Gebze Technical University
Computer Engineering

CSE 222 - 2019 Spring

HOMEWORK 5 REPORT

MUHAMMET BURAK ÖZÇELİK
151044050

Course Assistant: ÖZGÜ GÖKSU

1 INTRODUCTION

1.1 Problem Definition

Our problem is to compare the pixels of a given image with each other in a separate thread according to certain rules and print the highest value on the screen.

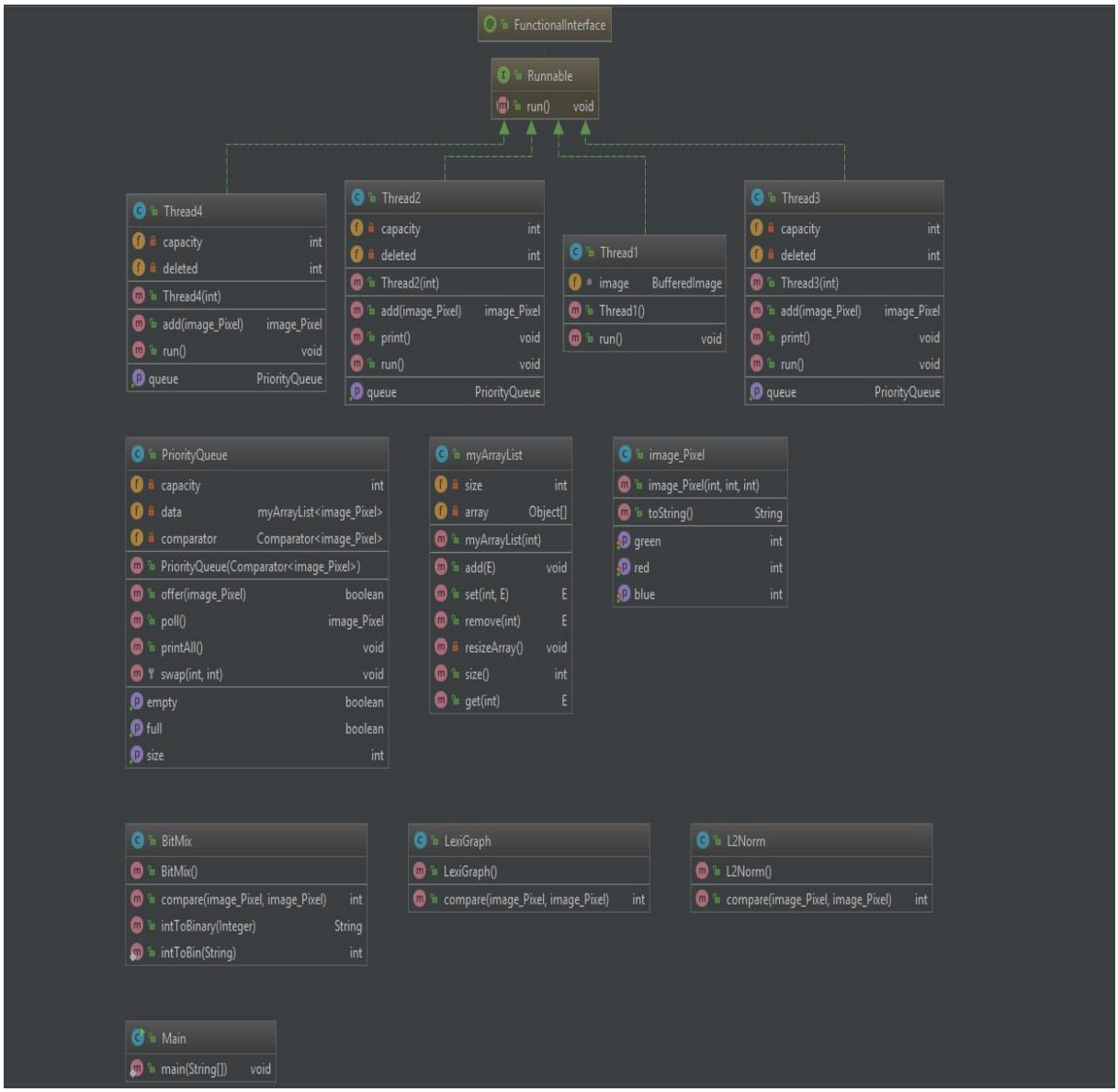
1.2 System Requirements

In this project, the user will be able to use the general priority queue (binary heap implementation) structure thanks to the class we have written.
L2 Norm, BitMix and LexiGraph classes within the poll and offer this priority queue in both the insertion and the maximum pixel comparison with the screen to print.
On the back of the work we will get this output by applying the compare method of the comparator class.
Project works at intelliJ on PC.

