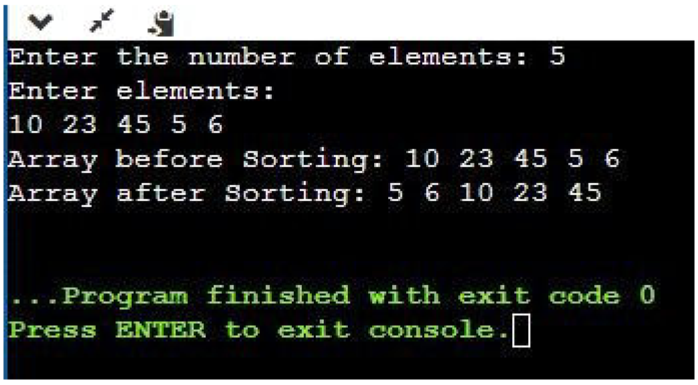
DAA Prac 06

NAME : Vijiyant Tanaji Shejwalkar

Reg NO: 2020BIT057

1. Insertion Sort

```
#include<iostream>

using namespace std;

void display(int *array, int size) {

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

    cout << array[i] << " ";

  cout << endl;

}

void insertionSort(int *array, int size) {

  int key, j;

  for(int i = 1; i<size; i++) {

    key = array[i];//take value

    j = i;

    while(j > 0 && array[j-1]>key) {

      array[j] = array[j-1];

      j--;

    }

    array[j] = key;   //insert in right place

  }

}

int main() {

  int n;
  cout << "Enter the number of elements: ";

  cin >> n;

  int 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);

  insertionSort(arr, n);

  cout << "Array after Sorting: ";

  display(arr, n);

}
```



```
Enter the number of elements: 5
Enter elements:
10 23 45 5 6
Array before Sorting: 10 23 45 5 6
Array after Sorting: 5 6 10 23 45


...Program finished with exit code 0
Press ENTER to exit console.
```

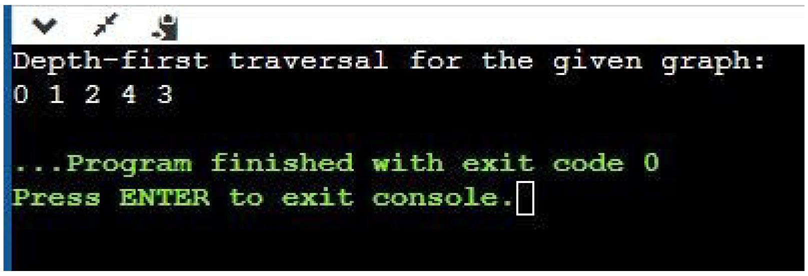
2. DFS

```
#include <iostream>

#include <list>

using namespace std;

//graph class for DFS travesal
```

```
class DFSGraph

{

int V;    // No. of vertices

list<int> *adjList;    // adjacency list

void DFS_util(int v, bool visited[]);  // A function used by DFS

public:

   // class Constructor

DFSGraph(int V)

   {

 this->V = V;

 adjList = new list<int>[V];

   }

   // function to add an edge to graph

void addEdge(int v, int w){

adjList[v].push_back(w); // Add w to v’s list.

   }


void DFS();    // DFS traversal function

};

void DFSGraph::DFS_util(int v, bool visited[])

{

   // current node v is visited

visited[v] = true;

cout << v << " ";


   // recursively process all the adjacent vertices of the node

list<int>::iterator i;

for(i = adjList[v].begin(); i != adjList[v].end(); ++i)

if(!visited[*i])
```

```
DFS_util(*i, visited);

}



// DFS traversal

void DFSGraph::DFS()

{

   // initially none of the vertices are visited

bool *visited = new bool[V];

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

visited[i] = false;



   // explore the vertices one by one by recursively calling  DFS_util

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

if (visited[i] == false)

DFS_util(i, visited);

}



int main()

{

   // Create a graph

DFSGraph gdfs(5);

gdfs.addEdge(0, 1);

gdfs.addEdge(0, 2);

gdfs.addEdge(0, 3);

gdfs.addEdge(1, 2);

gdfs.addEdge(2, 4);

gdfs.addEdge(3, 3);

gdfs.addEdge(4, 4);
```

```
cout << "Depth-first traversal for the given graph:"<<endl;

gdfs.DFS();


return 0;

}
```



```
Depth-first traversal for the given graph:
0 1 2 4 3

...Program finished with exit code 0
Press ENTER to exit console.
```

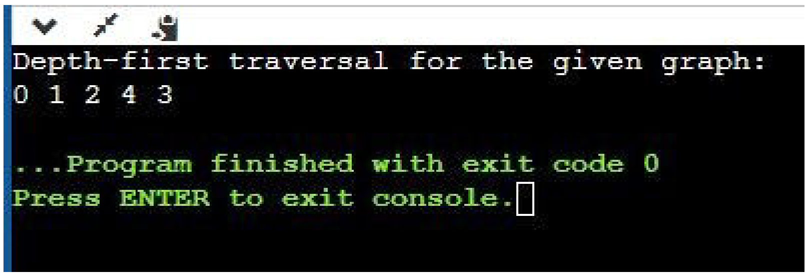
3. BFS

```
// Program to print BFS traversal from a given

// source vertex. BFS(int s) traverses vertices

// reachable from s.

#include <bits/stdc++.h>

using namespace std;


// This class represents a directed graph using

// adjacency list representation

class Graph {

        int V; // No. of vertices


        // Pointer to an array containing adjacency

        // lists

        vector<list<int> > adj;
```

```
public:

                    Graph(int V); // Constructor


                    // function to add an edge to graph

                    void addEdge(int v, int w);


                    // prints BFS traversal from a given source s

                    void BFS(int s);

};


Graph::Graph(int V)

{

                    this->V = V;

                    adj.resize(V);

}


void Graph::addEdge(int v, int w)

{

                    adj[v].push_back(w); // Add w to v's list.

}


void Graph::BFS(int s)

{

                    // Mark all the vertices as not visited

                    vector<bool> visited;

                    visited.resize(V, false);


                    // Create a queue for BFS

                    list<int> queue;


                    // Mark the current node as visited and enqueue it
```

```
		visited[s] = true;
		queue.push_back(s);

		while (!queue.empty()) {
			// Dequeue a vertex from queue and print it
			s = queue.front();
			cout << s << " ";
			queue.pop_front();

			// Get all adjacent vertices of the dequeued
			// vertex s. If a adjacent has not been visited,
			// then mark it visited and enqueue it
			for (auto adjacent : adj[s]) {
				if (!visited[adjacent]) {
					visited[adjacent] = true;
					queue.push_back(adjacent);
				}
			}
		}
}

// Driver program to test methods of graph class
int main()
{
		// Create a graph given in the above diagram
		Graph g(4);
		g.addEdge(0, 1);
		g.addEdge(0, 2);
		g.addEdge(1, 2);
		g.addEdge(2, 0);
		g.addEdge(2, 3);
```

```
                g.addEdge(3, 3);


                cout << "Following is Breadth First Traversal "
                                << "(starting from vertex 2) \n";
                g.BFS(2);


                return 0;
}
```



```
Depth-first traversal for the given graph:
0 1 2 4 3

...Program finished with exit code 0
Press ENTER to exit console.
```