Design and Analysis of Algorithms

PRACTICAL FILE

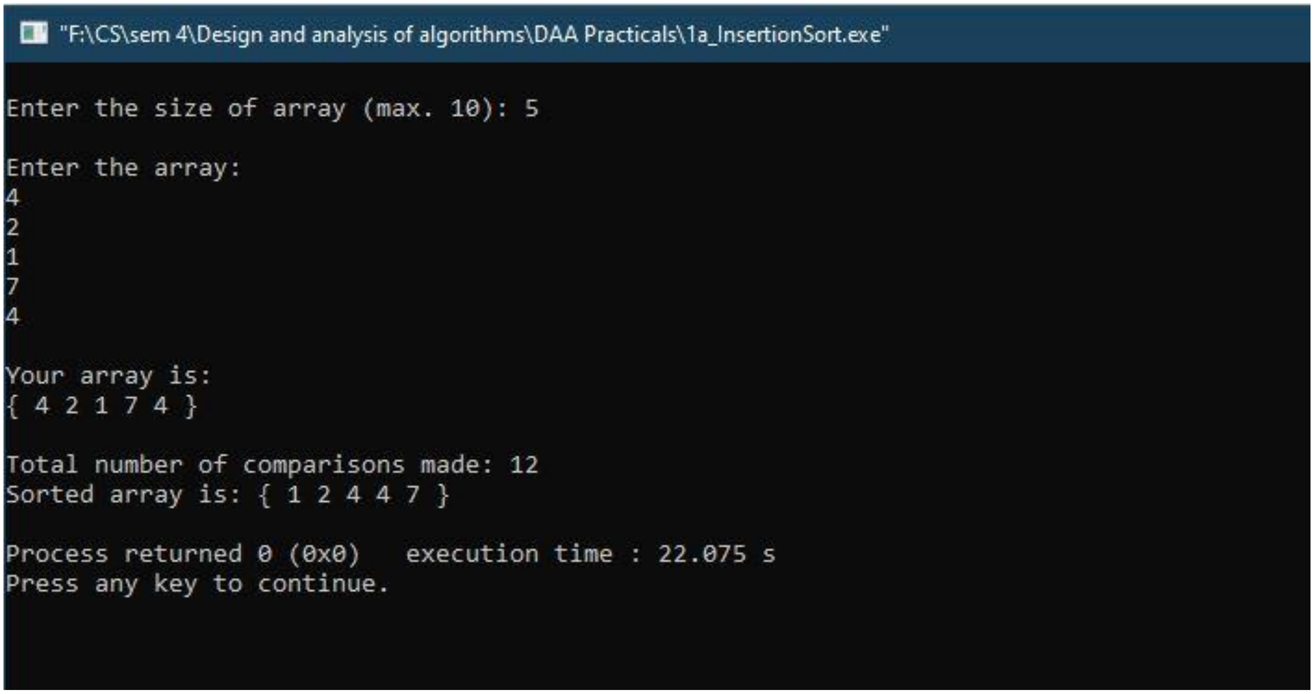
Submitted by,
Divya A.
College Roll No: 20201457
Exam Roll No: 20016570005
BSc (Hons) Computer Science
Ramanujan College

1. i. Implement Insertion Sort (The program should report the Number of comparisons)

INPUT

```
#include<iostream>
using namespace std;

void display(int *a, int size) {
    cout<<"{ ";
    for(int i=0; i<size; i++ )
     cout<<a[i]<<' ';
    cout<<"}"<<endl;
}

int insertionSort(int *a, int n)
{
  int i, j, k, comparison = 0;
  for(i=1; i<n; i++)
  {
    comparison++;
    k = a[i];
    for(j=i-1; (a[j]>k) && (j>=0); j--)
    {
        comparison += 2;
        a[j+1] = a[j];
    }
    a[j+1] = k;
  }
  return comparison;
}

int main()
{

 int size, i, *arr;

 cout<<"\nEnter the size of array (max. 10): ";
 cin>>size;
 arr = new int[size];
 cout<<"\nEnter the array: \n";
 for(i=0; i<size; i++)
   cin>>arr[i];

 cout<<"\nYour array is: \n";
 display(arr, size);

 cout<<"\nTotal number of comparisons made:
"<<insertionSort(arr, size);

 cout<<"\nSorted array is: ";
 display(arr, size);


}
```

OUTPUT



"F:\CS\sem 4\Design and analysis of algorithms\DAA Practicals\1a_InsertionSort.exe"

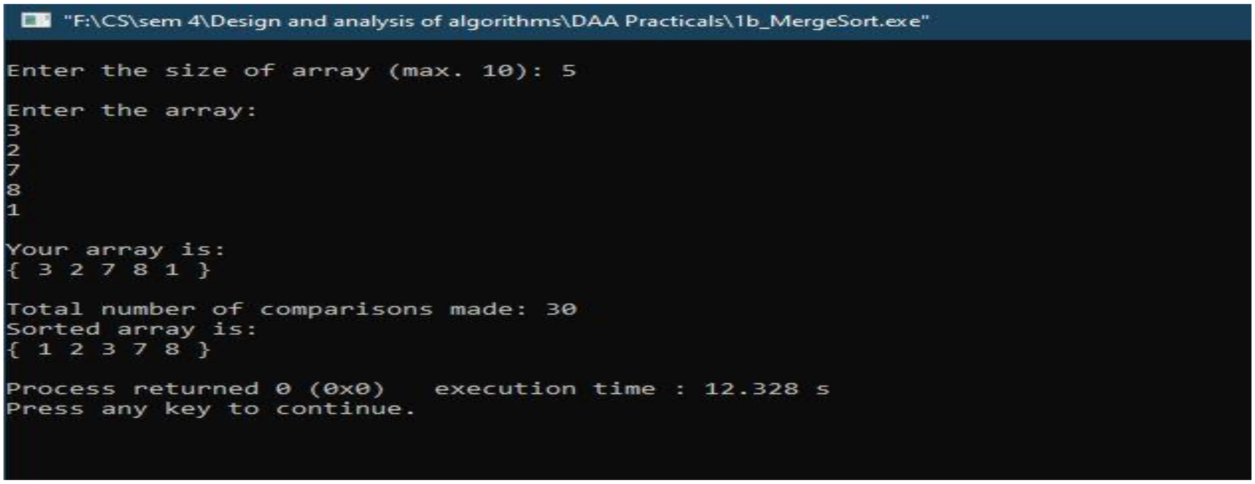
```
Enter the size of array (max. 10): 5

Enter the array:
4
2
1
7
4

Your array is:
{ 4 2 1 7 4 }

Total number of comparisons made: 12
Sorted array is: { 1 2 4 4 7 }

Process returned 0 (0x0)   execution time : 22.075 s
Press any key to continue.
```

1. ii. Implement Merge Sort (The program should report the number of comparisons)