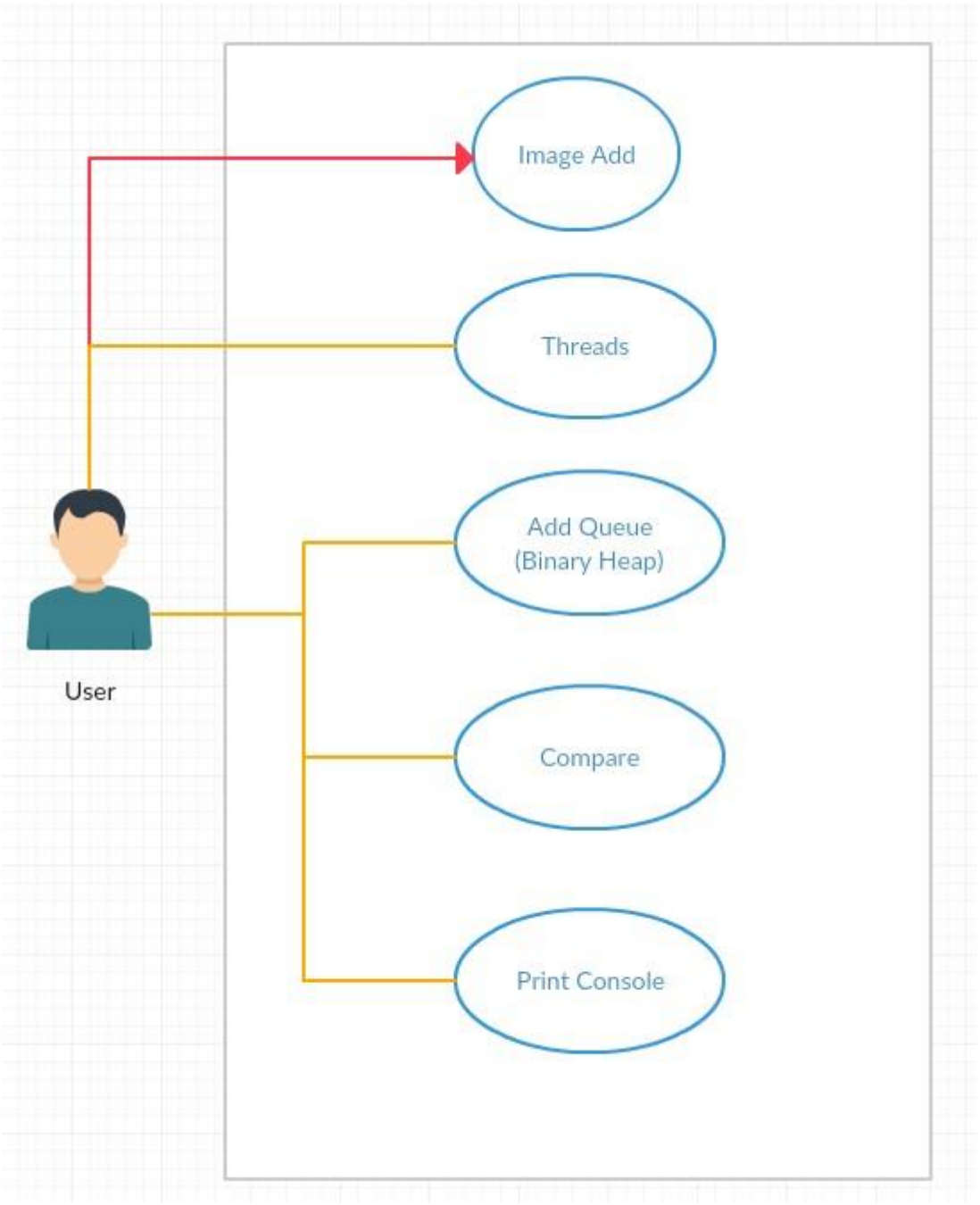
2 METHOD

2.1 Class Diagrams

Threads and priority queues are the classes we are asked to do from us.
Lexi Graph, L2Norm and BitMix classes are classes with comparator, each with a separate comparison.

