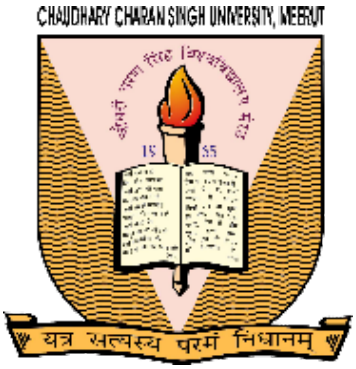
SIR CHHOTU RAM INSTITUTE OF ENGINEERING & TECHNOLOGY



Practical File

on

Design and Analysis of Algorithm

Branch – Computer Science / 5th Semester

Session: 2024 - 2025

Submitted By:

Student's Name

Roll no. 100200300

Submitted To:

Teacher's Name

Table of Contents

S. No.	Program
1	Write a Program to Perform Quick Sort
2	Write a Program to perform Merge Sort
3	Write a Program to perform Heap Sort
4	Write a Program to find Minimum Spanning Tree using Kruskal's Algorithm
5	Write a Program to implement N Queen Problem using Backtracking
6	Write a Program to Perform Travelling Salesman Problem
7	Write a Program to perform Knapsack Problem using Greedy Solution
8	Write a Program to perform Knapsack Problem using Dynamic Solution
9	Write a Program to find Minimum Spanning Tree using Prim's Algorithm
10	Write a Program to find Shortest Path to other vertices using Dijkstra Algorithm

1. WAP to perform the Quick Sort

Program Source Code:

```
def partition(array, low, high):
    pivot = array[high]
    i = low - 1
    for j in range(low, high):
        if array[j] <= pivot:
            i += 1
            array[i], array[j] = array[j], array[i]
    array[i + 1], array[high] = array[high], array[i + 1]
    return i + 1

def quick_sort(array, low, high):
    if low < high:
        pi = partition(array, low, high)
        quick_sort(array, low, pi - 1)
        quick_sort(array, pi + 1, high)

array = list(map(int, input("Enter the elements of the array
separated by spaces: ").split()))
quick_sort(array, 0, len(array) - 1)
print("Sorted array:", array)
```

Output:

Enter the elements of the array separated by spaces: 23 45 67 32 1 4 5
Sorted array: [1, 4, 5, 23, 32, 45, 67]

2. WAP to perform merge sort

Program Source Code:

```
def merge_sort(array):
    if len(array) > 1:
        mid = len(array) // 2
        L = array[:mid]
        R = array[mid:]

        merge_sort(L)
        merge_sort(R)

        i = j = k = 0

        while i < len(L) and j < len(R):
            if L[i] < R[j]:
                array[k] = L[i]
                i += 1
            else:
                array[k] = R[j]
                j += 1
            k += 1

        while i < len(L):
            array[k] = L[i]
            i += 1
            k += 1

        while j < len(R):
            array[k] = R[j]
            j += 1
            k += 1

array = list(map(int, input("Enter the elements of the array
separated by spaces: ").split()))
merge_sort(array)
print("Sorted array:", array)
```

Output:

Enter the elements of the array separated by spaces: 12 45 78 90 09 78 8 2
Sorted array: [2, 8, 9, 12, 45, 78, 78, 90]

3. WAP to perform Heap Sort.

Program Source Code:

```
def heapify(arr, n, i):
    largest = i
    l = 2 * i + 1
    r = 2 * i + 2

    if l < n and arr[l] > arr[largest]:
        largest = l

    if r < n and arr[r] > arr[largest]:
        largest = r

    if largest != i:
        arr[i], arr[largest] = arr[largest], arr[i]
        heapify(arr, n, largest)

def heap_sort(arr):
    n = len(arr)

    for i in range(n // 2 - 1, -1, -1):
        heapify(arr, n, i)

    for i in range(n - 1, 0, -1):
        arr[i], arr[0] = arr[0], arr[i]
        heapify(arr, i, 0)

array = list(map(int, input("Enter the elements of the array
separated by spaces: ").split()))
heap_sort(array)
print("Sorted array:", array)
```

Output:

Enter the elements of the array separated by spaces: 23 89 78 56 4 12 32 4
Sorted array: [4, 4, 12, 23, 32, 56, 78, 89]

