112Exam2

Wednesday, November 13, 2019 6:36 PM

CS112OldMidtermExa...

CS 112
MidtermExam A
65 minutes
Closed Book and no Aid allowed
15th November 2017
Instructor: Abbas Attarwala
Boston University

Note: It is a violation of the Honor Code to discuss this midterm exam question with anyone until after everyone in CS 112 (A and B section) has taken the exam.

Enter your First Name: ____________________

Enter your Last Name: ____________________

Enter your Student#: ____________________

"I pledge my honor that I have not violated the Honor Code during this examination."

Your Signature: ____________________

Question:	1	2	3	4	Total
Points:	13	8	20	0	41
Bonus Points:	0	0	0	3	3
Score:					

DO NOT TURN THIS PAGE UNTIL YOU HAVE BEEN ASKED TO DO SO

1. Given the following Linked List class[1]: **You will find the sub-questions on the next page.**

```
public class LinkedList<E extends Comparable<E>> {
        private static class Node<E>
        {
                private E data;
                private Node<E> next;

                public Node(E d)
                {
                        data=d;
                        next=null;
                }
        }
        private Node<E> head;
        private Node<E> tail;


        public boolean increasingSequence() throws EmptyLinkedList
        {
                //TODO: Complete the rest of this method.
```

```
        }

        /*
        * n must be non-null when _increasingSequence is called.
        *You must use recursion here.
        */
        private boolean _increasingSequence(Node<E> n)
        {
            if (n.next==null)
                return true;
            Node<E> previous=n;
            Node<E> next=n.next;
            //TODO: Complete the rest of this method.




        }
    }
```

[1] The template for the `LinkedList` class is exactly the same as seen in lecture

(a) Complete the method `increasingSequence`[2] such that the method returns back a `true` when the `LinkedList` is in an increasing order and returns back `false` when the `LinkedList` is not in an increasing order. This method returns back an exception `EmptyLinkedList`[3] when the `LinkedList` is empty (i.e. `head` points at `null`). You can call the `compareTo` method to compare two `Nodes`. We provide the following 9 `LinkedLists` and their expected output.

$3 --> 7 --> 5 --> null$	*must return* $FALSE$
$3 --> 4 --> 5 --> 6 --> null$	*must return* $TRUE$
$3 --> 5 --> 7 --> 9 --> null$	*must return* $TRUE$
$3 --> 3 --> 3-> null$	*must return* $FALSE$

$3 --> 4 --> 5 --> 7 --> null$	*must return TRUE*
$6 --> 5 --> 4 --> 3 --> null$	*must return FALSE*
$-1 --> 0 --> null$	*must return TRUE*
$3 --> null$	*must return TRUE*
< *EMPTYLINKEDLIST* >	*must throw EmptyLinkedList Exception*

You can use the space provided in the code to write your answer.

On .java file

(b) Write (do not solve) the recurrence $T(n)$ for the `_increasingSequence`.

$T(n) = T(n-1) + c$

$+3c$

(c) Solve the recurrence $T(n)$ for the `_increasingSequence` from b) above, in the worst case and find the $\mathcal{O}(...)$. Show us all the steps and insert your final $\mathcal{O}(...)$ answer by drawing a box around it.

$T(n) = T(n-1) + 1c$

$= [T(n-1-1) + c] + c = T(n-2) + 2c$

$= [(T(n-1-1-1) + c) + c] + c = T(n-3)$

$\vdots$

$\boxed{T(n-k) + k \cdot c}$

$T(n-n) + nc$

$T(0) + nc$

$= 1 + nc$

$= O(n)$

Total for Question

[2] This problem is very similar to `removeDuplicates` from review session.
[3] You do not have to create this class

2. Given the following general `Tree`. There is no coding required for this question.

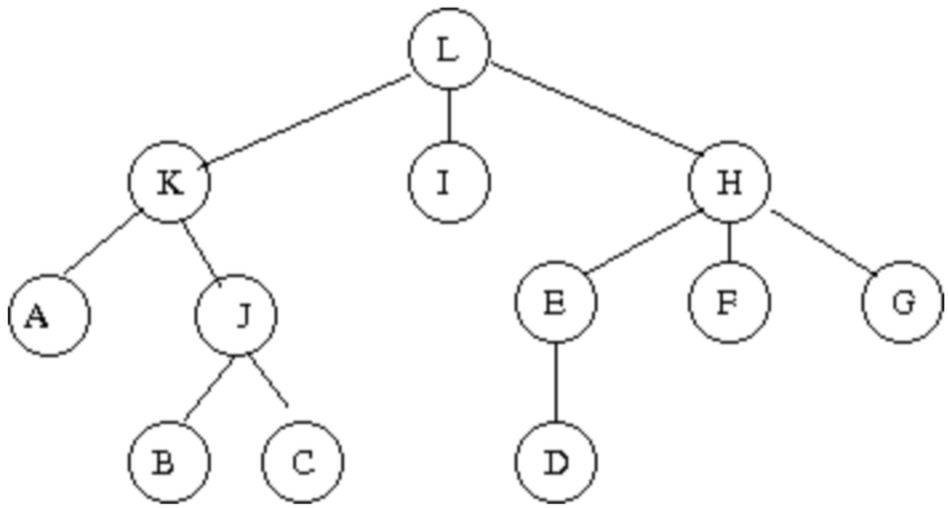

Figure 1: Tree with 12 nodes

(a) What is `PreOrder` traversal on the above tree?