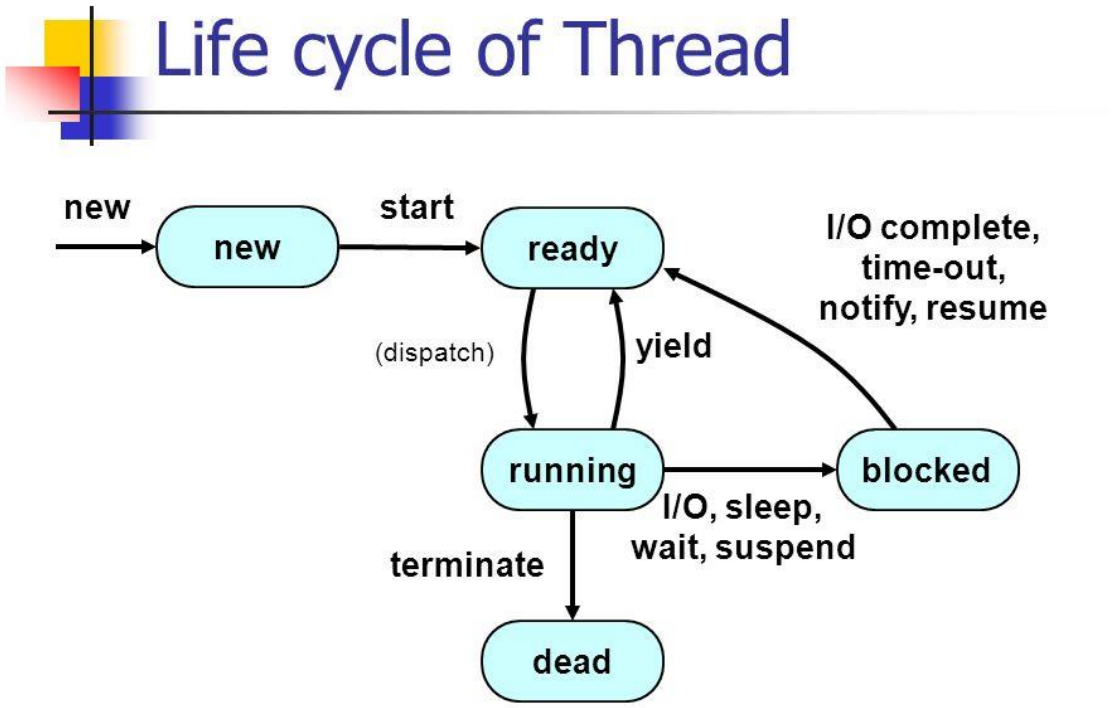

FunctionalInterface
Runnable
run() void
Thread4
capacity int
deleted int
Thread4(int)
add(image_Pixel) image_Pixel
run() void
queue PriorityQueue
Thread2
capacity int
deleted int
Thread2(int)
add(image_Pixel) image_Pixel
print() void
run() void
queue PriorityQueue
Thread1
image BufferedImage
Thread1()
run() void
Thread3
capacity int
deleted int
Thread3(int)
add(image_Pixel) image_Pixel
print() void
run() void
queue PriorityQueue
PriorityQueue
capacity int
data myArrayList<image_Pixel>
comparator Comparator<image_Pixel>
PriorityQueue(Comparator<image_Pixel>)
offer(image_Pixel) boolean
poll() image_Pixel
printAll() void
swap(int, int) void
empty boolean
full boolean
size int
myArrayList
size int
array Object[]
myArrayList(int)
add(E) void
set(int, E) E
remove(int) E
resizeArray() void
size() int
get(int) E
image_Pixel
image_Pixel(int, int, int)
toString() String
green int
red int
blue int
BitMix
BitMix()
compare(image_Pixel, image_Pixel) int
intToBinary(Integer) String
intToBin(String) int
LexiGraph
LexiGraph()
compare(image_Pixel, image_Pixel) int
L2Norm
L2Norm()
compare(image_Pixel, image_Pixel) int
Main
main(String[]) void

2.2 Use Case Diagrams



2.3 Problem Solution Approach

Let us consider the thread structure added below to better understand some stages.



