EECS 311: Data Structures and Data Management
Sample Midterm Questions

Advice: Skip problems that might take a while and come back to them later. If an algorithm is requested, be as succinct as possible, write your algorithms in pseodocode. You do not need to supply any proofs unless explicitly asked. All runtimes should be given using big-oh notation.

1. Simplify the following expressions if possible.

   (a) $O(3n^3 - 2n^2 + n)$.

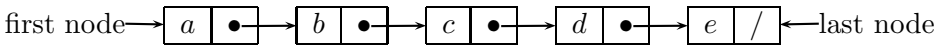
   (b) $O(\sqrt{n} + \log n)$.

   (c) $O(\log(n^9))$.

2. What is the runtime of the following code?

```
for (int i = 0; i < n; i++) {
   for (int j = 0; j < i; j++)
       cout << j << " ";
   cout << "\n";
}
```

3. Refer to the diagram below of a singly linked list to answer the following questions.



   (a) Suppose we implement a *queue* with a singly linked list. Which end of the list should items be dequeued, or does it not matter?

   (b) Suppose we implement a *stack* with a singly linked list. Which end of the list should items be popped, or does it not matter?

4. Define the *rank* of a key in a dictionary as the number of keys with value less than it. I.e., a key with rank i is the ith smallest key; the key with rank zero is the smallest key. Given an AVL tree implementation of a dictionary that supports *create*, *insert*, *delete*, and *find*, we wish to add support for an operation *find-ith* that returns the key with rank i.

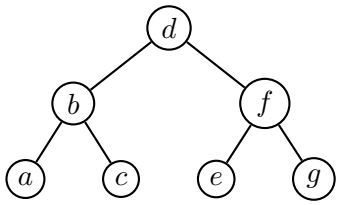
   (a) Your classmate Clarence proposes making the following change to the data structure: at each node, store the rank of the key. Clarence correctly suggests that with this change, *find-ith* can be written in $O(\log n)$ time. Unfortunately, with Clarence's proposal the other dictionary operations cannot be implemented in $O(\log n)$ time.

      i. Which ones?

      ii. Why not?

(b) Another classmate Claire suggests a subtle fix: instead of storing the rank of a key with respect to all keys in the dictionary, store the rank of the key with respect to its descendents in the tree. Call this the *local rank* of a key. Claire correctly suggests that with this modification all dictionary operations including the new find-ith operation can be implemented in $O(\log n)$ worst case runtime.

i. Referring to the binary search tree below, Give the *rank* and *local rank* of keys b, d, and f (i.e., fill in rest of the table).



key	rank	local rank
e	4	0
b		
d		
f		

ii. Give pseudo-code to implement *find-ith*.

iii. In order to complete our addition of the *find-ith* operation we must update local ranks appropriately on insertions and deletions. The main challenge in doing this is modifying the rotation procedures to maintain local ranks. Refer to the diagram below for *rotate-right* to describe how the local rank that is stored at each node (of b, d, and subtrees A, C, and E) should be updated after *rotate-right*.