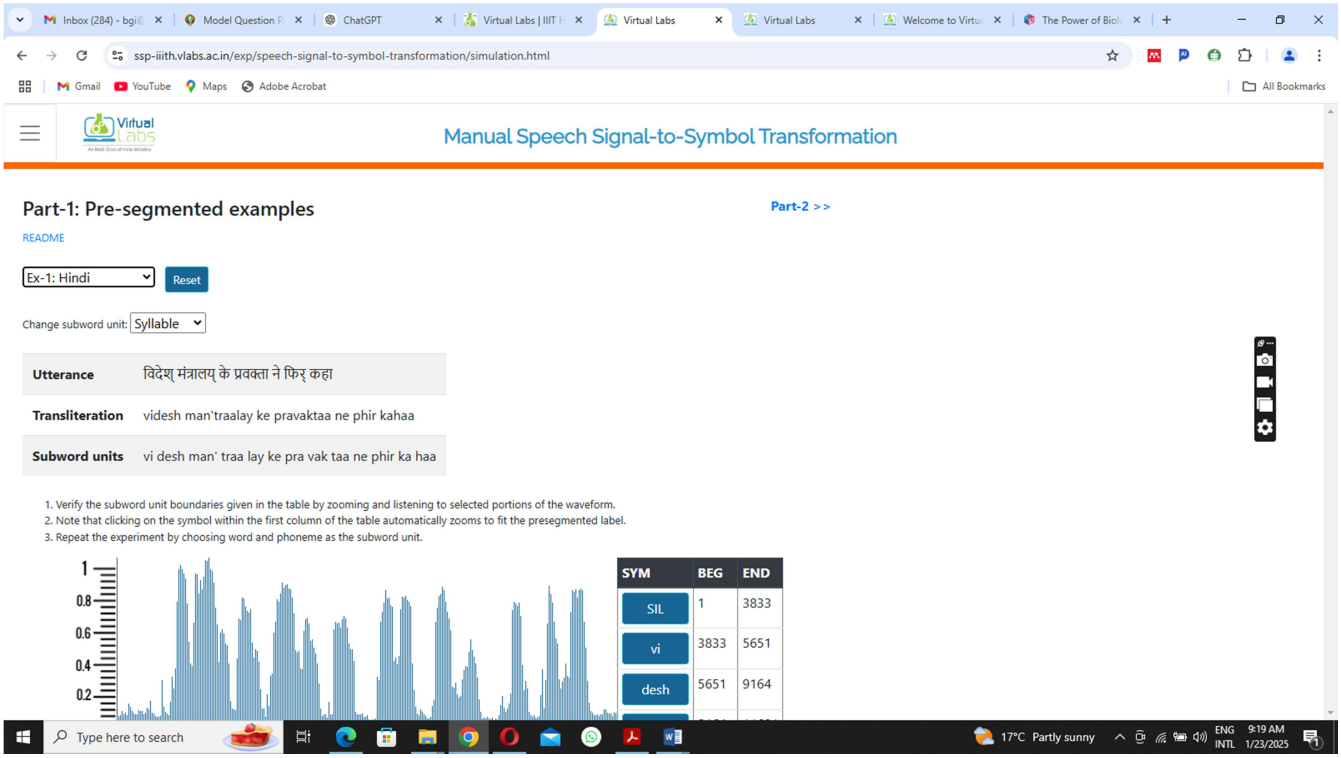
This part of the experiment is very similar to Part-1, except that the word/subword boundaries are not provided. The student is required to segment the given utterance into the chosen subword units by entering the correct subword boundaries.

Step1 : Select an utterance and choose the subword unit as word.

Step2 : Use the 'Zoom Slider', select and 'Play/Pause' buttons to select a portion of the waveform, listen, and enter the subword boundaries in the table.

Step3 : Look out for a message which pops up if the entered boundary value is 20ms more than the reference marking.
Step4 : Repeat the experiment by changing the subword unit to syllable.

Step5 : Repeat the experiment by changing the subword unit to phoneme.



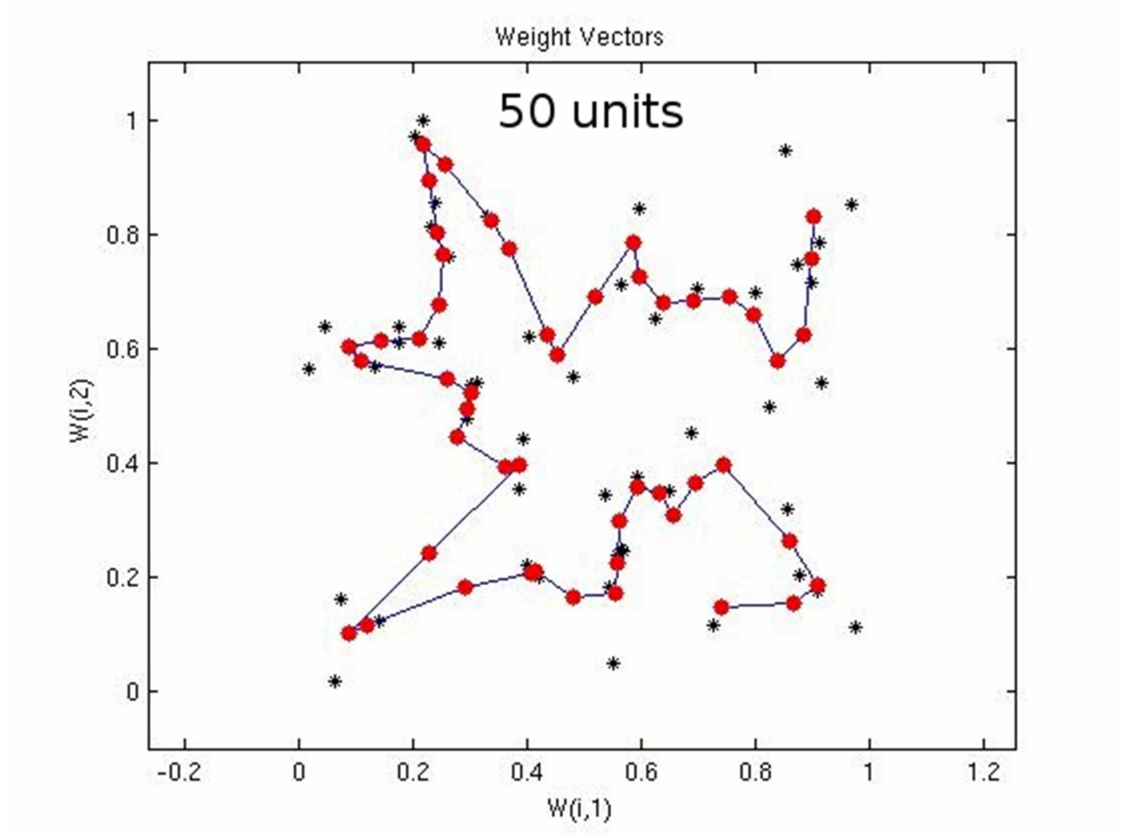
2. Travelling Salesman Problem Using Self Organizing Maps

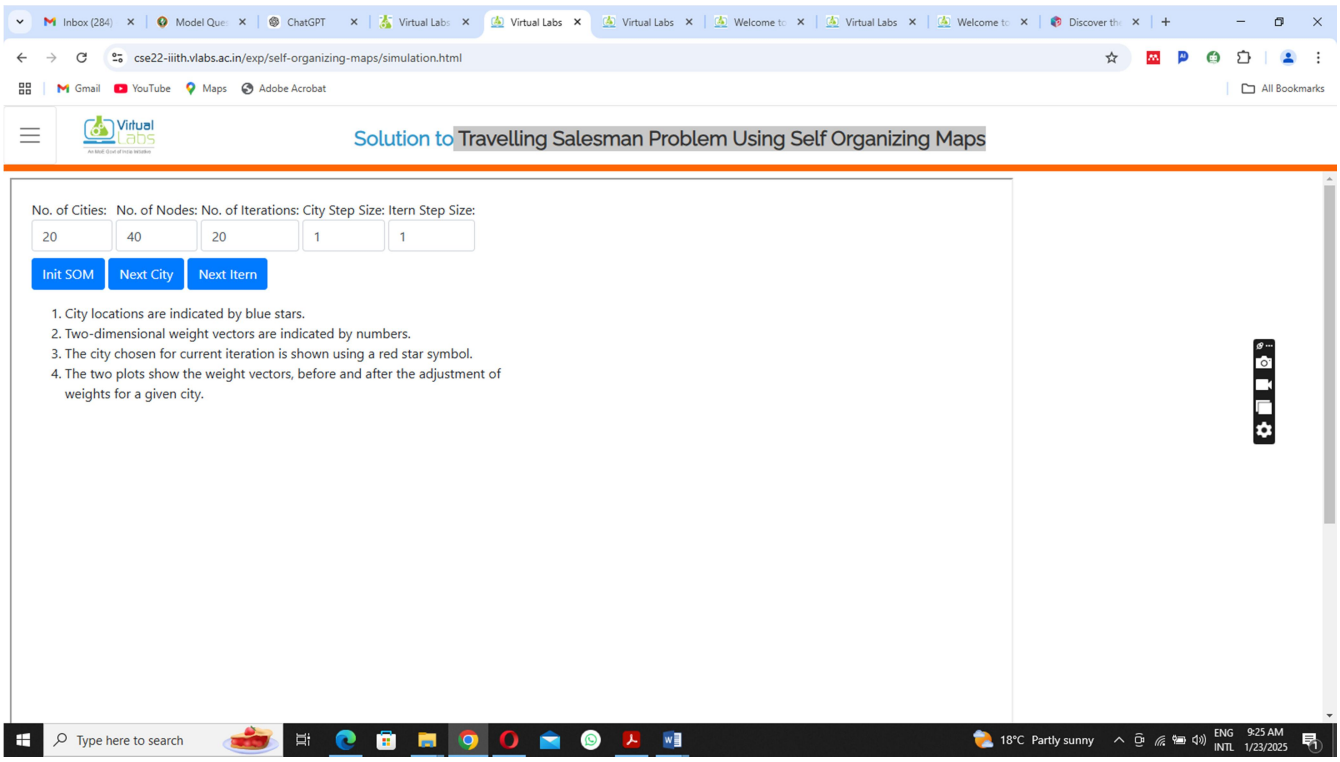
The objective of this experiment is to provide a suboptimal solution to the Travelling Salesman Problem (TSP), using the properties of self-organization feature maps (SOM). The focus is:

- To illustrate the principle of self-organization for addressing the travelling salesman problem
- To observe the suboptimal nature of the solution provided by SOM
- To study the effect of structure of SOM on the solution

Description of self-organizing feature map

Self-organizing map (SOM) was proposed by T. Kohonen [Kohonen, 1982a], and it provides a way of visualizing data. In particular, high dimensional data can be reduced to lower dimensions using SOM. The map also helps in grouping similar data together. Thus the map helps in visualizing the concept of clustering by grouping data with similar characteristics. A SOM attempts to cluster the training data by detecting similarity between data points/ feature vectors. The map does not require external supervision, and hence represents an unsupervised way of learning.




cse22-iiith.vlabs.ac.in/exp/self-organizing-maps/simulation.html
Solution to Travelling Salesman Problem Using Self Organizing Maps
No. of Cities: No. of Nodes: No. of Iterations: City Step Size: Itern Step Size:
20
40
20
1
1
Init SOM
Next City
Next Itern
1. City locations are indicated by blue stars.
2. Two-dimensional weight vectors are indicated by numbers.
3. The city chosen for current iteration is shown using a red star symbol.
4. The two plots show the weight vectors, before and after the adjustment of weights for a given city.
Type here to search
18°C Partly sunny
ENG INTL
9:25 AM 1/23/2025

KLS

Vishwanathrao Deshpande Institute of TechnologyHaliyal–581329

MongoDB LABORATORY

[BDSL456B]

for

IV Semester B.E.

Asprescribedby

VISVESVARAYA TECHNOLOGICAL UNIVERSITY,BELAGAVI–590014

(For the Academic Year 2025-2026)

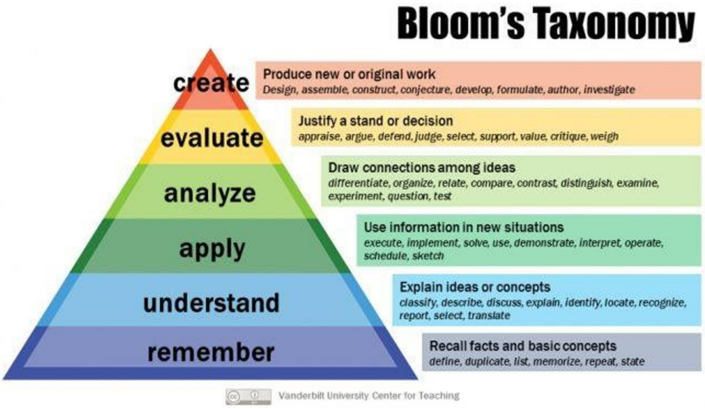
Prepared by
Prof. Ravindra Patil

Index

PROGRAM OUTCOMES(POs)

Program Outcomes as defined by NBA (PO) Engineering Graduates will be able to:

1. Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.



12. Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning

Vision (College)
To nurture talent and enrich society through excellence in technical education, research and innovation.
Mission (College)
1. To augment innovative Pedagogy, kindle quest for interdisciplinary learning & to enhance conceptual understanding. 2. To build competence, professional ethics & develop entrepreneurial thinking. 3. To strengthen Industry Institute Partnership & explore global collaborations. 4. To inculcate culture of socially responsible citizenship. 5. To focus on Holistic & Sustainable development.
Vision (Dept)
To lead the way in the creation and application of cutting-edge Computer science and engineering (AIML) technologies, advancing the frontiers of knowledge, and empowering future generations to drive innovation and transformation on a global scale.
Mission (Dept)
1. To train students with a strong conceptual understanding using innovative pedagogies, empowering them to excel in the dynamic fields of Artificial Intelligence and Machine Learning. 2. To imbibe professional, research, and entrepreneurial skills with a commitment to the nation's development at large. 3. To strengthen the industry-institute Interaction. 4. To promote life–long learning with a sense of societal & ethical responsibilities.

Program Educational Objectives (PEO)	
PEO1	To develop an ability to identify and analyze the requirements of Computer Science and Engineering in design and providing novel engineering solutions
PEO2	To develop abilities to work in team on multidisciplinary projects with effective communication skills, ethical qualities and leadership roles.
PEO3	To develop abilities for successful Computer Science Engineer and achieve higher career goals..
Program Specific Outcomes (PSO)	
PSO 1	Develop abilities to model real world problems using appropriate algorithms, computational theories and programming languages in the area of AI and ML

CO's And PO's Mapping Chart

Subject with code: MONGO DB LABORATORY (BDS456B) AY:2023-24

Semester: IV

S.No.	Description	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2
1	Make use of MangoDB commands and queries.	3												1	
2	Illustrate the role of aggregate pipelines to extract data.	3	2											1	
3	Demonstrate optimization of queries by creating indexes.	3	2											1	
4	Develop aggregate pipelines for text search in collections	3	2											1	

Degree of compliance Low:1 Medium:2 High:3

Evaluation:

	Particulars	Marks	Total
CIA	Performance	15	**30**
	Journal	10	
	Viva-voce	05	
	Lab IA	20	**20**
	Grand Total		**50**

Mapping of Experiments with CO, PO and PSO

Sl. No.	Experiment Details	CO	PO	PSO
1	a) Illustration of Where Clause, AND,OR operations in MongoDB. b) Execute the Commands of MongoDB and operations in MongoDB : Insert, Query, Update, Delete and Projection. (Note: use any collection)	1	1	1
2	a) Develop a MongoDB query to select certain fields and ignore some fields of the documents from any collection. b) Develop a MongoDB query to display the first 5 documents from the results obtained in a. [use of limit and find]	1	1	1
3	a) Execute query selectors (comparison selectors, logical selectors) and list out the results on any collection b) Execute query selectors (Geospatial selectors, Bitwise selectors) and list out the results on any collection	1	1	1
4	Create and demonstrate how projection operators ($, $elematch and $slice) would be used in the MondoDB.	1	1	1
5	Execute Aggregation operations ($avg, $min,$max, $push, $addToSet etc.). students encourage to execute several queries to demonstrate various aggregation operators)	2	2	1
6	Execute Aggregation Pipeline and its operations (pipeline must contain $match, $group, $sort, $project, $skip etc. students encourage to execute several queries to demonstrate various aggregation operators)	2	2	1
7	a) Find all listings with listing_url, name, address, host_picture_url in the listings And Reviews collection that have a host with a picture url b) Using E-commerce collection write a query to display reviews summary.	3	2	1
8	a) Demonstrate creation of different types of indexes on collection (unique, sparse, compound and multikey indexes) b) Demonstrate optimization of queries using indexes.	3	2	1
9	a) Develop a query to demonstrate Text search using catalog data collection for a given word b) Develop queries to illustrate excluding documents with certain words and phrases	4	2	1
10	Develop an aggregation pipeline to illustrate Text search on Catalog data collection.	4	2	1

Introduction, Architecture, Features & Example

What is MongoDB?

MongoDB is a document-oriented NoSQL database used for high volume data storage. MongoDB is a database which came into light around the mid-2000s. It falls under the category of a NoSQL database.

MongoDB Features

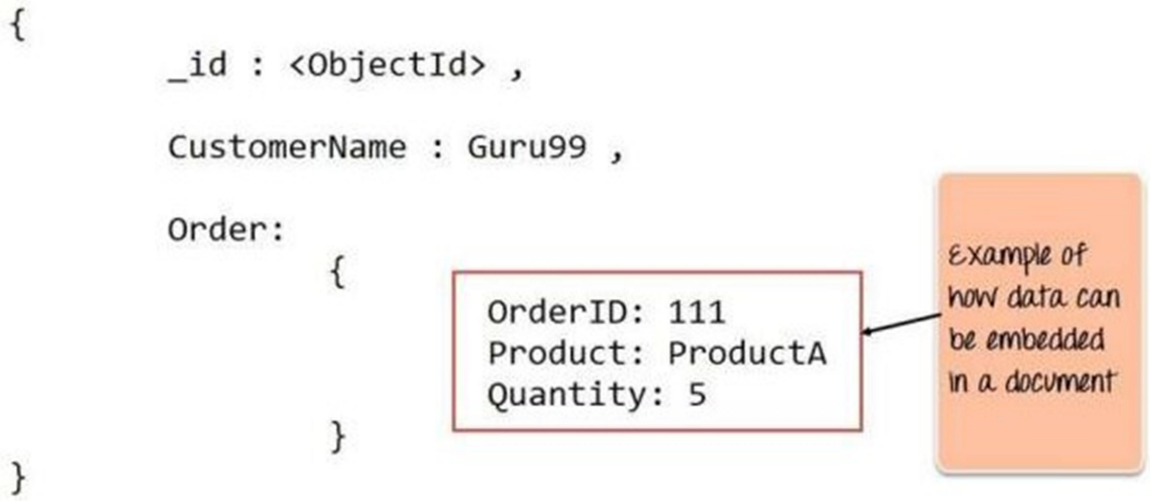
1. Each database contains collections which in turn contains documents. Each document can be different with a varying number of fields. The size and content of each document can be different from each other.
2. The document structure is more in line with how developers construct their classes and objects in their respective programming languages. Developers will often say that their classes are not rows and columns but have a clearstructure with key-value pairs.
3. As seen in the introduction with NoSQL databases, the rows (or documents as called in MongoDB) doesn't need to have a schema defined beforehand. Instead, the fields can be created on the fly.
4. The data model available within MongoDB allows you to represent hierarchical relationships, to store arrays, and other more complex structures more easily.
5. Scalability – The MongoDB environments are very scalable. Companies across the world have defined clusters with some of them running 100+ nodes with around millions of documents.

MongoDB Example

The below example shows how a document can be modeled in MongoDB.

1. The _id field is added by MongoDB to uniquely identify the document in the collection.

2. What you can note is that the Order Data (OrderID, Product, and Quantity) which in RDBMS will normally be stored in a separate table, while in MongoDB it is actually stored as an embedded document in the collection itself. This is one of the key differences in how data is modeled in MongoDB.



Key Components of MongoDB Architecture

Below are a few of the common terms used in MongoDB

1. _id – This is a field required in every MongoDB document. The _id field represents a unique value in the MongoDB document. The _id field is like the document's primary key. If you create a new document without an _id field, MongoDB will automatically create the field. So for example, if we see the example of the above customer table, Mongo DB will add a 24 digit unique identifier to each document in the collection.

_Id	CustomerID	CustomerName	OrderID
563479cc8a8a4246bd27d784	11	Guru99	111
563479cc7a8a4246bd47d784	22	Trevor Smith	222
563479cc9a8a4246bd57d784	33	Nicole	333

2. Collection – This is a grouping of MongoDB documents. A collection is the equivalent of a table which is created in any other RDMS such as Oracle or MS SQL. A collection exists within a single database. As seen from the introduction collections don't enforce any sort of structure.

