INDEX

S. No.	Title of the Experiment	Date Conducted	Date Submitted	Pages From/ to	Grade Marks	Remarks Initial
1	BFS (Breadth first search)	14/8/25		1-2		
2	DFS (Depth first search)	14/8/25		3-4		
3	Linear Search	21/8/25		5		
4	Binary Search	21/8/25		6-7		
5	Merge Sort	28/8/25		8-10		
6	Quick Sort	4/9/25		11-12		
7	MaxMin (Brute force)	11/9/25		13		
8	MaxMin (D and C)	11/9/25		14-15		
9	Knapsack (Brute force)	18/9/25		16-17		
10	Fractional Knapsack (Greedy)	18/9/25		18-19		
11	Prims	25/9/25		20-24		
12	Kruskals (greedy)	25/9/25		23-25		
13	Dijkstras	16/10/25		26		
14						
15						

Faculty In-charge

Algorithm:

```
Algorithm BFS(v)
// A breadth first search of G is carried out beginning at vertex v.
// For any node i, visited[i] = 1 if i has been visited already.
// The graph G and array visited[i] are global; visited[] is
// initialized to zero.
{
   u := v;
   visited[v] := 1;
   repeat
   {
      for all vertices w adjacent from u do
      {
         if (visited[w] = 0) then
         {
            add w to q;
            visited[w] := 1;
         }
      }
      if q is empty then return;
      Delete u from q;
   } until (false);
}
```

EXP. NO. 1)

MVSR Engineering College Page No. : 1 Date :

BFS (Breadth first search)

```
#include<stdio.h>
int a[20][20], q[20], visited[20], n, i, j, f=0, r=-1;
void bfs(int v)
{
    visited[v]=1;
    printf("%d\t", v);
    while(1)
    {
        for(i=1; i<=n; i++)
        {
            if(a[v][i]!=0 && visited[i]==0)
            {
                visited[i]=1;
                q[++r]=i;
            }
        }
        if(f>r)
            break;
        v=q[f++];
        printf("%d\t", v);
    }
}
int main(){
    int v;
    printf("\n Enter the number of vertices\n");
    scanf("%d", &n);
    printf("\n Enter graph data in matrix form:\n");
```

output :

```
Enter number of vertices : 6
Enter graph data in matrix form
0 1 1 0 0 0
1 0 0 1 0 0
1 0 0 0 0 0
0 1 0 0 0 0
0 0 0 0 0 1
0 0 0 0 1 0
Enter the starting vertex : 1
The nodes which are reachable are :
1 2 3 4
```

{ for (i=1; i<=n; i++)

void union (in

MVSR Engineering College

Page No. : 2
Date :

EXP. NO.

```
for (i=1; i<=n; i++) {
  for (j=1; j<=n; j++){
      scanf ("%d", &a[i][j]);
for (i=1; i<=n; i++)
   visited [i] = 0;
printf (" \n enter starting vertex");
scanf ("%d", &v);
printf ("\n the nodes which are reachable are: \n");
   bfs(v);
}
```

EXP. NO.

Algorithm · DFS(v)

// given a ~~undirected~~ unidirectional (directed) graph G = (V, E) with n
// vertices an array visited[] initially set to zero, this algorithm
// visits all vertices reachable from v, G and visited[] are global

```
{
    visited[v] := 1;
    for each vertex w adjacent from v do
    {
        if (visited[w] = 0) then DFS(w);
    }
}
```

2) DFS (Depth first search)

```
#include <stdio.h>
int a[20][20], reach[20], n;
void dfs(int v)
{
    int i; reach[v] = 1;
    for (i=1; i<=n; i++)
      if (a[v][i] && !reach[i])
      {
          printf ("\n %d → %d", v, i);
          dfs(i);
      }
}
void main() {
int i, j, count=0;
printf("\n Enter number of vertices:");
scanf ("%d", &n);
  for (i=1; i<=n; i++)
{
      reach[i] = 0;
    for (j=1; j<=n; j++)
          a[i][j] = 0;
}
printf ("\n Enter the adjacency matrix \n");
for (i=1; i<=n; i++)
        for (j=1; j<=n; j++)
            scanf ("%d", &a[i][j]);
dfs(1);
printf ("\n");
```

output:

