Practice Quiz 1

- Do not open this quiz booklet until you are directed to do so. Read all the instructions first.
- When the quiz begins, write your name on every page of this quiz booklet.
- The quiz contains five multi-part problems. You have 80 minutes to earn 80 points.
- This quiz booklet contains 11 pages, including this one and an extra sheet of scratch paper, which is included for your convenience.
- This quiz is closed book. You may use one handwritten Letter ($8\frac{1}{2}$'' × 11'') or A4 crib sheet. No calculators or programmable devices are permitted.
- Write your solutions in the space provided. Extra scratch paper may be provided if you need more room, although your answer should fit in the given space.
- Do not waste time and paper re-deriving facts that we have studied. It is sufficient to cite known results.
- Do not spend too much time on any one problem. Read them all through first, and attack them in the order that allows you to make the most progress. Generally, a problem’s point value is an indication of how much time to spend on it.
- Show your work, as partial credit will be given. You will be graded not only on the correctness of your answer, but also on the clarity with which you express it. Be neat.
- Good luck!

<table>
<thead>
<tr>
<th>Problem</th>
<th>Points</th>
<th>Grade</th>
<th>Initials</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>??</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>??</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>??</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>??</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>??</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>80</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Problem 1. Recurrences [?? points] (3 parts)**

For each of the following recurrences, give an asymptotically tight ($\Theta(\cdot)$) bound. Justify your answer by naming the particular case of the Master’s Method, by iterating the recurrences, or by using the substitution method. As usual, assume that for $n \leq 10$, $T(n) = O(1)$.

**Example:** [0 points] **Binary Search**

Recurrence: $T(n) = T(n/2) + c$

Solution by iteration:

\[
T(n) = T(n/4) + c + \sum_{i=0}^{\log n} c = c \log n = \Theta(\log n)
\]

(a) [?? points] $T(n) = 8T(n/2) + \Theta(n)$.

(b) [?? points] $T(n) = 9T(n/9) + \Theta(n\sqrt{n})$. 
(e) [?? points] \[ T(n) = T(\sqrt{n}) + \log n. \] (It is fine to assume that \( n \) is of the form \( 2^{2^k} \) in order to avoid floor and ceiling notation.)
Problem 2. True or False, and Justify [?? points] (4 parts)

Circle T or F for each of the following statements, and briefly explain why. The better your argument, the higher your grade, but be brief. Your justification is worth more points than your true-or-false designation.

(a) T F [?? points] \( f(n) = \Theta(g(n)) \) is equivalent to \( g(n) = \Theta(f(n)) \).

(b) T F [?? points] There is a deterministic algorithm to sort \( n \) numbers in the comparison model running in time \( \log F_n \), where \( F_n \) is the \( n \)th Fibonacci number. (Recall that the Fibonacci numbers are defined as follows: \( F_0 = 0, F_1 = 1 \) and for \( i > 1 \), \( F_i = F_{i-1} + F_{i-2} \).)
Problem 3. Hashing

Design a data structure called DISTINCT which maintains a multi-set of integers in the range \( \{1, 2, \ldots, n^3\} \) under insertions and deletions. A multi-set is a set of elements where duplicate keys are allowed (e.g., \( \{1, 2, 1, 3\} \) is a multi-set). At any time, the data structure should provide the current number of distinct elements in the multi-set. For example, the multi-set \( \{1, 2, 1, 3\} \) contains 3 distinct elements.

Design a randomized data structure that supports the above operations in \( O(1) \) expected time. You can assume that at any time the total number of distinct elements in the set is at most \( n \). Moreover, you can assume that you are given an \( O(n) \)-size empty block of memory at the beginning.
Problem 4. Plural Elements [?? points] (2 parts)
You are given an array of $n$ integers. For each of the questions below, remember to give a brief, clear explanation of why the algorithm works.

(a) [?? points] Suppose there exists an integer that appears more than $n/2$ times in the array. Give a linear time deterministic algorithm to find such an integer.

(b) [?? points] Now suppose there exists an integer that appears more than $n/k$ times, where $k > 2$ is a constant (suppose $n$ is divisible by $k$). Give a linear time deterministic algorithm to find all such integers.
Problem 5. Min and Max Revisited [?? points] (3 parts) In this problem we will investigate a divide and conquer approach for simultaneously finding the minimum and maximum of an array of $n$ integers. In the comparison model (i.e., your algorithm is only allowed to compare elements, but not add/subtract/index with them).

