CS 240: Data Structures and Data Management **Winter 2025**

Midterm Practice Problem

Note: This is a sample of problems designed to help prepare for the midterm exam. These problems do *not* encompass the entire coverage of the exam, and should not be used as a reference for its content.

1. **True or False**
   (a) The midterm for this course is on February 27th at 4:30pm to 6:20pm.
   (b) Students are allowed to bring a single letter-sized (8.5x11) or smaller paper with anything written or printed on it on both sides.
   (c) If $T_1(n) \in \Omega(f(n))$ and $T_2(n) \in O(g(n))$, then $\frac{T_1(n)}{T_2(n)} \in \Omega(\frac{f(n)}{g(n)})$
   (d) If $\lim_{n\to\infty} \frac{f(n)}{g(n)} = e^{42}$, then $f(n) \in \Theta(g(n))$
   (e) If $f(n) \in \Theta(g(n))$, then $\lim_{n\to\infty} \frac{f(n)}{g(n)} = L$ where $0 < L < \infty$
   (f) If at least one rotation was performed during AVL-delete, then the height of the AVL Tree after deletion is strictly less than the height of the AVL Tree before deletion.
   (g) All heaps satisfy AVL tree's height-balance requirement
   (h) A binary search tree with n leaves must have height in O(n)
   (i) The average-case and expected-case run-time of an algorithm must always be the same
   (j) A skip-list with n keys and height h must have a total of $\Theta(nh)$ nodes.
   (k) If we perform an odd number of search operations in a linked list with the transpose heuristic, the resulting linked list will always be different from the initial linked list.

2. **Order Notation and Recurrence Relation**
   (a) Show that $3n^2 - 8n + 2 \in \Theta(n^2)$ from first principles.
   (b) Prove from first principle that $14n + 22$ is $o(nlogn)$
   (c) Prove from first principle that $n \in \omega(2^{\sqrt{\log n}})$
   (d) Given $T(1) = 1$, resolve $T(n) = T(\frac{3n}{4}) + n$ by providing a Θ bound.
   (e) Disprove the following statement - if $f(n) \in o(n \log n)$, then $f(n) \in O(n)$

3. **Pseudo-code Analysis**
   Analyze following pieces of pseudo-code and give a tight bound on the running time as a function of n.
   (a) Analyze the following piece of pseudo-code and give a tight (Θ) bound on the running time as a function of n.

```
i = 2
x = 0
while (i < n):
    for j = 1 to n:
        for k = 1 to j:
            x = x + 1
    i = i * i
```

(b) Give a tight big-O bound for the expected runtime of the following algorithm.

```
ArrayAlg(A, n, k)
    // n = A.size()
    // A is a permutation of [0, ..., n-1]
    // k is in the set {0, ..., n-1}
    i = random(n)
    if A[i] == k then
        return i
    for j = 0 to n-1
        print("a")
    return ArrayAlg(A, n, k)
```

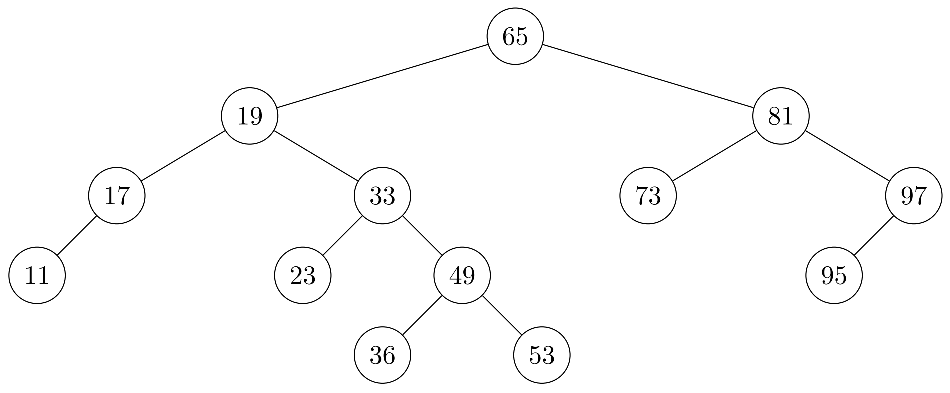
(c) Let A and B be two bit-strings of length n (modelled here as arrays where each entry is 0 or 1). A `string-compare` tests whether A is smaller, larger, or the same as B and works as follows:

```
str-cmp(A, B, n)
    for i = 0; i < n: i++ do
        if (A[i] < B[i]) then return "A is smaller"
        if (A[i] > B[i]) then return "A is bigger"
    return "They are equal"
```

Show that the average-case run-time of `str-cmp` is in $O(1)$. You may use without proof that $\sum_{i\geq 0} \frac{i}{2^i} \in O(1)$.

4. **Priority-Queue**
Given a family k sorted arrays $A_1, \ldots, A_k$, where the combination of the k arrays has n elements, give an $O(n \log k)$ time algorithm that produces a single sorted array containing all n elements. Hint: use a priority queue.

5. **Basics of AVL Tree**
Consider following AVL tree.



(a) Fill out height factor of each node. For example, node 33 will have height factor of 2.

(b) Fill out balance factor of each node. For example, node 33 will have height factor of 1.

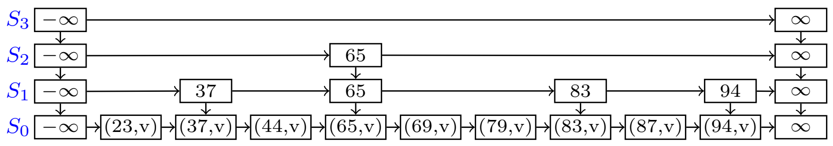
(c) Insert 61 into above AVL tree.

(d) Perform `delete(73)` on resulting tree (i.e. above tree after inserting 61)

6. **Lower bound finding**
Michael thinks he has discovered a new realization of priority-queues, which is comparison based and performs insert and delete-max in $O(\log\log n)$ time. Explain why this realization cannot be correct. Hint: do not tackle by talking about his implementation of priority queue.

7. **Epsilon**
Let $0 < \epsilon < 1$. Suppose that we have an array A of n items such that the first $n - n^{\epsilon}$ items are sorted. Describe an $O(n)$ time algorithm to sort A.

8. **Basic Skiplist**
Insert $(80, v)$ into the following skiplist with random coin flips $HHHHT$. Then, delete 83.



S3 −∞ ∞
S2 −∞ 65 ∞
S1 −∞ 37 65 83 94 ∞
S0 −∞ (23,v) (37,v) (44,v) (65,v) (69,v) (79,v) (83,v) (87,v) (94,v) ∞

9. **Numbers in Range**
We have an array A of n non-negative integers such that each integer is less than k. Give an $O(n+k)$ time preprocessing algorithm such that queries of the form “how many integers are there in A that are in the range $[a, b]$?” can be answered in $O(1)$ time. Note that a and b are not fixed; they are parameters given to the query algorithm.

10. **MTF Scenario**
Consider a linked list of n items where we perform m searches using the Move-To-Front heuristic, where $m > n$. For each of the following scenarios, give $\Theta()$ bounds on the worst-case runtimes in terms of m and n.

(a) 99% of the operations are on the same key x.

(b) At most $\sqrt{n}$ different elements are searched.

11. **Basic Trie**
Draw the uncompressed trie obtained by inserting the following strings into an initially empty trie:

$$1001\$, 001\$, 1111\$, 10110\$, 10\$, 11\$, 10100\$, 1\$, 000\$, 101\$$$