COMP 2210 Assignment 3 Part A

Group 38

Robinson Davis Steven Han

February 25, 2014

Abstract

Part A of activity three consisted of finding the big-Oh time complexity of method timetrial(int N), in the provided TimingLab class. Groups are informed of the fact that the method's time complexity will be proportional to N^k and according to a property of polynomial time complexity functions T(N), N^k is proportional to 2^k. It was determined that the best way to calculate time complexity given this information is call the timetrail(int N) method while iterating through increasing values of N, recording the runtime, finding the ratio between each runtime and taking $\log_2$ of the found ratios. When this is done one discovers that $k \approx 3$ indicating a time complexity of about $O(N^3)$.

1 Problem Overview

A repeatable experimental procedure that allows one to empirically discover big-Oh time complexity of provided method timetrail(int N) must be developed and performed. The source code is unavailable which means timing tests must be ran in order to determine time complexity which can be accomplished within jGRASP without any need for external programs and only minor modifications to provided code, mostly just for format. The data from these tests can be analyzed and will provide sufficient information to determine big-Oh time complexity.

2 Experimental Procedure

These experiments were conducted on a PC with the following specifications:

- Operating System: Windows 8.1 64-bit
- Processor: Intel i7-4700MQ CPU
- RAM: 8GB DDR3L SDRAM
- Java Version: 1.7.0_51

The first step was modifying the provided code to meet computer specification limitations. Any array over sixty-four elements took enough time to not be plausible, or necessary. Therefore the code was edited to test six N values: two, four, six, eight, sixteen, thirty-two, and sixty-four; minor formatting was added as well. This code is displayed in Figure 2(a).

```
// measure elapsed time for multiple calls to timeTrial
// with increasing N values
N = 2;
System.out.println("N Value\t\tRuntime\t\tRatio\t\tlgRatio");
for (int i = 0; i < 6; i++) {
   clock = new Clock();
   tl.timeTrial(N);
   System.out.print("    " + N + "\t\t");
   // calculate and print ratio
   elapsedTime = clock.elapsedTime();
   System.out.printf("%4.3f\t\t", elapsedTime);
   if (prevTime == 0) {
      prevTime = elapsedTime;
   }
   // find and print ratio
   ratio = elapsedTime/prevTime;
   System.out.printf("%4.3f\t\t", ratio);
   // print lgratio
   lgratio = java.lang.Math.log10(ratio)/java.lang.Math.log10(2);
   System.out.printf("%4.3f\n", lgratio);
   // double n and drag along prevTime
   N *= 2;
   prevTime = elapsedTime;
}
```

Figure 2(a) – Code to record and display time complexity experiment data.

With this code we can effectively record the time each call to timeTrial(N) takes, calculate the runtime between each call, and calculate the $\log_2$ of each ratio. Because of the given property of polynomial time complexity, shown in Figure 2(b), one can conclude that the $\log_2$ ratio will reveal the unknown exponent k. A total of five tests were executed, each to check for consistence amongst the executions. Once consistency was established the final test could effectively be recorded and analyzed as an accurate representation of time complexity.

$$T(N) \propto N^k \implies \frac{T(2N)}{T(N)} \propto \frac{(2N)^k}{N^k} = \frac{2^k N^k}{N^k} = 2^k$$

Figure 2(b) – Given property of polynomial time complexity functions T(N)

3 Data Collection and Analysis

Data collection involved the task of executing the code found in Figure 2(a). The output from this execution can be found below in Figure 3(a). To help visualize this data, a graph of the runtime for each N value is provided in Figure 3(b) as well as a graph for the ratios and $\log_2$ of the ratios is shown in Figure 3(c). Observing Figure 3(b) we see that timetrail(int N) has a quadratic growth rate, which correlates with the information that timetrial(int N) will have a time complexity of N^k.

```
----jGRASP exec: java -ea TimingLabClient

This call to method foo() took 0.014 seconds.
This call to method TimingLab.timeTrial(2 took 0.032 seconds.)


N Value         Runtime         Ratio           lgRatio
   2            0.026           1.000           0.000
   4            0.103           3.962           1.986
   8            0.830           8.058           3.010
   16           4.039           4.866           2.283
   32           31.502          7.799           2.963
   64           253.166         8.037           3.007

----jGRASP: operation complete.
```

Figure 3(a) – Result of executed code in Figure 2(a) **Note:** Runtime is in seconds.