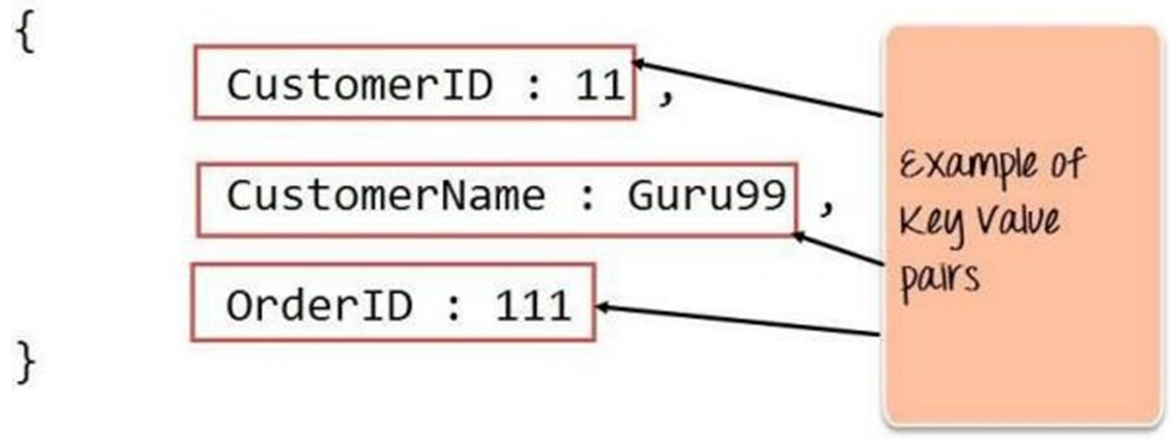
3. Cursor – This is a pointer to the result set of a query. Clients can iterate through a cursor to retrieve results.

4. Database – This is a container for collections like in RDMS wherein it is a container for tables. Each database gets its own set of files on the file system. A MongoDB server can store multiple databases.

5. Document - A record in a MongoDB collection is basically called a document. The document, in turn, will consist of field name and values.

6. Field - A name-value pair in a document. A document has zero or more fields. Fields are analogous to columns in relational databases

The following diagram shows an example of Fields with Key value pairs. So in the example below CustomerID and 11 is one of the key value pair's defined in the document.



7. JSON – This is known as JavaScript Object Notation. This is a humanreadable, plain text format for expressing structured data. JSON is currently supported in many programming languages.

Why Use MongoDB?

Below are the few of the reasons as to why one should start using MongoDB

1. Document-oriented – Since MongoDB is a NoSQL type database, instead of having data in a relational type format, it stores the data in documents. This makes MongoDB very flexible and adaptable to real business world situation and requirements.

2. Ad hoc queries - MongoDB supports searching by field, range queries,and regular expression searches. Queries can be made to return specific fields within documents.

3. Indexing - Indexes can be created to improve the performance of searches within MongoDB. Any field in a MongoDB document can be indexed.

4. Replication - MongoDB can provide high availability with replica sets. A replica set consists of two or more mongo DB instances. Each replica set member may act in the role of the primary or secondary replica at any time. The primary replica is the main server which interacts with the client and performs all the read/write operations. The Secondary replicas maintain a copy of the data of the primary using built-in replication. When a primary replica fails, the replica set automatically switches over to the secondary and then it becomes the primary server.

Install MongoDB Community Edition

Procedure
Follow these steps to install MongoDB Community Edition using the MongoDB Installer wizard. The installation process installs both the MongoDB binaries as well as the default configuration file <install directory>\bin\mongod.cfg.

- **Download the installer.**

Download the MongoDB Community .msi installer from the following link:
➢ MongoDB Download Center

a. In the **Version** dropdown, select the version of MongoDB to download.
b. In the **Platform** dropdown, select **Windows**.
c. In the **Package** dropdown, select **msi**.
d. Click **Download**.

- **Run the MongoDB installer.**

For example, from the Windows Explorer/File Explorer:

a. Go to the directory where you downloaded the MongoDB installer (.msi file). By default, this is your Downloads directory.
b. Double-click the .msi file.

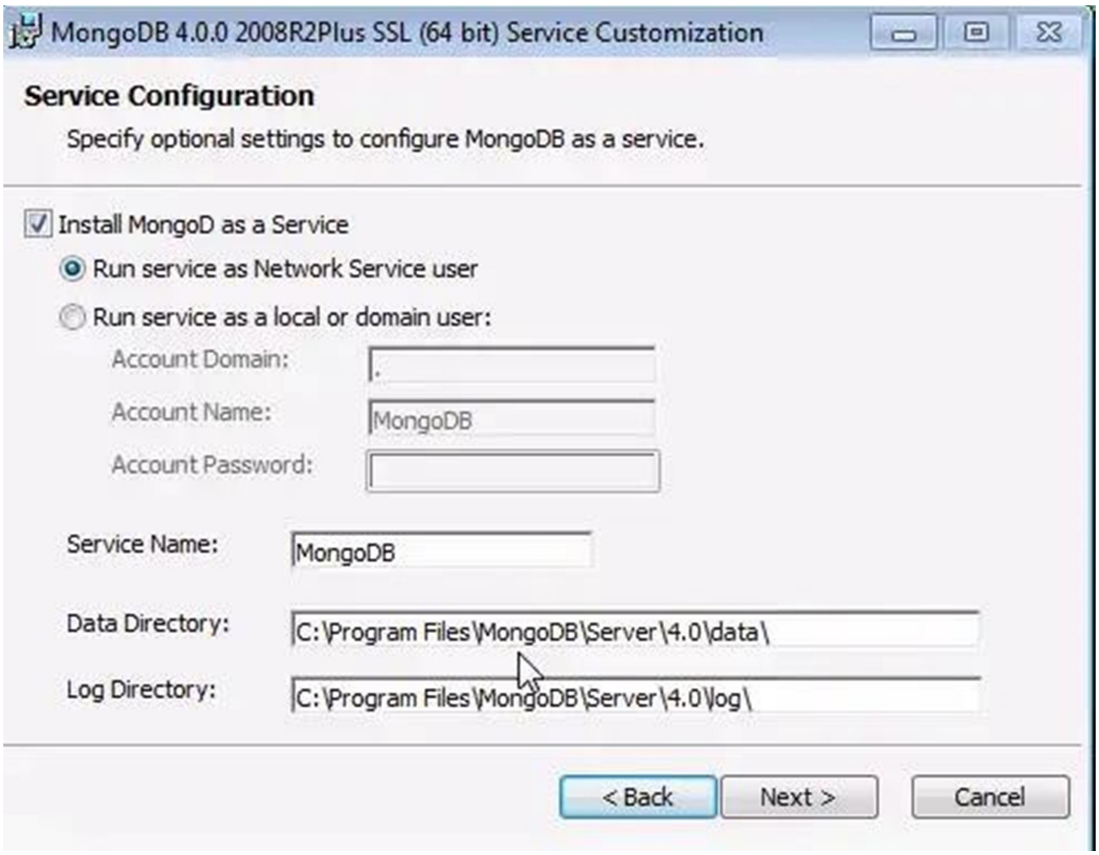
- **Follow the MongoDB Community Edition installation wizard.**

The wizard steps you through the installation of MongoDB and MongoDB Compass.

a. Choose Setup Type
You can choose either the **Complete** (recommended for most users) or **Custom** setup type. The **Complete** setup option installs MongoDB and the MongoDB tools to the default location. The **Custom** setup option allows you to specify which executables are installed and where.

b. Service Configuration
Starting in MongoDB 4.0, you can set up MongoDB as a Windows service during the install or just install the binaries.MongoDB Service MongoDB

The following installs and configures MongoDB as a Windows service.
Starting in MongoDB 4.0, you can configure and start MongoDB as a Windows service during the install, and the MongoDB service is started upon successful installation.



- Select **Install MongoD as a Service** MongoDB as a service.
- Select either:
  - **Run the service as Network Service user** (Default)
    This is a Windows user account that is built-in to Windows
    or
  - **Run the service as a local or domain user**
    - For an existing local user account, specify a period (i.e. .) for the **Account Domain** and specify the **Account Name** and the **Account Password** for the user.
    - For an existing domain user, specify the **Account Domain**, the **Account Name** and the **Account Password** for that user.
- **Service Name**. Specify the service name. Default name is MongoDB. If you already have a service with the specified name, you must choose another name.

- **Data Directory**. Specify the data directory, which corresponds to the --dbpath. If the directory does not exist, the installer will create the directory and sets the directory access to the service user.
- **Log Directory**. Specify the Log directory, which corresponds to the --logpath. If the directory does not exist, the installer will create the directory and sets the directory access to the service user.

c. **Install MongoDB Compass**
Optional. To have the wizard install MongoDB Compass, select **Install MongoDB Compass** (Default).

d. When ready, click **Install**.

Install mongosh

The .msi installer does not include mongosh. Follow the mongosh installation instructions to download and install the shell separately.

If You Installed MongoDB as a Windows Service

The MongoDB service starts upon successful installation.
If you would like to customize the service, you must stop the service. Customize the MongoDB instance by editing the configuration file at <install directory>\bin\mongod.cfg.
For information about the available configuration options, refer to Configuration File Options.
After making changes, start the service again.

If You Did Not Install MongoDB as a Windows Service

If you only installed the executables and did not install MongoDB as a Windows service, you must manually start the MongoDB instance.
See Run MongoDB Community Edition from the Command Interpreter for instructions to start a MongoDB instance.

Run MongoDB Community Edition as a Windows Service

Starting in version 4.0, you can install and configure MongoDB as a **Windows Service** during installation. The MongoDB service starts upon successful installation. Configure the MongoDB instance with the configuration file <install directory>\bin\mongod.cfg.
If you have not already done so, follow the mongosh installation instructions to download and install the MongoDB Shell (mongosh).
Be sure to add the path to your mongosh.exe binary to your PATH environment variable during installation.
Open a new **Command Interpreter** and enter mongosh.exe to connect to MongoDB.
For more information on connecting to a mongod using mongosh.exe, such as connecting to a MongoDB instance running on a different host and/or port, see Connect to a Deployment.
For information on CRUD (Create, Read, Update, Delete) operations, see:

- Insert Documents
- Query Documents
- Update Documents

- Delete Documents

Start MongoDB Community Edition as a Windows Service
To start/restart the MongoDB service, use the Services console:

1. From the Services console, locate the MongoDB service.
2. Right-click on the MongoDB service and click **Start**.

Stop MongoDB Community Edition as a Windows Service
To stop/pause the MongoDB service, use the Services console:

1. From the Services console, locate the MongoDB service.
2. Right-click on the MongoDB service and click **Stop** (or **Pause**).

Remove MongoDB Community Edition as a Windows Service
To remove the MongoDB service, first use the Services console to stop the service. Then open a Windows command prompt/interpreter (cmd.exe) as an **Administrator**, and run the following command:
sc.exe delete MongoDB

Run MongoDB Community Edition from the Command Interpreter
You can run MongoDB Community Edition from the Windows command prompt/interpreter (cmd.exe) instead of as a service.

Experiment :1

1a) Illustration of Where Clause, AND,OR operations in MongoDB.

1b) Execute the Commands of MongoDB and operations in MongoDB : Insert, Query, Update, Delete and Projection

In MongoDB, the find() method is used to query data from a collection. You can use the find() method along with the $and and $or operators to perform logical operations.

Let's consider a sample collection **students**:

```
{
 "name": "Alice",
 "gender": "female",
 "grade": 10,
 "gpa": 3.8
}
{
 "name": "Bob",
 "gender": "male",
 "grade": 11,
 "gpa": 4.2
}
{
 "name": "Charlie",
 "gender": "male",
 "grade": 11,
 "gpa": 3.9
}
{
 "name": "Diana",
 "gender": "female",
 "grade": 12,
 "gpa": 3.6
}
```

1. Using AND Operator:

Suppose we have a collection named students with documents containing information about students. We want to find students who are both male and have a GPA greater than 3.5. We can use the find() method with the $and operator:

```
db.students.find({   $and: [    { gender: "male" },    { gpa: { $gt: 3.5 } }  ] })
```

Output:

json

```
{ "name": "Bob", "gender": "male", "grade": 11, "gpa": 4.2 }
{ "name": "Charlie", "gender": "male", "grade": 11, "gpa": 3.9 }
```

2. Using OR Operator:

Suppose we want to find students who are either in grade 11 or have a GPA greater than 4.0. We can use the find() method with the $or operator:

```
db.students.find({   $or: [    { grade: 11 },    { gpa: { $gt: 4.0 } }   ] })
```

Output:

json

```
{ "name": "Bob", "gender": "male", "grade": 11, "gpa": 4.2 }
{ "name": "Charlie", "gender": "male", "grade": 11, "gpa": 3.9 }
{ "name": "Diana", "gender": "female", "grade": 12, "gpa": 3.6 }
```

Insert Documents:

To insert documents into a collection, you can use the insertOne() or insertMany() methods.

javascript
db.students.insertOne({ name: "Alice", gender: "female", grade: 10, gpa: 3.8 })

Output: This will insert the document into the students collection

Update Documents:

To update documents in a collection, you can use the updateOne() method.

javascript
db.students.updateOne({ name: "Alice" }, { $set: { gpa: 4.0 } })

Output: This will update the gpa field of the document where the name is "Alice"

Delete Documents:

To delete documents from a collection, you can use the deleteOne() method.

javascript

db.students.deleteOne({ name: "Alice" })

Output: This will delete the document where the name is "Alice".

Projection:
To project specific fields in the query results, you can use the second argument of the find() method with a projection object.

db.students.find({ grade: 10 }, { name: 1, gpa: 1 })

Output: This will retrieve the name and gpa fields of documents where the grade is 10.

{ "_id" : ObjectId("1234567890"), "name" : "Alice", "gpa" : 3.8 }

Experiment :2

a. **Develop a MongoDB query to select certain fields and ignore some fields of the documents from any collection.**
b. **Develop a MongoDB query to display the first 5 documents from the results obtained in a. [use of limit and find]**

Let's consider a sample collection:

json
{ "_id" : ObjectId("1234567890"), "name" : "Alice", "gender" : "female", "grade" : 10, "gpa" : 3.8 }
{ "_id" : ObjectId("0987654321"), "name" : "Bob", "gender" : "male", "grade" : 11, "gpa" : 4.2 }
{ "_id" : ObjectId("1357924680"), "name" : "Charlie", "gender" : "male", "grade

Let's assume we have a collection named students with documents containing various fields such as name, age, gender, grade, and gpa. Suppose we want to retrieve only the name and grade fields while ignoring the _id field. Here's how you can achieve this with MongoDB:

javascript

```
db.students.find({}, { name: 1, grade: 1, _id: 0 })
```

Output:

json
{ "name" : "Alice", "grade" : 10 }
{ "name" : "Bob", "grade" : 11 }
{ "name" : "Charlie", "grade" : 12 }

2b)Let's consider the same sample collection:

json

```
{ "name" : "Alice", "grade" : 10 }
{ "name" : "Bob", "grade" : 11 }
{ "name" : "Charlie", "grade" : 12 }
{ "name" : "David", "grade" : 9 }
{ "name" : "Eve", "grade" : 10 }
{ "name" : "Frank", "grade" : 11 }
{ "name" : "Grace", "grade" : 12 }
{ "name" : "Henry", "grade" : 9 }
{ "name" : "Ivy", "grade" : 10 }
{ "name" : "Jack", "grade" : 11 }
```

To display the first 5 documents from the results obtained in the previous query (query a.), We can simply add the limit() method to your query.

javascript

```
db.students.find({}, { name: 1, grade: 1, _id: 0 }).limit(5)
```

Output:

After executing the MongoDB query, the output would be:

json

```
{ "name" : "Alice", "grade" : 10 }
{ "name" : "Bob", "grade" : 11 }
{ "name" : "Charlie", "grade" : 12 }
{ "name" : "David", "grade" : 9 }
{ "name" : "Eve", "grade" : 10 }
```

Experiment :3

a. **Execute query selectors (comparison selectors, logical selectors) and list out the results on any collection**

b. **Execute query selectors (Geospatial selectors, Bitwise selectors) and list out the results on any collection**

Example 1: Using Comparison Selector $eq (equals)

```
db.students.find({ grade: { $eq: 10 } })
```

Output: It will retrieve all documents from the students collection where the grade field is equal to 10

```
[
  {
        _id: ObjectId('660c3aa92948120c76c00e7e'),
        name: 'Alice',
        gender: 'female',
        grade: 10,
        gpa: 3.8
  }
]
```

Example 2: Using Comparison Selector $gt (greater than)

Suppose we want to find students with a GPA greater than 3.5.

```
db.students.find({ gpa: { $gt: 3.5 } })
```

Output: It will retrieve all documents from the students collection where the gpa field is greater than 3.5.

```
[
  {
        _id: ObjectId('660c3aa92948120c76c00e7e'),
        name: 'Alice',
        gender: 'female',
        grade: 10,
        gpa: 3.8
  },
```

```
  {
        _id: ObjectId('660c3aee2948120c76c00e7f'),
        name: 'Bob',
        gender: 'male',
        grade: 11,
        gpa: 4.2
  },
  {
        _id: ObjectId('660c3b082948120c76c00e80'),
        name: 'Charlie',
        gender: 'male',
        grade: 11,
        gpa: 3.9
  },
]
```

Logical Selectors: $and $or

Example 3: Using Logical Selector $and

Suppose we want to find students who are in grade 10 and have a GPA greater than 3.5.

```
db.students.find({  $and: [     { grade: { $eq: 10 } },     { gpa: { $gt: 3.5 } }  ] })
```

Output: This will retrieve all documents from the students collection where both conditions (grade equals 10 and GPA greater than 3.5) are met.

```
  {
        _id: ObjectId('660c3aa92948120c76c00e7e'),
        name: 'Alice',
        gender: 'female',
        grade: 10,
        gpa: 3.8
  }
```

Example 4: Using Logical Selector $or

Suppose we want to find students who are in grade 10 or have a GPA greater than 4.0.

javascript

```
db.students.find({  $or: [     { grade: { $eq: 10 } },     { gpa: { $gt: 4.0 } }    ] })
```

Output:

```
[
  {
        _id: ObjectId('660c3aa92948120c76c00e7e'),
        name: 'Alice',
        gender: 'female',
        grade: 10,
        gpa: 3.8
  },
  {
        _id: ObjectId('660c3aee2948120c76c00e7f'),
        name: 'Bob',
        gender: 'male',
        grade: 11,
        gpa: 4.2
  }
]
```

3b Execute query selectors (Geospatial selectors, Bitwise selectors) and list out the results on any collection

let's consider an example where we have a collection named **locations**, and each document contains information about a location including its name and coordinates (latitude and longitude)

```
{
 "_id" : ObjectId("1234567890"),
 "name" : "Park",
 "location" : { "type": "Point", "coordinates": [ -73.935242, 40.730610 ] }
}
{
 "_id" : ObjectId("0987654321"),
 "name" : "Coffee Shop",
 "location" : { "type": "Point", "coordinates": [ -73.987501, 40.748817 ] }
}
{
 "_id" : ObjectId("2468013579"),
 "name" : "Library",
 "location" : { "type": "Point", "coordinates": [ -73.981326, 40.756210 ] }
}
```

$near

Find locations near a specified point. For example, let's find locations near the coordinates [-73.986610, 40.730985] within a maximum distance of 1000 meters.

javascript

```
db.locations.find({ location: { $near: {$geometry: {type: "Point",coordinates: [ -73.986610, 40.730985 ]
    }, $maxDistance: 1000 }
```

Output: This query will retrieve all documents from the locations collection where the location field is near the specified coordinates within a maximum distance of 1000 meters

2. **$geoWithin**

Find locations within a specified geometry. For example, let's find locations within a polygon representing a specific area.

2. $geoWithin

Find locations within a specified geometry. For example, let's find locations within a polygon representing a specific area.

```
javascript
db.locations.find({
location: {
   $geoWithin: {
     $geometry: {
       type: "Polygon",
       coordinates: [
         [
           [-74.000415, 40.730446],
           [-73.986168, 40.729287],
           [-73.984106, 40.736000],
           [-74.002701, 40.735297],
           [-74.000415, 40.730446]
         ]
       ]
     }
   }
 }
})
```

Output: This query will retrieve all documents from the locations collection where the location field is within the specified polygon geometry

Experiment :4
Create and demonstrate how projection operators ($, $elematch and $slice) would be used in the MondoDB.

Consider a collection named students with documents containing an array of exam scores. We want to project only the first exam score for each student.

json
```
{
  "_id" : ObjectId("1234567890"),
  "name" : "Alice",
  "exam_scores" : [ 85, 90, 78, 92 ]
}
{
  "_id" : ObjectId("0987654321"),
  "name" : "Bob",
  "exam_scores" : [ 72, 88, 90, 81 ]
}
```

1. Using $ Operator:
The $ operator projects the first element in an array that matches the query condition.

```
db.students.find({}, { "exam_scores.$": 1 })
```

Output: This query will retrieve all documents from the students collection and project only the first exam score for each student.

json

```
{ "_id" : ObjectId("1234567890"), "exam_scores" : [ 85 ] }
{ "_id" : ObjectId("0987654321"), "exam_scores" : [ 72 ] }
```

2. Using $elemMatch Operator:

The $elemMatch operator projects only the first element in an array that matches the specified condition.

Example:

Consider the same collection students, but this time we want to project only the exam scores greater than 80 for each student.

```
db.students.find({}, { "exam_scores": { $elemMatch: { $gt: 80 } } })
```

Output: This query will retrieve all documents from the students collection and project only the first exam score greater than 80 for each student.

json

```
{ "_id" : ObjectId("1234567890"), "exam_scores" : [ 85 ] }
{ "_id" : ObjectId("0987654321"), "exam_scores" : [ 88 ] }
```

3. Using $slice Operator:

The $slice operator limits the number of elements projected from an array.

