ADA LAB PROGRAMS

Program 1 :WAP to implement linear search algorithm repeat for different value of N , number of element in list to be searched and plot A graph of time taken versus N.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<time.h>
int linear(int  a[],int n,int key)
{
int i,flag=0;
for(i=0;i<n;i++)
{
if(a[i]==key)
{
return(i);
flag=0;
}
}
if(flag==0)
return(-1);
}
void main()
{
int *a,i,n,key,pos;
clock_t start,end;
clrscr();
printf("enter the size of an array");
scanf("%d",&n);
a=(int*)calloc(n,sizeof(int));
printf("elements are:");
for(i=0;i<n;i++)
{
a[i]=rand();
printf("%d\n",a[i]);
}
printf("enter key to search\n");
scanf("%d",&key);
start=clock();
delay(110);
pos=linear(a,n,key);
end=clock();
if(pos==-1)
{
```

```
printf("key not found");
}
else
{
printf("%dis at pos %d",key,pos+1);
}
printf("time taken=%f",(end-start/CLK_TCK));
getch();
}
```

Program 2: WAP to implement Binay search algorithm repeat for different value of N , number of element in list to be searched and plot A graph of time taken versus N.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<time.h>
void main()
{
int *a,flag,n,i,item,result,j,temp;
clock_t start,end;
printf("enter the size ofthe array:");
scanf("%d",&n);
a=(int*)calloc(n,sizeof(int));
printf("elements are:");
for(i=0;i<n;i++)
{
a[i]=rand();
printf("%d",a[i]);
}
for(i=0;i<n;i++)
for(j=i+1;j<n;j++)
{
if(a[i]>a[j])
{
temp=a[i];
a[i]=a[j];
a[j]=temp;
}
}

printf("sorted array:");
for(i=0;i<n;i++)
printf("enter the elements to be searched:");
scanf("%d",&item);
start=clock();
delay(110);
flag=bsearch(a,item,0,n-1);
end=clock();
if(flag==-1)
printf("the item %d not found",item);
else
printf("the item %d found at pos %d",item,flag+1);
printf("time taken=%f",(end,start)/(CLK_TCK) );
getch();
}
int bsearch(int a[],int item,int first,int last)
{
int middle;
if(first>last)
return(-1);
else
{
middle=(first+last)/2;
if(item<a[middle])
return(bsearch(a,item,first,middle-1));
else if(item>a[middle])
return(bsearch(a,item,middle+1,last));
else
return(middle);
}
}
```

Program 3: WAP to solve towers of Hanoi problem and execute it for different number of disks.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
void toh(int n,char a,char c,char b)
{
if(n==1)
{
printf("\n move disk 1 from pole %c to pole %c",a,c);
return;
}
toh(n-1,a,b,c);
printf("\n move disk %d from pole %c to pole %c",n,a,c);
toh(n-1,b,c,a);
}
void main()
```

```
{
int n;
clrscr();
printf("enter number of discs");
scanf("%d",&n);
toh(n,'a','c','b');
getch();
}
```

Program 4:WAP to sort given set of no. using selection sort algorithm. Repat for different value of N , number of element in list to be searched and plot A graph of time taken versus N.the element can be read from a file or can be generated using random number generator.

```
#include<stdio.h>
void main()
{
int a[100],num,min,i,j,temp;
clrscr();
printf("\n please enter the totall elements:");
scanf("%d",&num);
printf("\n please enter the array elements :");
for(i=0;i<num;i++)
scanf("%d",&a[i]);
for(i=0;i<num-1;i++)
{
min=i;
for(j=i+1;j<num;j++)
{
if(a[min]>a[j]){
min=j;
}
}
if(min!=i)
{
temp=a[i];
a[i]=a[min];
a[min]=temp;
}
}
printf("\nresult:");
for(i=0;i<num;i++)
{
printf("%d\n",a[i]);
}
printf("\n");
getch();
}
```

Program 5: WAP To find value of A using brute force based algorithm and divide and conquer based algorithm.

```
#include<stdio.h>
int power(int n1,int n2);
void main()
{
int base,a,result;
clrscr();
printf("enter the base number  :");
scanf("%d",&base);
printf("enter power number(positive integer):");
scanf("%d",&a);
result=power(base,a);
printf("%d^%d=%d",base,a,result);
getch();
}
int power(int base,int a)
{
if(a!=0)
return(base*power(base,a-1));
else
return 1;
}
```

Pogram 6: WAP to implement quick sort algorithm repeat for different value of N , number of element in list to be searched and plot A graph of time taken versus N.

