CS 344 - Sections 1,2,3 - Fall 2018
Homework 2.
Covers lecture on Sep 12,17,19,24,26
Due Oct 3.
80 points total plus extra credit

1 Problem 1 (30 points total)

For all the recursive formula problems below, you can assume that $T(2) = 2$.

- **Part 1:** (10 points) Say that you have an algorithm with recurrence formula $T(n) = 4T(n/2) + n^2$. Use a recursion tree to figure out the running time of your algorithm. Then use a proof by induction to prove that your answer is correct.

- **Part 2:** (10 points) Consider an algorithm that solves a problem of size n by dividing it into 3 pieces of size $n/3$, recursively solving each piece, and then combining the solutions in $O(n^3)$ time. What is the recurrence formula for this algorithm? Use a recursion tree to figure out the running time of the algorithm (in big-O notation, as always), and then check your result via a proof by induction.

- **Part 3:** (10 points) Say that you have an algorithm with recurrence formula $T(n) = 4T(n/3) + n$. Use a recursion tree to figure out the running time of your algorithm. Then use a proof by induction to prove that your answer is correct. (For the proof by induction, you will want to show $T(n) \leq$ C([something] - n), as I briefly mentioned in class.)

2 Problem 2 (25 points total)

Consider the $O(n)$-time SELECT(A,k) algorithm covered in class. In that algorithm, we split the array into chunks of size 5, found the median of each, recursively found the median of medians, partitioned around that median, and recursed onto the appropriate side. The corresponding recursive formula was $T(n) \leq T(n/5) + T(7n/10) + O(n)$.

Now, consider a modification of this algorithm where we initially split the array into chunks of size 9 instead of 5, and then proceed in the same fashion (i.e. recursively find median of medians, and so on.)

- **Part 1 1: (20 points)** What is the recursive formula for this modified algorithm? You must justify this new formula – i.e. you must show why subproblem-sizes are bounded according to the formula.

- **Part 2: (5 points):** Prove via induction that the algorithm has running time $O(n)$.

3 Problem 3 (25 points total)

Let A be some array of n numbers, with no duplicated elements, and recall that Rank(x) = k if x is the kth *smallest* element of A. Now define InverseRank(x) = n + 1 - Rank(x). It is easy to see that if InverseRank(x) = k, then x is the kth *largest* element of A.

Now, let us define a number x in A to be *special* if x = InverseRank(x). For example, if $A = -9, 8, 1, -1, 2$, then 2 is special because 2 is the 2nd largest number in the array, so InverseRank(2) = 2.

Consider the following problem:

- Input: array A of length n

- Output: return a special number x in A, or return "no solution" if none exists.

Questions:

- **Part 1 (5 points):** Give pseudocode for a $O(n \log(n))$ algorithm for the above problem.

- **Part 2 (20 points):** Give pseudocode for a $O(n)$ recursive algorithm for the above problem. Give a brief justification for why the algorithm is correct. The recurrence formula for your algorithm should be $T(n) \leq T(n/2) + O(n)$, which we proved in class solves to $T(n) = O(n)$.

  (HINT 1: write pseudocode for an algorithm FindSpecial(A, offset), which looks for a number x such that x = InverseRank(x) + offset. In the initial call you just have offset = 0, but as you recurse, you will want to change the offset value.)

  (HINT 2: use the high-level approach of Median Recursion described in class. Think about how looking at the median will allow us to recurse onto one of the sides and ignore the other.)

4 Problem 4 – Extra Credit

Given an array A, we say that elements $A[i]$ and $A[j]$ are swapped if $j > i$ but $A[j] < A[i]$. For example, if $A = [8, 5, 9, 7]$, then there are a total of 3 swapped pairs, namely 8 and 5; 8 and 7; and 9 and 7.

Describe a recursive algorithm that given an array A, determines the number of swapped pairs in the array in $O(n \log(n))$ time. To analyze the algorithm, you must state the recurrence formula, and argue that $T(n) = O(n \log(n))$ either with a recursion tree OR a proof by induction – for this problem, I won't make you do both. (HINT: your algorithm should piggy back on merge sort, and then add additional functionality to count the number of inversions. In particular, the solution I have in mind sorts the array as a side effect of counting the number of swaps.)