INPUT

```
#include<iostream>
using namespace std;


int comparison = 0;
void display(int *a, int size) {
    cout<<"{ ";
    for(int i=0; i<size; i++ )
     cout<<a[i]<<' ';
    cout<<"}"<<endl;
}

void merge(int *a, int beg, int mid, int end)
{
  int size = end - beg + 1;
  int *temp = new int[size];
  int i=beg, j=mid+1, k=0;

  //arranging in order
  while (i <= mid && j<=end) {
    temp[k++] = (a[i]<a[j]) ? a[i++] : a[j++];
    comparison += 3;
  }
  while(i<=mid){
    comparison++;
    temp[k++] = a[i++];
  }
  while(j<end){
    comparison++;
    temp[k++] = a[j++];
  }

  for(i=0; i<k; i++)
  {
    a[i+beg] = temp[i];
  }
}

void mergeSort(int* a,int beg, int end)
{
    if (beg < end) {
     comparison++;
        int mid = (beg+end)/2;
        mergeSort(a, beg, mid);
        mergeSort(a, mid+1, end);
        merge(a, beg, mid, end);
    }
}

int main()
{

 int size, i, *arr;

 cout<<"\nEnter the size of array (max. 10): ";
 cin>>size;
 arr = new int[size];
 cout<<"\nEnter the array: \n";
 for(i=0; i<size; i++)
   cin>>arr[i];


 cout<<"\nYour array is: \n";
 display(arr, size);

 mergeSort(arr, 0, size-1);
 cout<<"\nTotal number of comparisons made: "<<comparison;

 cout<<"\nSorted array is: \n";
 display(arr, size);

}
```

OUTPUT


"F:\CS\sem 4\Design and analysis of algorithms\DAA Practicals\1b_MergeSort.exe"

```

Enter the size of array (max. 10): 5

Enter the array:
3
2
7
8
1

Your array is:
{ 3 2 7 8 1 }

Total number of comparisons made: 30
Sorted array is:
{ 1 2 3 7 8 }

Process returned 0 (0x0)   execution time : 12.328 s
Press any key to continue.
```

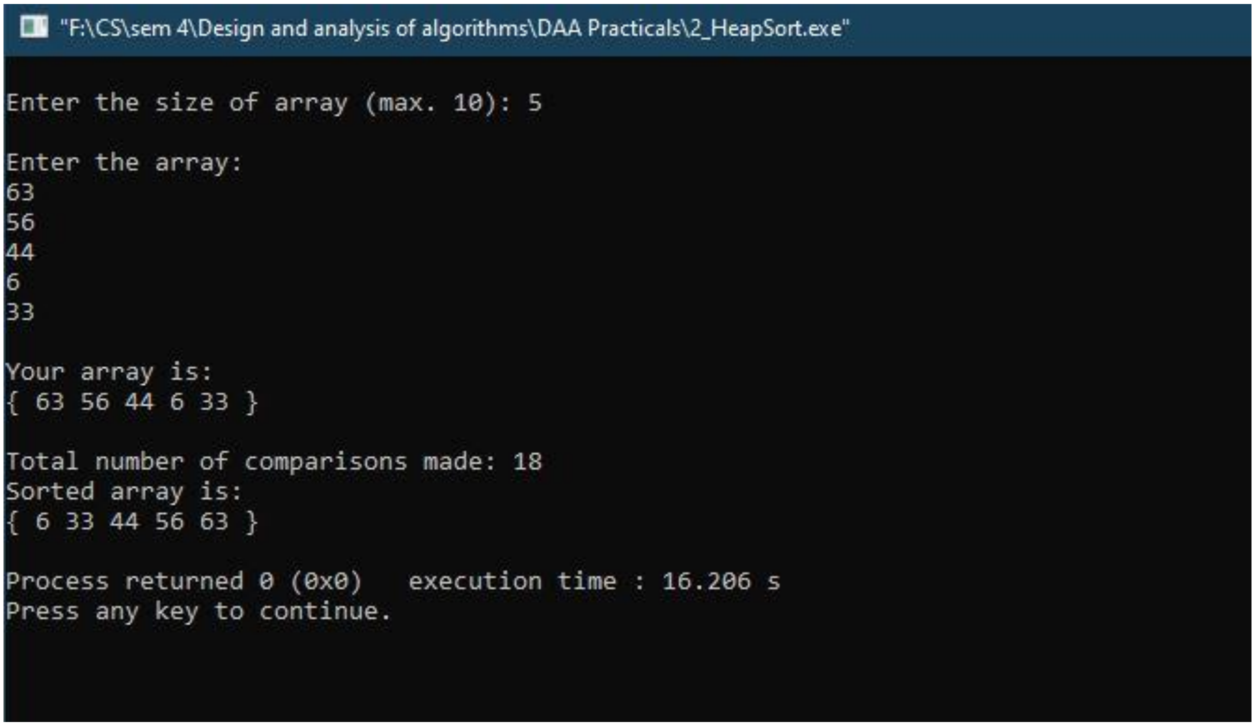
2. Implement Heap Sort(The program should report the number of comparisons)

INPUT

```
#include <iostream>

using namespace std;

int comparison = 0;

void display(int *a, int size) {
    cout<<"{ ";
    for(int i=0; i<size; i++ )
     cout<<a[i]<<' ';
    cout<<"}"<<endl;
}

void swap(int *a, int x, int y){
  int temp = a[y];
  a[y] = a[x];
  a[x] = temp;
}

void maxHeapify(int *a, int index, int heapSize)
{
  int left = index*2 + 1;
  int right = index*2 + 2;
  int largest = index;

  if(left < heapSize && a[left] > a[largest]){
    largest = left;
       comparison+=2;
  }
  if(right < heapSize && a[right] > a[largest]){
    largest = right;
       comparison+=2;
  }

  if(largest != index){
    comparison++;
    swap(a, largest, index);
    maxHeapify(a, largest, heapSize);
  }
}
```

```
void buildMaxHeap(int *a, int n)
{
  for (int i = (n/2) - 1; i >= 0; i--) {
    maxHeapify(a, i, n);
    comparison++;
  }
}

void heapSort(int *a, int size)
{
  buildMaxHeap(a, size);
  int heapSize = size, i;
  for(i=size-1; i>=0; i--) {
     swap(a, 0, i);
     heapSize--;
     comparison++;
     maxHeapify(a,0,heapSize);
  }
}

int main()
{
 int size, i, *arr;
 cout<<"\nEnter the size of array (max. 10): ";
 cin>>size;
 arr = new int[size];

 cout<<"\nEnter the array: \n";
 for(i=0; i<size; i++)
   cin>>arr[i];

 cout<<"\nYour array is: \n";
 display(arr, size);

 heapSort(arr, size);
 cout<<"\nTotal number of comparisons made: "<<comparison;

 cout<<"\nSorted array is: \n";
 display(arr, size);

}
```

OUTPUT



"F:\CS\sem 4\Design and analysis of algorithms\DAA Practicals\2_HeapSort.exe"

