Test 2 Practice : Compsci 201

Owen Astrachan

October 24, 2018

Name: ________________________________

NetID/Login: ________________

Community standard acknowledgment (signature) __________________________

<table>
<thead>
<tr>
<th>Problem</th>
<th>Value</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>NetID</td>
<td>1 pt.</td>
<td></td>
</tr>
<tr>
<td>Problem 1</td>
<td>8 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 2</td>
<td>18 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 3</td>
<td>16 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 4</td>
<td>12 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 5</td>
<td>16 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 6</td>
<td>12 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 7</td>
<td>32 pts.</td>
<td></td>
</tr>
<tr>
<td>TOTAL:</td>
<td>114 pts.</td>
<td></td>
</tr>
</tbody>
</table>

This test has 20 pages, be sure your test has them all. Write your NetID clearly on each page of this test (worth 1 point).

In writing code you do not need to worry about specifying the proper import statements. Don’t worry about getting function or method names exactly right. Assume that all libraries and packages we’ve discussed are imported in any code you write. You can write any helper methods you would like in solving the problems. You should show your work on any analysis questions.

You may consult your six (6) note sheets and no other resources. You may not use any computers, calculators, cell phones, or other human beings. Any note sheets must be turned in with your test.
PROBLEM 1:  (It Depends (8 points))

Part A (3 points)
What value is returned by the call calculate(2043) ? What is the runtime complexity (big-Oh, in terms of N) of the call calculate(N) ? Briefly justify your answers.

```java
public int calculate(int n){
    int prod = 1;
    while (prod < n){
        prod *= 2;
    }
    return prod;
}
```

Part B (5 points)
Consider method docalc below, the call docalc(6) evaluates to 21.

```java
public int docalc(int n){
    if (n == 0) return 0;
    return n + docalc(n-1);
}
```

Using big-Oh what is the running time of the call docalc(n) ? Justify your answer.

Using big-Oh what is the value returned by the call docalc(n) (note: complexity of value returned, not running time: use big-Oh)

Using big-Oh what is the running time of the call docalc(docalc(n)) based on your answers to the previous two questions. Justify.

Using big-Oh what is the value returned by the call docalc(docalc(n)) (again, based on previous answers, justify).
The **ArrayList** class and the **LinkedList** class both implement the `java.util.List` interface. The code below removes elements from a `List` until the list is empty – this code has a runtime of $O(n)$ for both **ArrayList** and **LinkedList** lists that contain $n$ elements. Recall that **ArrayList** objects have an internal array to store elements whereas **LinkedList** objects have an internal doubly-linked list to store elements.

```java
public double removeLast(List<String> list) {
    double start = System.nanoTime();
    while (list.size() != 0) {
        list.remove(list.size()-1);
    }
    double end = System.nanoTime();
    return (end - start) / 1e9;
}
```

**Part 1.1 (2 points)**
Which element is removed from the list by the call `list.remove(list.size()-1)`?

**Part 1.2 (6 points)**
Suppose a similar method `removeFirst` is implemented — identical to the code above except the single statement in the body of the loop is `list.remove(0)` — the method will have a runtime of $O(n)$ for one of **ArrayList** and **LinkedList** objects, but not for both.

**Briefly justify both answers below.**

What is the big-Oh runtime complexity of `removeFirst(list)` when list is an **ArrayList** and why?

What is the big-Oh runtime complexity of `removeFirst(list)` when list is a **LinkedList** and why?
Part 1.3 (4 points)

The method `total` below correctly calculates the sum of all values in a `List<Integer>` so that `total(list)` will work when `list` is an `ArrayList` and when it is a `LinkedList`. Explain why this method has a runtime of $O(n)$ to calculate the sum of all values in an `ArrayList` and a runtime of $O(n^2)$ to calculate the sum of all values in a `LinkedList`.

```java
public int total(List<Integer> list) {
    int sum = 0;
    for(int k=0; k < list.size(); k++){
        sum += list.get(k);
    }
    return sum;
}
```

Why $O(n)$ for an n-element `ArrayList`

Why $O(n^2)$ for an n-element `LinkedList`
Part 1.4 (4 points)
The method \texttt{itotal} below correctly calculates the sum of all values in a \texttt{List<Integer>} so that \texttt{itotal(list)} will work when \texttt{list} is an \texttt{ArrayList} and when it is a \texttt{LinkedList}. The big-Oh runtime complexity is the same for both types of list when \texttt{itotal} is called with a list of \textit{n} elements. What is the big-Oh complexity. Briefly justify your answer..

