EX2: Algorithmic Analysis II (Recursive Analysis)

Due date: Friday July 17, 2020 at 11:59 pm PDT

Instructions:

Submit your responses to the "EX2: Algorithmic Analysis II" assignment on Gradescope here: `https://www.gradescope.com/courses/141341/assignments/556707`. Make sure to log in to your Gradescope account using your UW email to access our course.

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. Modeling recursive code

Write a recurrence representing the runtime of the private method `printABCDStrings(String soFar, int sizeLeft)` in terms of `sizeLeft`. Assume that any + operations (String concatenation included) and `System.out.println()` calls take constant runtime. The public method is provided for context. Don't 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.

```java
/*
* Prints every string of length 'resultSize' composed of 0 or more
* a's, b's, c's, and d's.
*
* Note: If you took CSE 143 here, you might recognize that this is some
* recursive code that does "recursive backtracking". Recursive backtracking
* often has a corresponding decision tree that map out all possible states -
* these decision trees and their branching factors fit nicely into the
* framework of tree method analysis and modeling code as recurrences!
*/
public static void printABCDStrings(int resultSize) {
    printABCDStrings("", resultSize);
}

private static void printABCDStrings(String soFar, int sizeLeft) {
    if (sizeLeft == 0) {
        System.out.println(soFar);
    } else {
        printABCDStrings(soFar + "a", sizeLeft - 1);
        printABCDStrings(soFar + "b", sizeLeft - 1);
        printABCDStrings(soFar + "c", sizeLeft - 1);
        printABCDStrings(soFar + "d", sizeLeft - 1);
    }
}
```

2. 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* form of this equation by using the tree method.

(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 include both the **work** and **input size** for each node, in some clearly recognizable format (for example, you can base it off the tree on slide 24 of the LEC 07 slides: https://courses.cs.washington.edu/courses/cse373/20su/lectures/7.pdf). If you draw nodes too small to fit both of these labels inside, you can put the labels nearby, but make it clear what the label is for each node. **Label** i for each level except the last one. Don't worry about explicitly drawing all the nodes as the levels increase; just showing the general pattern is good enough, as this portion is mostly for your own understanding.

For this drawing portion, upload a pdf or image of your work. You can either scan or photograph a handwritten drawing or use an online drawing site like `https://awwapp.com/` and export your drawing to pdf or an image format.

(b) What is the size of the **input** to each node at level i? 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 amount of **work** done by a single node at the i-th recursive level?

(d) 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.

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

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

(g) 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.

(h) Combine your answers from previous parts to get an expression for the total work. You do not have to simplify the summation, just combine your previous answers up to this point into one expression.

Since this will require a summation in gradescope, you can copy paste the following: `$$\sum_{i=0}^{n - 1} i$$`. This will convert to

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

in Gradescope, and you can change the bounds or the inner expression as necessary.

3. Design Decisions

This is an open-ended question intended to get you thinking about tradeoffs in your code. There are many correct answers, as long as you back up your ideas with justification based on the underlying data structures. We generally aren't concerned with fine-grained details of the code; this is intended to be a high-level design.

For this problem, suppose that Zoom has hired you to build the audio transcript feature of their cloud recordings platform. In this platform, each caption (stored as a string) is associated with a timestamp (stored as an integer, representing the number of milliseconds since the start of the video). Together, these pieces of information are used to power a transcript of the video that shows what is currently being said alongside the video as it plays.

You are tasked with creating a high-level design for the system that will store and process these captions. We will represent this system as being a `CaptionManager` class, which stores the captions internally using some data structure and supports two different operations:

- **`nextCaption()`:** when this method is called, the `CaptionManager` should return two pieces of information (don't worry too much about how that would be represented in code): the current caption, and the number of milliseconds to wait between this and the next caption (0 if this is the last caption). Calling this method multiple times should return a new caption each time, in order from start to end (which means each call will update the state of the `CaptionManager`). You might imagine this method being used as the video player plays the video normally.
- **`getCaption(int timestamp)`:** when this method is called with an int parameter representing a timestamp in milliseconds, it should return the caption for that timestamp, or null if no such caption exists.

In this question, you will consider two different designs. For each design, you should have a single data structure that stores all of the captions, but you may have additional fields/state if you want. You can pick from the ADTs we've covered in class (List, Stack, Queue, Map), and the corresponding data structure implementations we've discussed.

(a) Design A: Suppose we want to optimize for the runtime of `nextCaption()` in the worst case. In 2-3 sentences, describe one way you could store the captions to make that runtime as fast as possible. Specifically, you should mention an ADT, a correponding data structure, and a high-level description of how the `nextCaption()` method would operate on it. Then give a simplified Θ bound for the worst case runtime of `nextCaption()` in your proposed solution.

(b) Using the same ADT and data structure from Design A, now describe how it would affect the worst case runtime of `getCaption(int timestamp)`. Specifically, you should give a high-level description of how the `getCaption(int timestamp)` method would operate on the data structure. Then give a simplified Θ bound for the worst case runtime of `getCaption(int timestamp)` in your proposed solution.

(c) Design B: Now, suppose we want to optimize for the runtime of `getCaption(int timestamp)` in the worst case. In 2-3 sentences, describe one way you could store the captions to make that runtime as fast as possible. Specifically, you should mention an ADT, a correponding data structure, and a high-level description of how the `getCaption(int timestamp)` method would operate on it. Then give a simplified Θ bound for the worst case runtime of `getCaption(int timestamp)` in your proposed solution.

(d) Using the same ADT and data structure from Design B, now describe how it would affect the worst case runtime of `nextCaption()`. Specifically, you should give a high-level description of how the `nextCaption()` method would operate on the data structure. Then give a simplified Θ bound for the worst case runtime of `nextCaption()` in your proposed solution. *Note: a previous version of this problem mistakenly asked for the theta bound of* `getCaption`*. We will accept either answer while grading.*

(e) Identify at least one property of the stored captions that would be different between the best case and worst case for the `getCaption(int timestamp)` runtime in your proposed Design B implementation. If your proposed Design B implementation doesn't have a different best and worst case, describe why.