Enter number of vertices: 4

Enter adjacency matrix:

0 1 1 0

1 0 0 1

1 0 0 0

0 1 0 0

1 → 2

2 → 4

1 → 3

graph is connected

```
for (i=1; i<=n; i++)
{
    if (reach[i])
        count++;
}
if (count==n)
    printf ("\n Graph is connected\n");
else
    printf ("\n Graph is not connected\n");
}
```

output:

5

10 50 30 70 20

30

element found at index : 30

Linear Search

```
#include <stdio.h>
int linearSearch (int arr[], int n, int key, int index) {
    if (index == n)
        return -1;
    if (arr[index] == key)
        return index;
    return linearSearch (arr, n, key, index+1);
}
int main(){
    int n, key;
    scanf ("%d", &n);
    int arr[n];
    for (int i=0; i<n; i++)
        scanf ("%d", &arr[i]);
    scanf ("%d", &key);
    int result = linearSearch (arr, n, key, 0);
    if (result != -1)
        printf ("element found at index %d\n", result);
    else
        printf ("element not found\n");
    return n;
}
```

```
Algorithm BinSrch(a, low, high, key)
// given an array a[low:high] of elements in nondecreasing order
// determines whether key is present and if so, return position j
// such that key = a[j]; else return 0
{
    if (high = low) then
    {
        if (key = a[low]) then
        {       return low; }
        else
                return 0;
    }

    else{
        mid := (low + high) / 2;
        if (key = a[mid]) then
            return mid;
        else if (key < a[mid]) then
            return BinSrch(a, low, mid-1, key);
        else
            return BinSrch(a, mid+1, high, key);
    }
}
```

EXP. NO.

MVSR Engineering College

Page No. : 6
Date :

4) Binary Search

```
#include <stdio.h>
int binarysearch (int arr[], int low, int high, int x) {
    if (low > high)
        return -1;
    int mid = (low + high)/2;
    if (arr[mid] == x)
        return mid;
    else if (x < arr[mid])
        return binarySearch (arr, low, mid-1, x);
    else
        return binarySearch (arr, mid+1, high, x);
}
int main() {
    int n, x;
    printf(" enter no. of elements : ");
    scanf ("%d", &n);
    int arr [n];
    printf (" enter %d elements (in sorted order): ", n);
    for (int i=0; i<n; i++){
        scanf ("%d", &arr [i]);
    }
    printf(" enter elements to search :");
    scanf (" %d", &x);
    int result = binarySearch (arr, 0, n-1, x);
    if (results == -1)
        printf (" element not found\n");
```

output :

```
Enter the number of elements : 7
enter 7 elements (in sorted order) : 2 4 6 8 10 12 14
enter the elements to search : 10
element found at index 4
```

```
else {
    printf("element found at index %d\n", result);
}

return 0;

}
```

```
Algorithm MergeSort (low, high) {
// a (low:high) is a global array to be sorted  Small(p) is true if
// there is only one element to sort. In this case list is already
// sorted.
{
   if (low < high) then  // if there are more than one element
   {
      // dividing P into subproblems
      mid := (low + high)/2 ;
      MergeSort (low, mid);
      MergeSort (mid+1, high);
      Merge (low, mid, high);
   }
}
}
```

EXP. NO.

MVSR Engineering College

Page No. : 8
Date :

