Quiz 1

- Do not open this quiz booklet until directed to do so. Read all the instructions on this page.
- When the quiz begins, write your name on the top of every page of this quiz booklet.
- You have 120 minutes to earn a maximum of 120 points. Do not spend too much time on any one problem. Skim them all first, and attack them in the order that allows you to make the most progress.
- **You are allowed one double-sided letter-sized sheet with your own notes**. No calculators, cell phones, or other programmable or communication devices are permitted.
- Write your solutions in the space provided. Pages will be scanned and separated for grading. If you need more space, write "Continued on S1" (or S2, S3, S4) and continue your solution on the referenced scratch page at the end of the exam.
- Do not waste time and paper rederiving facts that we have studied in lecture, recitation, or problem sets. Simply cite them.
- When writing an algorithm, a **clear** description in English will suffice. Pseudo-code is not required. Be sure to argue that your **algorithm is correct**, and analyze the **asymptotic running time of your algorithm**. Even if your algorithm does not meet a requested bound, you **may** receive partial credit for inefficient solutions that are correct.
- **Pay close attention to the instructions for each problem**. Depending on the problem, partial credit may be awarded for incomplete answers.

Problem	Parts	Points
0: Information	2	2
1: Tree Exploration	4	25
2: Sum 99	1	20
4: 2D Binary Search	2	20
3: Kerberos Confusion	1	20
5: Winter Commute	3	33
Total		120

Name: ______________________________

School Email: ______________________________

Problem 0. [2 points] **Information** (2 parts)

(a) [1 point] Write your name and email address on the cover page.

(b) [1 point] Write your name at the top of each page.

Problem 1. [25 points] **Tree Exploration** (4 parts)

(a) [5 points] The following integer array would comprise a binary max-heap, except that one integer key **does not satisfy** the max-heap property. Draw the binary tree associated with the array and circle the integer that violates the max-heap property.

```
[9,8,5,7,2,6,4,3,1]
```

(b) [5 points] Given a binary search tree containing duplicate keys, where two or more nodes contain the same key v, will **every** node containing key v be the child or parent of another node containing v? If yes, prove the claim. If no, provide a counter example, i.e., draw a binary search tree for which the claim does not hold.

(c) [5 points] Consider the set T of all binary trees containing exactly 7 nodes which also satisfy the **AVL Property**. What is the maximum height of any tree in T? Draw a tree from T that achieves maximum height. (Recall that the **height** of a tree is the number of **edges** contained in a longest root-to-leaf path.)

(d) [10 points] Given two nodes from a **complete** binary tree on n nodes (i.e. full except for the lowest level), describe an $O(\log n)$ time algorithm to return the node farthest from the root whose subtree contains both nodes. State whether your algorithm achieves a **worst-case**, **expected**, and/or **amortized** bound.

Problem 2. [20 points] **Sum 99** (1 part)

Given an array of n **distinct** integers, we would like to identify three integers from the array that sum to 99, if such a triple exists. If the array contains multiple such triples, a correct algorithm may return any such triple. For example, for array `[11, 6, 45, 93, -5, 50, 43]`, a correct algorithm will return either `[11, 45, 43]`, `[11, 93, -5]`, or `[6, 50, 43]` (triples may be returned in arbitrary order). Describe an $O(n^2)$ time algorithm to identify a triple of elements that sum to 99, if one exists. State whether your algorithm achieves a **worst-case**, **expected**, and/or **amortized** bound.

Problem 3. [20 points] **2D Binary Search** (2 parts)

A two-dimensional $n \times n$ matrix A is **two-dimensionally sorted** if every row and column is sorted, i.e., each element $A[i][j]$ is at least as large as any item in the same row to the left, or in the same column above it in the matrix. For example, the matrix below is two-dimensionally sorted. In this problem, **you may assume that all elements in the matrix are unique**.

```
A = [[ 1,  4,  7, 11, 15],
     [ 2,  5,  8, 12, 19],
     [ 3,  6,  9, 16, 22],
     [10, 13, 14, 17, 24],
     [18, 21, 23, 26, 30]]
```

(a) [10 points] Given an $n \times n$ square matrix that is two-dimensionally sorted, a brute force algorithm can search for an element in the matrix in $O(n^2)$ time. Describe an algorithm to solve two-dimensional search, asymptotically faster than $O(n^2)$, but asymptotically slower than $O(n)$. State whether your algorithm achieves a **worst-case**, **expected**, and/or **amortized** bound. (You will be asked for an $O(n)$ algorithm in the next part. We would like a **different** algorithm that improves on brute force.)