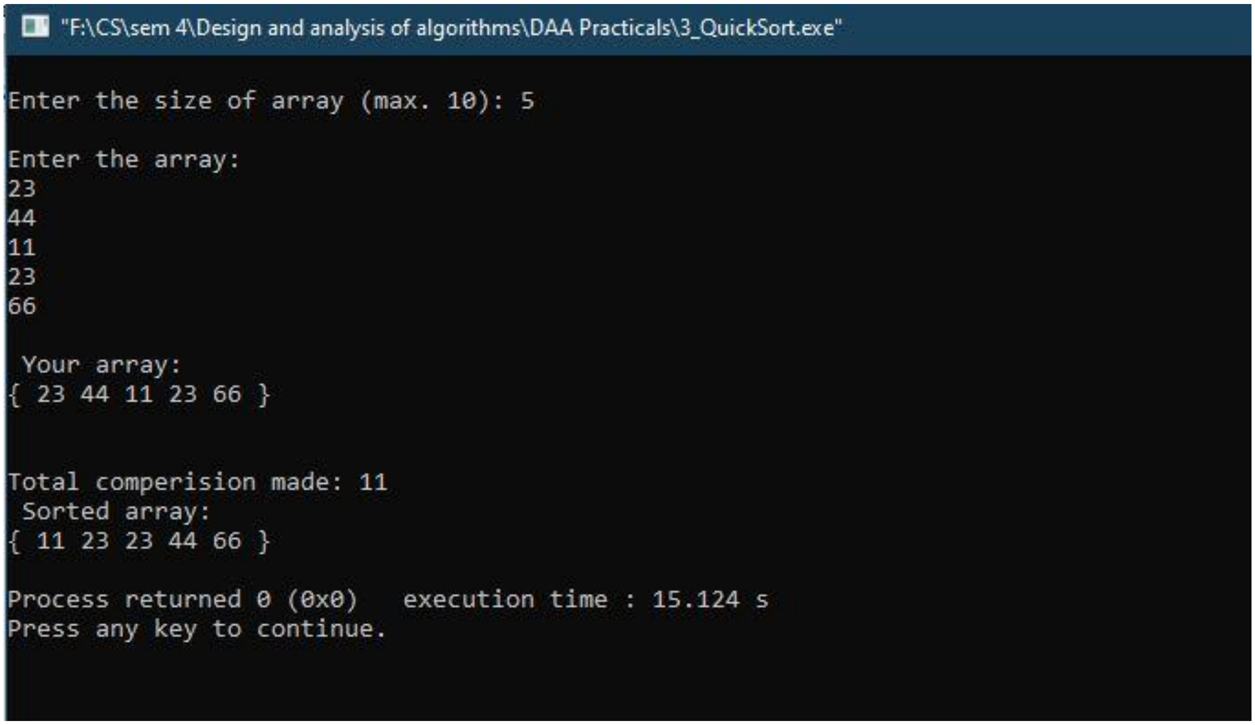
```
Enter the size of array (max. 10): 5

Enter the array:
63
56
44
6
33

Your array is:
{ 63 56 44 6 33 }

Total number of comparisons made: 18
Sorted array is:
{ 6 33 44 56 63 }

Process returned 0 (0x0)   execution time : 16.206 s
Press any key to continue.
```

3. Implement Randomized Quick sort (The program should report the number of comparisons)

INPUT

```
#include <iostream>
#include <stdlib.h>
#include <stdio.h>
using namespace std;

int comparison = 0;

void display(int *a, int size) {
    cout<<"{ ";
    for(int i=0; i<size; i++ )
     cout<<a[i]<<' ';
    cout<<"}"<<endl;
}

void swap(int *a, int x, int y){
  int temp = a[y];
  a[y] = a[x];
  a[x] = temp;
}

int partition(int *a, int p, int r)
{
  int i = p-1, j, x;
  for (j = p; j<r; j++)
    if(a[j] <= a[r]){
      comparison+=2;
      i++;
      swap(a, j, i);
    }
  swap(a, i+1, r);

  return i+1;
}

int randomizedPartition(int *a, int beg, int end)
{
  int t = (rand()%(end-beg)) + beg;
  swap(a, end, t);
  return partition(a, beg, end);
}

void randomizedQuickSort(int *a, int p, int r)
{
  if (p<r) {
    comparison++;
    int q = randomizedPartition(a, p, r);
    randomizedQuickSort(a, p, q-1);
    randomizedQuickSort(a, q+1, r);
  }
}

int main()
{

 int size, i, *arr;
 cout<<"\nEnter the size of array (max. 10): ";
 cin>>size;
 arr = new int[size];

 cout<<"\nEnter the array: \n";
 for(i=0; i<size; i++)
   cin>>arr[i];
 cout<<"\n Your array: \n";
 display(arr, size);


 randomizedQuickSort(arr, 0, size-1);
 cout<<"\n\nTotal comperision made: "<<comparison;

 cout<<"\n Sorted array: \n";
 display(arr, size);

}
```

OUTPUT



"F:\CS\sem 4\Design and analysis of algorithms\DAA Practicals\3_QuickSort.exe"

```
Enter the size of array (max. 10): 5

Enter the array:
23
44
11
23
66

 Your array:
{ 23 44 11 23 66 }


Total comperision made: 11
 Sorted array:
{ 11 23 23 44 66 }

Process returned 0 (0x0)   execution time : 15.124 s
Press any key to continue.
```

