

CS2110-Fall2014

Exam 3 for sgr7sg

Samantha Rafalowski

Your original score on this exam was 115.00 of 134 points.

Your exam was regraded to a 118.00; see the comments below.

Your breakdown of points per page is below.

Page	Score	Max
1	0	0
2	18	20
3	20	22
4	15	15
5	8	14
6	10	14
7	12	15
8	22	24
9	10	10
10	0	0

A graded scan of each page follows.

Regrader comments: I can give you half credit (+3 points) for your response overall however it wasn't clear enough in your response that the "data types" of the children nodes ("left" and "right") are also of type TreeNode. Calling same operations over the tree is true but isn't enough to explain this concept.

CS2110 Final Exam – Fall 2014

Name: Samantha Rafalawski
PRINT your name here

Pledge:

On my honor as a student, I have neither given nor received aid on this exam.

Samantha Rafalawski
Your signature here

You MUST do all three of the following:

1. Fill in your name and sign the pledge above. If you do not sign the pledge we will not grade your exam.
2. On the top of **every** page, including this one, write your email ID (e.g., mst3k).
3. On the bottom of **this** page, bubble-in your email ID.



Grading is done one page at a time. Keep each answer on the same page as the question it answers.

The total number of points on this exam is 134 points.

Policies

- This exam is closed-book, closed-notes, (closed everything). All you need for this exam is a pen or pencil.
- You may not discuss any part of the content or format of this exam with anyone before it is graded because anyone who has not taken the test yet might overhear. Discussing it before it is graded is considered a violation of the Honor Code.
- You may do the problems in any order. Later problems might be either harder or easier than earlier problems.
- Write clearly. If your answer is illegible it cannot be graded.
- State any assumptions you make next to the question(s) to which they apply.

Good luck and have a great winter break!

	(a)(b)(c)(d)(e)(f)(g)(h)(i)(j)(k)(l)(m)(n)(o)(p)(q)(r)(s)(t)(u)(v)(w)(x)(y)(z)	Do not write in this area					
	(a)(b)(c)(d)(e)(f)(g)(h)(i)(j)(k)(l)(m)(n)(o)(p)(q)(r)(s)(t)(u)(v)(w)(x)(y)(z)	Department	Course	Exam	Page	Version	
	(a)(b)(c)(d)(e)(f)(g)(h)(i)(j)(k)(l)(m)(n)(o)(p)(q)(r)(s)(t)(u)(v)(w)(x)(y)(z)	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	
	(a)(b)(c)(d)(e)(f)(g)(h)(i)(j)(k)(l)(m)(n)(o)(p)(q)(r)(s)(t)(u)(v)(w)(x)(y)(z)	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	
	(a)(b)(c)(d)(e)(f)(g)(h)(i)(j)(k)(l)(m)(n)(o)(p)(q)(r)(s)(t)(u)(v)(w)(x)(y)(z)	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	

Question 1 [6 points]: Let's say you have two algorithms that both solve the same problem correctly, and one is more efficient (i.e. has a better time complexity) than the other. Give two good reasons why you might still choose to implement the slower algorithm in your software.

- ✓ ① If you are only using small inputs in your code
- ✓ ② if the slower algorithm is simpler or easier to understand/implement (ie: recursive in the case of Fibon acci)

Question 2 [6 points]: Do you agree or disagree with the following statement? Briefly explain your reason(s).

Statement: "It is sufficient to perform asymptotic analysis on two algorithms with an input size $n = 5$. This input size will clearly show the relative differences in their efficiencies."

✓ I disagree with this statement as algorithms can perform differently depending on input size. Some algorithms may perform similarly to others at a small input like $n=5$ (ie $f(n)$ and $f(1)$), but very differently at large inputs.

Question 3 [8 points]: Circle "T" for true and "F" for false.

- a) T ☒ F Algorithms must complete in a finite time for any input
- b) T ☒ F Algorithms must be written in a programming language
- c) ☒ T F Algorithms must behave the same each time they are given the same input
- d) T ☒ F Algorithms could be interpreted in more than one way
- e) ☒ T F To execute efficiently, Binary Search always uses binary code
- f) ☒ T F For large enough inputs, the lower-order terms in the exact running time of an algorithm are dominated by the higher-order term
- g) T ☒ F Since linear search makes no assumptions about its input, it is always more desirable than binary search
- h) T ☒ F Algorithm A that has time complexity $O(n^3)$ is asymptotically superior to Algorithm B that has time complexity $O(n^2)$

Question 4 [4 points]: If an algorithm takes 400 milliseconds to process 50,000 data items and 1603 milliseconds to process 100,000 data items, its big-O seems to be: (Circle one.)

- ☒ a) $O(n^2)$
- b) $O(n)$
- c) $O(\log(n))$
- d) $O(n^3)$

Question 5 [4 points]: If the actual time complexity function of an algorithm is: $10n^2 + 4n + 2$, which of the following is a correct statement? (Circle one.)

- ☒ a) To define big-O, the whole equation is taken into consideration
- b) To define big-O, the factor "+ 2" can be discarded
- c) To define big-O, $4n+2$ can be ignored
- ☐ d) To define big-O, only the leading term is used and the coefficient is ignored

Question 6 [6 points]: Evaluate the statement given below. Indicate if you believe this statement is true or false and justify your answer. You may describe an example to justify your argument. (Do not write code to answer this question.)

Statement: Given a recursive algorithm that has elegant and concise code, it must mean that the resulting algorithm is efficient.

False. A counter-example to this would be Fibonacci, which recursively has a complexity of 2^k , but non-recursively runs much faster. This is because for some large inputs, computing recursively can approach infinite time complexity.

Question 7 [8 Points]: Which of the following are true statements about *abstract data types* (ADTs)? Circle the letter for each one that is true. (Zero or more could be true.)

- ☐ a) In Java, an abstract class is a good match to the concept of an ADT
- ☒ b) In Java, an interface is a good match to the concept of an ADT
- ☒ c) We could compare two ADTs by comparing the efficiency of the implementation of their operations
- ☐ d) In Java, it makes sense to say that Map defines a data structure that implements the HashMap ADT

HashMap implements map

Your score on this page is 20/22

Question 8 [6 points]: Examine the following recursive algorithm ("mysteryAlgo"). What would happen if the input to this algorithm ("m") was 3? Is this a problem?

If you feel the algorithm will run to completion and produce an output with an input of 3, write "This recursive algorithm works fine for input m=3", otherwise briefly explain why you feel there is a problem and suggest a way to modify the code below (if appropriate).

```
public static int mysteryAlgo(int m) {
    if (m == 5)
        return 1;
    else
        return m * mysteryAlgo(m-2);
}
```

This recursive algorithm will never reach the base case, thus does not make progress to the base case for an input less than 5. To modify the code, the base case would need to be 0 or else input must all be > 5.

$3 * \text{mystery}(1)$
 $+ 1 * \text{mystery}(-1)$
 $* (-1) * \text{mystery}(-3)$
 $* (-3) * \text{mystery}(-3)$

OK

Question 9 [9 points]: Binary search works correctly for lists that are in sorted order.

Values:	-7	-6	-5	-1	0	2	4	7	8	9	10	17	23	25	32
Index:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Given the list above, what sequence of **indices** will the binary search algorithm visit when looking for the following targets? Is the target found? (Circle Y for "yes" and N for "no")

