Test 2 Practice: Compsci 201

Owen Astrachan

November 10, 2016

Name: ___________________________________________

NetID/Login: __________

Honor code acknowledgment (signature) __________________________________________

<table>
<thead>
<tr>
<th>Problem</th>
<th>Value</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem 1</td>
<td>?? pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 2</td>
<td>?? pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 3</td>
<td>?? pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 4</td>
<td>?? pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 5</td>
<td>?? pts.</td>
<td></td>
</tr>
<tr>
<td>TOTAL:</td>
<td>75 pts.</td>
<td></td>
</tr>
</tbody>
</table>

This test has ?? pages, be sure your test has them all. Do NOT spend too much time on one question — remember that this class lasts 75 minutes.

That means you should spend no more than 1 minute per point. Put your NetID clearly on each page of this test (worth 1 extra 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.

There are two blank pages at the end of the test. Make a note on the appropriate problem if you use the extra sheet.
PROBLEM 1:  *(It Depends (8 points))*

**Part A (3 points)**
What value is returned by the call `calculate(2043)`? What is the 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).
PROBLEM 2:  (Reversal of Fortune (12 points))

Write the method reverse whose header is given below. The method reverse reverses the elements of the parameter queue q. For example, if q is represented by (a,b,c,d), with a the first element and d the last element of the queue, then after the call reverse(q) q is represented by (d,c,b,a).

Part A (4 points)
Write the method below that changes parameter q so that it is reversed. You may only define variables of type Stack, Queue, String, or int (no arrays, no ArrayList, etc.). Note that all Stack and Queue objects have a method size() that returns the number of elements in the object.

```java
Queue<String> q = new LinkedList<String>();
q.add("ant");
q.add("bat");
q.add("cat");
q.add("dog");
// contents of q are ("ant", "bat", "cat", "dog")

reverse(q);
// contents of q are ("dog", "cat", "bat", "ant");

public void reverse(Queue<String> q) {
```
**Part B (4 points)**

The code below is designed to reverse a queue as described earlier. When run, the code below generates an error message via an exception: *No Such Element Exception*.

Fix the code below so that it reverses a queue by adding one if statement in the code. You cannot add code other than an if statement whose guard/test uses parameter q somehow. You can use open/close braces in writing code.

```java
public static void reverse(Queue<String> q){

    String elt = q.remove();

    reverse(q);

    q.add(elt);

}
```

**Part C (4 points)**

Why does the *LinkedList* class implement the *Queue* interface, but the *ArrayList* class does not. Explain this in terms of performance using big-Oh to justify your answer.
PROBLEM 3: (Trees (32 points))

The questions in this problem will use the tree below.

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

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).

```java
public TreeNode copy(TreeNode root){
    if (root == null) // add code
        return new TreeNode(root.info,
            copy(root.left), // add code
            copy(root.right)); // add code

    return new TreeNode(root.info,
        copy(root.left), // add code
        copy(root.right)); // add code
}
```
Part D (5 points)
Write method $\text{oneChildCount}$ 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 $\text{octopus}$ and $\text{hedgehog}$ in the original tree each have one child.

public int oneChildCount(TreeNode root) {
    
    
}

Part E (5 points)
Implementing a priority queue using a balanced search tree results in $\text{add}$, $\text{peek}$, and $\text{remove}$ operations that are all $O(\log N)$ for a $N$-element priority queue. However, heaps are typically used to implement priority queues because they offer better performance. Explain why heaps are better than balanced binary trees for implementing priority queues (be brief). NOT yet covered in fall 16
Part F (8 points)

The two methods print and printItems below generate the following output:

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

when invoked with the call print(tree, new LinkedList<String>()); where tree is the root of the tree on the previous page.

```java
public static void printItems(LinkedList<String> list){
    Queue<String> copy = (Queue<String>) list.clone();
    while (copy.size() != 0){
        System.out.print(copy.remove()+" ");
    }
    System.out.println();
}
```

```java
public static void print(TreeNode root, LinkedList<String> q){
    if (root == null) return;
    if (root.left == null && root.right == null){
        q.add(root.info);
        printItems(q);
        q.removeLast();
        return;
    }
    q.add(root.info);
    print(root.left,q);
    print(root.right,q);
    q.removeLast();
}
```

If LinkedList is replaced everywhere with Stack – and add replaced by push; removeLast by pop and so forth in both methods above then what will be printed by the call print(tree, new Stack<String>()); – you should show four lines of output in your answer.

You should also briefly explain why the queue is cloned in the method printItems.
PROBLEM 4: \( (\text{store e-sort} \ (20 \text{ points})) \)

**Part A (5 points)**

Put the letter from the list of speakers on the left next to the phrase that person did say or could have said based on what we’ve studied in class. You can use a letter more than once, you don’t need to use all the letters.

A. Barack Obama
   1. ______ Bubblesort’s not the way to go

B. Nancy Leveson
   2. ______ I invented quicksort

C. Tony Hoare
   3. ______ I heart Python, Sorting, and more.

D. Fred Brooks
   4. ______ I was a pioneer in Software Safety

E. Tim Peters
   5. ______ I was Duke’s valedictorian, and started Computer Science at UNC and optimal

F. Owen Astrachan

**Part B (5 points)**

*not included*
The code below sorts an array of Strings representing DNA by creating arrays for every possible 4-character prefix, e.g., "aaaa" "aaag" "aaat" "agtc" "gatt" "tttt"; sorting each of these arrays, and then combining these sorted arrays together.

```java
public static void dnasort(String[] dna){
    Map<String,ArrayList<String>> prefixMap = new TreeMap<String,ArrayList<String>>();
    for(String s : dna){
        String prefix = s.substring(0, 4);
        ArrayList<String> list = prefixMap.get(prefix);
        if (list == null){
            list = new ArrayList<String>();
            prefixMap.put(prefix, list);
        }
        list.add(s);
    }
    for(ArrayList<String> list : prefixMap.values()){Collections.sort(list);}
    ArrayList<String> combined = new ArrayList<String>();
    for(ArrayList<String> list : prefixMap.values()){combined.addAll(list);}
    System.arraycopy(combined.toArray(new String[0]), 0, dna, 0, dna.length);
}
```

The code above is faster than calling `Arrays.sort(dna)` for an array of one-million strings representing dna strands (e.g., all the strings contain just the characters 'a', 'g', 't', 'c'). The code is slower when sorting an array of one-thousand strands. Explain why.

In answering the question you can make some assumptions about the data in the array being sorted. Each of the 256 different four-character prefixes is equally likely, so the number of strings in `String[] dna` beginning with "aagt" is the same as the number beginning with "gacc" and so on for all 256 different prefixes.

For reasoning about the problem assume the number of Strings in the array is a power of 2, say $2^{16}$ or $2^{20}$.
PROBLEM 5: *(SortLink)*

We use the lists diagrammed below in understanding the code and concerns in this problem.

The method $\text{merge}(\text{list1, list2})$ accepts two sorted linked lists as parameters and returns the result of altering links to merge the lists into a sorted list as shown in the diagram. No new nodes are created, but links are re-arranged and a pointer to the first node of the merge list returned. Complete method $\text{merge}$ below (next page) by adding code in the while loop.

The code *before* the loop sets $\text{front}$ to the smallest of the first nodes in list and list2 – this is the first node in the list returned.

The code after the loop takes care of any unprocessed nodes, but after the loop one (or both) of list1 and list2 are empty.
public static Node merge(Node list1, Node list2) {
    // make front point to first node in final list
    Node front = null;
    if (list1.info > list2.info) {
        front = list2;
        list2 = list2.next;
        front.next = null;  // unlink from list
    } else {
        front = list1;
        list1 = front.next;
        front.next = null;  // unlink from list
    }
    // add to end of front/last list
    Node last = front;
    while (list1 != null && list2 != null) {
        if (list1 == null) {
            last.next = list2;
        } else {
            last.next = list1;
        }
    }
    return front;
}