A document with text and images

Description automatically generated

A white paper with black text

Description automatically generated

A white paper with black text

Description automatically generated

Max priority code:

import java.util.Arrays;

public class MaxPriorityQueue{

    public void MaxHeapInsert(int[] A, int key){

        A[A.length] = Integer.MIN\_VALUE;

        HeapIncreaseKey(A, A.length, key);

    }

    public int HeapMax(int[] A){

        return A[0];

    }

    public int HeapExtractMax(int[] A){

        if(A.length < 1){

            System.out.println("Error: Heap underflow");

        }

        int max = A[0];

        A[0] = A[A.length];

        A = Arrays.copyOf(A, A.length-1);

        MaxHeapify(A, 0);

        return max;

    }

    public void HeapIncreaseKey(int[] A, int i, int key){

        if(key < A[i]){

            System.out.println("New key is smaller than current key");

            return;

        }

        A[i] = key;

        while(i > 0 && A[(i-1)/2] < A[i]){

            int temp = A[i];

            A[i] = A[(i-1)/2];

            A[(i-1)/2] = temp;

            i = (i-1)/2;

        }

    }

    public void MaxHeapify(int[] A, int i){

        int l = 2\*i + 1;

        int r = 2\*i + 2;

        int largest = i;

        if(l < A.length && A[l] > A[i]){

            largest = l;

        }

        if(r < A.length && A[r] > A[largest]){

            largest = r;

        }

        if(largest != i){

            int temp = A[i];

            A[i] = A[largest];

            A[largest] = temp;

            MaxHeapify(A, largest);

        }

    }

    public int Parent(int i){

        return (i-1)/2;

    }

    public int Left(int i){

        return 2\*i + 1;

    }

    public int Right(int i){

        return 2\*i + 2;

    }

    public static void main(String[] args){

        int[] A = {4, 1, 3, 2, 16, 9, 10, 14, 8, 7};

        MaxPriorityQueue mpq = new MaxPriorityQueue();

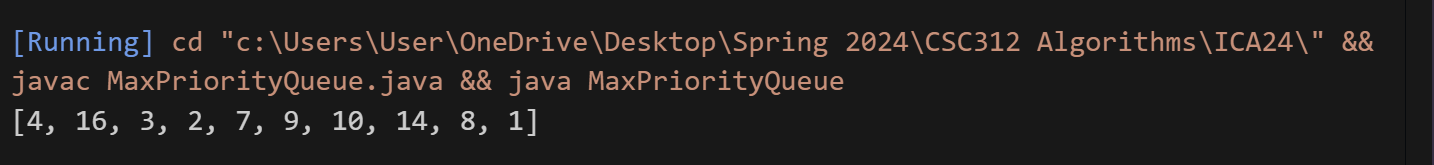
        mpq.MaxHeapify(A, 1);

        System.out.println(Arrays.toString(A));

    }

}

Output:



Min Priority Queue code:

import java.util.Arrays;

public class MinPriorityQueue {

    public void MinHeapInsert(int[] A, int key) {

        A[A.length] = Integer.MAX\_VALUE; // Assuming Integer.MAX\_VALUE represents positive infinity for minimum priority queue

        HeapDecreaseKey(A, A.length, key);

    }

    public int HeapMin(int[] A) {

        return A[0];

    }

    public int HeapExtractMin(int[] A) {

        if (A.length < 1) {

            System.out.println("Error: Heap underflow");

        }

        int min = A[0];

        A[0] = A[A.length - 1];

        A = Arrays.copyOf(A, A.length - 1);

        MinHeapify(A, 0);

        return min;

    }

    public void HeapDecreaseKey(int[] A, int i, int key) {

        if (key > A[i]) {

            System.out.println("New key is larger than current key");

            return;

        }

        A[i] = key;

        while (i > 0 && A[(i - 1) / 2] > A[i]) {

            int temp = A[i];

            A[i] = A[(i - 1) / 2];

            A[(i - 1) / 2] = temp;

            i = (i - 1) / 2;

        }

    }

    public void MinHeapify(int[] A, int i) {

        int l = 2 \* i + 1;

        int r = 2 \* i + 2;

        int smallest = i;

        if (l < A.length && A[l] < A[i]) {

            smallest = l;

        }

        if (r < A.length && A[r] < A[smallest]) {

            smallest = r;

        }

        if (smallest != i) {

            int temp = A[i];

            A[i] = A[smallest];

            A[smallest] = temp;

            MinHeapify(A, smallest);

        }

    }

    public int Parent(int i) {

        return (i - 1) / 2;

    }

    public int Left(int i) {

        return 2 \* i + 1;

    }

    public int Right(int i) {

        return 2 \* i + 2;

    }

    public static void main(String[] args) {

        int[] A = {4, 1, 3, 2, 16, 9, 10, 14, 8, 7};

        MinPriorityQueue mpq = new MinPriorityQueue();

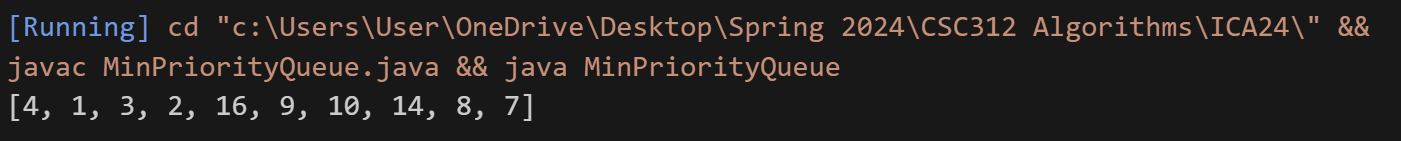
        mpq.MinHeapify(A, 1);

        System.out.println(Arrays.toString(A));

    }

}

Output:



All code works as intended!