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.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Parts</th>
<th>Points</th>
<th>Grade</th>
<th>Grader</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>7</td>
<td>35</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>24</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>36</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bonus</td>
<td>1</td>
<td>5</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:  
R01,2 Alin Tomescu  
R03 Deepak Narayanan  
R04 Joseph Henke  
R05 Casey O’Brien  
R06,7 Ilia Lebedev  
R08 Andreea Bodnari  
R09 Kevin Zatloukal  
R10 Deniz Oktay  
10,11AM  
12PM  
12PM  
1PM  
1,2PM  
2PM  
3PM  
4PM
Problem 1. Short Answer Questions [35 points] (7 parts)

(a) [5 points] If \( f(n) = O(g(n)) \) and \( g(n) = O(h(n)) \), then \( h(n) = \Omega(f(n)) \). True or False? Justify your answer.

(b) [5 points] \( f(n) \) is defined to be the running time of the program \( A(n) \):

```python
def A(n):
    # a tuple is an immutable version of a
    # list, so we can hash it
    atuple = tuple(range(0, n))

    # A set is a dictionary without values
    S = set()
    for i in range(0, n):
        for j in range(i+1, n):
            # add tuple (i,...,j-1) to set S
            S.add(atuple[i:j])
```

Give a \( \Theta \) bound for \( f(n) \).
(c) [5 points] In an effort to make MERGE-SORT faster, you decide to divide the array into $k$ equal sized, disjoint subarrays, where $k > 2$. This means that you have to merge $k$ lists. Write the recurrence for this algorithm assuming merge can be accomplished in $O(n \log(k))$ time and solve it, i.e., give a $\Theta$ bound.

(d) [5 points] A min-heap is an implementation of the Priority Queue abstract data type. Which of the following operations are supported **efficiently** by min-heaps: $\text{FIND}(value)$, $\text{FIND-MIN}()$, $\text{INSERT}(value)$, $\text{DELETE}(value)$, $\text{EXTRACT-MIN}()$?

(e) [5 points] Suppose that you have inserted 15 distinct items into a min-heap. The minimum element will be in the first position of the array. How many positions in the underlying array might contain the maximum element? *Hint: Draw a picture of the tree.*
(f) [5 points] Recall our representation of a heap as a binary tree embedded into an array. Suppose the same representation is used to store a binary search tree. How big of an array would be needed to store a best-case BST of \( n \) elements? How big of an array would be needed to store a worst-case BST of \( n \) elements? Assume \( n \) is \( 2^k - 1 \) for some \( k \).

(g) [5 points] Ben Bitdiddle modifies RADIX-SORT to use INSERTION-SORT to sort by digits, instead of COUNTING-SORT. Would the resulting algorithm still work correctly? What is the complexity of this new algorithm? The complexity of conventional RADIX-SORT on \( n \) numbers in base \( b \) with at most \( d \) digits is \( O((n + b) \cdot d) \).
Problem 2. Binary Search Trees [24 points] (3 parts)

This problem explores several aspects of binary search trees. In all the search trees we consider here, you may assume that the keys are taken from a totally ordered key space $K$, and all the keys stored in the tree are distinct. Totally ordered means that any pair of keys can be compared, and if they are different, one is bigger than the other.

(a) [8 points] Describe (in clear English or pseudocode) an algorithm that does the following: It takes as input an arbitrary (not necessarily balanced) binary tree with unique keys from set $K$ at the nodes, and returns a correct answer saying whether or not the tree is an actual binary search tree. Your algorithm should take time no worse than $O(n)$, where $n$ is the current number of elements in the tree. Include an analysis of the time complexity of your algorithm.
(b) [8 points] Describe an algorithm that begins with an arbitrary binary tree and produces a count of “how far” the tree is from a real binary search tree. Our measure of how far the tree is from being a BST is the number of distinct pairs of items in the tree that represent violations of the BST property transitively applied. For example in the tree below, (6, 3) is a violation because 6 is to the left of 3. The total number of violations is six: the pairs (6, 3), (6, 2), (6, 4), (3, 2), (6, 5), and (8, 7).