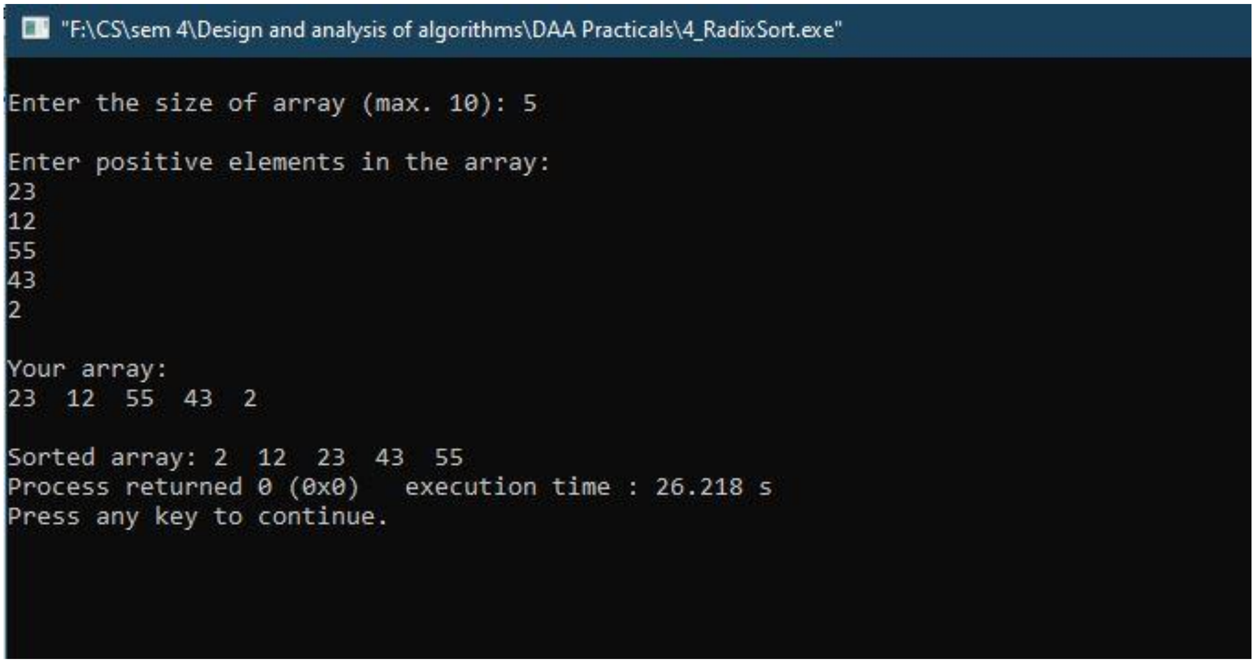
4. Implement Radix Sort

INPUT

```
#include <iostream>
using namespace std;
#include <conio.h>
#include <math.h>


void countSort(int arr[], int size, int num) {
    int x[10];
          for(int c=0; c<10; c++)
            x[c] = 0;

    int *disp = new int[size];
    for(int i=0; i<size; i++)
        x[int(arr[i]/num)%10]++;
    for(int i=1; i<10; i++) {
        x[i] += x[i-1];
    }
    for(int i=0; i<size; i++) {
        disp[x[int(arr[i]/num)%10] - 1] = arr[i];
        x[int(arr[i]/num)%10]--;
    }
    for(int i=0; i<size; i++) {
        arr[i] = disp[i];
    }
}

void radixSort(int arr[], int size) {
    int max = arr[0];

    for(int i=1; i<size; i++)
        if(max<arr[i])
            max = arr[i];

    for(int i=1; max/i>0; i*=10)
        countSort(arr, size, i);
}

 int main()
 {

     int arr[10], size, largest, i;

     cout<<"\nEnter the size of array (max. 10): ";
     cin>>size;
     cout<<"\nEnter positive elements in the array: \n";
     for(i=0; i<size; i++)
          cin>>arr[i];


     cout<<"\nYour array: \n";
     for(i=0; i<size; i++)
     {
          cout<<arr[i]<<"  ";
     }
```

```
      radixSort(arr, size);


      cout<<"\n\nSorted array: ";
      for(i=0; i<size; i++)
      {
           cout<<arr[i]<<"  ";
      }


  }
```

OUTPUT



"F:\CS\sem 4\Design and analysis of algorithms\DAA Practicals\4_RadixSort.exe"

```
Enter the size of array (max. 10): 5

Enter positive elements in the array:
23
12
55
43
2

Your array:
23  12  55  43  2

Sorted array: 2  12  23  43  55
Process returned 0 (0x0)   execution time : 26.218 s
Press any key to continue.
```

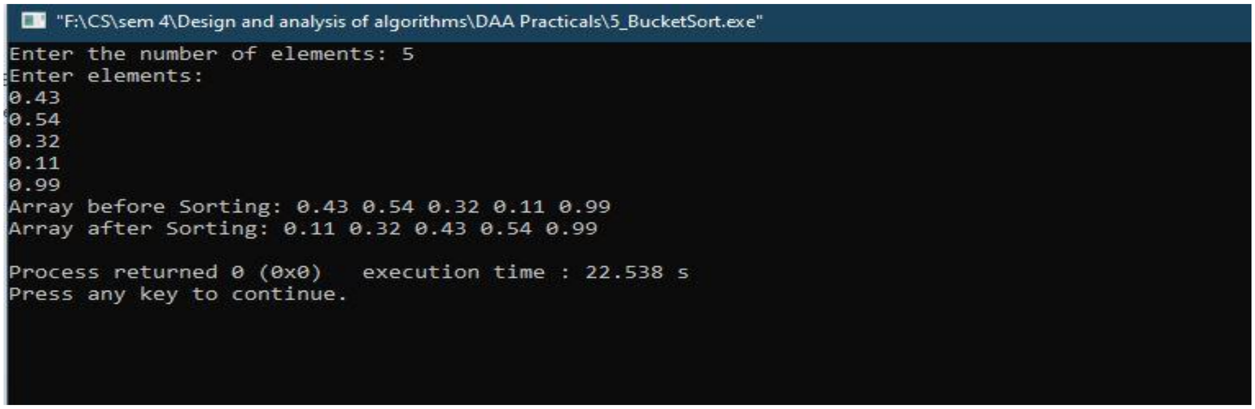
5. Implement Bucket Sort

INPUT

```
#include<iostream>
#include<vector>
#include<algorithm>
using namespace std;
void display(float *array, int size) {
   for(int i = 0; i<size; i++)
      cout << array[i] << " ";
   cout << endl;
}
```

```
void bucketSort(float *array, int size) {
   vector<float> bucket[size];
   for(int i = 0; i<size; i++)  {          //put elements into
different buckets
      bucket[int(size*array[i])].push_back(array[i]);
   }
   for(int i = 0; i<size; i++) {
      sort(bucket[i].begin(), bucket[i].end());      //sort
individual vectors
   }
   int index = 0;
   for(int i = 0; i<size; i++) {
      while(!bucket[i].empty()) {
         array[index++] = *(bucket[i].begin());
         bucket[i].erase(bucket[i].begin());
      }
   }
}
int main() {
   int n;
   cout << "Enter the number of elements: ";
   cin >> n;
   float arr[n];     //create an array with given number of
elements
   cout << "Enter elements:" << endl;
   for(int i = 0; i<n; i++) {
      cin >> arr[i];
   }
   cout << "Array before Sorting: ";
   display(arr, n);
   bucketSort(arr, n);
   cout << "Array after Sorting: ";
   display(arr, n);
}
```

OUTPUT


"F:\CS\sem 4\Design and analysis of algorithms\DAA Practicals\5_BucketSort.exe"

```
Enter the number of elements: 5
Enter elements:
0.43
0.54
0.32
0.11
0.99
Array before Sorting: 0.43 0.54 0.32 0.11 0.99
Array after Sorting: 0.11 0.32 0.43 0.54 0.99

Process returned 0 (0x0)   execution time : 22.538 s
Press any key to continue.
```

