Midterm 3: CompSci 201
Form A

Prof. Alex Steiger

April 17, 2024

General Directions

You should have an exam stapled with a separate answer sheet. Remove this answer sheet and verify it is a bubble sheet on the front side and fill-in-the-blank sheet on the back. Before you begin, **make sure to indicate your name and NetID**, and **verify that your form matches the answer sheet**. Do this in addition to signing this exam copy.

This exam has **20 problems** which are multiple choice (MC) and fill-in-the-blank problems of equal weight. Some MC problems have as few as two choices for answers, but every problem has five bubbles on the sheet. **Fill in bubble "A" for every fill-in-the-blank problem, then write your actual answer on the appropriate space on the back of the answer sheet.** If you erase to change an answer, be sure to erase completely.

You are welcome to use the exam itself for scratch work, but **only your answer sheet will be graded**.

For all problems you should assume that any necessary libraries (for example, from `java.util`) are imported. Where relevant, give the most tight analysis you can using big O notation. For example, if the running time is $O(N)$ then answering $O(N^2)$, while technically true, will not be considered correct.

You may not communicate with anyone while completing this exam. You may not access any electronic devices (including but not limited to phones, smartwatches, laptops, etc.) during the exam period. If you need to leave the exam room during the exam period, you should not communicate with anyone and should not access any electronic devices. You are allowed one 8.5x11 in. reference sheet that you bring with you.

You will have **60 minutes** to complete the exam. When you are finished, **turn in your answer sheet, your signed exam copy, and your reference sheet.**

The back of this first page contains code for the `ListNode` and `TreeNode` classes and common recurrences and their solutions which you may find useful.

By taking this exam, you are intending to, and promising to, adhere to the Duke Community Standard.

Print Name. _____ **NetID.** _____

Signature. _____ **Date.** _____

This Exam is Form A, please mark your answer sheet accordingly.

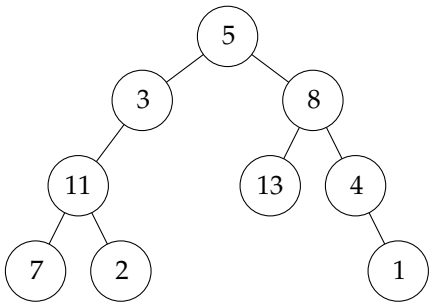
Common recurrences, their solutions, and the ListNode and TreeNode classes used throughout the test (same as lecture/APTs)

```
T(N) =  T(N/2) + O(1) -> O(log N)
T(N) =  T(N/2) + O(N) -> O(N)
T(N) = 2T(N/2) + O(1) -> O(N)
T(N) = 2T(N/2) + O(N) -> O(N log N)
T(N) =  T(N-1) + O(1) -> O(N)
T(N) =  T(N-1) + O(N) -> O(N^2)
T(N) = 2T(N-1) + O(1) -> O(2^N)
```

```
public class ListNode {
    int info;
    ListNode next;
    ListNode(int val) {
        info = val;
    }
    ListNode(int val, ListNode node) {
       info = val;
       next = node;
    }
```

```
public class TreeNode {
    int info;
    TreeNode left;
    TreeNode right;
    TreeNode(int val) {
        info = val;
    }
    TreeNode(int val, TreeNode lNode, TreeNode rNode) {
        info = val;
        left = lNode;
        right = rNode;
    }
}
```

The HowHeavy APT on APT Quiz 2 asks one to find, in a binary tree with ***distinct values*** (no duplicates), the sum of all values in the subtree rooted at the node with a given value `target`. If there is no such node with value `target`, 0 is to be returned (the weight of an empty subtree). For example, for the binary tree below, if `target=8` then 26 should be returned and if `target=10` then 0 should be returned (since 10 is not present in the tree).




```
1     private int sumAll(TreeNode tree) {
2         if (tree == null) return 0;
3         return EXPR_1;
4     }
5
6     private int weightIter(TreeNode tree, int target) {
7         if (tree == null) return 0;
8         Stack<TreeNode> toExplore = new Stack<>();
9         TreeNode current;
10        EXPR_2;
11        while (!toExplore.isEmpty()) {
12            current = toExplore.pop();
13            // LINE_3: if (???) { return ???; }
14            if (current.left != null) toExplore.push(current.left);
15            if (current.right != null) toExplore.push(current.right);
16        }
17        return 0;
18    }
```

