CS 577: Introduction to Algorithms — Homework 1

Out: 09/12/17 — **Due: 09/21/17**

Ground Rules

- You should try to solve all problems, not just the graded ones. There will be discussions of homework problems recitation sessions.
- Graded problems are to be turned in on the due date at the beginning of the lecture.
- Graded problems should be done in pairs.
- Please follow page limits for all graded problems.
- Write both your names clearly on your submissions, and turn in each problem on **a separate sheet of paper**.

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

2. 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 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?

   Can you improve the expression $\Theta(n)$ to something more precise?

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

3. 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 lists, where k is some integer in $\{1, \ldots, 2n\}$.

4. **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 the order of their sizes. The goal 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

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

says that you can only move pegs in an anti-clockwise manner. So, for example, in order to move a disk 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 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 $n \geq 1$.

(e) (extra credit) Can you prove an upper bound for $T(n)$?

5. 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.