Midterm 3: Compsci 201
Form A

Prof. Astrachan

November 25, 2024

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 A, please mark your answer sheet 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 A, please mark your answer sheet accordingly

PROBLEM 1:

The call `stuff()` for the method shown below on lines 12-17 results in printing `[orange, lemon, lime, watermelon]`.

```
 5  public int lcount(String s){
 6      int tot = 0;
 7      for(char ch : s.toCharArray()){
 8          if (ch == 'l' || ch == 'L') tot += 1;
 9      }
10      return tot;
11  }
12  public void stuff(){
13      String[] a = {"lime", "lemon", "watermelon", "orange"};
14      Arrays.sort(a);
15      Arrays.sort(a, (x,y)->lcount(x)-lcount(y));
16      System.out.println(Arrays.toString(a));
17  }
```

If line 14 is commented out, so that it does not execute, the output changes. What is the new output?

A. `[orange, lime, lemon, watermelon]`

B. `[orange, watermelon, lime, lemon]`

C. `[watermelon, orange, lime, lemon]`

D. `[watermelon, lime, lemon, orange]`

PROBLEM 2:

Consider the Java code below. What is the value of array `a` after the call to `Arrays.sort` shown. (Note, this code is **not** part of the code above.)

```
jshell> String[] a = {"lemon", "lime", "watermelon", "orange", "apple"}
a ==> String[5] { "lemon", "lime", "watermelon", "orange", "apple" }

jshell> Arrays.sort(a, Comparator.comparing(String::length))
```

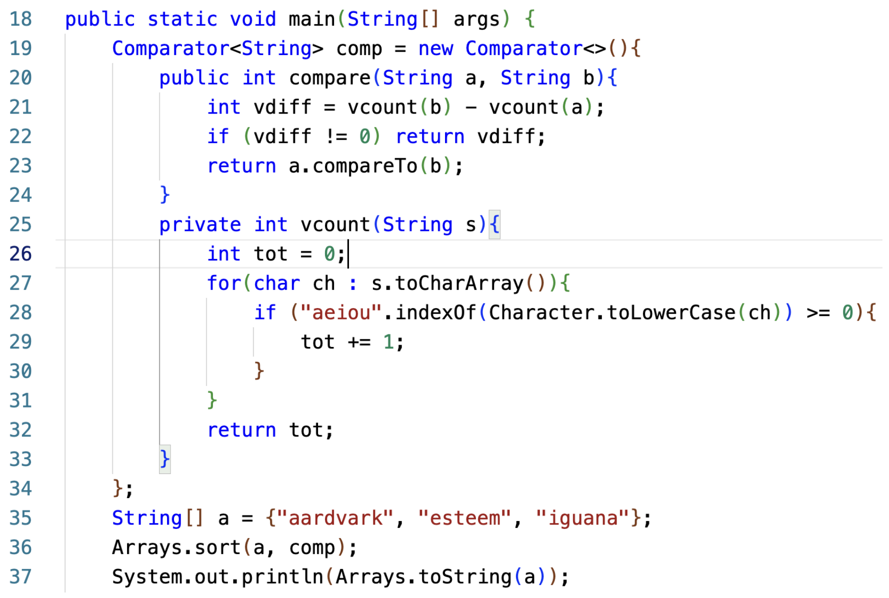
A. `{"lime", "lemon", "apple", "orange", "watermelon"}`

B. `{"lime", "apple", "lemon", "orange", "watermelon"}`

C. `{"watermelon", "orange", "apple", "lemon", "lime"}`

PROBLEM 3:

