CS 577: Introduction to Algorithms — Homework 1

Out: 02/03/2022 **Due: 02/17/2022**

Ground Rules

- Read the lecture notes and from the book: II Sorting and Statistics
- You should do all problems, but hand in only the graded problems.
- Write your name and student ID clearly on your submission.
- Clearly mark the beginning and end of your solution to each problem.

1. There are two algorithms whose running time complexity functions are $f(n)$ and $g(n)$ respectively. They both approach ∞ when $n \to \infty$.

   Part (A) Suppose $f(n)$ and $g(n)$ satisfy the following asymptotic relation

   $$\lim_{n\to\infty} \frac{\log f(n)}{\log g(n)} < \infty.$$

   Bob said, by this relationship, we can conclude that $f(n) = O(g(n))$. Is Bob right? Prove it or give a counter example for $f(n)$ and $g(n)$.

   Part (B) Suppose we have further found out that,

   $$\lim_{n\to\infty} \frac{\log f(n)}{\log g(n)} = 0.$$

   Alice said, by this relationship, we can conclude that $f(n) = o(g(n))$. Is Alice right? Prove it or give a counter example.

2. **Part (A)** In the code for InsertionSort we presented in class (as well as in the book, p. 18) the outer loop index j goes from 2 to n, the length of the input array A, and the inner index i of the while loop goes "backward".

   Write a version of the pseudocode for InsertionSort where the outer index j goes "backward", and the inner index i goes "forward".

   What is the inductive statement about your program that leads to the proof of its correctness? Prove your program correct.

   Part (B) Referring to SuperSort of hw0, after the first two recursive calls of SuperSort (for the first 2/3 and then the last 2/3 of the array), one insight (that led to the correctness of SuperSort) is that the largest 1/3 of elements of the whole array are now placed in the last 1/3 places, and are in increasing order.

   Now there is another property that exists for the first 2/3 of the array after these first two recursive calls of SuperSort. Use this property to modify SuperSort by replacing the last (the third) recursive call to SuperSort (with something else), and achieve a better asymptotic complexity. What is the asymptotic complexity of your modified SuperSort? Prove it.

3. Given an array $A[1:n]$ of integers, we say a pair of indices (i, j) is an *inversion* if $i < j$ and $A[i] > A[j]$.

   Obviously the running time of InsertionSort on a particular array $A[1:n]$ crucially depends on the number of inversions in A. Prove that the number of comparisons (on array elements) that InsertionSort(A) makes is

   $$\text{(The the number of inversions of } A) + \Theta(n)$$

   This quantity can certainly be smaller than $O(n^2)$. Does the statement above contradict the statement that InsertionSort is an $O(n^2)$ sorting algorithm? Why or why not?

   Suppose someone claims that if the first half and the second half of an input A are already sorted, then InsertionSort(A) must run in $o(n^2)$. Is this claim correct? Why or why not?

4. **Graded Problem (Page limit: 1 sheet; 2 sides)** You are given two sorted lists A and B with n distinct numbers in each. Develop a divide and conquer algorithm for finding the median[1] of the union of the lists in $O(\log n)$ time. Note that merging the lists would take $\Theta(n)$ time, so that would not be a good idea.

   Extend your algorithm to finding the kth smallest element in the union of the two (individually sorted) lists, where k is some integer in $\{1, \ldots, 2n\}$.

5. **Graded Problem (Page limit: 1 sheet; 2 sides)**

   The Towers of Hanoi puzzle has 3 towers labeled 0, 1, and 2. There are n pegs of sizes 1 to n, initially placed on tower 0 from top to bottom in increasing order of their sizes. The goal of the well-known puzzle is to move all n pegs to another tower, say tower 2. One can only move one peg at a time, and move the top peg on a tower to be placed on the top of another tower, and cannot place a larger peg on top of a smaller peg. The question is, what is the smallest number of moves needed.

   Our problem is a variant of the Towers of Hanoi puzzle: The three towers are still labeled 0, 1, and 2. There is one additional constraint: You are only allowed to move pegs from a tower labeled i to the next one labeled $(i+1) \mod 3$. You can imagine the towers as being arranged in a circular fashion, in which case, the constraint says that you can only move pegs in an anti-clockwise manner. So, for example, in order to move a peg from tower 0 to tower 2, you would have to first move it to tower 1 (if both $0 \to 1$ and $1 \to 2$ are legal moves at this point), and this sequence would cost two moves instead of just one.

   (a) Prove that starting at any legal configuration, there is at least one legal move.

   (b) Inductively prove that for any $n \geq 1$, this variant of the Towers of Hanoi puzzle (the goal is still moving all n pegs from tower 0 to tower 2) has a solution (consisting of finitely many legal moves.)

   (c) Give a recurrence of $T(n)$, the least number of moves needed for this variant of the Towers of Hanoi puzzle with n pegs.

   (Hint: You may find it useful to also introduce a quantity $S(n)$ for the least number of moves needed to move n pegs from tower 0 to tower 1.)

   (d) Prove inductively that $T(n) \geq (\sqrt{3}+1)^{n-1}$, for all $n \geq 1$.

6. You are given $n = 3^k$ coins, for some $k \geq 1$. All the coins look identical. However, one of the coins is defective – it weighs either slightly more or slightly less than a normal coin (you don't know which). You also have at your disposal a tester, which has two compartments. You may place any two disjoint subsets of coins in them, the tester tells you the result of: either two sets weigh equally, or the first subset weighs less than the second subset, or the opposite. Your goal is to determine which coin is defective, and to tell whether it is heavier or lighter than the normal one.

   (a) How many possible outcomes are there in a given problem on $n = 3^k$ coins?

   (b) Design an algorithm for this problem using at most $k+1$ tests.

   (c) Prove the correctness of your algorithm, in other words that it always correctly solves the problem.

[1] We define the median of a list of length m, where m is even, as the $m/2$th smallest element.