Recursive Code Modelling Review Session

19su Week 4

Part 1: Writing Recurrences

Give a recurrence formula for the running time of each of the following code snippets. You are not required to find the exact constant and you are free to use substitutions (c_1, c_2 etc.) if so desired. You also don't need to find the closed form.

1) CSE 332 18su

```
public static int question1(int n) {
        if (n <= 15) {
                return n * n;
        } else {
                for (int i = 0; i < n; i++) {
                        for (int j = 0; j < i; j++) {
                                System.out.println(i + j);
                        }
                }
                return question1(n / 2) + question1(n / 2);
        }
}
```

2)

```
public static double question2(double n) {
        if (n >= 15) {
                return question2(n – 1) * question2(n – 1);
        } else {
                int bonus = 0;
                for (int i = 0; i < n; i++) {
                        for (int j = 0; j < n; j++) {
                                bonus++;
                        }
                }
                return Math.sin(n – 1) * Math.cos(n – 1) + bonus;
        }
}
```

3) Let the original number of elements in `input` be n.

```
public static int question3 (DoubleLinkedList<Integer> list) {
        if (list.size() > 0) {
                int removed = list.remove();
                return removed + question3(list);
        }
        return 0;
}
```

4)

```
public static int question4 (int n) {
        if (n < 3)
                return 0;

        IDictionary<Integer, Integer> map = new ArrayDictionary<>();
        int count = 0;
        for (int i = 0; i < n * n; i++) {
                for (int j = i; j > 0; j--) {
                        map.put(i, j);
                        count++;
                }
        }

        return question4(n / 3) + 3 * count * question4(n / 3);
}
```

Part 2: Are You A Master Who Mastered Master?
For each of the recurrences below, use the Master Theorem to find the big-θ of the closed form (show *a*, *b* and *c*) or explain why you can't apply the Master Theorem.

1) CSE 373 19sp

$$T(n) = \begin{cases} 4, & n = 1 \\ 8T(n/2) + n^2, & otherwise \end{cases}$$

2) CSE 373 13wi

$$T(n) = \begin{cases} 10, & otherwise \\ T(n/2) + 3, & when\, n > 1 \end{cases}$$

3)

$$T(n) = \begin{cases} n^2, & n < 10 \\ n^2T(n/2) + n^2, & otherwise \end{cases}$$

4)

$$T(n) = \begin{cases} C_1, & n = 1 \\ T(n/2) + C_2e^n, & otherwise \end{cases}$$

5)

$$T(n) = \begin{cases} 1, & n \leq 23 \\ T(n - 2) + 2n, & otherwise \end{cases}$$

Part 3: Form 1040 -- U.S. Individual Income Tax Return (CSE 373 19sp)
Consider the same recurrence from Part 2 1):

$$T(n) = \begin{cases} 4, & n = 1 \\ 8T(n/2) + n^2, & otherwise \end{cases}$$

We again attempt to find the closed form, this time by applying the tree method. If you wish to draw the tree, there is space on the back of this sheet.

1	What is the size of the input at each level i? What is the amount of work done by each node at the i-th recursive level? As in class, we call the root level $i = 0$. This means at $i = 0$, your expression for the input should equal n.	
2	What is the total number of nodes at level i? As in class, we call the root level $i = 0$. This means at $i = 0$, your expression for the total number of nodes should equal 1.	
3	What is the total work done across the i-th recursive level? Use information from lines 1 and 2.	
4	What is the value of i does the last level (base case) of the tree occur at?	
5	What is the total work done across the base case level of the tree (i.e. the last level)? Use information from the recurrence and parts 2 and 4.	
6	Combine your answers from previous parts to get an expression for the total work (which you will simplify in part 7).	
7	Simplify the expression from part 6 to a closed form (to find the total work done by T(n)).	
8	From your answer to part 7, give a simplified tight Big-O of the runtime. Compare your answer to Part 2 – 1). Are they the same?	