Example:

Consider the same collection students, but this time we want to project only the first two exam scores for each student.

```
db.students.find({}, { "exam_scores": { $slice: 2 } })
```

Output: This query will retrieve all documents from the students collection and project only the first two exam scores for each student.

json

```
{ "_id" : ObjectId("1234567890"), "exam_scores" : [ 85, 90 ] }
{ "_id" : ObjectId("0987654321"), "exam_scores" : [ 72, 88 ] }
```

Experiment :5

Execute Aggregation operations ($avg, $min,$max, $push, $addToSet etc.). students encourage to execute several queries to demonstrate various aggregation operators)

Example 1: Using $avg Operator

Suppose we want to calculate the average GPA of all students.

javascript

```
db.students.aggregate([ { $group: { _id: null, avgGPA: { $avg: "$gpa" } } }])
```

Output: This query will calculate the average GPA of all students.

json

```
{ "_id" : null, "avgGPA" : 3.8 }
```

Example 2: Using $min Operator

Suppose we want to find the minimum GPA among all students.

javascript

```
db.students.aggregate([ { $group: { _id: null, minGPA: { $min: "$gpa" } } }])
```

Output: This query will find the minimum GPA among all students.

```
{ "_id" : null, "minGPA" : 3.2 }
```

Example 3: Using $max Operator

Suppose we want to find the maximum GPA among all students.

javascript

```
db.students.aggregate([ { $group: { _id: null, maxGPA: { $max: "$gpa" } } }])
```

Output: This query will find the maximum GPA among all students.

json

```
{ "_id" : null, "maxGPA" : 4.0 }
```

Example 4: Using $push Operator

Suppose we want to group students by grade and push their names into an array.

```
javascript
db.students.aggregate([
  {
    $group: {
      _id: "$grade",
      students: { $push: "$name" }
    }
  }
])
```

Output: This query will group students by their grade and push their names into an array.

```
json
{ "_id" : 10, "students" : [ "Alice", "Eve", "Ivy" ] }
{ "_id" : 11, "students" : [ "Bob", "Frank", "Jack" ] }
{ "_id" : 12, "students" : [ "Charlie", "Grace" ] }
```

Example 5: Using $addToSet Operator

Suppose we want to group students by gender and add their unique names into a set.

```
javascript
db.students.aggregate([
  {
    $group: {
      _id: "$gender",
      uniqueNames: { $addToSet: "$name" }
    }
  }
])
```

Output: This query will group students by their gender and add their unique names into a set.

```
json
{ "_id" : "female", "uniqueNames" : [ "Alice" ] }
{ "_id" : "male", "uniqueNames" : [ "Bob", "Charlie", "Frank", "Jack", "Eve", "Grace", "Ivy" ] }
```

Experiment :6

Execute Aggregation Pipeline and its operations (pipeline must contain $match, $group, $sort, $project, $skip etc. students encourage to execute several queries to demonstrate various aggregation operators)

Example 1: Aggregation Pipeline with $match, $group, and $project

Suppose we want to calculate the average GPA of female students who are in grade 10 or higher.

```
javascript
db.students.aggregate([
  {
   $match: {
     gender: "female",
     grade: { $gte: 10 }
   }
  },
  {
   $group: {
     _id: null,
     avgGPA: { $avg: "$gpa" }
   }
  },
  {
   $project: {
     _id: 0,
     avgGPA: 1
   }
  }
])
```

Output: This pipeline will calculate the average GPA of female students who are in grade 10 or higher.

```
json
{ "avgGPA" : 3.85 }
```

Example 2: Aggregation Pipeline with $sort and $limit
Suppose we want to find the top 3 students with the highest GPAs.

javascript
```
db.students.aggregate([
  {
   $sort: { gpa: -1 }
  },
  {
   $limit: 3
  }
])
```

Output: This pipeline will sort the students by GPA in descending order and return the top 3 students.

json
```
{ "_id" : ObjectId("1234567890"), "name" : "Alice", "gender" : "female", "grade" : 10, "gpa" : 4.0 }
{ "_id" : ObjectId("0987654321"), "name" : "Bob", "gender" : "male", "grade" : 11, "gpa" : 3.9 }
{ "_id" : ObjectId("2468013579"), "name" : "Charlie", "gender" : "male", "grade" : 12, "gpa" : 3.9 }
```

Example 3: Aggregation Pipeline with $skip
Suppose we want to skip the first two students and retrieve the rest.

javascript

```
db.students.aggregate([
  {
   $skip: 2
  }
])
```

Output: This pipeline will skip the first two students and return the rest.

json
```
{ "_id" : ObjectId("2468013579"), "name" : "Charlie", "gender" : "male", "grade" : 12, "gpa" : 3.9 }
{ "_id" : ObjectId("1357924680"), "name" : "David", "gender" : "male", "grade" : 9, "gpa" : 3.2 }
{ "_id" : ObjectId("9876543210"), "name" : "Eve", "gender" : "female", "grade" : 10, "gpa" : 3.8 }
```

Experiment :7

a. **Find all listings with listing_url, name, address, host_picture_url in the listings And Reviews collection that have a host with a picture url**

b. **Using E-commerce collection write a query to display reviews summary**

Structure of 'listingsAndReviews' collection:


```
{
  _id: '10059872',
  listing_url: 'https://www.airbnb.com/rooms/10059872',
  name: 'Soho Cozy, Spacious and Convenient',
  summary: 'Clean, fully furnish, Spacious 1 bedroom flat just off the escalator in Mid Levels. 2 minutes From Soho Bar and Restaurants. Located in a quiet alley 1 minute from Sun Yat Sen',
  space: '',
  description: 'Clean, fully furnish, Spacious 1 bedroom flat just off the escalator in Mid Levels. 2 minutes From Soho Bar and Restaurants. Located in a quiet alley 1 minute from Sun Yat Sen',
  neighborhood_overview: '',
  notes: '',
  transit: '',
  access: '',
  interaction: '',
  house_rules: '',
  property_type: 'Apartment',
  room_type: 'Entire home/apt',
  bed_type: 'Real Bed',
  minimum_nights: '4',
  maximum_nights: '20',
  cancellation_policy: 'flexible',
  last_scraped: ISODate("2019-03-11T04:00:00.000Z"),
  calendar_last_scraped: ISODate("2019-03-11T04:00:00.000Z"),
  first_review: ISODate("2015-12-19T05:00:00.000Z"),
  last_review: ISODate("2018-03-27T04:00:00.000Z"),
  accommodates: 3,
  bedrooms: 1,
  beds: 2,
  number_of_reviews: 3,
  bathrooms: Decimal128("1.0"),
  amenities: [
    'Air conditioning',
    'Kitchen',
    'Smoking allowed',
    'Doorman',
```

```
    'Elevator',
    'Heating',
    'Family/kid friendly',
    'Essentials',
    '24-hour check-in',
    'translation missing: en.hosting_amenity_50'
  ],
  price: Decimal128("699.00"),
  weekly_price: Decimal128("5000.00"),
  extra_people: Decimal128("0.00"),
  guests_included: Decimal128("1"),
  images: {
    thumbnail_url: '',
    medium_url: '',
    picture_url: 'https://a0.muscache.com/im/pictures/4533a1dc-6fd8-4167-938d-
391c6eebbc19.jpg?aki_policy=large',
    xl_picture_url: ''
  },
  host: {
    host_id: '51624384',
    host_url: 'https://www.airbnb.com/users/show/51624384',
    host_name: 'Giovanni',
    host_location: 'Hong Kong, Hong Kong',
    host_about: '',
    host_thumbnail_url: 'https://a0.muscache.com/im/pictures/264b82a7-756f-4da8-b607-
dc9759e2a10f.jpg?aki_policy=profile_small',
    host_picture_url: 'https://a0.muscache.com/im/pictures/264b82a7-756f-4da8-b607-
dc9759e2a10f.jpg?aki_policy=profile_x_medium',
    host_neighbourhood: 'Soho',
    host_is_superhost: false,
    host_has_profile_pic: true,
    host_identity_verified: false,
    host_listings_count: 1,
    host_total_listings_count: 1,
    host_verifications: [ 'email', 'phone', 'reviews', 'jumio', 'government_id' ]
  },
  address: {
    street: 'Hong Kong, Hong Kong Island, Hong Kong',
    suburb: 'Central & Western District',
    government_area: 'Central & Western',
    market: 'Hong Kong',
    country: 'Hong Kong',
```

To find all listings with the specified fields (listing_url, name, address, host_picture_url) in the listingsAndReviews collection that have a host with a picture URL, you can use the $match stage to filter documents with non-empty host_picture_url field. Then, you can use the $project stage to include only the desired fields in the output.

```
db.listingsAndReviews.aggregate([
{
$match: {
"host.host_picture_url": { $exists: true, $ne: null }
 }
},
{ $project: {
listing_url: 1,
name: 1,
address: 1, "host.host_picture_url": 1
}
}
])
```

Output:

```
{
   _id: '10059244',
   listing_url:  'https://www.airbnb.com/rooms/10059244',
   name: 'Ligne verte - à 15 min de métro du centre ville.',
   host: {
     host_picture_url:                'https://a0.muscache.com/im/pictures/user/c64773a6-da5f-4c1a-91fe-
f17131e4473b.jpg?aki_policy=profile_x_medium'
   },
   address: {
     street: 'Montréal, Québec, Canada',
     suburb: 'Hochelaga-Maisonneuve',
     government_area: 'Mercier-Hochelaga-Maisonneuve',
     market: 'Montreal',
     country: 'Canada',
     country_code: 'CA',
     location: {
       type: 'Point',
       coordinates: [ -73.54949, 45.54548 ],
       is_location_exact: false
     }
   }
```

b.**Using E-commerce collection write a query to display reviews summary.**

Let's assume we have an e-commerce collection named products containing documents representing different products, and each product document has an array field named reviews containing individual review documents with fields such as rating and comment.

```
db.products.aggregate([
  {
   $project: {
     _id: 1,
     name: 1,
     averageRating: { $avg: "$reviews.rating" },
     totalReviews: { $size: "$reviews" },
     highestRating: { $max: "$reviews.rating" },
     lowestRating: { $min: "$reviews.rating" }
   }
  }
])
```

This aggregation pipeline calculates several summary statistics for each product's reviews:
averageRating: Calculates the average rating of all reviews for each product.
totalReviews: Counts the total number of reviews for each product.
highestRating: Finds the highest rating among all reviews for each product.
lowestRating: Finds the lowest rating among all reviews for each product.

Output

```
[
  {
    "_id": 1,
    "name": "Product A",
    "averageRating": 4.5,
    "totalReviews": 10,
    "highestRating": 5,
    "lowestRating": 3
  },
  {
    "_id": 2,
    "name": "Product B",
    "averageRating": 3.8,
    "totalReviews": 5,
    "highestRating": 4,
    "lowestRating": 2
  },
]
```

Experiment :8

a. Demonstrate creation of different types of indexes on collection (unique, sparse, compound and multikey indexes)

b. Demonstrate optimization of queries using indexes

1. Unique Index:

A unique index ensures that the indexed field's values are unique across the collection.

Example:

Let's create a unique index on the username field in the users collection.

```
db.users.createIndex({ username: 1 }, { unique: true })
```

Output:

```
{
 "createdCollectionAutomatically" : false,
 "numIndexesBefore" : 1,
 "numIndexesAfter" : 2,
 "ok" : 1
}
```

2. Sparse Index:

A sparse index only includes documents in the index that have the indexed field. Let's create a sparse index on the email field in the users collection.

```
db.users.createIndex({ email: 1 }, { sparse: true })
```

Output:

json

```
{
 "createdCollectionAutomatically" : false,
 "numIndexesBefore" : 2,
 "numIndexesAfter" : 3,
 "ok" : 1
}
```

3. Compound Index:
A compound index includes multiple fields and supports queries on these fields collectively.

Example:
Let's create a compound index on the category and price fields in the products collection.

javascript

```
db.products.createIndex({ category: 1, price: -1 })
```

Output:

```
{
  "createdCollectionAutomatically" : false,
  "numIndexesBefore" : 1,
  "numIndexesAfter" : 2,
  "ok" : 1
}
```

4. Multikey Index:
A multikey index is used when a field contains an array of values.

Example:
Let's create a multikey index on the tags field in the posts collection.

```
db.posts.createIndex({ tags: 1 })
```

output

```
{
  "createdCollectionAutomatically" : false,
  "numIndexesBefore" : 1,
  "numIndexesAfter" : 2,
  "ok" : 1
}
```

b.Demonstrate optimization of queries using indexes

Suppose we have a collection named products containing documents representing various products with fields like name, category, and price. We'll create a compound index on the category and price fields to optimize queries that filter products by category and sort them by price.

Example:
Create Index:

javascript

```
db.products.createIndex({ category: 1, price: 1 })
```

Query Optimization:

Suppose we want to find all products in the "Electronics" category sorted by price in ascending order.

javascript

```
db.products.find({ category: "Electronics" }).sort({ price: 1 })
```

Output:
After creating the compound index and executing the optimized query, MongoDB will utilize the index to efficiently retrieve and sort the results.

Experiment :9

a. **Develop a query to demonstrate Text search using catalog data collection for a given word**
b. **Develop queries to illustrate excluding documents with certain words and phrases**

MongoDB provides a good technique that is text search. Using this technique we can find a piece of text or a specified word from the string fields. Or in other words. By using the $text operator with the $search parameter, we specify the word we want to search for.MongoDB will utilize a text index (if present) to efficiently search for the specified word across all indexed fields

First, we create a collection and add some documents to the collection:

In the following example, we are working with:

Database: aiml

Collection: content

```
> db.content.find().pretty()
{
        "_id" : ObjectId("603622eef19652db63812eb5"),
        "name" : "Rohit",
        "line" : "I love dogs and cats"
}
{
        "_id" : ObjectId("603622eef19652db63812eb6"),
        "name" : "Priya",
        "line" : "I love dogs and cats"
}
{
        "_id" : ObjectId("603622eef19652db63812eb7"),
        "name" : "Suman",
        "line" : "I dont like dogs and cats but i like cow"
}
>
```

Create index using createIndex() method. So we can search text over the name and line fields:

db.content.createIndex({name:"text",line:"text"})

```
> db.content.createIndex({name:"text",line:"text"})
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "ok" : 1
}
>
```

Search Text: we are going to search all the document which contain text love.

db.content.find({$text:{$search:"love"}})

```
> db.content.find({$text:{$search:"love"}})
{ "_id" : ObjectId("603622eef19652db63812eb6"), "name" : "Priya", "line" : "I lo
ve dogs and cats" }
{ "_id" : ObjectId("603622eef19652db63812eb5"), "name" : "Rohit", "line" : "I lo
ve dogs and cats" }
>
```

One more example in which we are going to search all the document which contain dog text:

db.content.find({$text:{$search:"dog"}})

```
> db.content.find({$text:{$search:"dog"}})
{ "_id" : ObjectId("603622eef19652db63812eb6"), "name" : "Priya", "line" : "I lo
ve dogs and cats" }
{ "_id" : ObjectId("603622eef19652db63812eb5"), "name" : "Rohit", "line" : "I lo
ve dogs and cats" }
{ "_id" : ObjectId("603622eef19652db63812eb7"), "name" : "Suman", "line" : "I do
nt like dogs and cats but i like cow" }
>
```

9b Develop queries to illustrate excluding documents with certain words and phrases

To exclude documents with certain words and phrases in MongoDB, we can use the $not operator along with regular expressions to negate the matching criteria.

Example 1: Excluding Documents Containing Specific Word

Suppose we have a collection named products containing documents with a description field. We want to exclude documents where the description contains the word "outdoor".

javascript

```
db.products.find({ description: { $not: /outdoor/ } })
```

This query will return all documents from the products collection where the description field does not contain the word "outdoor".

Example 2: Excluding Documents Containing Specific Phrase

Suppose we want to exclude documents where the description contains the phrase "waterproof case".

```
db.products.find({ description: { $not: /waterproof case/ } })
```

This query will return all documents from the products collection where the description field does not contain the phrase "waterproof case".

Experiment :10

Develop an aggregation pipeline to illustrate Text search on Catalog data collection

We can search text using the aggregation pipeline with the help of the $text query operator in the $match stage. But there are some restrictions for using the $text operator:

The first stage in the pipeline must be the $match stage that contains the $text operator.
Only once in the stage will a text operator occur.
The expression of the text operator cannot appear in expressions of $or or $not.

Database: aiml
Collection: people

```
> db.people.find().pretty()
{
        "_id" : ObjectId("603630cff19652db63812ebd"),
        "name" : "Nikhil",
        "pet" : "Dog"
}
{
        "_id" : ObjectId("603630cff19652db63812ebe"),
        "name" : "Anil",
        "pet" : "Dog"
}
{
        "_id" : ObjectId("603630cff19652db63812ebf"),
        "name" : "Rohit",
        "pet" : "Cat"
}
{
        "_id" : ObjectId("603630cff19652db63812ec0"),
        "name" : "Vikash",
        "pet" : "Cat"
}
{
        "_id" : ObjectId("603630cff19652db63812ec1"),
        "name" : "Suresh",
        "pet" : "Dog"
}
>
```

Count the number of the document in which pet value is a cat:

db.people.aggregate([{$match:{$text:{$search:"Cat"}}},{$group:{_id:null,total:{$sum:1}}}])

```
> db.people.aggregate([{$match:{$text:{$search:"Cat"}}},{$group:{_id:null,total:
{$sum:1}}}])
{ "_id" : null, "total" : 2 }
>
```

Count the number of the document in which pet value is a dog:

db.people.aggregate([{$match:{$text:{$search:"Dog"}}},{$group:{_id:null,total:{$sum:1}}}])

```
> db.people.aggregate([{$match:{$text:{$search:"Dog"}}},{$group:{_id:null,total:
{$sum:1}}}])
{ "_id" : null, "total" : 3 }
>
```

Return the Sorted result by using text score:

db.people.aggregate([{$match:{$text:{$search:"Dog"}}},{$sort:{score:{$meta:"textScore"}}},
{$project:{_id:0,name:1}}])

```
> db.people.aggregate([{$match:{$text:{$search:"Dog"}}},{$sort:{score:{$meta:"te
xtScore"}}},{$project:{_id:0,name:1}}])
{ "name" : "Suresh" }
{ "name" : "Nikhil" }
{ "name" : "Anil" }
>
```

Virtual Experiment – 1

Title: Student Management System using MongoDB

Aim

To design and implement a Student Management System using MongoDB to store, retrieve, update, and delete student records.

Software Required

- MongoDB Atlas (Cloud) or MongoDB Community Server
- MongoDB Compass / MongoDB Shell

Theory (Brief)

MongoDB is a NoSQL document-oriented database that stores data in JSON-like documents. It is suitable for handling semi-structured data such as student records efficiently.

Procedure

1. Create a database named `collegeDB`.
2. Create a collection named `students`.
3. Insert student documents with fields: USN, Name, Branch, Semester, and CGPA.
4. Retrieve student details based on branch and semester.
5. Update CGPA for a specific student.
6. Delete a student record.

Sample Commands

```
use collegeDB

db.students.insertMany([
  { usn: "2VD24CI001", name: "Aachal Chavan", branch: "CSE", sem: 4, cgpa: 8.5 },
  { usn: "2VD24CI002", name: "Aditya Patil", branch: "CSE", sem: 4, cgpa: 7.9 }
])

db.students.find({ branch: "CSE" })

db.students.updateOne(
  { usn: "2VD24CI002" },
  { $set: { cgpa: 8.2 } }
)

db.students.deleteOne({ usn: "2VD24CI001" })
```

Result

Student records were successfully inserted, retrieved, updated, and deleted using MongoDB.

Virtual Experiment – 2

Title: Online Book Store using MongoDB

Aim

To implement an Online Book Store database using MongoDB to manage book inventory and customer orders.

Software Required

- MongoDB Atlas
- MongoDB Compass / MongoDB Shell

Theory (Brief)

MongoDB supports flexible schema design, making it suitable for applications like online stores where product attributes may vary.

Procedure

1. Create a database named `bookStoreDB`.
2. Create a collection named `books`.
3. Insert book documents with title, author, price, and stock.
4. Search for books within a price range.
5. Update stock after purchase.
6. Remove out-of-stock books.

Sample Commands

```
use bookStoreDB

db.books.insertMany([
  { title: "MongoDB Basics", author: "John Doe", price: 450, stock: 10 },
  { title: "NoSQL Guide", author: "Jane Smith", price: 600, stock: 5 }
])

db.books.find({ price: { $lt: 500 } })

db.books.updateOne(
  { title: "MongoDB Basics" },
  { $inc: { stock: -1 } }
)

db.books.deleteMany({ stock: 0 })
```

Result

The Online Book Store database was successfully implemented and managed using MongoDB.

KLS Vishwanathrao Deshpande Institute of TechnologyHaliyal– 581329

Analysis & Design of Algorithms Lab [BCSL404]

for

IV Semester B.E.

As prescribed by

VISVESVARAYA TECHNOLOGICAL UNIVERSITY,BELAGAVI–590014

(For the Academic Year 2025-2026)

Prepared by

Prof. Ekata Shanbhag

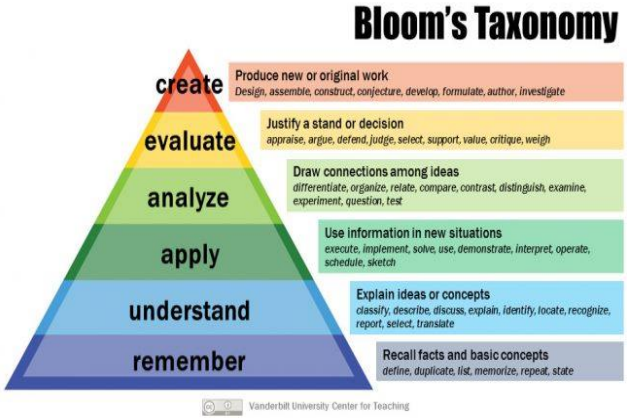
Department of Computer Science & Engineering(AIML)

Index

PROGRAM OUTCOMES(POs)

Program Outcomes as defined by NBA (PO) Engineering Graduates will be able to:

1. Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.



