Due on February 4, 2016
125 points total

General Directions: If you are asked to provide an algorithm, you should clearly define each step of the procedure, establish its correctness, and then analyze its overall running time (for this assignment, this means arguing why your algorithm achieves the target running time specified by the question). There is no need to write pseudo-code; an unambiguous description of your algorithm in plain text will suffice.

Please make sure that you **START EARLY**. Some of these questions might take you several hours to solve. Since this is a mathematical/theoretical class, the ratio between how much time you spend thinking and how long it takes you to write your solution will be high (at least higher than what you might be used to in coding assignments).

All the answers must be typed, preferably using LaTeX. If you are unfamiliar with LaTeX, you are strongly encouraged to learn it. However, answers typed in other text processing software and properly converted to a pdf file will also be accepted. Before submitting the pdf file, please make sure that it can be opened using any standard pdf reader (such as Acrobat Reader) and your entire answer is readable. **Handwritten answers or pdf files that cannot be opened will not be graded and will not receive any credit.**

Finally, please read the detailed collaboration policy given on the course website. You are **not** allowed to discuss homework problems in groups of more than 3 students. **Failure to adhere to these guidelines will be promptly reported to the relevant authority without exception.**

For some questions, the following equations might be useful:

$$\sum_{i=0}^{\infty} r^i = \frac{1}{1-r} \text{ for } 0 < r < 1.$$

$$\sum_{i=0}^{n} r^i = \frac{1-r^{n+1}}{1-r} \text{ for } r > 0, r \neq 1.$$

Problem 1 (20 points)
(a) (5 points each) Let $f_1(n), f_2(n), g_1(n)$, and $g_2(n)$ be functions of $n \in \mathbb{Z}^+$ such that $f_1(n) = \Theta(f_2(n))$ and $g_1(n) = \Theta(g_2(n))$. First, show it is not necessarily true that

$$f_1(n) - g_1(n) = \Theta(f_2(n) - g_2(n)).$$

(So here, you need to explicitly define four functions that produce a counterexample to this implication). Then, prove the following is in fact true:

$$f_1(n)/g_1(n) = \Theta(f_2(n)/g_2(n)).$$

(b) (5 points each) Solve the following recurrences for their closed forms. Express your final answer as a tight asymptotic bound (i.e., give your answer as $\Theta(f(n))$).

You can solve these however you like, but it should be clear from your explanation how you obtained your answer. If you use the tree-expansion method, simply describe the structure of the tree in words — there is no need to include any kind of figure. You do not need to give the formal inductive proofs shown in class.

(i) $T(n) = 2T(\sqrt{n}) + \Theta(1)$

(ii) $T(n) = T(n/2) + \Theta(\sqrt{n})$

Problem 2 (20 points)
Professor Dumbledore (a long lost brother of the beloved headmaster) teaches algorithms in Magicland. After the first midterm exam, he wants to carefully scrutinize the students' performance, but unfortunately, the TAs handed him an unsorted array of scores $L = x_1, \ldots, x_n$ and left on vacation in Spring break. Having come to know of your prowess in algorithms after taking COMPSCI 330, he recruits you to come up with an algorithm to sort these scores. If there were only k distinct scores among all n scores, can you sort L in $O(n \log k)$ time? Note that this does not necessarily imply that the set of possible scores is $\{1, \ldots, k\}$. Also note that k is much smaller than n; therefore, an $O(n \log n)$ time algorithm will not suffice. (An $O(n \log k + k^2)$ solution will get partial credit.)

Problem 3 (25 points)
In class, we learned a linear time algorithm that finds the median of an array of numbers. We now wish to solve a more general problem using this algorithm. The input is an (unsorted) array of numbers $x_1, x_2, \ldots, x_n$, each with an associated positive weight (we denote the weight of x_i by w_i) such that the sum of weights is 1 (i.e., $\sum_{i=1}^{n} w_i = 1$). Suppose ϕ defines an ordering of indices such that $x_{\phi(1)}, x_{\phi(2)}, \ldots, x_{\phi(n)}$ is sorted. The *weighted median* of the numbers is the value $x_{\phi(m)}$ such that $\sum_{i=1}^{m-1} w_{\phi(i)} < 0.5$ and $\sum_{i=1}^{m} w_{\phi(i)} \geq 0.5$.

(a) (10 points) Give an algorithm that finds the weighted median of an array in $O(n)$ time. You may assume that you have an algorithm to find the median of an array (without weights) in

$O(n)$ time (e.g., the algorithm you learned in class). This algorithm can be used as a "black box" (like a function call) and need not be described in your solution. Remember that the array is unsorted and sorting the array at any point during the algorithm will take $\Omega(n \log n)$ time.