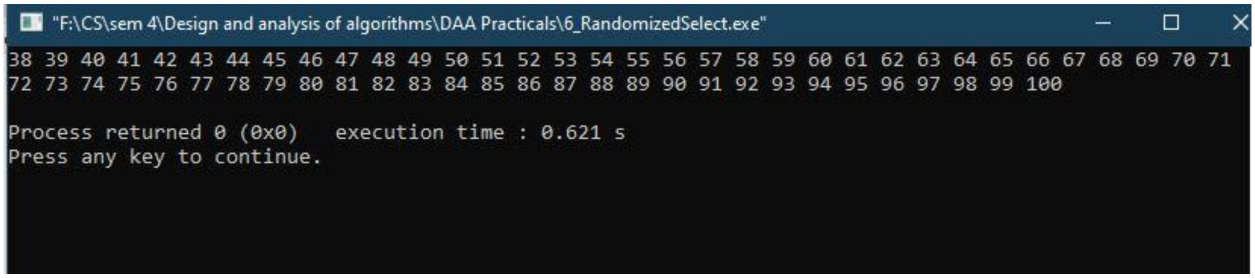
6. Implement Randomized Select

INPUT

```
#include<iostream>
#include<cstdlib>
#include<ctime>
#define MAX 100
using namespace std;
void random_shuffle(int arr[]) {
   //function to shuffle the array elements into random
positions
   srand(time(NULL));
   for (int i = MAX - 1; i > 0; i--) {
      int j = rand()%(i + 1);
      int temp = arr[i];
      arr[i] = arr[j];
      arr[j] = temp;
   }
}
// Partitioning the array on the basis of values at high as
pivot value.
int Partition(int a[], int low, int high) {
   int pivot, index, i;
   index = low;
   pivot = high;
   for(i=low; i < high; i++) {
      // finding index of pivot.
      if(a[i] < a[pivot]) {
         swap(a[i], a[index]);
         index++;
      }
   }
   swap(a[pivot], a[index]);
   return index;
}
int RandomPivotPartition(int a[], int low, int high){
   // Random selection of pivot.
   int pvt, n, temp;
   n = rand();
   pvt = low + n%(high-low+1); // Randomizing the pivot value
from sub-array.
   swap(a[high], a[pvt]);
   return Partition(a, low, high);
}
```

```
void quick_sort(int arr[], int p, int q) {
   //recursively sort the list
   int pindex;
   if(p < q) {
      pindex = RandomPivotPartition(arr, p, q); //randomly
choose pivot
      // Recursively implementing QuickSort.
      quick_sort(arr, p, pindex-1);
    quick_sort(arr, pindex+1, q);
}
}
int main() {
int i;
int arr[MAX];
for (i = 0;i < MAX; i++)
arr[i] = i + 1;
random_shuffle(arr); //To randomize the array
quick_sort(arr, 0, MAX - 1); //sort the elements of array
for (i = 0; i < MAX; i++)
cout << arr[i] << " ";
cout << endl;
return 0;
}
```

OUTPUT


"F:\CS\sem 4\Design and analysis of algorithms\DAA Practicals\6_RandomizedSelect.exe"

```
38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71
72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

Process returned 0 (0x0)   execution time : 0.621 s
Press any key to continue.
```

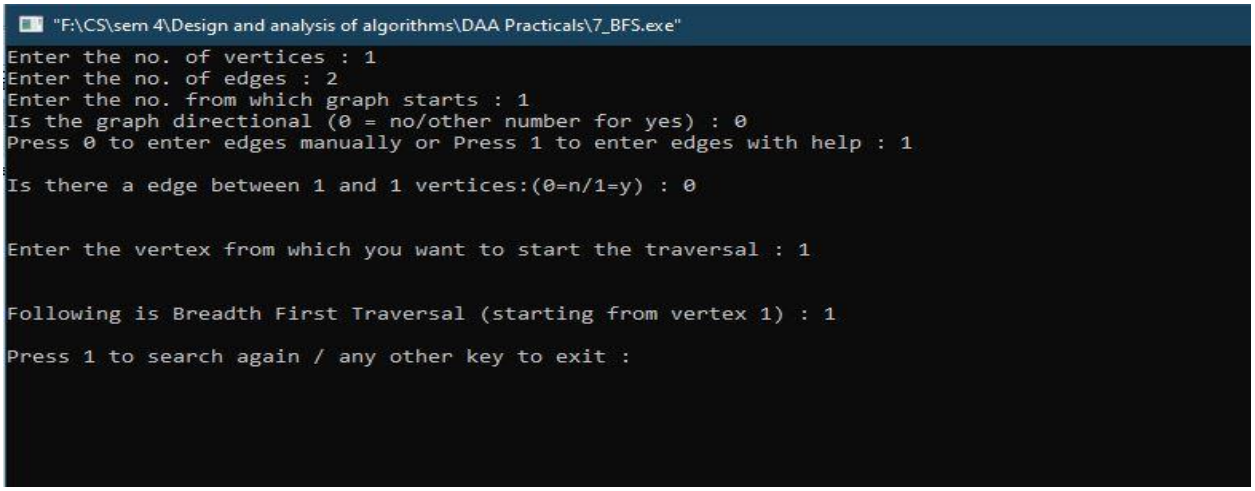
7. Implement Breadth-First Search in a graph

```
#include<iostream>
#include <list>
using namespace std;
class Graph
{
    int V;
    list<int> *adj;
public:
    Graph(int V);
    void addEdge(int v, int w);
    void BFS(int s,int strt);
```

```
};

Graph::Graph(int V)
{
    this->V = V;
    adj = new list<int>[V];
}
void Graph::addEdge(int v, int w)
{
    adj[v].push_back(w);
}
void Graph::BFS(int s,int strt)
{
    bool *visited = new bool[V];
    for(int i = 0; i < V; i++)
        visited[i] = false;
    list<int> queue;
    visited[s] = true;
    queue.push_back(s);
    list<int>::iterator i;
    while(!queue.empty())
    {
        s = queue.front();
        cout << s + strt << " ";
        queue.pop_front();

        for (i = adj[s].begin(); i != adj[s].end(); ++i)
        {
            if (!visited[*i])
            {
                visited[*i] = true;
                queue.push_back(*i);
            }
        }
    }
}
int main()
{
    int nv,ne,start,dir,help,ep,stf,ext;
    begin:
    cout<<"Enter the no. of vertices : ";
    cin>>nv;
    cout<<"Enter the no. of edges : ";
    cin>>ne;

    if(nv<0||ne<0){
            cout<<"\n\nError : no. of vertices or edges cannot
be zero.\n\n";
            goto begin;
    }
    cout<<"Enter the no. from which graph starts : ";
    cin>>start;
    cout<<"Is the graph directional (0 = no/other number for
yes) : ";
    cin>>dir;
    Graph g(nv);
    cout<<"Press 0 to enter edges manually or Press 1 to enter
edges with help : ";
    cin>>help;
    cout<<endl;
    if(help==1){
if(dir==0){
        for(int i = 0;i<nv;i++){
            for(int j = i;j<nv;j++){
                cout<<"Is there a edge between "<<i+start<<" and
"<<j+start<<" vertices:(0=n/1=y) : ";
                cin>>ep;
                if(ep==1){
                        g.addEdge(i,j);
                        g.addEdge(j,i);
                        }
                    }
                }
            }
            else{
                    for(int i = 0;i<nv;i++){
                    for(int j = 0;j<nv;j++){
                        cout<<"Is there a edge between
"<<i+start<<" and "<<j+start<<" vertices:(0=n/1=y) : ";
                        cin>>ep;
                        if(ep==1){
                                g.addEdge(i,j);
                        }
                    }
                }
            }
        }
        else if(help ==0){
                if(dir==0){
                    while(ne!=0){
                        int i,j;
        cout<<"Enter both vertices of an edge with a space
between them : ";
```

```
        cin>>i>>j;
        g.addEdge(i-start,j-start);
        g.addEdge(j-start,i-start);
        ne--;
        }
    }
    else{
            while(ne!=0){
                int i,j;
                cout<<"Enter starting and ending vertices of an
edge with a space between them :";
                cin>>i>>j;
                g.addEdge(i-start,j-start);
                ne--;
                }
            }
        }
        cout<<"\n\nEnter the vertex from which you want to start
the traversal : ";
        cin>>stf;
        cout << "\n\nFollowing is Breadth First Traversal "<<
"(starting from vertex "<<stf<<") : ";
        g.BFS(stf-start,start);
        cout<<"\n\nPress 1 to search again / any other key to
exit : ";
        cin>>ext;
        if(ext == 1)
            goto begin;
        return 0;
}
```

