EX2: Modeling Recursive Code & Design Decisions

Due date: Friday April 22, 2022 at 11:59 pm PDT

Latest turn-in date: Monday April 25th, 2022 at 11:59 pm PDT

Instructions:

High-level collaboration is allowed, but exercises are to be completed and submitted individually. Submit your responses to the "EX2: Modeling Recursive Code & Design Decisions" assignment on Gradescope here: https://www.gradescope.com/courses/379339/assignments/2000391/.
Make sure to log in to your Gradescope account using your UW email to access our course.

1. Modeling Recursive Code

For (a) and (b), write recurrences representing the runtimes of each method in terms of `n`. Assume that any +, -, *, /, % operations and `System.out.println()` calls take constant runtime. Don't worry about finding the exact constants for the non-recursive term. For example, if the running time is T(n) = 4T(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 the recurrence for `mystery(n, 1)`, where the intial value of the parameter `step = 1`.

```
1  /*
2  * A mystery method
3  */
4  public void mystery(int n, int step) {
5      if (n <= step) {
6          return;
7      }
8
9      for (int i = 0; i < n; i += step) {
10         int a = i + n / 2;
11         System.out.print(a * a + " ");
12     }
13     System.out.println();
14     mystery(n, step * 2);
15     mystery(n, step * 2);
16 }
```

(b) Write a recurrence for `mystery2(n)`.

```java
/*
* Another mystery method
*/
public int mystery2(int n) {
    if (n < 10000) {
        int result = 1;
        for (int i = 0; i < n; i++) {
            result *= 2;
        }
        return result;
    } else if (n % 2 != 0) {
        int result = 0;
        for (int i = 0; i < n * n; i++) {
            result += i;
        }
        System.out.println(result);
        return 5 * mystery2(n / 2);
    } else {
        int j = 2 ^ n;          // This is an exponent
        while (j > 100) {
            if (n % 2 != 3) {
                j -= 1;
            } else {
                j = j / 2;
            }
        }
        return mystery2(n - 2) + j;
    }
}
```

2. Master Theorem

For recurrences in this form, where a, b, c, e are constants:

$$T(n) = \begin{cases} d & \text{if } n \leq \text{some constant} \\ aT(n/b) + e \cdot n^c & \text{otherwise} \end{cases}$$

$$T(n) \text{ is } \begin{cases} \Theta(n^c) & \text{if } \log_b(a) < c \\ \Theta(n^c \log n) & \text{if } \log_b(a) = c \\ \Theta\left(n^{\log_b(a)}\right) & \text{if } \log_b(a) > c \end{cases}$$

For each of your recurrences in (a) and (b), use the Master Theorem to find the big-Θ of the closed form or explain why Master Theorem doesn't apply.

(a) `mystery`

(b) `mystery2`

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 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, the caption and timestamp create the transcript that plays alongside the video.

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 operations:

- **`nextCaption()`:**
  - **Return:** the current caption and the number of milliseconds to wait between this caption and the next caption (the last caption will have 0 milliseconds cause it's the last caption) (don't worry too much about how that would be represented in code).
  - ***Note:*** Calling this method multiple times should return a new caption each time (which means each call will update the state of the `CaptionManager`). You might imagine this method being used to sync the transcript with the video as the video player plays the video.
- **`getCaption(int timestamp)`:**
  - **Parameter:** int parameter representing a timestamp in milliseconds.
  - **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.

Please keep your answer to each question **in 2-3 sentences**.

(a) Design A: Suppose we want to optimize for the runtime of `nextCaption()` in the worst case.

(i) Describe one way you could store the captions to make that runtime as fast as possible. Specifically, you should give:

- an ADT
- a corresponding data structure, and a high-level description of how the `nextCaption()` method would operate on it
- a simplified Θ bound for the worst case runtime of `nextCaption()` in your proposed solution

(ii) 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
- a simplified Θ bound for the worst case runtime of `getCaption(int timestamp)` in your proposed solution

(b) Design B: Now, suppose we want to optimize for the runtime of `getCaption(int timestamp)` in the worst case.

(i) Describe one way you could store the captions to make that runtime as fast as possible. Specifically, you should give:

- an ADT
- a correponding data structure, and a high-level description of how the `getCaption(int timestamp)` method would operate on it
- a simplified Θ bound for the worst case runtime of `getCaption(int timestamp)` in your proposed solution

(ii) 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
- a simplified Θ bound for the worst case runtime of `nextCaption()` in your proposed solution

(iii) 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.