Figure 1: weightIter method

Consider the incomplete solution above, called `weightIter`, which has access to helper function `sumAll`. `sumAll` should return the sum of all values in the given tree.

PROBLEM 1: What is the missing statement for `EXPR_1` in `sumAll`? Write it in the appropriate fill-in-the blank area of the back of the answer sheet. **Bubble A for this problem on the front of the answer sheet.**

PROBLEM 2: What is the missing statement for `EXPR_2` in `weightIter`? Write it in the appropriate fill-in-the blank area of the back of the answer sheet. **Bubble A for this problem on the front of the answer sheet.**

PROBLEM 3: What is the missing line for `LINE_3` in `weightIter`? It is of the form "`if (???) return ???;`" where `???` are expressions. Write it in the appropriate fill-in-the blank area of the back of the answer sheet. **Bubble A for this problem on the front of the answer sheet.**

```
1     public int weightRec(TreeNode tree, int target) {
2         return weightHelp(tree, target, false);
3     }
4
5     public int weightHelp(TreeNode tree, int target, boolean found) {
6         if (tree == null) return 0;
7         if (EXPR_4) {
8             return tree.info + weightHelp(tree.left, target, true)
9                              + weightHelp(tree.right, target, true);
10        } else {
11            return  weightHelp(tree.left, target, false)
12                  + weightHelp(tree.right, target, false);
13        }
14    }
```

Figure 2: weightIter method

Consider the incomplete solution above, called `weightRec`, which is a wrapper method for the recursive method `weightHelp`.

PROBLEM 4: What is the missing statement for `EXPR_4` in `weightRec`?

A. `!found`

B. `tree.info == target`

C. `found || tree.info == target`

D. `!found && tree.info == target`

PROBLEM 5: As described above, the HowHeavy APT assumes that the given binary tree contains distinct values (no duplicates). Consider the following variant of problem where the tree ***may*** have duplicates: Given a binary tree, ***possibly containing duplicate values***, and a value `target`, return the sum of all values that are contained in ***any*** subtree rooted at a node with given value `target`. That is, return the sum of all values which are at or below a node with value `target`; the values being counted may be in different subtrees. Which of `weightIter` and `weightRec` correctly solve this variant of the problem?

A. Neither solve this problem

B. `weightIter` ***is*** correct but ***not*** `weightRec`

C. `weightRec` ***is*** correct but ***not*** `weightIter`

D. Both solve this problem

```
1    public void traverse(TreeNode node) {
2        if (node == null) return;
3        traverse(node.right);
4        System.out.println(node.info + " ");
5        traverse(node.left);
6    }
```

Figure 3: traverse method

Consider the `traverse` method shown in Figure 3. This recursive method prints the info values of all nodes in the given tree in some order. The next two questions are about this method.

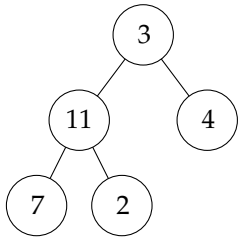


Figure 4: example tree

PROBLEM 6: What will be printed by calling the `traverse` method on the given binary tree in Figure 4? Write it in the appropriate fill-in-the blank area of the back of the answer sheet. **Bubble A for this problem on the front of the answer sheet.**

PROBLEM 7: What is the asymptotic runtime of `traverse` on any binary tree in terms of N, the number of nodes? The table of recurrence relations and their solutions on the back of the cover page may be useful.

A. $O(\log N)$

B. $O(N)$

C. $O(N \log N)$

D. $O(N^2)$

E. Cannot be determined—it depends on the balancedness of the tree

PROBLEM 8: A binary tree is *full and complete* if every level of the tree has as many nodes as possible, i.e., the level with the root has one node, the next level below has two nodes, the next level thereafter has four nodes, and so on. In a full and complete binary tree with N nodes, what best characterizes the *height* of the root, i.e., the number of edges on any root-to-leaf path?

A. $O(\log N)$

B. $O(N)$

C. $O(N \log N)$

D. $O(N^2)$

E. $O(2^N)$

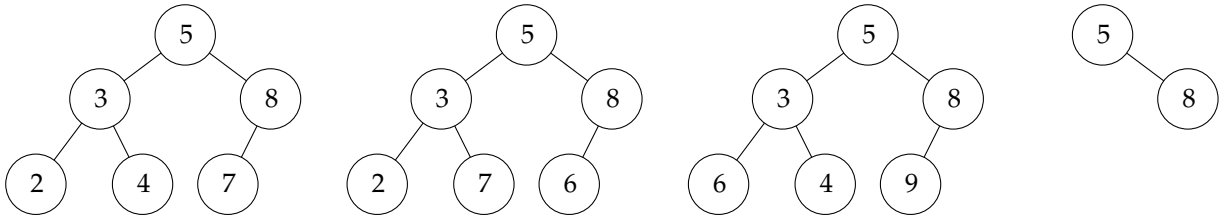


Figure 5: four binary trees

PROBLEM 9: How many of the above binary trees in Figure 5 are ***binary search trees***?

A. 0

B. 1

C. 2

D. 3

E. 4

PROBLEM 10: How many of the above binary trees in Figure 5 are ***binary heaps***?

A. 0

B. 1

C. 2

D. 3

E. 4

PROBLEM 11: **True or False**: For any binary search tree, any value not already contained in the tree can be inserted as a leaf of the tree so that it remains a binary search tree.

A. True

B. False

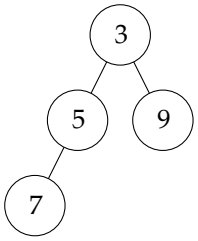


Figure 6: example heap

The next two problems are about the binary heap above in Figure 6. We write the array representation of the heap using the 1-indexing convention that leaves an X at the beginning to mark the blank position 0. For example, we write the array representation of the binary heap above as: [X, 3, 5, 9, 7].

PROBLEM 12: Suppose you add 4 and then 2 to this heap, in that order, one-by-one, using the insertion procedure for array-based binary heaps from lecture. What is the array representation of the resulting heap, using the format above? Write it in the appropriate fill-in-the blank area of the back of the answer sheet. **Bubble A for this problem on the front of the answer sheet.**

PROBLEM 13: Suppose you remove the minimum value, 3, from this heap (without performing the additions from the previous problem). What is the array representation of the resulting heap, using the format above? Write it in the appropriate fill-in-the blank area of the back of the answer sheet. **Bubble A for this problem on the front of the answer sheet.**

MrBeast (a popular YouTuber) is organizing their schedule. He has many events coming up that he may be able to attend. Treating the current time as time 0, each event is specified by two integers, `beginTime` and `endTime`, where `beginTime` is the number of hours from now that the event begins, and `endTime` is the number of hours from now that the event will end. (For all events, `beginTime < endTime`.)

MrBeast wants to maximize the number of events that he can attend, which all have to satisfy two constraints: (a) He can only attend one event at a time (there is only one of him, after all) and (b) any event must be attended in full. For sake of this problem, suppose MrBeast can instantly attend an event that begins immediately after another attended event; for example, he can attend an event that ends at hour 5 then another event that begins at hour 5.

To decide the maximum number of events he can attend, he employs the following greedy strategy: From the current moment in time (originally time 0 before all events start), choose the next event that ***ends earliest among the events that begin later***. This greedy strategy always returns the maximum number of events that can be attended, the following code correctly implements this greedy strategy.

```
1     private class Event {
2         int beginTime;
3         int endTime;
4         Event(int b, int e) { beginTime = b; endTime = e; }
5     }
6
7     public int maxEvents(List<Event> list) {
8         int count = 0;
9         int currTime = 0;
10        Comparator<Event> comp = (a,b) ->  a.endTime - b.endTime;
11        PriorityQueue<Event> pq = new PriorityQueue<>(comp);
12        pq.addAll(list);
13        while (pq.size() > 0) {
14            Event current = pq.remove();
15            if (currTime <= current.beginTime) { // if begins later
16                currTime = current.endTime; // attend until end
17                count++;
18            }
19        }
20        return count;
21    }
```

Figure 7: maxEvents method

PROBLEM 14: Suppose MrBeast's possible events are as depicted below in Figure 8: at hours 1-4, 2-3, 3-7, 5-9, 6-10, 9-14, 10-11, and 12-15. What is the maximum number of events that MrBeast can attend? *[Hint: Run the algorithm above, or solve it through any other means. MrBeast can only attend events that do not overlap at all or only overlap at their endpoints.]*

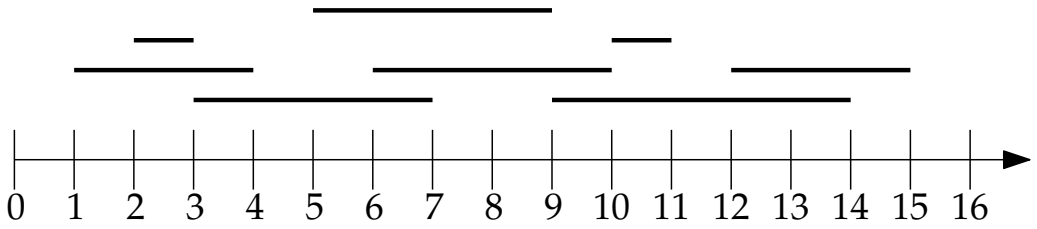


Figure 8: example events

A. 2

B. 3

C. 4

D. 5

E. None of the above

PROBLEM 15: What is asymptotic runtime of `maxEvents` given a list of N events?

A. $O(\log N)$

B. $O(N)$

C. $O(N \log N)$

D. $O(N^2)$

E. $O(N^2 \log N)$

As discussed in lecture, AVL trees are an implementation of self-balancing binary search trees. Let the *height* of a node in a binary tree by the number of ***nodes*** on the longest path from that node to any leaf of the tree. For example, leaf nodes have height 1, and the parent of two leaf nodes has height 2.

AVL trees use the concept of the *balance factor* of a node, which is the difference of the height of the node's right child and the node's left child. For example, a leaf node has balance factor 0. The example binary search tree in Figure 9 below shows each node's balance factor inside the node; the actual values in the search tree are not shown. In an AVL tree, a node is considered *balanced* if its balance factor is -1, 0, or 1.

The following method `numBalanced` correctly returns the number of nodes that are balanced in the given tree. For example, it returns 7 for the example tree shown.

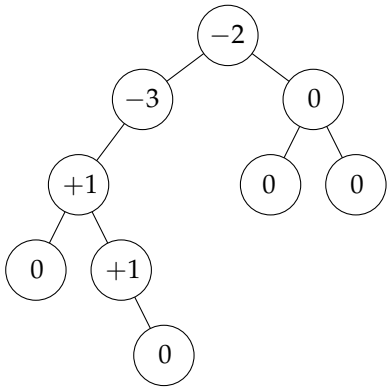


Figure 9: example AVL tree with balance factors

```
1     public int height(TreeNode node) {
2         if (node == null) return 0;
3         return 1 + Math.max(height(node.left), height(node.right));
4     }
5
6     public int numBalanced(TreeNode node) {
7         if (node == null) return 0;
8         int balFactor = height(node.right) - height(node.left);
9         int count = 0;
10        if (-1 <= balFactor && balFactor <= 1) count++;
11        count += numBalanced(node.left) + numBalanced(node.right);
12        return count;
13    }
```

Figure 10: numBalanced and height methods

PROBLEM 16: What is the asymptotic runtime complexity of `numBalanced` on an ***balanced*** tree in terms of N, the number of nodes? The table of recurrence relations and their solutions on the back of the cover page may be useful. *[Hint: First, what is the asymptotic runtime of* `height`*?]*

A. $O(1)$

B. $O(\log N)$

C. $O(N)$

D. $O(N \log N)$

E. $O(N^2)$

PROBLEM 17: What is the asymptotic runtime complexity of `numBalanced` on an ***unbalanced*** tree in terms of N, the number of nodes? The table of recurrence relations and their solutions on the back of the cover page may be useful. *[Hint: First, what is the asymptotic runtime of `height`?]*

A. $O(1)$

B. $O(\log N)$

C. $O(N)$

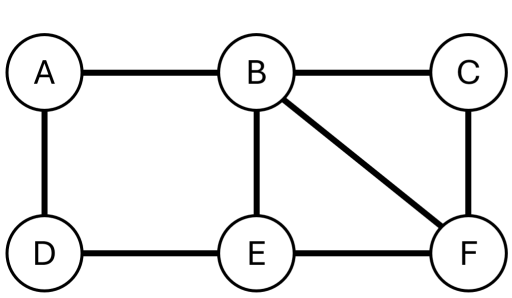
D. $O(N \log N)$

E. $O(N^2)$

```
1   public void RAFS(Map<Character, List<Character>> aList, char start) {
2       HashSet<Character> visited = new HashSet<>();
3       PriorityQueue<Character> toExplore =
4               new PriorityQueue<>(Comparator.reverseOrder());
5       toExplore.add(start);
6       visited.add(start);
7       while (!toExplore.isEmpty()) {
8           Character current = toExplore.remove();
9           System.out.print(" " + current);
10          for (Character neighbor : aList.get(current)) {
11              if (!visited.contains(neighbor)) {
12                  visited.add(neighbor);
13                  toExplore.add(neighbor);
14              }
15          }
16      }
17  }
```

Figure 11: RAFS method

Consider the `RAFS` method shown in Figure 11. This method performs a graph traversal, where the nodes in `toExplore` are ordered in ***reverse alphabetical order*** as opposed to the temporal orderings of BFS and DFS. The graph is represented by an adjacency list, and its nodes are represented by single characters. The next two questions are about this method and the example graph below in Figure 12. The neighbors of each node are ordered as shown in its adjacency list.



Adjacency List:
A=[B, D]
B=[A, C, E, F]
C=[B, F]
D=[A, E]
E=[B, D, F]
F=[B, C, E]

Figure 12: example graph

PROBLEM 18: What is printed when calling the method on the example graph and `start = 'C'`?

A. F D E C B A

B. C B F A E D

C. C F E D B A

D. C B A D E F

E. None of the above

PROBLEM 19: Suppose we replace `toExplore` with a `Queue` instead of the `PriorityQueue`. Specifically, suppose lines 3-4 are replaced with

```
Queue<Character> toExplore = new LinkedList<>();
```

With this modification, what is printed when calling the method on the example graph and `start = 'C'`?

A. F D E C B A

B. C B F A E D

C. C F E D B A

D. C B A D E F

E. None of the above

PROBLEM 20: **True or False**: There exists graphs where, when starting from the same node, BFS and DFS explore nodes in the same order.

A. True

B. False

This page intentionally left blank. You may use it for scratch if you wish, but you should submit it with the exam.