Program - 1

Design and implement a C/C++ program to execute a Binary Search program using Recursion

Code:

```
#include <stdio.h>
int BinarySearch(int array[], int start_index, int end_index, int element)
{
	if (end_index >= start_index)
	{
		int middle = start_index + (end_index - start_index )/2;
		if (array[middle] == element)
			return middle;
		if (array[middle] > element)
			return BinarySearch(array, start_index, middle-1, element);
		return BinarySearch(array, middle+1, end_index, element);
	}
	return -1;
}
int main(void)
{
	int array[20], n, element,i=0;
	printf("Enter the number of elements ");
	scanf("%d",&n);
	printf("Enter elements in sorted order");
	for(i=0;i<n;i++)
	{
		scanf("%d",&array[i]);
	}
	printf("Enter key value");
	scanf("%d",&element);
	int found = BinarySearch(array, 0, n, element);
	if(found == -1 )
	{
		printf("Element not found in the array ");
	}
	else
	{
		printf("Element found at index : %d",found+1);
```

```
        }
        return 0;
}
```

Output:

```
Enter the number of elements 5
Enter elements in sorted order 1 2 3 4 5
Enter key value 3
Element found at index: 3

Enter the number of elements 5
Enter elements in sorted order 1 2 3 4 5
Enter key value 6
Element not found in the array
```

Program – 2

1) Write a C/C++ program called Breadth first search (BFS). Print all the nodes reachable from a given starting node in a digraph using BFS method.

Code:

```
#include<stdio.h>
int I,j,n,f=0,r=0,a[10][10],q[10],visited[10];
void bfs(int u)
{
        int v;
        visited[u]=1;
        q[r]=u;
        while(f<=r)
        {
                u=q[f++];
                for(v=1;v<=n;v++)
                {
                     if(a[u][v]==1 && visited[v]==0)
                        {
                                Visited[v]=1;
                                q[++r]=v;
                        }
                }
        }
}
void main()
{
        int source;
        printf("Enter the number of vertices\n");
        scanf("%d",&n);
        printf("Enter the adjacency matrix of the directed graph\n");
        for(i=1;i<=n;i++)
                for(j=1;j<=n;j++)
                        scanf("%d",&a[i][j]);
        printf("Enter the source vertex\n");
        scanf("%d",&source);
        for(i=1;i<=n;i++)
                visited[i]=0;
```

```
        bfs(source);
        printf("From vertex %d the vertices\n",source);
        for(i=1;i<=n;i++)
        {
                if(visited[i]==1)
                printf("%d is visited \n",i);
        }
}
```

Output:

```
Enter the number of nodes:
4
Enter the adjacency matrix
0 1 1 0
0 0 0 1
0 0 0 0
0 0 0 0
Enter the source
1
1 is visited
2 is visited
3 is visited
4 is visited
```

b) Write a C/C++ program called Depth first search (DFS). Print all the nodes Reachable from a given starting node in a digraph using DFS method.

Code:

```
#include<stdio.h>
int n,a[10][10],visited[10];
void dfs(int u)
{
        int v;
        visited[u]=1;
        for(v=1;v<=n;v++)
                if(a[u][v]==1&& visited[v]==0)
                dfs(v);
}
void main()
```

```
{
        int I,j,source;
        printf("Enter the number of vertices of the graph:");
        scanf("%d",&n);
        printf("Enter the adjacency matrix.....\n");
        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;
        for(i=1;i<=n;i++)
        {
                dfs(i);
                for(j=1;j<n;j++)
                {
                        if(visited[j]!=1)
                        {
                                printf("%d is not reachable from %d\n",j,i);
                                printf("So graph is not connected");
                                exit(0);
                        }
                }
                for(j=1;j<=n;j++)
                        visited [j]=0;
        }
        printf("\n\n\n");
        printf("Graph is connected\n");
}
```

Output:

```
Enter the number of vertices
3
Enter the cost adjacency matrix
0   1   1
1   0   0
1   0   0
Enter the source of matrix
1
Graph is connected
```

Program - 3

Design, develop, and execute a program called MERGE_SORT to sort a given set of elements using the merge sort method and determine the time required to sort the elements. Repeat the experiment for different values of n, the number of elements in the list to be sorted and plot a graph of the time taken versus n. The elements should be read from a file and also be generated using the random number generator.

