• This exam is **closed book, closed notes**, 50 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. (25 points [writing/solving recurrence relations])

A CS 141 student has been trying to speed-up Karatsuba’s divide-and-conquer integer multiplication algorithm. Given two numbers $x, y$ with $n$ bits each, her/his algorithm (1) first divides both $x$ and $y$ into four equal-length pieces, then (2) expresses the product $x \cdot y$ using $p$ multiplications of these $n/4$-bit pieces, followed by a constant number of additions, subtractions and shifts. How small $p$ needs to be in order to give a faster algorithm than the Karatsuba’s algorithm covered in class? You can assume $n$ to be a power of 4, and $p > 4$. Justify your answer using the Master Theorem. Note: Use the fact that $\log_2 3 = \log_4 9$. 
Problem 2. (25 points [writing/solving recurrence relations])

In the algorithm SELECT described in class (linear-time selection), the input elements are divided into $\lceil \frac{n}{5} \rceil$ groups of 5. Suppose you modify the algorithm to divide the input elements into $\lceil \frac{n}{7} \rceil$ 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 3. (18 points (divide & conquer))

In homework 3 you have solved the problem of the majority element using divide and conquer. This is a variant on that problem. We say that an array $A$ has a one-third-majority item if that item appears at least $n/3$ times in $A$. Given an unsorted array $A[1 \ldots n]$ of $n$ items we want to determine whether $A$ has a one-third-majority element, and if so, return such an item. If there is more than one one-third-majority element in $A$, we are OK with any one of them. Consider the following divide and conquer algorithm for this problem:

**Algorithm** Find-One-Third-Majority ($A :$ array)

1. $n \leftarrow |A|$  
2. if $n \leq 3$ then return $A[1]$  
3. Partition $A$ into $A_1, A_2, A_3$ where $A_1$ is the first third of $A$, $A_2$ is the second third of $A$, and $A_3$ is the last third of $A$  
4. $a_1 \leftarrow$ Find-One-Third-Majority($A_1$)  
5. $a_2 \leftarrow$ Find-One-Third-Majority($A_2$)  
6. $a_3 \leftarrow$ Find-One-Third-Majority($A_3$)  
7. if the number of occurrences of $a_1$ in $A$ is $\geq n/3$ then return $a_1$  
8. else if the number of occurrences of $a_2$ in $A$ is $\geq n/3$ then return $a_2$  
9. else if the number of occurrences of $a_3$ in $A$ is $\geq n/3$ then return $a_3$  
10. else return False

Is this algorithm correct i.e., does it always return a one-third-majority element, if $A$ has one in it (or FALSE otherwise)? Give a counterexample if your answer is "No", a brief argument of correctness if your answer is "Yes". Assume $n$ is a power of 3.
Problem 4. (25 points [divide & conquer])

Suppose you are given an array $A = \{a_1, a_2, \ldots, a_n\}$ of $n$ distinct integers. You are told that
the sequence of values $a_1, a_2, \ldots, a_n$ is unimodal, that is for some index $p \in [1, n]$, the values in the
array increase up to position $p$, and then decrease the remainder of the way until position $n$. Give
an algorithm to find the position $p$ in $O(\log n)$ time. You can assume $n$ to be a power of 2.