```
void quicksort(int numbers[],int array_size)
{
q_sort(numbers,0,array_size-1);
}
void q_sort(int numbers[],int left,int right)
{
int pivot,l_hold,r_hold;
l_hold=left;
r_hold=right;
pivot=numbers[left];
while(left<right)
{
```

```
while((numbers[right]>=pivot)&&(left<right))right--;
if(left!=right)
{
numbers[left]=numbers[right];left++;
}
while((numbers[left]<=pivot)&&(left<right))left++;
if(left!=right)
{
numbers[right]=numbers[left];right--;
}
}
numbers[left]=pivot;
pivot=left;
left=l_hold;
right=r_hold;
if(left<pivot)
q_sort(numbers,left,pivot-1);
if(right>pivot)
q_sort(numbers,pivot+1,right);
}
```

Program 7:WAP to find binomial co-effcient C(N,K),using brute force algorithm and also dynamicprogramming based algorithm.

```
#include<stdio.h>
#include<conio.h>
long int bin(int n,int k)
{
int i,j;
long int arr[20][30];
for(i=0;i<=n;i++)
{
for(j=0;j<=(k<i?k:i);j++)
{
if(i==j || j==0)
{
arr[i][j]=1;
}
else
{
arr[i][j]=arr[i-1][j]+arr[i-1][j-1];
}
}
}
return(arr[n][k]);
}
void main()
```

```
{
int n,k;
clrscr();
printf("\nenter the value of n");
scanf("%d",&n);
printf("\nenter the value of k");
scanf("%d",&k);
if(n<0 || k<0 || k>n)
{
printf("\n value cannot be calculated!!");
}
else
{
printf("\n the binomial coefficient is %d",bin(n,k));
}
getch();
}
```

Program 8 :WAP to impent floyds algorithm and nfind the lengths of shortest paths from every pairs of vertices in a graph

```
#include<stdio.h>
#include<conio.h>
int min(int,int);
void floyds(int p[10][10],int n)
{
int i,j,k;
for(k=1;k<=n;k++)
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
if(i==j)
p[i][j]=0;else
p[i][j]=min(p[i][j],p[i][k]+p[k][j]);
}
int min(int a,int b){
if(a<b)
return(a);else
return(b);
}
void main(){
int p[10][10],w,n,e,u,v,i,j;
clrscr();
printf("\n enter number of vertices:");
scanf("%d",&n);
printf("\n enter number of edges:");
scanf("%d",&e);
for(i=1;i<=n;i++){
```

```
for(j=1;j<=n;j++)
p[i][j]=999;
}
for(i=1;i<=e;i++){
printf("\n enter number of verticesof edges%d with its weight:",i);
scanf("%d %d %d",&u,&v,&w);
p[u][v]=w;
}
printf("\n matrix of input data:");
for(i=1;i<=n;i++){
for(j=1;j<=n;j++)
printf("%d\t",p[i][j]);
printf("\n");
}
floyds(p,n);
printf("\n transitive closure:");
for(i=1;i<=n;i++){
for(j=1;j<=n;j++)
printf("%d\t",p[i][j]);
printf("\n");
}
printf("\n the shortest path are:");
for(i=1;i<=n;i++)
for(j=1;j<=n;j++){
if(i!=j)
printf("\n <%d,%d>=%d",i,j,p[i][j]);
}
getch();
}
```

Program 9:WAP to evaluate a polynomial using brute force based algorithm and using homers rule and compare their performances,

```
#include<stdio.h>
#include<stdlib.h>
#define MAXSIZE 10
void main()
{
int array[MAXSIZE];
int i,num,power;
float x,polySum;
clrscr();
printf("enter the order of polynomial\n");
scanf("%d",&num);
printf("enter the value of x \n");
scanf("%f",&x);
printf("enter %d coefficints\n",num+1);
for(i=0;i<=num;i++)
{
scanf("%d",&array[i]);
}
polySum=array[0];
for(i=0;i<=num;i++)
{
polySum=polySum*x+array[i];
}
power=num;
printf("given polynomial is:\n");
for(i=0;i<=num;i++)
{
if(power<0)
{
break;
}
if(array[i]>0)
printf("+");
else if(array[i]<0)
printf("-");
else
printf(" ");
printf("%dx^%d",abs(array[i]),power--);
}
printf("\n sum of polynomial=%6.2f\n",polySum);
getch();
}
```

Program 10: WAP to solve the string matching problem using boyer moore apparaoch