Code:

```
#include<stdio.h>
#include<time.h>
#define MAXSIZE 30000
#define NTIMES 5000
void merge(int a[],int low,int mid,int high)
{
        int i,j,k,p;
        int b[MAXSIZE];
        i=low;
        j=mid+1;
        k=low;
        while(i<=mid&&j<=high)
        {
                if(a[i]<=a[j])
                {
                        b[k]=a[i];
                        i=i+1;
                }
                else
                {
                        b[k]=a[j];
                        j=j+1;
                }
                k=k+1;
        }
        while(i<=mid)
        {
                b[k]=a[i];
                i=i+1;
```

```
                k=k+1;
        }
        while(j<=high)
        {
                b[k]=a[j];
                j=j+1;
                k=k+1;
        }
        for(i=low;i<=high;i++)
        {
                a[i]=b[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);
        }
}
int main()
{
        int a[MAXSIZE],j,i,n,k;
        double runtime=0;
        clock_t start,end;
        printf("Enter the value of n\n");
        scanf("%d",&n);
        for(k=1;k<=NTIMES;k++)
        {
                srand(1);
                for(i=1;i<=n;i++)
                        a[i]=rand();
                start=clock();
                mergesort(a,1,n);
                end=clock();
                runtime=runtime+((end-start)/CLK_TCK);
```

```
        }
        runtime=runtime/NTIMES;
        printf("\nSorted elements are\n");
        for(i=1;i<=n;i++)
                printf("%d\n",a[i]);
        printf("Time taken for sorting is %lf seconds",runtime);
}
```

Output:

```
        Enter the value of n 500
        Time taken for sorting is 000582 seconds
```

Program - 4

Design, develop, and execute a program called QUICK_SORT to sort a given set of elements using the Quick sort method and determine the time required to Sort the elements. Repeat the experiment for different values of n, the number of elements in the list to be sorted and plot a graph of the time taken versus n. The elements should be read from a file and also be generated using the random number generator.

Code:

```
#include<stdio.h>
#include<time.h>
#include<stdlib.h>
#define MAXSIZE 30000
#define NTIMES 5000
int partition (int a[], int low,int high)
{
        int p,i,j,temp;
        p=a[low];
        i=low+1;
        j=high;

        while(1)
        {

                while((a[i]<=p)&&(i<high))
                        i++;
                while(( a[j]>p)&&(j>=low))
                        j--;
                if(i<j)
                {
                        temp=a[i];
                        a[i]=a[j];
                        a[j]=temp;
                }
                else
                {
                        temp=a[low];
                        a[low]=a[j];
                        a[j]=temp;
                        return j;
                }
        }
}
```

```
void quicksort(int a[],int low,int high)
{
        int s;
        if(low<high)
        {
                s=partition(a,low,high);
                quicksort(a,low,s-1);
                quicksort(a,s+1,high);
        }
}
void main()
{
        int a[MAXSIZE],i,n,k;
        clock_t start,end;
        double runtime=0;
        printf("\n Enter the size of array: \n");
        scanf("%d",&n);
        for(k=0;k<NTIMES;k++)
        {
                srand(1);
                for(i=0;i<n;i++)
                a[i]=rand();
                start=clock();
                quicksort(a,0,n-1);
                end=clock();
                runtime=runtime+((end-start)/CLK_TCK);
        }
        runtime=runtime/NTIMES;
        printf("\n Sorted elements are \n");
        for(i=0;i<n;i++)
                printf("%d \n",a[i]);
        printf("Time taken for sorting is %lf seconds",runtime);
}
```

Output:

```
Enter the value of n 500
Time taken for sorting is 8.07340 seconds
```

Program - 5

a) Write a C/C++ program to solve All-Pairs Shortest Paths problem using Floyd's algorithm.

Code:

```
#include<stdio.h>


#include<time.h>
 int min(int a,int b)
{

        return (a<b)?a:b;
}
void floyds(int cost[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++)

                                    cost[i][j]=min(cost[i][j],cost[i][k]+cost[k][j]);
}
int main()
{

         int n,cost[10][10],source,i,j;
         printf("Enter the no of vertices of the graph\n");
         scanf("%d",&n);
         printf("Enter the cost matrix of the graph\n");
         for(i=1;i<=n;i++)
         for(j=1;j<=n;j++)
         {
                  scanf("%d",&cost[i][j]);
         }
         floyds(cost,n);
         for(i=1;i<=n;i++)
```

```
        {
                for(j=1;j<=n;j++)
                {
                        printf("%d\t",cost[i][j]);
                }
                 printf("\n");
        }
}
```

Output:

```
Enter the no of vertices of the graph
4
Enter the cost matrix of the graph
0 999 3 999
2 0 999 999
999 7 0 1
6 999 999 0
The all pairs shortest path is
0       10      3       4
2       0       5       6
7       7       0       1
6       16      9       0
```

b) Write a C/C++ program to find the transitive closure using Warshal's algorithm.

