INDEX

Tittle

PART A

/**********/

Aim : Write a program to sort a list of N elements using selection sort technique.

Name :

Reg.no:U05DP22S0

Date :19/08/24

/**********/

```
array=[]
n=int(input("Enter number of elements:"))
for i in range(0,n):
    ele=int(input("Enter the elements:"))
    array.append(ele)
print("Elements before sorting:",array)
for i in range(n):
    minimum=i
    for j in range(i+1,n):
        if(array[j]<array[minimum]):
            minimum=j
    temp=array[i]
    array[i]=array[minimum]
    array[minimum]=temp
print("Element after sorting:",array)
```

OUTPUT 1:

```
Enter number of elements:4
Enter the elements:2
Enter the elements:1
Enter the elements:7
Enter the elements:4
Elements before sorting: [2, 1, 7, 4]
Element after sorting: [1, 2, 4, 7]
```

OUTPUT 2:

```
Enter number of elements:4
Enter the elements:0
Enter the elements:5
Enter the elements:4
Enter the elements:-5
Elements before sorting: [0, 5, 4, -5]
Element after sorting: [-5, 0, 4, 5]
```

/**********/

Aim : Write a program to read 'n' numbers, find minimum and maximum value in an array using divide and conquer.

Name :

Reg.No:U05DP22S0

Date :19/08/24

/**********/

```
def find_min_max(arr,start,end):
    if start==end:
        return arr[start],arr[start]
    if end-start==1:
        if arr[start]<arr[end]:
            return arr[start],arr[end]
        else:
            return arr[end],arr[start]
    mid=(start+end)//2
    min_left,max_left=find_min_max(arr,start,mid)
    min_right,max_right=find_min_max(arr,mid+1,end)
    min_val=min(min_left,min_right)
    max_val=max(max_left,max_right)
    return min_val,max_val
n=int(input("Enter the number of elements:"))
arr=[]
for i in range(n):
    num=int(input(f"Enter elements{i+1}:"))
    arr.append(num)
min_val,max_val=find_min_max(arr,0,n-1)
print(f"minimum value:{min_val}")
print(f"maximum value:{max_val}")
```

OUTPUT 1:

```
Enter the number of elements:4
Enter elements1:5
Enter elements2:1
Enter elements3:7
Enter elements4:-4
minimum value:-4
maximum value:7
```

/**********/

Aim : 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.

Name :

Reg.No: U05DP22S0

Date :19/08/24

/**********/

```
from random import*
from time import*
def mergeSort(array):
    if len(array)>1:
        m=len(array)//2
        L=array[:m]
        R=array[m:]
        mergeSort(L)
        mergeSort(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
def printList(array):
    for i in range(len(array)):
        print(array[i],end="")
    print()
if __name__=='__main__':
    start_time=process_time()
```

```
n=int(input("Enter how many elements?"))
array=[]
for j in range(n):
    array.append(randint(5000,15000))
print(array)
mergeSort(array)
print("sorted array is:")
printList(array)
print("Time taken to sort is ",(process_time()-start_time)*1000,"ms")
```

OUTPUT 1:

```
Enter how many elements?5
[9785, 7165, 5400, 5554, 12191]
sorted array is:
540055547165978512191
Time taken to sort is  0.0 ms
```

/**********/

Aim : 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.

Name :

Reg.No: U05DP22S0

Date :19/08/24

/**********/

```
from random import*
from time import *
def quick_sort(array):
    if len(array)<=1:
        return array

    pivot=array[randint(0,len(array)-1)]
    less=[x for x in array if x<pivot]
    equal=[x for x in array if x==pivot]
    greater=[x for x in array if x>pivot]

    return quick_sort(less)+equal+quick_sort(greater)

def printList(array):
    for i in range(len(array)):
        print(array[i],end="")
    print()
if __name__=='__main__':
    start_time=process_time()
    n=int(input("Enter how many elemenhts?"))
    array=[]
    for j in range(n):
        array.append(randint(5000,15000))
    print(array)
    sorted_numbers=quick_sort(array)
    print("Sorted array is:")
    printList(sorted_numbers)
    print("Time taken to sort is",(process_time()-start_time)*1000,"ms")
```

OUTPUT 1:

```
Enter how many elemenhts?4
[14969, 9739, 7711, 6503]
Sorted array is:
65037711973914969
Time taken to sort is 0.0 ms
```

/**********/

Aim : Write a program to sort a list of N elements using Insertion Sort Technique.