OUTPUT



"F:\CS\sem 4\Design and analysis of algorithms\DAA Practicals\7_BFS.exe"

```
Enter the no. of vertices : 1
Enter the no. of edges : 2
Enter the no. from which graph starts : 1
Is the graph directional (0 = no/other number for yes) : 0
Press 0 to enter edges manually or Press 1 to enter edges with help : 1

Is there a edge between 1 and 1 vertices:(0=n/1=y) : 0


Enter the vertex from which you want to start the traversal : 1


Following is Breadth First Traversal (starting from vertex 1) : 1

Press 1 to search again / any other key to exit :
```

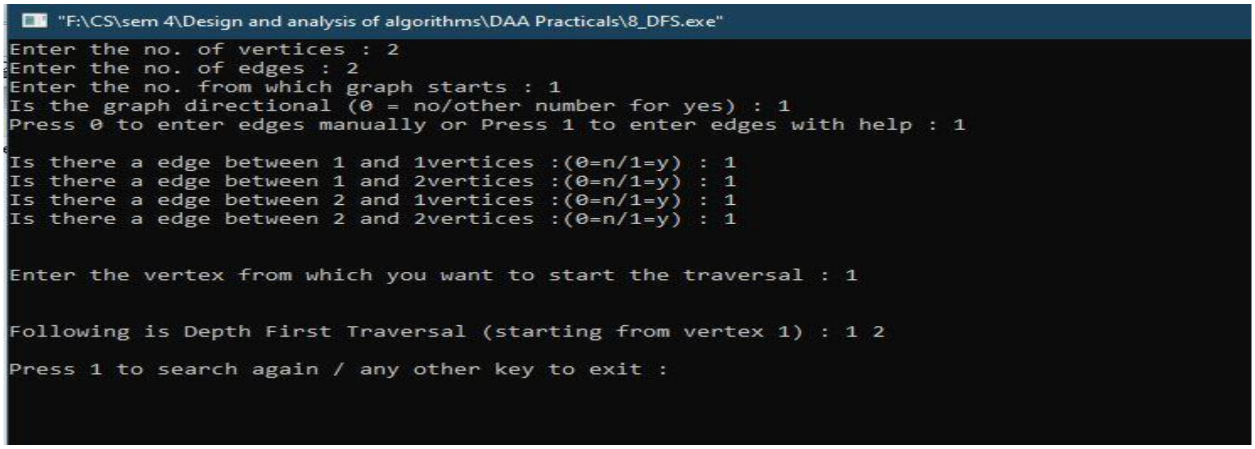
8. Implement Depth-First Search in a graph

INPUT

```
#include<iostream>
#include <list>
using namespace std;
int start;
class Graph {
    int numVertices;
    list<int> *adjLists;
    bool *visited;
    public:
        Graph(int V);
        void addEdge(int src, int dest);
        void DFS(int vertex);
    };
    Graph::Graph(int vertices) {
        numVertices = vertices;
        adjLists = new list<int>[vertices];
        visited = new bool[vertices];
    }
    void Graph::addEdge(int src, int dest) {
        adjLists[src].push_front(dest);
    }
    void Graph::DFS(int vertex) {
        visited[vertex] = true;
        list<int> adjList = adjLists[vertex];
        cout << vertex +start<< " ";
        list<int>::iterator i;
        for (i = adjList.begin(); i != adjList.end(); ++i)
            if (!visited[*i])
            DFS(*i);
    }
    int main()
    {
        int nv,ne,dir,help,ep,stf,ext;
        begin:
            cout<<"Enter the no. of vertices : ";
            cin>>nv;
            cout<<"Enter the no. of edges : ";
            cin>>ne;
            if(nv<0||ne<0){
                    cout<<"\n\nError : no. of vertices or edges
cannot be zero.\n\n";
```

```
            goto begin;
            }
            cout<<"Enter the no. from which graph starts : ";
            cin>>start;
            cout<<"Is the graph directional (0 = no/other number
for yes) : ";
            cin>>dir;
            Graph g(nv);
            cout<<"Press 0 to enter edges manually or Press 1 to
enter edges with help : ";
            cin>>help;
            cout<<endl;
            if(help==1){
                    if(dir==0){
                        for(int i = 0;i<nv;i++){
                            for(int j = i;j<nv;j++){
                                cout<<"Is there a edge between
"<<i+start<<" and "<<j+start<<"vertices :(0=n/1=y) : ";
                                cin>>ep;
                                if(ep==1){
                                        g.addEdge(i,j);
                                        g.addEdge(j,i);
                                        }
                                    }
                                }
                            }
                        else{
                                for(int i = 0;i<nv;i++){
                                    for(int j = 0;j<nv;j++){
                                        cout<<"Is there a edge
between "<<i+start<<" and "<<j+start<<"vertices :(0=n/1=y) : ";
                                        cin>>ep;
                                        if(ep==1){
                                                g.addEdge(i,j);
                                        }
                                    }
                                }
                            }
                        }
                        else if(help ==0){
                                if(dir==0){
                                    while(ne!=0){
                                        int i,j;
                                        cout<<"Enter both
vertices of an edge with a space between them : ";
                                        cin>>i>>j;
                                        g.addEdge(i-start,j-
start);
                                        g.addEdge(j-start,i-
start);
                                        ne--;
                                        }
                                }
                                else{
                                        while(ne!=0){
                                            int i,j;
                                            cout<<"Enter
starting and ending vertices of an edge with a space between
them : ";
                                            cin>>i>>j;
                                            g.addEdge(i-start,j-
start);
                                            ne--;
                                        }
                                    }
                                }
                                cout<<"\n\nEnter the vertex from
which you want to start the traversal : ";
                                cin>>stf;
                                cout << "\n\nFollowing is Depth
First Traversal "<< "(starting from vertex "<<stf<<") : ";
                                g.DFS(stf-start);
                                cout<<"\n\nPress 1 to search
again / any other key to exit : ";
                                cin>>ext;
                                if(ext == 1)
                                goto begin;
                                return 0;
                       }
```

OUTPUT



"F:\CS\sem 4\Design and analysis of algorithms\DAA Practicals\8_DFS.exe"

```
Enter the no. of vertices : 2
Enter the no. of edges : 2
Enter the no. from which graph starts : 1
Is the graph directional (0 = no/other number for yes) : 1
Press 0 to enter edges manually or Press 1 to enter edges with help : 1

Is there a edge between 1 and 1vertices :(0=n/1=y) : 1
Is there a edge between 1 and 2vertices :(0=n/1=y) : 1
Is there a edge between 2 and 1vertices :(0=n/1=y) : 1
Is there a edge between 2 and 2vertices :(0=n/1=y) : 1


Enter the vertex from which you want to start the traversal : 1


Following is Depth First Traversal (starting from vertex 1) : 1 2

Press 1 to search again / any other key to exit :
```

