Homework 03: AVL trees, algorithm design and analysis

Due date: October 26, 2018 at **noon** (12:00 pm)

Instructions:

Submit a typed or neatly handwritten scan of your responses on Canvas in PDF format.

Note: you will need to submit a separate PDF per each section.

1. AVL tree rotations

Submit your answers here: https://canvas.uw.edu/courses/1219673/assignments/4468675

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

Submit your answers here: https://canvas.uw.edu/courses/1219673/assignments/4468676

(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 unfolding 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

Submit your answers here: https://canvas.uw.edu/courses/1219673/assignments/4468677

(a) Show that $6n + n\log(n) \in \Omega\left(10\log(n)\right)$ 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 $\log_3(n) \in \mathcal{O}\left(\log_5(n)\right)$ 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.

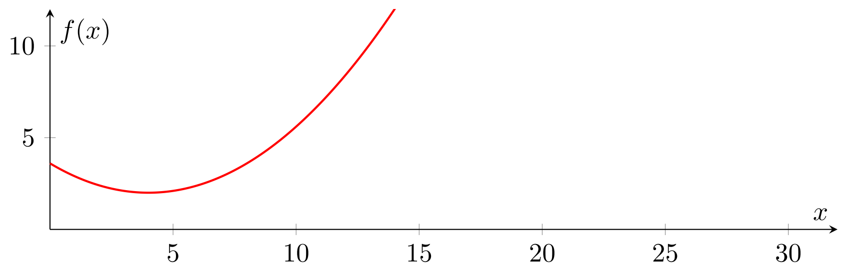
4. Asymptotic analysis: Visually

Submit your answers here: `https://canvas.uw.edu/courses/1219673/assignments/4468678`

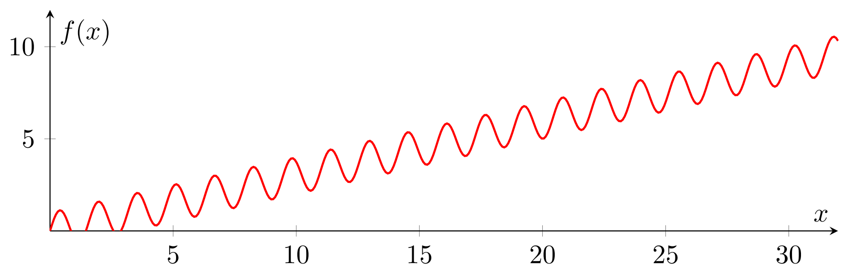
For each of the following plots, provide a tight big-$\mathcal{O}$ bound, a tight big-Ω bound, and a big-Θ bound. You do not need to show your work; just list the bounds. If a particular bound doesn't exist for a given plot, briefly explain why. Assume that the plotted functions continue to follow the 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.

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

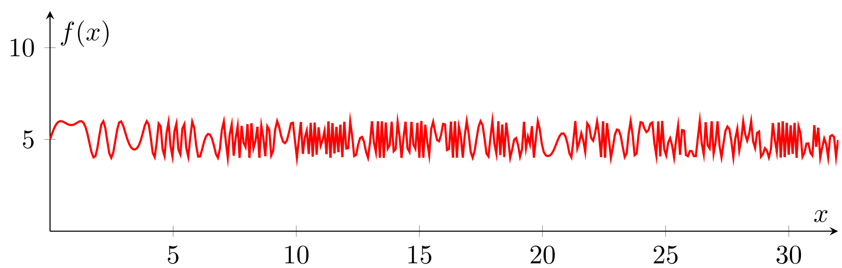
(a)



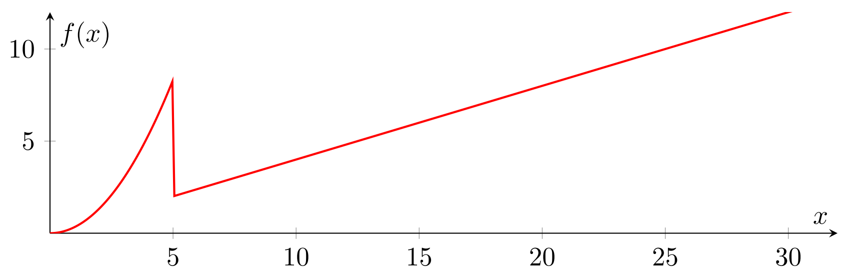
(b)



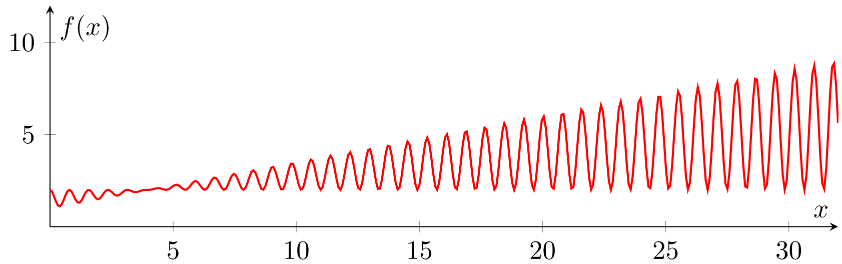
(c)



(d)



(e)



5. Modeling code

Submit your answers here: https://canvas.uw.edu/courses/1219673/assignments/4468679

(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.

```
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. Algorithm design: Merge

Submit your answers here: `https://canvas.uw.edu/courses/1219673/assignments/4468680`

Suppose we are given two sorted arrays containing comparable elements (such as integers or strings). Our goal is to design an algorithm that returns a new array containing all items from both arrays in sorted order. The input arrays should remain unmodified. The algorithm should throw an exception when given invalid input.

For example, suppose we receive as input the arrays `[-5, 0, 0, 2]` and `[-1, 2, 3]`. The output should be the array `[-5, -1, 0, 0, 2, 2, 3]`.

(a) Write an **English description** or **high-level pseudocode** describing an algorithm to perform this task. Note: do **NOT** submit Java code. We want to see a high-level description of the algorithm, not a low-level one.

Please see the following link for more details on what an acceptable response to this question should look like.
`https://courses.cs.washington.edu/courses/cse373/18au/resources/explaining-algorithms.html`

(b) Provide a tight big-Θ bound of the worst-case runtime of your algorithm. Write your answer in terms of n and m, where n is the length of the first input array and m is the length of the second. Briefly justify your answer.

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

The following link may be a good source of inspiration when coming up with test inputs:
`https://courses.cs.washington.edu/courses/cse373/18au/resources/tips-on-testing-code.html`

7. Testing

Submit your answer here: https://canvas.uw.edu/courses/1219673/assignments/4468681

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.

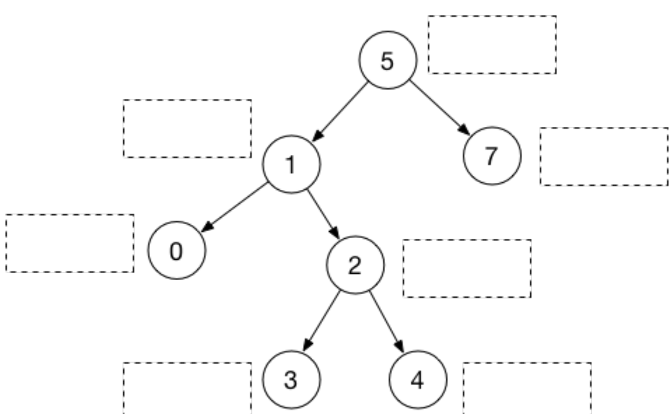


Figure 1: See 7a.

(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.