Name :

Reg.No: U05DP22S0

Date :26/08/24

/**********/

```
array=[]
n=int(input("Enter number of elements:"))
for i in range(0,n):
   ele=int(input(f"Enter element{i+1}:"))
   array.append(ele)
print("Elements before sorting:",array)
for i in range(1,len(array)):
      key=array[i]
      j=i-1
      while j>=0 and key <array[j]:
         array[j+1]= array[j]
         j-=1
      array[j+1]=key
print("Elements after sorting:",array)
```

OUTPUT 1:

```
Enter number of elements:5
Enter element1:8
Enter element2:-3
Enter element3:0
Enter element4:33
Enter element5:-7
Elements before sorting: [8, -3, 0, 33, -7]
Elements after sorting: [-7, -3, 0, 8, 33]
```

/**********/

Aim : Write program to implement the BFS algorithm for a graph.

Name :

Reg.No: U05DP22S0

Date :26/08/24

/**********/

```
from collections import defaultdict,deque
class Graph:
   def __init__(self):
      self.graph=defaultdict(list)
   def add_edge(self,u,v):
      self.graph[u].append(v)
   def bfs(self,start):
      visited=set()
      queue=deque()

      visited.add(start)
      queue.append(start)

      while queue:
            vertex=queue.popleft()
            print(vertex,end=' ')

            for neighbor in self.graph[vertex]:
               if neighbor not in visited:
                  visited.add(neighbor)
                  queue.append(neighbor)

g=Graph()
g.add_edge('A','B')
g.add_edge('A','C')
g.add_edge('B','D')
g.add_edge('B','E')
g.add_edge('C','F')

print("Breadth -First Search starting from vertex 'A':")
g.bfs('A')
```

OUTPUT 1:

```
Breadth -First Search starting from vertex 'A':
A B C D E F
```

/**********/

Aim : Write program to implement the DFS algorithm for a graph.

Name :

Reg.No: U05DP22S0

Date :26/08/24

/**********/

```
class Graph:
    def __init__(self):
        self.graph={}
    def add_edge(self,vertex,neighbor):
        if vertex in self.graph:
            self.graph[vertex].append(neighbor)
        else:
            self.graph[vertex]=[neighbor]
    def dfs(self,start_vertex,visited=set()):
        if start_vertex not in visited:
            print(start_vertex,end=' ')
            visited.add(start_vertex)
            if start_vertex in self.graph:
                for neighbor in self.graph[start_vertex]:
                    self.dfs(neighbor,visited)

g=Graph()

g.add_edge('A','B')
g.add_edge('A','C')
g.add_edge('B','D')
g.add_edge('B','E')
g.add_edge('C','F')

print("Depth-First search starting from vertex 'A':")
g.dfs('A')
```

OUTPUT 1:

```
Depth-First search starting from vertex 'A':
A B D E C F
```

/**********/

Aim : Write a program to implement Strassen's Matrix Multiplication of 2*2 Matrixes.

Name :

Reg.No: U05DP22S0

Date :26/08/24

/**********/

```
def strassen_matrix_multiply(a,b):
    if len(a)!=2 or len(b)!=2 or len(a[0])!=2 or len(b[0])!=2:
        raise ValueError("Input matrices must be 2*2")
    a11,a12,a21,a22=a[0][0],a[0][1],a[1][0],a[1][1]
    b11, b12, b21, b22 = b[0][0], b[0][1], b[1][0], b[1][1]

    m1=(a11+a22)*(b11+b22)
    m2=(a21+a22)*b11
    m3=a11*(b12-b22)
    m4=a22*(b21-b11)
    m5=(a11+a12)*b22
    m6=(a21-a11)*(b11+b12)
    m7=(a12-a22)*(b21+b22)

    c11=m1+m4-m5+m7
    c12=m3+m5
    c21=m2+m4
    c22=m1-m2+m3+m6

    result=[[c11,c12],[c21,c22]]
    return result

matrix1=[[1,2],[3,4]]
matrix2=[[5,6],[7,8]]

result=strassen_matrix_multiply(matrix1,matrix2)
print("The resultant matrix is")
for row in result:
    print(row)
```

OUTPUT 1:

```
The resultant matrix is
[19, 22]
[43, 50]
```

PART B

/**********/

Aim : Write program to implement backtracking algorithm for solving problems like N queens.

Name :

Reg.No: U05DP22S0

Date :03/09/24

/**********/

```
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_nqueens_util(board,col,n):
    if col>=n:
        return True
    for i in range(n):
        if is_safe(board,i,col,n):
            board[i][col]=1
            if solve_nqueens_util(board,col+1,n):
                return True
            board[i][col]=0
    return False
def solve_nqueens(n):
    board=[[0 for _ in range(n)]for _ in range(n)]
    if not solve_nqueens_util(board,0,n):
        print("no solution exists")
        return
    print_solution(board)
def print_solution(board):
    for row in board:
        print("".join(map(str,row)))
if __name__=="__main__":
    n=int(input("Enter the number of queens(N):"))
    solve_nqueens(n)
```

OUTPUT 1:

```
Enter the number of queens(N):5
10000
00010
01000
00001
00100
```

OUTPUT 2;

```
Enter the number of queens(N):3
no solution exists
```

/**********/

Aim : Design and implement in 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. For example, if S={1, 2, 5, 6, 8} and d= 9, there are two solutions {1,2,6}and {1,8}. Display a suitable message, if the given problem instance doesn't have a solution.