What we want to do is to use this structure to reach the largest pixel in each thread with a priority queue. The first thread reads the pixels in our image through the java.awt.image.BufferedImage library and fills them into priority queues for use in other threads. After the first 100 pixels are filled, the second, third and fourth threads start to work. The main priority here is the parallel operation of all threads. Our second, third, and fourth threads will work until our priority queues are empty. These queues, sorted by the Compare method, allow us to print the screen by removing the highest value pixel with the poll method.

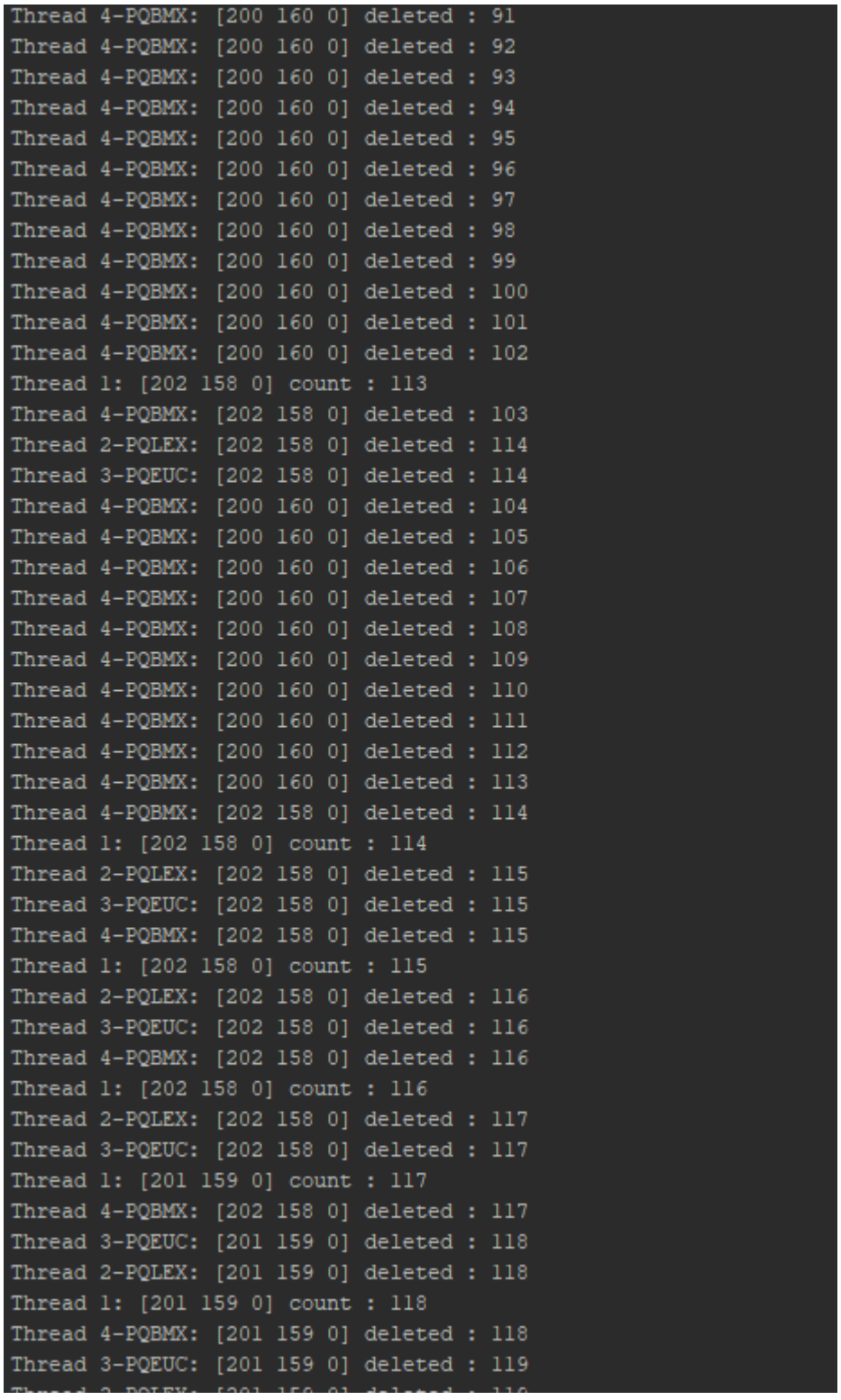
3 RESULT

3.1 Test Cases

I tested my program with ”example.png”. There are 360k pixels but I tested it until 200 pixels on this image and threads works correctly. Test results is given 3.2 section Running Results.