5. Merge Sort

```
#include <stdio.h>
#include <stdlib.h>
void Merge (int a[], int low, int mid, int high)
{
    int i, j, k, temp[high];
    i = low;
    k = low;
    j = mid+1;
    while(i<=mid && j<=high){
        if (a[i] < a[j])
        {
            temp[k] = a[i];
            k++;
            i++;
        }
    else{
        temp[k] = a[j];
        k++;
        j++;
    }
}
while (i<=mid)
{
    temp[k] = a[i];
    k++;
    i++;
}
while (j<=high){
    temp[k] = a[j];
```

```
Algorithm Merge (low, mid, high)
// a[low:high] is a global array containing sorted in a[low:mid] and
// in a [mid+1 : high] - The goal is to merge these two sets into a
// single set residing in a[low:high] is an Auxillary global array
{
    h:=low ; i:=low ; j:=mid+1 ;
    while ( (h≤mid) and (j≤high)) do
    {
        if ( a[h] ≤ a [j] ) then
        {
            b[i] := a[h] ; h = h+1;
        }
        else{
            b[i] := a[j] ; j = j+1;
        }
        i := i+1 ;
    if (h>mid) then {
        for k:=j to high do {
            b[i] := a[k] ; i = i+1;
        }
    else {
        for k:=h to mid do
            { b[i] := a [k] ; i = i+1 ;
            }
        }
}
    for k:=low to high do
    {
        a[k] := b [k] ;
    }
```

```
        k++; j++;
    }
    for (i=low; i<=high; i++)
    {
        a[i] = temp[i];
    }
    void mergesort (int a[], int low, int high)
    {
        int mid;
        if (low < high)
        {
            mid = (low+high)/2;
        mergeSort( a, low, mid);
        mergeSort(a, mid+1, high);
        merge( a, low, mid, high);
        }
    }
    void main()
    {
        int n, *a, k;
        printf ("\n enter how many numbers \n");
        scanf ("%d", &n);
        a = (int *) calloc (n, sizeof (int));
        printf ("\n enter %d elements \n", n);
        for (k=0; k<n; k++) {
                scanf ("%d", &a[k]);
        }
        mergeSort (a, 0, n-1);
```

output:

```
enter how many numbers : 6
enter 6 elements :
-1  -2  0  12  3  2
Sorted numbers are :
-2  -1  0  2  3  12
```

```
printf ("Sorted numbers are \n");
for (k=0 ; k<n; k++)
{
        printf ("%d\t", a[k]);
}

}
```

```
Algorithm Partition (a, m, p)
{
    v := a[m] ; i := m ; j := p;
    repeat
    {
        repeat
            i = i + 1;
        until (a[i] ≥ v);

        repeat
            j := j + 1;
        until (a[j] ≤ v);
        if (i < j) the interchange (a, i, j);
    } until (i ≥ j)

    a[m] := a[j] ; a[j] := v ; return j;
}

Algorithm interchange (a, i, j)
{
    p := a[i];
    a[i] := a[j];
    a[j] := p;
}
```

EXP. NO.

MVSR Engineering College

Page No. : 11
Date :

6. QuickSort

```
#include<stdio.h>
#include<time.h>
#include<stdlib.h>
void QuickSort(int a[3, int low, int high){
    if (low < high){
        int i, j, pivot, k, temp;
        pivot = a[low];
        i = low;
        j = high;
        while (i<j)
        {
            while( a[i] <=pivot && i<=j)
                i=i+1;
            while( a[j] >pivot)
                j=j-1;
            if(i<j){
                temp = a[i];
                a[i] = a[j];
                a[j] = temp;
            }
        }
a[low] = a[j];
a[j] = pivot
Quicksort (a, low, j-1);
Quicksort ( a, j+1 , high);
}
}
```

```
Algorithm Quicksort (p,q)
// sorts the elements a[p] ... a[n] which reside in the global
// array a[1:n] into a ascending order
{
    if (p<q) then
    {
        j := partition(a, p, q+1);
        Quicksort (p, j-1);
        Quicksort (j+1, q);
    }
}
```

Output:

```
Enter number of numbers : 5
The random numbers :
5  4  13  2  1
sorted Numbers are :
1  2  4  5  13
Time taken is : 1.00000e-05
```

EXP. NO.