Name :

Reg.No: U05DP22S0

Date :03/09/24

/**********/

```
def subset_sum_util(nums,target_sum,current_sum,start,path,result):
    if current_sum==target_sum:
        result.append(path.copy())
        return
    for i in range(start,len(nums)):
        if current_sum+nums[i]<=target_sum:
            path.append(nums[i])
            subset_sum_util(nums,target_sum,current_sum+nums[i],i+1,path,result)
            path.pop()
def find_subset_sum(nums,target_sum):
    result=[]
    subset_sum_util(nums,target_sum,0,0,[],result)
    if not result:
        print("No subset with sum {} exists:".format(target_sum))
    else:

        print("subsets with sum {} found:".format(target_sum))
        for subset in result:
            print(subset)
if __name__=="__main__":
    n=int(input("enter the number of elements:"))
    set_of_numbers=[]
    for i in range(n):
        set_of_numbers.append(int(input(f"Enter the {i+1} element:")))
    target_sum=int(input("Enter the sum:"))
    find_subset_sum(set_of_numbers,target_sum)
```

OUTPUT 1:

```
enter the number of elements:5
Enter the 1 element:2
Enter the 2 element:1
Enter the 3 element:5
Enter the 4 element:3
Enter the 5 element:4
Enter the sum:6
subsets with sum 6 found:
[2, 1, 3]
[2, 4]
[1, 5]
```

OUTPUT 2:

```
enter the number of elements:3
Enter the 1 element:7
Enter the 2 element:3
Enter the 3 element:5
Enter the sum:1
No subset with sum 1 exists:
```

/**/

Aim : Write a program find shortest paths to other vertices using Dijkstra's algorithm.

Name :

Reg.No: U05DP22S0

Date :10/09/24

/**/

```
class Graph():
    def __init__(self,vertices):
        self.V=vertices
        self.graph=[[0 for column in range(vertices)]for row in range(vertices)]

    def printSolution(self,dist):
        print("vertex \t distance from source")
        for node in range(self.V):
            print(node,"\t \t",dist[node])
    def minDistance(self,dist,sptSet):
        min=1e7
        for v in range(self.V):
            if dist[v]< min and sptSet[v]==False:
                min=dist[v]
                min_index=v
        return min_index
    def dijkstra(self,src):
        dist=[1e7]*self.V
        dist[src]=0
        sptSet=[False]*self.V
        for cout in range(self.V):
            u=self.minDistance(dist,sptSet)
            sptSet[u]=True
            for v in range(self.V):
                if(self.graph[u][v]>0 and
                sptSet[v]==False and
                dist[v]>dist[u]+self.graph[u][v]):
                    dist[v]=dist[u]+self.graph[u][v]
        self.printSolution(dist)
g=Graph(5)
g.graph=[[0,3,0,7,0],
        [3,0,4,2,0],
        [0,4,0,5,6],
        [7,2,5,0,4],
```

```
        [0,0,6,4,0]
    ]
g.dijkstra(0)
```

OUTPUT 1:

```
vertex   distance from source
0        0
1        3
2        7
3        5
4        9
```

/**********/

Aim : Write a program to perform Knapsack Problem using Greedy Solution.

Name :

Reg.No: U05DP22S0

Date :10/09/24

/**********/

```
def knapsack(capacity,weights,values,n):
    if n==0 or capacity==0:
        return 0
    if weights[n-1]>capacity:
        return knapsack(capacity,weights,values,n-1)
    else:
        include_item=values[n-1]+knapsack(capacity-weights[n-1],weights,values,n-1)
        exclude_item=knapsack(capacity,weights,values,n-1)
        return max(include_item,exclude_item)
capacity=10
weights=[7,3,4,5]
values=[42,12,40,25]
n=len(values)
result=knapsack(capacity,weights,values,n)
print(f"The maximum value that can be obtained is {result}")
```

OUTPUT 1:

```
The maximum value that can be obtained is 65
```

/**********/

Aim : Write program to implement greedy algorithm for job sequencing with deadlines.