The recurrence is restated for your convenience:

$$T(n) = \begin{cases} 4, & n = 1 \\ 8T(n/2) + n^2, & otherwise \end{cases}$$

Part 4: Smells Like CSE 143 (CSE 373 18wi Practice)
In this problem, we will consider an algorithm named `isBalanced(String str)` that returns "true" if the input string has a "balanced" number of parenthesis and false otherwise. We say a string has "balanced" parenthesis if each opening paren is paired with a matching closing one. Here are some examples:

((a)b)	Balanced
(x)(y)(z)	Balanced
((((	Unbalanced
)))z(	Unbalanced

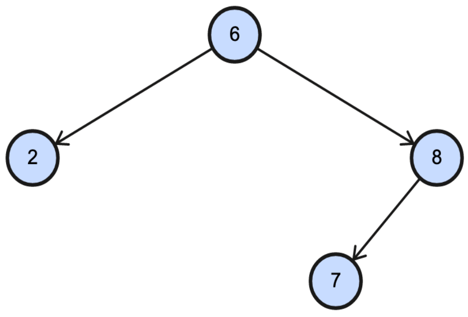
1) List at least four distinct <u>kinds</u> of inputs you would try passing into the `isBalanced` algorithm to test it. For each input, also list the expected outcome (assuming the algorithm was implemented correctly). Be sure to think about different edge cases.

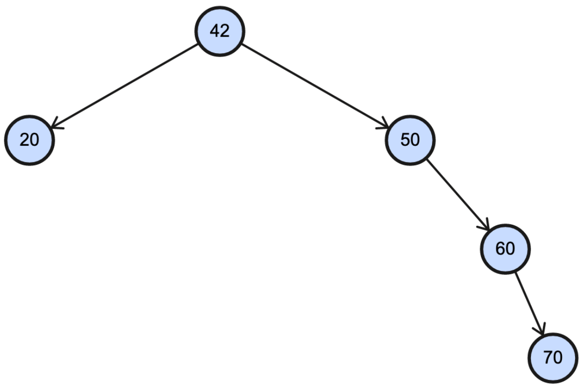
2) Here is one (buggy) implementation of this algorithm in Java. List every bug you can find.

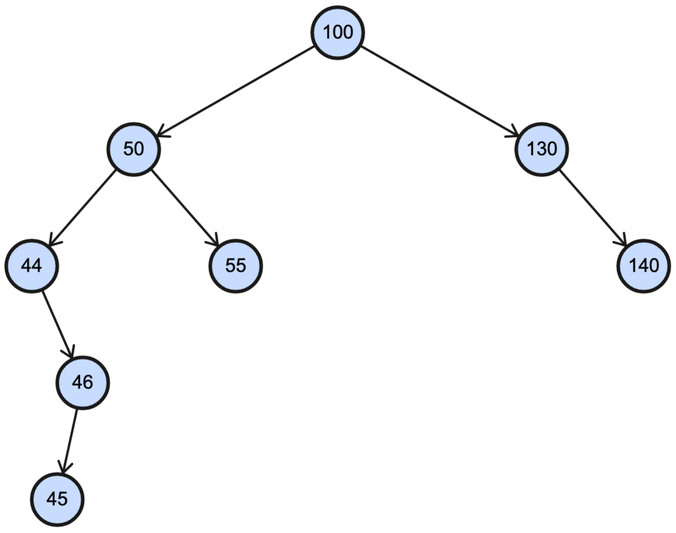
```
public static boolean isBalanced(String str) {
       if (str == null || str.size() == 0) {
              return false;
       }

       int numUnmatedOpenParens = 0;
       for (char c : str) {
              if (c == '(') {
                     numUnmatedOpenParens++;
              }
              } else {
                     numUnmatedOpenParens--;
              }
       }
       return numUnmatedOpenParens == 0;
}
```

Part 5: Oh AVL Tree!
Three Binary Search Trees are given to you below, but they are not necessarily AVL trees. Determine if the BSTs are also valid AVL trees. If not, point out <u>all nodes that are unbalanced</u> per the definition of an AVL tree.







J. Melezinek, "Binary Search Tree," BinaryTreeVisualiser. [Online]. Available: http://btv.melezinek.cz/binary-search-tree.html.