(a) 8

7, 11, 9, 8

Target found? Y / N

(b) 35

7, 11, 13, 14

Target found? Y / N

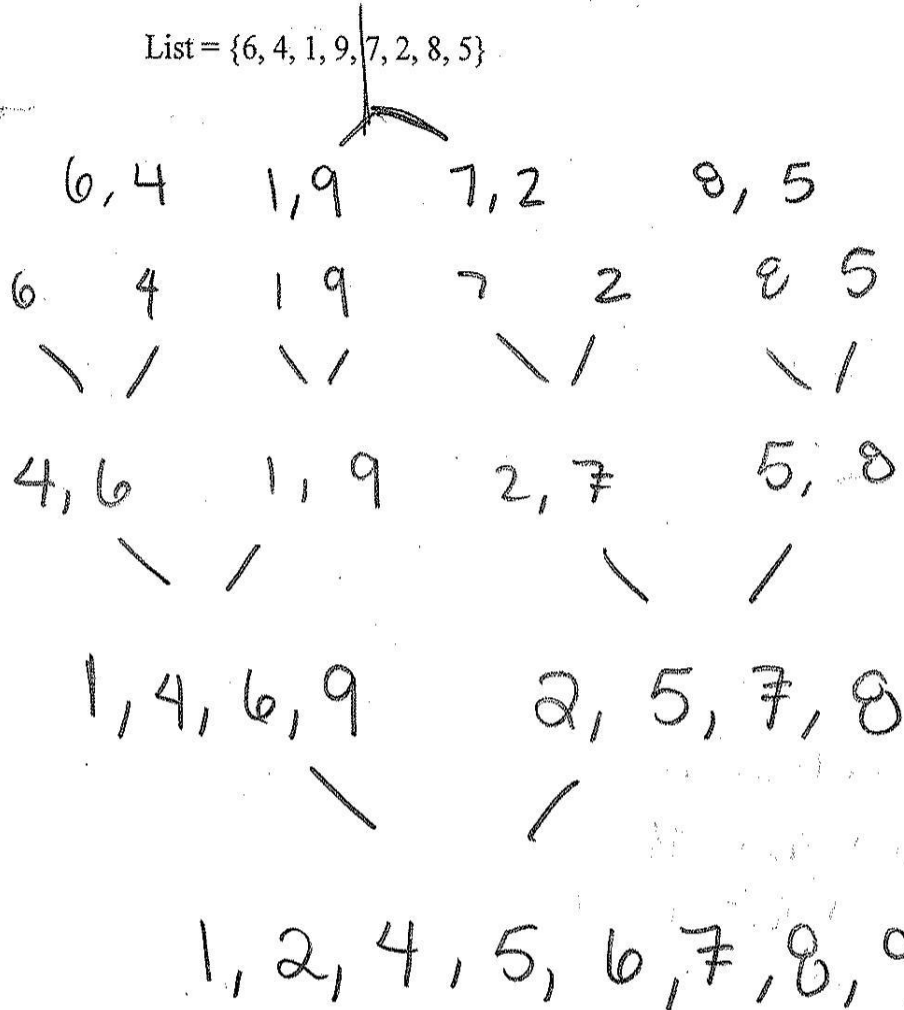
(c) -5

7, 3, 1, 2 ✓

Target found? Y / N

Question 10 [8 points]: Perform MergeSort on the following list of integers. Show enough steps to illustrate the various stages of MergeSort as well as the final sorted list.

List = {6, 4, 1, 9, 7, 2, 8, 5}



Question 11 [6 points]:

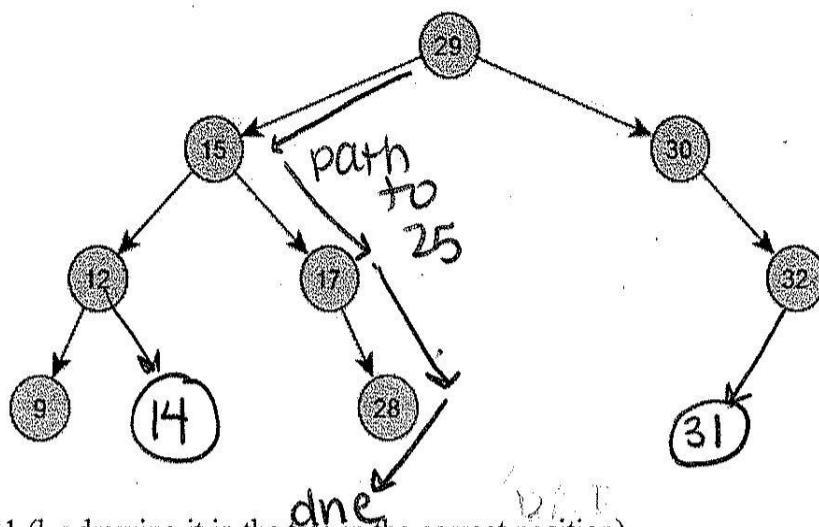
We say that the `TreeNode` class we studied is a recursive data structure. Explain why in a sentence or two. (Note: this is not a question about recursive algorithms!)

`TreeNode` is a recursive data structure because the same operations must be called over to create a Tree or find a `TreeNode`. For example, to find the root of a Tree, the parent of each node must continually be called until it doesn't exist.

left and right children are of type `TreeNode`

Your score on this page is 8/14

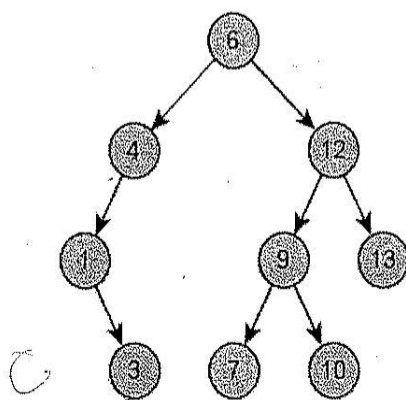
Question 12 [8 points]: Perform the following operations on the **Binary Search Tree** given below.



- Add node 31 (by drawing it in the tree in the correct position)
- Add node 14
- Find node 25, draw on the tree (above) to show the path you would take to search for node 25. Explain how you know 25 is not present in the tree?