3.2 Running Results

```
Thread 1: [202 158 0] count : 90
Thread 1: [202 158 0] count : 91
Thread 1: [202 158 0] count : 92
Thread 1: [202 158 0] count : 93
Thread 1: [201 159 0] count : 94
Thread 1: [201 159 0] count : 95
Thread 1: [201 159 0] count : 96
Thread 1: [201 159 0] count : 97
Thread 1: [200 159 0] count : 98
Thread 1: [200 159 0] count : 99
//... Inserting all the way to at least first 100 pixels..
Thread 1: [202 158 0] count : 100
Thread 1: [202 158 0] count : 101
Thread 1: [202 158 0] count : 102
Thread 1: [202 158 0] count : 103
Thread 1: [202 158 0] count : 104
Thread 1: [202 159 0] count : 105
Thread 1: [201 159 0] count : 106
Thread 1: [201 159 0] count : 107
Thread 1: [201 159 0] count : 108
Thread 1: [200 159 0] count : 109
Thread 1: [202 158 0] count : 110
Thread 1: [202 158 0] count : 111
Thread 1: [202 158 0] count : 112
Thread 2-PQLEX: [202 159 0] deleted : 1
Thread 2-PQLEX: [202 159 0] deleted : 2
Thread 2-PQLEX: [202 158 0] deleted : 3
Thread 2-PQLEX: [202 158 0] deleted : 4
Thread 3-PQEUC: [202 159 0] deleted : 1
Thread 2-PQLEX: [202 158 0] deleted : 5
Thread 3-PQEUC: [202 159 0] deleted : 2
Thread 2-PQLEX: [202 158 0] deleted : 6
Thread 3-PQEUC: [202 158 0] deleted : 3
Thread 2-PQLEX: [202 158 0] deleted : 7
Thread 3-PQEUC: [202 158 0] deleted : 4
Thread 2-PQLEX: [202 158 0] deleted : 8
Thread 3-PQEUC: [202 158 0] deleted : 5
Thread 2-PQLEX: [202 158 0] deleted : 9
Thread 3-PQEUC: [202 158 0] deleted : 6
Thread 2-PQLEX: [202 158 0] deleted : 10
```



3.3 Analyze Time Complexity

- **myArrayList Implementations**

```java
import java.util.Arrays;
public class myArrayList<E> {
    private int size = 0;
    private Object array[];

    public myArrayList(int capacity) { array = new Object[capacity]; }

    public void add(E e) {
        if (size == array.length) {
            resizeArray();
        }
        array[size++] = e;
    }
    public E set(int index, E newVal) {
        if (index < 0 || index >= size()) throw new ArrayIndexOutOfBoundsException();
        E old = (E) array[index];
        array[index] = newVal;
        return old;
    }
    public E remove(int index) {
        E removeItem = (E) array[index];
        for (int i = index; i < size() - 1; i++) {
            array[i] = array[i + 1];
        }
        size--;
        return removeItem;
    }

    private void resizeArray() {
        int newSize = array.length * 2;
        array = Arrays.copyOf(array, newSize);
    }
    public int size() { return size; }
    /unchecked/
    public E get(int i) {
        if (i>= size || i <0) {
            throw new IndexOutOfBoundsException("Index: " + i + ", Size " + i );
        }
        return (E) array[i];
    }
}
```

Add, set, get, size, method's complexities are Θ (1) but remove and resizeArray method's are Θ (n).

- **PriorityQueue Implementation**

```java
public int getSize() { return size; }

public boolean offer(image_Pixel item) {
    data.add(item);
    size++;
    int child = data.size()-1;
    int parent = (child - 1) / 2;
    while(((comparator.compare(data.get(parent),data.get(child)) < 0))){
        swap(parent,child);
        child = parent;
        parent = (child - 1) / 2;
    }
    return true;
}

public image_Pixel poll() {
    if(isEmpty()) {
        return null;
    }
    image_Pixel result = data.get(0);

    if(data.size() == 1) {
        data.remove( index: 0);
        size--;
        return result;
    }
    data.set(0,data.remove( index: data.size() - 1));
    int parent = 0;

    while(true) {
        int leftChild = 2 * parent + 1;
        if(leftChild >= data.size()) {
            break;
        }
        int rightChild = leftChild + 1;
        int maxChild = leftChild;

        if(rightChild < data.size() && comparator.compare(data.get(leftChild), data.get(rightChild)) < 0) {
            maxChild = rightChild;
        }
        if(comparator.compare(data.get(parent),data.get(maxChild)) < 0) {
            swap(parent,maxChild);
            parent = maxChild;
        }
        else {
            break;
        }
    }

    size--;
    return result;
}
```

Offer method's complexity is Θ(1) because add method is Θ(1), compare method is Θ(1), swap method is Θ(1).

