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 every page of this quiz booklet.
- You have 120 minutes to earn 120 points. Do not spend too much time on any one problem. Read them all first, and attack them in the order that allows you to make the most progress.
- **You are allowed a 1-page cheat sheet.** No calculators or programmable devices are permitted. No cell phones or other communications devices are permitted.
- Write your solutions in the space provided. If you need more space, write on the back of the sheet containing the problem. Pages may be separated for grading.
- Do not waste time and paper rederiving facts that we have studied. Simply cite them.
- When writing an algorithm, a **clear** description in English will suffice. Pseudo-code is not required unless asked for.
- **Pay close attention to the instructions for each problem.** Depending on the problem, partial credit may be awarded for incomplete answers.
- **Blank answers will receive approximately 25% of the credit for the problem.** If your answer shows no understanding of the problem, you will receive less than 25%. So, if you have no idea how to attack a problem, you should leave it blank.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Parts</th>
<th>Points</th>
<th>Grade</th>
<th>Grader</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>25</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>120</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Name: ____________________________

Circle your recitation:

- R01 Gurtej Ashwin 10AM
- R02 Gurtej Ashwin 11AM
- R03 Jennifer Ashwin 12PM
- R04 Jennifer Quanquan 1PM
- R05 Arvind Quanquan 2PM
- R06 Arvind Vlad 3PM
- R07 Justin Vlad 11AM
- R08 Justin Sherwin 12PM
Problem 1. [20 points] Fun with sorting

(a) [15 points]
Consider the following sorting algorithms on $n$ integers ranging from 1 to $n^2$:
1. Insertion sort
2. Merge sort
3. Heap sort
4. Counting sort
5. Radix sort
6. AVL sort (insert all items into an AVL tree, then read them in order by performing an in-order traversal in $\Theta(n)$ time)

For each of the following properties, determine which algorithms in the above list have that property. You may refer to them by index. You do not need to justify your answer.

Example Question: This algorithm correctly sorts a list of elements.
Example Solution: 1,2,3,4,5,6

i. This algorithm runs in time $O(n \log n)$.

ii. This algorithm is stable.

iii. This algorithm is in-place.

iv. For all values of $n$, there is an array of $n$ distinct elements which this algorithm sorts in $O(n)$ time.

v. This algorithm works in the comparison model (i.e., the only manipulation of the elements, other than data movement, is by pairwise comparisons: $<$, $>$, $\leq$, $\geq$, $\neq$).
(b) [5 points] For each of the following pairs of elements \((x, y)\), state whether \(x \geq y\), \(x \leq y\), or \(x\) and \(y\) are incomparable, i.e., not determinable from the information given.

You do not need to justify your answer.

**Example Question:** \(x\) is the first element and \(y\) is the last element of a sorted list.

**Example Solution:** \(x \leq y\)

   i. \(x\) is the root and \(y\) is a leaf of a max-heap.

   ii. \(x\) is the left child and \(y\) is the right child of a node \(z\) in a max-heap.

   iii. \(x\) is the root and \(y\) is a leaf of an AVL tree.

   iv. \(x\) is the left child and \(y\) is the right child of a node \(z\) in an AVL tree.

   v. Partway through merge-sorting an array \(A\), we have recursively sorted the two halves \(A_1\) and \(A_2\) of \(A\), but not yet merged them. Then \(x\) is the first element of \(A_1\) and \(y\) is the first element of \(A_2\).
Problem 2. [25 points] Asymptotics and Recurrences

(a) [5 points] Is it always true that \( f(n) + g(n) = \Theta(\min(f(n), g(n))) \)? If so, prove it. If not, find a counterexample and show that this statement is false.

(b) [5 points] Is it always true that a real-valued function \( f \) satisfies \( f(n) = O((f(n))^2) \)? If so, prove it. If not, find a counterexample and show that this statement is false.

(c) [5 points] Suppose we have a hash table, with collisions resolved by chaining, and we implement table doubling to resize the table but do not change the hash function. Why is this a bad idea?
(d) [5 points] Write and solve the recurrence relation for a \textit{three-way mergesort}, which splits the array into three equal-size pieces, recursively sorts each piece, and then merges them into one sorted array. (You do not need to give the algorithmic details, but your analysis must be correct.) Show your work. You may assume that $n$ is a power of 3.