Your algorithm should run in time $O(n + k)$, where $k$ is the number of violations. Include a time complexity analysis.
(c) [8 points] Describe (in clear English or pseudocode) an $O(n)$ algorithm for balancing an arbitrary binary search tree, that is, for producing a new binary search tree with the same elements as the original one but with height $O(\log(n))$. Include a time complexity analysis.
Problem 3. Treehash [12 points] (4 parts)

Suppose we store \( n \) elements in an \( m \)-slot hash table using chaining, but we store each chain (set of elements hashing to the same slot) using a binary search tree (BST) instead of a linked list. Each element corresponds to a key/value pair and there are no duplicates. Also suppose that \( m = n \), so the load factor \( \alpha = n/m = 1 \). Assume that there will be no resizing of the table.

(a) [3 points] What is the expected running time of insert in this hash table? Why? Assume simple uniform hashing.

(b) [3 points] What is the worst-case running time of insert in this hash table? Why? (Do not assume simple uniform hashing.)

(c) [3 points] Suppose that the BST is replaced with an AVL tree. What is the worst-case running time of insert in this new hash table? Why? (Do not assume simple uniform hashing.)

(d) [3 points] Suppose that the AVL tree is replaced with a heap. What is the worst-case running time of insert in this new hash table? Why? (Do not assume simple uniform hashing.)
Problem 4. Hash Table Analysis [12 points] (2 parts)
You are given a hash table with $n$ keys and $m$ slots, with the simple uniform hashing assumption (each key is equally likely to be hashed into each slot). Collisions are resolved by chaining.

(a) [4 points] What is the probability that the first slot ends up empty?

(b) [8 points] What is the expected number of slots that end up not being empty?
Problem 5. Snootheby’s [36 points] (6 parts)

Snootheby’s is an auction house that sells expensive art to very rich people. The prices at which they sell their art vary wildly, depending on the current economic situation and the whims of their customers. Consequently, Snootheby’s finds it important to keep careful track of the typical prices at which they have recently sold their art.

Snootheby’s has hired you to design a system (a data structure and algorithms) to keep track of their typical sales prices, specifically, of the $k$ middle sales prices for the current year so far, for some fairly small value $k$ that they will provide as a parameter. For example, if the number of sales and sales prices so far is $n$ and $n - k$ is an odd number, then the number of prices higher than the “middle” ones should be $\lfloor \frac{n-k}{2} \rfloor$ or $\lceil \frac{n-k}{2} \rceil$—we don’t care which.

Their accounting starts anew each year, on January 1. Sales data gets recorded in a buffer, from which we will process the information, one sale at a time.

(a) [12 points] Design a data structure to keep track of the $k$ middle sales prices for the current year so far. Your data structure must support two operations:

- **QUERY():** Return the middle $k$ sales prices for the year so far. (If there are fewer than $k$ sales so far then return all of their prices.) This should run in time no worse than $O(k)$.

- **ADD($x$):** Add information about the next sale, with sales price $x$, into the data structure. This should run in time no worse than $O(\log(n) + k)$, where $n$ is the current number of recorded sales.

Describe clearly the data you would maintain and how it would be organized (e.g., “Store a list of such and such sorted in such order”).
(b) [8 points] State (but do not prove) two invariants that are always preserved by your algorithm, that is, a collection of properties that are always true after any number of ADD operations have been performed, and that imply that QUERY operations always return correct answers. The two invariants should correspond to an invariant about the prices (keys in your data structure) and an invariant about the sizes of your data structures.
(c) [3 points] Describe (in clear English or pseudocode) how your QUERY operation works. Recall that your goal is an \(O(k)\) operation.

(d) [3 points] Analyze the time complexity of your QUERY operation. The goal is an accurate analysis of the complexity of your operation and you will receive full credit for a correct analysis regardless of time complexity.
(e) [6 points] Describe (in clear English or pseudocode) how your ADD($x$) operation works. Recall that your goal is an $O(\log(n) + k)$ operation.

(f) [4 points] Analyze the time complexity of your ADD($x$) operation. The goal is an accurate analysis of the complexity of your operation and you will receive full credit for a correct analysis regardless of time complexity.
Problem 6. [5 points] (1 part)

**BONUS QUESTION: ONLY ATTEMPT AFTER YOU HAVE COMPLETED THE OTHER QUESTIONS**

Give an English word included in the Merriam Webster online dictionary that has 6 or more of the same letter or *argue* that such a word does not exist. Points will be given for creative and humorous arguments. (Stresslessness is not an included word!)