Poll method's complexity is Θ(1) because this always find maxChild and swap its location to the root (array[0]) or best case size is 1 it returns array[0]. The biggest value is always at array[0].

```
public boolean isEmpty() {
    if(size == 0){
        return true;
    }
    return false;
}
protected void swap(int index1, int index2) {
    image_Pixel tmp = data.get(index1);
    data.set(index1, data.get(index2));
    data.set(index2, tmp);
}
public int getSize() { return size; }
```

isEmpty, swap, method's complexities are Θ(1).

- Threads complexities are Θ(n), they run until the image data's end.
- Compare Methods Complexities

```
public class L2Norm implements Comparator<image_Pixel> {

    public L2Norm(){

    }
    @Override
    public int compare(image_Pixel o1, image_Pixel o2) {
        double temp1 = Math.sqrt(Math.pow(o1.getRed(),2)+Math.pow(o1.getGreen(),2)+Math.pow(o1.getBlue(),2));
        double temp2 = Math.sqrt(Math.pow(o2.getRed(),2)+Math.pow(o2.getGreen(),2)+Math.pow(o2.getBlue(),2));

        if(temp1 - temp2 > 0)
            return 1;
        else if( temp1 - temp2 == 0){
            return 0;
        }
        else {
            return -1;
        }
    }
}
```

Θ(1).

```
public class LexiGraph implements Comparator<image_Pixel> {

    @Override
    public int compare(image_Pixel o1, image_Pixel o2) {
        if(o1.getRed() == o2.getRed()){
            if(o1.getGreen() == o2.getGreen()) {
                return (o1.getBlue() - o2.getBlue());
            }
            else {
                return (o1.getGreen() - o2.getGreen());
            }
        }
        else {
            return (o1.getRed() - o2.getRed());
        }
    }
    public LexiGraph() {

    }
}
```

Θ(1).

```java
@Override
public int compare(image_Pixel o1, image_Pixel o2) {
    String s1_red = intToBinary(o1.getRed());
    String s1_green = intToBinary(o1.getGreen());
    String s1_blue = intToBinary(o1.getBlue());
    String s2_red = intToBinary(o2.getRed());
    String s2_green = intToBinary(o2.getGreen());
    String s2_blue = intToBinary(o2.getBlue());

    char[] cmp1 = new char[25];
    char[] cmp2 = new char[25];
    int a = 0;
    int b = 0;
    for (int i = 7; i >=0 ; i--) {
        cmp1[a++] = s1_red.toCharArray()[i];
        cmp1[a++] = s1_green.toCharArray()[i];
        cmp1[a++] = s1_blue.toCharArray()[i];

        cmp2[b++] = s2_red.toCharArray()[i];
        cmp2[b++] = s2_green.toCharArray()[i];
        cmp2[b++] = s2_blue.toCharArray()[i];

    }
    cmp1[a] = '\0';
    cmp2[b] = '\0';

    String str1 = new String(cmp1);
    String str2 = new String(cmp2);

    int int1 = intToBin(str1);
    int int2 = intToBin(str2);

    if(int1 - int2 > 0) {
        return 1;
    }
    else if( int1 - int2 == 0){
        return 0;
    }
    else if(int1 - int2 < 0){
        return -1;
    }
    return 0;
}

public String intToBinary(Integer x){
    String binary = Integer.toBinaryString(x);
    String output = String.format("%8s",binary).replace( oldChar: ' ', newChar: '0');
    return output;
}
```

```java
public static int intToBin(String str){
    double j=0;
    for(int i=0;i<str.length();i++){
        if(str.charAt(i)== '1'){
            j=j+ Math.pow(2,str.length()-1-i);
        }

    }
    return (int) j;
}
}
```

Θ(m). M is always constant. 8 bit binary value of decimal number.