Homework 3: AVL trees, testing and analysis

Due date: Wednesday May 1, 2019 at 11:59 pm

Instructions:

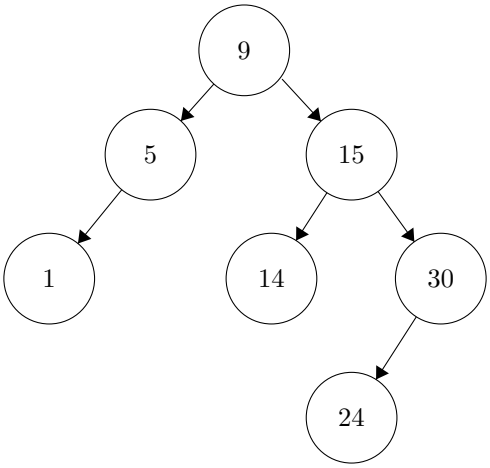
Submit a typed or neatly handwritten scan of your responses to the "HW 3" assignment on Gradescope here: https://www.gradescope.com/courses/47703. Make sure to log in to your Gradescope account using your UW email to access our course.

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. AVL trees

Consider the following AVL Tree:



(a) What value could you insert into this tree to cause no rotations? Show the tree after inserting this value.

(b) What value could you insert into this tree to cause a single rotation? Show the tree after inserting this value.

(c) What value could you insert into this tree to cause a double rotation? Show the tree after inserting this value.

(d) Draw the resulting tree after inserting the following values into the original tree: 0, 8, 51, 19. Make sure to label your final answer if you choose to draw intermediate trees.

2. 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.

$$\sum_{i=0}^{n-1}\left(\sum_{j=0}^{i-1} j + \sum_{j=0}^{n^2-1} 9i\right)$$

(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**:

$$\sum_{j=0}^{n^2-1} 9i$$

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.

3. Asymptotic analysis: Mathematically

(a) Show that $10\log_2(n)+15 \in \mathcal{O}\left(n\log_2(n)\right)$ is true by finding a c and n_0 that satisfy the definition of "dominated by" and big-$\mathcal{O}$. Please show your work.

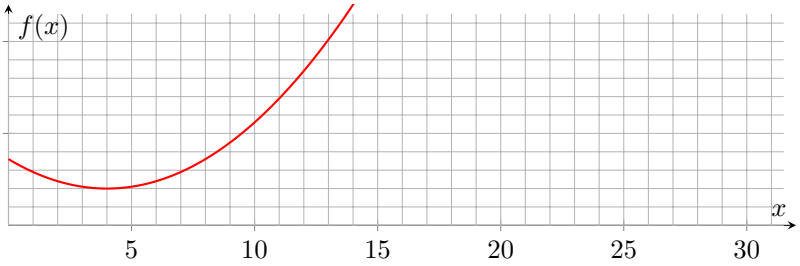
(b) Show that $\log_2(n) \in \mathcal{O}\left(\log_8(n)\right)$ by finding a c and n_0 that satisfy the definition of "dominated by" and big-$\mathcal{O}$. Please show your work. As a hint, you will need to use the change-of-base logarithm identity somewhere.

4. Asymptotic analysis: Visually

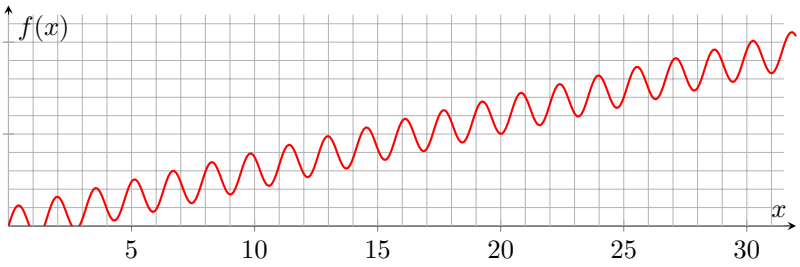
For each of the following plots, provide a tight big-$\mathcal{O}$ bound, a tight big-Ω bound, and a big-Θ bound. You do not need to show your work; just list the bounds. If a particular bound doesn't exist for a given plot, briefly explain why. Assume that the plotted functions continue to follow the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

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

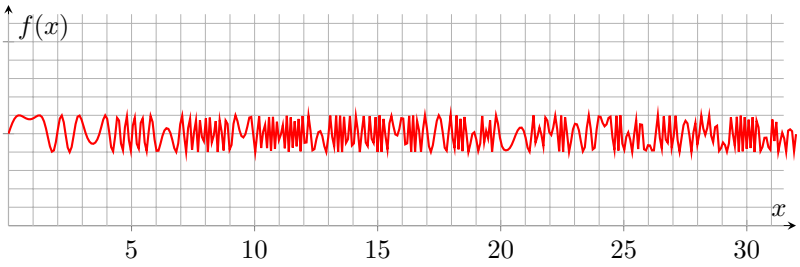
(a)