```
#include<limits.h>
#include<string.h>
#include<stdio.h>
#define NO_OF_CHARS 256
int max(int a,int b) {return(a>b)?a:b;}
void badCharHeuristic(char*str,int size,int badchar[NO_OF_CHARS])
{
int i;
for(i=0;i<NO_OF_CHARS;i++)
badchar[i]=-1;
for(i=0;i<size;i++)
badchar[(int) str[i]]=i;
}
void search(char *txt,char *pat)
{
```

```
int m=strlen(pat);
int n=strlen(txt);
int badchar[NO_OF_CHARS];
int s=0;
badCharHeuristic(pat,m,badchar);
while(s<=(n-m))
{
int j=m-1;
while(j>=0 && pat[j]==txt[s+j])
j--;
if(j<0)
{
printf("\n pattern occurs at shift=%d",s);
s+=(s+m<n)? m-badchar[txt[s+m]]:1;
}
else
s+=max(1,j-badchar[txt[s+j]]);
}
}
void main()
{
char txt[]="SRISREENI";
char pat[]="SRE";
clrscr();
search(txt,pat);
getch();
}
```

Program 11:WAP to solve the string matching problem using KMP algorithm,

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void prefixSuffixArray(char* pat,int M,int* pps)
{
int length=0;
int i=1;
pps[0]=0;

while(i<M){
if(pat[i]==pat[length]){
length++;
pps[i]=length;
i++;
}
else{
if(length!=0)
length=pps[length-1];
else{
pps[i]=0;
i++;
}
}
}
}
void KMPAlgorithm(char* text,char* pattern)
{
int M=strlen(pattern);
int N=strlen(text);
int pps[M];
int i=0;
int j=0;
prefixSuffixArray(pattern,M,pps);

while(i<N){
if(pattern[j]==text[i]){
j++;
i++;
}
if(j==M){
printf("found pattern at index %d",i-j);
j=pps[j-i];
}
else if(i<N&&pattern[j]!=text[i]){
if(j!=0)
j=pps[j-1];
else
i=i+
1;
}
}
}
void main()
{
char text[]="sreenivasrao";
char pattern[]="sree";
printf("the pattern is found inthe text inthe following index");
KMPAlgorithm(text,pattern);
getch();
}
```

Program 12: WAP to implement BFS traversal algorithym.

```
#include <stdio.h>
#include<stdlib.h>
#define SIZE 40
struct queue
{
int items[SIZE];
int front;
int rear;
};

struct queue* createQueue();

void enqueue(struct queue* q, int);
int dequeue(struct queue* q);
void display(struct queue* q);
int isEmpty(struct queue* q);
void printQueue(struct queue* q);
struct node
{
int vertex;
struct node* next;
};

struct node* createNode(int);
struct Graph{
int numVertices;
struct node** adjLists;
int* visited;
};

//BPS algorithm

void bfs(struct Graph* graph, int startVertex)
 {
  struct queue* q=createQueue();

graph->visited[startVertex]=1;

enqueue(q,startVertex);

while (!isEmpty(q)){

printQueue(q);

int currentVertex=dequeue(q);
printf(" Visited %d\n", currentVertex);
```

```
 struct node* temp = graph->adjLists[currentVertex];

while (temp) {

int adjVertex=temp->vertex;

if (graph->visited[adjVertex]==0)
{
graph->visited[adjVertex]= 1;
enqueu(q, adjVertex);
}
temp=temp->next;
}
}
}
//Creating a node

struct node* createNode(int v){
struct node* newNode=malloc(sizeof(struct node));
newNode->vertex=v;
newNode->next = NULL;
 return newNode;
}
// Creating a graph

struct Graph* createGraph(int vertices) {

struct Graph *graph = malloc(sizeof(struct Graph));
graph->numVertices = vertices;

graph-> adjLists = malloc(vertices* sizeof(struct node*));
graph-> visited = malloc(vertices* sizeof(int));

int i;

for (i=0; i< vertices; i++) {

graph->adjLists[i] = NULL;

graph->visited[i] = 0;
  }
return graph;
}
//Add edge

void addEdge(struct Graph* graph, int src, int dest) {

//Add edge from src to dest
```

```
struct node* newNode = createNode(dest);

newNode->next = graph-> adjlists[src];

graph-> adjLists[src] = newNode;
}
//Add edge from dest to are

newNode=createNode(src);

newNode->next=graph->adjLists(dest);

graph-> adjLists[dest] = newNode;

//Create a queue

struct queue* createQueue() {

struct queue* q=malloc(sizeof(struct queue));

q->front = -1;
q-> rear=-1;

return q;

// Check if the queue is empty
int isEmpty(struct queue* q) {

if (q->rear==-1)

return 1;

else

return 0;

//Adding elements into queue

void enqueue(struct queue* q, int value) {

if (q->rear==SIZE -1)

printf("\nQueue is Full!");

else {

if (q->front == -1)
```

```
q->front = 0;
q->rear++;

q->items[q->rear] = value;
}
}

// Removing elements from queue
int dequeue(struct queue*q){

int item;

if (isEmpty(q)) {

printf("Queue is empty");

item = -1;

} else {

item=q->items[q->front];

q->front++;
 if (q->front> q->rear) {

printf("Resetting queue");
 q->front=q->rear = -1;

return item;
}
}

// Print the queue

void printQueue(struct queue* q) {

int i = q->front;

if (isEmpty(q)) {

printf("Queue is empty");

} else {

printf("\nQueue contains \n");

for (i=q->front; i <q->rear + 1, i++) {
```

```
printf("%d", q->items[i]);
}
}
}
int main() {
struct Graph* graph = createGraph(6);
addEdge(graph, 0, 1);
addEdge(graph, 0, 2);
addEdge(graph, 1, 4);
addEdge(graph, 2, 4);

addEdge(graph, 1, 2);

addEdge(graph, 1, 3);

addEdge(graph, 3, 4);

bfs(graph, 0);
return 0;
}
}
```