Vision (College)	
To nurture talent and enrich society through excellence in technical education, research and innovation.	
Mission (College)	
1. To augment innovative Pedagogy, kindle quest for interdisciplinary learning & to enhance conceptual understanding. 2. To build competence, professional ethics & develop entrepreneurial thinking. 3. To strengthen Industry Institute Partnership & explore global collaborations. 4. To inculcate culture of socially responsible citizenship. 5. To focus on Holistic & Sustainable development.	
Vision (Dept)	
To lead the way in the creation and application of cutting-edge Computer science and engineering (AIML) technologies, advancing the frontiers of knowledge, and empowering future generations to drive innovation and transformation on a global scale.	
Mission (Dept)	
To train students with a strong conceptual understanding using innovative pedagogies, empowering them to excel in the dynamic fields of Artificial Intelligence and Machine Learning. To imbibe professional, research, and entrepreneurial skills with a commitment to the nation's development at large. To strengthen the industry-institute Interaction. To promote life–long learning with a sense of societal & ethical responsibilities.	
Program Educational Objectives (PEO)	
PEO1	To develop an ability to identify and analyze the requirements of Computer Science and Engineering in design and providing novel engineering solutions.
PEO2	To develop abilities to work in team on multidisciplinary projects with effective communication skills, ethical qualities and leadership roles.

PEO3	To develop abilities for successful Computer Science Engineer and achieve higher career goals.
Program Specific Outcomes (PSO)	
PSO 1	To develop abilities to model real world problems using appropriate algorithms, computational theories and programming languages in the area of AI and ML..
PSO 2	To develop software applications and products in specialized areas of Artificial Intelligence and Machine Learning.

CO's And PO's Mapping Chart

Subject with code: DATA STRUCTURES LABORATORY (BCSL305) AY: 2023-24

Semester: III

S.No.	Description	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2
1	Develop programs to solve computational problems using suitable algorithm design strategy.	3											1	1	
2	Compare algorithm design strategies by developing equivalent programs and observing running times for analysis (Empirical).	3	2	1	1								1	1	
3	Make use of suitable integrated development tools to develop programs	3	2	1	1								1	1	
4	Choose appropriate algorithm design techniques to develop solution to the computational and complex problems	3	2	1	1								1	1	
5	Demonstrate and present the development of program, its execution and running time(s) and record the results/inferences	3	1	1	1								1	1	

Degree of compliance Low: 1 Medium: 2 High: 3

Evaluation:

	Particulars	Marks	Total
CIA	Performance	20	**30**
	Journal	05	
	Viva-voce	05	
	Addon Course	05	**20**
	Virtual Lab Experiments	05	
	Lab IA	10	
	Grand Total		**50**

Mapping of Experiments with CO, PO and PSO

Sl. No.	Experiment Details	CO	PO	PSO
1	Design and implement C/C++ Program to find Minimum Cost Spanning Tree of a given connected undirected graph using Kruskal's algorithm.	CO1	1	1
2	Design and implement C/C++ Program to find Minimum Cost Spanning Tree of a given connected undirected graph using Prim's algorithm.	CO1	1	1
3	3 a. Design and implement C/C++ Program to solve All-Pairs Shortest Paths problem using Floyd's algorithm. b. Design and implement C/C++ Program to find the transitive closure using Warshal's algorithm	CO1	1	1
4	Design and implement C/C++ Program to find shortest paths from a given vertex in a weighted connected graph to other vertices using Dijkstra's algorithm.	CO1	1	1
5	5 Design and implement C/C++ Program to obtain the Topological ordering of vertices in a given digraph	CO2	2	1
6	Design and implement C/C++ Program to solve 0/1 Knapsack problem using Dynamic Programming method	CO2	2	1
7	7 Design and implement C/C++ Program to solve discrete Knapsack and continuous Knapsack problems using greedy approximation method	CO2	2	1
8	Design and implement C/C++ Program to find a subset of a given set S = {sl , s2,.....,sn} of n positive integers whose sum is equal to a given positive integer d.	CO2	2	1
9	Design and implement C/C++ Program to sort a given set of n integer elements using Selection Sort method and compute its time complexity. Run the program for varied values of n> 5000 and record the time taken to sort. Plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.	CO3, CO5	3	1
10	Design and implement C/C++ Program to sort a given set of n integer elements using Quick Sort method and compute its time complexity. Run the program for varied values of n> 5000 and record the time taken to sort. Plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator	CO3, CO5	3	1
11	Design and implement C/C++ Program to sort a given set of n integer elements using Merge Sort method and compute its time complexity. Run the program for varied values of n> 5000, and record the time taken to sort. Plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.	CO3, CO5	3	1
12	Design and implement C/C++ Program for N Queen's problem using Backtracking	CO4	3	1

1. Design and implement C/C++ Program to find Minimum Cost Spanning Tree of a given connected undirected graph using Kruskal's algorithm.

DESCRIPTION:

Kruskal's algorithm is an algorithm in graph theory that finds a minimum spanning tree for a connectedweighted graph. This means it finds a subset of the edges that forms a tree that includes every vertex, where the total weight of all the edges in the tree is minimized. If the graph is not connected, then it finds a minimum spanning forest (a minimum spanning tree for each connected component). Kruskal'salgorithm is an example of a greedy algorithm

ALGORITHM:

Let G = (V, E) be the given graph, with | V| = n
{
Start with a graph T = (V,$ \phi$) consisting of only the
vertices of G and no edges; /* This can be viewed as n connected components, each vertex being one connected component */
Arrange E in the order of increasing costs;
for (i = 1, i$ \leq$n - 1, i + +)
{
Select the next smallest cost edge;
if (the edge connects two different connected components)
add the edge to T;
}
}

Program

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int i,j,k,a,b,u,v,n,ne=1;
int min,mincost=0,cost[9][9],parent[9];
int find(int);
int uni(int,int);
void main()
{
        clrscr();
        printf("\n\tImplementation of Kruskal's algorithm\n");
        printf("\nEnter the no. of vertices:");
```

```
        scanf("%d",&n);
        printf("\nEnter the cost adjacency matrix:\n");
        for(i=1;i<=n;i++)
        {
                for(j=1;j<=n;j++)
                {
                        scanf("%d",&cost[i][j]);
                        if(cost[i][j]==0)
                        cost[i][j]=999;
                }
        }
        printf("The edges of Minimum Cost Spanning Tree are\n");
        while(ne < n)
        {
        for(i=1,min=999;i<=n;i++)
                {
                        for(j=1;j <= n;j++)
                        {
                                if(cost[i][j] < min)
                                {
                                        min=cost[i][j];
                                        a=u=i;
                                        b=v=j;
                                }
                        }
                }
                u=find(u);
                v=find(v);
                if(uni(u,v))
                {
                        printf("%d edge (%d,%d) =%d\n",ne++,a,b,min);
                        mincost +=min;
                }
                cost[a][b]=cost[b][a]=999;
        }
        printf("\n\tMinimum cost = %d\n",mincost);
getch();
}
```

```
int find(int i)
{
        while(parent[i])
        i=parent[i];
        return i;
}
int uni(int i,int j)
{
if(i!=j)
{
        parent[j]=i;
        return 1;
}
return 0;
}
```

Output:

```
          Implementation of Kruskal's algorithm

 Enter the no. of vertices:7

 Enter the cost adjacency matrix:
 0 28 0 0 0 10 0
 28 0 16 0 0 0 14
 0 16 0 12 0 0 0
 0 0 12 0 22 0 18
 0 0 0 22 0 25 24
 10 0 0 0 25 0 0
 0 14 0 0 24 0 0
 The edges of Minimum Cost Spanning Tree are
 1 edge (1,6) =10
 2 edge (3,4) =12
 3 edge (2,7) =14
 4 edge (2,3) =16
 5 edge (4,5) =22
 6 edge (5,6) =25

          Minimum cost = 99
```

2. Design and implement C/C++ Program to find Minimum Cost Spanning Tree of a given connected undirected graph using Prim's algorithm.

Description:

The program uses prim's algorithm which is based on minimum spanning tree for a connected undirected graph.A predefinined cost adjecency matrix is the input.To find the minimum spanning tree, we choose the source node at random and in every step we find the node which is closest as well as having the least cost from the previously selected node.And also the cost of selected edge is being added to variable sum.Based on the value of sum, the presence of the minimum spanning tree is found.

Algorithm:

Input: A non-empty connected weighted graph with vertices V and edges E (the weights can be negative).

Initialize: V_{new} = {x}, where x is an arbitrary node (starting point) from V, E_{new} = {}

Repeat until V_{new} = V:

Choose an edge {u, v} with minimal weight such that u is in Vnew and v is not (if there are multiple edges with the same weight, any of them may be picked)

Add v to V_{new}, and {u, v} to E_{new}

Output: V_{new} and E_{new} describe a minimal spanning tree

Program

```
#include<stdio.h>
#include<conio.h>
int n,cost[10][10],temp,nears[10];
void readv();
void primsalg();
void readv()
{
        int i,j;
        printf("\n Enter the No of nodes or vertices:");
        scanf("%d",&n);
        printf("\n Enter the Cost Adjacency matrix of the given graph:");
        for(i=1;i<=n;i++)
        {
                for(j=1;j<=n;j++)
                {
                        scanf("%d",&cost[i][j]);
                        if((cost[i][j]==0) && (i!=j))
                        {
```

```
                                cost[i][j]=999;
                        }
                }
        }
}
void primsalg()
{
        int k,l,min,a,t[10][10],u,i,j,mincost=0;
        min=999;
        for(i=1;i<=n;i++) //To Find the Minimum Edge E(k,l)
        {
                for(u=1;u<=n;u++)
                {
                        if(i!=u)
                        {
                                if(cost[i][u]<min)
                                {
                                        min=cost[i][u];
                                        k=i;
                                        l=u;
                                }
                        }
                }
        }
t[1][1]=k;
t[1][2]=l;
printf("\n The Minimum Cost Spanning tree is...");
printf("\n(%d,%d)-->%d",k,l,min);
for(i=1;i<=n;i++)
{
         if(i!=k)
         {
                 if(cost[i][l]<cost[i][k])
                 {
                        nears[i]=l;
                 }
                 else
                 {
```

```
                        nears[i]=k;
                }
        }
}
nears[k]=nears[l]=0;
mincost=min;
for(i=2;i<=n-1;i++)
{
        j = findnextindex(cost,nears);
        t[i][1]=j;
        t[i][2]=nears[j];
        printf("\n(%d,%d)-->%d",t[i][1],t[i][2],cost[j][nears[j]]);
        mincost=mincost+cost[j][nears[j]];
        nears[j]=0;
        for(k=1;k<=n;k++)
        {
                if(nears[k]!=0 && cost[k][nears[k]]>cost[k][j])
                {
                        nears[k]=j;
                }
        }
}
printf("\n The Required Mincost of the Spanning Tree is:%d",mincost);
}
int findnextindex(int cost[10][10],int nears[10])
{
        int min=999,a,k,p;
        for(a=1;a<=n;a++)
        {
                p=nears[a];
                if(p!=0)
                {
                        if(cost[a][p]<min)
                        {
                                min=cost[a][p];
                                k=a;
                        }
                }
```

```
        }
        return k;
}
void main()
{
        clrscr();
        readv();
        primsalg();
        getch();
}
```

Output:

```
 Enter the No of nodes or vertices:7

 Enter the Cost Adjacency matrix of the given graph:0 28 0 0 0 10 0
28 0 16 0 0 0 14
0 16 0 12 0 0 0
0 0 12 0 22 0 18
0 0 0 22 0 25 24
10 0 0 0 25 0 0
0 14 0 0 24 0 0

 The Minimum Cost Spanning tree is...
(1,6)-->10
(5,6)-->25
(4,5)-->22
(3,4)-->12
(2,3)-->16
(7,2)-->14
 The Required Mincost of the Spanning Tree is:99
```

3a. Design and implement C/C++ Program to solve All-Pairs Shortest Paths problem using Floyd's algorithm

DESCRIPTION

The Floyd's algorithm is a graph analysis algorithm for finding shortest paths in a weighted graph with positive or negative edge weights (but with no negative cycles, see below) and also for finding transitive closure of a relation R. A single execution of the algorithm will find the lengths (summed weights) of the shortest paths between all pairs of vertices, though it does not return details of the paths themselves.

ALGORITHM:

```
let dist be a |V| x |V| array of minimum distances initialized to infinity
for each vertex v
  dist[v][v] <- 0
for each edge (u,v)
  dist[u][v] <- w(u,v)  // the weight of the edge (u,v)
for k from 1 to |V|
  for i from 1 to |V|
    for j from 1 to |V|
      if dist[i][j] > dist[i][k] + dist[k][j]
        dist[i][j] <- dist[i][k] + dist[k][j]
      end if
```

Program

```
#include<stdio.h>
#include<conio.h>
void readf();
void amin();
int cost[20][20],a[20][20];
int i,j,k,n;
void readf()
{
        printf("\n Enter the no of vertices:");
        scanf("%d",&n);
        printf("\n Enter the Cost of vertices:");
        for(i=0;i<n;i++)
        {
                for(j=0;j<n;j++)
                {
                        scanf("%d",&cost[i][j]);
                        if(cost[i][j]==0 && (i!=j))
```

```
                                cost[i][j]=999;
                        a[i][j]=cost[i][j];
                }
        }
}
void amin()
{
        for(k=0;k<n;k++)
        {
                for(i=0;i<n;i++)
                {
                        for(j=0;j<n;j++)
                        {
                                if(a[i][j]>a[i][k]+a[k][j])
                                {
                                        a[i][j]=a[i][k]+a[k][j];
                                }
                        }
                }
        }
        printf("\n The All pair shortest path is:");
        for(i=0;i<n;i++)
        {
                printf("\n");
                for(j=0;j<n;j++)
                {
                        printf("%d\t",a[i][j]);
                }
        }
}
void main()
{
        clrscr();
        readf();
        amin();
        getch();
}
```

Output:

```
Enter the no of vertices:3
Enter the Cost of vertices:
0 4 11
6 0 2
3 0 0
The All pair shortest path is:
0   4   6
5   0   2
3   7   0
```

3b. Design and implement C/C++ Program to find the transitive closure using Warshal's algorithm.

DESCRIPTION :

Warshall's algorithm determines whether there is a path between any two nodes in the graph. It does not give the number of the paths between two nodes. According to Warshall's algorith,a path exists between two vertices i, j, iff there is a path from i to j or there is a path from i to j through 1,..,k intermadiate nodes.

ALGORITHM:

```
n = |V|
t(0) = the adjacency matrix for G

for i in 1..n do
   t(0)[i,i] = True
end for

for k in 1..n do
   for i in 1..n do
      for j in 1..n do
         t(k)[i,j] = t(k-1)[i,j] OR
            (t(k-1)[i,k] AND t(k-1)[k,j])
      end for
   end for
end for
return t(n)
```

Program

```
#include<stdio.h>
const int MAX = 100;
void WarshallTransitiveClosure(int graph[MAX][MAX], int numVert);
int main(void)
{
   int i, j, numVert;
   int graph[MAX][MAX];
   printf("Warshall's Transitive Closure\n");
   printf("Enter the number of vertices : ");
   scanf("%d",&numVert);
```

```
   printf("Enter the adjacency matrix :-\n");
   for (i=0; i<numVert; i++)
      for (j=0; j<numVert; j++)
         scanf("%d",&graph[i][j]);
   WarshallTransitiveClosure(graph, numVert);
   printf("\nThe transitive closure for the given graph is :-\n");
   for (i=0; i<numVert; i++)
   {
      for (j=0; j<numVert; j++)
      {
         printf("%d\t",graph[i][j]);
      }
      printf("\n");
   }
   return 0;
}

void WarshallTransitiveClosure(int graph[MAX][MAX], int numVert)
{
   int i,j,k;
   for (k=0; k<numVert; k++)
   {
      for (i=0; i<numVert; i++)
      {
         for (j=0; j<numVert; j++)
         {
            if (graph[i][j] || (graph[i][k] && graph[k][j]))
               graph[i][j] = 1;
         }
      }
   }
}
```

OUTPUT

Enter the number of vertices : 4 Enter the adjacency matrix :- 0 0 1 0 0 0 0 1 1 0 0 0 0 1 0 0
The transitive closure for the given graph is :- 1 0 1 0
0 1 0 1
1 0 1 0
0 1 0 1
Warshall's Transitive Closure Enter the number of vertices : 4 Enter the adjacency matrix :-
0 1 1 0 1 0 0 1 1 0 0 1 0 1 1 0
The transitive closure for the given graph is :- 1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1

4. Design and implement C/C++ Program to find shortest paths from a given vertex in a weighted connected graph to other vertices using Dijkstra's algorithm.

DESCRIPTION:

To find the shortest path between points, the weight or length of a path is calculated as the sum of the weights of the edges in the path.

A path is a shortest path is there is no path from x to y with lower weight.

Dijkstra's algorithm finds the shortest path from x to y in order of increasing distance from x. That is, it chooses the first minimum edge, stores this value and adds the next minimum value from the next edge it selects.

It starts out at one vertex and branches out by selecting certain edges that lead to new vertices.

It is similar to the minimum spanning tree algorithm, in that it is "greedy", always choosing the closest edge in hopes of an optimal solution.

ALGORITHM:

```
function Dijkstra(Graph, source):
    for each vertex v in Graph:                 // Initializations
    dist[v]  := infinity ;                      // Unknown distance function from
                                                // source to v
    previous[v]  := undefined ;                 // Previous node in optimal path
    end for                                     // from source
    dist[source]  := 0 ;                        // Distance from source to source
    Q := the set of all nodes in Graph ;        // All nodes in the graph are
                                                // unoptimized - thus are in Q
    while Q is not empty:                       // The main loop
    u := vertex in Q with smallest distance in dist[] ;   // Source node in first case
    remove u from Q ;
    if dist[u] = infinity:
      break ;                                   // all remaining vertices are
    end if                                      // inaccessible from source

    for each neighbor v of u:                   // where v has not yet been
                                                // removed from Q.
      alt := dist[u] + dist_between(u, v) ;
      if alt < dist[v]:                         // Relax (u,v,a)
        dist[v]  := alt ;
        previous[v]  := u ;
        decrease-key v in Q;                    // Reorder v in the Queue
      end if
    end for
```

```
    end while
    return dist;
end function
```

Program

```
#include<ostream>
#include<cstdio>
using namespace std;

const int MAXNODES = 10,INF = 9999;

void fnDijkstra(int [][MAXNODES], int [], int [], int[], int, int, int);

int main(void)
{
   int n,cost[MAXNODES][MAXNODES],dist[MAXNODES],visited[MAXNODES],path[MAXNODES],i,j,source,dest;

   cout << "\nEnter the number of nodes\n";
   cin >> n;
   cout << "Enter the Cost Matrix\n" << endl;
   for (i=0;i<n;i++)
   for (j=0;j<n;j++)
      cin >> cost[i][j];


   for (source = 0; source < n; source++)
   {
   getchar();
   cout << "\n//For Source Vertex : " << source  << " shortest path to other vertices//"<< endl;
   for (dest=0; dest < n; dest++)
   {
      fnDijkstra(cost,dist,path,visited,source,dest,n);


      if (dist[dest] == INF)
         cout << dest << " not reachable" << endl;
      else
      {
         cout << endl;
         i = dest;
      do
         {
            cout << i << "<--";
```

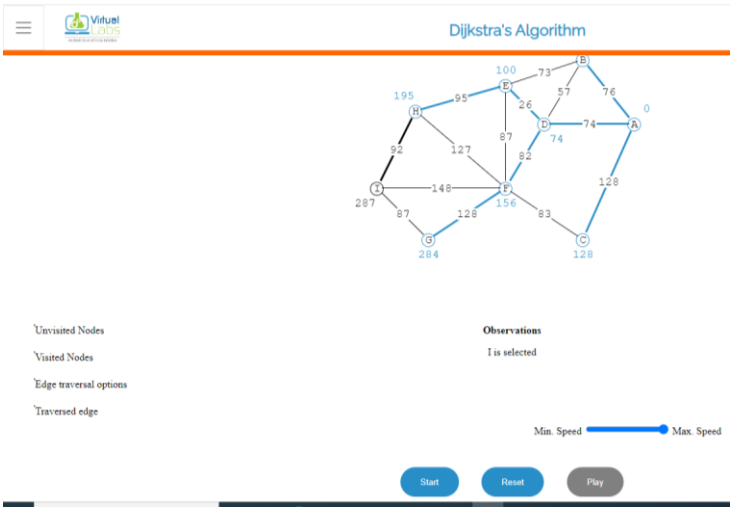
```
            i = path[i];
         }while (i!= source);
         cout << i << " = " << dist[dest] << endl;
      }
   }
   cout << "Press Enter to continue...";
   }

   return 0;
}

void fnDijkstra(int c[MAXNODES][MAXNODES], int d[MAXNODES], int p[MAXNODES], int s[MAXNODES], int so,
int de, int n)
{
   int i,j,a,b,min;

   for (i=0;i<n;i++)
   {
   s[i] = 0;
   d[i] = c[so][i];
   p[i] = so;
   }

   s[so] = 1;

   for (i=1;i<n;i++)
   {
   min = INF;
   a = -1;
   for (j=0;j<n;j++)
   {
      if (s[j] == 0)
      {
         if (d[j] < min)
         {
            min = d[j];
            a = j;
         }
      }
   }

   if (a == -1) return;

   s[a] = 1;
```

```
   if (a == de) return;

   for (b=0;b<n;b++)
   {
      if (s[b] == 0)
      {
         if (d[a] + c[a][b] <d[b])
         {
            d[b] = d[a] + c[a][b];
            p[b] = a;
         }
      }
   }
   }
}
```

Output

Enter the number of nodes 5 Enter the Cost Matrix
0 3 9999 7 9999 3 0 4 2 9999 9999 4 0 5 6 7 2 5 0 4 9999 9999 6 4 0
//For Source Vertex : 0 shortest path to other vertices//
0<--0 = 0
1<--0 = 3
2<--1<--0 = 7
3<--1<--0 = 5
4<--3<--1<--0 = 9 Press Enter to continue...
//For Source Vertex : 1 shortest path to other vertices//
0<--1 = 3
1<--1 = 0
2<--1 = 4
3<--1 = 2
4<--3<--1 = 6 Press Enter to continue...
//For Source Vertex : 2 shortest path to other vertices//
0<--1<--2 = 7
1<--2 = 4
2<--2 = 0
3<--2 = 5
4<--2 = 6 Press Enter to continue...
//For Source Vertex : 3 shortest path to other vertices//
0<--1<--3 = 5
1<--3 = 2
2<--3 = 5
3<--3 = 0
4<--3 = 4 Press Enter to continue...

//For Source Vertex : 4 shortest path to other vertices//
0<--1<--3<--4 = 9
1<--3<--4 = 6
2<--4 = 6
3<--4 = 4
4<--4 = 0 Press Enter to continue



5. Design and implement C/C++ Program to obtain the Topological ordering of vertices in a given digraph

DESCRIPTION :

Topological sorting for Directed Acyclic Graph (DAG) is a linear ordering of vertices such that for every directed edge uv, vertex u comes before v in the ordering.Topological Sorting for a graph is not possible if the graph is not a DAG. Input parameters: int a[MAX][MAX] - adjacency matrix of the input graph int n - no of vertices in the graph

ALGORITHM :

L Empty list that will contain the sorted elements S Set of all nodes with no incoming edges while S is non-empty do remove a node n from S add n to tail of L for each node m with an edge e from n to m do remove edge e from the graph if m has no other incoming edges then insert m into S if graph has edges then return error (graph has at least one cycle) else return L (a topologically sorted order)

CODE :