LKAJBCIHEDFG

(b) What is `PostOrder` traversal on the above tree?

ABCJKIDEFGHL

(c) What is `breadth-first order` traversal or levelwise order traversal on the above tree?

L KIH AJEFG BCD

(d) How many `leaf` nodes does the above tree contain?

7

(e) What is the `height` of the above tree?

4

Total for Questio

3. Given the following `BinaryTree` class

```
public class BinaryTree<T> {
        private static class Node<T>
        {
                private int data;
                private Node<T> left;
                private Node<T> right;

                public Node(int d)
                {
                        data=d;
                }
        }
        private Node<T> root; //points at the root of the tree

        /* Returns the height of the tree. The height of an empty tree OR
        * a tree with just a single node is 0. You must use recursion
        *on _getHeight to answer this
        *question. */
        public int getHeight()
        {
                //TODO
                return _getHeight(root);
        }

        private int _getHeight(Node<T> r)
        {
                //TODO
                if (r == null ||(r.left==null && r.right==null))
                        return 0;

                int leftHeight = _getHeight(r.left) + 1;
                int rightHeight = _getHeight(r.right) + 1;

                return max(leftHeight, rightHeight);
```

NULL => 0

=>

```
        }
```



```
        /* Returns the clone of BinaryTree represented by the reference 'this'
        *Pay careful attention to the return types of clone
        *and _clone. */
        public BinaryTree<T> clone ()
        {
                BinaryTree<T> ret=new BinaryTree<>();
                //TODO








        }
        private Node<T> _clone(Node<T> r)
        {
                if (r==null)
                        return null;
                //TODO








        }
}
```

(a) In the lecture, we calculated the height of a general `Tree` using recursion. You are now asked to find the height of a `BinaryTree`[4]. You will use recursion and complete `getHeight` and `_getHeight`. You can use the space in the code to answer this question.

On .java file

(b) In your assignment3 you cloned a `Set` and then in your assignment6 you cloned a `LinkedList`. In this question, we ask you to clone a `BinaryTree`. You will use recursion and complete `clone` and `_clone` You can use the space in the code to answer this question.

on .java file.

[4] Hint: You can use the `max` function from the `Math` class

(c) In the `main` function provided, write code so that you create the following `BinaryTree`[5] using the above `BinaryTree` class.

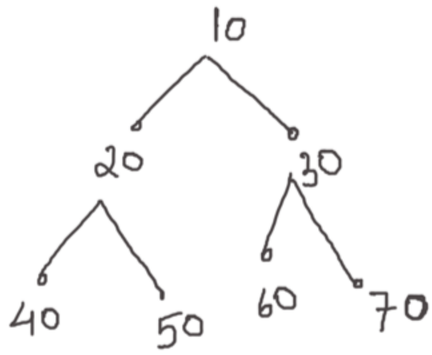

Figure 2: BinaryTree with 7 nodes

```
//You can assume that this main function belongs inside the above
//BinaryTree class.
public static void main(String[] args)
{
```

}

Total for Question

5 In lecture we created similar trees by hardcoding values to test various algorithms. You can use similar technique when answering this question

4. (a) Given the following code[6]: (1 ½ (b

①
```
for (i=1; i<=n; i++)
    for (j=1;j<=i;j++)
        k=k+1;
```
②

You are asked to count the actual number of times the instruction $k = k + 1$ is executed and then find the $\mathcal{O}(...)$. Your friend from NorthEastern performs the following series of steps:

① ② i

$$= \sum_{i=1}^{n} \sum_{j=1}^{n} 1$$

$$= \sum_{i=1}^{n} n$$

$$= n \sum_{i=1}^{n} 1$$

$= n * n$

$= n^2$

i.e. n^2 *is* $\mathcal{O}(n^2)$. The final $\mathcal{O}(n^2)$ answer is correct. However, there is a flaw in the above steps. **What is the flaw and repeat the steps[7] such that the flaw is corrected.**

$$= \sum_{i=1}^{n} \sum_{j=1}^{i} 1$$

$$= (1)\cdot 1 + (1)\cdot 2 + \dots (1)\cdot n$$

$$= 1 + 2 + \dots + n$$

$$= \frac{n(n+1)}{2} = \frac{n^2+n}{2} = \frac{1}{2}\cdot(n^2+n) = O(n^2)$$

[6]Similar to Piazza post @199

[7]You must also use the $\sum$ in your step(s) similar to what your friend did.

(b) What is the $\mathcal{O}(\dots)$ for the following piece of code? Make sure that you explain your final answer. (1 ½ (b

```
public void someFunction(int n){
    int i,j,k,count=0;
    for (i=n/2;i<=n;i++)
```

$\leftarrow O(n/2) \cdot O(\frac{1}{2}\cdot n) = O(n)$

```
    {
        for (j=1;j<=n;j=2*j)
        {
            for (k=1;k<=n;k=k*2)
            {
                count++;
            }
        }
    }
}
```

$O(\log_2 n)$

$O(\log_2 n)$

$O(n \cdot \log n \cdot \log n) = O(n \log^2 n)$

$O(n + \log n + \log n) = O(n)$

Extra Rough Page