CS5800: Algorithms Spring 2018
Assignment 2

Out: 19 January 2018 **Due: 29 January 2018, 8:59pm**

Instructions:

- The assignment is due at the time and date specified. Late assignments will not be accepted.
- You must work on all the problems and write the solutions by yourself! Finding solutions to homework problems on the web, or by asking students in and outside the course is strictly prohibited. This would be defeat the purpose of learning by doing the assignment.
- You must submit typed solutions. You may use plain text or a word processor like Microsoft Word or LaTeX for your submissions. You may hand-sketch and scan any diagrams that you support your answer.
- If you are not comfortable with Word or Latex, please solve these problems by hand before devoting time to typing them. Do not waste precious time investigating typesetting up front!

1. (20 points) Solve the following recurrence relations using any method (substitution, recursion tree or Master theorem):

(a) $\mathrm{T(n)} = 3\mathrm{T}(\frac{n}{4}) + \sqrt{n}$, $\mathrm{T}(1) = 1$

(b) $\mathrm{T(n)} = 9\mathrm{T}(\frac{n}{3}) + 5n^2$, $\mathrm{T}(1) = 1$

(c) $\mathrm{T(n)} = \mathrm{T}(\frac{n}{3}) + \mathrm{T}(\frac{2n}{3}) + cn^2$, $\mathrm{T}(1) = 1$

2. (20 points) All airlines are mandated to conduct an additional security check at the departing gate for each international flight headed to USA. This check involves a passport and document check. After checking, passengers queue to board the aircraft.

In order to perform the security check, there are k security counters, each with its own queue. Due to space constraints, only up to n passengers can stand in a queue. The airline determines the value of k depending on equipment, size of gate, etc. Each incoming passenger has a priority, according to different things (e.g. family size, enrollment in the federal verification program, etc.). In each queue, passengers stand in decreasing order of priority (highest priority first, and so on).

The process is as follows: airline officials span a queue, checking documents. After all queues have been processed, passengers are released into the secure area of the gate where they form a single line to board the aircraft. This line is also arranged in decreasing order of priority.

You are a programmer employed by the airline, tasked with performing a discrete-event simulation of this process. Your aim is to specifically simulate the process of forming a single boarding queue from the security queues. You use arrays to represent the queues.

(a) A simple algorithm is to merge the first queue with the second, and then the result with the third and so on. This way ensures that only some queues are released at a time. Perform an efficiency analysis of the time taken by this algorithm.

(b) Provide an alternative algorithm that can create the boarding queue from the security queues in $O(kn \lg k)$ time. Your algorithm can be in pseudo-code form, or a numbered sequence of clear, concise steps. You do not need to provide a proof of correctness of the algorithm, but must prove that it works in the given time.

3. (20 points) You are provided with a set of n intervals (s_i, t_i), such that $s_i < t_i$. You can further assume that all s_i and t_i are distinct. Provide an algorithm that determines whether any two intervals overlap each other, that runs in $O(n \lg n)$ time.

(a) Write the algorithm in pseudo-code or a numbered sequence of clear, concise steps.

(b) Prove that your algorithm works correctly.

(c) Prove an analysis of the efficiency of your algorithm in time and space.

(Hint: use divide-and-conquer)

4. (25 points) The mergesort algorithm from class can be termed as a "top-down" algorithm: start with the original array and keep dividing it into smaller sub-arrays, and merge on your way up. One can visualize this in the form of a recursion tree whose root is the original array, intermediate nodes are subarrays and leaves are 1-element pieces. Thus, this algorithm starts at the root and recurses to the leaves, merging on its way back up the tree.

An alternative is to start at the leaves. We complete all merges of the leaves into 2-element subarrays, and then go a level up the tree. We then merge all subarrays at that level, before climbing up a level. This process continues until we merge two half-arrays into an array of the original size. Effectively this algorithm is a sequence of strategic merges. This is a "bottom-up" version of the same algorithm.

You must implement this algorithm in Java. Some starter code and data sets have been provided to you. The code is a program ready to be run (Type "javac SortCheck.java" to compile and "java SortCheck -i <name-of-input-file>" to run with any of the provided input files).

Your objective is to implement this algorithm so that the program uses your implementation instead of what is currently using. The program automatically verifies whether the sorting was correctly performed.

You will be graded on (a) correctness of your implementation (b) neatness of your code (indentation, proper use of variable names, etc.) (c) documentation, in the form of useful comments above and in the sorting method, and (d) answering the following questions about this algorithm.

(i) Provide a proof that this bottom-up algorithm also works in $O(n \lg n)$ time

(ii) Provide at least one advantage and one disadvantage of this algorithm vs the traditional top-down version. Your advantages should make sense for a programmer who is trying to choose between these alternatives. Assume that the data sets are large.