```
#include <stdio.h>
const int MAX = 10;
void fnTopological(int a[MAX][MAX], int n);
int main(void)
{
   int a[MAX][MAX],n;
   int i,j;

   printf("Topological Sorting Algorithm -\n");
   printf("\nEnter the number of vertices : ");
    scanf("%d",&n);

   printf("Enter the adjacency matrix -\n");
   for (i=0; i<n; i++)
      for (j=0; j<n; j++)
         scanf("%d",&a[i][j]);

   fnTopological(a,n);
   printf("\n");
   return 0;
}
```

```
void fnTopological(int a[MAX][MAX], int n)
{
   int in[MAX], out[MAX], stack[MAX], top=-1;
   int i,j,k=0;

   for (i=0;i<n;i++)
   {
      in[i] = 0;
      for (j=0; j<n; j++)
         if (a[j][i] == 1)
            in[i]++;
   }

   while(1)
   {
      for (i=0;i<n;i++)
      {
         if (in[i] == 0)
         {
            stack[++top] = i;
            in[i] = -1;
         }
      }

      if (top == -1)
         break;

      out[k] = stack[top--];

      for (i=0;i<n;i++)
      {
         if (a[out[k]][i] == 1)
            in[i]--;
      }
      k++;
   }

   printf("Topological Sorting (JOB SEQUENCE) is:- \n");
```

```
   for (i=0;i<k;i++)
      printf("%d ",out[i] + 1);
}
```

OUTPUT

Input Graph : 5 vertices 0 0 1 0 0 0 0 1 0 0 0 0 0 1 1 0 0 0 0 1 0 0 0 0 0

Topological Sorting (JOB SEQUENCE) is:- 2 1 3 4 5

6. Design and implement C/C++ Program to solve 0/1 Knapsack problem using Dynamic Programming method.

DESCRIPTION:

The Knapsack problem is probably one of the most interesting and most popular in computer science, especially when we talk about dynamic programming. The knapsack problem is a problem in combinatorial optimization. Given a set of items, each with a weight and a value, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible. It derives its name from the problem faced by someone who is constrained by a fixed-size knapsack and must fill it with the most valuable items.

ALGORITHM:

```
Input:
 Values (stored in array v or profit)
 Weights (stored in array w or weight)
 Number of distinct items (n)
 Knapsack capacity (W)
for j from 0 to W do
  m[0, j] = 0
end for
for i from 1 to n do
  for j from 0 to W do
    if w[i] <= j then
      m[i, j] = max(m[i-1, j], m[i-1, j-w[i]] + v[i])
    else
      m[i, j] = m[i-1, j]
    end if
  end for
end for
```

Program

```
#include <iostream>
#include <cstdlib>
using namespace std;
const int MAX = 10;
inline int max(int a, int b);
void fnProfitTable(int w[MAX], int p[MAX], int n, int c, int t[MAX][MAX]);
void fnSelectItems(int n,int c, int t[MAX][MAX], int w[MAX], int l[MAX]);<\pre>
   int main(void)
```

```
{
   int i, j, totalProfit;
   int weight[MAX];
   int profit[MAX];
   int capacity;
   int num;
   int loaded[MAX];
   int table[MAX][MAX];
   cout<<"Enter the maxium number of objects : ";
   cin >> num;
   cout << "Enter the weights : \n";
   for (i=1; i<=num; i++)
  {
   cout << "\nWeight " << i << ": ";
   cin >> weight[i];
  }
  cout << "\nEnter the profits : \n";
  for (i=1; i<=num; i++)
 {
   cout << "\nProfit " << i << ": ";
   cin >> profit[i];
 }
cout << "\nEnter the maximum capacity : ";
cin >> capacity;
totalProfit = 0;
for( i=1; i<=num; i++)
   loaded[i] = 0;
fnProfitTable(weight,profit,num,capacity,table);
fnSelectItems(num,capacity,table,weight,loaded);
cout << "Profit Matrix\n";
for (i=0; i<=num; i++)
{
   for(j=0; j<=capacity; j++)
   {
   cout <<"\t"<<table[i][j];
   }
   cout << endl;
}
```

```
   cout << "\nItem numbers which are loaded : \n{ ";
   for (i=1; i<=num; i++)
   {
      if (loaded[i])
      {
         cout <<i << " ";
         totalProfit += profit[i];
      }
   }
   cout << "}" << endl;
   cout << "\nTotal Profit : " << totalProfit << endl;
   return 0;
}
inline int max(int a, int b)
{
   return a>b ? a : b;
}


void fnProfitTable(int w[MAX], int p[MAX], int n, int c, int t[MAX][MAX])
{
   int i,j;

   for (j=0; j<=c; j++)
      t[0][j] = 0;

   for (i=0; i<=n; i++)
      t[i][0] = 0;

   for (i=1; i<=n; i++)
   {
      for (j=1; j<=c; j++)
      {
         if (j-w[i] < 0)
            t[i][j] = t[i-1][j];
         else
            t[i][j] = max( t[i-1][j], p[i] + t[i-1][j-w[i]]);
      }
```

```
   }
}
void fnSelectItems(int n,int c, int t[MAX][MAX], int w[MAX], int l[MAX])
{
   int i,j;

   i = n;
   j = c;

   while (i >= 1 && j >= 1)
   {
         if (t[i][j] != t[i-1][j])
         {
            l[i] = 1;
            j = j - w[i];
            i--;
         }
         else
            i--;
   }
}
```

OUTPUT:

Enter the maxium number of objects : 4 Enter the weights :
Weight 1: 2
Weight 2: 1
Weight 3: 3
Weight 4: 2
Enter the profits :
Profit 1: 12
Profit 2: 10
Profit 3: 20
Profit 4: 15
Enter the maximum capacity : 5 Profit Matrix 0 0 0 0 0 0 0 0 12 12 12 12 0 10 12 22 22 22 0 10 12 22 30 32 0 10 15 25 30 37
Item numbers which are loaded : { 1 2 4 }
Total Profit : 37

7. Design and implement C/C++ Program to solve discrete Knapsack and continuous Knapsack problems using greedy approximation method.

Program

```
#include <stdio.h>
#include <stdlib.h>
typedef struct {
   char *name;
   int weight;
   int value;
   int count;
} item_t;

item_t items[] = {
   {"map",                  9,   150,   1},
   {"compass",             13,    35,   1},
   {"water",              153,   200,   2},
   {"sandwich",            50,    60,   2},
   {"glucose",             15,    60,   2},
   {"tin",                 68,    45,   3},
   {"banana",              27,    60,   3},
   {"apple",               39,    40,   3},
   {"cheese",              23,    30,   1},
   {"beer",                52,    10,   3},
   {"suntan cream",        11,    70,   1},
   {"camera",              32,    30,   1},
   {"T-shirt",             24,    15,   2},
   {"trousers",            48,    10,   2},
   {"umbrella",            73,    40,   1},
   {"waterproof trousers",    42,    70,   1},
   {"waterproof overclothes", 43,    75,   1},
   {"note-case",           22,    80,   1},
   {"sunglasses",           7,    20,   1},
   {"towel",               18,    12,   2},
   {"socks",                4,    50,   1},
   {"book",                30,    10,   2},
};
```

```
int n = sizeof (items) / sizeof (item_t);

int *knapsack (int w) {
   int i, j, k, v, *mm, **m, *s;
   mm = calloc((n + 1) * (w + 1), sizeof (int));
   m = malloc((n + 1) * sizeof (int *));
   m[0] = mm;
   for (i = 1; i <= n; i++) {
      m[i] = &mm[i * (w + 1)];
      for (j = 0; j <= w; j++) {
         m[i][j] = m[i - 1][j];
         for (k = 1; k <= items[i - 1].count; k++) {
            if (k * items[i - 1].weight > j) {
               break;
            }
            v = m[i - 1][j - k * items[i - 1].weight] + k * items[i - 1].value;
            if (v > m[i][j]) {
               m[i][j] = v;
            }
         }
      }
   }
   s = calloc(n, sizeof (int));
   for (i = n, j = w; i > 0; i--) {
      int v = m[i][j];
      for (k = 0; v != m[i - 1][j] + k * items[i - 1].value; k++) {
         s[i - 1]++;
         j -= items[i - 1].weight;
      }
   }
   free(mm);
   free(m);
   return s;
}

int main () {
   int i, tc = 0, tw = 0, tv = 0, *s;
```

```
    s = knapsack(400);
    for (i = 0; i < n; i++) {
        if (s[i]) {
            printf("%-22s %5d %5d %5d\n", items[i].name, s[i], s[i] * items[i].weight, s[i] * items[i].value);
            tc += s[i];
            tw += s[i] * items[i].weight;
            tv += s[i] * items[i].value;
        }
    }
    printf("%-22s %5d %5d %5d\n", "count, weight, value:", tc, tw, tv);
    return 0;
}
```

Output:

```
map                    1    9   150
compass                1   13    35
water                  1  153   200
glucose                2   30   120
banana                 3   81   180
cheese                 1   23    30
suntan cream           1   11    70
waterproof overclothes 1   43    75
note-case              1   22    80
sunglasses             1    7    20
socks                  1    4    50
count, weight, value:    14  396  1010
```

8. Design and implement C/C++ Program to find a subset of a given set S = {sl , s2,.....,sn} of n positive integers whose sum is equal to a given positive integer d.

Desrcription:

In computer science, the subset sum problem is an important problem in complexity theory and cryptography. The problem is this: Given a set of integers, is there a non-empty subset whose sum is zero? For example, given the set {-7, -3, -2, 5, 8}, the answer is yes because the subset {-3, -2, 5} sums to zero. The problem is NP-complete. Input: The number of elements. The Weights of each element. Total Required Weight. Output:Subsets in which the sum of elements is equal to the given required weight(input).

Program

```
#include <iostream>
using namespace std;

// Constant definitions
const int MAX = 100;

// class definitions
class SubSet
{
   int stk[MAX], set[MAX];
   int size, top, count;
   public:
   SubSet()
   {
      top = -1;
      count = 0;
   }
   void getInfo(void);
   void push(int data);
   void pop(void);
   void display(void);
   int fnFindSubset(int pos, int sum);
};
void SubSet :: getInfo(void)
{
   int i;
   cout << "Enter the maximum number of elements : ";
```

```
   cin >> size;

   cout << "Enter the weights of the elements : \n";
   for (i=1; i<=size; i++)
      cin >> set[i];
}
void SubSet :: push(int data)
{
   stk[++top] = data;
}
void SubSet :: pop(void)
{
   top--;
}
void SubSet :: display()
{
   int i;
   cout << "\nSOLUTION #"<< ++count <<" IS\n{ ";
   for (i=0; i<=top; i++)
      cout << stk[i] << " ";
   cout << "}" << endl;
}
int SubSet :: fnFindSubset(int pos, int sum)
{
   int i;
   static int foundSoln = 0;

   if (sum>0)
   {
      for (i=pos; i<=size; i++)
      {
         push(set[i]);
         fnFindSubset(i+1, sum - set[i]);
         pop();
      }
   }
   if (sum == 0)
   {
```

```
      display();
      foundSoln = 1;
   }
   return foundSoln;
}
int main(void)
{
   int i,sum;
   SubSet set1;
   set1.getInfo();
   cout << "Enter the total required weight : ";
   cin >> sum;
   cout << endl;
   if (!set1.fnFindSubset(1, sum))
      cout << "\n\nThe given problem instance doesnt have any solution." << endl;
   else
      cout << "\n\nThe above-mentioned sets are the required solution to the given instance." << endl;
   return 0;
}
```

OUTPUT

SAMPLE 1

Enter the maximum number of elements : 5

Enter the weights of the elements :

1 2 3 4 5

Enter the total required weight : 5

SOLUTION #1 IS

{ 1 4 }

SOLUTION #2 IS

{ 2 3 }

SOLUTION #3 IS

{ 5 }

The above-mentioned sets are the required solution to the given instance.

SAMPLE 2

Enter the maximum number of elements : 4

Enter the weights of the elements :

1 2 3 4

Enter the total required weight : 11

The given problem instance doesnt have any solution.

9. Design and implement C/C++ Program to sort a given set of n integer elements using Selection Sort method and compute its time complexity. Run the program for varied values of n> 5000 and record the time taken to sort. Plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator

Algorithm

```
SELECTION SORT(arr, n)
  Step 1: Repeat Steps 2 and 3 for i = 0 to n-1
Step 2: CALL SMALLEST(arr, i, n, pos)
Step 3: SWAP arr[i] with arr[pos]
[END OF LOOP]
Step 4: EXIT
  SMALLEST (arr, i, n, pos)
Step 1: [INITIALIZE] SET SMALL = arr[i]
Step 2: [INITIALIZE] SET pos = i
Step 3: Repeat for j = i+1 to n
if (SMALL > arr[j])
    SET SMALL = arr[j]
SET pos = j
[END OF if]
[END OF LOOP]
Step 4: RETURN pos
```

Program

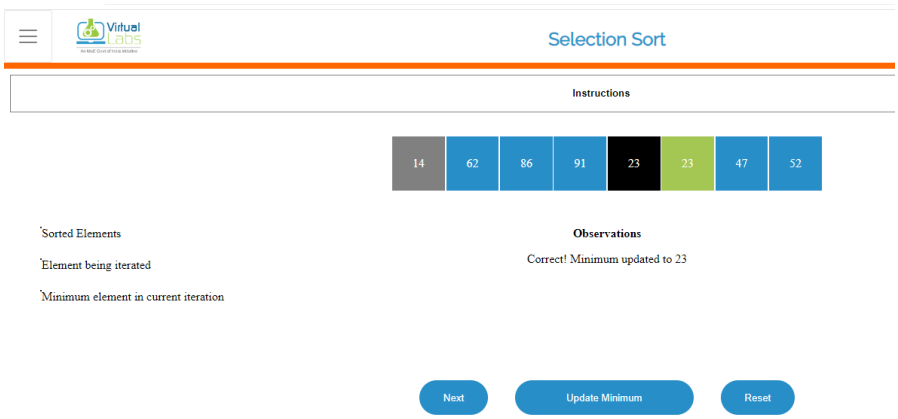
```
#include <stdio.h>
void selection_sort (int *a, int n) {
   int i, j, m, t;
   for (i = 0; i < n; i++) {
      for (j = i, m = i; j < n; j++) {
         if (a[j] < a[m]) {
            m = j;
         }
      }
      t = a[i];
      a[i] = a[m];
      a[m] = t;
   }
}
```

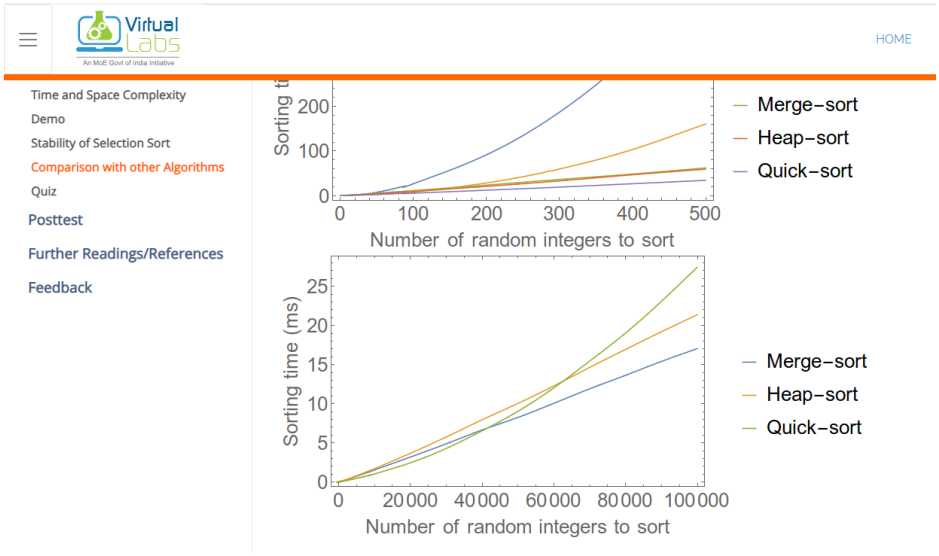
```
int main () {
   int a[] = {4, 65, 2, -31, 0, 99, 2, 83, 782, 1};
   int n = sizeof a / sizeof a[0];
   int i;
   for (i = 0; i < n; i++)
      printf("%d%s", a[i], i == n - 1 ? "\n" : " ");
   selection_sort(a, n);
   for (i = 0; i < n; i++)
      printf("%d%s", a[i], i == n - 1 ? "\n" : " ");
   return 0;
}
```

Output:

Input : 4 65 2 -31 0 99 2 83 782 1

Output: -31 0 1 2 2 4 65 83 99 782





Comparison with other sorting algorithms

Algorithm Sort	Algorithm Average	Time Best	Time Worst
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Insertion Sort	$O(n^2)$	O(n)	$O(n^2)$
Heap Sort	O(n*log(n))	O(n*log(n))	O(n*log(n))
Merge Sort	O(n*log(n))	O(n*log(n))	O(n*log(n))
Quick Sort	O(n*log(n))	O(n*log(n))	$O(n^2)$

10. Design and implement C/C++ Program to sort a given set of n integer elements using Quick Sort method and compute its time complexity. Run the program for varied values of n> 5000 and record the time taken to sort. Plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.

Description:

The program is based on the Quicksort algorithm which is an instantiation of divide and conquer method of solving the problem. Here the given array is partitioned every time and the sub-array is sorted. Dividing is based on an element called pivot. A divide and conquer algorithm works by recursively breaking down a problem into two or more sub-problems of the same (or related) type, until these become simple enough to be solved directly. The solutions to the sub-problems are then combined to give a solution to the original problem.

Program

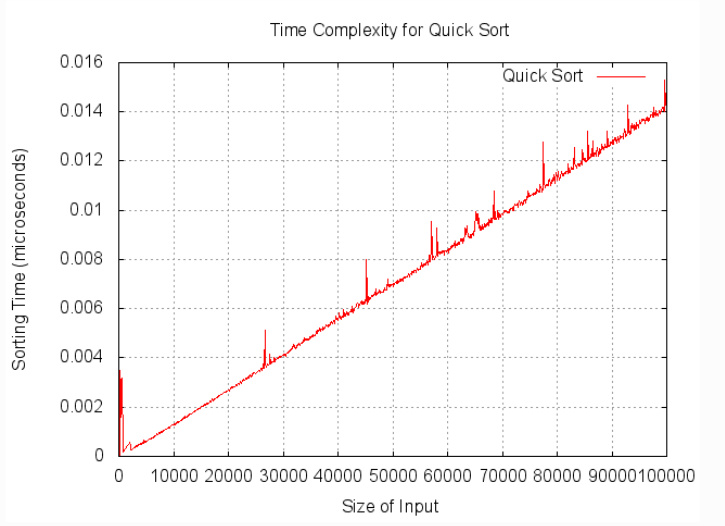
```
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <time.h>

void fnGenRandInput(int [], int);
void fnDispArray( int [], int);
int fnPartition(int [], int , int );
void fnQuickSort(int [], int , int );
inline void fnSwap(int*, int*);


inline void fnSwap(int *a, int *b)
{
  int t = *a; *a = *b; *b = t;
}
int main( int argc, char **argv)
{

