Lab-1(Insertion sort, Selection sort)

Objective:

- The objective of this lab is to implement two different type of sorting algorithm.

Prog-1

Title: A program to implement Insertion Sort.

Source Code:

```
#include<stdio.h>
int main() {
int array[100],n,i,j,key;
printf("Enter the Number of elements: ");
scanf("%d",&n);
printf("\nEnter the Elements:\n");
for(i=1;i<=n;i++)
scanf("%d",&array[i]);
for(j=2;j<=n;j++)  {
key=array[j];
i=j-1;
while(i>0&&array[i]>key) {
array[i+1]=array[i];
i=i-1;  }
array[i+1]=key;  }
printf("Elements sorted in ascending order:\n");
for(i=1;i<=n;i++)
printf(" %d",array[i]);
return 0; }
```

Sample Output:

```
Enter the Number of elements: 5

Enter the Elements:
23 1 77 56 92
Elements sorted in ascending order:
 1 23 56 77 92
```

Prog-2

Title: A program to implement Selection Sort.

Source Code:

```
#include<stdio.h>
int main() {
int A[100], n,i,j,index_min,temp;
printf("Enter number of elements:\n");
scanf("%d", &n);
printf("Enter %d integers:\n", n);
for (i = 0; i < n; i++)
scanf("%d", &A[i]);
for(i=0;i<n-1;i++) {
index_min=i;
for(j=i+1;j<n;j++) {
if(A[j]<A[index_min]) {
index_min=j;  }  }
if(index_min!=i)  {
temp=A[i];
A[i]=A[index_min];
A[index_min]=temp;} }
 printf("Sorted list in ascending order:\n");
```

```
for (i = 0; i < n; i++)
printf("%d\n", A[i]);
return 0; }
```

Sample Output:

```
Enter number of elements:
5
Enter 5 integers:
34 2 78 1 93
Sorted list in ascending order:
1
2
34
78
93
```

Lab-2(Merge sort)

Objective:

- The objective of this lab is to implement a sorting algorithm.

Prog-1

Title: A program to implement Merge Sort.

Source Code:

```
#include<stdio.h>
void merge(int AR[],int p,int q,int r){
int n1=q-p+1;
int i,j,k;
int n2=r-q;
int L[n1];
int R[n2];
```

```
for(i=0;i<n1;i++) {
L[i]=AR[p+i]; }
for(j=0;j<n2;j++){
R[j]=AR[q+j+1]; }
L[n1]=-9999;
R[n2]=-9999;
i=0;
j=0;
k=p;
while(i<n1 && j<n2) {
if(L[i] <= R[j]) {
AR[k] = L[i];
i++; }
else {
AR[k] = R[j];
j++; }
k++; }
while(i<n1) {
AR[k] = L[i];
i++;
k++;  }
while(j<n2) {
AR[k] = R[j];
j++;
k++; } }
void merge_sort(int A[], int p, int r){
int q;
if(p < r){
```

```
q=(p+r)/2;
merge_sort(A,p,q);
merge_sort(A,q+1,r);
merge(A,p,q,r);  } }
void printArray(int A[],int n)
{  int i;
printf("Elements sorted in ascending order: \n");
for(i=0;i<n;i++) {
printf("%d\n",A[i]); } }
int main()  {
int n,i,A[100];
printf("Enter the Number of elements: ");
scanf("%d",&n);
printf("Enter the elements: \n");
for(i=0;i<n;i++){
scanf("%d",&A[i]); }
merge_sort(A,0,n-1);
printArray(A,n);
return 0; }
```

Sample Output:

```
Enter the Number of elements: 5
Enter the elements:
23 14 6 93 4
Elements sorted in ascending order:
4
6
14
23
93
```

Lab-3(Quick sort, Heap sort)

Objective:

- The objective of this lab is to implement two different type of sorting algorithm.

Prog-1

Title: A program to implement Quick Sort.

Source Code:

```
#include<stdio.h>
void swap(int* a, int* b) {
int t = *a;
*a = *b;
*b = t; }
int partition (int A[], int low, int high) {
int pivot = A[high];
int i = (low - 1);
for (int j = low; j <= high- 1; j++) {
if (A[j] < pivot) {
i++;
swap(&A[i], &A[j]); } }
swap(&A[i + 1], &A[high]);
return (i + 1); }
void quickSort(int A[], int low, int high) {
if (low < high) {
int pi = partition(A, low, high);
quickSort(A, low, pi - 1);
quickSort(A, pi + 1, high); } }
void printArray(int A[], int size) {
int i;
```

```
for (i=0; i < size; i++)
printf("%d ", A[i]);
printf("\n"); }
int main() {
int A[] = {10, 7, 8, 9, 1, 5};
int n = sizeof(A)/sizeof(A[0]);
quickSort(A, 0, n-1);
printf("Sorted Aay: \n");
printArray(A, n);
return 0; }
```

Sample Output:

```
Sorted Aay:
1 5 7 8 9 10
```

Prog-2

Title: A program to implement Heap Sort.

