Exercise 2: Summation, Modeling Recursive Code, The Tree Method, Testing/Debugging

Due date: Friday July 19, 2019 at 11:59 pm

Instructions:

Submit a typed or neatly handwritten scan of your responses to the "Exercise 2" assignment on Gradescope here: https://www.gradescope.com/courses/52323. Make sure to log in to your Gradescope account using your UW email to access our course.

For more details on how to submit, see
https://courses.cs.washington.edu/courses/cse401/18au/hw/submitting_hw_guide.pdf.

These problems are meant to be done **individually**. If you do want to discuss problems with a partner or group, make sure that you're writing your answers individually later on. Check our course's collaboration policy if you have questions.

1. Simplifying expressions

We will simplify the following summation to produce a closed form. Show your work in all parts, clearly stating when you apply each summation identity. We will walk you through each step of this process.

$$\sum_{i=0}^{n-1}\left(\sum_{j=0}^{i-1} j + \sum_{j=0}^{n^3-1} 3i\right)$$

(a) First, derive the closed form of the following summation on its **own**:

$$\sum_{j=0}^{i-1} j$$

Hint: Note that this summation iterates over j. This means any value not in terms of j can be considered a constant.

(b) Next, derive the closed form of the following summation on its **own**:

$$\sum_{j=0}^{n^3-1} 3i$$

Hint: Note that this summation iterates over j. This means any value not in terms of j can be considered a constant.

(c) Great! Now let's call the closed-form expression you got in Part a) A, and the one you got in Part b) B. Simplify the following summation to its closed form.

$$\sum_{i=0}^{n-1} (A+B)$$

Hint 1: Hey! This is the thing we said we wanted to compute at the very beginning of the problem! (Can you see why?)
Hint 2: This time, the summation iterates over i. This means any value not in terms of i can be considered a constant.

2. Modeling recursive code

For the following problems, Give a recurrence formula for the running time of this code. Do not worry about finding the exact constants for the non-recursive term (for example if the running time is A(n) = 4A(n/3) + 25n, you need to get the 4 and the 3 right but you don't have to worry about getting the 25 right).

(a) Write a recurrence representing the runtime of this method, `funFib`, in terms of `n`.

```java
public static void funFib(int n) {
    if (n <= 1) {
        return n;
    } else {
        return funFib(n - 1) + funFib(n - 2);
    }
}
```

(b) In this part, we will write a recurrence representing the runtime of this public method, `mystery`.

```java
    public static void mystery(int[] data) {
        int n = data.length;
        if (n < 2) {
            return;
        }
        int mid = n / 2;
        int[] l = new int[mid];
        int[] r = new int[n - mid];

        for (int i = 0; i < mid; i++) {
            l[i] = data[i];
        }

        for (int i = mid; i < n; i++) {
            r[i - mid] = data[i];
        }
        mystery(l);
        mystery(r);

        converge(data, l, r, mid, n - mid);
    }

    public static void converge(int[] data, int[] l, int[] r, int left, int right) {
        int i = 0, j = 0, k = 0;
        while (i < left && j < right) {
            if (l[i] <= r[j]) {
                data[k++] = l[i++];
            }
            else {
                data[k++] = r[j++];
            }
        }
        while (i < left) {
            data[k++] = l[i++];
        }
        while (j < right) {
            data[k++] = r[j++];
        }
    }
```

(i) Let m be `l.length + r.length`. What is the big-Θ running time of `converge` in terms of m?

(ii) Let n be `data.length`. What is the big-Θ running time of lines 6-16 (i.e. the non-recursive work when $n > 2$ excluding the call to `converge`)?

(iii) Now write a recurrence for the running time of mystery in terms of n, where n is `data.length`.

3. The tree method

Consider the following recurrence:

$$A(n) = \begin{cases} 5 & \text{if } n = 1 \\ 4A(n/2) + n^3 & \text{otherwise} \end{cases}$$

We want to find an *exact* closed form of this equation by using the tree method. Note that in all parts requiring work, you **must** show your work to receive any credit.