Code:

```
include<stdio.h>
void warshal(int cost[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++)
                                cost[i][j]=(cost[i][j])||(cost[i][k]&&cost[k][j]);
}
void main()
{
        int n,cost[10][10],source,i,j;
        printf("Enter the no of vertices of graph\n");
        scanf("%d",&n);
        printf("Enter adjacency matrix of graph\n");
```

```
        for(i=1;i<=n;i++)
                for(j=1;j<=n;j++)
                        scanf("%d",&cost[i][j]);
        warshal(cost,n);
        printf("The path matrix is:\n");
        for(i=1;i<=n;i++)
                for(j=1;j<=n;j++)
                        printf("%d\t",cost[i][j]);
                printf("\n");
}
```

Output:

```
Enter the no of vertices of graph
4
Enter adjacency matrix of graph
0 1 0 0
0 0 0 1
0 0 0 0
1 0 1 0
the path matrix is:
1 1 1 1
1 1 1 1
0 0 0 0
1 1 1 1
```

Program -6

Devise and code a C/C++ program aimed at sorting an array of elements through the use of Heap Sort.

Code:

```
#include <stdio.h>
void swap(int* a, int* b)
{
	int temp = *a;
	*a = *b;
	*b = temp;
}
void heapify(int arr[], int N, int i)
{
	int largest = i;
	int left = 2 * i + 1;
	int right = 2 * i + 2;
	if (left < N && arr[left] > arr[largest])
	largest = left;
	if(right < N && arr[right] > arr[largest])
	largest = right;
	if(largest != i)
	{
		swap(&arr[i], &arr[largest]);
		heapify(arr, N, largest);
	}
}
void heapSort(int arr[], int N)
{
	for (int i = N / 2 - 1; i >= 0; i--)
		heapify(arr, N, i);
	for (int i = N - 1; i >= 0; i--)
	{
		swap(&arr[0], &arr[i]);
		heapify(arr, i, 0);
	}
}
void printArray(int arr[], int N)
{
	for (int i = 0; i < N; i++)
```

```
                printf("%d ", arr[i]);
        printf("\n");
}
int main()
{
        int arr[1000], N,n,i;
        printf("Enter the number of elements ");
        scanf("%d",&n);
        printf("Enter elements ");
        for(i=0;i<n;i++)
        {
                scanf("%d",&arr[i]);
        }
        heapSort(arr, n);
        printf("Sorted array is\n");
        printArray(arr, n);
}
```

Output:

```
Enter the number of elements 10
Enter elements 2 4 3 1 0 9 6 7 8 5
Sorted array is
0 1 2 3 4 5 6 7 8 9
```

Program – 7

Design and implement a C/C++ program called KNAPSACK to Implement 0/1 Knapsack problem using Dynamic Programming.

Code:

```
#include<stdio.h>
int max(int a,int b)
{
        if(a>b)
        return a;
        else
       return b;
}
void main()
{
        int n,j,i,capacity;
        int weight[20],value[20];
         int v[50][50],w;
        printf("\n\n\t\tEnter the number of items:");
        scanf("%d",&n);
        printf("\t\t--------------------------------\n");
        printf("\n\t\t Enter : WEIGHTS -VALUES \n");
        printf("\t\t--------------------------------\n");
        printf("\t\t--------------------------------\n\t\t");
        for(i=1;i<=n;i++)
        {
                scanf("%d",&weight[i]);
                scanf("%d",&value[i]);
                printf("\t\t");
        }
        printf("--------------------------------\n");
        printf("\t\tEnter the capacity of KnapSack:");
        scanf("%d",&capacity);
        printf("\t\t--------------------------------\n\t\t");
        for(i=0;i<=n;i++)
        {
                for(j=0;j<=capacity;j++)
                {
                        if(i==0||j==0)
                        {
                                v[i][j]=0;
                        }
                        else if(j-weight[i]>=0)
                        {
```

```
                                        v[i][j]=max(v[i-1][j],v[i-1][j-weight[i]]+value[i]);
                                }


                                else
                                  {
                                        v[i][j]=v[i-1][j];
                                  }
                           printf("%4d",v[i][j]);
                          }
                printf("\n\t\t");
                }
                w=capacity;
                printf("The items in the knapsack :\n");

                for(i=n;i>0;i--)
                        {
                                printf("\t\t");

                                 if(v[i][w]==v[i-1][w])
                                        continue;
                                 else
                                 {
                                        w=w-weight[i];
                                        printf("%3d",i);
                                 }
                        }
                printf("\n\t\t Total Profit=%d",v[n][capacity]);
        }
```

Output:

```
enter  the no of items  4
enter the wt of 4 item:  2    1     3     2
enter the values(profit) : 12   10    20    15
enter the knapsack capacity  5
the o/p is
 0        0        0        0        0        0
 0        0       12       12       12       12
 0       10       12       22       22       22
 0       10       12       22       30       32
 0       10       15       25       30       37
 the o.s is37
 the vaidity of items r:
 item:validity:weight:profit:
[1]:  1 : 2 :12 :
[2]:  1 : 1 :10 :
[3]:  0 : 3 :20 :
[4]:  1 : 2 :15 :
```

Program – 8

Design and implement a C/C++ program to find Minimum Cost Spanning Tree of a given connected undirected graph using Prims's algorithm.

