No.	Questions
1.	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.	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.
3.	Suppose that each row of an $n \times n$ array A consists of 1's and 0's such that, in any row of A, all the 1's come before any 0's in that row. Assuming A is already in memory, describe a method running in $O(n)$ time (not $O(n^2)$ time) for finding the row of A that contains the most 1's. Suppose that each row of an $n \times n$ array A consists of 1's and 0's such that, in any row i of A, all the 1's come before any 0's in that row. Suppose further that the number of 1's in row i is at least the number in row $i+1$, for $i = 0, 1, \ldots, n-2$. Assuming A is already in memory, describe a method running in $O(n)$ time (not $O(n^2)$ time) for counting the number of 1's in the array A.
4.	Give a recursive algorithm to compute the product of two positive integers m and n using only addition. Describe, using pseudocode, a method for multiplying an $n \times m$ matrix A and an $m \times p$ matrix B. Recall that the product $C = AB$ is defined so that $C[i][j] = \sum_{k=1}^{m} A[i][k] \cdot B[k][j]$. What is the running time of your method?
5.	Show that the running time of the merge-sort algorithm on an n-element sequence s $O(n \log n)$, even when n is not a power of 2.

	Suppose we modify the deterministic version of the quick-sort algorithm so that, instead of selecting the last element in an n-element sequence as the pivot, we choose the element at index $\lfloor n/2 \rfloor$, that is, an element in the middle of the sequence. What is the running time of this version of quick-sort on a sequence that is already sorted?
6.	Suppose we modify the deterministic version of the quick-sort algorithm so that, instead of selecting the last element in an n-element sequence as the pivot, we choose the element at index $\lfloor n/2 \rfloor$, that is, an element in the middle of the sequence. What is the running time of this version of quick-sort on a sequence that is already sorted? Consider again the modification of the deterministic version of the quick-sort algorithm so that, instead of selecting the last element in an n-element sequence as the pivot, we choose the element at index $\lfloor n/2 \rfloor$. Describe the kind of sequence that would cause this version of quick-sort to run in $\Theta(n^2)$ time.
7.	Suppose we are given an n-element sequence S such that each element in S represents a different vote in an election, where each vote is given as an integer representing the ID of the chosen candidate. Without making any assumptions about who is running or even how many candidates there are, design an $O(n \log n)$-time algorithm to see who wins the election S represents, assuming the candidate with the most votes wins.
8.	Suppose we are given two sequences A and B of n elements, possibly containing duplicates, on which a total order relation is defined. Describe an efficient algorithm for determining if A and B contain the same set of elements (possibly in different orders). What is the running time of this method? Suppose we are given a sequence S of n elements, on which a total order relation is defined. Describe an efficient method for determining whether there are two equal elements in S. What is the running time of your method?
9.	Bob has a set, A, of n nuts and a set, B, of n bolts, such that each nut has a unique matching bolt. Unfortunately, the nuts in A all look the same, and the bolts in B all look the same as well. The only comparison that Bob can make is to take a nut-bolt pair (a, b), such that $a \in A$ and $b \in B$, and test if the threads of a are larger, smaller, or a perfect match with the threads of b. Describe an efficient algorithm for Bob to match up all of his nuts and bolts. What is the running time of this algorithm?
10.	*A stack of fake coins* There are n stacks of n identical-looking coins. All of the coins in one of these stacks are counterfeit, while all the coins in the other stacks are genuine. Every genuine coin weighs 10 grams; every fake weighs 11 grams. You have an analytical scale that can determine the exact weight of any number of coins. **a.** Devise a brute-force algorithm to identify the stack with the fake coins and determine its worst-case efficiency class. **b.** What is the minimum number of weighings needed to identify the stack with the fake coins?