Test 2 : Compsci 201

Owen Astrachan

November 15, 2019

Name: _____

NetID/Login: _____

Community standard acknowledgment (signature) _____

	value	grade
NetID	1 pt.	
Problem 1	17 pts.	
Problem 2	18 pts.	
Problem 3	28 pts.	
Problem 4	10 pts.	
Total:	74 pts.	

This test has 12 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.

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)`	$O(2^n)$

`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;
    }
}
```

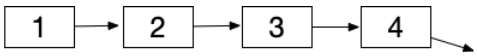
```
public class ListNode {
    int info;
    ListNode next;
    ListNode(int val) {
        info = val;
    }
    ListNode(int val,
             ListNode link){
        info = val;
        next = link;
    }
}
```

PROBLEM 1 : (*hint,lint,mint,pint (17 points)*)

Part A (6 points)

Write method `fromArray` that returns a pointer to the first node of a linked list with the same values and in the same order as the parameter `array`.

For example, the call `fromArray(new int[]{1,2,3,4})` should return a pointer to the first node of this list:



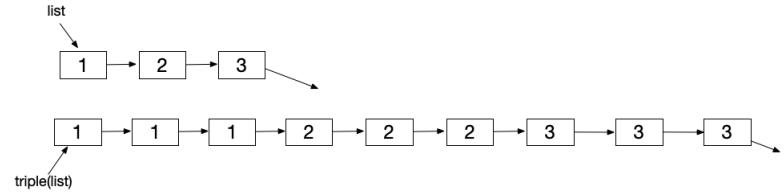
If `array` has no values, the method should return `null`.

```
public ListNode fromArray(int[] array) {









}
```

Part B (3 points)

Describe in words or code how to change the code you wrote in Part A so that it returns a list of only those values in the `array` parameter that have an odd index, e.g. 1,3,5, ... – for example for the call `fromArray(new int[]{1,2,3,4})` the list returned has two values: 2 and 4, since these have indexes 1 and 3, respectively.

Parts C and D refer to the recursive method `triple` that returns a linked-list with three times as many nodes as in `list` by adding two nodes with the same value after each node in the original list. For example the call `triple(list)` for the list on top in the diagram below returns the list diagrammed.




```
131     public ListNode triple(ListNode list) {
132         if (list == null) return null;
133         ListNode after = triple(list.next);
134         list.next = after;
135         return new ListNode(list.info, new ListNode(list.info,list));
136     }
```

Part C (3 points)

Write a recurrence relation and the big-Oh solution to the recurrence relation for the run-time of `triple` on an n-node list

Part D (5 points)

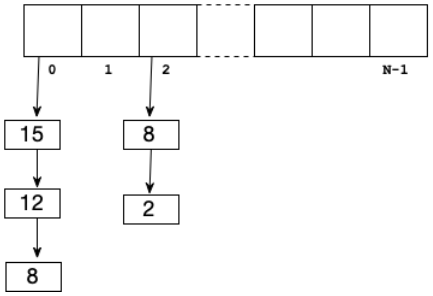
Write the method `multiply` such that `multiply(list,3)` returns the same list as `triple(list)` and, more generally, `multiply(list,n)` returns a list with n occurrences of each node/value in the original list parameter. If `list` is `[1,2]`, then `multiply(list,5)` returns `[1,1,1,1,1,2,2,2,2,2]`

The code you write must be recursive. You will receive no credit unless you write a recursive method.

```
public ListNode multiply(ListNode list, int size) {









}
```

PROBLEM 2 : (***sequence of sequences (18 points)***)

Consider an array of linked-lists, for example `ListNode[] array` where each element of `array` is a pointer to a linked list as diagrammed below, so that `array[0]` points to the first node of the list [15,12,8]; `array[2]` points to the first node of the list [8,2]; and other array values point at other lists not shown.



Part A (4 points)

Complete method `sumArray` that returns the total of all values stored in all nodes in all the the linked lists in `array`.

In writing `sumArray` you **must** call the method `sumList` shown below and use the value it returns — for example the code you write should not refer to any .next fields.

```
public int sumArray(ListNode[] array) {
```

```
}
public int sumList(ListNode list) {
   if (list == null) return 0;
   return list.info + sumList(list.next);
}
```

Part B (2 points)

The names of both `sumArray` and `sumList` can be changed to `sum` and a class containing these methods would still compile and run. Circle the correct statement.

True, both methods can have the same names since the parameters are different.

False, the methods cannot each be named `sum` in the same class.

Part C (6 points)

Write method `find` that returns a pointer to the node of a linked list containing `target`, or `null` if no node in the linked list contains `target`. For example, the call `find(array[0],8)` returns a pointer to the node containing 8 given the array above and the call `find(array[2],5)` returns `null`.

```
public ListNode find(ListNode list, int target) {









}
```

Part D (6 points)

Write method `find` that returns the index of the linked list in `array` that contains `target`, or -1 if there is no linked list in `array` that contains `target`. You must call the method `find` in Part C, assume it works correctly. If more than one linked list in `array` contains `target`, return the index that is smallest of those indexes that point to linked lists containing `target`.

For example, `find(array,2)` returns 2, `find(array,12)` returns 0, and `find(array,8)` returns 0, since 0 is less than 2.

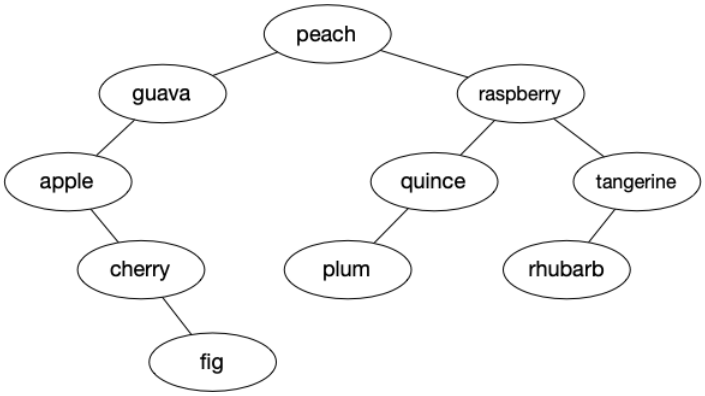
```
public int find(ListNode[] array, int target) {









}
```

PROBLEM 3 : (*Figleaf, Bigleaf (28 points)*)

Part A (3 points)

The tree shown below is a search tree. The string `"rhubarb"` must have been added to the tree after the string `"tangerine"` since it is a child of `"tangerine"`.

Of the 10 strings in the tree, which could have been inserted as the last of the 10? There is more than one string that could have been inserted last, list each that could be inserted last.



Part B (8 points)

The height of this tree is five: the longest root-to-leaf path is from peach to fig which contains five nodes.

Which of the following nine fruits can be added to the tree so that it is still a search tree, and so that the height after the addition is still five. Consider fruits in isolation: only one value is added, then the height is checked and the value removed. For example, avocado can be added and the height of the tree will still be five.

Circle the fruits whose singular addition will not increase the height.

```
banana          date          durian          kiwi          lemon

mango           pear          pomegranate     watermelon
```

Part C (5 points)

Assume `TreeNodes` store String values.

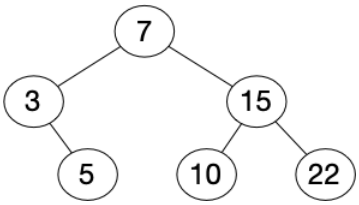
The method `revAlpha` below should return an `ArrayList` whose elements are in reverse alphabetical order (largest to smallest) when its parameter is a search tree. Complete `alphaHelp` so that `revAlpha` works as specified. You should write at most 15 lines of code. Hint: an in-order traversal visits the nodes of a search tree in alphabetical order. Your solution must run in $O(n)$ time for an n-node tree.

You may not add to or alter the code in `revAlpha`.

```
public ArrayList<String> revAlpha(TreeNode tree) {
    ArrayList<String> list = new ArrayList<>();
    alphaHelp(tree,list);
    return list;
}

private void alphaHelp(TreeNode tree, List<String> list) {









}
```

Part D (6 points)

The method `sum` below returns the sum of all the nodes in a binary tree. For the tree diagrammed, the call `sum(root)` returns 62.




```
public int sum(TreeNode root) {
    if (root == null) return 0;
    return root.info + sum(root.left) + sum(root.right);
}
```

Let $T(n)$ be the time for `sum` to run on an n-node tree. Write the recurrence relation and the big-Oh solution to the recurrence relation for both the average case, when trees are roughly balanced and the worst case, when all nodes are in a left subtree (or, equivalently, a right subtree), e.g., nodes have only left children.

Be sure to write the recurrence and the big-Oh solution to that recurrence.

average: $T(n) =$

solution:

worst: $T(n) =$

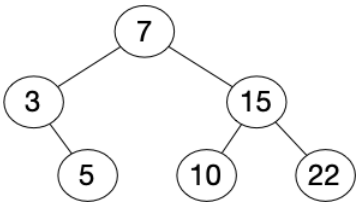
solution

Part E (6 points)

The root is at level zero, its children at level one, and in general the children of a node at level N are at level N+1. Complete the helper method `levelHelp` below so that the call `evenLevelSum(root)` returns the sum of all nodes at even levels in the tree parameter. For the tree below, the call should return 44 since the nodes at even levels are 7 (level zero), 5, 10, and 22 (level two).

The call `levelHelp(root,true)` below indicates that the global root, at level zero, is at an even level since the second argument in the call is `true`.

Do not change or add code to `evenLevelSum`.




```
public int evenLevelSum(TreeNode root) {
    return levelHelp(root,true);
}

/**
  * return sum of all nodes at even levels in tree
  * where root is at even level if isEvenLevel == true
  * and root is NOT at an even level if isEvenLevel == false
  */
public int levelHelp(TreeNode root, boolean isEvenLevel) {









}
```

PROBLEM 4 : (*Missing Link (10 points)*)

The code below refers to classes and methods in the DNA/Link assignment. In the example shown the code prints 27 for both numbers since three occurrences of `"acg"` are each replaced by a string of nine characters.

```
11      public void testDNA() {
12          IDnaStrand lstrand = new LinkStrand( s: "acgacgacg");
13          String enzyme = "acg";
14          String splicee = "aaaaaaaaa";
15          IDnaStrand ldna = lstrand.cutAndSplice(enzyme,splicee);
16          System.out.printf("strand %d\n",ldna.size());
17          System.out.printf("string %d\n",ldna.toString().length());
18      }
```

Part A (3 points)

If `new LinkStrand` is replaced by `new StringStrand` or by `new StringBuilderStrand` the code above still runs and the output is the same (with no other changes to the code). In a sentence or two explain why the code still runs with the same output.

Part B (3 points)

If the String `splicee` is replaced by a longer (more characters) string, e.g., a string of one-million a's, the output will change since the new strand created by `cutAndSplice` will be longer. However, the runtime does not change when the length of `splicee` changes from 9 to one million. Briefly, explain why the runtime does not change and does not depend on the length of `splicee`.

(continued)

Part C (4 points)

The code above will run if `splicee` has one billion a's, but will crash in the call to `ldna.toString().length()` with an out-of-memory error. The value of `ldna.size()` is correctly printed as three-billion, then the program crashes.

Why does the call `ldna.toString()` crash with an out of memory error but the call `ldna.size()` does not?