  FILE *fp;
  struct timeval tv;
  double dStart,dEnd;
  int iaArr[500000],iNum,iPos,iKey,i,iChoice;

  for(;;)
  {
  printf("\n1.Plot the Graph\n2.QuickSort\n3.Exit");
  printf("\nEnter your choice\n");
  scanf("%d",&iChoice);
```

```
   switch(iChoice)
   {
      case 1:
         fp = fopen("QuickPlot.dat","w");

         for(i=100;i<100000;i+=100)
         {
            fnGenRandInput(iaArr,i);

            gettimeofday(&tv,NULL);
            dStart = tv.tv_sec + (tv.tv_usec/1000000.0);

            fnQuickSort(iaArr,0,i-1);

            gettimeofday(&tv,NULL);
            dEnd = tv.tv_sec + (tv.tv_usec/1000000.0);

            fprintf(fp,"%d\t%lf\n",i,dEnd-dStart);

         }
         fclose(fp);

         printf("\nData File generated and stored in file < QuickPlot.dat >.\n Use a plotting utility\n");
      break;

      case 2:
         printf("\nEnter the number of elements to sort\n");
         scanf("%d",&iNum);
         printf("\nUnsorted Array\n");
         fnGenRandInput(iaArr,iNum);
         fnDispArray(iaArr,iNum);
         fnQuickSort(iaArr,0,iNum-1);
         printf("\nSorted Array\n");
         fnDispArray(iaArr,iNum);
      break;

      case 3:
         exit(0);
   }
   }

   return 0;
}
```

```
int fnPartition(int a[], int l, int r)
{
   int i,j,temp;
   int p;

   p = a[l];
   i = l;
   j = r+1;

   do
   {
      do { i++; } while (a[i] < p);
      do { j--; } while (a[j] > p);

      fnSwap(&a[i], &a[j]);
   }
   while (i<j);

   fnSwap(&a[i], &a[j]);
   fnSwap(&a[l], &a[j]);

   return j;
}

void fnQuickSort(int a[], int l, int r)
{
   int s;

   if (l < r)
   {
      s = fnPartition(a, l, r);
      fnQuickSort(a, l, s-1);
      fnQuickSort(a, s+1, r);
   }
}

void fnGenRandInput(int X[], int n)
{
   int i;

   srand(time(NULL));
   for(i=0;i<n;i++)
   {
      X[i] = rand()%10000;
```

```
   }

}

void fnDispArray( int X[], int n)
{
   int i;

   for(i=0;i<n;i++)
      printf(" %5d \n",X[i]);

}
```

OUTPUT:



11. Design and implement C/C++ Program to sort a given set of n integer elements using Merge Sort method and compute its time complexity. Run the program for varied values of n> 5000, and record the time taken to sort. Plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.

DESCRIPTION:

Merge sort is an O(n log n) comparison-based sorting algorithm. Most implementations produce a stable sort, meaning that the implementation preserves the input order of equal elements in the sorted output. It is a divide and conquer algorithm.

ALGORITHM:

Mergesort(A[O .. n - 1])
Sorts array A[O .. n - 1] by recursive mergesort
Input: An array A[O .. n - 1] of orderable elements
Output: Array A[O .. n - 1] sorted in nondecreasing order
Merge(B[O .. p- 1], C[O .. q -1], A[O.. p + q -1])
Merges two sorted arrays into one sorted array
Input: Arrays B[O .. p -1] and C[O .. q -1] both sorted
Output: Sorted array A[O .. p + q -1] of the elements of Band C

Program

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <omp.h>

void simplemerge(int a[], int low, int mid, int high)
{
  int i,j,k,c[20000];
  i=low;
  j=mid+1;
  k=low;
  int tid;
  omp_set_num_threads(10);
  {
    tid=omp_get_thread_num();
    while(i<=mid&&j<=high)
    {
      if(a[i] < a[j])
```

```
        {
           c[k]=a[i];
           //printf("%d%d",tid,c[k]);
           i++;
           k++;
        }
        else
        {
           c[k]=a[j];
           //printf("%d%d", tid, c[k]);
           j++;
           k++;
        }
     }
  }
  while(i<=mid)
  {
     c[k]=a[i];
     i++;
     k++;
  }
  while(j<=high)
  {
     c[k]=a[j];
     j++;
     k++;
  }
  for(k=low;k<=high;k++)
  a[k]=c[k];
}

void merge(int a[],int low,int high)
{
  int mid;
  if(low < high)
  {
     mid=(low+high)/2;
     merge(a,low,mid);
```

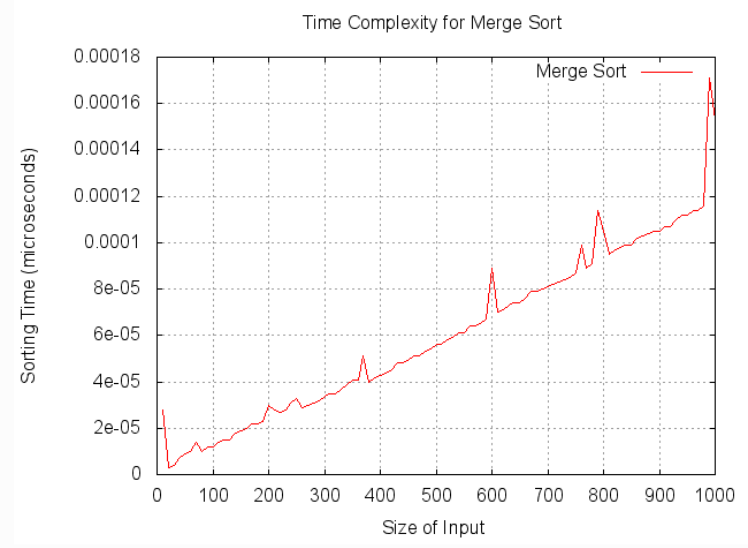
```
      merge(a,mid+1,high);
      simplemerge(a,low,mid,high);
   }
}

void getnumber(int a[], int n)
{
   int i;
   for(i=0;i < n;i++)
      a[i]=rand()%100;
}

int main()
{
   FILE *fp;
   int a[2000],i;
   struct timeval tv;
   double start, end, elapse;
   fp=fopen("mergesort.txt","w");
   for(i=10;i<=1000;i+=10)
   {
      getnumber(a,i);
      gettimeofday(&tv,NULL);
      start=tv.tv_sec+(tv.tv_usec/1000000.0);
      merge(a,0,i-1);
      gettimeofday(&tv,NULL);
      end=tv.tv_sec+(tv.tv_usec/1000000.0);
      elapse=end-start;
      fprintf(fp,"%d\t%lf\n",i,elapse);
   }
   fclose(fp);
   system("gnuplot");
   return 0;
}
```

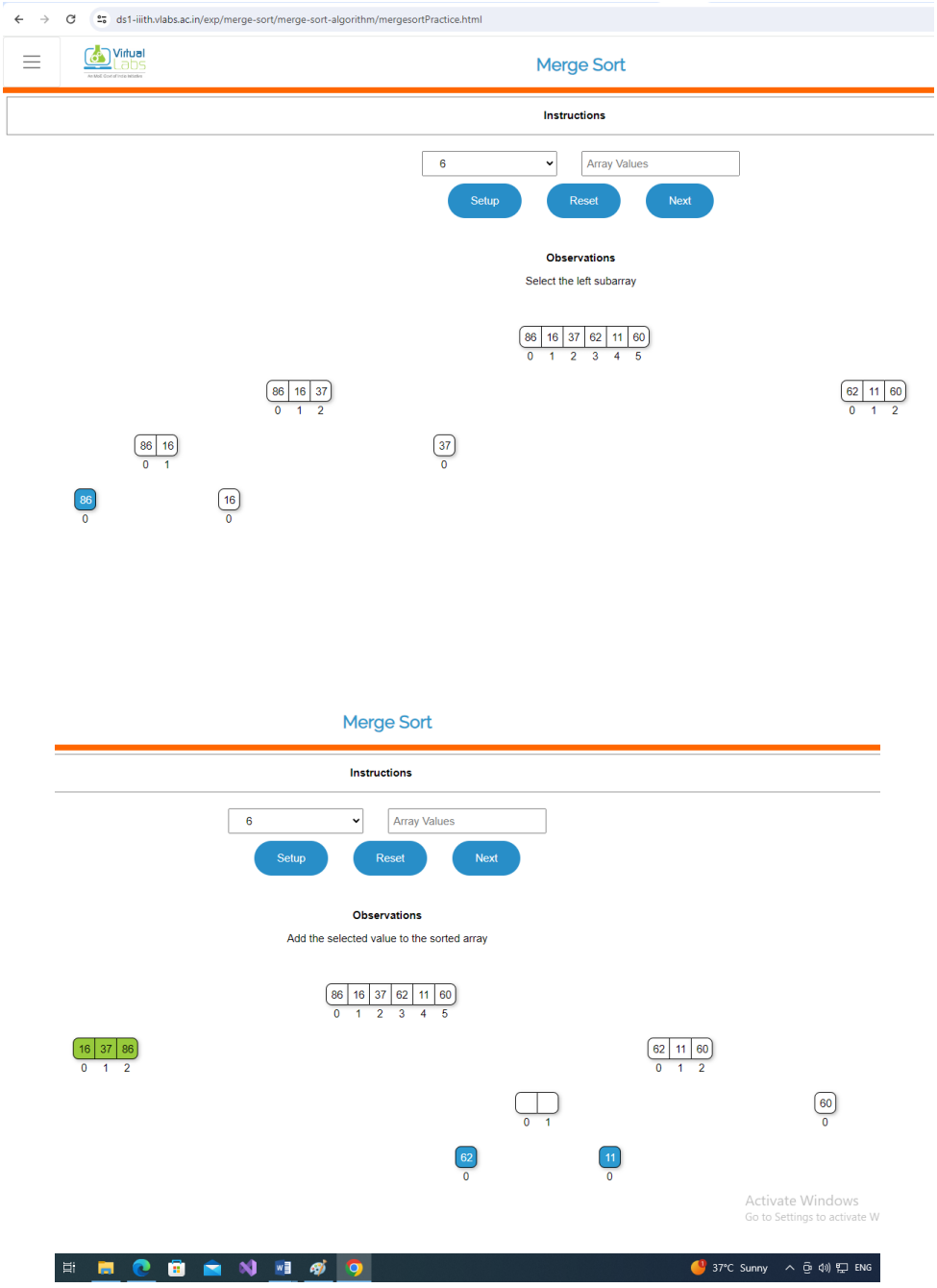
mergesort.gpl

Gnuplot script file for plotting data in file "mergesort.txt" This file is called mergesort.gpl

```
set terminal png font arial
set title "Time Complexity for Merge Sort"
set autoscale
set xlabel "Size of Input"
set ylabel "Sorting Time (microseconds)"
set grid
set output "mergesort.png"
plot "mergesort.txt" t "Merge Sort" with lines
```

OUTPUT




ds1-iiith.vlabs.ac.in/exp/merge-sort/merge-sort-algorithm/mergesortPractice.html
Merge Sort
Instructions
6
Array Values
Setup
Reset
Next
Observations
Select the left subarray
86 16 37 62 11 60
0 1 2 3 4 5
86 16 37
0 1 2
62 11 60
0 1 2
86 16
0 1
37
0
86
0
16
0
Merge Sort
Instructions
6
Array Values
Setup
Reset
Next
Observations
Add the selected value to the sorted array
86 16 37 62 11 60
0 1 2 3 4 5
16 37 86
0 1 2
62 11 60
0 1 2
60
0
62
0
11
0
Activate Windows
Go to Settings to activate W
37°C Sunny
ENG

12 Design and implement C/C++ Program for N Queen's problem using Backtracking

```
#include<stdio.h>
#include<math.h>
 int board[20],count;

int main()
{
int n,i,j;
void queen(int row,int n);

printf(" - N Queens Problem Using Backtracking -");
printf("\n\nEnter number of Queens:");
scanf("%d",&n);
queen(1,n);
return 0;
}
 //function for printing the solution
void print(int n)
{
int i,j;
printf("\n\nSolution %d:\n\n",++count);

for(i=1;i<=n;++i)
  printf("\t%d",i);

for(i=1;i<=n;++i)
{
  printf("\n\n%d",i);
  for(j=1;j<=n;++j) //for nxn board
  {
   if(board[i]==j)
    printf("\tQ"); //queen at i,j position
   else
    printf("\t-"); //empty slot
  }
}
}
```

```
/*funtion to check conflicts
If no conflict for desired postion returns 1 otherwise returns 0*/
int place(int row,int column)
{
int i;
for(i=1;i<=row-1;++i)
{
  //checking column and digonal conflicts
  if(board[i]==column)
   return 0;
  else
   if(abs(board[i]-column)==abs(i-row))
    return 0;
}

return 1; //no conflicts
}
 //function to check for proper positioning of queen
void queen(int row,int n)
{
int column;
for(column=1;column<=n;++column)
{
  if(place(row,column))
  {
   board[row]=column; //no conflicts so place queen
   if(row==n) //dead end
    print(n); //printing the board configuration
   else //try queen with next position
    queen(row+1,n);
  }
}
}
```

Output :

0 0 1 0

1 0 0 0

0 0 0 1

0 1 0 0

Virtual Lab

1. Depth First Search

Learning Objectives of the Experiment

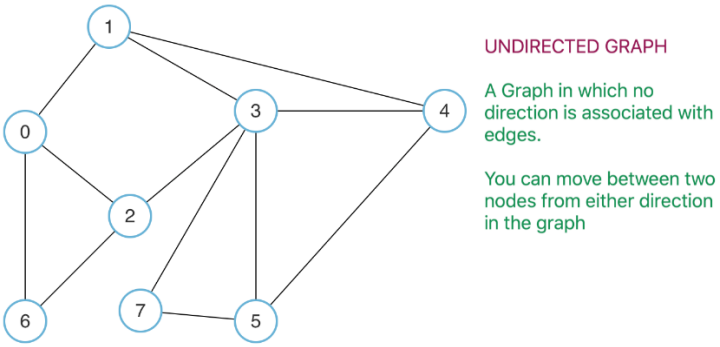
In this experiment, we will be learn about:

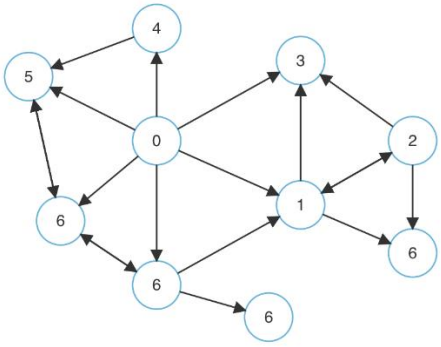
1. Understand the basics of graphs and their representations.
2. Understand the working of Depth First Traversal Algorithm for searching nodes.
3. Given a graph, understand the progression of the Depth First Traversal Algorithm and search for particular nodes.
4. Demonstrate the knowledge of time complexity of the Depth First Search Traversal algorithm.

Definition

A graph is a pictorial representation of a set of objects where some pairs of objects are connected by links. The interconnected objects are represented by points termed as vertices, and the links that connect the vertices are called edges.

Undirected Graph



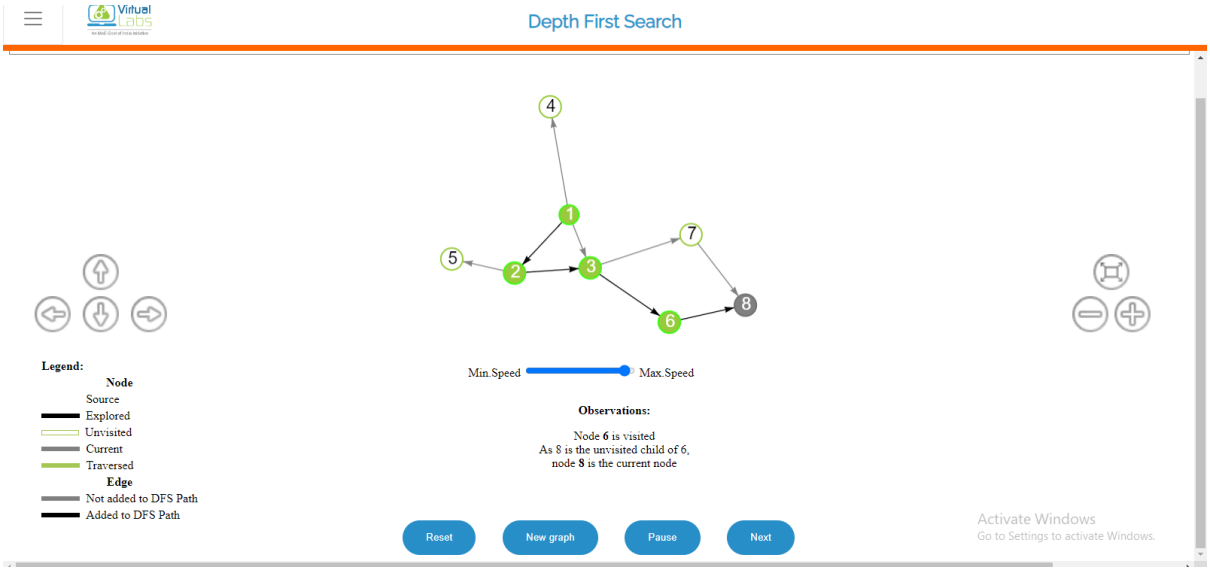


DIRECTED GRAPH

A Graph in which a direction is associated with each edge between a pair of vertices.

You can move from one node to another only if the edge is directed in that way.

Directed Graph



Breadth First Search

Learning Objectives of the Experiment

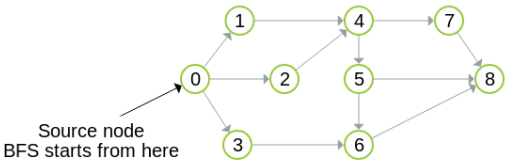
In this experiment, we will be able to do the following:

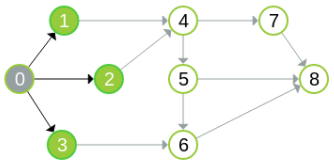
1. Study the basic concepts of Graphs and Queues and their representation.
2. Study the various Graph traversal algorithms and difference between them.
3. Understand the BFS Graph traversal using queues.
4. Demonstrate knowledge of time complexity and space complexity of performing a BFS on a given graph.

Definition

Breadth First Search (BFS) is a technique for traversing a finite graph. BFS visits the neighbour vertices before visiting the child vertices, and a queue is used in the search process. This algorithm is often used to find the shortest path from one vertex to another.

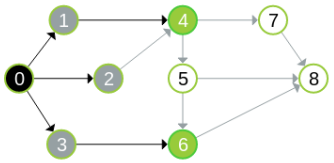
Here we start traversing from a selected node (source node) and traverse the graph layer/level wise thus exploring the neighbour nodes (nodes which are directly connected to source node). We must move to the next layer/level only after traversing the current layer/level completely.





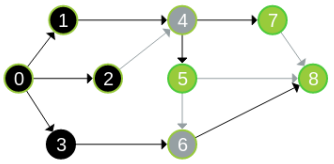
Currently on Level 0 (i.e node '0') and exploring its children present on Level 1 (i.e nodes '1' ,'2' ,'3')

Current node
Visited node
Explored node



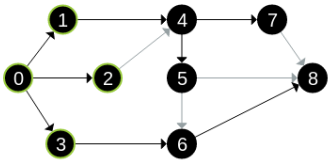
Nodes present on Level 0 are explored (i.e node '0')

Currently on Level 1 (i.e nodes '1', '2', '3') and exploring its children present on Level 2 (i.e nodes '4', '6')



Nodes present on Level 1 are explored (i.e nodes '1', '2', '3')

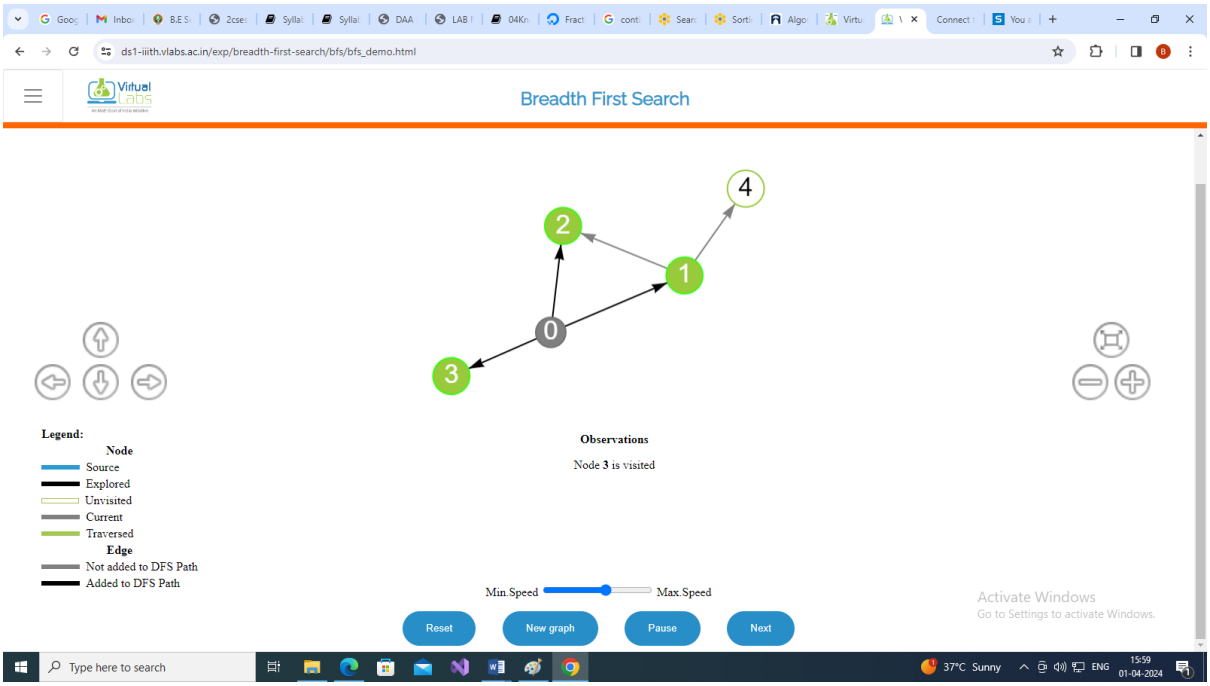
Currently on Level 2 (i.e nodes '4', '6') and exploring its children present on Level 3 (i.e nodes '5', '7', '8')

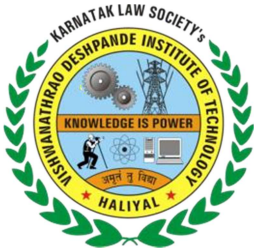


Nodes present on Level 2 are explored (i.e nodes '4', '6')

Currently on Level 3 (i.e nodes '5', '7', '8')

As there are no children for the nodes present on Level 3,they are considered as explored and BFS is done on node '0'


ds1-iiith.vlabs.ac.in/exp/breadth-first-search/bfs/bfs_demo.html
Virtual Labs
Breadth First Search
4
2
1
0
3
Legend:
Node
Source
Explored
Unvisited
Current
Traversed
Edge
Not added to DFS Path
Added to DFS Path
Observations
Node 3 is visited
Min.Speed
Max.Speed
Reset
New graph
Pause
Next
Activate Windows
Go to Settings to activate Windows.
Type here to search
37°C Sunny
ENG
15:59
01-04-2024



Karnatak Law Society's
Vishwanathrao Deshpande Institute of Technology
Haliyal – 581329

DATABASE MANAGEMENT SYSTEMS
[BCS403]
for
IVth Semester B.E.

