**COMP 2210 Empirical Analysis Assignment – Part A**

Austin Keelin

September 19, 2016

**Abstract**

This experiment was done to determine the big-Oh running time of the method (each student has a different method, determined using banner ID’s as the key). The method involves increasing N and recording the elapsed time to determine the log ratio, which can be used to find the big-Oh running time. Through four similar iterations of a method to determine the time complexity, the big-Oh running time was determined to be O (N4).

1. **Problem Overview**

In general, the problem was to use the scientific method in designing an experiment to find the time complexity of an algorithm. Specifically, to determine the time complexity of the timeTrial(int N) in the TimingLab class. Each student uses their Banner ID as a key required for the TimingLab constructor, which will create a TimingLab object whose timeTrial method is different for each key. The time complexity for each student was guaranteed to be proportional to Nk where k is a positive integer. The work in this experiment rests on the property of polynomial time complexity function as shown and explained in figure 1.

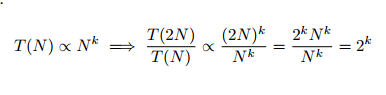


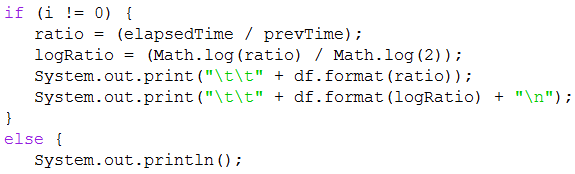
Figure 1. This property shows that as N is doubled, the ratio of the run time of

the method on the current value of N to the method’s run time on the previous

value of N converges to a constant R, which is equal to 2k, and thus k = log2R.

1. **Experimental Procedure**

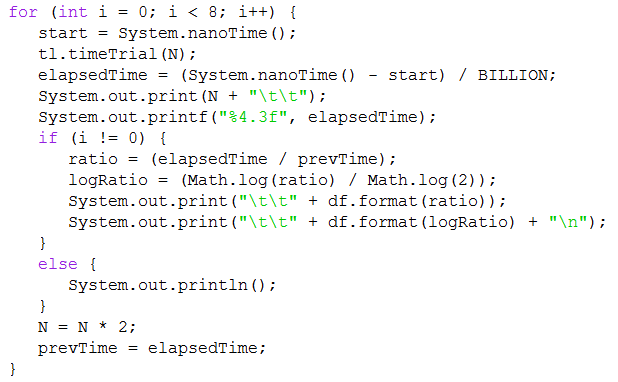
I decided editing the given TimeLabClient class would be better than starting my own class for experimentation. TimeLabClient already included a for loop which would double N and print the current value of N and the time elapsed. First thing to change was to use ratio and then create a new variable log ratio, both important for discovering the time complexity. TimeLabClient included a variable for ratio (elapsedTime / prevTime) but did not use it in the for loop. I added code to the for loop to find ratio. Log ratio is used for finding k, where k = log2R. Since there is no method in the Math class to find log base 2 of R, instead k = logR / log2 was used, which will give you the same result. Finding the ratio and log ratio for each iteration was set under an if statement to avoid finding these values for the first iteration, as they couldn’t exist.



Listing 1: Code which shows how ratio and logRatio are found and then printed.

Also shows how printing these values for the first iteration is skipped.

Now that I had a way to determine K for my method, the only other major change to TimingLabClient was to change the format for how the data was printed. Next step was to test the method to see if it could successfully determine and print N, time, ratio, and log ratio. Although I was unlucky and received a method which would take a very long time to reach a N value of 128, the class still worked as expected and I was able to determine the time complexity.



Listing 2: The entire for loop I used, the most important code in finding the time complexity.

Lastly, it is important to note the environment in which this experiment was performed. All of the data was collected through use of jGRASP 2.0.2\_01 on an ASUS laptop with Windows 10 and a 64-bit operating system.

1. **Data Collection and Analysis**

The first run was when I realized when I would have a problem and need to probably lower my problem size, which was initially at 8. This run reached a N value of 32 after 17 minutes; however, I calculated that reaching 64 would take roughly 4.79 hours. I accepted that I would need to end this run and lower the problem size. Note: for each table, N will tell the problem set value for each iteration, T will tell the elapsed time in seconds, Ratio will tell the ratio (elapsed time / previous time) for each iteration, and finally Log Ratio will tell the value of k, where k = log2R.

Table 1: Running time data for N = 8.

|  |  |  |  |
| --- | --- | --- | --- |
| **N** | **T (sec)** | **Ratio** | **Log Ratio** |
| 8 | 4.183 |  |  |
| 16 | 66.308 | 15.852 | 3.987 |
| 32 | 1069.571 | 16.13 | 4.012 |
|  |  |  |  |

For the second run, I decided to lower the problem size to 2. Although I was not sure, I hoped that this change would allow N to reach 64 in a reasonable time. Of course this was not the case, as my calculations estimated reaching 64 after 7.12 hours; but, this set did support a convergence of 4 for the log ratio, similar to what the first test had.

Table 2: Running time data for N = 2.

|  |  |  |  |
| --- | --- | --- | --- |
| **N** | **T (sec)** | **Ratio** | **Log Ratio** |
| 2 | 0.031 |  |  |
| 4 | 0.417 | 13.678 | 3.774 |
| 8 | 6.452 | 15.462 | 3.951 |
| 16 | 76.804 | 11.905 | 3.573 |
| 32 | 1403.264 | 18.271 | 4.191 |

Since I was unable to reach higher values of N such as shown in the sample report, I again decided to lower the value of N to 1. This was just a precautionary move as I figured since I can’t reach high values for N, might as well have as many values of N that could be reached in a reasonable time and I ran the method with N = 1 twice. These two trials support the data from tables 1 and 2. The result of this experiment is that the time complexity for the TimingLab class is O (N4).

Table 3: Running time data for first trial of N = 1.

|  |  |  |  |
| --- | --- | --- | --- |
| **N** | **T (sec)** | **Ratio** | **Log Ratio** |
| 1 | 0.002 |  |  |
| 2 | 0.028 | 12.257 | 3.616 |
| 4 | 0.48 | 17.444 | 4.125 |
| 8 | 5.648 | 11.758 | 3.556 |
| 16 | 88.468 | 15.662 | 3.969 |
| 32 | 1323.86 | 14.964 | 3.903 |

Table 4: Running time data for second trial of N = 1.

|  |  |  |  |
| --- | --- | --- | --- |
| **N** | **T (sec)** | **Ratio** | **Log Ratio** |
| 1 | 0.002 |  |  |
| 2 | 0.024 | 10.313 | 3.366 |
| 4 | 0.458 | 18.945 | 4.244 |
| 8 | 5.185 | 11.33 | 3.502 |
| 16 | 82.444 | 15.900 | 3.991 |
| 32 | 1255.769 | 15.232 | 3.929 |

1. **Interpretation**

The big-Oh running time for the TimingLab class must be O (N4). The main evidence for this conclusion is the data presented in tables 3 and 4 (even though both tables 1 and 2 also support this claim). In both runs of the data presented in these tables, the log ratio approaches 4 and is close enough to 4 that this is the only possibility for the time complexity.