Consider the code shown below on lines 35-37 whose execution results in printing three strings. What is printed when this code is executed? The code uses a `Comparator` object defined on lines 19-34.



```
18  public static void main(String[] args) {
19      Comparator<String> comp = new Comparator<>(){
20          public int compare(String a, String b){
21              int vdiff = vcount(b) - vcount(a);
22              if (vdiff != 0) return vdiff;
23              return a.compareTo(b);
24          }
25          private int vcount(String s){
26              int tot = 0;
27              for(char ch : s.toCharArray()){
28                  if ("aeiou".indexOf(Character.toLowerCase(ch)) >= 0){
29                      tot += 1;
30                  }
31              }
32              return tot;
33          }
34      };
35      String[] a = {"aardvark", "esteem", "iguana"};
36      Arrays.sort(a, comp);
37      System.out.println(Arrays.toString(a));
```

A. `"aardvark esteem iguana"`

B. `"esteem aardvark iguana"`

C. `"iguana aardvark esteem"`

D. `"iguana esteem aardvark"`

PROBLEM 4:

If line 21 is changed to `int vdiff = vcount(a) - vcount(b)` will the output change?

A. Yes, the output will change

B. No, the output will not change

PROBLEM 5:

What sorting algorithm did Prof. Astrachan write a paper about, the same algorithm that President Obama thought would "be the wrong way to go in sorting a million 32-bit integers"?

A. Selection Sort

B. Tim Sort

C. Bubble Sort

D. Merge Sort

E. Quick Sort

The strings A, B, and C are pushed, in that order, onto an initially empty stack *s*. Three pop operations are interleaved with the three push operations resulting in letters being popped in some order. Every pop operation will be valid, i.e., there will be at least one string on the stack when the pop operation executes.

For example the sequence of operations below results in the letters B A C being popped, in that order (B is popped first). This sequence can be described as `push, push, pop, pop, push, pop` Any interleaving of three `push` and three `pop` operations yields some ordering of the letters A, B, and C as a result of the pop operations. The first push is always `s.push("A")`, the second push is `s.push("B")`, and the third is `s.push("C")`.

```
  Stack<String> s = new Stack<>();
  s.push("A");
  s.push("B");
  s.pop();
  s.pop();
  s.push("C");
  s.pop();
```

PROBLEM 6:

What is the sequence that yields `A C B`?

A. push push push pop pop pop

B. push pop push pop push pop

C. push pop push push pop pop

PROBLEM 7:

Of the six distinct orderings of the letters A B C, one ordering CANNOT result using push and pop operations as described above. The code example above illustrates that B A C can result; the previous question indicates that A C B can result. Which of the following orderings CANNOT result?

A. A B C

B. B C A

C. C A B

D. C B A

PROBLEM 8:

Suppose that the Strings A, B, and C are added, in that order, onto an initially empty Queue. Three remove operations are interleaved with the add operations resulting in the Strings being removed in some order. In the previous questions, it was stated that five of the six orderings could be achieved by interleaving stack push and pop operations. How many different orderings can be achieved by interleaving add and remove operations with a queue?

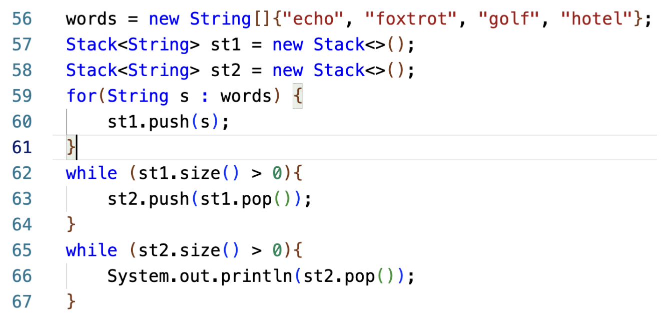
A. one ordering: A B C

B. between 2 and 5 orderings

C. all orderings

PROBLEM 9:

What is printed (one item per line) when the code below executes?



```java
words = new String[]{"echo", "foxtrot", "golf", "hotel"};
Stack<String> st1 = new Stack<>();
Stack<String> st2 = new Stack<>();
for(String s : words) {
    st1.push(s);
}
while (st1.size() > 0){
    st2.push(st1.pop());
}
while (st2.size() > 0){
    System.out.println(st2.pop());
}
```

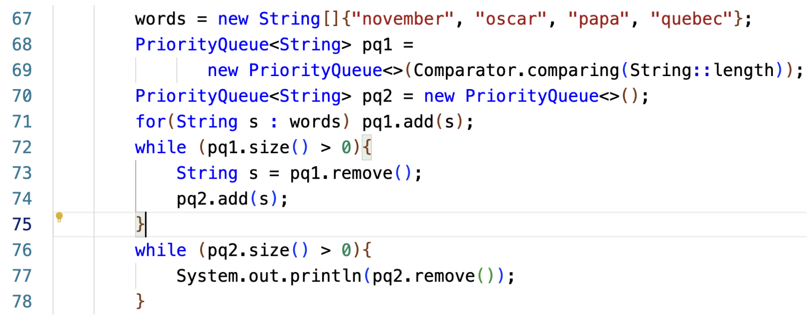
A. `echo foxtrot golf hotel`

B. `hotel golf foxtrot echo`

C. `Exception: java.util.EmptyStackException`

PROBLEM 10:

When the code below executes, the output is (one string per line) `november oscar papa quebec`



```java
67        words = new String[]{"november", "oscar", "papa", "quebec"};
68        PriorityQueue<String> pq1 =
69                new PriorityQueue<>(Comparator.comparing(String::length));
70        PriorityQueue<String> pq2 = new PriorityQueue<>();
71        for(String s : words) pq1.add(s);
72        while (pq1.size() > 0){
73            String s = pq1.remove();
74            pq2.add(s);
75        }
76        while (pq2.size() > 0){
77            System.out.println(pq2.remove());
78        }
```

In what order are the strings added to `pq2` on line 74?

A. `papa oscar quebec november`

B. `papa quebec oscar november`

C. `november oscar papa quebec`

D. `november quebec oscar papa`

PROBLEM 11:

If the definition of `pq1` is changed to the following, will the output change?

```java
PriorityQueue<String> pq1 =
    new PriorityQueue<>(Comparator.comparing(String::length).reversed());
```

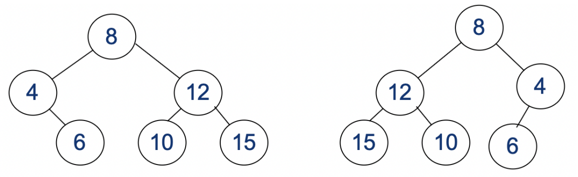
A. Yes, the output will be different.

B. No, the output will be the same.

The method `copy` returns an exact copy of the tree passed as parameter to `copy` and does not alter the tree referenced by the parameter.

```java
public TreeNode copy(TreeNode t){
    if (t == null) return null;
    return new TreeNode(t.info,
                        copy(t.left),
                        copy(t.right));
}
```

Consider modifying `copy` to create a *mirror copy* in which every node that's a left-child becomes a right-child of the same parent node, and vice versa. Except for the root, every node is either a left-child or a right-child. An example of a mirror copy is shown below: the tree on the right is a *mirror copy* of the tree on the left, and vice vera, the tree on the left is a mirror copy of the tree on the right. The code for creating a *mirror tree* is shown, it is missing two statements



```java
3     public TreeNode mirrorTree(TreeNode t){
4         if (t == null) return null;
5         return new TreeNode(t.info,
6                             // recursive call
7                             // recursive call
8                             );
```

You'll complete method `mirrorTree` so that it returns a *mirror copy* of its tree parameter as shown and described above

PROBLEM 12:

What is the recursive call on line 6 that will make `mirrorTree` work as intended. Write the answer in the fill-in-the-blank section on the back of the answer sheet. **Bubble A for this question on the front of the answer sheet.**

PROBLEM 13:

When completed, the big-Oh complexity of `mirrorTree` and `copy` is the same when the tree parameter is **roughly balanced**. What is that complexity?

A. $O(\log N)$

B. $O(N)$

C. $O(N \log N)$

D. $O(N^2)$

PROBLEM 14:

When completed, the big-Oh complexity of `mirrorTree` and `copy` is the same when the tree parameter is **completely unbalanced**. What is that complexigty?

A. $O(\log N)$

B. $O(N)$

C. $O(N \log N)$

D. $O(N^2)$

The code below is from a working (passes all tests) version of `topMatches` for `HashListAutocomplete` in the *P4: Autocomplete* assignment. The instance variable `myMap` is initialized to reference a `HashMap` as specified in the assignment.

```
18  @Override
19  public List<Term> topMatches(String prefix, int k) {
20      if (k < 0) {
21          throw new IllegalArgumentException("Illegal value of k:"+k);
22      }
23
24      if (prefix.length() > MAX_PREFIX) {
25          prefix = prefix.substring(0, MAX_PREFIX);
26      }
27      List<Term> all = myMap.get(prefix);
28      if (all == null){
29          return new ArrayList<>();
30      }
31      List<Term> list = all.subList(0, Math.min(k, all.size()));
32      return list;
33  }
```

PROBLEM 15:

If there are N total terms read and processed by the `initialize` method and `M` terms that match the prefix, what is the complexity of executing line 27 (do not count the time for `initialize` to execute)?

A. $O(1)$

B. $O(M)$

C. $O(N)$

D. $O(M + N)$

PROBLEM 16:

The code on lines 31 and 32 returns the top/best/heaviest k terms. What is the complexity of executing these two lines where N and M are as in the previous problem and there are k terms matching the prefix.

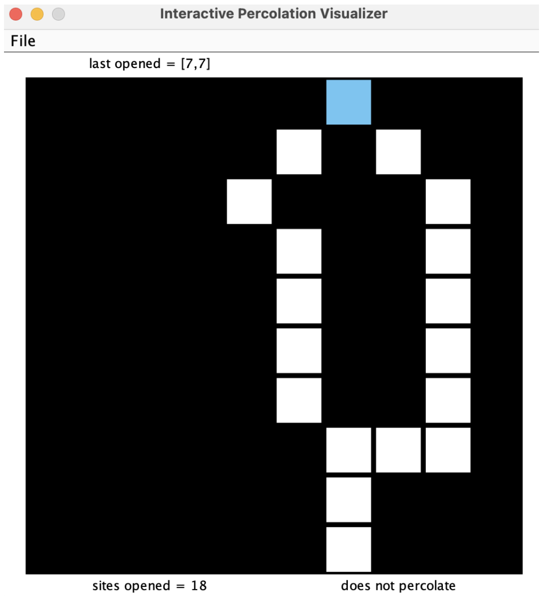
A. $O(1)$

B. $O(k)$

C. $O(M)$

D. $O(M + N)$

The image below shows 18 cells/sites open in a 10x10 grid using the *Interactive Percolation Visualizer* from P5.



PROBLEM 17:

What is the minimum number of additional cells/sites the user must open in the system visualized above before the system percolates?

A. 1

B. 2

C. 3

D. none of the above

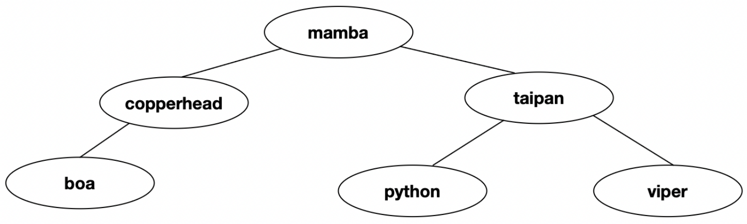
PROBLEM 18:

True or False: it is possible for a user to open $O(N^2)$ cells/sites in an $N \times N$ grid without having the system percolate using the *Interactive Percolation Visualizer*.

A. True, it is possible

B. False, if $O(N^2)$ sites are open, the system will percolate

The next three problems use the tree below which is a search tree.



PROBLEM 19:

Consider these two traversals of the tree shown

1. *boa, copperhead, python, viper, taipan, mamba*
2. *mamba, copperhead, boa, taipan, python, viper*

Which one statement below is true about these traversals?

A. 1 is postorder, 2 is preorder

B. 1 is postorder, and 2 is **not** preorder

C. 1 is **not** postorder, and 2 is preorder

D. 1 is **not** postorder, and 2 is **not** preorder

PROBLEM 20:

Which *one* word **cannot** be inserted as a right child of `copperhead` so the tree remains a search tree after the insertion?

A. coral

B. diamondback

C. krait

D. sidewinder

PROBLEM 21:

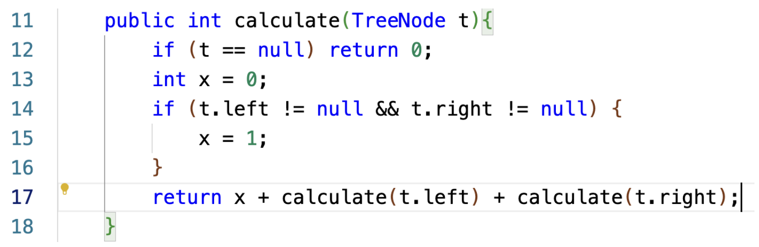
What does the call `calculate(t)` return when `t` references the root node (mamba) of this tree?

A. 0

B. 1

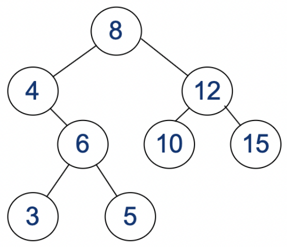
C. 2

D. 3



```java
public int calculate(TreeNode t){
    if (t == null) return 0;
    int x = 0;
    if (t.left != null && t.right != null) {
        x = 1;
    }
    return x + calculate(t.left) + calculate(t.right);
}
```

Consider the tree below. The non-leaf nodes are roots of subtrees with the sums: 14 (root 6), 18 (root 4), 37 (root 12), and 63 (root 8). These sums are calculated using the method `treeSum` on the right.



```java
public int treeSum(TreeNode t){
    if (t == null) return 0;
    return t.info +
            treeSum(t.left) +
            treeSum(t.right);
}
```

The code below computes and prints all sums by calling a method `allSums`. For the tree above this code will print `[63, 18, 14, 37]`.

```java
ArrayList<Integer> list = new ArrayList<>();
allSums(t,list);
System.out.println(list);
```

You'll complete method `allSums`, you may call `treeSum` shown above.

```java
12  public void allSums(TreeNode t, ArrayList<Integer> sumList){
13      if (t == null) return;
14      if (t.left == null && t.right == null) return;
15
16      // add a value to sumList
17      // make a recursive call
18      // make a recursive call
19  }
```

PROBLEM 22:

What is the code on 16? Write the answer in the fill-in-the-blank section on the back of the answer sheet. **Bubble A for this question on the front of the answer sheet.**

PROBLEM 23:

What is the code on 17 (it should reference `t.left`) Write the answer in the fill-in-the-blank section on the back of the answer sheet. **Bubble A for this question on the front of the answer sheet.**

PROBLEM 24:

Will the code on line 18 be different than the code on line 17 other than using `t.right` rather than `t.left`?

A. It will be different, more than just replacing `.left` with `.right`

B. It will be the same: replace `.left` with `.right`.

PROBLEM 25:

What is the complexity of calling `allSums` as implemented above with a tree of N nodes that is roughly balanced?

A. $O(N)$

B. $O(N \log N)$

C. $O(N^2)$