As prescribed by

VISVESVARAYA TECHNOLOGICAL UNIVERSITY,
BELAGAVI – 590014
Academic Year: 2025-26

Prepared by

Prof. Bheerappa sasanoor

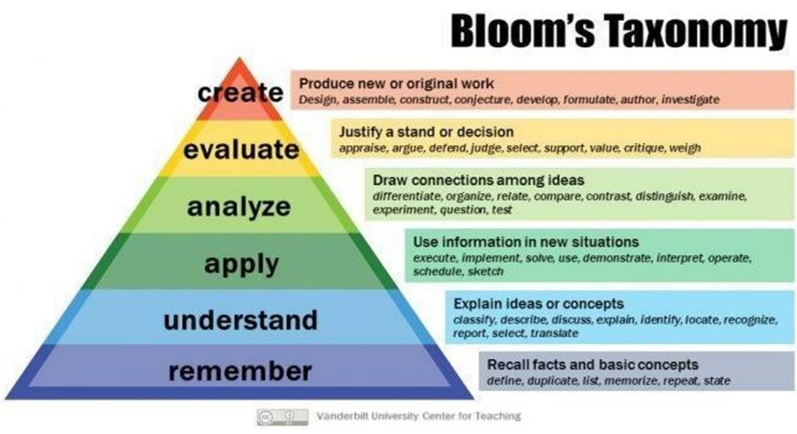
Department of Computer Science& Engineering (AI & ML)

Index

PROGRAM OUTCOMES(POs)

Program Outcomes as defined by NBA (PO) Engineering Graduates will be able to:

1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.



11. **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

Vision (College)	
To grow beyond leaps and bounds as an Institute of par Excellence in the Arena of Technical Education developing Human Resources of High calibre with sound character	
Mission (College)	
To train the students to Emerge as outstanding skilled Technocrats Imbibed with Professional Ethics and Managerial skills with Commitment to the Society and Nation at Large	
Vision (Dept)	
	*To grow beyond leaps and bounds as a department of par excellence in the arena of Computer Science and Engineering education in developing human resources meeting global standards.*
Mission (Dept)	
To train students with strong theoretical foundation and practical knowledge, to imbibe professional ethics and managerial skills with commitment to the society.. 	
Program Educational Objectives (PEO)	
PEO1	To develop an ability to identify and analyze the requirements of Computer Science and Engineering in design and providing novel engineering solutions.
PEO2	To develop abilities to work in team on multidisciplinary projects with effective communication skills, ethical qualities and leadership roles.
PEO3	To develop abilities for successful Computer Science Engineer and achieve higher career goals.
Program Specific Outcomes (PSO)	
PSO 1	To develop ability to model real world problems using appropriate data structure and suitable algorithm in the area of Data Processing, System Engineering, Networking for varying complexity.
PSO 2	To develop an ability to use modern computer languages, environments and platforms in creating innovative career.

INTRODUCTION TO DBMS COMMAND'S

INTRODUCTION TO ORACLE

SQL

SQL stands for Structured Query Language. SQL is used to create, remove, alter the database and database objects
in a database management system and to store, retrieve, update the data in a database.
SQL is a standard language
for creating, accessing, manipulating database management system. SQL works for all modern relational database
management systems, like SQL Server, Oracle, MySQL, etc.

Different types of SQL commands

SQL commands can be categorized into five categories based on their functionality

Different types of SQL commands

SQL commands can be categorized into five categories based on their functionality

DDL (Data Definition language)

A Data Definition Language (DDL) statement is used to define the database structure or schema.
Aim: To study and execute the DDL commands in RDBMS.
DDL commands:

1. CREATE
2. ALTER
3. DROP
4. RENAME
5. TRUNCATE

1. CREATE Command

CREATE is a DDL command used to create databases, tables, triggers and other database objects.

Syntax to create a new table:

CREATE
TABLE
table_name (

column_Name1 data_type (**size of** the **column**) , column_Name2 data_type (

size of the **column**) , column_Name3
data_type (**size of** the **column**) ,

column_NameN data_type (**size of** the **column**)

) ;

Suppose, you want to create a **Student** table with five columns in the SQL database. To do this, you have to write the following DDL command:

Example 1:

```
CREATE
   TABLE
   Student (
   Roll_No. int ,
   First_Name
   varchar (20) ,
   Last_Name
   varchar (20) ,
   Age int ,
   Marks int,
   Dob Date
   ) ;
```

SQL>**desc** Student;

Example 2:

```
create table Employee (
 empid varchar(10), empname varchar(20) , gender varchar(7),
  age number(3), dept varchar(15) , doj Date
 );
```

SQL> desc Employee

Example 3:

```
create table BOOK
( Book_id varchar(4), Title varchar(10),
```

Publisher_name varchar(10), Pub_year int
);

SQL> desc BOOK;

ALTER

This command is used to add, delete or change columns in the existing table. The user needs to know the existing table name and can do add, delete or modify tasks easily.

Syntax:–

ALTER TABLE table_name ADD column_name datatype;

.ADD:

SQL> alter table employee add(designation varchar(15)); Table altered.

II.MODIFY

SQL> alter table employee modify (designation varchar(20)); Table altered

Example 1:

ALTER TABLE Student ADD CGPA number; SQL>desc Student; **Example 2:**

ALTER TABLE Employee ADD Salary number;

SQL>desc Employee;

Example 3:

ALTER TABLE BOOK

ADD Author_nmae varchar(20); SQL>desc Student;

RENAME:

It is possible to change name of table with or without data in it using simple RENAME command.

We can rename any table object at any point of time.

Syntax –

RENAME <Table Name> To <New_Table_Name>;

Example:

RENAME TABLE Employee To EMP;

2. TRUNCAT:

This command is used to remove all rows from the table, but the structure of the table still

exists.

Syntax

Syntax to remove an existing table.

TRUNCATE TABLE table_name;

Example:

TRUNCATE TABLE Student;

3. DROP

This command is used to remove an existing table along with its structure from the Database.

Syntax–

Syntax to drop an existing table. DROP TABLE table_name;

Example: DROP TABLE Student_info;

DML(DATA MANIPULATION LANGUAGE):

Data manipulation language allows the users to query and manipulate data in existing schema in object.

It allows following data to insert, delete, update and recovery data in schema object.

DML COMMANDS:

- ❖ INSERT
- ❖ UPDATE
- ❖ DELETE

1. INSERT

This command is used to enter the information or values into a row. We can connect one or more records to a single table within a repository using this instruction.

Syntax:

Insert into Table_ Name Values(column1, column2, ...);

Example:

```
CREATE TABLE Student (
Roll_No int ,
First_Name varchar (20) , Last_Name varchar (20) , Marks int,
Dob Date
) ;
```

SQL>desc Student;

SQL> insert into Student values('01','Adit,'k ',25,'11-02-2004');

SQL> insert into Student values(02,"Arpitha","S", 20,'21-12-2003'); SQL> insert into Student values(03,"Jorge","D", 20, 18,'10-08-2001');

Insert 2 more rows.

SQL>desc Student;

2. UPDATE

This allows the user to update the particular column value using the where clause condition. This command is used to alter existing table records.

Syntax:

UPDATE <table_ name>

SET <column_ name = value> WHERE condition;

Example:

UPDATE Students SET Marks= 21
WHERE First_ name = "Arpitha";

SQL>desc Students;

3. DELETE

a) Delete some rows

DELETE statement is used to delete rows from a table. Generally DELETE statement removes one or more records from a table.

Syntax:

DELETE FROM table_name WHERE condition;

Example:

Delete from Students where Roll_no='111';

b) Delete All rows:

- It will remove all the rows from the table and does not free the space contained by the table.

Syntax:

DELETE FROM table_name;

Example:

Delete from student;

DQL(Data Query Language)

DQL stands for the. DQL command is used for fetching the data. DQL command is used for selecting data from the table, view, temp table, table variable, etc. There is only one command under DQL which is the SELECT command.

Syntax :**SELECT * FROM Employee;**

The SELECT statement can be used in many ways.

1. Selecting some columns :

To select specified number of columns from the table the Following command is used.

Syntax:

SELECT column_ name FROM table_ name;

Example:

Select First_ name, Last_ name from Students;

SQL> desc Students;

2. Select All Columns:

To select all columns from the table * is used instead of column names.

Syntax:

SELECT * FROM table_name;

Example:

Select * from Students;

SQL> desc Students;

3. Select using DISTINCT:

The DISTINCT keyword is used to return only different values (i.e.) this command does not select the duplicate values from the table.

Syntax:

SELECT DISTINCT column name(s) FROM table_name;

Example:

SELECT DISTINCT Roll_No FROM Students;

4. Select using IN:

If you want to get the rows which contain certain values, the best way to do it is to use the IN conditional expression.

Syntax:

SELECT column name(s) FROM table_name

WHERE Column name IN (value1, value2,……,value-n);

Example:

SELECT * FROM students

WHERE students_name IN (Arpitha, Jorge);

5. Select using BETWEEN:

BETWEEN can be used to get those items that fall within a range.

Syntax:

SELECT column name FROM table_name

WHERE Column name BETWEEN value1 AND value2;

Example:

SELECT * FROM **student** WHERE mark **BETWEEN** 80 and 100;

6. Renaming:

The select statement can be used to rename either a column or the entire table. Syntax:

Renaming a column:

SELECT column name AS new name FROM table_name;

Example:

SELECT **First_name** As **Name** FROM Students;

Renaming a table:

RENAME old_table_name TO **new_table_name;**

Example:

RENAME Student TO **Stu_details;**

7. SELECT DATE

It is used to retrieve a date from a database. If you want to find a particular date from a database, you can use this statement.

Syntax:

SELECT Column_Names(S) from **table-name WHERE** condition(date_column);

Example: **SELECT * FROM Students WHERE** DOB >= '11-12-2000';

8. SELECT NULL

Null values are used to represent missing unknown data. There can be two conditions:

1. Where SQL is NULL
2. Where SQL is NOT NULL

If in a table, a column is optional, it is very easy to insert data in column or update an existing record without adding a value in this column. This means that field has null value.

Where SQL is NULL:

Syntax:

```
SELECT COLUMN_NAME(S) FROM
TABLE_NAME WHERE COLUMN_NAME
IS NULL;
```

Example:

```
SELECT Student_name, Marks FROM STUDENTS

WHERE MARKS IS NULL;
```

Where SQL is NOT NULL:

Syntax:

```
SELECT COLUMN_NAME(S) FROM
TABLE_NAME WHERE COLUMN_NAME
IS NOT NULL;
```

Example:

```
SELECT Student_name, Marks FROM STUDENTS

WHERE MARKS IS NOT NULL;
```

9. ORDER BY Clause

ORDER BY is a clause in SQL which shows the result-set of the **SELECT statement in either ascending or descending order.**

with one row Syntax:

```
SELECT Column_Name FROM
Table_Name ORDER BY
Column_Name;
```

Example:

```
SELECT * FROM Students ORDER BY Reg_no;
```

a) Wit Multiple row Syntax:

SELECT Column_Name(S) FROM Table_Name ORDER BY Column_Name(S);

Example:

SELECT reg_no,first_name,marks FROM Students ORDER BY first_name, marks;

b) Ascending Order(ASC)/Descending Order(DESC)

c) SELECT Column_Name(S) FROM Table_Name ORDER BY Column_Name(S) ASC;

Example:

SELECT * FROM Students ORDER BY Marks ASC;

d) with WHER E Clause Syntax:

SELECT Column_Name(S)FROM Table_Name

WHERE [condition] ORDER BY Column_Name [ASC | DESC];

Example:

SELECT * FROM Students WHERE Marks >80 ORDER BY Marks;

GROUP BY clause

GROUP BY is an SQL keyword used in the SELECT query for arranging the same values of a column in the group by using SQL functions.

Syntax:

SELECT Column_Name(S) **FROM** Table_Name

GROUP BY

Column_Name(S);

Example:

SELECT COUNT (marks), Student_name **GROUP BY** marks;

a) with MIN clause

Group by with MIN clause shows the minimum value for the given where clause.

Syntax:

SELECT MIN(Column_Name) **FROM** Table_Name; Example:

SELECT MIN (Marks) FROM Students;

b) with MAX clause

Group by with MIN clause shows the minimum value for the given where clause.

Syntax:

SELECT Column_Name, **MAX**(Column_Name) **FROM** Table_Name; Example:

SELECT Stu_Subject, MAX (Marks) FROM Students;

c) COUNT() Function

The COUNT() function returns the number of rows in a database table. Syntax:

COUNT(*)

or

COUNT([ALL|DISTINCT] expression)

Example:

1. **SELECT COUNT(*)**
   FROM Student;
2. **SELECT COUNT(Marks)**
   FROM Student
   GROUP BY marks
   HAVING COUNT(*) > 50;

d) SUM() clause

It is used to return the total summed value of an expression.

Syntax:

SELECT SUM(aggregate_expression)
FROM tables
[**WHERE** conditions];

Example:

SELECT SUM(Marks)
FROM Student
Where roll_no='111';

DCL (DATA CONTROL LANGUAGE)

DCL stands for data control language. DCL commands are used for providing and taking back the access rights on the database and database objects. DCL command used for controlling user's access on the data. Most used DCL commands are GRANT and REVOKE

GRANT

used to provide access right to the user.

Syntax: GRANT INSERT, DELETE ON Employee TO user;

REVOKE

REVOKE command is used to take back access right from the user, it cancels access right of the user from the database object.

Syntax

REVOKE ALL ON Employee FROM user;

TCL (Transaction Control Language)

TCL commands are used for handling transactions in the database. Transactions ensure data integrity in the multi-user environment.

TCL commands can rollback and commit data modification in the database. The most used TCL commands are COMMIT, ROLLBACK, SAVEPOINT, and SET TRANSACTION.

COMMIT

COMMIT command is used to save or apply the modification in the database.

ROLLBACK

ROLLBACK command is used to undo the modification.

SAVEPOINT

SAVEPOINT command is used to temporarily save a transaction, the transaction can roll back to this point when it's needed.

Syntax :

Just write COMMIT or ROLLBACK or SAVEPOINT

Experiment 1: Create a table called Employee & execute the following.
Employee(EMPNO,ENAME,JOB, MANAGER_NO, SAL, COMMISSION)

1. Create a user and grant all permissions to the user.
2. Insert the any three records in the employee table contains attributes EMPNO, ENAME JOB, MANAGER_NO, SAL, COMMISSION and use rollback. Check the result.
3. Add primary key constraint and not null constraint to the employee table.
4. Insert null values to the employee table and verify the result.

1. Create a user EMPUSER1 and grant permissions

```
CREATE USER empuser1 IDENTIFIED BY emp123;
```

User EMPUSER1 created.

Grant required privileges

```
GRANT CONNECT, RESOURCE TO empuser1;
```

Grant succeeded

```
GRANT CREATE TABLE, CREATE VIEW, CREATE SEQUENCE TO empuser1;
```

Grant succeeded

2. Create EMPLOYEE table

```
CREATE TABLE Employee
(
   EMPNO       NUMBER,
   ENAME       VARCHAR2(20),
   JOB         VARCHAR2(20),
   MANAGER_NO  NUMBER,
   SAL         NUMBER(8,2),
   COMMISSION  NUMBER(8,2)
);
```

Table EMPLOYEE created.

Insert three records and use ROLLBACK

```
INSERT INTO Employee VALUES (101, 'RAMESH', 'MANAGER', 100, 50000, 5000);
INSERT INTO Employee VALUES (102, 'SURESH', 'CLERK', 101, 20000, 2000);
INSERT INTO Employee VALUES (103, 'MAHESH', 'ANALYST', 101, 30000, 3000);
```

View inserted data

```
SELECT * FROM Employee;
```

	EMPNO	ENAME	JOB	MANAGER_NO	SAL	COMMISSION
1	101	RAMESH	MANAGER	100	50000	5000
2	102	SURESH	CLERK	101	20000	2000
3	103	MAHESH	ANALYST	101	30000	3000

Perform Rollback

```
ROLLBACK;
```

Verify rollback

EMPNO	ENAME	JOB	MANAGER...	SAL	COMMISS...

3. Add PRIMARY KEY and NOT NULL constraints

Add PRIMARY KEY constraint

ALTER TABLE Employee
ADD CONSTRAINT emp_pk PRIMARY KEY (EMPNO);
Table EMPLOYEE altered.

Add NOT NULL constraint

ALTER TABLE Employee
MODIFY ENAME NOT NULL;
Table EMPLOYEE altered.

4. Insert NULL values and verify result

INSERT INTO Employee VALUES (104, NULL, 'CLERK', 101, 18000, 1000);

```
Error starting at line : 25 in command -
INSERT INTO Employee VALUES (104, NULL, 'CLERK', 101, 18000, 1000)
Error report -
ORA-01400: cannot insert NULL into ("SYSTEM"."EMPLOYEE"."ENAME")
```

desc Employee;

```
Error report -
ORA-01400: cannot insert NULL into ("SYSTEM"."EMPLOYEE"."ENAME")

Name        Null?    Type
----------- -------- ------------
EMPNO       NOT NULL NUMBER
ENAME       NOT NULL VARCHAR2(20)
JOB                  VARCHAR2(20)
MANAGER_NO           NUMBER
SAL                  NUMBER(8,2)
COMMISSION           NUMBER(8,2)
```

EXPERIMENT 2:

Create a table called Employee that contain attributes EMPNO,ENAME,JOB, MGR,SAL & execute the following.

1. Add a column commission with domain to the Employeetable.
2. Insert any five records into the table.
3. Update the column details of job
4. Rename the column of Employ table using alter command.
5. Delete the employee whose Empno is 105.

Create the table EMPLOYEE1

```
CREATE TABLE Employee1 (
   EMPNO NUMBER,
   ENAME VARCHAR2(20),
   JOB  VARCHAR2(20),
   MGR  NUMBER,
   SAL   NUMBER(8,2)
);
```

Table EMPLOYEE1 created.

1. Add a column COMMISSION with domain

```
ALTER TABLE Employee1
ADD COMMISSION NUMBER(8,2);
```

2. Insert any five records

```
INSERT INTO Employee1 VALUES (101, 'RAMESH', 'MANAGER', 100, 50000, 5000);
INSERT INTO Employee1 VALUES (102, 'SURESH', 'CLERK', 101, 20000, 2000);
INSERT INTO Employee1 VALUES (103, 'MAHESH', 'ANALYST', 101, 30000, 3000);
INSERT INTO Employee1 VALUES (104, 'RAJU', 'SALESMAN', 101, 25000, 2500);
INSERT INTO Employee1 VALUES (105, 'ANITA', 'CLERK', 102, 18000, 1500);
```

Verify insertion

```
SELECT * FROM Employee1;
```

EMPNO	ENAME	JOB	MGR	SAL	COMMISSION
101	RAMESH	MANAGER	100	50000	5000
102	SURESH	CLERK	101	20000	2000
103	MAHESH	ANALYST	101	30000	3000
104	RAJU	SALESMAN	101	25000	2500
105	ANITA	CLERK	102	18000	1500

3. Update the JOB column details

```
UPDATE Employee1
SET JOB = 'SENIOR CLERK'
WHERE EMPNO = 102;
```

1 row updated.