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <limits.h>

#define V 5

int minKey(int key[], bool mstSet[]) {
   int min = INT_MAX, min_index;
   for (int v = 0; v < V; v++) {
      if (mstSet[v] == false && key[v] < min) {
         min = key[v];
         min_index = v;
      }
   }
   return min_index;
}

void printMST(int parent[], int graph[V][V]) {
   printf("Edge   Weight\n");
   for (int i = 1; i < V; i++)
      printf("%d - %d    %d \n", parent[i], i, graph[i][parent[i]]);
}

void primMST(int graph[V][V]) {
   int parent[V];
   int key[V];
   bool mstSet[V];
   for (int i = 0; i < V; i++) {
      key[i] = INT_MAX;
      mstSet[i] = false;
   }
   key[0] = 0;
   parent[0] = -1;
   for (int count = 0; count < V - 1; count++) {
      int u = minKey(key, mstSet);
      mstSet[u] = true;
      for (int v = 0; v < V; v++) {
         if (graph[u][v] && mstSet[v] == false && graph[u][v] < key[v]) {
```

```
                parent[v] = u;
                key[v] = graph[u][v];
            }
        }
    }
    printMST(parent, graph);
}

int main() {
    int graph[V][V] = {{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}};
    primMST(graph);
    return 0;
}
```

Output:

```
Edge   Weight
0 - 1    2
1 - 2    3
0 - 3    6
1 - 4    5
```

Program – 9

Investigate the time complexity of Kruskal's algorithm by varying the number of vertices and edges in a graph. Plot the results and discuss the findings.

Code:

```
#include<stdio.h>
int parent[10];
void main()
{
        int mincost=0,cost[10][10],n,i,j,ne,a,b,min,u,v;
        printf("Enter the number of vertices of the graph \n");
        scanf("%d",&n);
        printf("Enter the cost 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;
                   }
                   ne=1;
                   printf("\nThe edges of minimum spanning tree are: \n");
                   while(ne<n)
                   {
                        for(min=999,i=1;i<=n;i++)
                        for(j=1;j<=n;j++)
                        if(cost[i][j]<min)
                        {
                                min=cost[i][j];
                                a=u=i;
                                b=v=j;
                        }
                        while(parent[u])
                                u=parent[u];
                        while(parent[v])
                                v=parent[v];
                        if(v!=u)
                        {
                                printf("%d edge(%d,%d)=%d\n",ne++,a,b,min);
                                mincost+=min;
                                parent[b]=a;
                        }
                        cost[a][b]=cost[b][a]=999;
                   }
        printf("\nThe minimum cost of spanning tree is %d\n",mincost);
}
```

Output:

```
enter the no of vertices:
4
enter t cost of matrix:
0 10 30 40
10 0 20 50
30 20 0 60
40 50 60 0
the edge of spanning tree are:
1)edge(1,2)=10
2)edge(2,3)=20
3)edge(1,4)=40

 mincost=70
```

Program – 10

Design and develop a C/C++ program for N Queen's problem using Backtracking.

Code:

```
#include<stdio.h>
int x[10];
void nqueens(int,int);
void main()
{
int n,i,j;
printf("\nEnter the no of queens\n");
scanf("%d",&n);
if(n==0||n==2||n==3)
{
printf("No solutions");
}
else
nqueens(1,n);
}
int place(int k,int i)
{
int j;
for(j=1;j<k;j++)
if(x[j]==i||(abs(x[j]-i)==abs(j-k)))
return 0;
return 1;
}
void nqueens(int k,int n)
{
int i,j;
for(i=1;i<=n;i++)
if(place(k,i))
{
x[k]=i;
if(k==n)
{
printf("\n");
printf("Solution is:");
for(j=1;j<=n;j++)
printf("%2d",x[j]);
```

```
printf("\n");
for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)
{
if(x[i]==j)
printf("Q%d\t",i);
else
printf("*\t");
}
printf("\n");
printf("\n");
}
}
else
nqueens(k+1,n);
}
}
```

Output:

Enter the no of queens
4
Solution is: 2 4 1 3

*	**Q1**	*	*
*	*	*	**Q2**
Q3	*	*	*
*	*	**Q4**	*

Solution is: 3 1 4 2

*	*	**Q1**	*
Q2	*	*	*
*	*	*	**Q3**
*	**Q4**	*	*