Assume for simplicity that $n$ is a power of 2.

For each of the questions below, remember to give a brief and clear justification.

(a) [?? points] Suppose you compute the maximum separately and the minimum separately, and output both. How many comparisons does this require?
(b) [?? points] For the rest of this problem, we would like to reduce the leading constant factor in the number of comparisons. We want a result that is not necessarily better asymptotically, but in terms of the exact number of comparisons.

Let's consider an idea for designing a divide and conquer solution for simultaneously computing the minimum and maximum element. The plan is to break the problem into two sublists, compute the min and max of each sublist and then combine the results. An outline of such an algorithm is given below. In the following page, you will be given the opportunity to fill in some details:

\[
\text{MINMAX}(A, n);
\]

If \( n = 2 \) then \( . . . \) fill in base case
else Let \( A_1 = \) left half and \( A_2 = \) right half
\[
(a, b) = \text{MINMAX}(A_1, n/2)
\]
\[
(c, d) = \text{MINMAX}(A_2, n/2)
\]
\( . . . \) fill in combine stage

(i) Fill in the details for the case \( n = 2 \).

(ii) Fill in the details for the combine step here.
(c) [?? points]
Let $T(n) = c_1 n - c_2$ be the number of comparisons used by the algorithm. Find an exact expression for $T(n)$ (i.e., the best values of the constants $c_1$ and $c_2$). (You may do this by writing out and solving the recurrence for $T(n)$ using the substitution method.) Is this really better than the naive algorithm?
Problem 6. Matrix Multiplication [?? points] (2 parts)

In this problem we are given three matrices and the question is to compute their product. For each of the two cases below, report the fastest algorithm you know to compute the product of the given three matrices. You need to prove correctness of the algorithm and argue its running time.

(a) [?? points] Given matrices $A, B, C$, each of dimension $n \times n$, give a fast algorithm to compute $A \cdot B \cdot C$? What is its asymptotic running time? (You may use algorithms mentioned in lectures without describing them.)

(b) [5 points] Now, suppose $C$ is of dimension $n \times 1$ (i.e., $C$ is a vertical vector), and $A$ and $B$ are still of dimension $n \times n$. Give a fast algorithm for computing $A \cdot B \cdot C$? What is its asymptotic running time?
Problem 7. Joining and Splitting 2-3-4 Trees

The JOIN operator takes as input two 2-3-4 trees, $T_1$ and $T_2$, and an element $x$ such that for any $y_1 \in T_1$ and $y_2 \in T_2$, we have $\text{key}[y_1] < \text{key}[x] < \text{key}[y_2]$. As output JOIN returns a 2-3-4 tree $T$ containing the node $x$ and all the elements of $T_1$ and $T_2$.

The SPLIT operator is like an “inverse” JOIN: given a 2-3-4 tree $T$ and an element $x \in T$, SPLIT creates a tree $T_1$ consisting of all elements in $T - \{x\}$ whose keys are less than $\text{key}[x]$, and a tree $T_2$ consisting of all elements in $T - \{x\}$ whose keys are greater than $\text{key}[x]$.

In this problem, we will efficiently implement JOIN and SPLIT. For convenience, you may assume that all elements have unique keys.

(a) Suppose that in every node $x$ of the 2-3-4 tree there is a new field $\text{height}[x]$ that stores the height of the subtree rooted at $x$. Show how to modify INSERT and DELETE to maintain the $\text{height}$ of each node while still running in $O(\log n)$ time. Remember that all leaves in a 2-3-4 tree have the same depth.

(b) Using part (a), give an $O(1 + |h_1 - h_2|)$-time JOIN algorithm, where $h_1$ and $h_2$ are the heights of the two input 2-3-4 trees.
(c) Give an $O(\log n)$-time SPLIT algorithm. Your algorithm will take a 2-3-4 tree $T$ and key $k$ as input. To write your SPLIT algorithm, you should take advantage of the search path from $T$’s root to the node that would contain $k$. This path will consist of a set of keys $\{k_1, \ldots, k_m\}$. Consider the left and right subtrees of each key $k_i$ and their relationship to $k$. You may use your JOIN procedure from part (b) in your solution.