Source Code:

```
#include<istream>
using namespace std;
void max_heapify(int A[], int n, int i) {
int temp=0;
int largest = i;
int l = 2*i + 1;
int r = 2*i + 2;
if (l < n && A[l] > A[largest])
largest = l;
```

```
if (r < n && A[r] > A[largest])
largest = r;
if (largest != i) {
temp=A[i];
A[i]=A[largest];
A[largest]=temp;
max_heapify(A, n, largest); }  }
void heapSort(int A[], int n) {
for (int i = n / 2 - 1; i >= 0; i--)
max_heapify(A, n, i);
for (int i=n-1; i>=0; i--)
{ swap(A[0], A[i]);
max_heapify(A, i, 0);  } }
void printArray(int A[], int n) {
for (int i=0; i<n; ++i)
printf("%d\n",A[i]); }
int main() {
int A[100],n,i;
printf("Enter the number of Elements: \n");
scanf("%d",&n);
printf("Enter the Numbers to be sorted: \n");
for(i=0;i<n;i++)
scanf("%d",&A[i]);
heapSort(A, n);
printf("Sorted array in ascending order:\n");
printArray(A, n); }
```

Sample Output:

```
Enter the number of Elements:
5
Enter the Numbers to be sorted:
34 28 9 33 1
Sorted array in ascending order:
1
9
28
33
34
```

Lab-4 (BFS,DFS)

Objective:

- The objective of this lab is to implement two different type of searching algorithm.

Prog-1

Title: A program to implement Breadth First Search (BFS) algorithm.

Source Code:

```
#include<stdio.h>

int a[20][20], q[20], visited[20], n, i, j, f = 0, r = -1;

void bfs(int v) {

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

if(a[v][i] && !visited[i])

q[++r] = i;

if(f <= r) {

visited[q[f]] = 1;

bfs(q[f++]); }}

int main() {

int v;
```

```
printf("Enter the number of vertices:");
scanf("%d", &n);
for(i=1; i <= n; i++) {
q[i] = 0;
visited[i] = 0; }
printf("Enter graph data in matrix form:\n");
for(i=1; i<=n; i++) {
for(j=1;j<=n;j++) {
scanf("%d", &a[i][j]); } }
printf("Enter the starting vertex:");
scanf("%d", &v);
bfs(v);
printf("The node which are reachable are:\n");
for(i=1; i <= n; i++) {
if(visited[i])
printf("%d\t", i);
else {
printf("Bfs is not possible. Not all nodes are reachable");
break; } } }
```

Sample Output:

```
Enter the number of vertices:3
Enter graph data in matrix form:
2 5 3
5 3 1
7 3 5
Enter the starting vertex:2
The node which are reachable are:
1       2       3
```

Prog-2

Title: A program to implement Depth First Search(DFS) algorithm.

Source Code:

```
#include<stdio.h>
#include<conio.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); } }
int main() {
int i,j,count=0;
printf("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("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");
for (i=1;i<=n;i++) {
```

```
if(reach[i])
count++; }
if(count==n)
printf("Graph is connected"); else
printf("Graph is not connected");
return 0; }
```

Sample Output:

```
Enter number of vertices:4
Enter the adjacency matrix:
2 5 2 5
6 1 2 0
4 2 5 3
5 1 9 2

 1->2
 2->3
 3->4
Graph is connected
```

Home work (three sorting algorithm)

Prog-1

Title: A program to implement Radix Sort algorithm.

Source Code:

```
#include<stdio.h>
int A[100];
int POCKET[10][9];
int N;
```

```
void RADIX(void);
int main()  {
int i,j;
for(i=0;i<10;i++)
for(j=0;j<9;j++)
POCKET[i][j]=-9999;
printf("Enter the Number of elements to be sorted: ");
scanf("%d",&N);
printf("\nEnter the Elements: \n");
for(i=0;i<N;i++){
scanf("%d",&A[i]);  }
RADIX();
printf("Here are your elements sorted in ascending order: \n");
for(i=N-1;i>=0;i--)  {
printf("%d\n",A[i]);  } }
void RADIX(void){
int a,b,c,temp,l,x=1;
int k=3;
for(k=1;k<=3;k++){
for(l=0;l<N;l++){
temp=(A[l]/x)%10;
POCKET[temp][l]=A[l]; }
x=x*10;
c=0;
for(a=9;a>=0;a--){
for(b=0;b<9;b++){
if(POCKET[a][b]!=-9999){
```

```
A[c]=POCKET[a][b];
c=c+1; } }  }
for(a=0;a<10;a++)
for(b=0;b<9;b++)
POCKET[a][b]=-9999;  }  }
```

Sample Output:

```
Enter the Number of elements to be sorted: 5

Enter the Elements:
33 2 53 89 1
Here are your elements sorted in ascending order:
1
2
33
53
89
```

Prog-2

Title: A program to implement Counting Sort algorithm.

