CS161 Homework 1 **Due:** 15 April 2016, 12 noon
Submit on Gradescope **Handed out:** 8 April 2016

Instructions: Please answer the following questions to the best of your ability. If you are asked to show your work, please include relevant calculations for deriving your answer. If you are asked to explain your answer, give a short ($\sim$ 1 sentence) intuitive description of your answer. If you are asked to prove a result, please write a complete proof at the level of detail and rigor expected in prior CS Theory classes (i.e. 103). When writing proofs, please strive for clarity and brevity (in that order). Cite any sources you reference.

1 Lights On (12 points)

There are n people entering and exiting a room. For each $i \in \{1, \ldots, n\}$, person i enters at time a_i and exits at time b_i (assume $b_i > a_i$ for all i), and all the a_i, b_i are distinct. At the beginning of the day, the lights in the room are switched off, and the first person who enters the room switches them on. In order to conserve electricity, if person i leaves the room at time b_i and there is no one else present in the room at time b_i, then person i will switch the lights off. The next person to enter will then switch them on again. Given the values $(a_1, b_1), (a_2, b_2), \ldots (a_n, b_n)$, we want to find the number of times the lights get switched on.

Design the following algorithms, and prove the correctness and running time of each algorithm.

(a) A $\Theta(n^2)$ algorithm that computes the number of times the lights get switched on.

(b) An $O(n \log n)$ algorithm that computes the number of times the lights get switched on. (Hint: start by sorting the entry and exit times.)

2 Select with Other Group Sizes (15 points)

In the lecture, you have seen the linear time Algorithm Select that finds the k^{th} smallest element of an array of n elements in $O(n)$ time. The algorithm *mysteriously* divides n numbers into groups of 5 elements and proceeds. In this problem, we analyze Select with different group sizes other than 5.

In Parts (a) - (c), we consider groups of size $2\ell + 1$ for $\ell \geq 1$. In Part (d), we consider groups of size 6. You may assume that n given elements are distinct. You may also ignore all the floor and ceiling functions whenever they pop up.

(a) Assuming we divide n elements into groups of $2\ell + 1$, derive an upper bound on the number of elements greater than the median of medians; this number will be a function of ℓ. Do the same for the number of elements smaller than the median of medians.

(b) Derive a recurrence for the worst-case running time $T(n)$ in terms of ℓ. You can assume that the median of each group of size $2\ell + 1$ is found in $O(\ell^2)$ time.

(c) Using the substitution method, solve the recurrence from Part (b) to obtain $T(n) = O(n)$. Note that the substitution method will not work for all possible values of ℓ. State a sufficient condition on ℓ under which the substitution method works. What is the running time of the algorithm for the cases when the substitution method does not give $O(n)$?

(d) Assuming we divide the n elements into groups of 6, repeat Parts (a) - (c). The median of 6 elements is the 3rd smallest number.

3 Even Sub-arrays (15 points)

Suppose you are given an array of positive integers A of size n. For $i, j \in \mathbf{N}$ and $1 \leq i \leq j \leq n$, a sub-array of A is an array of the form $[A[i], A[i+1], \ldots, A[j]]$ where $i, j \in \mathbf{N}$, $1 \leq i \leq j \leq n$ and $A[i]$ refers to the ith element of A. A prefix of A is a sub-array of the form $[A[1], \ldots, A[i]]$ for $i \in \mathbf{N}$ and $1 \leq i \leq n$. A suffix of A is a sub-array of the form $[A[i], \ldots, A[n]]$ for $i \in \mathbf{N}$ and $1 \leq i \leq n$. The sum of a sub-array refers to the sum of all its elements. In this problem we will design a divide and conquer algorithm that counts the number of sub-arrays with *even* sum.

(a) A naive approach to this problem would consist of computing the sum of all possible sub-arrays and then returning the number of these of even sum. What would be the resultant running time of this algorithm?

(b) Now show how to compute the sums of all prefixes of A in $O(n)$ time. Given these values, show that the sum of any sub-array can be computed in $O(1)$ time. How long does it take now to go through all possible sub-arrays and return the ones of even sum?

(c) You are not satisfied with the running time of the last algorithm and realize that you can solve this problem faster using a divide-and-conquer approach. Now suppose that you divide A into two disjoint sub-arrays L and R of length $n/2$ [1] and determine the number of sub-arrays of even sum for both of these two sub-arrays recursively. Why is this information insufficient to compute the number of even sum sub-arrays of A? Be concise.

(d) Suppose now that your recursive algorithm computes and returns extra information beyond the number of even sum sub-arrays: what *constant* size extra information can be returned so that from the recursive solutions for L and R one can recover in $O(1)$ time (1) the number of even sum sub-arrays of A and (2) also the extra information to return when you pop out of the recursion for A? (Hint: Think about the number of prefixes and suffixes with even and odd sum in R and L.)

(e) Write a recurrence for your divide-and-conquer algorithm in part (d). Solve the recurrence using your favorite method. What is the running time of the algorithm?

4 Recurrences Galore (15 points)

Please solve the recurrences below giving tight upper-bounds of the form $T(n) \leq O(f(n))$ for an appropriate function f. You can use any method from class. Show your work. Note: Unless otherwise stated, log refers to base 2 log, and ln refers to natural log.

(a) $T(n) = 14 \cdot T(\frac{n}{3}) + n^2 \ln n$.

(b) $T(n) = 4 \cdot T(\frac{n}{4}) + n \cdot (\log n)^2$.

(c) $T(n) = 161^2 \cdot T(\sqrt[161]{n}) + 161 \cdot (\log n)^2$.

(d) $T(n) = \left(T(\frac{n}{161})\right)^{161} \cdot n$.

(e) $T(n) = 3 \cdot T(n/4) + n \log n$.

(f) $T(n) = T(\lfloor \sqrt[4]{n} \rfloor) + T(\lceil \sqrt{n} \rceil) + \log n$. When analyzing this one, you can ignore the floor and ceiling functions for simplicity.

[1] We will assume that n is a power of 2.