\begin{verbatim}
public int itotal(List<Integer> list) {
    int sum = 0;
    for(int val : list) {
        sum += val;
    }
    return sum;
}
\end{verbatim}

What is the big-Oh runtime of \texttt{itotal} for both \texttt{ArrayList} and \texttt{LinkedList} (it’s the same expression)

Why is this the complexity for both types of \texttt{List}?
PROBLEM 3 : (Orings, Othings (16 points))

In this problem you’ll reason about two methods that each return a tree with the same shape as a parameter tree. In the returned tree each node’s .info field is equal to the number of nodes of the tree with that node as root. For example, given the tree below on the left, the returned tree is shown on the right. In the tree shown below the returned tree’s root has the value 11, where as the tree’s right subtree has the value 8 as its root.

Part 3.1 (2 points)
What value is stored in every leaf of the returned tree?

Part 3.2 (2 points)
If the root is at level zero, the trees above have deepest leaves at level 4. By definition in a complete tree there are are $2^k$ nodes at level $k$ for every level $k = 0, 1, \ldots$; and there a total of $2^{n+1} - 1$ nodes in a complete tree whose deepest leaves are at level $n$. What value is stored in the root of the tree returned if the tree parameter is a complete tree with deepest nodes at level 9? You must supply an exact, numerical answer.
Part 3.3 (4 points)
The method countLabel shown below correctly returns a tree with the same shape.
Label each line below with an expression involving $O(\cdot)$ or $T(\cdot)$ when countLabel is called with an $N$ node tree, so $T(N)$ is the time for countLabel to execute with an $N$ node tree.

On this page label lines for the average case. Be sure to label each line of method countLabel with either an $O(\cdot)$ expression or a $T(\cdot)$ expression for the average case when trees are roughly balanced. $T(N)$ is the time for countLabel to run.
You don’t need to label code in the method count. You do need to label the call of count in countLabel.

```java
public TreeNode countLabel(TreeNode tree) {
    if (tree == null) return null;
    TreeNode left = countLabel(tree.left);
    TreeNode right = countLabel(tree.right);
    int size = count(tree);
    return new TreeNode(size, left, right);
}
```

```java
public int count(TreeNode tree) {
    if (tree == null) return 0;
    return 1 + count(tree.left) + count(tree.right);
}
```

Be sure to write the recurrence relation and its solution.
Part 3.4 (4 points)

Label each line below with an expression involving $O(\cdot)$ or $T(\cdot)$ when `countLabel` is called with an $N$ node tree, so $T(N)$ is the time for `countLabel` to execute with an $N$ node tree.

On this page label lines for the worst case. Be sure to label each line of method `countLabel` with either an $O(\cdot)$ expression or a $T(\cdot)$ expression for the worst case when trees are completely unbalanced, e.g., all nodes in the right subtree. $T(N)$ is the time for `countLabel` to run. You don’t need to label code in the method `count`. You do need to label the call of `count` in `countLabel`.

```
public TreeNode countLabel(TreeNode tree) {
    if (tree == null) return null;

    TreeNode left = countLabel(tree.left);
    TreeNode right = countLabel(tree.right);
    int size = count(tree);
    return new TreeNode(size, left, right);
}

public int count(TreeNode tree) {
    if (tree == null) return 0;
    return 1 + count(tree.left) + count(tree.right);
}
```

Label each line with $O(\cdot)$ or $T(\cdot)$

Be sure to write the recurrence relation and its solution.
Part 3.5 (4 points)
The method `countLabelAux` correctly returns a tree with the same shape. You are to determine the average case complexity of this method. You must develop a recurrence relation for `countLabelAux` — the average case is when trees are roughly balanced.

Label each line below with an expression involving $O(\cdot)$ or $T(\cdot)$ when `countLabel` is called with an $N$ node tree, so $T(N)$ is the time for `countLabelAux` to execute with an $N$ node tree.

This is for the average case, when trees are roughly balanced.

```java
public TreeNode countLabelAux(TreeNode tree) {
    if (tree == null) return null;
    if (tree.left == null && tree.right == null)
        return new TreeNode(1,null,null);
    TreeNode left = countLabelAux(tree.left);
    TreeNode right = countLabelAux(tree.right);
    int lcount = 0;
    int rcount = 0;
    if (left != null) lcount = left.info;
    if (right != null) rcount = right.info;
    return new TreeNode(1 + lcount + rcount,left,right);
}
```

Label each line with $O(\cdot)$ or $T(\cdot)$

Be sure to write the recurrence relation and its solution.
The `ListNode` class at the beginning of this test allows linked-list nodes to hold any types. For example, the code below creates a linked list of three strings as shown at the top of the diagram to the right.

```java
ListNode<String> list =
    new ListNode<>("hello",
        new ListNode<>("big",
            new ListNode<>("crocodile",null)));
```

Write the method `convert`, which is started below. The method creates a new linked-list storing integer values such that the value in the \(i^{th}\) node returned is the length of the string of the \(i^{th}\) node that’s a parameter to `convert`. The list in the diagram shown on the bottom above illustrates this.

**Part A: (6 points)**

Complete the method `convert` below. You must write the method iteratively, that is with a loop.

```java
public ListNode<Integer> convert(ListNode<String> list){
    ListNode<Integer> first = new ListNode<>(list.info.length(),null);
    list = list.next;
    ListNode<Integer> last = first;

    // write code below
    }
```
Part B: (6 points)
Complete the method `convertRec` below. You must use write the method recursively, that is without a loop and without any collections or arrays. The method `convertRec` returns the same list that `convert` returns when the parameters to the methods are the same.

```java
class ListNode {
    int value;
    ListNode next;
}

class Solution {
    public ListNode<Integer> convertRec(ListNode<String> list) {
        if (list == null) return null;
        // write code here
    }
}
```
Consider the binary search tree shown below. You’ll be asked several questions about trees using this tree as an example. Strings are inserted into the tree in natural or lexicographical order.

In answering the questions you can use these words, or any other words you choose:

anteater, badger, bear, cougar, dog, elephant, ferret, fox, giraffe, hippo, jaguar, kangaroo, koala, leopard, llama, meerkat, mole, mouse, mule, newt, orangutan, ostrich, otter, panda, pelican, tiger, walrus, yak, zebra

The inorder traversal of the tree is bobcat, crow, dingo, moose, narwhal, octopus, penguin, salmon.

Part 4.1 (3 points)

What is the post-order traversal of the tree shown?
Part 4.2 (3 points)
If the recursive calls in method `inOrder` are swapped, so that the right subtree of `t` is visited, then `t.info` printed, then the left subtree visited, then what is the order of nodes visited for the tree shown with this modified search?

Part 4.3 (3 points)
In the drawing below, label where these strings are inserted, in the order shown: `bear`, `cougar`, `elephant`, `badger`.

```
Moose
   Dingo
     Bobcat
   Penguin
     Narwhal
       Octopus
     Salmon
   Crow
```

Part 4.4 (3 points)
The height of the tree shown is four since the longest root-to-leaf path has four nodes. List three strings/animals such that if they are inserted into the search tree in the order you list them the height of the resulting tree will be seven.
Part 4.5 (2 points)
If a sorted list of $n$ strings is inserted one-at-a-time into an initially empty search tree that does no balancing after each insertion the complexity of the $n$ insertions will be greater than $O(n)$. What is the complexity of the $n$ insertions from a sorted list and why?

Part 4.6 (2 points)
If a sorted list of $n$ strings is inserted one-at-a-time into a java.util.TreeSet (which internally uses a balanced Red-Black tree) the complexity of the $n$ insertions will be greater than $O(n)$. What is the complexity of the $n$ insertions from a sorted list and why?
PROBLEM 6:  (Making Trees (12 points))

A collection (ArrayList) of \( N \) random 26-character strings is created and then the strings are inserted into an initially empty search tree one-at-a-time so that the tree remains a search tree after each insertion. Then the collection of strings is sorted and the strings are inserted into another initially empty search tree one-at-a-time. Timings for different sizes of the list of random strings are shown below on the left for randomly generated strings and when the list of strings is sorted. A graph is shown on the right that plots the timings for the sorted strings and fits a plot showing that the timings is \( O(N^2) \) for inserting \( N \) strings.

<table>
<thead>
<tr>
<th>size</th>
<th>random</th>
<th>sorted</th>
</tr>
</thead>
<tbody>
<tr>
<td>5000</td>
<td>0.015</td>
<td>0.117</td>
</tr>
<tr>
<td>10000</td>
<td>0.012</td>
<td>0.333</td>
</tr>
<tr>
<td>15000</td>
<td>0.015</td>
<td>1.000</td>
</tr>
<tr>
<td>20000</td>
<td>0.014</td>
<td>2.216</td>
</tr>
<tr>
<td>25000</td>
<td>0.015</td>
<td>4.012</td>
</tr>
<tr>
<td>30000</td>
<td>0.019</td>
<td>6.272</td>
</tr>
<tr>
<td>35000</td>
<td>0.024</td>
<td>10.468</td>
</tr>
<tr>
<td>40000</td>
<td>0.031</td>
<td>16.359</td>
</tr>
<tr>
<td>45000</td>
<td>0.034</td>
<td>21.605</td>
</tr>
<tr>
<td>50000</td>
<td>0.035</td>
<td>26.540</td>
</tr>
</tbody>
</table>

The code to insert all the strings into an initially empty tree is shown below.

```java
public TreeNode collectionToTree(Collection<String> coll) {
    TreeNode tree = null;
    for(String s : coll) {
        tree = bstInsert(tree, s);
    }
    return tree;
}
```

The code for `bstInsert` does no tree balancing. Here is the code, though you do not need to understand it to answer the questions here.

```java
public TreeNode bstInsert(TreeNode root, String value) {
    if (root == null) {
        root = new TreeNode(value, null, null);
        return root;
    }
    int comp = value.compareTo(root.info);
    if (comp <= 0) {
        root.left = bstInsert(root.left, value);
    } else {
        root.right = bstInsert(root.right, value);
    }
    return root;
}
```
Part A (3 points)

Explain in at most three sentences why inserting the strings one-at-a-time from a sorted list to create a search tree is $O(N^2)$ for a list of N-strings. The code to do this is shown below.

```java
ArrayList<String> list = getRandomStrings(size);
Collections.sort(list);
TreeNode tree = collectionToTree(list);
```
Part B (3 points)
If the strings were sorted in reverse alphabetical order would the runtime still be $O(N^2)$ for a list of $N$ strings? Explain/justify your answer.

Part C (3 points)
Explain in at most three sentences why the time for insertion shown under the column random above is $O(N\log(N))$ for a list of $N$ randomly ordered strings. The code to do this is shown below.

```java
ArrayList<String> list = getRandomStrings(size);
Collections.shuffle(list);
TreeNode tree = collectionToTree(list);
```

Part D (3 points)
The method bstInsert that creates the search tree from a list (see code at beginning of problem) uses iteration and a while loop. If this method is replaced by a standard recursive tree insertion algorithm creating the tree from a shuffled list of strings works correctly, but creating the tree from a sorted list of strings generates a stack overflow error in the recursive tree insert method.

In a few sentences explain why the sorted list generates the error and the shuffled list does not. You must explain both for full credit.
The questions in this problem will use the tree below.

```
  iguana
 /    
|      |
|      |
eagle octopus
 /     /     
|     |     |
|     |     |
  bison hedgehog  wallaby
 /     /     /     
|     |     |     |
|     |     |     |
  badger crayfish hyena
```

**Part A (4 points)**

The *inorder* traversal of the tree below is:

badger, bison, crayfish, eagle, hedgehog, hyena, iguana, octopus, wallaby.

What is the post-order traversal?

**Part B (6 points)**

Show where the values *fox*, *koala*, and *zebra* would appear if inserted into the search tree above by adding the nodes in the diagram.
Part C (4 points)

Complete the method below that returns a copy of the tree shown above (in general a copy of any tree). You must add code in two places and you should add no more than six lines.

```java
public TreeNode copy(TreeNode root){
    if (root == null) {
        // add code
    }
    // add code

    return new TreeNode(root.info,
    // add code
    );
}
```
Part D (5 points)

Write method oneChildCount that returns the number of nodes in a tree that have one child (e.g., not zero children and not two children). In the tree at the beginning of this problem the value returned would be two since the nodes labeled *octopus* and *hedgehog* in the original tree each have one child.

```java
public int oneChildCount(TreeNode root) {
    // Your implementation here
}
```