

PRACTICAL:-06

INSERTION SORT:

// C++ program for insertion sort

#include <bits/stdc++.h>

using namespace std;

// Function to sort an array using

// insertion sort

void insertionSort(int arr[], int n)

{

int i, key, j;

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

{

key = arr[i];

j = i - 1;

// Move elements of arr[0..i-1],

// that are greater than key, to one

// position ahead of their

// current position

while (j >= 0 && arr[j] > key)

```

        {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}

```

// A utility function to print an array

// of size n

```
void printArray(int arr[], int n)
```

```

{
    int i;
    for (i = 0; i < n; i++)
        cout << arr[i] << " ";
    cout << endl;
}

```

// Driver code

```
int main()
```

```

{
    int arr[] = { 12, 11, 13, 5, 6 };
    int N = sizeof(arr) / sizeof(arr[0]);
}

```

```

        insertionSort(arr, N);

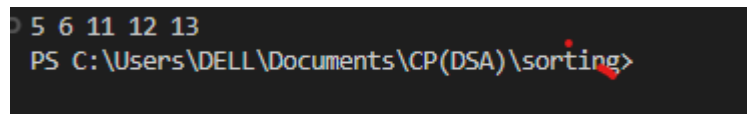
        printArray(arr, N);

        return 0;

    }

```

Output:



```

5 6 11 12 13
PS C:\Users\DELL\Documents\CP(DSA)\sorting>

```

DFS:

```
// C++ program to print DFS
```

```
// traversal for a given
```

```
// graph
```

```
#include <bits/stdc++.h>
```

```
using namespace std;
```

```
class Graph {
```

```
    // A function used by DFS
```

```
    void DFSUtil(int v);
```

```
public:
```

```
    map<int, bool> visited;
```

```
    map<int, list<int> > adj;
```

```
    // function to add an edge to graph
```

```
    void addEdge(int v, int w);
```

```

        // prints DFS traversal of the complete graph
        void DFS();

};

void Graph::addEdge(int v, int w)
{
    adj[v].push_back(w); // Add w to v's list.
}

void Graph::DFSUtil(int v)
{
    // Mark the current node as visited and print it
    visited[v] = true;
    cout << v << " ";

    // Recur for all the vertices adjacent to this vertex
    list<int>::iterator i;
    for (i = adj[v].begin(); i != adj[v].end(); ++i)
        if (!visited[*i])
            DFSUtil(*i);
}

// The function to do DFS traversal. It uses recursive

```

```
// DFSUtil()

void Graph::DFS()
{
    // Call the recursive helper function to print DFS
    // traversal starting from all vertices one by one
    for (auto i : adj)
        if (visited[i.first] == false)
            DFSUtil(i.first);
}
```

```
// Driver's Code
```

```
int main()
{
    // Create a graph given in the above diagram
    Graph g;
    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 Depth First Traversal \n";
```

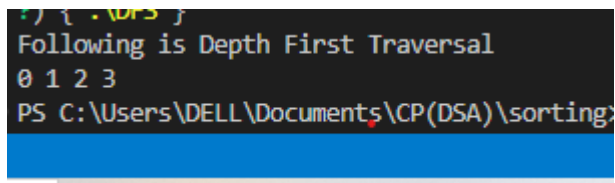
```

        // Function call
        g.DFS();

        return 0;
    }

```

Output:



```

1) { .DFS }
Following is Depth First Traversal
0 1 2 3
PS C:\Users\DELL\Documents\CP(DSA)\sorting>

```

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;
}

```

Output:

```

?) { .\bfs }
Following is Breadth First Traversal (starting from vertex 2)
2 0 3 1
PS C:\Users\DELL\Documents\CP(DSA)\sorting> 

```