Program 13: WAP to find minimum spinning tree of given graph using prims algorithm.

```
#include<stdio.h>
#include<stdlib.h>
#define infinity 9999
#define MAX 20

int G[MAX][MAX], spanning[MAX][MAX],n;

int prims();

void main()

{

int i,j, total_cost;

printf("Enter no. of vertices.");

scanf("%d", &n);

printf("\nEnter the adjacency matrix\n");
```

```
for(i=0;i<n;i++)

for(j=0;j<n;j++)
scanf("%d", &G[i][j]);

total_cost=prims();

printf("\nspanning tree matrix\n");

for(i=0;i<n;i++)

{

printf("\n");

for(j=0;j<n;j++)

printf("%d\t", spanning[i][j]);

}

printf("\n\nTotal cost of spanning tree-%d", total_cost);

getch();

}

int prims()

{
int cost[MAX][MAX];

int u,v,min_distance, distance[MAX], from[MAX];
int visited[MAX],no_of_edges,i,min_cost,j;

//create cost matrix, spanning00

for(i=0;i<n;i++)

for(j=0;j<n;j++)

{
if(G[i][j]==0)

cost[i][j]=infinity ;

else
```

```
cost[i][j]=G[i][j]=0;

spanning[i][j]=0;

}

//initialise visited[],distance] and from []

distance[0]=0;

visited[0]=1;

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

{

distance[i]=cost[0][0];

from[i]=0;

visited[i]=0;
}
min_cost =0; //cost of spanning tree

no_of_edges=n-1; //no. of edges to be added while(no_of_edges>0) {
while(no_of_edges>0)
{
//find the vertex at minimum distance from the tree min_distance infinity

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

if(visited[i]==0 && distance[i]<min_distance)
 {
 v=i;

min_distance=distance[i];

}

u=from[v];

//insert the edge in spanning tree spanning[u][v]-distance[v],

spanning[v][u]=distance[v];
spanning[u][v]=distance[v];
no_of_edges--;
```

```
visited[v]=1;

//updated the distance] array

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

if(visited[i]==0&&cost[i][v]<distance[i])

{

distance[i]=cost[i][v];

from [i]=v;

}

min_cost=min_cost+cost[u][v];

}

return(min_cost);

}
```

Program 14:WAP to obtain the topological ordering of vertices in a given digraoh. Compue thetransitive closure of a given directed fraph usingwarshalls algorithm.

```
#include<stdio.h> using namespace std;
void warshall(int[10][10],int);
int main()
{

int a[10][10],i,j,n;

printf("Enter 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",&a[i][j]);
printf("The adjacency matirx is: \n");
 for(i=1;i<=n;i++)
 {
for(j=1;j<=n;j++)
{
 printf("%d\t",a[i][j]);
}

printf("\n");

}
warshall(a,n);
 return 0;
}

void warshall(int p[10][10], int n)

{

int i,j,k;
 for(k=1;k<=n;k++)
 {
for(j=1;j<=n;j++)
  {
for(i=1;i<=n;i++)
{
if((p[i][j]==0) && (p[i][k]==1) && (p[k][j]==1))
 {
 p[i][j]=1;
 }
 }
}
}
printf("\nThe path matrix is \n");
for(i=1;i<=n;i++)

{

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

{

printf("%d\t", p[i][j]);

}
printf("\n");

}
}
```