(b) (5 points) Alice Algorithmix lives on a long, narrow island somewhere off the coast of North Carolina where all the cities on the island are on a single straight line. The island government has decided to build an airport that will serve all the cities and has asked Alice to come up with a location that would minimize the average travel time to the airport for the residents of the island. (The travel time of a resident is proportional to her distance from the airport.) Can you help Alice solve the problem in $O(n)$ time if there are n cities? You can assume that Alice gives you the locations of the cities and their respective populations.

(c) (10 points) What if instead of Alice Algorithmix, you were asked this question by Bob Bitfiddler, who is a student at Duke and wants to find a location for an airport that will serve all residents of Durham county? You are given the coordinates of all neighborhoods in the county and their respective populations. As earlier, the goal is to minimize the average travel time of residents to the airport, where the travel time is now proportional to the "Manhattan distance" defined as $|x_1 - x_2| + |y_1 - y_2|$ for two points (x_1, y_1) and (x_2, y_2). Your algorithm should run in $O(n)$ time if there are n neighborhoods.

Hint: The last two parts are related to the first part. You can use an algorithm you designed for the first part as a black box in the last two parts.

Problem 4 (10 points)
Alarmed at the performance of students in the first midterm, Professor Dumbledore (remember him?) decided to allow limited consultation in the second midterm. He first arranged students in an arbitrary order and then stipulated that every student is only allowed to discuss answers with her two neighbors (one neighbor for the students at the two ends... tough luck). Not surprisingly, students who discussed among themselves ended up obtaining very similar scores: in fact, the scores of any pair of adjacent students turned out to differ by at most 1. Formally, if $X = x_1, x_2, \ldots, x_n$ is an array of students' scores in the order in which they were arranged, then $|x_i - x_{i+1}| \leq 1$ for $1 \leq i \leq n - 1$. Also, suppose $x_1 \leq x_n$. Given a particular number $x_1 \leq b \leq x_n$, can you help the professor find at least one student who got a score of b in $O(\log n)$ time? Note that there may be multiple students with a score of b but you only need to find one. You can assume that all the scores are integers and b is an integer as well.

Problem 5 (35 points)
Now that you have solved Professor Dumbledore's problem, your old friend Alice Algorithmix (remember the girl who lives on the long, narrow island?) asks for your help once again.

(a) (10 points) This time, she has been contracted by the department of geological survey who want to find the highest altitude in the island. Since this is essentially a volcanic island and a

narrow one at that, the altitude data that Alice has is just an array of n numbers that increase till some index and decrease thereafter. Formally, if $x_1, x_2, \ldots, x_n$ are the altitudes, there exists some index i such that $x_j > x_{j-1}$ for all $j \leq i$ and $x_j < x_{j-1}$ for all $j > i$. Clearly, x_i is the altitude that Alice needs to find. Can you help her find this index i in $O(\log n)$ time?

(b) (25 points) If Alice comes, can Ben be far behind? Now, our beloved Duke student Bob Bitfiddler has a $n \times n$ 2-D array on his hands which gives the altitudes of all points in Durham county (let's assume Durham is square). But, Durham is not quite a volcano! So, the altitudes form an arbitrary matrix of numbers. Bob would have ideally liked to find the highest point in Durham, but since he does not have the nice property that Alice had, he settles for an easier target: find a *locally* highest point, i.e., an entry in the $n \times n$ 2-D array that is as large as both its horizontal neighbors *and* both its vertical neighbors. (Bob hopes that this will at least be a hillock if not a mountain!) Formally, M is an $n \times n$ array, then $M[i, j]$ is a locally highest point if $M[i, j] \geq \max(M[i-1, j], M[i+1, j], M[i, j-1], M[i, j+1])$. Note that an entry on the boarder of the grid (which will only have 2 or 3 valid vertical/horizontal neighbors) just needs to be as large as its valid neighbors in order to be considered a locally highest point. Can you help Bob find such an entry in $O(n)$ time? (An $O(n \log n)$ solution will get partial credit.)

Problem 6 (15 points)

Your friend Bob Bitddler has now come to you with a different problem. He is looking to open convenience stores in the small town of Linesville, NC. Linesville has just a single east-west street called Straight Street, along which there are n homes in total. Having surveyed the residents of Linesville, Bob has come to know that they are willing to walk at most one mile to go to a convenience store. Can you help Bob set up a minimum number of stores such that every resident has one within walking distance? The locations of the houses are given to you from east to west, and your algorithm should run in $O(n)$ time.