Midterm 3: CompSci 201

Owen Astrachan

November 29, 2023

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 33 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 B, 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 B, please mark your answer sheet accordingly

PROBLEM 1:

What is printed after the three lines below execute?

```
String [] c = {"alpha", "bee", "ant", "acorn"};
Arrays.sort(c, Comparator.reverseOrder());
System.out.println(Arrays.toString(c));
```

A. `{"bee", "ant", "alpha", "acorn"}`

B. `{"bee", "ant", "acorn", "alpha"}`

C. `{"ant", "bee", "alpha", "acorn"}`

D. `{"acorn", "alpha", "ant", "bee"}`

PROBLEM 2:

```
String [] c = {"alpha", "bee", "ant", "acorn"};
Arrays.sort(c, Comparator.comparing(String::length));
System.out.println(Arrays.toString(c));
```

A. `["ant", "bee", "acorn", "alpha"]`

B. `["ant", "bee", "alpha", "acorn"]`

C. `["bee", "ant", "alpha", "acorn"]`

D. `["bee", "ant", "acorn", "alpha"]`

PROBLEM 3:

Every binary tree printed using an `inOrder` traversal (code at beginning of test) results in printing thev alues in alphabetical order.

A. False, only binary *search* trees are printed in alphabetical order.

B. True, every binary tree is printed in alphabetical order.

The next two questions refer to the method `foo` below.

```
 public void foo(int k) {
     if (k > 0) {
         subfunc(k);
         foo(k/2);
     }
}
```

PROBLEM 4:

Assume that the call `subfunc(k)` executes in $O(1)$ time. Which of the following best characterizes the running time of the call `foo(n)`?

A. $O(1)$

B. $O(\log n)$

C. $O(n)$

D. $O(n \log n)$

E. $O(n^2)$

PROBLEM 5:

Assume that the call `subfunc(k)` executes in $O(k)$ time. Which of the following best characterizes the running time of the call `foo(n)`?

A. $O(1)$

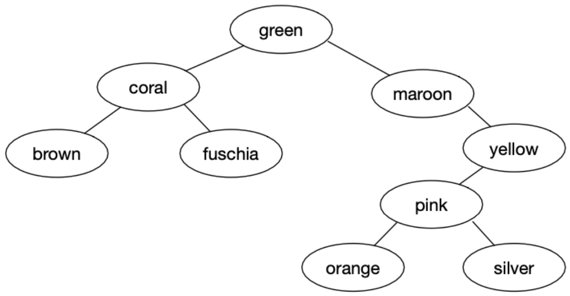
B. $O(\log n)$

C. $O(n)$

D. $O(n \log n)$

E. $O(n^2)$

In the next three problems you're asked to reason about the *binary search tree* shown below in which nodes store `String` values.



An inorder traversal of this tree prints the values in alphabetical order since the tree is a search tree.

PROBLEM 6:

What are the first five strings printed by the code below by the call `preOrder(root)` where `root` points to *green* in the diagram. Code for `preOrder` can be found at the beginning of the test.

A. `green coral brown fuschia maroon`

B. `silver organe pink yellow maroon`

C. `fuschia brown coral silver orange`

D. `green coral fuschia brown maroon`

PROBLEM 7:

If the string `"red"` is added to the tree so that the tree remains a search tree, what node will be the parent of the new node added?

A. orange

B. silver

C. yellow

D. maroon

PROBLEM 8:

If the string `"coral"` is replaced by `"aqua"`, does the tree remain a search tree?

A. yes, it is still a search tree

B. no, it is not a search tree

PROBLEM 9:

The method `count` should return the number of nodes in its tree parameter. For example, for the tree above, the number of nodes is nine. What expression should be returned on line 69 to make the method work correctly?

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.**

```
67    public int count(TreeNode t){
68        if (t == null) return 0;
69        return EXPRESSION;
70    }
```

PROBLEM 10:

To print the values of a tree in *level order*, e.g., for the tree above:

```
green coral maroon brown fuschia yellow pink orange silver
```

what data structure is typically used?

A. Stack

B. Queue

C. Map

D. ArrayList

The next four problems are based on concepts from the *AutoComplete* project. In that project the class `BruteAutocomplete` included this code for the method `topMatches`

```
33     @Override
34     public List<Term> topMatches(String prefix, int k) {
35         if (k < 0) {⋯
38 
39         // maintain pq of size k
40         PriorityQueue<Term> pq =
41                 new PriorityQueue<>(Comparator.comparing(Term::getWeight));
42         for (Term t : myTerms) {
43             if (!t.getWord().startsWith(prefix)) {
44                 continue; // don't process if doesn't begin with prefix
45             }
46 
47             if (pq.size() < k) {
48                 pq.add(t);
49             } else if (pq.peek().getWeight() < t.getWeight()) {
50                 pq.remove();
51                 pq.add(t);
52             }
53         }
```

The assignment write-up indicates that this code, followed by the code that stores the contents of the `pq` in a `LinkedList list` and returns `list`, have complexity $O(N + M \log k)$ where N is the total number of terms, M is the number of terms with a matching prefix, and k is the number of "top" matches.

PROBLEM 11:

What part of the code shown above contributes the N in the complexity analysis?

A. The `for` loop on line 42, e.g., even when no terms match the prefix (so $M == 0$).

B. The code on lines 47-52 which add matching prefix terms to the priority queue.

C. The code not shown, after the loop, that removes elements from the priority queue and adds them to the front of a `LinkedList list`.

PROBLEM 12:

When a `Term` is removed by the code on line 50, which one of the following is true about the `Term` removed?

A. It has the highest weight of all the elements in the priority queue at the time of removal.

B. It has the lowest weight of all the elements in the priority queue at the time of removal.

C. It has the alphabetically first String of all the elements in the priority queue at the time of removal.

D. It has the alphabetically last String of all the elements in the priority queue at the time of removal.

PROBLEM 13:

The code below is proposed as a replacement for the code shown (and missing) above for method `topMatches` This code produces exactly the same result as the code in the previous problem. However, this code has a different asymptotic complexity.

```java
ArrayList<Term> list = new ArrayList<>();
for(Term t : myTerms) {
    if (t.getWord().startsWith(prefix)) {
        list.add(t);
    }
}
Collections.sort(list,Comparator.comparing(Term::getWeight)
                                .reversed());
return list.subList(0,Math.min(list.size(),k));
```

What is the asymptotic complexity of this code? (Assume that k is much less than M which is less than N.)

A. $O(N + M \log M)$

B. $O(N + N \log M)$

C. $O(M + N \log k)$

D. $O(N + N \log k)$

PROBLEM 14:

For the class `BinarySearchAutocomplete` the code on lines 42-45 at the beginning of these problems is replaced by the code that follows, where `first` and `last` where assigned values before the loop, these values were obtained by calling `firstIndexOf` and `lastIndexOf`, respectively, which used *binary search* code to find the values efficiently.

```java
for(int j=first; j <= last; j++) {
    Term t = myTerms[j];
```

The assignment of values to `first` and `last` before the loop has complexity $O(\log N)$. Recall that there are M matching terms. The complexity of `BinarySearchAutocomplete.topMatches` is:

A. $O(\log N + M \log k)$

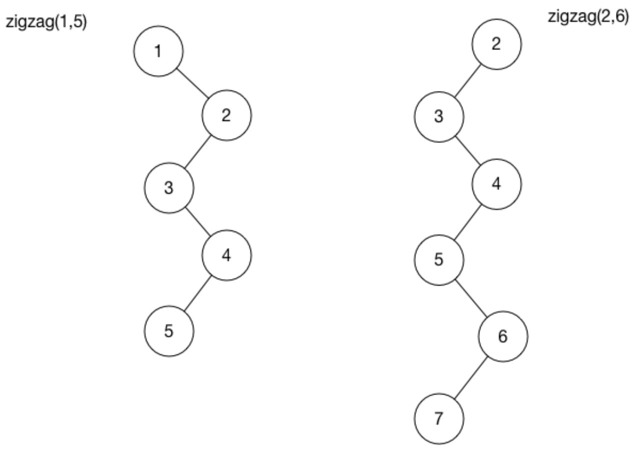
B. $O(M + M \log k)$

C. $O(N + M(\log k + \log N))$

D. $O(N \log M)$

Consider a *zigzag* tree as described here. A binary tree is a *zigzag tree* if for every node in the tree (except leaf nodes), nodes with odd values have only a right-child, nodes with even values have only a left-child and the value in each node (except the root) is one more than the parent value.

The diagram below shows the results of two calls of the method `zigzag` that returns a *zigzag* tree. The parameter `numNodes` is the number of nodes in the returned tree. The parameter `rootVal` is the value in the root of the tree returned. An incomplete version of `zigzag` appears after the diagram.



PROBLEM 15:

Assume that the value of parameter `numNodes` is greater than or equal to zero. What is the missing recursive call that makes the method work correctly? 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.**

```
25     public TreeNode zigzag(int rootVal, int numNodes){
26         if (numNodes == 0) return null;
27         TreeNode below = // recursive call
28         if (rootVal % 2 == 0) {
29             return new TreeNode(rootVal, below,null);
30         }
31         else {
32             return new TreeNode(rootVal,null,below);
33         }
34     }
```

PROBLEM 16:

For the call `zigzag(v,n)` how many nodes are there in the tree returned? Assume line 27 has the correct call.

A. $n-1$

B. n

C. $n+1$

PROBLEM 17:

For the call `zigzag(v,n)` what value is in the leaf of the tree returned? Assume line 27 has the correct call.

A. $v + n$

B. $v + n + 1$

C. $v + n - 1$

PROBLEM 18:

If `null` is replaced by `below` on both lines 29 and 32 how many values will be printed using a standard `inOrder` traversal of the tree returned by the call `zigzag(v,n)`? Assume line 27 has the correct call.

A. $2^n - 1$

B. 2^{n-1}

C. $2^{n-1} - 1$

PROBLEM 19:

As in the previous problem, if `null` is replaced by `below` on both lines 29 and 32, what is the runtime of the call `zigzag(v,n)`? Assume line 27 has the correct call.

A. $O(\log n)$

B. $O(n)$

C. $O(n^2)$

D. $O(2^n)$

PROBLEM 20:

The interface Comparator contains a method compare that returns an int value. When is the value returned equal to zero in a class implementing Comparator<String>?

A. When one of the two parameters to compare is null

B. When the two parameters are conceptually equal, according to the logic of the class implementing Comparator<String>.

C. When a.compareTo(b) == 0 for String parameters a and b.

In the next three questions you're asked to reason about a file of words whose contents are shown below.

```
the ant the ant the large ferocious ant
the frog the frog the small ferocious frog
my smile my smile because of the ant and the frog
```

The code below is run using the file above as input.

```
138    Scanner s = new Scanner(new File("data/words.txt"));
139    Map<String,Integer> map = new HashMap<>();
140    while (s.hasNext()){
141        String st = s.next();
142        map.putIfAbsent(st,0);
143        map.put(st,map.get(st)+1);
144    }
145    System.out.println(map);
```

The output printed by the code on line 145 is shown below.

```
{the=8, small=1, large=1, ferocious=2, frog=4, ant=4, and=1, of=1, because=1, my=2, smile=2}
```

PROBLEM 21:

If HashMap on line 139 is replaced by TreeMap, the output printed by line 145 changes. What are the first three key/values printed by the code when a TreeMap is used?

A. and=1, of=1, because=1

B. and=1, ant=4, because=1

C. the=8, frog=4, ant=4

PROBLEM 22:

The code below executes after the code filling the map with values shown above. Which of the following could be the first three key/value pairs printed? (the actual output has one key/value pair per line).

```java
Comparator<String> comp = Comparator.reverseOrder();
ArrayList<String> list = new ArrayList<>(map.keySet());
Collections.sort(list,comp);
for(String st : list){
    System.out.printf("%s=%d\n",st,map.get(st));
}
```

A. `the=8, frog=4, ant=4`

B. `the=8, smile=2, small=1`

C. `and=1, ant=4, because=1`

PROBLEM 23:

The sorting code in the previous problem is replaced by the code below. which executes after the loop reading the file and filling the map.

```java
public class Comp implements Comparator<String> {
    Map<String,Integer> myMap;
    public Comp(Map<String,Integer> map){
        myMap = map;
    }
    public int compare(String a, String b) {
        int diff =  myMap.get(b) - myMap.get(a);
        if (diff == 0) return a.compareTo(b);
        return diff;
    }
}
```

```java
ArrayList<String> list = new ArrayList<>(map.keySet());
Collections.sort(list, new Comp(map));
for(String st : list){
    System.out.printf("%s=%d\n",st,map.get(st));
}
```

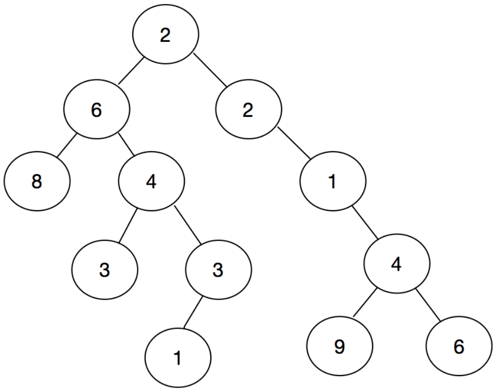
Which of the following could be the first five key/value pairs printed? (the actual output has one key/value pair per line).

A. `the=8, ant=4, frog=4, ferocious=2, my=2`

B. `the=8, frog=4, ant=4, my=2, ferocious=2`

C. `the=8, frog=4, ant=4, smile=2, my=2`

In the next four problems, assume all values in a tree are positive integers. The tree shown here will be used in examples.



PROBLEM 24:

The method `maxLeaf` below correctly returns the largest/maximal leaf-Node value; 9 in the tree above.

```
public int maxLeaf(TreeNode t) {
    if (t == null) return 0;
    if (t.left == null && t.right == null) {
        return t.info;
    }
    return Math.max(maxLeaf(t.left),maxLeaf(t.right));
```

What is the recurrence relation that describes the runtime of `maxLeaf` *when trees are roughly balanced.* $T(N)$ is the time for `maxLeaf` to run on an N-node tree.

A. $T(N) = 2T(N/2) + O(1)$

B. $T(N) = T(N/2) + O(N)$

C. $T(N) = T(N-1) + O(1)$

D. $T(N) = T(N-1) + O(N)$

PROBLEM 25:

What is the recurrence relation that describes the runtime of `maxLeaf` when *trees are extremely unbalanced*, e.g., for every node all child-nodes are in the left-subtree and none in the right-subtree. $T(N)$ is the time for `maxLeaf` to run on an N-node tree.

A. $T(N) = 2T(N/2) + O(1)$

B. $T(N) = T(N/2) + O(N)$

C. $T(N) = T(N-1) + O(1)$

D. $T(N) = T(N-1) + O(N)$

PROBLEM 26:

Method `maxPath` below is missing a return value on line 46 so that it correctly returns the sum of the nodes on the maximal root-to-leaf path in a binary tree. In the tree shown above the root-to-leaf paths sum to 16, 15, 16, 18, and 15 since the paths are 2-6-8, 2-6-4-3, 2-6-4-3-1, 2-2-1-4-9, and 2-2-1-4-6. The method should return 18, the maximal sum.

```
39  public int maxPath(TreeNode t) {
40      if (t == null) return 0;
41      if (t.left == null && t.right == null){
42          return t.info;
43      }
44      int lmax = maxPath(t.left);
45      int rmax = maxPath(t.right);
46      return // value to return here
47  }
```

What expression should be returned on line 46 so the method works correctly? Evaluating the expression should be $O(1)$, independent of the number of nodes in the tree parameter.

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 27:

Is the asymptotic, big-Oh, runtime of `maxLeaf` different when the tree parameter is roughly balanced compared to when the tree is extremely unbalanced?

A. Yes, the big-Oh runtime is different for an unbalanced tree compared to a balanced tree.

B. No, the big-Oh runtimes are the same regardless of the shape of the tree.

For the next two problems you'll answer questions about a sorting algorithm that sorts a file of strings which is so large that all the strings do not fit into memory at once. The file consists of exactly $k \cdot m$ strings where m is the largest number of strings that can fit into memory at one time. In analyzing the complexity of the algorithm you should use O-notation expressing your answers in terms of *both* k and m.

PROBLEM 28:

The first part of the algorithm follows: Create k different files by repeating this step k times.

> Read m items from the large file into memory, sort the m strings with `Collections.sort` and then write the now-sorted strings to a new file of m elements.

What is the big-O time-complexity of this first part of the algorithm that reads one big file and creates k smaller files, each of which contains m sorted strings? 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 29:

The second part of the algorithm follows: First, one string is read from each of the k sorted files and stored with an indication of the file it came from in a priority queue implemented using `java.util.PriorityQueue` using a `Comparator` based on comparing the string value. The priority queue now has k elements.

The following process is repeated until the priority queue is empty:

- remove the smallest (string) element from the priority queue, write it to a final output file
- if the file from which the string came is not empty then
  - read next string from the file and store the string in the priority queue with a reference of the file from which it was read

Eventually the final output file will be sorted and contain exactly the strings in the original very large file.

What is the time-complexity of this priority queue part of the algorithm that reads the k files and creates one output file?

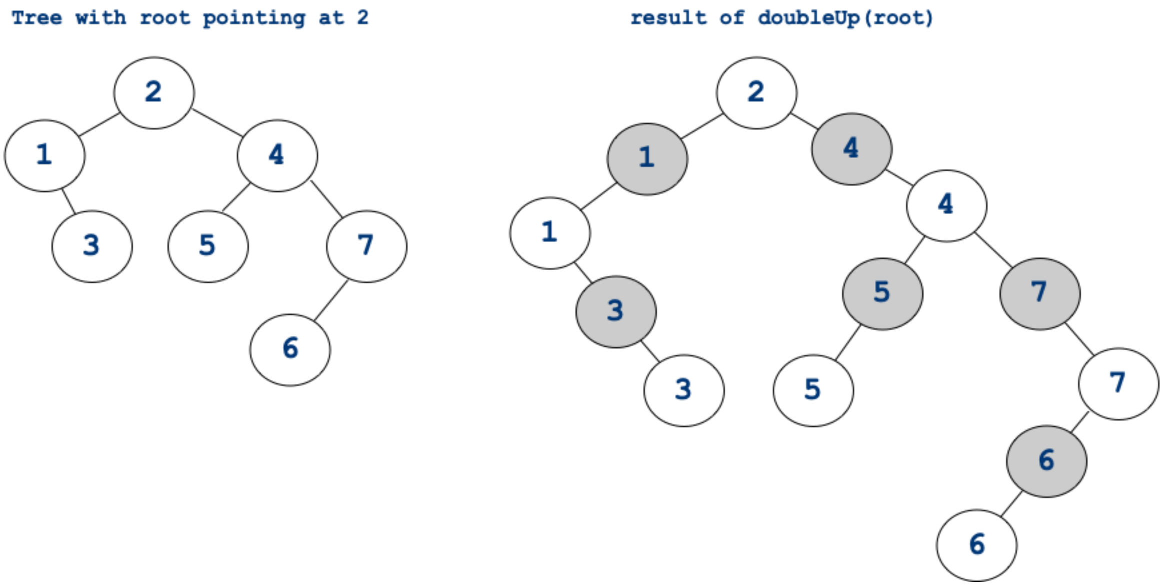
A. $O(km \log k)$

B. $O(km \log m)$

C. $O(k \log k)$

D. $O(m \log k)$

The next three problems are based on the concept of a *doubleUp* tree. The diagram below shows a binary tree on the left and the result of `doubleUp(root)` of that tree on the right, where `root` references the root node containing the value 2 of the tree on the left.



To create a *doubleUp* tree, for every node N in a tree T:

- The *doubleUp* of a `null` tree is `null`
- First *doubleUp* N's left subtree and right subtree, call these `dleft`, and `dright`, respectively. Node N references these as its left-child and right-child, respectively.
- If N has a non-null left-child/subtree, create a new left-child of N, a Node containing the same value as the root of the left-subtree, the new Node's left subtree is `dleft`, its right-child is `null`.
- If N has a non-null right-child/subtree, create a new right-child of N, a Node containing the same value as root of the right subtree, the new Node's right subtree is `dright`, its left-child is `null`.

In the diagram above, the new nodes created are shaded gray.

PROBLEM 30:

If `root` points to the root of a tree containing 22 nodes, how many nodes are in the tree returned by `doubleUp(root)`?

A. 43

B. 44

C. it depends on how many left-children and right-children there are, the answer cannot be determined.

PROBLEM 31:

The code for method `doubleUp` below will correctly return a *doubled-up* tree of its tree parameter.

What is the expression/value for `EXPR_1` on line 76 that makes this code correct?

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.**

```
71  public TreeNode doubleUp(TreeNode t){
72     if (t == null) return null;
73     t.left = doubleUp(t.left);
74     t.right = doubleUp(t.right);
75     if (t.left != null) {
76         t.left = new TreeNode(EXPR_1, t.left,null);
77     }
78     if (t.right != null) {
79         t.right = new TreeNode(EXPR_2, null,t.right);
80     }
81     return t;
82  }
```

PROBLEM 32:

Similarly, what is the value of `EXPR_2` on line 79 that makes this method correct?

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 33:

Extra Credit

In class, the problem of finding the shortest wordladder between `"colts"` and `"house"`, e.g., `horse house rouse route routs bouts bolts colts` was used as a reference to a song-title for different songs by Kanye and Del Shannon. What is the title of the song?

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.**