CS 141, Spring 2019 Midterm 1/A

First name:

Last name:

Student ID:

- This exam is **closed book, closed notes**, 80 minutes long
- Read the questions carefully
- No electronic equipment allowed (cell phones, tablets, computers, . . .)
- Write legibly. What can't be read will not be graded
- Use pseudocode, Python, or English to describe your algorithms (no C/C++/Java)
- When designing an algorithm, you are allowed to use any algorithm or data structure we explained in CS 141 or CS 14, without giving its details, unless the question specifically requires that you give such details
- Always remember to analyze the time complexity of your solution
- If you have a question about the meaning of a question, come to the front of the class

Problem 1. (15 points [writing/solving recurrence relations])

Solve exactly (that is, without using any asymptotic notation) the following recurrence relation by iterative substitutions

$$T(n) = \begin{cases} 1 & n = 1 \\ T\left(\frac{n}{4}\right) + \sqrt{n} & n > 1 \end{cases}$$

Problem 2. (15 points [writing/solving recurrence relations])

Using the Master method, give an asymptotic tight bound for $T(n)$ in the following recurrence relation

$$T(n) = \begin{cases} 1 & n = 1 \\ 3\,T\left(\frac{n}{9}\right) + \sqrt{n}\log n & n > 1 \end{cases}$$

Problem 3. (15 points [writing/solving recurrence relations])

In the algorithm SELECT described in class (linear-time selection), the input elements are divided into $\lceil n/5 \rceil$ groups of 5. Suppose you modify the algorithm to divide the input elements into $\lceil n/7 \rceil$ groups of 7 instead. Let $T(n)$ denote the worst-case running time of the modified algorithm as a function of the input size n. Write a recurrence relation for $T(n)$, but do NOT solve it.

Problem 4. (20 points [divide & conquer])

An array A is said to have a *majority element* if half or more than half of the entries in A are exactly the same. Given an unsorted array $A[1 \ldots n]$ of n items (where n is a power of two) we want to determine whether A has a majority element, and if so, return such an item. Consider the following divide and conquer algorithm for this problem.

Algorithm FIND-MAJORITY (A : **array**)
1 $n \leftarrow |A|$
2 **if** $n \leq 1$ **then return** $A[1]$
3 **else**
4 $a_1 \leftarrow$ FIND-MAJORITY$(A[1, \ldots, n/2])$
5 $a_2 \leftarrow$ FIND-MAJORITY$(A[n/2+1, \ldots, n])$
6 **if** the number of times a_1 appears in A is $\geq n/2$ **then return** a_1
7 **else if** the number of times a_2 appears in A is $\geq n/2$ **then return** a_2
8 **else return** NULL

Is this algorithm correct i.e., does FIND-MAJORITY always return the majority element, if A has one in it (or NULL otherwise)? Give a counterexample if your answer is "No", a brief argument of correctness (e.g., a proof by induction on n, the size of A) if your answer is "Yes".

Problem 5. (15 points [divide & conquer])

Write the pseudo-code for the $O(n \log n)$-time algorithm for closest pair that we described in class.

Problem 6. (20 points [divide & conquer])

Suppose you have k sorted arrays, each with n elements, and you want to combine them into a single sorted array of kn elements. Describe a divide-and-conquer algorithm that takes $O(kn \log k)$ time. Make sure you explain why your algorithm runs in $O(kn \log k)$ time.

Hint: Use as a sub-routine ("black-box") the MERGE algorithm we described in class to merge two sorted arrays with n elements in $O(n)$-time.