Verify update

SELECT EMPNO, ENAME, JOB FROM Employee1;

EMPNO	ENAME	JOB
101	RAMESH	MANAGER
102	SURESH	SENIOR CLERK
103	MAHESH	ANALYST
104	RAJU	SALESMAN
105	ANITA	CLERK

4. *Rename a column using ALTER command*

ALTER TABLE Employee1
RENAME COLUMN MGR TO MANAGER_NO;
Table EMPLOYEE1 altered.

Verify structure

DESC Employee1;

```
Table EMPLOYEE1 altered.

Name        Null? Type
----------- ----- ------------
EMPNO             NUMBER
ENAME             VARCHAR2(20)
JOB               VARCHAR2(20)
MANAGER_NO        NUMBER
SAL               NUMBER(8,2)
COMMISSION        NUMBER(8,2)
```

5. *Delete employee whose EMPNO is 105*

DELETE FROM Employee1
WHERE EMPNO = 105;
1 row deleted.

Verify deletion

SELECT * FROM Employee1;

EMPNO	ENAME	JOB	MANAGER_NO	SAL	COMMISSION
101	RAMESH	MANAGER	100	50000	5000
102	SURESH	SENIOR CLERK	101	20000	2000
103	MAHESH	ANALYST	101	30000	3000
104	RAJU	SALESMAN	101	25000	2500

EXPERIMENT 3:

Queries using aggregate functions(COUNT,AVG,MIN,MAX,SUM),Group by,Orderby. Employee(E_id, E_name, Age, Salary)

1. Create Employee table containing all Records E_id, E_name, Age, Salary.
2. Count number of employee names from employeetable
3. Find the Maximum age from employee table.
4. Find the Minimum age from employee table.
5. Find salaries of employee in Ascending Order.
6. Find grouped salaries of employees.

1. Create EMPLOYEE table

```
CREATE TABLE Employee2 (
   E_id  NUMBER,
   E_name VARCHAR2(30),
   Age   NUMBER,
   Salary NUMBER(10,2)
);
```

*Describe the structure of the **EMPLOYEE2** table*

```
Name   Null? Type
------ ----- ------------
E_ID         NUMBER
E_NAME       VARCHAR2(30)
AGE          NUMBER
SALARY       NUMBER(10,2)
```

Insert records into EMPLOYEE2

```
INSERT INTO Employee2 VALUES (1, 'RAMESH', 35, 50000);
INSERT INTO Employee2 VALUES (2, 'SURESH', 28, 30000);
INSERT INTO Employee2 VALUES (3, 'MAHESH', 40, 60000);
INSERT INTO Employee2 VALUES (4, 'RAJU', 25, 25000);
INSERT INTO Employee2 VALUES (5, 'ANITA', 30, 45000);
row inserted.
```

2. Count number of employee names

```
SELECT COUNT(E_name) AS Total_Employees
FROM Employee2;
```

	TOTAL_EMPLOYEES
1	5

3. Find the maximum age

```
SELECT MAX(Age) AS Maximum_Age
FROM Employee2;
```

	MAXIMUM_AGE
1	40

4. Find the minimum age

	MINIMUM_AGE
1	25

5. Find salaries of employees in ascending order

```
SELECT E_name, Salary
FROM Employee2
ORDER BY Salary ASC;
```

E_NAME	SALARY
RAJU	25000
SURESH	30000
ANITA	45000
RAMESH	50000
MAHESH	60000

6. Find grouped salaries of employees

SALARY	EMPLOYEE_COUNT
50000	1
30000	1
60000	1
25000	1
45000	1

EXPERIMENT 4:

Create a row level trigger for the customers table that would fire for INSERT or UPDATE or DELETE operations performed on the CUSTOMERS table. This trigger will display the salary difference between the old & new Salary.

CUSTOMERS (ID, NAME, AGE, ADDRESS, SALARY)

1. Create the CUSTOMERS table

```
CREATE TABLE CUSTOMERS (
   ID NUMBER(5),
   NAME VARCHAR2(20),
   AGE NUMBER(3),
   ADDRESS VARCHAR2(30),
   SALARY NUMBER(8,2)
);
```

2. Insert sample records

```
INSERT INTO CUSTOMERS VALUES (1, 'Ramesh', 32, 'Ahmedabad', 20000);
INSERT INTO CUSTOMERS VALUES (2, 'Khilan', 25, 'Delhi', 15000);
INSERT INTO CUSTOMERS VALUES (3, 'Kaushik', 23, 'Kota', 18000);

COMMIT;
```

3. Create ROW LEVEL Trigger

```
CREATE OR REPLACE TRIGGER trg_salary_diff
BEFORE INSERT OR UPDATE OR DELETE
ON CUSTOMERS
FOR EACH ROW
BEGIN
   IF INSERTING THEN
      DBMS_OUTPUT.PUT_LINE('Old Salary: NULL');
      DBMS_OUTPUT.PUT_LINE('New Salary: ' || :NEW.SALARY);
      DBMS_OUTPUT.PUT_LINE('Salary Difference: ' || :NEW.SALARY);

   ELSIF UPDATING THEN
      DBMS_OUTPUT.PUT_LINE('Old Salary: ' || :OLD.SALARY);
      DBMS_OUTPUT.PUT_LINE('New Salary: ' || :NEW.SALARY);
      DBMS_OUTPUT.PUT_LINE('Salary Difference: ' ||
                  (:NEW.SALARY - :OLD.SALARY));

   ELSIF DELETING THEN
      DBMS_OUTPUT.PUT_LINE('Old Salary: ' || :OLD.SALARY);
      DBMS_OUTPUT.PUT_LINE('New Salary: NULL');
      DBMS_OUTPUT.PUT_LINE('Salary Difference: ' ||
                  (- :OLD.SALARY));
   END IF;
END;
```

4. Enable output

```
SET SERVEROUTPUT ON;
```

5. Test INSERT operation

```
INSERT INTO CUSTOMERS VALUES (4, 'Chaitali', 27, 'Mumbai', 22000);
```

Output

Old Salary: NULL
New Salary: 22000
Salary Difference: 22000

6. Test UPDATE operation

```
UPDATE CUSTOMERS
SET SALARY = 25000
WHERE ID = 1;
```

Output
Old Salary: 20000
New Salary: 25000
Salary Difference: 5000

7. Test DELETE operation

```
DELETE FROM CUSTOMERS
WHERE ID = 2;
```

Output
Old Salary: 15000
New Salary: NULL
Salary Difference: -15000

EXPERIMENT 5:

Create cursor for Employee table & extract the values from the table. Declare the variables, Open the cursor & extract the values from the cursor. Close the cursor.

Create table Employee

```
create table employee (E_id number,
E_name varchar(10), age number,
sal number);

Table created.
```

Insert values into the table

Insert any five records into Employee table

```
insert into Employee values(10,'abhi',25 ,10000);
insert into employee values(20,'rohith',30,9000);
insert into employee values(30,'david',28,9000);
insert into employee values(40,'rahul',29,7000);
insert into employee values(50,'pramod',31,8000);

SQL>select * from employee;
```

E_id	E_name	Age	Sal
10	Abhi	25	10000
20	Rohith	30	9000
30	David	28	9000
40	Rahul	29	7000
50	Pramod	31	8000

Create the cursor ,extracting the value from Employee table and close the cursor.

SQL> set serveroutput on;

SQL> declare cursor c1 is select id, sal from Cust;

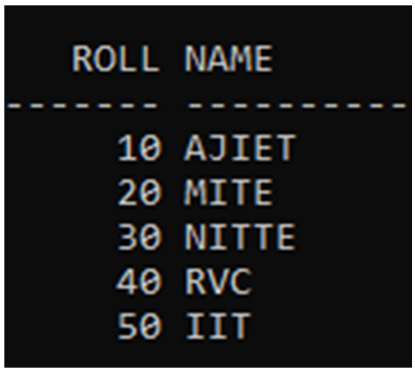
```
vid int;
vsal int;
begin
dbms_output.put_line('Emp ID' ||'                                    '||'Emp sal');
dbms_output.put_line('-----------------------------------------------');
open c1;
loop
fetch c1 into vid,vsal;
exit when c1%notfound;
dbms_output.put_line(vid || '                                    ' || vsal);
end loop;
close  c1;
end;
/
```

E_id	E_name	Age	Sal
10	Abhi	25	10000
20	Rohith	30	9000
30	David	28	9000
40	Rahul	29	7000
50	Pramod	31	8000

EXPERIMENT 6:

Write a PL/SQL block of code using parameterized Cursor, that will merge the data available in the newly created table N_RollCall with the data available in the table O_RollCall. If the data in the first table already exist in the second table then that data should be skipped.

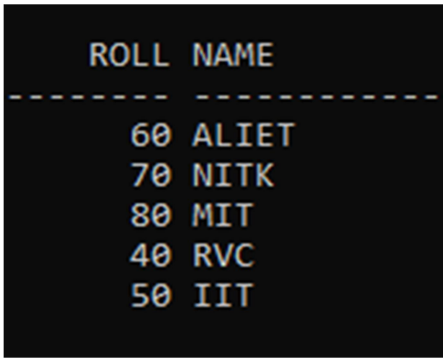
```
//Create table O-Rollcall:

CREATE TABLE O_Rollcall
(
  roll INT PRIMARY KEY,
  name VARCHAR(20)
);

//insert values into the table
INSERT INTO N_Rollcall VALUES (60, 'ALIET');
INSERT INTO N_Rollcall VALUES (70, 'NITK');
INSERT INTO N_Rollcall VALUES (80, 'MIT');
INSERT INTO N_Rollcall VALUES (40, 'RVC');
INSERT INTO N_Rollcall VALUES (50, 'IIT');

SQL>select * from O_Rollcall;
```



ROLL	NAME
10	AJIET
20	MITE
30	NITTE
40	RVC
50	IIT

//Create table N_Rollcall:

```
CREATE TABLE N_Rollcall
(
  roll INT,
  name VARCHAR(20)
);
SQL>select * from N_Rollcall;
```



ROLL	NAME
60	ALIET
70	NITK
80	MIT
40	RVC
50	IIT

```
CREATE OR REPLACE PROCEDURE roll_details AS
```

```
 v_roll O_Rollcall.roll%TYPE;
 v_name O_Rollcall.name%TYPE;
 cnt NUMBER;

 CURSOR c1 IS
   SELECT roll, name FROM N_Rollcall;

BEGIN
 OPEN c1;

 LOOP
   FETCH c1 INTO v_roll, v_name;
   EXIT WHEN c1%NOTFOUND;

   SELECT COUNT(*)
   INTO cnt
   FROM O_Rollcall
   WHERE roll = v_roll;

   IF cnt = 0 THEN
     INSERT INTO O_Rollcall VALUES (v_roll, v_name);
   END IF;

 END LOOP;

 CLOSE c1;
END;
/
Procedure created.
SQL> CALL roll_details();
Call completed.
SQL> SELECT * FROM O_Rollcall;
```

```
    ROLL NAME
-------- ----------
      60 ALIET
      70 NITK
      80 MIT
      40 RVC
      50 IIT
      10 AJIET
      20 MITE
      30 NITTE
```

EXPERIMENT 7:

Install an Open Source NoSQL Data base MangoDB & perform basic CRUD(Create, Read,Update & Delete) operations. Execute MangoDB basic Queries using CRUD operations.

How to Install and Configure MongoDB in Ubuntu?

MongoDB is a popular NoSQL database offering flexibility, scalability, and ease of use. Installing and configuring MongoDB in Ubuntu is a straightforward process, but it requires careful attention to detail to ensure a smooth setup.

In this guide, we'll learn how to install and configure MongoDB in Ubuntu. We'll walk you through each step, from installation to configuration, enabling you to harness the power of MongoDB on your Ubuntu system.

Let's look at the requirements for installing MongoDB in Ubuntu.

Steps to Install and Configure MongoDB in Ubuntu

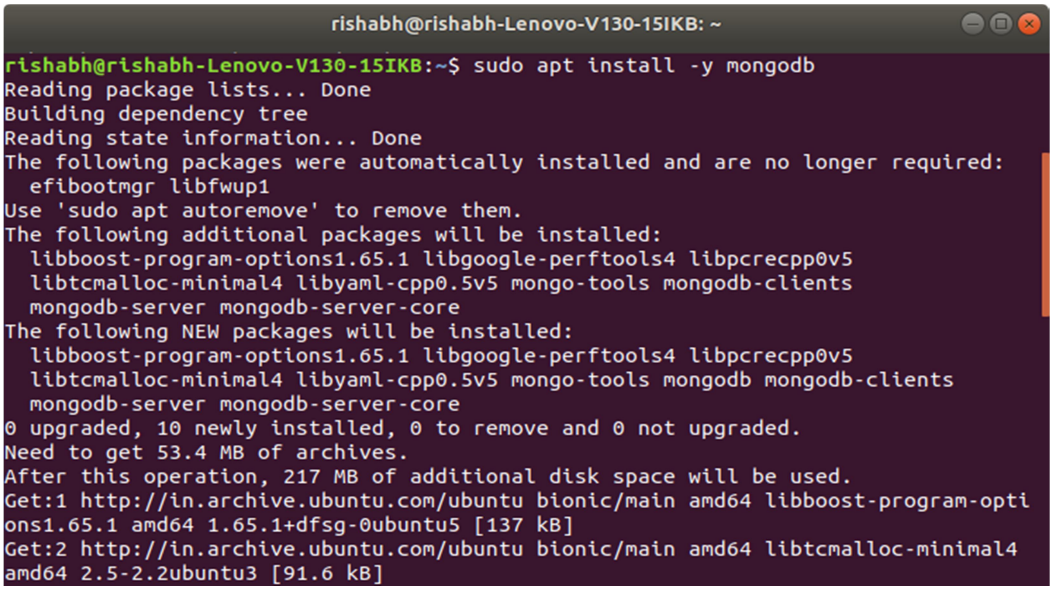
MongoDB can be installed on Ubuntu with the use of the following commands. These commands are easy to run on the terminal and make the installation process handy. Follow the steps given below to install MongoDB:

Step 1: First you need to update and upgrade your system repository to install MongoDB. Type the following command in your terminal and then press Enter.

```
sudo apt update && sudo apt upgrade
```

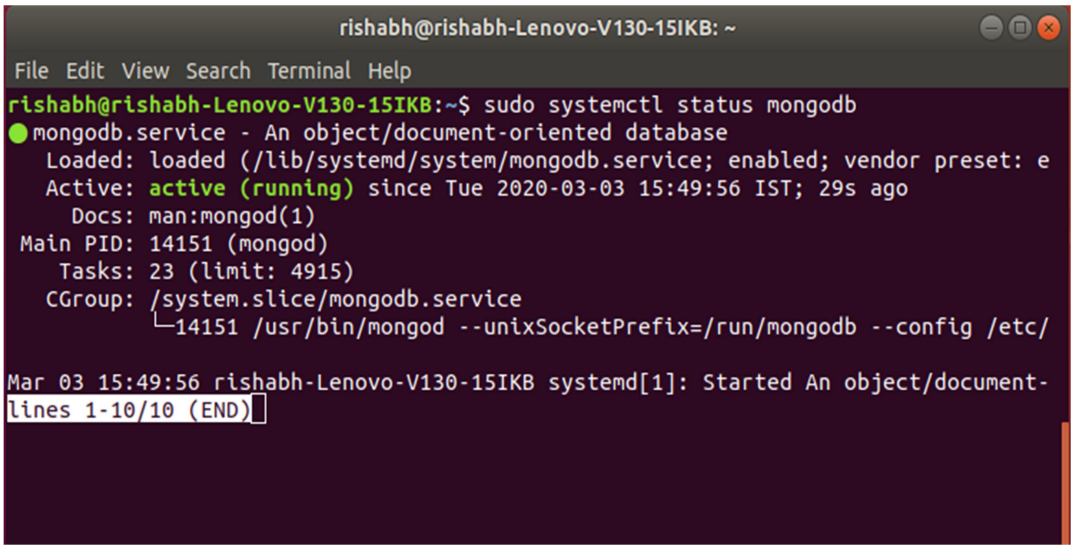
Step 2: Now, install the MongoDB package using 'apt'. Type the following command and press Enter.

```
sudo apt install -y mongodb
```



Step 3: Check the service status for MongoDB with the help of following command:

```
sudo systemctl status mongodb
```



systemctl verifies that MongoDB server is up and running.

Step 4: Now check if the installation process is done correctly and everything is working fine. Go through the following command:

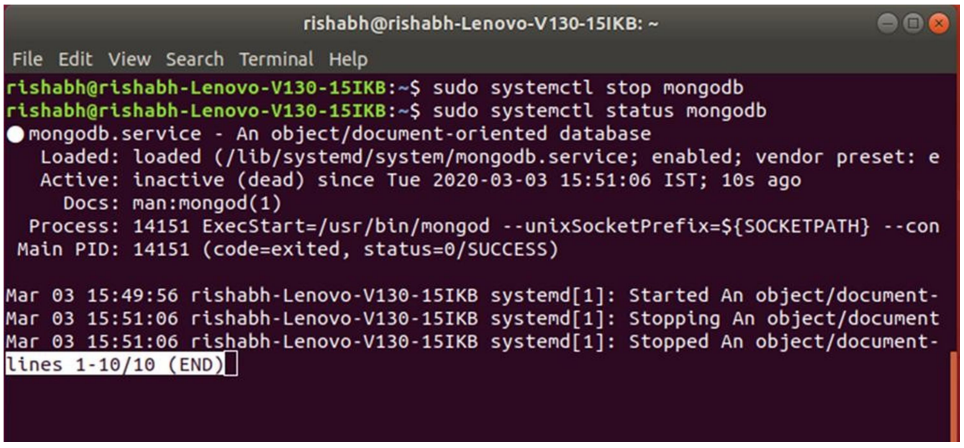
```
mongo --eval 'db.runCommand({ connectionStatus: 1 })'
```

the value “1” in ok field indicates that the server is working properly with no errors.

Step 5: MongoDB services can be started and stopped with the use of following commands: To stop running the MongoDB service, use command :

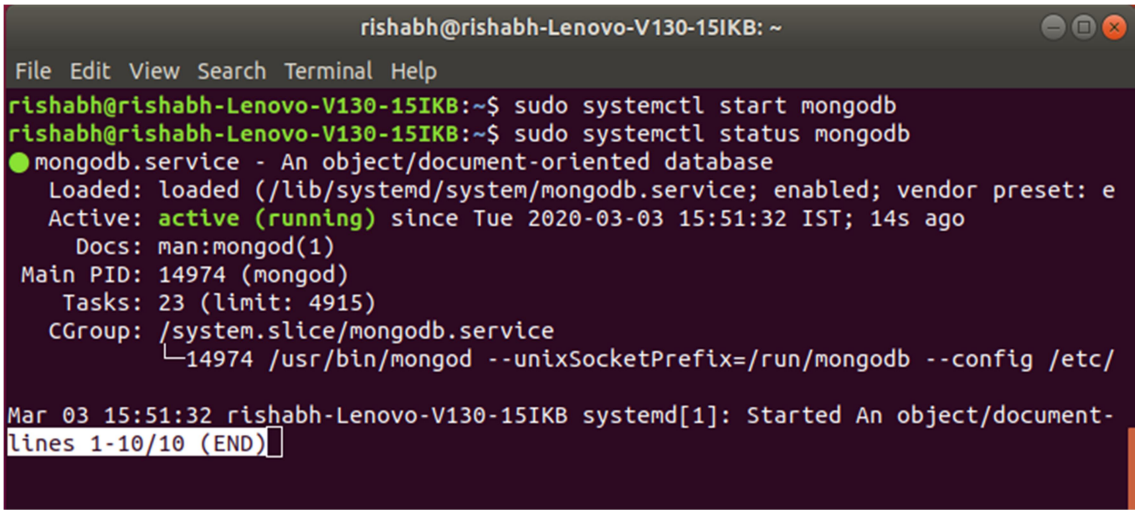
```
sudo systemctl stop mongodb
```

MongoDB service has been stopped and can be checked by using the status command:



As it can be seen that the service has stopped, to start the service we can use :

sudo systemctl start mongodb



Step 6: Accessing the MongoDB Shell

MongoDB provides a command-line interface called the MongoDB shell, which allows you to interact with the database.

To access the MongoDB shell, simply type the following command in your terminal:

mongo

You are now connected to the MongoDB server, and you can start executing commands to createdatabases, collections, and documents.

CRUD Operations:

1. Create (Insert)

To create or insert data into a MongoDB collection, you use the insertOne() or insertMany() methods.

Insert a single document:

```
db.collection('yourCollection').insertOne({ key: value });
```

Insert multiple documents:

```
db.collection('yourCollection').insertMany([
{ key1: value1 },
{ key2: value2 },
]);
```

2. Read (Query)

To read or retrieve data from a MongoDB collection, you use the find() method.

Find all documents:

```
db.collection('yourCollection').find();
```

Find documents with a specific condition:

```
db.collection('yourCollection').find({ key: value });
```

3. Update

To update existing documents in a MongoDB collection, you use the updateOne()or updateMany()

methods.

Update a single document:

db.collection('yourCollection').updateOne(

{ key: value }, // filter

{ $set: { newField: newValue } } // update operation

);

Update multiple documents:

db.collection('yourCollection').updateMany(

{ key: value }, // filter

{ $set: { newField: newValue } } // update operation

);

4. Delete

To delete documents from a MongoDB collection, you use the deleteOne()or deleteMany()methods.

Delete a single document: db.collection('yourCollection').deleteOne({ key: value }); Delete multiple

documents: db.collection('yourCollection').deleteMany({ key: value });