MVSR Engineering College

Page No. : 12
Date :

```
Void main() {
    int n, *a, k;
    clock_t st, et;
    double ts;
    printf("\n Enter how many numbers ");
    scanf("%d", &n);
    a = (int *) calloc (n, sizeof(int));
    printf("\n The Random numbers \n");
    for (k=0; k<n; k++) {
        a[k] = n-k;
        printf("%d\t", a[k]);
    }
    st = clock();
    Quicksort (a, 0, n-1);
    et = clock();
    ts = (double) (et-st) / clocks_PER_SEC;
    printf("\n sorted numbers are :\n");
    for (k=0; k<n; k++)
        printf("%d\t", a[k]);
    printf("\n Time taken is %e", ts);

}
```

while (

1. Algorithm Greedy Knapsack(

EXP. NO. 7.

```
Algorithm StraightMaxMin(a, n, max, min)
//set max to the first index element, min also. of a[1:n]
{
    max := min := a[i];
        for  i := 2 to n do
        {
            if (a[i] > max) then  max := a[i];
            if (a[i] < min) then  min := a[i];
        }
}
```

output:

5

10 4 23 7 15.

Min: 4

Max: 23

EXP. NO. 7.

MVSR Engineering College

Page No. : 13
Date :

MAXMIN (Brute force)

```
#include <stdio.h>
int main() {
    int n;
    scanf ("%d", &n);
    int arr[n];
    for (int i = 0; i<n; i++)
        scanf ("%d", &arr[i]);
    int min = arr[n];
    int max = arr[0];
    for (int i=1; i<n; i++) {
        if (arr[i] < min)
            min = arr[i];
        if (arr[i] > max) {
            max = arr[i];
    }
    printf ("Min : %d \n Max = %d \n", min, max);
    return n;
}
```

```
while (P[i] != -1)
{

Algorithm MaxMin(i, j, max, min)
// a[1:n] is a global array. the effect is to set max, min to
// the largest, smallest values in a[i,j]
{
    if (i = j) then max := min := a[i];
    else if (i = j - 1) then
    {
        if (a[i] < a[j]) then {
            max := a[j]; min := a[i];
        }
        else {
            max := a[i];
            min := a[j];
        }
    }
    else {
        mid := (i + j) / 2
        MaxMin(i, mid, max, min);
        MaxMin(mid+1, j, max1, min1);
        if(max < max1) then max := max1;
        if (min > min1) then min := min1;
    }
}
```

EXP. NO. 8.

Page No.: 14
Date :

Max Min (divide and conquer)

```
#include <stdio.h>
int max, min;
int a[100];
void maxmin(int i, int j){
    int max1, min1, mid;
if (i==j)
        max = min = a[i];
else if (i==j-1){
        if (a[i]<a[j])
            max = a[j];
            min = a[i];
        else
            max = a[i];
            min = a[j];
}
else {
        max = a[i];
        mid = (i+j)/2
        maxmin(i, mid);
        max1 = max; min1 = min;
        maxmin(mid+1, j);
        if (max < max1)
            max = max1;
        if (min > min1)
            min = min1;
}
}
```

output :

```
enter n : 5
enter nums :
3 9 7 2 5
Minimum is : 2
Maximum is : 9
```

EXP. NO.

MVSR Engineering College

Page No. : 15
Date :

```
int main() {

    int i, num;
    printf("\n enter n \n");
    scanf("%d", &num);
    printf("enter nums \n");
    for (i=1; i<=num; i++)
        scanf("%d", &a[i]);
    max = a[0];
    min = a[0];
    maxmin(1, num);
    printf("Minimum is %d \n", min);
    printf("Maximum is %d \n", max);
return 0;

}
```

EXP. NO.

MVSR Engineering College

Page No. : 16
Date :