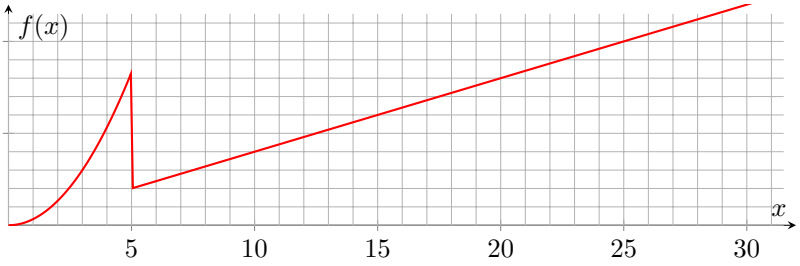
(b)



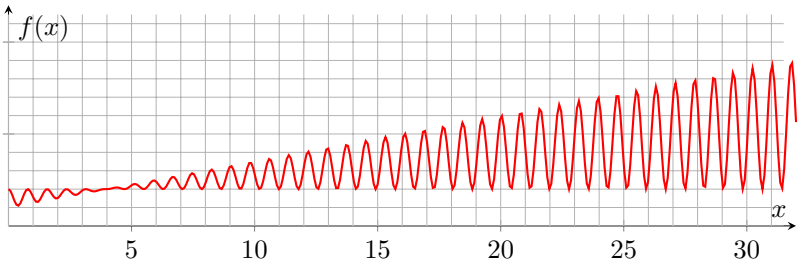
(c)



(d)



(e)



5. Modeling code

Consider the following code for the `processLetters` method:

```
1  public ArrayDictionary<Character, Integer> processLetters(DoubleLinkedList<String> input) {
2      ArrayDictionary<Character, Integer> result = new ArrayDictionary<Character, Integer>();
3      for (String s : input) {
4          for (int i = 0; i < s.length(); i++) {
5              char c = s.charAt(i);
6              if (result.containsKey(c)) {
7                  int count = results.get(c) + 1;
8                  result.put(c, count);
9              } else {
10                 result.put(c, 1);
11             }
12         }
13     }
14     return result;
15 }
```

Answer the following questions about the runtime of the `processLetters` method. Consider the parameter `input` to contain `n` `Strings`, each with `m` `chars`. Assume that there are an infinite number of possible `chars` in `Strings`. Since parts (a)–(e) ask about runtime code in loops, only consider the runtime on the last loop iteration.

Remember that "best/worst-case" refer to the inputs that yield the best or worst possible runtime, respectively.

(a) What is the simplified tight big-$\mathcal{O}$ bound for the worst-case runtime of line 6 on the final iteration of the inner loop? Your answer should be in terms of `n` and `m`.

For (b)–(d), assume that the line of code actually gets called and the dictionary size is $\texttt{n} \times \texttt{m} - 1$.

(b) What is the worst-case runtime of line 8 on the final iteration of the inner loop? Your answer should be in terms of `n` and `m` and should include their numerical coefficients (i.e., do not simplify their coefficients to stand in values of "c").

(c) In terms of `n` and `m` what is the best-case runtime of line 8 on the final iteration of the inner loop? Your answer should be in terms of `n` and `m` and should include their numerical coefficients.

(d) What is the average-case runtime of line 8 on the final iteration of the inner loop? Your answer should be in terms of `n` and `m` and should include their numerical coefficients. Average-case runtime is defined as the average runtime across all possible inputs. (Hint: consider all the possible positions for `c` in the `ArrayDictionary`.)

(e) What is the worst-case runtime of the loop between lines 4 and 12 on the final iteration of the outer loop? Your answer should be in terms of `m` and `n`, but you may simplify all constants to stand in values such as c_1.

(f) What is the simplified tight big-$\mathcal{O}$ bound for the worst-case runtime of `processLetters`?

6. Modeling recursive code

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

```
1  public static void fun(int n) {
2      if (n <= 1) {
3          System.out.print("hi");
4      } else if (n <= 100) {
5          for (int i = 0; i < n; i++) {
6              System.out.println("lol");
7          }
8      } else {
9          for (int i = 0; i < n; i++) {
10             for (int j = 0; j < 5; j++) {
11                 System.out.println("haha");
12             }
13         }
14         for (int i = 0; i < 3; i++) {
15             fun(n - 3);
16         }
17     }
18 }
```

(b) Write a recurrence representing the runtime of the private function in terms of `sizeLeft`. The public method is provided for context.

```
1  /*
2   * Prints every string of length 'resultSize' composed of 0 or more
3   * a's, b's, c's, and d's.
4   *
5   * Note: If you took CSE 143 here, you might recognize that this is some
6   * recursive code that does "recursive backtracking". Recursive backtracking
7   * often has a corresponding decision tree that map out all possible states -
8   * these decision trees and their branching factors fit nicely into the
9   * framework of tree method analysis and modeling code as recurrences!
10  */
11 public static void printABCDStrings(int resultSize) {
12     printABCDStrings("", resultSize);
13 }
14
15 private static void printABCDStrings(String soFar, int sizeLeft) {
16     if (sizeLeft == 0) {
17         System.out.println(soFar);
18     } else {
19         printABCDStrings(soFar + "a", sizeLeft - 1);
20         printABCDStrings(soFar + "b", sizeLeft - 1);
21         printABCDStrings(soFar + "c", sizeLeft - 1);
22         printABCDStrings(soFar + "d", sizeLeft - 1);
23     }
24 }
```

(c) Write a recurrence representing the runtime of this private function. The public method is provided for context.