(a) Draw your recurrence tree. Your drawing must include the **top three levels of the tree**, as well as a portion of the final level. It should be in the same style as the Section 3 slides Tree (link: `https://tinyurl.com/y22b83zd`). In particular, you must write both the **work** and **input size** for each node (if you draw nodes too small to fit both of these labels inside, you may put the labels nearby, but it must be clear what the label is for each node). You must also **label** i for each level except the last one.

(b) What is the size of the **input** to each node at 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 that at $i = 0$, your expression for the input should equal n.

(c) What is the total number of nodes at level i? As in class, we call the root level $i = 0$. This means that at $i = 0$, your expression for the total number of nodes should equal 1.

(d) What is the total work done across the i-th *recursive* level? Be sure to show your work.

(e) What value of i does the last level of the tree occur at? Be sure to show your work.

(f) What is the total work done across the base case level of the tree (i.e. the last level)? Be sure to show your work.

(g) Combine your answers from previous parts to get an expression for the total work. Simplify the expression to a closed form. Be sure to show your work.

(h) From the closed form you computed above, give a tight-Θ bound. No need to prove it. (Hint: You may need to simplify the closed form.)

4. Testing and Debugging

Bob's manager asks him to write a method to validate a binary search tree (BST). Bob writes the following `isBST` method in the `IntTree` class to check if a given `IntTree` is a valid BST. But this method has at least one fundamental bug. Answer the following questions and help Bob find the bug in his method.

```java
public class IntTree {
    private IntTreeNode overallRoot;

    // constructors and other methods omitted for clarity

    private class IntTreeNode {
        public int data;
        public IntTreeNode left;
        public IntTreeNode right;

        // constructors omitted for clarity
    }
    public boolean isBST() {
        return isBST(overallRoot);
    }
    private boolean isBST(IntTreeNode root) {
        if (root == null) {
            return true;
        } else if (root.left != null && root.left.data >= root.data) {
            return false;
        } else if (root.right != null && root.right.data <= root.data) {
            return false;
        } else {
            return isBST(root.left) && isBST(root.right);
        }
    }
}
```

(a) To prove that his method is correct, Bob gives you the following IntTree (Figure 1) to test his `isBST` method. The `isBST` method recursively calls `isBST` on each node.In the dotted box next to each node in the tree (Figure 1) write the output of `isBST` ("True" or "False") when it is called on that node; write "N/A" in the box if `isBST` is never called on the node. Beware of Java Boolean short-circuiting in this problem. If you are not familiar with short-circuiting, here is a quick overview: https://stackoverflow.com/a/8759917/8842326

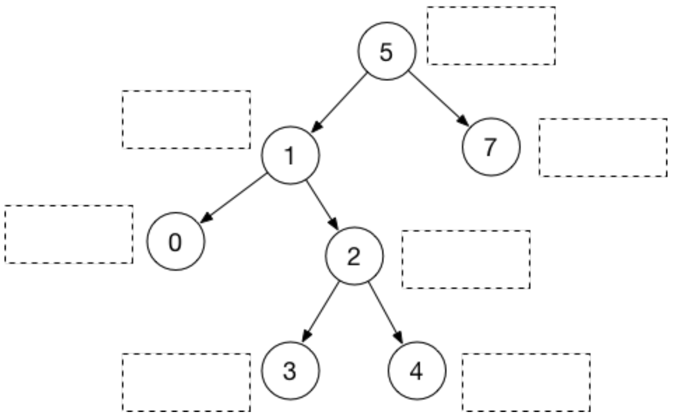

Figure 1: See 7a.

(b) To convince Bob that his method is incorrect, find a test example of an invalid BST for which Bob's `isBST` method would return true, failing to detect an invalid BST. **Your input must be a binary tree with each key being a distinct integer (e.g. you could not give a tree where the root has three children, or a tree that has a `null` data field)**. Draw your invalid BST test example.

(c) As an awesome CSE373 student you have realized that in order to fix Bob's `isBST` method, your fix will require modifying the parameters passed into `isBST`. How would you modify the method header, and outline how you could use the different set of parameters to fix Bob's `isBST` method bug? Explain in at most 3 - 4 sentences.