**EXPERIMENT NO.7**

**Date:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Aim:** Breadth first Search Code

**Algorithm/Pseudo-Code:**

For all nodes except *s*

d[v]=∞

d[*s*]=0

Q= s

While Q ≠ Ø

u= Dequeue(Q)

for all adjacent nodes v of u

if d[v]= ∞

then d[v]=d[u]+1

Enqueue(Q,v)

**Program:**

import java.io.\*;

import java.util.\*;

class Graph

{

    private int V;   // No. of vertices

    private LinkedList<Integer> adj[]; //Adjacency Lists

    Graph(int v)

    {

        V = v;

        adj = new LinkedList[v];

        for (int i=0; i<v; ++i)

            adj[i] = new LinkedList();

    }

    void addEdge(int v,int w)

    {

        adj[v].add(w);

    }

    void BFS(int s)

    {

        boolean visited[] = new boolean[V];

        LinkedList<Integer> queue = new LinkedList<Integer>();

        visited[s]=true;

        queue.add(s);

        while (queue.size() != 0)

        {

            s = queue.poll();

            System.out.print(s+" ");

            Iterator<Integer> i = adj[s].listIterator();

            while (i.hasNext())

            {

                int n = i.next();

                if (!visited[n])

                {

                    visited[n] = true;

                    queue.add(n);

                }

            }

        }

    }

    public static void main(String args[])

    {

        Graph g = new Graph(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);

        System.out.println("Following is Breadth First Traversal "+

                           "(starting from vertex 2)");

        g.BFS(2);

    }

}

**Output:**

**A screenshot of a cell phone

Description automatically generated**

**EXPERIMENT NO.8**

**Date:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Aim:** Code for Dfs

**Algorithm/Pseudo-Code:**

Initialization:

mark all vertices as unvisited,

visit(s)

while the stack is not empty:

pop (v,w)

if w is not visited

add (v,w) to tree T

visit(w)

Procedure visit(v)

mark v as visited

for each edge (v,w)

push (v,w) in the stack

**Program:**

import java.io.\*;

import java.util.\*;

class Graph

{

    private int V;

    private LinkedList<Integer> adj[];

    Graph(int v)

    {

        V = v;

        adj = new LinkedList[v];

        for (int i=0; i<v; ++i)

            adj[i] = new LinkedList();

    }

    void addEdge(int v, int w)

    {

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

    }

    void DFSUtil(int v,boolean visited[])

    {

        visited[v] = true;

        System.out.print(v+" ");

        Iterator<Integer> i = adj[v].listIterator();

        while (i.hasNext())

        {

            int n = i.next();

            if (!visited[n])

                DFSUtil(n, visited);

        }

    }

    void DFS(int v)

    {

        boolean visited[] = new boolean[V];

        DFSUtil(v, visited);

    }

    public static void main(String args[])

    {

        Graph g = new Graph(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);

        System.out.println("Following is Depth First Traversal "+

                           "(starting from vertex 2)");

        g.DFS(2);

    }

}

**Output:**

**A screenshot of a cell phone

Description automatically generated**

**EXPERIMENT NO.9**

**Date:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Aim:Heap Sort Code**

**Algorithm/Pseudo-Code:**

*heapify the array;*

*while the array isn’t empty {*

*remove and replace the root;*

*reheap the new root node;  
}*

**Program:**

public class HeapSort

{

    public void sort(int arr[])

    {

        int n = arr.length;

        for (int i = n / 2 - 1; i >= 0; i--)

            heapify(arr, n, i);

        for (int i=n-1; i>=0; i--)

        {

            int temp = arr[0];

            arr[0] = arr[i];

            arr[i] = temp;

            heapify(arr, i, 0);

        }

    }

    void heapify(int arr[], int n, int i)

    {

        int largest = i; // Initialize largest as root

        int l = 2\*i + 1; // left = 2\*i + 1

        int r = 2\*i + 2; // right = 2\*i + 2

        if (l < n && arr[l] > arr[largest])

            largest = l;

        if (r < n && arr[r] > arr[largest])

            largest = r;

        if (largest != i) {

            int swap = arr[i];

            arr[i] = arr[largest];

            arr[largest] = swap;

            heapify(arr, n, largest);

        }

    }

    static void printArray(int arr[])

    {

        int n = arr.length;

        for (int i=0; i<n; ++i)