4. WAP to find Minimum Spanning Tree using Kruskal's Algorithm

Program Source Code:

```
class Graph:
    def __init__(self, vertices):
        self.V = vertices
        self.graph = []

    def add_edge(self, u, v, w):
        self.graph.append([u, v, w])

    def find(self, parent, i):
        if parent[i] == i:
            return i
        return self.find(parent, parent[i])

    def union(self, parent, rank, x, y):
        if rank[x] < rank[y]:
            parent[x] = y
        elif rank[x] > rank[y]:
            parent[y] = x
        else:
            parent[y] = x
            rank[x] += 1

    def kruskal_mst(self):
        result = []
        i = e = 0
        self.graph = sorted(self.graph, key=lambda item: item[2])
        parent = []
        rank = []

        for node in range(self.V):
            parent.append(node)
            rank.append(0)

        while e < self.V - 1:
            u, v, w = self.graph[i]
            i += 1
            x = self.find(parent, u)
            y = self.find(parent, v)

            if x != y:
                e += 1
                result.append([u, v, w])
                self.union(parent, rank, x, y)

        print("Edges in the MST:")
        for u, v, w in result:
            print(f"{u} -- {v} == {w}")
```

```
vertices = int(input("Enter the number of vertices: "))
edges = int(input("Enter the number of edges: "))
graph = Graph(vertices)

print("Enter each edge as 'u v w' where u and v are vertices and w
is the weight:")
for _ in range(edges):
    u, v, w = map(int, input().split())
    graph.add_edge(u, v, w)

graph.kruskal_mst()
```

Output:

Enter the number of vertices: 4
Enter the number of edges: 5
Enter each edge as 'u v w' where u and v are vertices and w is the weight:
0 1 10
0 2 6
0 3 5
1 3 15
2 3 4
Edges in the MST:
2 -- 3 == 4
0 -- 3 == 5
0 -- 1 == 10

5. WAP to implement N Queen Problem using Backtracking

Program Source Code:

```
def is_safe(board, row, col, n):
    for i in range(col):
        if board[row][i] == 1:
            return False
    for i, j in zip(range(row, -1, -1), range(col, -1, -1)):
        if board[i][j] == 1:
            return False
    for i, j in zip(range(row, n, 1), range(col, -1, -1)):
        if board[i][j] == 1:
            return False
    return True

def solve_n_queens(board, col, n):
    if col >= n:
        for row in board:
            print(" ".join(str(x) for x in row))
        print()
        return True

    res = False
    for i in range(n):
        if is_safe(board, i, col, n):
            board[i][col] = 1
            res = solve_n_queens(board, col + 1, n) or res
            board[i][col] = 0
    return res

def n_queens(n):
    board = [[0 for _ in range(n)] for _ in range(n)]
    if not solve_n_queens(board, 0, n):
        print("No solution exists")
    else:
        print("Solutions printed above")

n = int(input("Enter the number of queens: "))
n_queens(n)
```

Output:

```
Enter the number of queens: 4
0 0 1 0
1 0 0 0
0 0 0 1
0 1 0 0

0 1 0 0
0 0 0 1
1 0 0 0
0 0 1 0

Solutions printed above
```

---

6. WAP t to Perform Travelling Salesman Problem

Program Source Code:

```
from itertools import permutations

def travelling_salesman_problem(graph, s):
    vertices = [i for i in range(len(graph)) if i != s]
    min_path = float('inf')
    for perm in permutations(vertices):
        current_path_weight = 0
        k = s
        for j in perm:
            current_path_weight += graph[k][j]
            k = j
        current_path_weight += graph[k][s]
        min_path = min(min_path, current_path_weight)
    return min_path

graph = []
n = int(input("Enter the number of vertices: "))
print("Enter the adjacency matrix (row by row, space-separated):")
for _ in range(n):
    graph.append(list(map(int, input().split())))

s = int(input("Enter the starting vertex: "))
print("The minimum path weight is:",
travelling_salesman_problem(graph, s))
```

Output:

Enter the number of vertices: 4
Enter the adjacency matrix (row by row, space-separated):
0 10 15 20
10 0 35 25
15 35 0 30
20 25 30 0
Enter the starting vertex: 0
The minimum path weight is: 80

7. WAP to perform Knapsack Problem using Greedy Solution

Program Source Code:

```
def knapsack_greedy(values, weights, capacity):
    items = list(range(len(values)))
    ratio = [v / w for v, w in zip(values, weights)]
    items.sort(key=lambda i: ratio[i], reverse=True)

    total_value = 0
    for i in items:
        if weights[i] <= capacity:
            capacity -= weights[i]
            total_value += values[i]
        else:
            total_value += values[i] * (capacity / weights[i])
            break

    return total_value

values = list(map(int, input("Enter the values of the items:
").split()))
weights = list(map(int, input("Enter the weights of the items:
").split()))
capacity = int(input("Enter the capacity of the knapsack: "))

print("Maximum value in the knapsack:", knapsack_greedy(values,
weights, capacity))
```

Output:

Enter the values of the items: 60 100 120
Enter the weights of the items: 10 20 30
Enter the capacity of the knapsack: 50
Maximum value in the knapsack: 240.0

8. WAP to perform Knapsack Problem using Dynamic Solution.

Program Source Code:

```
def knapsack_dynamic(values, weights, capacity):
    n = len(values)
    dp = [[0] * (capacity + 1) for _ in range(n + 1)]

    for i in range(1, n + 1):
        for w in range(1, capacity + 1):
            if weights[i - 1] <= w:
                dp[i][w] = max(values[i - 1] + dp[i - 1][w -
weights[i - 1]], dp[i - 1][w])
            else:
                dp[i][w] = dp[i - 1][w]

    return dp[n][capacity]

values = list(map(int, input("Enter the values of the items:
").split()))

weights = list(map(int, input("Enter the weights of the items:
").split()))

capacity = int(input("Enter the capacity of the knapsack: "))

print("Maximum value in the knapsack:", knapsack_dynamic(values,
weights, capacity))
```

Output:

Enter the values of the items: 60 100 120

Enter the weights of the items: 10 20 30

Enter the capacity of the knapsack: 50

Maximum value in the knapsack: 220

9. WAP to find Minimum Spanning Tree using Prim's Algorithm

Program Source Code:

```
import sys

def prims_algorithm(graph):
    n = len(graph)
    key = [sys.maxsize] * n
    parent = [-1] * n
    key[0] = 0
    mst_set = [False] * n

    for _ in range(n):
        u = min((key[i], i) for i in range(n) if not mst_set[i])[1]
        mst_set[u] = True

        for v in range(n):
            if graph[u][v] and not mst_set[v] and graph[u][v] <
key[v]:
                key[v] = graph[u][v]
                parent[v] = u

    print("Edge \tWeight")
    for i in range(1, n):
        print(f"{parent[i]} - {i} \t{graph[i][parent[i]]}")

graph = []
n = int(input("Enter the number of vertices: "))
print("Enter the adjacency matrix (row by row, space-separated):")
for _ in range(n):
    graph.append(list(map(int, input().split())))

prims_algorithm(graph)
```

Output:

Enter the number of vertices: 5

Enter the adjacency matrix (row by row, space-separated):

0 2 0 6 0

2 0 3 8 5

0 3 0 0 7

6 8 0 0 9

0 5 7 9 0

Edge Weight

0 - 1 2

1 - 2 3

0 - 3 6

1 - 4 5

10.WAP to find Shortest Path to other vertices using Dijkstra Algorithm

Program Source Code:

```
import sys

def dijkstra(graph, src):
    n = len(graph)
    dist = [sys.maxsize] * n
    dist[src] = 0
    spt_set = [False] * n

    for _ in range(n):
        u = min((dist[i], i) for i in range(n) if not spt_set[i])[1]
        spt_set[u] = True

        for v in range(n):
            if graph[u][v] and not spt_set[v] and dist[u] +
graph[u][v] < dist[v]:
                dist[v] = dist[u] + graph[u][v]

    print("Vertex \tDistance from Source")
    for i in range(n):
        print(f"{i} \t{dist[i]}")

graph = []
n = int(input("Enter the number of vertices: "))
print("Enter the adjacency matrix (row by row, space-separated):")
for _ in range(n):
    graph.append(list(map(int, input().split())))

src = int(input("Enter the source vertex: "))
dijkstra(graph, src)
```

Output:

Enter the number of vertices: 5

Enter the adjacency matrix (row by row, space-separated):

0 10 0 0 5

0 0 1 0 2

0 0 0 4 0

7 0 6 0 0

0 3 9 2 0

Enter the source vertex: 0

Vertex Distance from Source

0 0

1 8

2 9

3 7

4 5
