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 5 problems, some with multiple parts. You have 120 minutes to earn 100 points.
- This quiz booklet contains 7 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 examination period.
- This quiz is closed book. You may use one $8\frac{1}{2}'' \times 11''$ crib sheet. 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	5	25		
2	2	20		
3	1	15		
4	2	15		
5	1	25		
Total		100		

Name: ______________________________

Circle the name of your recitation instructor:

Yoyo Zhou (10AM) Michael Lieberman (3PM)

Problem 1. Asymptotic Notation [25 points] (5 parts)

State whether each statement below is **True** or **False**. You must briefly justify all your answers to receive full credit.

(a) If $f(n) = \Theta(g(n))$ and $g(n) = \Theta(h(n))$, then $h(n) = \Theta(f(n))$

(b) If $f(n) = O(g(n))$ and $g(n) = O(h(n))$, then $h(n) = \Omega(f(n))$

(c) If $f(n) = O(g(n))$ and $g(n) = O(f(n))$ then $f(n) = g(n)$

(d) $\frac{n}{100} = \Omega(n)$

(e) $f(n) = \Theta(n^2)$, where $f(n)$ is defined to be the running time of the program `A(n)`:

```
def A(n):
  atuple = tuple(range(0, n)) # a tuple is an immutable version of a
                              # list, so we can hash it
  S = set()
  for i in range(0, n):
    for j in range(i+1, n):
      S.add(atuple[i:j])   # add tuple (i,...,j-1) to set S
```

Problem 2. Weight-Balanced Binary Search Trees [20 points] (2 parts)

Recall from class our definition of a *weight-balanced binary search tree*: it maintains the weight balance of each node x, such that if the weight of x is $w(x)$, the weight of each child is at least $\alpha \cdot w(x)$, where $\alpha = 0.29$. (The *weight* of x is one more than the number of nodes in the subtree rooted at x.) This property is maintained when a node is inserted by performing rotations and double rotations going up the insertion path to fix any nodes that are not weight balanced.

Let T be an arbitrary weight-balanced tree with n nodes, for some $n \geq 1$.

(a) Show that there is a leaf node in T that could be removed without unbalancing any subtrees of T (including T itself).

(b) Argue that T could have been created by a sequence of n insertions in such a way that *no* rotations were ever performed as T was built; i.e. that the growing tree was *always* weight-balanced.

Problem 3. Linked List Equivalence [15 points] (1 parts)

Let S and T be two sets of numbers, represented as unordered linked lists of distinct numbers. All you have are pointers to the heads of the lists, but **you do not know the list lengths**. Describe an $O(\min\{|S|, |T|\})$-expected-time algorithm to determine whether $S = T$. You may assume that any operation on one or two numbers can be performed in constant time.

Problem 4. Hash Table Analysis [15 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) What is the probability that the first slot ends up empty?

(b) What is the expected number of slots that end up not being empty?

Problem 5. Dynamic Programming [25 points] (1 parts)

You are given a sequence of n numbers (positive or negative):

$$x_1, x_2, \ldots, x_n$$

Your job is to select a subset of these numbers of maximum total sum, subject to the constraint that you can't select two elements that are adjacent (that is, if you pick x_i then you cannot pick either x_{i-1} or x_{i+1}).

Explain how you can find, in time polynomial in n, the subset of maximum total sum.