American University of Armenia, CSE
CS121 Data Structures B
Spring 2022

Homework Assignment 6

Due Date: Saturday, April 23 by 23:59 electronically on Moodle

Please solve the programming tasks using Java, following good coding practices (details are on Moodle). Don't forget to submit the corresponding assignment report.

1. **(15 points)** Implement a generic class `MaxPriorityQueue<V>` that defines a maximum priority queue **efficiently** using a (minimum) `PriorityQueue` as the underlying data structure. Note that keys are characters (assuming there are 128 possible characters) and `V` denotes the type parameter for values. The implementation should rely on the *natural* ordering of characters, i.e. the largest character key should be prioritised most. The public methods of `MaxPriorityQueue<V>` should be similar to the ones for `PriorityQueue`, except instead of `min()` it should offer a `max()` method, and instead of `removeMin()`—a `removeMax()` method.

   Test it in a program. In case an `SortedPriorityQueue` instance is used for the underlying representation, what are the running times of the public methods of `MaxPriorityQueue<V>`? Briefly justify your answer.

2. **(15 points)** Define a `Person` class representing a person by their name, surname, patronymic and social security number (10 digit string). The class should only contain four String instance variables, a constructor taking four parameters and four accessors. Make this class comparable (using `Comparable` interface) based on the lexicographic order of the name, surname and patronymic in that order. That is, the person with a "smaller" name comes first in the ordering. If the first names of two persons are equal, then the person with a "smaller" surname comes first, and so on. In addition, define a comparator (using `Comparator` interface) for comparing `Person` objects based on their SSNs.

   Illustrate the use of both comparison approaches in a program by sorting an array of 6 persons with both approaches. Check out Java documentation to see how this can be done with `java.util.Arrays.sort` method.

3. **(20 points)** Write a recursive method that, given an array of entries and a comparator for the key-type, checks if the array represents a max-heap. Test your method in a program.

4. **(25 points)** Define a generic class `StackSortedPriorityQueue<K,V>` that implements a *sorted* priority queue, but uses a single stack for the underlying representation instead of a linked positional list. Test it in a program. Specify the space complexity and the running times of each of its public methods. You are not allowed to use additional containers in your methods.

5. (20 points) Write a program that takes as an input a sequence of integers (at most seven digits each) and prints these integers in the sorted order of their product of digits. If multiple integers have the same product of digits, only the first such integer is printed. Your program should use a priority queue to arrange the ordering of the integers and a (unsorted) map for the detection of equal product of digits. For the map, you can use `java.util.HashMap`. Test your program with an `UnsortedPriorityQueue`, a `SortedPriorityQueue`, and a `HeapPriorityQueue`; compare their performance on the same sequence of 1000 integers, recording actual execution times. The output format of printed values should look as follows:

$(key_1, value_1)$
$(key_2, value_2)$
...
$(key_n, value_n)$