Name : Pede Amruta Dipakrao

Reg. no.:2020Bit070

Assignment 6

write a C/C++ program to implement

Decrease and conquer algorithm

1) Insertion sort

2) DFS

3) BFS

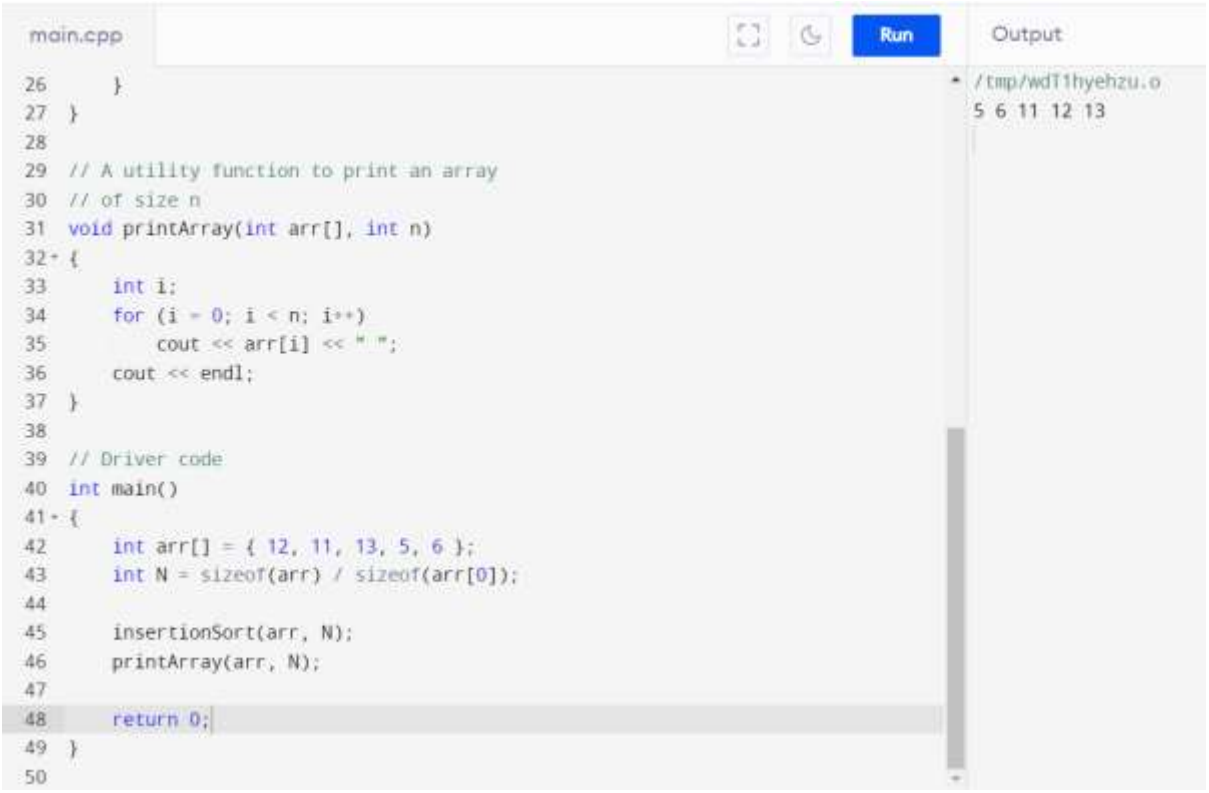
1)Insertion sort:

```
// 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:



2) DFS:

```
// C++ program to print DFS traversal from
// a given vertex in a given graph
#include <bits/stdc++.h>
using namespace std;

// Graph class represents a directed graph
// using adjacency list representation
class Graph {
```

```
public:
        map<int, bool> visited;
        map<int, list<int> > adj;

        // function to add an edge to graph
        void addEdge(int v, int w);

        // DFS traversal of the vertices
        // reachable from v
        void DFS(int v);
};

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

void Graph::DFS(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])
```

```
                        DFS(*i);
}

// 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"
                        " (starting from vertex 2) \n";

        // Function call
        g.DFS(2);

        return 0;
}
```

Output:

3)BFS:

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


class Solution {
  public:
    // Function to return Breadth First Traversal of given graph.
    vector<int> bfsOfGraph(int V, vector<int> adj[]) {
        int vis[V] = {0};
        vis[0] = 1;
        queue<int> q;
        // push the initial starting node
        q.push(0);
        vector<int> bfs;
        // iterate till the queue is empty
        while(!q.empty()) {
          // get the topmost element in the queue
           int node = q.front();
```

```
            q.pop();
            bfs.push_back(node);
            // traverse for all its neighbours
            for(auto it : adj[node]) {
               // if the neighbour has previously not been visited,
               // store in Q and mark as visited
               if(!vis[it]) {
                  vis[it] = 1;
                  q.push(it);
               }
            }
         }
         return bfs;
      }
};

void addEdge(vector <int> adj[], int u, int v) {
   adj[u].push_back(v);
   adj[v].push_back(u);
}

void printAns(vector <int> &ans) {
   for (int i = 0; i < ans.size(); i++) {
      cout << ans[i] << " ";
   }
}

int main()
{
```

```
    vector <int> adj[6];

    addEdge(adj, 0, 1);
    addEdge(adj, 1, 2);
    addEdge(adj, 1, 3);
    addEdge(adj, 0, 4);

    Solution obj;
    vector <int> ans = obj.bfsOfGraph(5, adj);
    printAns(ans);

    return 0;
}
```

Output: