American University of Armenia, CSE
CS121 Data Structures A, C
Fall 2021

Homework Assignment 6

Due Date: Sunday, November 28 by 23:59 electronically on Moodle

Please solve the programming tasks either in Java or C++, following good coding practices (details are on Moodle).

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 integers and `V` denotes the type parameter for values. The implementation should rely on the *natural* ordering of integers, i.e. the largest integer 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 `UnsortedPriorityQueue` instance is used for the underlying representation, what are the running times of the constructor and the other public methods of `MaxPriorityQueue<V>`? Briefly justify your answer.

2. **(15 points)** Define a minimalistic `Triangle` class representing a triangle with the integer lengths of its three sides. The class should only contain a constructor taking three parameters (one for each of the side lengths) and three accessors (one for each of the side lengths). Make this class comparable (using `Comparable` interface) based on the side lengths. That is, the triangle with a smaller shortest side comes first in the ordering. If the shortest sides of two triangles are equal, then the triangle with a smaller second-shortest side comes first, and so on. In addition, define a comparator (using `Comparator` interface) for comparing `Triangle` objects based on their areas, i.e. the triangle with a smaller area comes first.

   Illustrate the use of both comparison approaches in a program by sorting an array of 5 triangles with both approaches. Check out Java documentation to see how this can be done with `java.util.Arrays.sort` method.

3. **(18 points)** Define a generic class `ArrayListSortedPriorityQueue<K,V>` that implements a *sorted* priority queue, but uses an array list 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. Try to speed up the method implementations as much as possible, e.g. improvements of lower terms can sometimes be significant enough.

4. **(19 points)** Write a `heapsort` method that implements **in-place** heapsort for a given array list of entries with character keys. Make sure you use the array list efficiently.

5. **(15 points)** Write a method that takes an array arr of n entries and a comparator for the key-type and produces an array of k largest entries in arr, based on their keys. The execution time of your algorithm should be $O(n + k \log n)$ and it should use the logic of a max-heap.

6. **(18 points)** Write a program that reads a sequence of integers and prints these integers in the sorted order of their sums of digits. If multiple integers have the same sum 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 sums 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.