Source Code:

```
#include <stdio.h>
void countingSort(int array[], int size) {
int output[10];
int max = array[0];
for (int i = 1; i < size; i++) {
if (array[i] > max)
max = array[i]; }
int count[10];
for (int i = 0; i <= max; ++i) {
```

```
count[i] = 0; }
for (int i = 0; i < size; i++) {
count[array[i]]++; }
for (int i = 1; i <= max; i++) {
count[i] += count[i - 1]; }
for (int i = size - 1; i >= 0; i--) {
output[count[array[i]] - 1] = array[i];
count[array[i]]--; }
for (int i = 0; i < size; i++) {
array[i] = output[i]; } }
void printArray(int array[], int size) {
for (int i = 0; i < size; ++i) {
printf("%d  ", array[i]); }
printf("\n"); }
int main() {
int array[] = {4, 2, 2, 8, 3, 3, 1};
int n = sizeof(array) / sizeof(array[0]);
countingSort(array, n);
printArray(array, n); }
```

Sample Output:

```
1   2   2   3   3   4   8
```

Prog-3

Title: A program to implement Bucket Sort algorithm.

Source Code:

```
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;
void bucketSort(float arr[], int n) {
vector<float> b[n];
for (int i=0; i<n; i++) {
int bi = n*arr[i]; // Index in bucket
b[bi].push_back(arr[i]); }
for (int i=0; i<n; i++)
sort(b[i].begin(), b[i].end()); int index = 0;
for (int i = 0; i < n; i++)
for (int j = 0; j < b[i].size(); j++)
arr[index++] = b[i][j]; }
int main() { float arr[] = {0.897, 0.565, 0.656, 0.1234, 0.665, 0.3434};
int n = sizeof(arr)/sizeof(arr[0]);
bucketSort(arr, n);
cout << "Sorted array is \n";
for (int i=0; i<n; i++)
cout << arr[i] << " ";
return 0;  }
```

Sample Output:

```
Sorted array is:
0.1234 0.3434 0.565 0.656 0.665 0.897
```

Lab-5 (Activity Selection problem, 0-1 knapsack)

Objective:

- The objective of this lab is to implement two different type of problem of greedy algorithm.

Prog-1

Title: A program to implement activity selection problem.

Source Code:

```
#include <stdio.h>
void ActivitySelection(int start[], int finish[], int n){
  printf("The following activities are selected:\n");
  int j = 0;
  printf("%d ", j);
  int i;
  for (i = 1; i < n; i++  {
  if (start[i] >= finish[j]   {
 printf("%d ", i);
 j = i;  }  } }
int main() {
int start[] = {1, 3, 2, 0, 5, 8, 11};
int finish[] = {3, 4, 5, 7, 9, 10, 12};
int n = sizeof(start) / sizeof(start[0]);
ActivitySelection(start, finish, n);
return 0; }
```

Sample Output:

```
Following activities are selected:1 3 7 10
```

Prog-2

Title: A program to implement 0-1 knapsack problem.

Source Code:

```
#include<stdio.h>
int max(int a, int b) {
if(a>b) {return a;}
else {return b;}}
int knapsack (int W, int wt[], int val[], int n) {
int i, w;  int knap[n+1] [W+1];
for (i = 0; i <= n; i++) {
for (w = 0; w <= W; w++) {
if (i==0 || w==0)
knap[i][w] = 0;
else if (wt[i-1] <= w)
knap[i][w] = max(val[i-1] + knap[i-1][w-wt[i-1]], knap[i-1][w]);
else  knap[i][w] = knap[i-1][w]; } }
return knap[n][W]; }
int main() {
int val[] = {20, 25, 40};
int wt[] = {25, 20, 30};
int W = 50;
int n = sizeof(val)/sizeof(val[0]);
printf("The solution is : %d", knapsack(W, wt, val, n));
return 0; }
```

Sample Output:

```
The solution is : 65
```

Lab-6 (Prim's algorithm, Kruskals algorithm)

Objective:

- The objective of this lab is to implement two different type of algorithms to find the shortest path from a graph using MST(Minimum Spanning Tree)

Prog-1

Title: A program to implement Prim's algorithm.