25 is not present because there is no node that exists in the right subtree of 15 that is 25. Also, 28 has no leaves, which is where 25 would be.

Question 13 [6 points]: Consider the following tree:



- If you wanted to print the data in the tree starting with the smallest value and ending with the largest, which traversal method would you use? Circle your answer.

in-order

pre-order

post-order

left, right, root

- Show the results of applying post-order traversal by putting one number in each box below, starting with the first node visited.

3	1	4	6	7	10	9	13	12
---	---	---	---	---	----	---	----	----

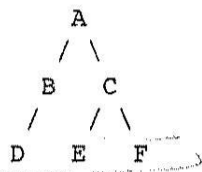
4 6 7 10 9 13 12 ?

Question 14 [15 points]:

Write a recursive method `calcBalanceVal()` for the `TreeNode` class that we studied to do the following. We want to calculate a "balance-value" for a tree rooted at the current node, where the balance-value is a measure of how balanced the tree is. The rules for calculating this value are:

- 12 pts**
1. If the current node is a leaf, return 1.0 as the balance-value for the node. ← base case
 2. If the node has two subtrees, the node's balance-value is the average of its left-subtree's balance-value and its right-subtree's balance value. ← recursive case
 3. If the node has just one subtree, the node's balance-value is the average of the subtree's balance-value and 0.0. (Same as saying one-half the subtree's balance-value.)

The following example shows a tree and then lists the balance values for each node:



Leaves: D, E and F have values 1.0 by rule #1 above.
 Node C has value 1.0 by rule #2 above.
 Node B has value 0.5 by rule #3 above.
 Node A has value 0.75 by rule #2 above.

```
public double calcBalanceVal() { // a method in class TreeNode
```

```
    if (this.left == null && this.right == null)
        return 1;
```

```
    else if (this.left == null || this.right == null)
        return ((calcBalanceVal(this)) + 0) / 2;
```

```
    else return ((calcBalanceVal(this.left)) + (calcBalanceVal(this.right))) / 2;
```

```
    }
```

Recursive calls not correct. See Key

Questions on material from previous exams follow. We'll drop the lowest score of these 5.

Question 15 [6 Points]: In software engineering, what percentage of the total cost of a software system over its entire lifetime is due to maintenance?

67 %

Question 16 [6 Points]: (Write two terms that complete these sentences to make correct statements about testing.) When a programmer tests methods and classes as they are being developed, this is called unit testing. If he or she designs the tests without using any knowledge of the code that implement the classes, this is an example of black box testing.

Question 17 [6 Points]: Describe two ways that Java interfaces are different than Java abstract classes.

① Interfaces have a built-in compare method but abstract classes do not (you'd define them)

② Interfaces are implemented, abstract classes are extended

Question 18 [6 Points]: Assume we define a method play() in an interface called Playable, and classes Song and Video implement this interface. A class IPod has a method void playItem(Playable p), which does some things including calling play() on the object p that is passed to this method.

Circle the letter for any or all of the following statements that are true:

- 2
- ☒ (a) Inside the method playItem(), we normally use instanceof on p to determine what kind of object has been passed before we can play that object
 - ☐ (b) Passing an object p in this way is a good example of using abstraction in a program
 - ☐ (c) The concept of run-time polymorphism is something that applies here in how playItem() will play an object

Question 19 [6 Points]: In a mobile application, you could set up a kind of "proximity alert" where some of your code will be executed when the phone's location gets close enough to a specific location. In the GUI library, you can set up code that executes when a text-field is edited or the mouse is pressed. These are both examples of a type of programming known as:

event-driven programming

Question 20 [10 Points]: You are given a List of Strings called *nameList*, where each item is a name. Write code to do the following. (This code does not have to be in a method.)

(a) Define a *TreeMap* object named *counts* and use it to store how often each name occurs in *nameList*.

(b) Create a *TreeSet* called *longNames* and use it to store all the names with a length greater than 5.

For example, if *nameList* contains these as Strings:

Kate, William, LeBron, Kate, William, Harry, Kate

then your code should make *counts* store the value 3 for "Kate", 2 for "William", 1 for "LeBron", and 1 for "Harry". Your code should make the set *longNames* store: "William", "LeBron"

```
TreeMap <String, Integer> counts = new TreeMap<String, Integer>();
TreeSet <String> longNames = new TreeSet<String>();
for (String s : nameList) {
    if (s.length() >= 5) {
        longNames.add(s);
    }
    if (!counts.containsKey(s)) {
        counts.put(s, 1);
    }
    else {
        int n = counts.getValue(s);
        counts.put(s, n+1);
    }
}
```

