Homework 4: AVL trees, testing and analysis

Due date: Friday February 15, 2019 at 11:59 pm

Instructions:

Submit a typed or neatly handwritten scan of your responses on Gradescope here: https://www.gradescope.com/courses/41121 (the assignment is "HW 4"). You will log in with your UW NetID.

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. AVL tree rotations

Draw the AVL tree that results from inserting the keys: 3, 1, 4, 5, 9, 2, 6, 7, 8 in that order into an initially empty AVL tree. You are only required to show the final tree, although drawing intermediate trees may result in partial credit. If you draw intermediate trees, **please circle your final tree for any credit**.

2. Simplifying expressions

(a) Simplify the following summation to produce a closed form. Show your work, clearly stating when you apply each summation identity.

$$\sum_{i=0}^{n-1}\left(\sum_{j=0}^{i-1} j + \sum_{j=0}^{n^2-1} 5i\right)$$

(b) Convert the following recurrence into a summation by applying the unrolling technique discussed in lecture. Then, simplify your summation to find a closed form. You may assume that the initial input n is always > 7.

$$E(n) = \begin{cases} 4 & \text{When } n \leq 7 \\ E(n-1) + n & \text{Otherwise} \end{cases}$$

3. Asymptotic analysis: Mathematically

(a) Show that $6n + n\log(n) \in \Omega(10\log(n))$ is true by finding a c and n_0 that satisfy the definition of "dominates" and big-Ω. Please show your work.

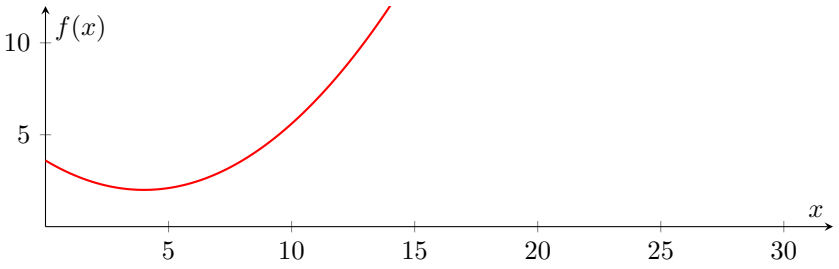
(b) Show that $\log_3(n) \in \mathcal{O}(\log_5(n))$ by finding a c and n_0 that satisfy the definition of "dominated by" and big-$\mathcal{O}$. Please show your work. As a hint, you will need to use the change-of-base logarithm identity somewhere.

4. Asymptotic analysis: Visually

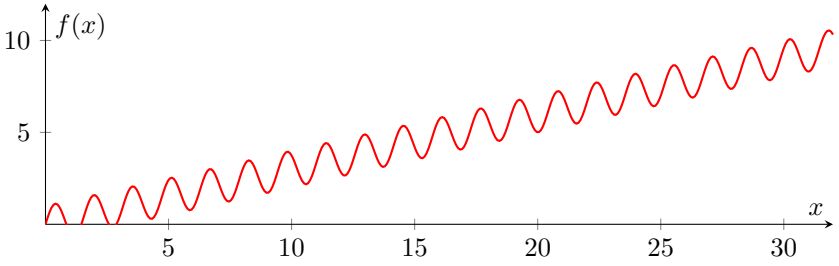
For each of the following plots, provide a tight big-$\mathcal{O}$ bound, a tight big-Ω bound, and a big-Θ bound. You do not need to show your work; just list the bounds. If a particular bound doesn't exist for a given plot, briefly explain why. Assume that the plotted functions continue to follow the same trend shown in the plots as x increases. Each provided bound must either be a constant or a simple polynomial, **from the following possible answers**.

$$n^2, 1, n, log(n), n!, 1/n$$

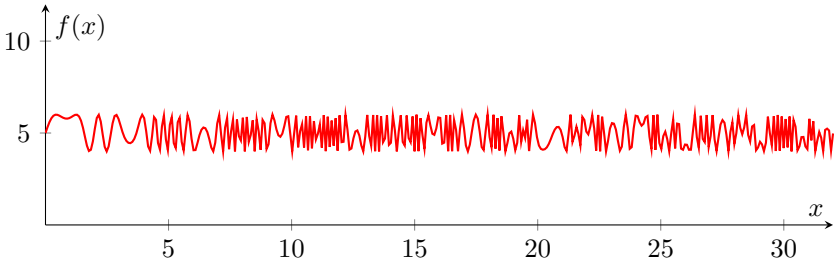
(a)