(b) [10 points] One can perform two-dimensional search in linear time by starting a search from a corner. Give an $O(n)$ time algorithm for two-dimensional search. State whether your algorithm achieves a **worst-case**, **expected**, and/or **amortized** bound.

Problem 4. [20 points] **Kerberos Confusion** (1 part)

MIT administrators have trouble distinguishing between similar-looking Kerberos usernames. A Kerberos username is a sequence of at most eight alpha-numeric characters. The administrators would like to determine how many Kerberos usernames are **similar** to another: two Kerberos usernames are similar if one can be transformed to the other by replacing a single character. For example, username `fun6006` is similar to usernames `fun6046` and `fuu6006`, but not username `6006fun`. Given a list of n unique Kerberos usernames, design a **worst-case** $O(n)$ algorithm to find every username from the list that is similar to another username from the list. You may receive significant partial credit for an **expected** $O(n)$ time algorithm.

Problem 5. [33 points] **Winter Commute** (3 parts)

The recent snowy weather in Cambridge is causing trouble for students living on Massachusetts Avenue who bike to MIT. Biking in a bike lane containing too much snow can be dangerous: a student could be injured, or even worse, they might miss class!

A group of enterprising students embed sensors at one foot intervals along Mass. Ave. to measure the height of snow at their location along the bike path. The collected data is upload to an online database every few hours. If the height of snow in the Mass. Ave. bike lane is too large, as measured by any sensor between a biker's house and MIT, then the biker will not be able to go to school. Each student living along Mass. Ave. in Cambridge (only north of MIT) wants to be able to query the current maximum height of snow in the bike path between MIT and their home, located a known number of feet from MIT along Mass. Ave.

(a) [5 points] Design a data structure to process and store new data from the n sensors into the online database in worst-case $O(n)$ time, that supports student queries in worst-case $O(1)$ time.

(b) [5 points] It turns out that the database does not update frequently enough for many students, as snow plows periodically remove (and sometimes add!) snow to the bike lane. The enterprising students outfit the snow plows with trackers that can send to the database, updates of the following form: h inches of snow have been uniformly added or removed from the bike lane at and between distances a and b, measured in feet from MIT along Mass. Ave. Adapt your data structure from part (a) to update snow estimates based on a snow plow update in worst-case $O(n)$ time.

Unfortunately, there are so many sensors in the network that performing snow plow updates in linear time is much too slow. You want to redesign your data structure to:

1. process and store new sensor data in $O(n)$ time,
2. respond to a student query in $O(\log n)$ time, and
3. process a snow plow update in $O(\log n)$ time.

(c) [10 points] Describe how to augment a binary search tree to process new sensor data in worst-case $O(n)$ time, and respond to student queries in worst-case $O(\log n)$ time. Hint: use an augmentation that will make supporting updates in the next part easier. For this part, a rigorous proof of correctness is not required.

(d) [10 points] Describe how to augment your data structure from part (c) to support snow-plow updates in worst-case $O(\log n)$ time, in addition to the other operations. For this part, a rigorous proof of correctness is not required.

SCRATCH PAPER 1. DO NOT REMOVE FROM THE EXAM.

You can use this paper to write a longer solution if you run out of space, but be sure to refer to the scratch paper next to the question itself.

SCRATCH PAPER 2. DO NOT REMOVE FROM THE EXAM.

You can use this paper to write a longer solution if you run out of space, but be sure to refer to the scratch paper next to the question itself.

SCRATCH PAPER 3. DO NOT REMOVE FROM THE EXAM.

You can use this paper to write a longer solution if you run out of space, but be sure to refer to the scratch paper next to the question itself.

SCRATCH PAPER 4. DO NOT REMOVE FROM THE EXAM.

You can use this paper to write a longer solution if you run out of space, but be sure to refer to the scratch paper next to the question itself.