(e) [5 points] What is the asymptotic runtime of an algorithm with the following recurrence relation? Show your work.

\[ T(n) = T(\sqrt{n}) + \Theta(1) \]

\textit{Hint:} What happens when you substitute $n = 2^m$?
Problem 3. [20 points] Dream Team

Story (feel free to skip this paragraph): Ben Bitdiddle has just been appointed as the captain of his school’s BitBall team, an intense game with \( k \) players on a team. He wants to create the ultimate Dream Team and needs your help. Because the game is so focused on teamwork, Ben realizes he should only put players on the team who are friends with another player already on the team (initially Ben is the only player on the team). Unfortunately, potential players are not allowed to talk to Ben until he has already decided whether they are on the team (to prevent bribery and blackmail—getting on the team is a big deal), so he won’t be able to deduce who is friends with whom beforehand. As a result, Ben decides on the following strategy:

To recruit \( k \) players out of \( N \) total players, initially Ben considers himself the only eligible player. At each step, Ben picks the highest-rated eligible player to put on the team, then asks that player for a list of his friends, and considers all of those friends eligible. Note that a player can be on the team only once, so after they are recruited, they cannot become eligible again.

You may assume that Ben has access to a function \( \text{RECRUITPLAYER}(P) \), which takes the name \( P \) of a player to recruit, and returns a list of the names of that player’s friends. This list is guaranteed to have length at most a known parameter \( m \), and \( \text{RECRUITPLAYER} \) is guaranteed to run in \( O(m) \) time. Ben also has a function \( \text{SKILL}(P) \) that returns the numeric skill rating of the player named \( P \) in constant time.

Using data structures covered in 6.006, describe an algorithm to draft Ben’s Dream Team (or report failure) and briefly analyze its runtime. Your algorithm should start by calling \( \text{RECRUITPLAYER}(“Ben”) \) to add Ben to the team and get a list of his friends. You do not need to reimplement operations described in class; you may use them as a black box. For full credit, your algorithm should run in time \( O(mk \log (mk)) \).
Problem 4. [20 points] Perfect Pairs

Alyssa P. Hacker has a large unsorted list \( A \) of \( n \) numbers, and a target number \( k \). In \( O(n) \) expected time, she wants to compute the number of pairs of numbers in \( A \) that sum to \( k \). Assume for this problem that all arithmetical operations between two numbers take \( O(1) \) time. Also, note that \( A \) may contain repeated numbers.

For example, if \( A = [-1, -1, 3.5, 4.5, 9, 4, 4, 5, -2] \) and \( k = 8 \), then the answer would be 4 because \((-1, 9)\) can be made in two ways, \((4, 4)\) and \((3.5, 4.5)\) can be made in one way, and these are the only pairs that sum to 8.

(a) [5 points] What data structure will best help Alyssa with this task?

(b) [20 points] Describe an \( O(n) \) expected time algorithm to solve this problem using the data structure you specified in (a). Partial credit will be given to algorithms that work properly under the assumption that \( A \) contains no repeats and/or that \( A \) does not contain \( k/2 \).
Problem 5. [20 points] **Data Structure Mashup**

Recall the following operations on a data structure storing a dynamic set $S$ of $n$ distinct numbers:

- $\text{INSERT}(k)$: Add $k$ to $S$.
- $\text{SEARCH}(k)$: Return $\text{TRUE}$ if $k \in S$; otherwise return $\text{FALSE}$.
- $\text{SUCCESSOR}(k)$: Return the smallest value $x \in S$ with $x > k$, or $\infty$ if none exists.

(a) [10 points] Fill in the running times of the operations above for each of the data structures below. For hashing, assume a table size of $m = \Theta(n)$, assume hashing with chaining, and state expected time bounds.

<table>
<thead>
<tr>
<th>Data structure</th>
<th>INSERT</th>
<th>SEARCH</th>
<th>SUCCESSOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unsorted array</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sorted array</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AVL tree</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hash table</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Heap</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(b) [10 points] Describe a data structure achieving the following running times. You should give explicit algorithms for each of the three operations. You may use any algorithms covered in class as a black box. *Hint:* Combine some data structures from the list above.

<table>
<thead>
<tr>
<th>Data structure</th>
<th>INSERT</th>
<th>SEARCH</th>
<th>SUCCESSOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ultra</td>
<td>$O(\log n)$</td>
<td>$O(1)$</td>
<td>$O(\log n)$</td>
</tr>
</tbody>
</table>
Problem 6. [15 points] Are We Related?

Suppose that you have an AVL tree $T$ with $n$ elements already inserted into it. Describe an efficient algorithm that, given pointers to two nodes $x$ and $y$ in $T$, finds the lowest common ancestor of both $x$ and $y$. For partial credit, achieve a running time of $O(\log n)$. For full credit, achieve a running time of $O(d)$ where $d$ is the distance between nodes $x$ and $y$ in the tree.

For example, if $T$ were the following tree, and $x$ pointed to the node with value 2 and $y$ pointed to the node with value 7, then your algorithm should return the node with value 4.