Name :

Reg.No: U05DP22S0

Date :10/09/24

/**********/

```
def job_sequencing_with_deadline(jobs):
    jobs.sort(key=lambda x:x[2],reverse=True)
    max_deadline=max(job[1]for job in jobs)
    schedule=[-1]*max_deadline
    for job in jobs:
        deadline=job[1]-1
        while deadline>=0 and schedule[deadline]!=-1:
            deadline-=1
        if deadline>=0:
            schedule[deadline]=job[0]
    return[job for job in schedule if job!=-1]
jobs=[(1,4,20),(2,1,10),(3,1,40),(4,1,30)]
result=job_sequencing_with_deadline(jobs)
print("Jobs sequence:",result)
```

OUTPUT 1:

```
Jobs sequence: [3, 1]
```

/**********/

Aim : Write a program to perform Travelling Salesman Problem.

Name :

Reg.No: U05DP22S0

Date :17/09/24

/**********/

```
import itertools
def calculate_total_distance(tour,distance_matrix):
    total_distance=0
    for i in range (len(tour)-1):
        total_distance+=distance_matrix[tour[i]][tour[i+1]]
    total_distance+=distance_matrix[tour[-1]][tour[0]]
    return total_distance
def tsp_bruteforce(distance_matrix):
    num_cities=len(distance_matrix)
    cities=list(range(num_cities))
    shortest_tour=None
    shortest_distance=float('inf')
    for tour in itertools.permutations(cities):
        tour_distance=calculate_total_distance(tour,distance_matrix)
        if tour_distance<shortest_distance:
            shortest_distance=tour_distance
            shortest_tour=tour
    return shortest_tour,shortest_distance
if __name__=="__main__":
    distance_matrix=[
        [0,10,15,20],
        [10,0,35,25],
        [15,35,0,30],
        [15,25,30,0]
    ]
    shortest_tour,shortest_distance=tsp_bruteforce(distance_matrix)
    print("Shortest tour:",shortest_tour)
    print("Shortest distance:",shortest_distance)
```

OUTPUT 1:

```
Shortest tour: (0, 1, 3, 2)
Shortest distance: 80
```

/**/

Aim : Write a program that implements Prim's algorithm to generate minimum cost spanning Tree.

Name :

Reg.No: U05DP22S0

Date :17/09/24

/**/

```
import sys
class Graph():
    def __init__(self,vertices):
        self.V=vertices
        self.graph=[[0 for column in range(vertices)]for row in range(vertices)]
    def printMST(self,parent):
        print("Edge \t Weight")
        for i in range(1,self.V):
            print(parent[i],"-",i,"\t",self.graph[i][parent[i]])
    def minKey(self,key,mstSet):
        min=sys.maxsize
        for v in range(self.V):
            if key[v]<min and mstSet[v]==False:
                min=key[v]
                min_index=v
        return min_index
    def primMST(self):
        key=[sys.maxsize]*self.V
        parent=[None]*self.V
        key[0]=0
        mstSet=[False]*self.V
        parent[0]=-1
        for cout in range(self.V):
            u=self.minKey(key,mstSet)
            mstSet[u]=True
            for v in range(self.V):
                if self.graph[u][v]>0 and mstSet[v]==False and key[v]>self.graph[u][v]:
                    key[v]=self.graph[u][v]
                    parent[v]=u
        self.printMST(parent)
if __name__=="__main__":
  g=Graph(5)
```

```
g.graph=[[0,5,7,0,2],
      [5,0,0,6,3],
      [7,0,0,4,4],
      [0,6,4,0,5],
      [2,3,4,5,0]]
g.primMST()
```

OUTPUT 1:

```
Edge     Weight
4 - 1    3
4 - 2    4
2 - 3    4
0 - 4    2
```

/*/**

Aim : Write a program that implements Kruskal's algorithm to generate minimum cost spanning tree.

Name :

Reg.No: U05DP22S0

Date :24/09/24

/*/**

```
class Graph:
    def __init__(self,vertices):
        self.V=vertices
        self.graph=[]
    def addEdge(self,u,v,w):
        self.graph.append([u,v,w])
    def find(self,parent,i):
        if parent[i]!=i:
            parent[i]=self.find(parent,parent[i])
        return 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 KruskalMST(self):
        result=[]
        i=0
        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=i+1
            x=self.find(parent,u)
            y=self.find(parent,v)
```

```
            if x!=y:
                e=e+1
                result.append([u,v,w])
                self.union(parent,rank,x,y)
        minimumCost=0
        print("Edges in the Constructed MST")
        for u,v,weight in result:
            minimumCost+=weight
            print("%d%d==%d"%(u,v,weight))
        print("minimum spanning Tree",minimumCost)
if __name__ == '__main__':
    g=Graph(4)
    g.addEdge(0,1,10)
    g.addEdge(0,2,6)
    g.addEdge(0,3,5)
    g.addEdge(1,3,15)
    g.addEdge(2,3,4)
    g.KruskalMST()
```

OUTPUT 1:

```
Edges in the Constructed MST
23==4
03==5
01==10
minimum spanning Tree 19
```