            System.out.print(arr[i]+" ");

        System.out.println();

    }

    public static void main(String args[])

    {

        int arr[] = {12, 11, 13, 5, 6, 7};

        int n = arr.length;

        HeapSort ob = new HeapSort();

        ob.sort(arr);

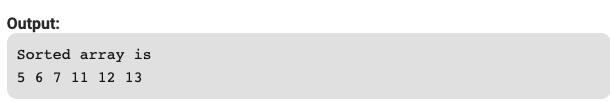
        System.out.println("Sorted array is");

        printArray(arr);

    }

}

**Output:**

****

**EXPERIMENT NO.10**

**Date:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Aim:** QuickSort code

**Algorithm/Pseudo-Code:**

1. if left < right

Partition a[left...right] such that:

all a[left...p-1] are less than a[p], and

all a[p+1...right] are >= a[p]

Quicksort a[left...p-1]

Quicksort a[p+1...right]

2. End

**Program:**

public class MyQuickSort {

    private int array[];

    private int length;

    public void sort(int[] inputArr) {

        if (inputArr == null || inputArr.length == 0) {

            return;

        }

        this.array = inputArr;

        length = inputArr.length;

        quickSort(0, length - 1);

    }

    private void quickSort(int lowerIndex, int higherIndex) {

        int i = lowerIndex;

        int j = higherIndex;

        int pivot = array[lowerIndex+(higherIndex-lowerIndex)/2];

        while (i <= j) {

            while (array[i] < pivot) {

                i++;

            }

            while (array[j] > pivot) {

                j--;

            }

            if (i <= j) {

                exchangeNumbers(i, j);

                i++;

                j--;

            }

        }

        if (lowerIndex < j)

            quickSort(lowerIndex, j);

        if (i < higherIndex)

            quickSort(i, higherIndex);

    }

    private void exchangeNumbers(int i, int j) {

        int temp = array[i];

        array[i] = array[j];

        array[j] = temp;

    }

    public static void main(String a[]){

        MyQuickSort sorter = new MyQuickSort();

        int[] input = {24,2,45,20,56,75,2,56,99,53,12};

        sorter.sort(input);

        for(int i:input){

            System.out.print(i);

            System.out.print(" ");

        }

    }

}

**Output:**

**A close up of a logo

Description automatically generated**

**EXPERIMENT NO.11**

**Date:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Aim:** MergeSort Code

**Algorithm/Pseudo-Code:**

Divide the unsorted list into n sublists, each comprising 1 element (a list of 1 element is supposed sorted).

Repeatedly merge sublists to produce newly sorted sublists until there is only 1 sublist remaining. This will be the sorted list.

**Program:**

class MergeSort

{

    void merge(int arr[], int l, int m, int r)

    {

        int n1 = m - l + 1;

        int n2 = r - m;

        int L[] = new int [n1];

        int R[] = new int [n2];

        for (int i=0; i<n1; ++i)

            L[i] = arr[l + i];

        for (int j=0; j<n2; ++j)

            R[j] = arr[m + 1+ j];

        int i = 0, j = 0;

        int k = l;

        while (i < n1 && j < n2)

        {

            if (L[i] <= R[j])

            {

                arr[k] = L[i];

                i++;

            }

            else

            {

                arr[k] = R[j];

                j++;

            }

            k++;

        }

        while (i < n1)

        {

            arr[k] = L[i];

            i++;

            k++;

        }

        while (j < n2)

        {

            arr[k] = R[j];

            j++;

            k++;

        }

    }

    void sort(int arr[], int l, int r)

    {

        if (l < r)

        {

            int m = (l+r)/2;

            sort(arr, l, m);

            sort(arr , m+1, r);

            merge(arr, l, m, r);

        }

    }

    static void printArray(int arr[])

    {

        int n = arr.length;

        for (int i=0; i<n; ++i)

            System.out.print(arr[i] + " ");

        System.out.println();

    }

    public static void main(String args[])

    {

        int arr[] = {12, 11, 13, 5, 6, 7};

        System.out.println("Given Array");

        printArray(arr);

        MergeSort ob = new MergeSort();

        ob.sort(arr, 0, arr.length-1);

        System.out.println("\nSorted array");

        printArray(arr);

    }

}

**Output:**

**A close up of a logo

Description automatically generated**