Matthew Cheng

Assignment 1 Writeup

Next-date

Your program should be as efficient as possible regarding execution time and memory usage. Create a PDF document in which you describe the strategy that you used in your algorithms to ensure efficiency (execution time and memory). Submit this document via Blackboard by the deadline. For example, you could use specific data types (i.e., with modifiers) to minimize memory usage, or use pointers/references to avoid copying variables in function calls, or reduce the amount of program instructions, etc.

In this program, I made several efforts to ensure that I have good execution time. For example, after each check in my validateInput function, I have a separate DateException throw in order to prevent any unnecessary code from running. If an invalid part of the date is found at any point during the execution of the function, it will terminate and then the exception will be caught. I then included a return 0 after the error message is displayed to make sure that the function stops running. If I didn't do that, it would still run like normal even after the error message was displayed.

In addition, I created a leap function that was declared in the header file, as I needed to use it in both .cpp files. This way I prevent reusing code too much. In this function, I also minimize the amount of checks that need to be made. I can accomplish this by performing the leap year checks in reverse (as in 400 first, then 100, then 4), so that no "or" statements need to be made.

When declaring and initializing my SimpleDate object, dateobj, I used the memory address of date instead of the date variable itself in order to save memory.

Lastly, throughout the rest of my program, I make sure that I only perform an operation the minimum required times. For example, when I parse the date from the command line and then create the SimpleDate object, the date is broken up into month, day, and year and then both the string and integer versions of the numbers are stored as attributes. This benefits execution time, as all six variables are required for toString and nextDate. This prevents me from needing to convert the date variables back and forth between strings and integers. In my if else blocks, I tried to place my longer "or" statements towards the end in order to minimize the potential amount of checks that would need to be made.

Figures

Your program should be as efficient as possible regarding execution time and memory usage. In the same PDF document from the previous program:

1. Describe the strategy that you used in your algorithms to ensure efficiency (execution time and memory).

In this project, I had four classes: FigurePrinter and the three actual figures that inherit from it. DiamondPrinter inherits printDownwardTriangle from TrianglePrinter as well. Dynamic polymorphism was used here in order to increase efficiency so that I don't need to create separate functions for each class. Instead, I can override previous functions, such as the printFigure function.

In order to take advantage of dynamic polymorphism, I used a FigurePrinter pointer, called printer, in my main function. That way, I only needed to declare the one pointer, instead of three variables, to print all three figures. After using the pointer, I would delete it from memory, as there is no garbage collection in C++, and then I would redefine it for a different figure.

Lastly, I passed the same variable n into each method using local variables instead of declaring a new attribute for each class. That way I would take advantage of the stack, which is used for variables with a limited lifetime, instead of the heap.

2. Estimate the runtime complexity/efficiency of your whole program as a function of **n**, using Big-O notation (e.g., $O(n)$). Provide a brief explanation/justification of the process you followed to derive this function. For example, if your answer is that your program is $O(n)$, i.e., its efficiency is linear with respect to the input **n**, then explain how you analyzed your code and reached that conclusion.

My main function includes declarations and initializations of the variables temp, n, and printer. It also has output lines, deletion lines, and a return line. All of these are independent of n and are therefore O(1), or constant time. For example, each declaration and the return would consist of one task. The output statements and printer initializations would make up three tasks.

The only lines that will meaningfully affect the efficiency are the printFigure(n) calls. In this program, the three printFigure(n) calls correspond to the three classes that inherit from FigurePrinter in order to print one of each figure. Each of these has a while loop with two nested for loops.

First, for TrianglePrinter's printDownwardTriangle(), the while loop decrements the integer specified by the user while it is still greater than 0. This means that the loop runs n/2 times, rounding up for odd numbers. Since it is dependent on n, that means the loop is O(n). The first for loop starts at n – count, which is the specified integer minus the current count of the while loop. It then decrements this variable by 2 each iteration, so it runs n – count/2 times. Again it is dependent on n, so it is also O(n). The second for loop starts at the current while count and decrements by 1 each iteration, so it runs count times. However, count is dependent on n, so this loop is also O(n). Since the for loops are in line with each other and are both O(n), collectively they are also O(n). Since the for loops are nested within the O(n) while loop however, this method is O(n^2).

DiamondPrinter's printUpwardTriangle() also has a while loop with two nested for loops. This while loop starts at 0 and increments by 2 until it reaches the specified integer, but stops before it actually reaches, meaning that it runs one less time than in printDownwardTriangle(). Since the other while loop is linear time, this one less run does not actually impact the runtime, making this while loop O(n) as well. The for loops in this method are the exact same as in printDownwardTriangle(), meaning that the method itself is also O(n^2).

Lastly, printDownwardTrapezoid() has almost the exact same code as printDownwardTriangle, except it only iterates half as many times. Dividing a linear time action by 2 does not actually impact its runtime, as it is still dependent on n, so this method is also O(n^2).

In the main function, printDownwardTriangle() is called twice, printUpwardTriangle() is called once, and printDownwardTrapezoid() is called once, all by the printFigure() calls. This would be dependent on 4n^2, but just like we ignored the ½ before, we can ignore the 4 here, as it is still quadratic time. This means that the entire program has O(n/2) performance.