# Test 2 Redux: Compsci 201

Owen Astrachan

April 19, 2019

Name: ______________________________________________

NetID/Login: _____________

Community standard acknowledgment (signature) ________________________________________

| | value | grade |
|---|---|---|
| Problem 1 | 4 pts. | |
| Problem 2 | 8 pts. | |
| Problem 3 | 6 pts. | |
| TOTAL: | 18 pts. | |

This test has 6 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 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 where it is asked for.

You may consult your note sheet 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 {
    int info;
    TreeNode left;
    TreeNode right;

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

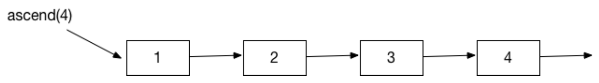
```
public class ListNode {
    int info;
    ListNode next;
    ListNode(int val) {
        info = val;
    }
    ListNode(int val,
             ListNode link){
         info = val;
         next = link;
    }
}
```

**PROBLEM 1 :** (***ListUp (4 points)***)

Write method `ascend` that returns a pointer a linked-list containing the values 1,2, ..., `size`, where `size` is the parameter to `ascend`.

The diagram below shows the result of the call `ascend(4)` — assume that `size` will always be greater than or equal to 1.

You can write `ascend` iteratively or recursively.



```
public ListNode ascend(int size) {





}
```

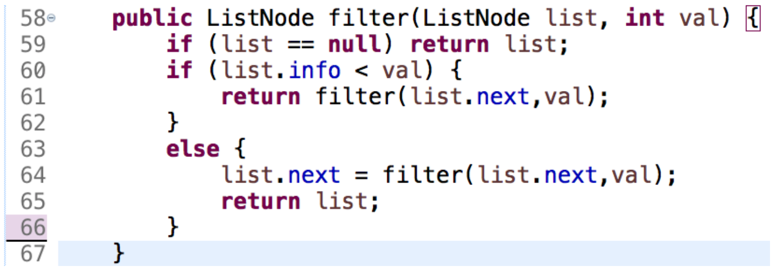
**PROBLEM 2 :** (***Filters (8 points)***)

The method `filter` effectively removes all nodes whose `info` field is less than the value of parameter `val`. An alternate view is that `filter` returns a list in which nodes containing `info` fields that are greater than or equal to `val` are retained.

| list | call | returned list |
|---|---|---|
| `[13,7,12,10]` | `filter(list,11)` | `[13,12]` |
| `[13,7,12,10]` | `filter(list,15)` | `[]` |
| `[13,7,12,10]` | `filter(list,6)` | `[13,7,12,10]` |

**Part A (2 points)**

The recursive method shown below is correct. Write a recurrence relation for this method where $T(n)$ is the time for filter to execute on an $n$-node list. Write the solution to the recurrence relation.



```java
    public ListNode filter(ListNode list, int val) {
        if (list == null) return list;
        if (list.info < val) {
            return filter(list.next,val);
        }
        else {
            list.next = filter(list.next,val);
            return list;
        }
    }
```

**Part B (6 points)**

Write an iterative solution. The values in the list returned should be in the same relative order that they are in parameter `list`. You can create only one new node, otherwise use the nodes that exist in parameter `list`.

***You can write the code in any way as long as you write it iteratively and create at most one new node.***

You may find the following ideas useful in writing a solution.

- Create a dummy header node (see below) that acts like the first node of the list that will be returned, the actual list returned is `dummy.next`.
- Keep `last` pointing to the last node of the list that will be returned. When traversing the parameter list, add retained nodes/values to the end of this list, using and updating `last`.

If you write the code to use this idea, you'll use these statements before your loop:

```java
ListNode dummy = new ListNode(Integer.MAX_VALUE,null);
ListNode last = dummy;
```

You'll then have a while loop. After the while loop you'll include this code:

```java
last.next = null;
return dummy.next;
```

(continued on next page)

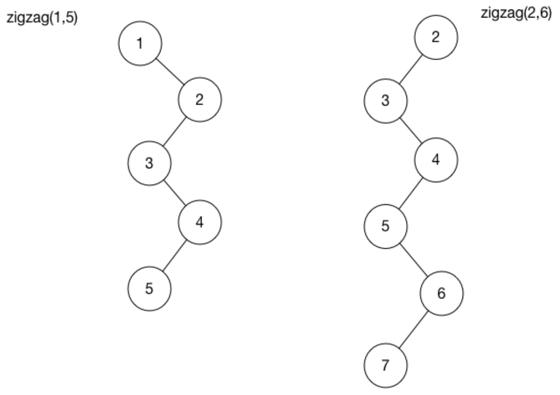
```
public ListNode filter(ListNode list,int val) {









}
```

**PROBLEM 3 :** (*Zzzzzzzzzz (6 points)*)

Write method `zigzag` that returns a binary tree in which each internal node has only one non-null child. If the value in a node is odd, the only child is a right-child. If the value in a node is even, the only child is a left-child.

The diagram below shows the result of two calls. The parameter `numNodes` is the number of nodes in the returned tree. The parameter `value` is the value in the root of the tree returned.

Except for the root, which has no parent, every node's value should be one more than its parent. Assume that `numNodes` will always be greater than or equal to one.



```
/**
 * @param value is stored in the root of the tree returned
 * @param numNodes is the number of nodes in the treet returned
 */
public TreeNode zigzag(int value, int numNodes) {

    TreeNode root = new TreeNode(value);
    if (numNodes == 1) return root;















}
```