9. Write a program to determine the minimum spanning tree of a graph using both Prims and Kruskals algorithm

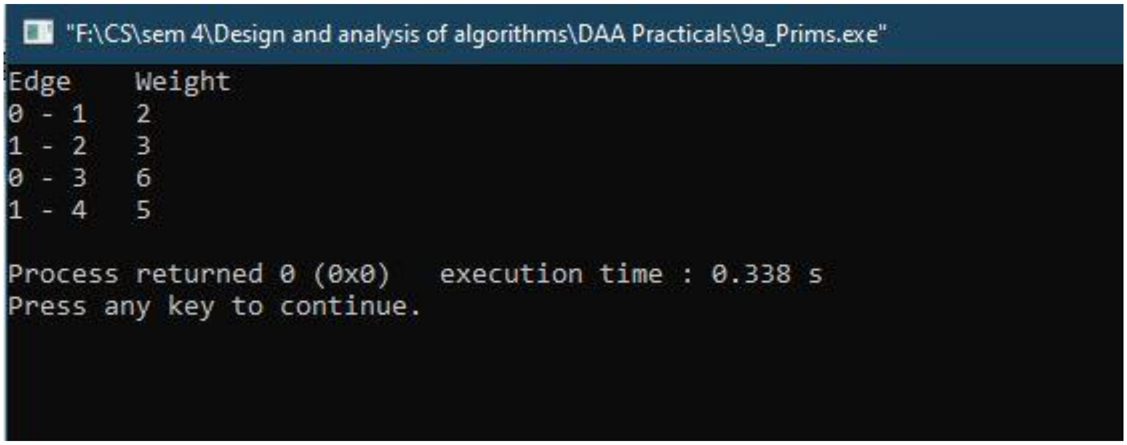
a)

INPUT

```
#include <bits/stdc++.h>
using namespace std;

// Number of vertices in the graph
#define V 5

// A utility function to find the vertex with
// minimum key value, from the set of vertices
// not yet included in MST
int minKey(int key[], bool mstSet[])
{
    // Initialize min value
    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;
}

// A utility function to print the
// constructed MST stored in parent[]
void printMST(int parent[], int graph[V][V])
{
    cout<<"Edge \tWeight\n";
    for (int i = 1; i < V; i++)
        cout<<parent[i]<<" - "<<i<<" \t"<<graph[i][parent[i]]<<"
\n";
}

// Function to construct and print MST for
// a graph represented using adjacency
// matrix representation
void primMST(int graph[V][V])
{
    // Array to store constructed MST
    int parent[V];
    // Key values used to pick minimum weight edge in cut
    int key[V];

    // To represent set of vertices included in MST
    bool mstSet[V];

    // Initialize all keys as INFINITE
    for (int i = 0; i < V; i++)
        key[i] = INT_MAX, mstSet[i] = false;

    // Always include first 1st vertex in MST.
    // Make key 0 so that this vertex is picked as first vertex.
    key[0] = 0;
    parent[0] = -1; // First node is always root of MST

    // The MST will have V vertices
    for (int count = 0; count < V - 1; count++)
    {
        // Pick the minimum key vertex from the
        // set of vertices not yet included in MST
        int u = minKey(key, mstSet);

        // Add the picked vertex to the MST Set
        mstSet[u] = true;

        // Update key value and parent index of
        // the adjacent vertices of the picked vertex.
        // Consider only those vertices which are not
        // yet included in MST
        for (int v = 0; v < V; v++)

            // graph[u][v] is non zero only for adjacent
vertices of m
            // mstSet[v] is false for vertices not yet included
in MST
            // Update the key only if graph[u][v] is smaller
than key[v]
            if (graph[u][v] && mstSet[v] == false && graph[u][v]
< key[v])
                parent[v] = u, key[v] = graph[u][v];
    }

    // print the constructed MST
    printMST(parent, graph);
}
```

```
// Driver code
int main()
{
    /* Let us create the following graph
        2 3
    (0)--(1)--(2)
    | / \ |
    6| 8/ \5 |7
    | / \ |
    (3)-------(4)
            9     */
    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 } };

    // Print the solution
    primMST(graph);

    return 0;
}
```

OUTPUT



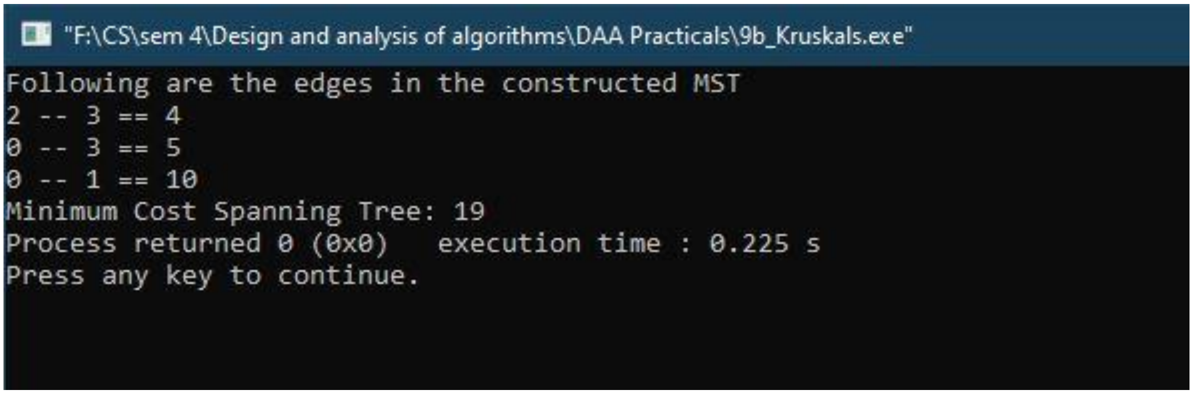
"F:\CS\sem 4\Design and analysis of algorithms\DAA Practicals\9a_Prims.exe"

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

Process returned 0 (0x0)   execution time : 0.338 s
Press any key to continue.
```

b)

```
#include <bits/stdc++.h>
using namespace std;

// Number of vertices in the graph
#define V 5

// A utility function to find the vertex with
// minimum key value, from the set of vertices
// not yet included in MST
int minKey(int key[], bool mstSet[])
{
    // Initialize min value
    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;
}

// A utility function to print the
// constructed MST stored in parent[]
void printMST(int parent[], int graph[V][V])
{
    cout<<"Edge \tWeight\n";
    for (int i = 1; i < V; i++)
        cout<<parent[i]<<" - "<<i<<" \t"<<graph[i][parent[i]]<<"
\n";
}

// Function to construct and print MST for
// a graph represented using adjacency
// matrix representation
void primMST(int graph[V][V])
{
    // Array to store constructed MST
    int parent[V];

    // Key values used to pick minimum weight edge in cut
    int key[V];

    // To represent set of vertices included in MST
    bool mstSet[V];

    // Initialize all keys as INFINITE
    for (int i = 0; i < V; i++)
        key[i] = INT_MAX, mstSet[i] = false;

    // Always include first 1st vertex in MST.
    // Make key 0 so that this vertex is picked as first vertex.
    key[0] = 0;
    parent[0] = -1; // First node is always root of MST
    // The MST will have V vertices
    for (int count = 0; count < V - 1; count++)
    {
        // Pick the minimum key vertex from the
        // set of vertices not yet included in MST
        int u = minKey(key, mstSet);

        // Add the picked vertex to the MST Set
        mstSet[u] = true;

        // Update key value and parent index of
        // the adjacent vertices of the picked vertex.
        // Consider only those vertices which are not
        // yet included in MST
        for (int v = 0; v < V; v++)

            // graph[u][v] is non zero only for adjacent
vertices of m
            // mstSet[v] is false for vertices not yet included
in MST
            // Update the key only if graph[u][v] is smaller
than key[v]
            if (graph[u][v] && mstSet[v] == false && graph[u][v]
< key[v])
                parent[v] = u, key[v] = graph[u][v];
    }

    // print the constructed MST
    printMST(parent, graph);
}

// Driver code
int main()
{
    /* Let us create the following graph
        2 3
    (0)--(1)--(2)
    | / \ |
    6| 8/ \5 |7
    | / \ |
    (3)-------(4)
            9     */
    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 } };
    // Print the solution
    primMST(graph);

    return 0;
}
```

OUTPUT



"F:\CS\sem 4\Design and analysis of algorithms\DAA Practicals\9b_Kruskals.exe"

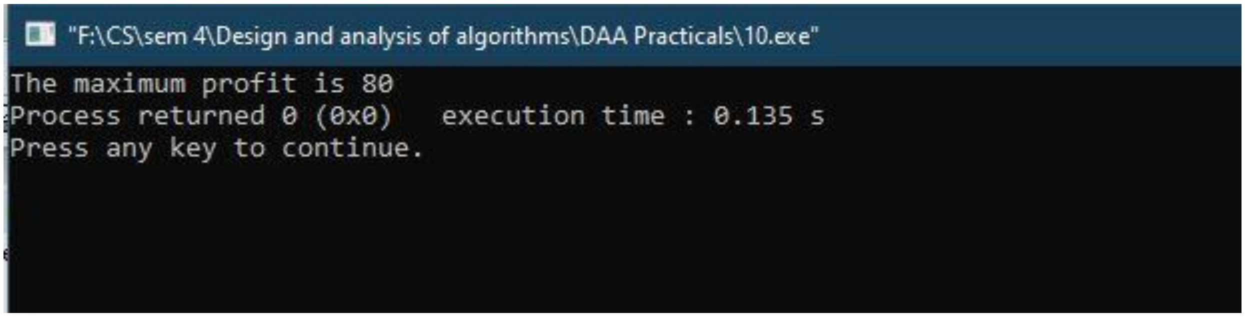
```
Following are the edges in the constructed MST
2 -- 3 == 4
0 -- 3 == 5
0 -- 1 == 10
Minimum Cost Spanning Tree: 19
Process returned 0 (0x0)   execution time : 0.225 s
Press any key to continue.
```

10. Write a program to solve the weighted interval scheduling problem

INPUT

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

// Data structure to store a Job
struct Job {
    int start, finish, profit;
};

// Function to find the maximum profit of non-overlapping jobs
using LIS
int findMaxProfit(vector<Job> jobs)        // no-ref, no-const
{
    // sort the jobs according to increasing order of their
start time
    sort(jobs.begin(), jobs.end(),
        [](Job &x, Job &y) {
            return x.start < y.start;
        });

    // get the number of jobs
    int n = jobs.size();

    // base case
    if (n == 0) {
        return 0;
    }

    // `maxProfit[i]` stores the maximum profit of non-
conflicting jobs
    // ending at the i'th job
    int maxProfit[n];

    // consider every job
    for (int i = 0; i < n; i++)
    {
        // initialize current profit to 0
        maxProfit[i] = 0;

        // consider each `j` less than `i`
        for (int j = 0; j < i; j++)
        {
            // if the j'th job is not conflicting with the i'th
job and
            // is leading to the maximum profit
            if (jobs[j].finish <= jobs[i].start && maxProfit[i]
< maxProfit[j]) {
                maxProfit[i] = maxProfit[j];
            }
        }

        // end the current task with i'th job
        maxProfit[i] += jobs[i].profit;
    }

    // return the maximum profit
    return *max_element(maxProfit, maxProfit + n);
}

int main()
{
    vector<Job> jobs {
        { 0, 6, 60 },
        { 5, 9, 50 },
        { 1, 4, 30 },
        { 5, 7, 30 },
        { 3, 5, 10 },
        { 7, 8, 10 }
    };

    cout << "The maximum profit is " << findMaxProfit(jobs);

    return 0;
}
```

OUTPUT



"F:\CS\sem 4\Design and analysis of algorithms\DAA Practicals\10.exe"

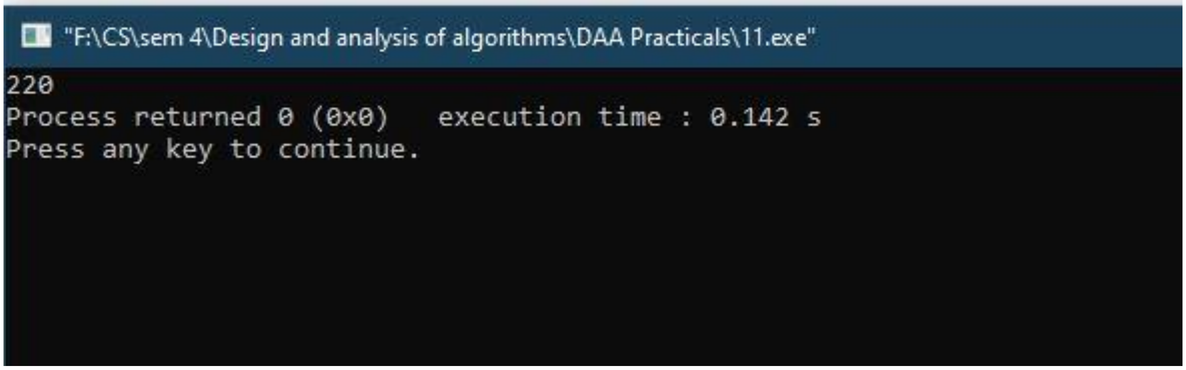
```
The maximum profit is 80
Process returned 0 (0x0)   execution time : 0.135 s
Press any key to continue.
```

11. Write a program to solve the 0-1 knapsack problem

INPUT

```
#include <bits/stdc++.h>
using namespace std;

// A utility function that returns
// maximum of two integers
int max(int a, int b) { return (a > b) ? a : b; }

// Returns the maximum value that
// can be put in a knapsack of capacity W
int knapSack(int W, int wt[], int val[], int n)
{

    // Base Case
    if (n == 0 || W == 0)
        return 0;

    // If weight of the nth item is more
    // than Knapsack capacity W, then
    // this item cannot be included
    // in the optimal solution
    if (wt[n - 1] > W)
        return knapSack(W, wt, val, n - 1);
    // Return the maximum of two cases:
    // (1) nth item included
    // (2) not included
    else
        return max(
            val[n - 1]
                + knapSack(W - wt[n - 1],
                           wt, val, n - 1),
            knapSack(W, wt, val, n - 1));
}

// Driver code
int main()
{
    int val[] = { 60, 100, 120 };
    int wt[] = { 10, 20, 30 };
    int W = 50;
    int n = sizeof(val) / sizeof(val[0]);
    cout << knapSack(W, wt, val, n);
    return 0;
}
```

OUTPUT



"F:\CS\sem 4\Design and analysis of algorithms\DAA Practicals\11.exe"

```
220
Process returned 0 (0x0)   execution time : 0.142 s
Press any key to continue.
```