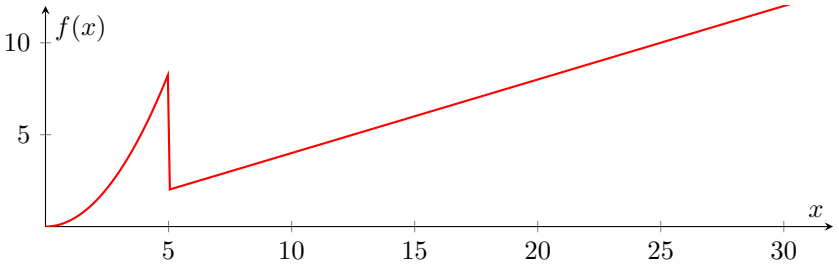
(b)



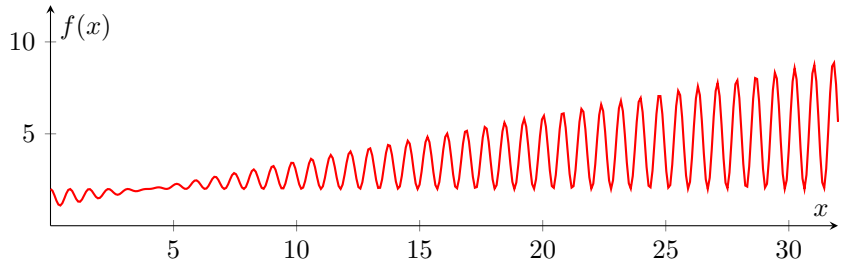
(c)



(d)



(e)



5. Modeling code

(a) Construct a mathematical function $T_1(n)$ modeling the approximate *worst-case runtime* of the `mystery1` method. Your answer should be written as a summation. You do not need to find the closed form of this summation. You may simplify all constants to stand in variables such as C1 or C2 (you do not need to attempt to count the exact number of operations)

```
public static int mystery1(int n) {
    int out = 0;
    for (int i = 0; i < n; i++) {
        if (i % 5 == 0) {
            for (int j = 0; j < i; j++) {
                out += 2;
            }
        }
    }
    return out;
}
```

(b) Construct a mathematical function $T_2(n)$ modeling the approximate *worst-case runtime* of the `mystery2` method. Your answer should be a recurrence. You do not need to find the closed form of this recurrence.

```
public static int mystery2(int n) {
    if (n == 0) {
        return 3;
    } else {
        return mystery2(n / 2) + n;
    }
}
```

6. The Tree Method

Consider the following recurrence:

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

We want to find an *exact* closed form of this equation by using the tree method.

(a) (2 points) Draw your recurrence tree (as shown in the class; see lecture 18, slide 19).

(b) (1 point) What is the size of the input at level i (as in class, call the root level 0)?

(c) (2 points) what is the number of nodes at level i?

Note: let $i = 0$ indicate the level corresponding to the root node. So, when $i = 0$, your expression should be equal to 1.

(d) (3 points) What is the total work at the i-th *recursive* level? Be sure to show your work for full credit.

(e) (1 point) What is the last level of the tree?

(f) (2 points) What is the work done in the base case?

(g) (3 points) 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 for full credit.

(h) (1 point) From the closed form you computed above, give a tight-$\mathcal{O}$ bound. No need to prove it. (Hint: You may need to simplify the closed form.)

7. Testing

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.

```
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.

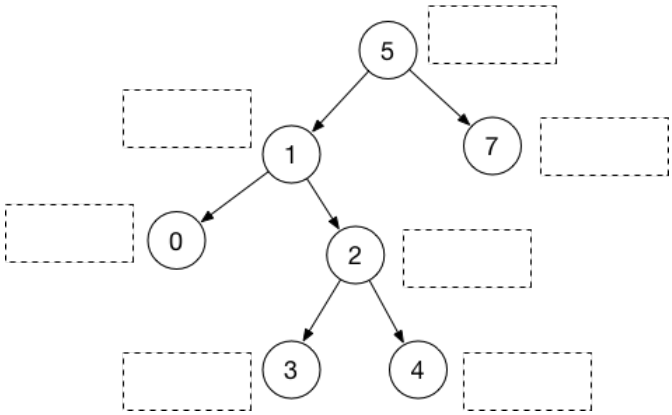


Figure 1: See 7a.

(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. Draw your invalid BST test example.

(c) Identify at least one bug in Bob's `isBST` method and explain it in one or two sentences.