Source Code:

```
#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() {
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; }
```

Sample Output:

```
        Implementation of Kruskal's algorithm

Enter the no. of vertices:3

Enter the cost adjacency matrix:
0 8 9
3 0 3
0 3 0
The edges of Minimum Cost Spanning Tree are
1 edge (2,1) =3
2 edge (2,3) =3
```

Prog-2

Title: A program to implement Prim’s algorithm.

Source Code:

```
#include<stdio.h>
#include<conio.h>
int a,b,u,v,n,i,j,ne=1;
int visited[10]= {0},min,mincost=0,cost[10][10];
void main() {
printf("\nEnter the number of nodes:")
scanf("%d",&n);
printf("\nEnter 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)
cost[i][j]=999; }
visited[1]=1;
printf("\n");
while (ne < n) {
for (i=1 ,min=999; i<=n; i++)
for (j=1; j<=n; j++)
if(cost[i][j]< min)
if(visited[i]!=0)
{ min=cost[i][j];
a=u=i; b=v=j;}
if(visited[u]==0 || visited[v]==0) {
printf("\n Edge %d:(%d %d) cost:%d",ne++,a,b,min);
mincost+=min;
```

visited[b]=1; }

cost[a][b]=cost[b][a]=999; }

printf("\n Minimun cost=%d",mincost);

getch(); }

Sample Output:

```
Enter the adjacency matrix:
0 4 0 0 0 0 0 8 0
4 0 8 0 0 0 0 11 0
0 8 0 7 0 4 0 0 2
0 0 7 0 9 14 0 0 0
0 0 0 9 0 10 0 0 0
0 0 4 14 10 0 2 0 0
0 0 0 0 0 2 0 1 6
8 11 0 0 0 0 1 0 7
0 0 2 0 0 0 6 7 0

 Edge 1:(1 2) cost:4
 Edge 2:(1 8) cost:8
 Edge 3:(8 7) cost:1
 Edge 4:(7 6) cost:2
 Edge 5:(6 3) cost:4
 Edge 6:(3 9) cost:2
 Edge 7:(3 4) cost:7
 Edge 8:(4 5) cost:9
 Minimun cost=37
```

Lab-7 (Dijkstra's algorithm, Bellman Ford algorithm)

Objective:

- The objective of this lab is to implement two different type of algorithms to find the single shortest path from a graph.

Prog-1

Title: A program to implement Dijkstra's algorithm.

Source Code:

```
#include <stdio.h>
#include <limits.h>
#define V 5
int minDistance(int dist[],bool sptSet[]) {
int min = INT_MAX, min_index;
for (int v = 0; v < V; v++)
if (sptSet[v] == false &&
dist[v] <= min)
min = dist[v], min_index = v;
return min_index; }
void printPath(int parent[], int j) {
if (parent[j] == - 1)   return;
printPath(parent, parent[j]);
printf("%d ", j); }
int printSolution(int dist[], int n,
int parent[]) { int src = 0;
printf("Vertex\t        Distance\t       Path");
for (int i = 1; i < V; i++) {
printf("\n%d -> %d \t\t %d\t\t%d ",
src, i, dist[i], src);
printPath(parent, i); } }
void dijkstra(int graph[V][V], int src) {
int dist[V]; int parent[V];
for (int i = 0; i < V; i++) { parent[0] = -1;
dist[i] = INT_MAX;
```

```
sptSet[i] = false; }
dist[src] = 0;
for (int count = 0; count < V - 1; count++)  {
int u = minDistance(dist, sptSet);
sptSet[u] = true;
for (int v = 0; v < V; v++)
if (!sptSet[v] && graph[u][v] &&
dist[u] + graph[u][v] < dist[v]) {parent[v] = u;
dist[v] = dist[u] + graph[u][v]; } }
printSolution(dist, V, parent); }
int main() {
int graph[V][V] = { {0, 10, 3, 9, 5},
{0, 0, 1, 2, 3},
{0, 4, 0, 6, 2},
{0, 0, 0, 0, 7},
{0, 0, 0, 9, 0},  };
dijkstra(graph, 0);
return 0; }
```

Sample Output:

```
Vertex              Distance             Path
0 -> 1               7                 0 2 1
0 -> 2               3                 0 2
0 -> 3               9                 0 3
0 -> 4               5                 0 4
```

Prog-2

Title: A program to implement Bellman Ford algorithm.