9. Knapsack (Brute Force)

```
#include <stdio.h>
#include <math.h>
int knapsack (float P[], float w[], float capacity, int n) {
  int bestchoice[20], A[20];
  int i,j,k;
  float bestWeight = 0, bestValue = 0;
  float tempWeight, tempValue;
  for (i=0; i<n; i++){
      A[i]=0;
      bestchoice[i] = 0;
  for (i=1; i<= pow(2,n); i++) {
      int j = n-1; tempWeight = 0; tempValue=0;
      while (A[j]!=0 && j>0) {
                A[j] = 0;
                j = j-1;
      } A[j] = 1;
  for (k=0; k<n; k++) {
      if (A[k] == 1) {
          tempweight += w[k];
          tempValue += P[k];
}}

if ((tempvalue > bestValue) && (tempWeight <= Capacity )){
          bestValue = tempValue;
          bestWeight = tempWeight;
          for (k=0; k<n; k++)
                bestchoice[k] = A[k];
      }
}
```

```
printf ("solution vector \n");
for (i=0; i<n; i++)
        printf ("%d", bestChoice[i]);
printf ("\n");

    printf ("Best Value = %.2f \n", bestValue);
    printf ("Best Weight = %.2f \n", bestWeight);

    return 0;
}
int main() {
    int n, i;
    float capacity;
    float Profit[20], weight[20];

    printf ("enter no. of items");
    scanf ("%d", &n);
    printf ("enter knapsack capacity");
    scanf ("%f", &capacity);
    printf ("enter profits of items \n");
    for (i=0; i<n; i++)
        scanf ("%f", &profit[i]);

    printf ("enter weights of items");
    for (i=0; i<n; i++)
        scanf ("%f", &weight[i]);

    knapsack (profit, weight, capacity, n);
    return 0;
}
```

10. Knapsack (Greedy):

```
#include <stdio.h>
void knapsack (int n, float weight[], float profit[], float capacity) {
    float x[20], tp=0;
    int i;
    float u = capacity;
    for (i=0; i<n; i++)
        x[i] = 0.0;
    for (i=0; i<n; i++) {
        if (weight[i] > u)
            break;
        else {
            x[i] = 1.0;
            tp = tp + profit[i];
            u = u - weight[i];
        }
    }
    if (i<n) {
        x[i] = u/ weight[i];
        tp = tp + (x[i] * profit[i]);
    }
    printf ("the result vector is \n");
    for (i=0; i<n; i++)
        printf ("%f", x[i]);

    printf ("Max profit is: %f \n", tp);
}

int main() {
    float weight[20], profit[20], capacity;
    int num, j, i;
```

EXP. NO.

MVSR Engineering College

Page No. : 19
Date :

```
float ratio[20], temp;
printf("enter n");
scanf("%d", &num);
printf("enter weights");
for(i=0; i<num; i++)
        scanf("%f", &weight[i]);
printf("enter profits");
for(i=0; i<num; i++)
        scanf("%f", &profit[i]);
printf("knapsack capacity");
scanf("%f", &capacity);
for(i=0; i<num; i++)
        ratio[i] = profit[i] / weight[i];

for(i=0; i<num; i++){
  for(j=0; j<num-1; j++){
        if(ratio[j] < ratio[j+1]){
                temp = ratio[j];
                ratio[j] = ratio[j+1];
                ratio[j+1] = temp;
                temp = weight[j];
                weight[j] = weight[j+1];
                weight[j+1] = temp;
                temp = profit[j];
                profit[j] = profit[j+1];
        }       profit[j+1] = temp;
  }
}
knapsack(num, weight, profit, capacity);
return 0;
}
```

EXP. NO. 11.

MVSR Engineering College

Page No. : 20
Date :

Prims

```
#include <stdio.h>
#define MAX_NODES 10
#define INF 99999
int cost[MAX_NODES][MAX_NODES];
int near[MAX_NODES];
int t[MAX_NODES][2];

int main() {
    int n, i, j, k, l, min, mincost = 0, minval, minj;
    print("Enter the number of nodes");
    scanf("%d", &n);
    printf("Enter the 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 && i!=j){
                cost[i][j] = INF;
            }
        }
    }
    min = INF;
    for (i=1; i<=n; i++){
        for (i=1; i<=n; i++){
            for (j=1; j<=n; j++) {
                if (cost[i][j] < min){
                    min = cost[i][j];
                    k = i;
                    l = j;
                }
            }
        }
    }
```

EXP. NO.

MVSR Engineering College

Page No. : 21
Date :

```
t[i][1] = k;
t[i][2] = l;
mincost = cost[k][l];

for (i=1; i<=n; i++) {
    if (cost[i][l] < cost[i][k])
        near[i] = l;
    else
        near[i] = k;
}
near[k] = near[l] = 0;

for (i = 2; i <= n-1; i++){
    minval = INF;
    minj = 0;
    for (j = 1; j <= n; j++){
        if (near[j] != 0 && cost[j][near[j]] < minval){
            minval = cost[j][near[j]];
            minj = j;
        }
    }
    t[i][1] = minj;
    t[i][2] = near[minj];
    mincost += cost[minj][near[minj]];
    near[minj] = 0;
    for (k=1; k<=n; k++){
        if (near[k] != 0 && cost[k][near[k]] > cost[k][minj])
            near[k] = minj;
    }
}
```

EXP. NO.

MVSR Engineering College

Page No.: 22
Date :

```
printf ("\n min cost = %d \n", mincost);
printf ("\n Edges in Mini. cost spanning tree : \n");
for (i=1; i<=n-1; i++){
        printf ("%d - %d \n", t[i][1], t[i][2]);
}
return 0;
}
```

EXP. NO.

MVSR Engineering College

Page No.: 23
Date :

12. Kruskals (Greedy)

```
#include <stdio.h>
# define MAX 20
# define INF 999
int a[MAX][MAX];
int p[MAX];
int ne = 1;
int mincost = 0;

int find (int i){
    while (p[i] != -1)
        i = p[i];
    return i;
}

void Union (int i, int j){
    p[i] = j;
}

int main (){
    int n, i, j, v1, v2, u, v, cmin;

    printf ("enter number of vertices");
    scanf ("%d", &n);

    for (i=1; i<=n; i++){
        p[i] = -1;
    }
    printf ("Enter adjacency matrix: \n");
```

```
for(i=1 ; i<=n; i++){
    for(j=1 ; j<=n ; j++){
        scanf("%d", &a[i][j]);
        if (a[i][j] == 0 && i!=j)
            a[i][j] = INF;
    }
}
printf(" Edges in the minimum cost spanning Tree \n");

while(ne<n){
    cmin = INF;
    v1 = v2 = 0;

    for(i=1 ; i<=n; i++){
        for(j=1 ; j<=n ; j++){
            if (a[i][j] < cmin){
                cmin = a[i][j];
                v1 = i;
                v2 = j;
            }
        }
    }
    u = find(v1);
    v = find(v2);
    if (u != v){
        union(u,v);
        printf("(%d, %d) = %d\n", v1, v2, cmin);
        mincost += cmin;
        ne++;
    }
```

EXP. NO.

MVSR Engineering College

Page No. : 25
Date :

```
    a[v1][v2] = a[v2][v1] = INF;
}
printf("Minimum cost = %d\n", mincost);
return 0;
}
```

Dijkstras (single Source shortest path)

```
#include <stdio.h>
#include <stdbool.h>
#define MAX 100
#define INF 99999

void dijkstra (int cost[MAX][MAX], int dist[MAX], int n, int start){
    bool s[MAX];
    int i,j;
    for(i=0;i<n;i++){
        dist[i] = INF;
        s[i] = false;
    }
    dist[start] = 0;

    for(int count=0; count<n-1; count++){
        int minDist = INF, u = -1;
        for(i=0; i<n; i++){
            if(!s[i] && dist[i]<=minDist){
                minDist = dist[i];
                u=i;
            }
        }
    }
    s[u] = true;
    for(j=0;j<n;j++){
        if(!s[j] && cost[u][j] != INF && dist[u] != INF
            dist[u] + cost[u][j] < dist[j]){
            dist[j] = dist[u] + cost[u][j];
        }
    }
}
```

EXP. NO.

MVSR Engineering College

Page No. : 47
Date :

```
int main() {
    int n, start;
    int cost[MAX][MAX], dist[MAX];

    printf("Enter the number of vertices");
    scanf("%d", &n);

    printf("Enter the adjacency matrix");
    for(int i=0; i<n; i++){
        for(int j=0; j<n; j++) {
            scanf("%d", &cost[i][j]);
        }
    }
    printf("Enter the starting vertex (0 to %d):", n-1);
    scanf("%d", &start);

    dijkstra(cost, dist, n, start);

    printf("Shortest distances from vertex %d \n", start);
    for(int i=0; i<n; i++){
        printf("To vertex %d : %d \n", i, dist[i]);
    }

    return 0;
}
```