Introduction to Algorithms
Fall 2022 Take-home Exam
Due: 19/11/2022

1. (20 points) Every patient who want to go to the Akdeniz University hospital on a particular day must make an appointment beforehand. For each day, patients with appointments are given distinct sequential patient numbers starting from 1 so that the patient with the first appointment gets the patient number 1, the second one gets 2 and so on. Unfortunately, the patients come to their appointments in a totally random order. Also, some patients do not come to their appointments. Assume that n patients got patient numbers on a particular day and one of them did not come to his/her appointment. Patient numbers of patients that showed up are stored in an array A of size $n - 1$ where the patient numbers are ordered with respect to their arrival times. We want to design an $O(n)$ complexity algorithm to find the patient with the lowest patient number who did not show up for his/her appointment. For example, let $n = 12$ with $A = [12, 7, 10, 9, 5, 4, 11, 1, 6, 8, 2]$. Then, the algorithm must return 3.
   (a) Understand the problem, design an algorithm satisfying the $O(n)$ complexity, write the pseudocode of your algorithm and then implement it as a method in Java programming language.
   Hint: Use divide-and-conquer. Try to collect smaller patient numbers to the left and bigger numbers to the right. In your solution, you MUST utilize and modify the **Partition** algorithm that is used by QuickSort (solutions that do not use partitioning and recursion, such as subtracting the sum of the elements in A from the sum of the integers up-to n will not get any point). You should distribute the patient numbers to both partitions in a clever way so that the complexity could be $O(n)$. You should enforce the Partition algorithm to partition almost from the half by assigning a tricky pivot each time the modified Partition is called. If needed, you can also let the **Partition** algorithm take additional parameter(s). After partitioning, decide which part might have the missing patient and continue conquering that part recursively. Note that sorting the array will not help due to the time-complexity constraint. However, you will still need a recursive partitioning.
   (b) Formulate the recurrence of your algorithm and discuss why it is $O(n)$. Then, inside your main method, you will generate sequential arrays of sizes $m = 10K, 20K, 30K, 40K, 50K, 60K, 70K, 80K, 90K, 100K$ and randomly remove one element to obtain the corresponding A's. Then, you will run your method on each array by putting timers just before and after the function call to calculate a running time (i.e. take the difference of finish and start times). Finally, plot the calculated running times as a function of n and discuss whether the resulting plot is as expected.

2. (15 points) The YouTube video at http://www.youtube.com/watch?v=ywWBy6J5gz8 shows Hungarian Kukullomenti legenyes folk dancers implementing a version of QuickSort. Use this video to answer the following questions:
   (a) Implement the same Quicksort algorithm demonstrated by the folk dancers.
   (b) Generate 10 random array instances with sizes $n = 10K, 20K, 30K, 40K, 50K, 60K, 70K, 80K, 90K, 100K$. Plot the runtime of your Quicksort algorithm as a function of n. Again, you should put timers to previous and next lines of the line where call Quicksort and then, calculate the difference. Discuss whether the practical plot behaves according to the theoretical complexity.

3. (15 points) A bank office keeps record of its customers in an array $A[1...n]$ sorted by account numbers. However, a bank officer accidentally shifts the array elements $k < n$ positions to the right in a circular-fashion. Note that $k1$ and $k2$ are not necessarily constants (e.g. might be a function of n). For example, let $A = [9, 12, 17, 21, 33, 41]$. When shifted $k = 2$ positions to the right, A becomes $[33, 41, 9, 12, 17, m21]$. However, if $k = 4$, A becomes $[17, 21, 33, 41, 9, 12]$. The officer realizes the problem and wants to fix it by sorting the shifted version of the array with Insertion-sort. Find the complexity of the Insertion sort for such an input scenario. Generate random arrays of size $n = 2^{16}$ and shift the array for $k = 1, n/16, 2n/16, 3n/16, 4n/16, 5n/16, 6n/16, 7n/16, 8n/16, 9n/16, 10n/16, 11n/16, 12n/16, 13n/16, 14n/16, 15n/16, n - 1$ positions. For each k, measure the actual running times to sort the shifted array and organize the calculated times into a table. Discuss the results and compare the behaviors of the actual running times and the theoretical complexity.