GTU Computer Science and Engineering
CSE 222/505 – Spring 2019
Homework #05
Due Date: 09/04/2019 05:55 (yes, that's tuesday morning!)

CONTEXT: Imagine a color digital image of width W and height H. *You can use any online software library in order to load it into your program.* Once you load it, it will be represented as a 2D array (with W columns and H rows) of 3D 8bit unsigned integer valued vectors. Each element of this 2D array will correspond to a color point (or pixel) in this image. Each color pixel is made up of 3 integer values in the range 0-255, inclusive. The first dimension corresponds to the amount of red in that color, the second dimension to the amount of green in that color and the third dimension to the amount of blue in that color.

OBJECTIVE: we want to construct **max** priority queues for color pixels. We want to enter those pixels one by one into priority queues, and then extract them according to different priority schemes (or ordering relations).

INPUT: your program will admit as command line input a color digital image's path (in either png or jpg format).

HOW: Your program will support 3 vector (or color) comparison methods: lexicographical comparison (LEX), Euclidean norm based comparison (EUC) and bitmix comparison (BMX).
- LEX: standard lexicographical comparison from discrete math.
- EUC: whichever vector has the greater L2 norm is considered greater (this is actually a pre-ordering relation since it's not anti-symmetric but that's ok for this context).
- BMX: read section 3 of the attached pdf.

Your program will have 4 threads (study the online thread tutorial of Java on how to create and start them).

Thread 1: responsible for reading the pixels from the image starting from the top-left corner and advancing column first until the bottom-right pixel. Every color pixel read, will be inserted to each of the 3 priority queues (named PQLEX, PQEUC and PQBMX: one per ordering relation). As soon as the first 100 pixels are inserted, it will create and start the following 3 threads, and then continue inserting the remaining pixels.

Thread 2: will remove from PQLEX the color pixels and print them on screen one after the other, up until a total of WxH pixels are printed.

Thread 3: will do the same as Thread2 but it will use PQEUC.

Thread 4: will do the same as Thread2 but it will use PQBMX.

Note on threads 2, 3, 4: careful, if the PQ that is being processed by a thread happens to be temporarily empty, and the number of pixels it has processed so far is less than WxH, then it must wait **without keeping the CPU busy** (e.g. without constanly checking if there is an element in the priority queue). Hint: take a look into the producer consumer problem.

Note: if thread 1 tries to insert a value into a PQ run by thread X while thread X hasn't completed its removing method from it, all hell will break loose, and the PQ will become corrupt. Similarly, thread X must not try to remove from its PQ anything while thread 1 hasn't completed inserting into it. Hint: take a look into java's `synchronized` keyword.

OUTPUT: all your output will be at the terminal, threads 2, 3, 4, will print out WxH 3D color pixel values, in the order that they are extracted from their corresponding PQs, and thread 1 will print the pixel values in the order that they inserted:

e.g.

```
Thread 1: [100, 50, 200]
Thread 1: [78, 11, 111]
//... etc inserting all the way to at least the first 100 pixels..
Thread2-PQLEX: [255,45,178]
Thread 1: [99, 1, 77]
Thread4-PQBMX: [255,234,128]
Thread3-PQEUC: [255,247,198]
…
```

[The above values are imaginary].
It is up to the OS to determine which thread will run how many times before yielding its priority to another thread. Maybe thread 2 will run 300 times before switching to thread 1, or 200 times before switching to thread 4, etc. If everything goes fine, each thread must print out exactly WxH pixels.

RULES:
- code your own data structures (except for the image loading part).
- no plagiarism (except for the image loading part, steal what you can, leave nothing behind).
- **use a binary heap as the underlying implementation of your priority queue; bonus points if you implement a binomial or even more if you implement a fibonacci heap.**
- **You will have only one priority queue implementation, the constructor of which you will customize with a Comparator object parameter to make it prioritize accordingly the elements inside it.**
- don't use the `sleep` methods of the Thread class. Just don't, it's bad design.
- use OOP principles in your design.
- provide the class diagram in your report. Bonus points for using design patterns. Explain which you used and why.

PS:
• If you have any questions about the hw, please send an email to ogoksu@gtu.edu.tr

- Submit your homework as studentID.zip and includes the following files:
  - Project file
  - Report.pdf
- The implementation will be 75 points and the report is 25 points.
- Do not write any non-English words in your java file, your comments or your report!
- You have to implement all of them using your own code (unless otherwise specified), do not use any code from the internet or something else!

Good Luck!