Usually it's more clear what the parameters for the recurrence are, but this method has many parameters, and the main input `data` doesn't decrease in size on each recursive call.

If you take a look at the base case and recursive call inputs, you'll see that binary search recursively narrows down the remaining region to search in rather than decreasing the size of the input.

You might find it difficult to write the recurrence in terms of `low` and `high` directly. Instead, you should write the recurrence in terms of a single parameter: the size of the remaining region to search through—call this m. Then, ask yourself how m decreases with each call, and pass that forward in your recurrence.

```
1  /*
2   * Here's some (modified) code for binary search - a standard example for code
3   * that runs in log(n) time worst case!
4   * Implementing binary search (recursively and/or iteratively) is a common
5   * interview question!
6   */
7  public static void funBinarySearch(int[] data, int target) {
8      funBinarySearch(data, 0, data.length - 1, target);
9  }
10
11 /*
12  * The first call to this private helper method should search through the entire
13  * array, since 0 to data.length - 1 is the full range of remaining values to
14  * search through. The recursive calls will decrease the size of the range being
15  * searched by changing 'low' and 'high'.
16  */
17 private static void funBinarySearch(int[] data, int low, int high, int target) {
18     if (low > high) {
19         System.out.println("Done.");
20     } else {
21         int mid = (low + high) / 2;
22         if (data[mid] == target)  {
23             System.out.println("Found it at index " + mid +"!");
24         }
25         if (data[mid] < target) {
26             funBinarySearch(data, mid + 1, high, target);
27         } else { // data[mid] >= target
28             funBinarySearch(data, low, mid - 1, target);
29         }
30     }
31 }
```

7. 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, 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, Problem 5f solutions (available from the course website). In particular, you must write both the **input** and the **work** done inside 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 for full credit.

(e) What value of i does the last level of the tree occur at?

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

(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 for full credit.

(h) 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.)

8. 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.

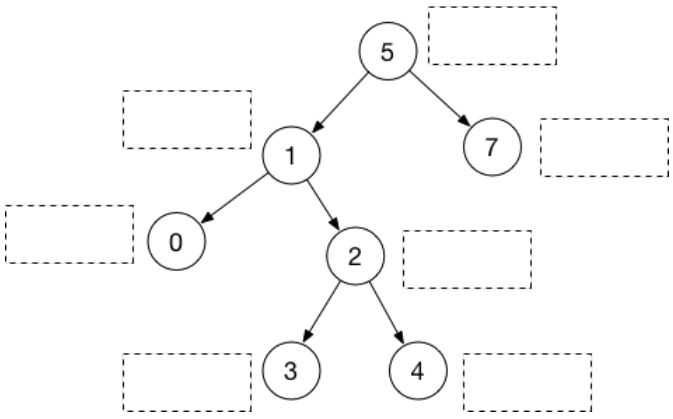


Figure 1: See 7a.

(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.

9. Design decisions

Imagine you've been tasked with writing a program to process all the files in your Downloads folder. Who knows what is important to store or what you'll be using it for - it's your own Downloads folder, after all. Which ADT and data structure implementation would help you represent and solve this problem the best? These types of design decision questions are usually open ended and multiple correct answers exist, given reasonable justification. To answer this general question, you must explicitly answer each of the following sub-questions to form your answer and receive credit.

Note: For this problem, assume that all your files in your Downloads folders are not more folders. If you did have folders within folders within folders and so on ... you would probably consider some type of recursive representation like a tree.

(a) Out of a List, Stack, Queue, or Dictionary, which ADT do you choose and why does the ADT fit? Specify one use case for your Downloads folder and what method(s) of the ADT are involved to perform the use case.

(b) Assume runtime is the most important concern in this implementation. Our two main data structure implementations are an array-based structure or a node-based structure, like a linked list. Which data structure implementation do you choose and why does the implementation fit? Specify 1 operation (a specific method call or the use case from above, which might be multiple method calls) that this implementation is optimized for, and state what the runtime is in terms of a simplified tight big-$\mathcal{O}$.

(c) Now choose a **different** ADT and the data structure implementation from the options provided to represent and solve this same problem.

(i) Which different ADT do you choose and why does the ADT fit? Specify a different use case for the Downloads folder than earlier, and what method(s) of the ADT that performing this use case involves. For this use case, explain whether or not it makes sense to use the ADT you selected in part a.

(ii) Now consider the data structure implementation that you did not choose in part b. Explain 1 reason that you would choose this implementation instead of the one chosen in part A, for this new ADT. You may explain for a general use case of the ADT instead of the specific use case of the Downloads folder mentioned above.