Source Code:

```
#include <stdio.h>
#include <stdlib.h>
int Bellman_Ford(int G[20][20] , int V, int E, int edge[20][2])
{ int i,u,v,k,distance[20],parent[20],S,flag=1;
for(i=0;i<V;i++)
distance[i] = 1000 , parent[i] = -1 ;
printf("Enter source: ");
scanf("%d",&S);
distance[S-1]=0 ;
for(i=0;i<V-1;i++) {
for(k=0;k<E;k++) {
u = edge[k][0] , v = edge[k][1] ;
if(distance[u]+G[u][v] < distance[v])
distance[v] = distance[u] + G[u][v] , parent[v]=u ; } }
for(k=0;k<E;k++) {
u = edge[k][0] , v = edge[k][1] ;
if(distance[u]+G[u][v] < distance[v])
flag = 0 ; }
if(flag)
for(i=0;i<V;i++)
printf("Vertex %d -> cost = %d parent = %d\n",i+1,distance[i],parent[i]+1);
return flag; }
int main() {
int V,edge[20][2],G[20][20],i,j,k=0;
printf("BELLMAN FORD\n");
```

```
printf("Enter no. of vertices: ");
scanf("%d",&V);
printf("Enter graph in matrix form:\n");
for(i=0;i<V;i++)
for(j=0;j<V;j++) {
scanf("%d",&G[i][j]);
if(G[i][j]!=0)
edge[k][0]=i,edge[k++][1]=j; }
if(Bellman_Ford(G,V,k,edge))
printf("\nNo negative weight cycle\n");
else printf("\nNegative weight cycle exists\n");
return 0; }
```

Sample Output:

```
BELLMAN FORD
Enter no. of vertices: 5
Enter graph in matrix form:
0 10 11 5 14
0 0 1 2 5
11 21 0 16 4
9 3 9 0 2
7 17 -6 -3 0
Enter source: 1

Negative weight cycle exists
```

Lab-8 (Floyd Warshall algorithm)

Objective:

- The objective of this lab is to implement two different type of algorithms to find the shortest path from a graph using MST (Minimum Spanning Tree)

Prog-1

Title: A program to implement Floyd Warshall algorithm.

Source Code:

```
#include<iostream>

#include<iomanip>

#define NODE 4

#define INF 999

using namespace std;

int costMat[NODE][NODE] = {

{0, 8, INF, 1},

{INF, 0, 1, INF},

{4, INF, 0, INF},

{INF, 2, 9, 0}, };

void floydWarshal() {

int cost[NODE][NODE];

for(int i = 0; i<NODE; i++)

for(int j = 0; j<NODE; j++)

cost[i][j] = costMat[i][j];

for(int k = 0; k<NODE; k++) {

for(int i = 0; i<NODE; i++)

for(int j = 0; j<NODE; j++)

if(cost[i][k]+cost[k][j] < cost[i][j])

cost[i][j] = cost[i][k]+cost[k][j]; }
```

```
cout << "The matrix:" << endl;
for(int i = 0; i<NODE; i++) {
for(int j = 0; j<NODE; j++)
cout << setw(3) << cost[i][j];
cout << endl; } }
int main() {
floydWarshal(); }
```

Sample Output:

```
The matrix:
  0  3  4  1
  5  0  1  6
  4  7  0  5
  7  2  3  0
```

Lab-9 (Rod cutting problem)

Objective:

- The objective of this lab is to implement a dynamic problem.

Prog-1

Title: A program to implement Rod Cutting problem.

Source Code:

```
#include<iostream>
#include<climits>
using namespace std;
int rodCutting(int n, int value[])
{  int i,j;
int result[n+1];
```

```
result[0]=0;
for(i=1; i<=n; i++) {
result[i]=INT_MIN;
for(j=0; j<i; j++) {
result[i]=max(result[i],value[j]+result[i-(j+1)]); } }
return result[n];  }
int main() { int n;
cout<<"Enter the length of the rod:"<<endl;
cin>>n;
int value[n];
cout<<"Enter the values of pieces of rod of all size:"<<endl;
for(int i=0; i<n; i++)
cin>>value[i];
cout<<"Maximum obtainable value by cutting up the rod in many pieces are:"<<endl;
cout<<rodCutting(n,value);
cout<<endl;
return 0; }
```

Sample Output:

```
Enter the length of the rod:
1 5 8 9 10 17 17 20 24 30
Enter the values of pieces of rod of all size:
Maximum obtainable value by cutting up the rod in many pieces are:
5
```

Index:

List of Implemented Algorithms

Sorting:

Searching:

Greedy Algorithms:

Dynamic Programming: