Midterm 3: Compsci 201
Form B

Profs. Astrachan and Nemecek

November 19/20, 2025

Name: ____

NetID: ____

In submitting this test, I affirm that I have followed the Duke Community Standard.

Community standard acknowledgement (signature) ______________________________________

You should bubble in answers for 25 questions on this exam.

The bubble sheet is for multiple choice questions. On the other/back side you'll find areas for fill-in-the blank questions. Please bubble in an answer for every question, choosing **Option A** for fill-in-the-blank questions as directed.

Common Recurrences and their solutions:

label	recurrence	solution
A	`T(n) = T(n/2) + O(1)`	$O(\log n)$
B	`T(n) = T(n/2) + O(n)`	$O(n)$
C	`T(n) = 2T(n/2) + O(1)`	$O(n)$
D	`T(n) = 2T(n/2) + O(n)`	$O(n \log n)$
E	`T(n) = T(n-1) + O(1)`	$O(n)$
F	`T(n) = T(n-1) + O(n)`	$O(n^2)$
G	`T(n) = 2T(n-1) + O(1)`	$O(2^n)$

This Exam is Form B, please make sure your answer sheet is marked accordingly!

`TreeNode` and `ListNode` classes as used on this test. In some problems the type of the `info` field may change from `int` to `String` and *vice versa*.

```
public class TreeNode {
    String info;
    TreeNode left;
    TreeNode right;

    TreeNode(String x){
        info = x;
    }
    TreeNode(String x,TreeNode lNode,
                      TreeNode rNode){
        info = x;
        left = lNode;
        right = rNode;
    }
}
```

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

Tree Traversal Code

```
public void inOrder(TreeNode root) {
    if (root != null) {
        inOrder(root.left);
        System.out.println(root.info);
        inOrder(root.right);
    }
}
```

```
public void preOrder(TreeNode root) {
    if (root != null) {
        System.out.println(root.info);
        preOrder(root.left);
        preOrder(root.right);
    }
}
```

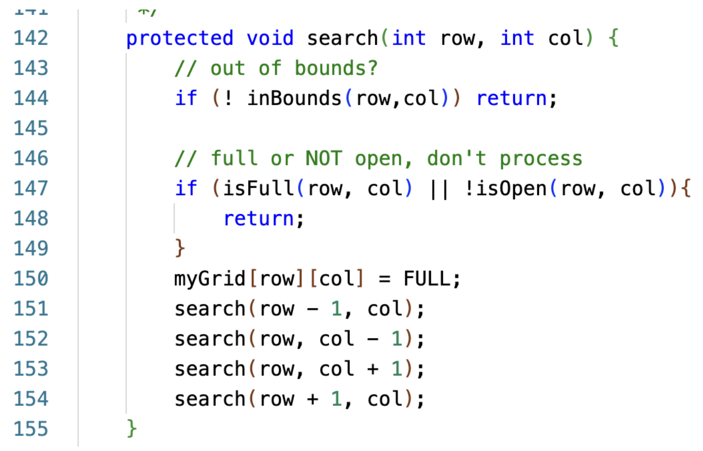
```
public void postOrder(TreeNode root) {
    if (root != null) {
        postOrder(root.left);
        postOrder(root.right);
        System.out.println(root.info);
    }
}
```

This Exam is Form B, please make sure your answer sheet is marked accordingly!

The next three problems are about P5: Percolation, in which you ran simulations using different models to estimate what's called the *percolation threshold*.

PROBLEM 1:

The code below is the method `PercolationDefault.search`, a recursive version of depth-first search.



```
142     protected void search(int row, int col) {
143         // out of bounds?
144         if (! inBounds(row,col)) return;
145 
146         // full or NOT open, don't process
147         if (isFull(row, col) || !isOpen(row, col)){
148             return;
149         }
150         myGrid[row][col] = FULL;
151         search(row - 1, col);
152         search(row, col - 1);
153         search(row, col + 1);
154         search(row + 1, col);
155     }
```

The code above visits the neighbor(s) of a cell in the order up-left-right-down. If lines 151-154 are shuffled into another order, e.g., left-right-down-up (there are 24 such orders) will the estimate of p*, the percolation threshold, change?

A. The estimate will not change. It's possible that timings will not be identical, but they will be similar.

B. The estimate will change. Going up first provides a better estimate of p* since the top is the area from which water flows.

C. It cannot be determined if the estimate will change.

PROBLEM 2:

The class `PercolationDefault` uses a recursive version of depth-first search. You were asked to write `PercolationDFS` that uses a `java.util.Stack` object rather than recursion.

Does `PercolationDFS` allow for bigger grids to be explored/simulated than `PercolationDefault`?

A. No, the maximum grid size that can be explored/simulated is the same for both.

B. Yes, the maximum grid size that can be explored/simulated is greater with `PercolationDFS`.

C. The answer depends on the seed of the Random Number generator used to open cells.

The next three questions are about P4: Autocomplete.

PROBLEM 3:

The classes `BruteAutocomplete` and `BinarySearchAutocomplete` return the same value for the method `sizeInBytes` when they're constructed from the same data, calculated from the instance variable `Term[] myTerms` that they each have.

The class `HashListAutocomplete` does **not store** this instance variable. When a `HashListAutocomplete` object `hasher` is constructed from the same data as the other two classes, which one of the following is true about the value returned by a correct implementation of `hasher.sizeInBytes()`?

A. The value is less than what's returned by the other classes constructed from the same data.

B. The value is equal to what's returned by the other classes constructed from the same data.

C. The value is greater than what's returned by the other classes constructed from the same data.

PROBLEM 4:

You were given code for `BruteAutocomplete` and you were asked to implement `BinarySearchAutocomplete` and `HashListAutocomplete` as part of P4, that's three *Autocomplete* classes.

These three classes worked with two *client* classes `AutocompleteMain` and `BenchMarkForAutocomplete`. Which of the following best explains why these last two classes could work with all three `Autocomplete` classes?

A. Each of the three classes implement the `Autocompletor` interface and the two client classes are written to work with that interface.

B. The two client programs have sequences of `if` statements for the code to determine which version of an *Autocomplete* class to call.

C. There are three versions of each *client* program: one for each *Autocomplete* class.

PROBLEM 5:

Which best explains why you were asked to write method `BinarySearchLibrary.firstIndex` instead of relying on the `java.util.BinarySearch` class, which runs in $O(\log(n))$ time, to find a value in a list of n values?

A. The Java library generates an exception when the list is too large and the calculation of `(first+last)/2` generates integer overflow.

B. The Java library returns a value matching the search term, but not necessarily the value with the lowest index when more than one value matches the search term.

C. Implementing a function even when it's in a standard library teaches you how to test and develop code, and in this case that development used a loop invariant.

The code shown below sorts its parameter `list`.

```java
public void sortPQ(List<String> list) {
    PriorityQueue<String> pq =
            new PriorityQueue<>(Comparator.comparing(String::length));
    pq.addAll(list);
    list.clear();
    while (pq.size() > 0) {
        list.add(pq.remove());
    }
}
```

PROBLEM 6:

If `list` contains N elements, what is the runtime of `sortPQ(list)` ?

A. $O(N)$

B. $O(N\log(N))$

C. $O(N^2)$

PROBLEM 7:

If the contents of `list` are

```
list = ["burgundy", "vermillion", "maroon", "red"]
```

What are the contents of `list` after the call `sortPQ(list)`?

A. `[burgundy, vermillion, maroon, red]`

B. `[red, maroon, burgundy, vermillion]`

C. `[burgundy, maroon, red, vermillion]`

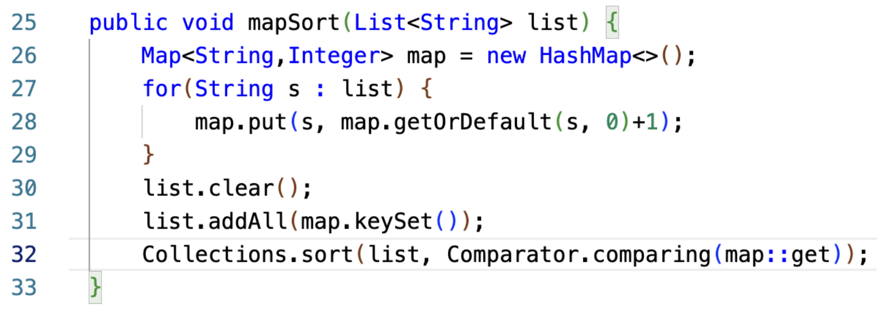
D. `[vermillion, burgundy, maroon, red]`

PROBLEM 8:

Consider an `ArrayList<String> list` with values as shown:

```
[one, two, two, three, three, three, four, four, four, four]
```

What are the contents of `list` after the call `mapSort(list)`?



```
25  public void mapSort(List<String> list) {
26      Map<String,Integer> map = new HashMap<>();
27      for(String s : list) {
28          map.put(s, map.getOrDefault(s, 0)+1);
29      }
30      list.clear();
31      list.addAll(map.keySet());
32      Collections.sort(list, Comparator.comparing(map::get));
33  }
```

A. `[one, two, two, three, three, three, four, four, four, four]`

B. `[four, one, two, three]`

C. `[one, two, three, four]`

D. `[four, three, two, one]`

PROBLEM 9:

Does the order of the values in parameter `list` when method `mapSort` finishes change if `new TreeMap<>()` replaces `new HashMap<>()` on line 26?

A. There is some parameter `list` for which the order of the values will change if `new TreeMap<>()` is used, but not for all values of `list`.

B. The order of values in `list` will *always* be the same when `TreeMap` is used instead of `HashMap`.

C. The order of values in `list` will *always* be different when `TreeMap` is used instead of `HashMap`.

PROBLEM 10:

Consider a stack `st` holding the values shown below, with `orange` at the top of the stack and `cherry` at the bottom, i.e., `cherry` was pushed onto the stack first.

```
st = ["cherry", "guava", "mango", "orange"]
```

The call `st = repeat(st)` stores values in `st` as shown below (cherry now at the top of the stack):

```
st = ["orange", "mango", "guava", "cherry"]
```

```
public static Stack<String> repeat(Stack<String> st) {
    List<String> list = new ArrayList<>();
    while (st.size() > 0) list.add(st.pop());
    for(String s : list) st.push(s);
    return st;
}
```

If there are n calls of `repeat` nested, e.g.,

```
st = repeat(repeat(repeat(...(repeat(st)))));
```

What values of n will leave the order of elements in `st` unchanged after the nested calls?

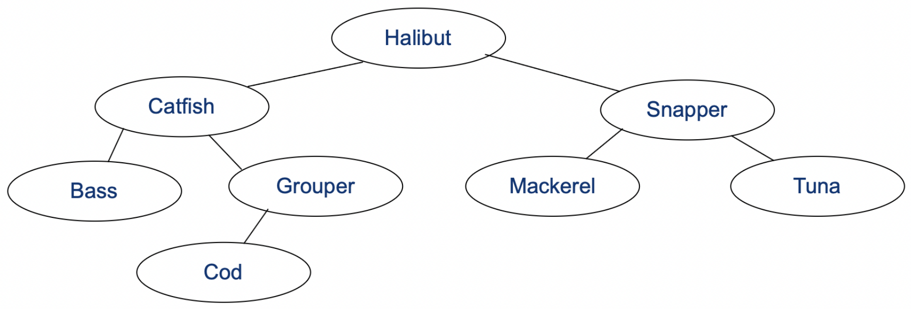
A. $n = 2$ only

B. all n such that `n % 2 == 0` (even n).

C. all n such that `n % 2 == 1` (odd n).

PROBLEM 11:

Consider the tree below.



What is printed as a result of the call `traverse(root)` if `root` points at Halibut?

```java
public void traverse(TreeNode root) {
    Queue<TreeNode> list = new LinkedList<>();
    if (root != null) {
        list.add(root);
    }
    while (list.size() > 0) {
        root = list.remove();
        System.out.printf("%s\n",root.info);
        if (root.right != null) list.add(root.right);
        if (root.left != null) list.add(root.left);
    }
}
```

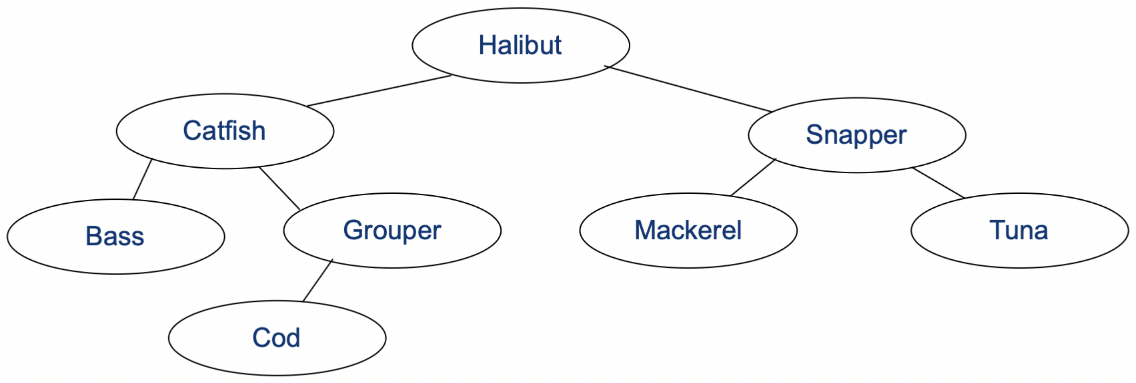
A. Halibut, Catfish, Snapper, Bass, Grouper, Mackerel, Tuna, Cod

B. Halibut, Snapper, Catfish, Tuna, Mackerel, Grouper, Bass, Cod

C. Bass, Catfish, Cod, Grouper, Halibut, Mackerel, Snapper, Tuna

D. Halibut, Catfish, Bass, Grouper, Cod, Snapper, Mackerel, Tuna

The next three problems use this tree.



PROBLEM 12:

The tree shown above is a search tree. If Haddock is added as the *right child of Grouper*, is the tree still a search tree?

A. Yes, it will still be a search tree

B. No, it will no longer be a search tree

PROBLEM 13:

What are the last four values visited in a *post-order traversal* of the tree shown above whose root is Halibut?

A. Halibut, Snapper, Mackerel, Tuna

B. Halibut, Mackerel, Tuna, Snapper

C. Mackerel, Tuna, Snapper, Halibut

D. Tuna, Mackerel, Snapper, Halibut

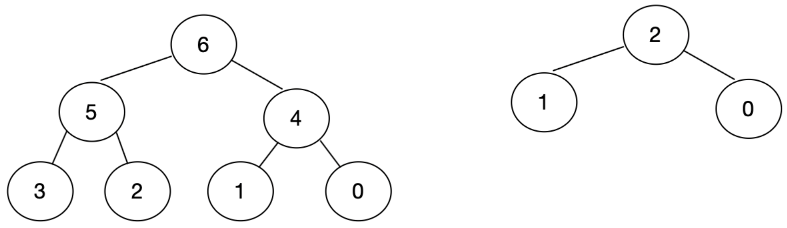
PROBLEM 14:

If Salmon and Pickerel are added to the tree, in that order, so that it is still a search tree, the height of the resulting tree will be five rather than four as it is shown above.
If Pickerel is added first, and then Salmon, will the height of the resulting tree still be five?

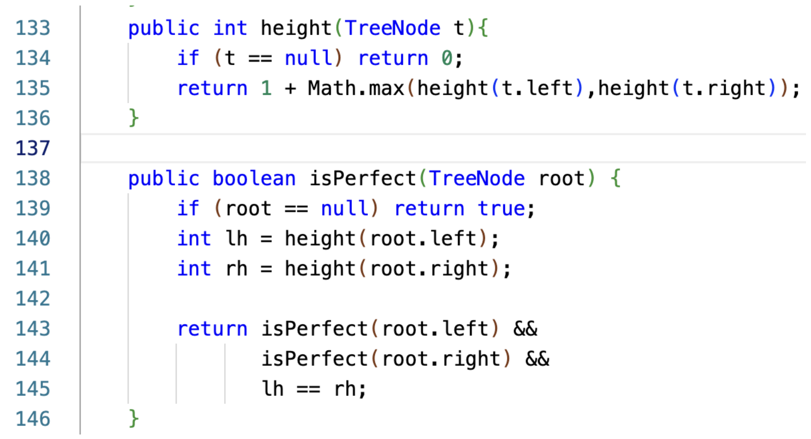
A. Yes, it will be five

B. No, it will be four

The next four problems concern a *perfect binary tree*. In such a tree every node except leaf nodes has two children and all the leaves are at the same level. The diagram below shows two *perfect* trees:



One way to determine if a tree is perfect is: if it is true that for *every node* both: (1) its left and right subtrees are perfect and (2) the height of the left subtree is equal to the height of the right subtree. The code shown below captures this method of determining if a tree is a perfect tree. It returns true for both trees shown above and, in general, returns true if and only if parameter `root` references a perfect tree.



```
public int height(TreeNode t){
    if (t == null) return 0;
    return 1 + Math.max(height(t.left),height(t.right));
}

public boolean isPerfect(TreeNode root) {
    if (root == null) return true;
    int lh = height(root.left);
    int rh = height(root.right);

    return isPerfect(root.left) &&
           isPerfect(root.right) &&
           lh == rh;
}
```

PROBLEM 15:

What is the runtime of `isPerfect(root)` if `root` is a roughly balanced tree of N nodes?

A. $O(N)$

B. $O(N \log(N))$

C. $O(N^2)$

PROBLEM 16:

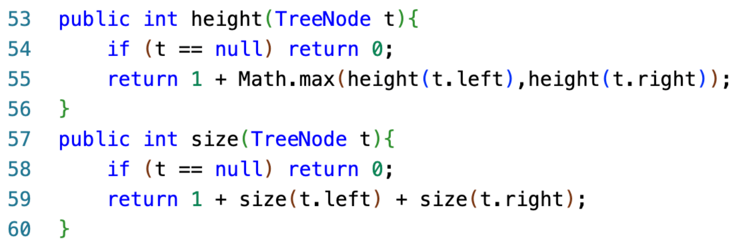
What is the runtime of `isPerfect(root)` if `root` is a completely unbalanced tree of N nodes?

A. $O(N)$

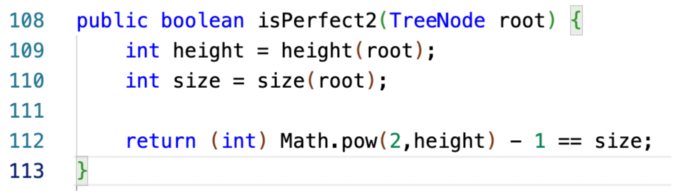
B. $O(N \log(N))$

C. $O(N^2)$

It can be proved that a binary tree T is perfect if and only if $2^{height(T)} - 1 = size(T)$, so the method `isPerfect2` shown below is correct.



```java
public int height(TreeNode t){
    if (t == null) return 0;
    return 1 + Math.max(height(t.left),height(t.right));
}
public int size(TreeNode t){
    if (t == null) return 0;
    return 1 + size(t.left) + size(t.right);
}
```



```java
public boolean isPerfect2(TreeNode root) {
    int height = height(root);
    int size = size(root);

    return (int) Math.pow(2,height) - 1 == size;
}
```

PROBLEM 17:

What is the runtime of `isPerfect2(root)` when `root` is a roughly balanced tree of N nodes?

A. $O(N)$

B. $O(N \log(N))$

C. $O(N^2)$

PROBLEM 18:

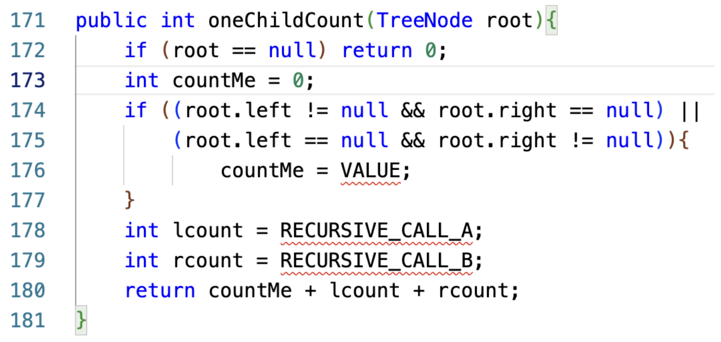
What is the runtime of `isPerfect2(root)` when `root` is a completely unbalanced tree of N nodes?

A. $O(N)$

B. $O(N \log(N))$

C. $O(N^2)$

In a perfect tree, every node has either zero children (leaves) or two children (internal nodes). The method `oneChildCount` below is intended to return the number of nodes in its parameter that have exactly one child (neither zero nor two).



```
171  public int oneChildCount(TreeNode root){
172      if (root == null) return 0;
173      int countMe = 0;
174      if ((root.left != null && root.right == null) ||
175          (root.left == null && root.right != null)){
176              countMe = VALUE;
177      }
178      int lcount = RECURSIVE_CALL_A;
179      int rcount = RECURSIVE_CALL_B;
180      return countMe + lcount + rcount;
181  }
```

PROBLEM 19:

Bubble A for this question on the front of the answer sheet and fill-in-the blank area for this question on the back of the bubble answer sheet.

Assuming lines 178 and 179 are filled in correctly, what should `VALUE` be on line 176 so that the code works correctly?

PROBLEM 20:

Bubble A for this question on the front of the answer sheet and fill-in-the blank area for this question on the back of the bubble answer sheet.

If lines 176 and 179 are are completed correctly, what is the value of `RECURSIVE_CALL_A` on **line 178** so that the code works correctly? (Note: `RECURSIVE_CALL_B` is similar, but you are not asked about line 179.)

PROBLEM 21:

If lines 176, 178, and 179 are filled in correctly, the runtime of `oneChildCount` will be the same whether its tree parameter is roughly balanced or completely unbalanced. What is the runtime for a tree with N nodes?

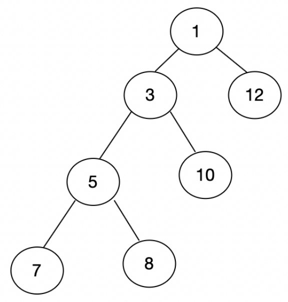
A. $O(\log(N))$

B. $O(N)$

C. $O(N \log(N))$

D. $O(N^2)$

Consider finding all root-to-leaf paths in a tree, and the related problem of finding the longest root-to-leaf path in a tree. For the tree shown below, the longest path would be `1-3-5-7` (or `1-3-5-8` which has the same maximal length).



PROBLEM 22:

If there are N leaves in a tree, there will be N root-to-leaf paths in the tree. What is the shortest possible *longest* root-to-leaf path in a tree with N leaves (consider the tree of N leaves that has minimal height)?

A. $O(\log(N))$

B. $O(N)$

C. $O(N \log(N))$

D. $O(N^2)$

PROBLEM 23:

If there are N leaves in a tree, there will be N root-to-leaf paths in the tree. What is the longest possible root-to-leaf path in a tree with N leaves (consider the tree of N leaves that has maximal height)?

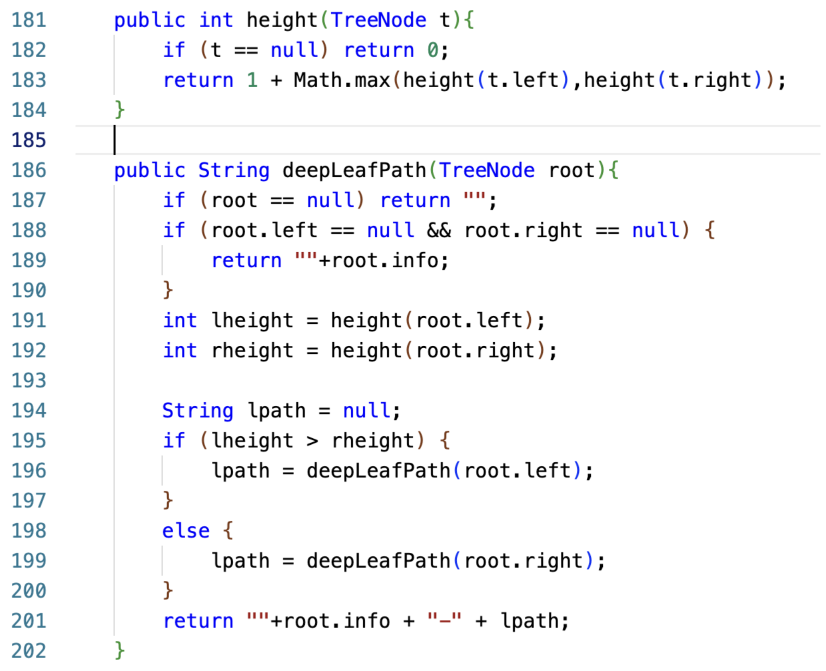
A. $O(\log(N))$

B. $O(N)$

C. $O(N \log(N))$

D. $O(N^2)$

The code below returns a maximal root-to-leaf path in the tree passed as a parameter. For the tree shown above, it will return `1-3-5-8` (which is one of two longest paths).



```java
public int height(TreeNode t){
    if (t == null) return 0;
    return 1 + Math.max(height(t.left),height(t.right));
}

public String deepLeafPath(TreeNode root){
    if (root == null) return "";
    if (root.left == null && root.right == null) {
        return ""+root.info;
    }
    int lheight = height(root.left);
    int rheight = height(root.right);

    String lpath = null;
    if (lheight > rheight) {
        lpath = deepLeafPath(root.left);
    }
    else {
        lpath = deepLeafPath(root.right);
    }
    return ""+root.info + "-" + lpath;
}
```

PROBLEM 24:

What is the runtime of `deepLeafPath(t)` when `t` is a roughly balanced tree of N nodes?

A. $O(N)$

B. $O(N \log(N))$

C. $O(N^2)$

PROBLEM 25:

What is the runtime of `deepLeafPath(t)` when `t` is a completely unbalanced tree of N nodes?

A. $O(N)$

B. $O(N \log(N))$

C. $O(N^2)$