Name:- Yash Sanjay Kale
Reg No:- 2020BIT047

Practical 06: write a C/C++ program to implement Decrease and conquer algorithm

1) Insertion sort

```
// C++ program for insertion sort

#include <bits/stdc++.h>
using namespace std;
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;
	}
}
void printArray(int arr[], int n)
{
	int i;
	for (i = 0; i < n; i++)
		cout << arr[i] << " ";
	cout << endl;
}
int main()
{
	int arr[] = { 12, 11, 13, 5, 6 };
	int N = sizeof(arr) / sizeof(arr[0]);

	insertionSort(arr, N);
	printArray(arr, N);

	return 0;
}
```

2) DFS

```
#include <bits/stdc++.h>
using namespace std;
class Graph {
public:
	map<int, bool> visited;
	map<int, list<int> > adj;

	void addEdge(int v, int w);
	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)
{
      visited[v] = true;
      cout << v << " ";
      list<int>::iterator i;
      for (i = adj[v].begin(); i != adj[v].end(); ++i)
            if (!visited[*i])
                  DFS(*i);
}


int main()
{

      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";

      g.DFS(2);

      return 0;
}
```

3) BFS

```
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>

#define MAX_VERTICES 50
typedef struct Graph_t {
      int V; // No. of vertices
      bool adj[MAX_VERTICES][MAX_VERTICES];
} Graph;
Graph* Graph_create(int V)
{
      Graph* g = malloc(sizeof(Graph));
      g->V = V;

      for (int i = 0; i < V; i++) {
            for (int j = 0; j < V; j++) {
                  g->adj[i][j] = false;
            }
      }

      return g;
}
```

```
void Graph_destroy(Graph* g) { free(g); }


void Graph_addEdge(Graph* g, int v, int w)
{
      g->adj[v][w] = true; // Add w to v’s list.
}


void Graph_BFS(Graph* g, int s)
{

      bool visited[MAX_VERTICES];
      for (int i = 0; i < g->V; i++) {
            visited[i] = false;
      }
      int queue[MAX_VERTICES];
      int front = 0, rear = 0;
      visited[s] = true;
      queue[rear++] = s;

      while (front != rear) {

            s = queue[front++];
            printf("%d ", s);
            for (int adjecent = 0; adjecent < g->V;
                  adjecent++) {
                  if (g->adj[s][adjecent] && !visited[adjecent]) {
                        visited[adjecent] = true;
                        queue[rear++] = adjecent;
                  }
            }
      }
}

int main()
{
      Graph* g = Graph_create(4);
      Graph_addEdge(g, 0, 1);
      Graph_addEdge(g, 0, 2);
      Graph_addEdge(g, 1, 2);
      Graph_addEdge(g, 2, 0);
      Graph_addEdge(g, 2, 3);
      Graph_addEdge(g, 3, 3);

      printf("Following is Breadth First Traversal "
            "(starting from vertex 2) \n");
      Graph_BFS(g, 2);

      Graph_destroy(g);

      return 0;
}
```