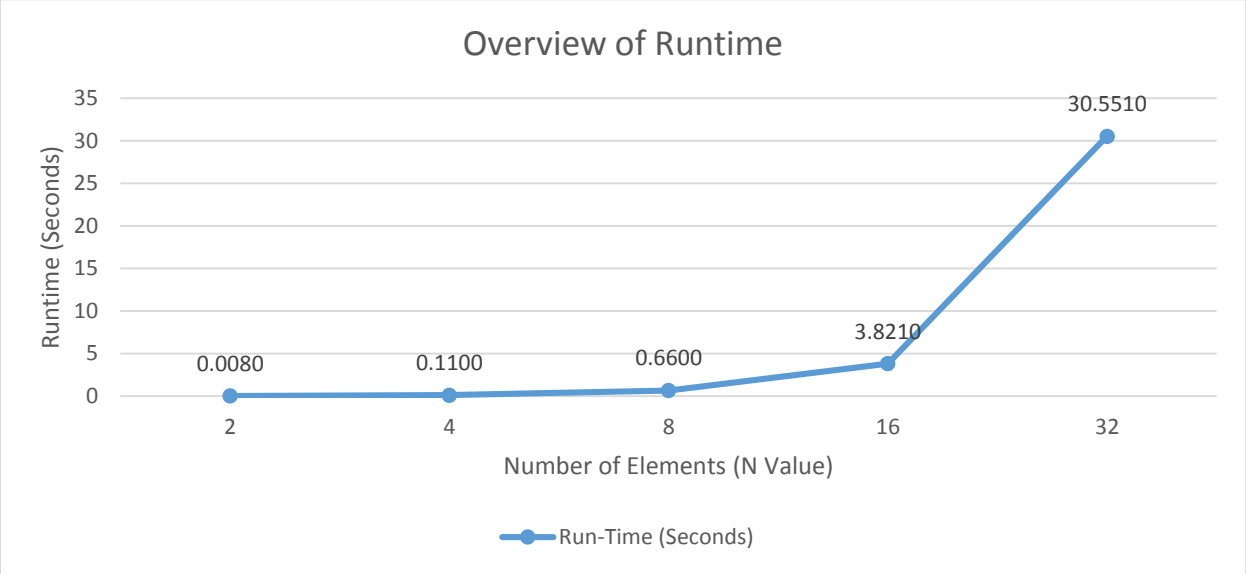

Figure 3(b) – Graph of information from Figure 3(a).

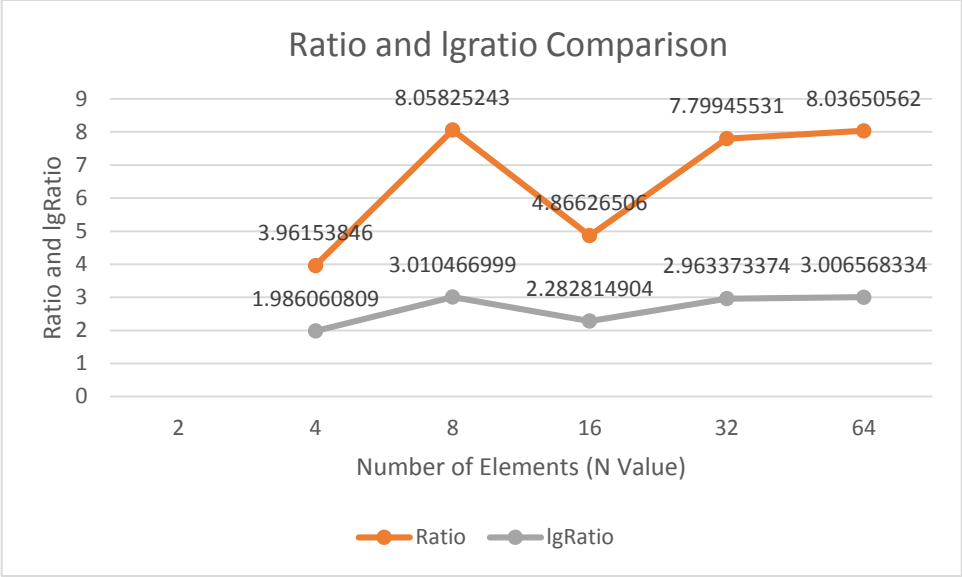

Figure 3(c) – Comparison between ratio and the $\log_2$(ratio)

4 Interpretation

The goal was to develop a repeatable experiment that would allow a person to find the time complexity of a given method. The tests described above provide sufficient steps in order to accomplish this goal, and will also provide useful for Part B of this activity. As the $\log_2$ of each ratio is recorded, it can be observed approaching three. Once again using the provided property of polynomial time complexity functions T(N) in Figure 2(b), we know that the $\log_2$ of each ratio will provide an educated guess on the unknown k value. Therefore it can be concluded that $k \approx 3$ and that the timeTrail(int N) method has a time complexity of $O(N^3)$.