Discrete Structure Practical (2020-21)

Set –B (University Roll No.: 20003570023)

Q1 - Write a program to implement selection sort. Find the number of comparisons during each pass and display the intermediate result.

A1 – Source Code

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

void displayArray(int* &array , int* &size){
   cout<<"Displaying Your Array...."<<endl;
   cout<<"[ ";
   for (int i = 0; i < *size; i++){
      cout<<array[i];
      if(i!= *size-1){
         cout<<" ,";
      }
   }
   cout<<" ]";
   cout<<endl;
}
void swap(int* &a,int &b){
   int temp;
   temp = *a;
   *a   = b;
   b    = temp;
}
void selectionSort(int* &array,int* &length){
   cout<<endl;
   cout<<"Implementing Selection Sort...."<<endl;
   cout<<endl;
   int* minElement = new int();
   for (int i = 0; i < *length; i++)
   {
      minElement = &array[i];
      int comparisonCount = 0;
      for (int j = i+1; j < *length; j++)
      {
         if(array[j]< *minElement){
            minElement =&array[j];
         }
         comparisonCount++;
      }
```

```
        swap(minElement,array[i]);
        cout<<"ITERATION : "<<i+1<<endl;
        displayArray(array,length);
        cout<<endl;
    }
}
int main(){
    int  arr[]  = {55,3,4,6,76,89,34,1,7,8,9,0};
    int *array  = arr;
    int *length = new int(12);
    displayArray(array,length);
    selectionSort(array,length);
    displayArray(array,length);

    return 0;
}
```

Output –

```
Displaying Your Array....
[ 55 ,3 ,4 ,6 ,76 ,89 ,34 ,1 ,7 ,8 ,9 ,0 ]

Implementing Selection Sort....

ITERATION : 1
Displaying Your Array....
[ 0 ,3 ,4 ,6 ,76 ,89 ,34 ,1 ,7 ,8 ,9 ,55 ]

ITERATION : 2
Displaying Your Array....
[ 0 ,1 ,4 ,6 ,76 ,89 ,34 ,3 ,7 ,8 ,9 ,55 ]

ITERATION : 3
Displaying Your Array....
[ 0 ,1 ,3 ,6 ,76 ,89 ,34 ,4 ,7 ,8 ,9 ,55 ]

ITERATION : 4
Displaying Your Array....
[ 0 ,1 ,3 ,4 ,76 ,89 ,34 ,6 ,7 ,8 ,9 ,55 ]

ITERATION : 5
Displaying Your Array....
[ 0 ,1 ,3 ,4 ,6 ,89 ,34 ,76 ,7 ,8 ,9 ,55 ]

ITERATION : 6
Displaying Your Array....
[ 0 ,1 ,3 ,4 ,6 ,7 ,34 ,76 ,89 ,8 ,9 ,55 ]

ITERATION : 7
Displaying Your Array....
[ 0 ,1 ,3 ,4 ,6 ,7 ,8 ,76 ,89 ,34 ,9 ,55 ]

ITERATION : 8
Displaying Your Array....
[ 0 ,1 ,3 ,4 ,6 ,7 ,8 ,9 ,89 ,34 ,76 ,55 ]

ITERATION : 9
Displaying Your Array....
[ 0 ,1 ,3 ,4 ,6 ,7 ,8 ,9 ,34 ,89 ,76 ,55 ]
```

```
ITERATION : 10
Displaying Your Array....
[ 0 ,1 ,3 ,4 ,6 ,7 ,8 ,9 ,34 ,55 ,76 ,89 ]

ITERATION : 11
Displaying Your Array....
[ 0 ,1 ,3 ,4 ,6 ,7 ,8 ,9 ,34 ,55 ,76 ,89 ]

ITERATION : 12
Displaying Your Array....
[ 0 ,1 ,3 ,4 ,6 ,7 ,8 ,9 ,34 ,55 ,76 ,89 ]

Displaying Your Array....
[ 0 ,1 ,3 ,4 ,6 ,7 ,8 ,9 ,34 ,55 ,76 ,89 ]
```

Q2 - Write a program to accept directed graph G using adjacency matrices and compute the in-degree and out-degree of each vertex.

A2 – Source Code

```
#include <iostream.h>
#include <conio.h>

using namespace std;

// Function to print the in and out degrees
// of all the vertices of the given graph
void findInOutDegree(list<list<int>> adjlist, int n)
{
   int* iN = new int[n]();
   int* ouT = new int[n]();

   list<list<int> >::iterator nest_list;
   int i = 0;

   for(nest_list = adjlist.begin();
      nest_list != adjlist.end();
      nest_list++)
   {
      list<int> lst = *nest_list;

      // Out degree for ith vertex will be the count
      // of direct paths from i to other vertices
      ouT[i] = lst.size();

      for(auto it = lst.begin();
            it != lst.end(); it++)
      {
         // Every vertex that has an incoming
         // edge from i
         iN[*it]++;
      }
      i++;
   }

   cout << "Vertex\t\tIn\t\tOut" << endl;
   for(int k = 0; k < n; k++)
   {
      cout << k << "\t\t"
          << iN[k] << "\t\t"
          << ouT[k] << endl;
   }
```

```
}

// Driver code
int main()
{

    // Adjacency list representation of the graph
    list<list<int>> adjlist;

    // Vertices 1 and 2 have an incoming edge
    // from vertex 0
    list<int> tmp;
    tmp.push_back(1);
    tmp.push_back(2);
    adjlist.push_back(tmp);
    tmp.clear();

    // Vertex 3 has an incoming edge
    // from vertex 1
    tmp.push_back(3);
    adjlist.push_back(tmp);
    tmp.clear();

    // Vertices 0, 5 and 6 have an incoming
    // edge from vertex 2
    tmp.push_back(0);
    tmp.push_back(5);
    tmp.push_back(6);
    adjlist.push_back(tmp);
    tmp.clear();

    // Vertices 1 and 4 have an incoming
    // edge from vertex 3
    tmp.push_back(1);
    tmp.push_back(4);
    adjlist.push_back(tmp);
    tmp.clear();

    // Vertices 2 and 3 have an incoming
    // edge from vertex 4
    tmp.push_back(2);
    tmp.push_back(3);
    adjlist.push_back(tmp);
    tmp.clear();

    // Vertices 4 and 6 have an incoming
    // edge from vertex 5
```

```
    tmp.push_back(4);
    tmp.push_back(6);
    adjlist.push_back(tmp);
    tmp.clear();

    // Vertex 5 has an incoming
    // edge from vertex 6
    tmp.push_back(5);
    adjlist.push_back(tmp);
    tmp.clear();

    int n = adjlist.size();

    findInOutDegree(adjlist, n);
}
```

Output –