Quiz 1

- Do not open this quiz booklet until you are 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.
- This quiz contains 7 problems, some with multiple parts. You have 120 minutes to earn 120 points.
- This quiz booklet contains 10 pages, including this one. Two extra sheets of scratch paper are attached. Please detach them before turning in your quiz at the end of the exam period.
- This quiz is closed book. You may use one $8\frac{1}{2}'' \times 11''$ or A4 crib sheet (both sides). No calculators or programmable 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. Do not put part of the answer to one problem on the back of the sheet for another problem, since the pages may be separated for grading.
- Do not waste time and paper rederiving 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.
- 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!

Problem	Parts	Points	Grade	Grader
1	1	15		
2	1	20		
3	2	15		
4	2	30		
5	1	10		
6	1	15		
7	2	15		
Total		120		

Name: ______________________________

Circle your recitation time:
Hueihan Jhuang: **(10AM) (11AM)** Victor Costan **(2PM) (3PM)**

Problem 1. Asymptotic workout [15 points]

For each function $f(n)$ along the left side of the table, and for each function $g(n)$ across the top, write O, Ω, or Θ in the appropriate space, depending on whether $f(n) = O(g(n))$, $f(n) = \Omega(g(n))$, or $f(n) = \Theta(g(n))$. If more than one such relation holds between $f(n)$ and $g(n)$, write only the strongest one. The first row is a demo solution for $f(n) = n^2$.

		$g(n)$		
		n	$n \lg n$	n^2
$f(n)$	n^2	Ω	Ω	Θ
	$n^{1.5}$			
	$\sqrt{2^n}$			
	$n\sqrt{\lg n}$			
	$n \log_{30} n$			
	n^3			

Problem 2. Table of Speed [20 points]

For each of the representations of a set of elements along the left side of the table, write down the asymptotic running time for each of the operations along the top. For hashing, give the expected running time assuming simple uniform hashing; for all other data structures, give the worst-case running time. Give tight asymptotic bounds using Θ notation. If we have not discussed how to perform a particular operation on a particular structure, answer for the most reasonable implementation you can imagine.

		Operation			
		Insert	Extract-min	Contains	Minimum
Data Structure	Unsorted linked list				
	Sorted linked list				
	Min heap				
	Max heap				
	Hashing with chaining and $\alpha = 1$				

Definitions of operations:

- Insert(S, x): add element x to the set S.
- Extract-min(S): remove the minimum element from the set S and return it.
- Contains(S, x): return whether element x is in the set S.
- Minimum(S): return the minimum element in set S (without extraction).

Problem 3. Indie heap operations [15 points] (2 parts)

(a) [10 points] Suppose you have a max-heap stored in an array $A[1..n]$, where $A[1]$ stores the maximum element. Give pseudocode for an efficient algorithm to implement the operation find-second-maximum(A), which finds the next-to-largest key stored in the max-heap A. Your algorithm should not modify the heap.

(b) [5 points] Suppose you have an array $A[1..n]$ of n elements in arbitrary order. Does the following alternate implementation of build-max-heap work? In other words, does it correctly build a max heap from the given elements $A[1..n]$? Why or why not?

```
build-max-heap($A$):
    for $i$ from 1 to $n/2$:
        max-heapify($A, i$)
```

This algorithm calls heapify starting at the root and working its way down the tree, instead of the other way around.

Problem 4. Madly merging many menus [30 points] (2 parts)

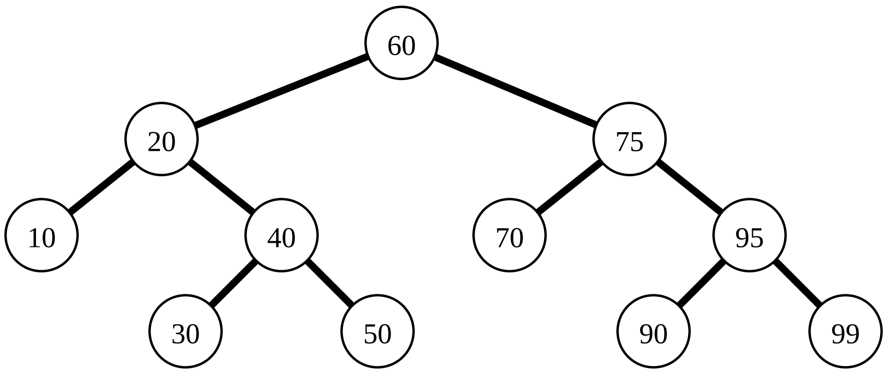
Professor Sortun uses the following algorithm for merging k sorted lists, each having n/k elements. She takes the first list and merges it with the second list using a linear-time algorithm for merging two sorted lists, such as the merging algorithm used in merge sort. Then, she merges the resulting list of $2n/k$ elements with the third list, merges the list of $3n/k$ elements that results with the fourth list, and so forth, until she ends up with a single sorted list of all n elements.

(a) [10 points] Analyze the worst-case running time of the professor's algorithm in terms of n and k.

(b) [20 points] Briefly describe an algorithm for merging k sorted lists, each of length n/k, whose worst-case running time is $O(n \lg k)$. Briefly justify the running time of your algorithm. (If you cannot achieve $O(n \lg k)$, do the best you can for partial credit.)

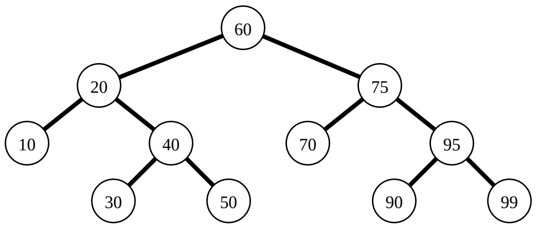
Problem 5. Being Adel'son-Vel'skiĭ & Landis [10 points]

Suppose that we insert 42 into the AVL tree below using the AVL insertion algorithm. On the next page, show the resulting tree after rebalancing. We also recommend showing intermediate steps for partial credit.



For reference, we include the Python code for AVL insertion, written in terms of rotations:

```
def insert(self, t):
    node = bst.BST.insert(self, t)
    while node is not None:
        if height(node.left) >= 2 + height(node.right):
            if height(node.left.left) >= height(node.left.right):
                self.right_rotate(node)
            else:
                self.left_rotate(node.left)
                self.right_rotate(node)
        elif height(node.right) >= 2 + height(node.left):
            if height(node.right.right) >= height(node.right.left):
                self.left_rotate(node)
            else:
                self.right_rotate(node.right)
                self.left_rotate(node)
        node = node.parent
```



Initial AVL tree

Problem 6. Honey, I shrunk the heap [15 points]

Suppose we have a heap containing $n = 2^k$ elements in an array of size n, and suppose that we repeatedly extract the minimum element, n times, never performing insertions. To make the heap space efficient, we move the heap over to an array of size 2^j whenever an extraction decreases the number of elements to 2^j for any integer j. Suppose that the cost of each such move is $\Theta(2^j)$. What is the total movement cost caused by n extract-mins starting from the heap of n elements? (Ignore the $\Theta(n \lg n)$ cost from the heapify operations themselves.)

Problem 7. Trash hees [15 points] (2 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 an AVL tree instead of a linked list. Also suppose that $m = n$, so the load factor $\alpha = n/m = 1$.

(a) [5 points] What is the expected running time of insert, delete, and search in this hash table? Why? Assume simple uniform hashing.

(b) [10 points] What is the worst-case running time of insert, delete, and search in this hash table? Why? (Do not assume simple uniform hashing.)

SCRATCH PAPER

SCRATCH PAPER