COMP 2210 Assignment 3 Part A

Group 44

John C. Carroll James C. Bass

February 24, 2014

Abstract

This document presents an example for any student to follow when determining big-Oh time complexities of any given method without seeing the source code. In this experiment, by way of testing, using successively larger values of N, and k iterations, conclusive data was collected. That data supports the hypothesis based on the property of the polynomial time complexity functions *T(N).* From analysis of the recorded data, a well educated guess as to what the time complexity was made. The big-Oh time complexity for key(44) is $O(N^3)$.

1. Problem Overview

When writing algorithms involving large sets of objects and information, efficiency can make all the difference in speed and usability. The source code is unavailable in this problem.

The goal of this experiment is to determine the time complexity of the timeTrial (int N) method in the TimingLab class. One must develop and perform a repeatable experimental procedure that will lead to the discovery of the big-Oh time complexity which is specific to each group. Each group is assigned a key to use. (i.e. Group 23 would invoke the constructor SortingLab(22)). Also, the Clock class should be used to gather the experimental timing data.

In this experiment, trials can be ran on the TimingLab class using the provided TimingLabClient, as it has relevant example code for use. Modifications should be made to this client and observable data should then be yielded. Deductions and justified calculations about the class method's big-Oh time complexity should then be made through its analysis.

2. Experimental Procedure

Experimental environment: a group member's home personal computer.

Machine specifications:

Operating System

Windows 7 Professional 64-bit SP1

CPU

Intel Core i7 3770K @ 4.2GHz (Over-clocked)	38 °C (average) 52 °C (when running test)

RAM

G.SKILL 16.0GB Dual-Channel DDR3 1866MHz @ 2133MHz (Over-clocked) Timing (9-10-9-28)

Motherboard

ASUSTeK COMPUTER INC. P8Z77-V (LGA1155)	34 °C

Graphics

3072MB ATI AMD Radeon HD 7950 (Gigabyte™)	43 °C

Storage

232GB Samsung SSD 840 Series (SSD)	33 °C
1863GB Seagate (SATA)	40 °C

Java

Java Runtime Environment

Path	C:\Program Files (x86)\Java\jre7\bin\java.exe
Version	7.0
Update	51
Build	13

Java Runtime Environment

Path	C:\Program Files\Java\jdk1.7.0_25\bin\java.exe
Version	7.0
Update	25
Build	17

Java Runtime Environment

Path	C:\Program Files\Java\jre7\bin\java.exe
Version	7.0
Update	25
Build	17

The procedure for the experiment went as follows:

Created the project inside of the same project folder as the files that were provided and added all of them to the project. With set-up complete, the given TimingLabClient was ran using its default source code.

The client printed the elapsed time value for a starting size of N, then doubled it with each iteration, and reprinted by default.

```
N = 2;
System.out.println("   Run       N       Time      Ratio    lg Ratio");

for (int i = 0; i < 13; i++) {
    clock = new Clock();
    tl.timeTrial(N);
    prevTime = elapsedTime;

    elapsedTime = clock.elapsedTime();
    ratio = elapsedTime/prevTime;
    lgratio = (Math.log(ratio) / Math.log(2));

    System.out.printf("%7d %7d", i+1, N);
    System.out.printf("%10.3fs %10.3f %10.3f\n", elapsedTime, ratio, lgratio);

    N *= 2;
}

}
```

Screenshot 1 - Focuses on added code

Screenshot (1) entails the code changed, besides the key value at the top to match the group's number: 44. N's value was changed to 2 and iterated until there was a noticeable slowing down with the machine taking longer to process. From starting value N = 2, the method did not work relatively slowly until the problem size reached N = 1024. The method was iterated 13 times because it gave the data a larger spread.

The iterator was given a formula for the ratio, which is the current time divided by the previous time. From there, the log-ratio ($\log_2$(ratio)) was added. The code added had printed: the number of the run, value of N, Time elapsed, the ratio, and the log-ratio. This data provided the necessary information needed to identify the big-Oh time complexity. This is explained further in part 3's analysis.

3. Data Collection and Analysis

The experiment was made to effectively calculate and output the results. Below is a screenshot of that output. The elapsed times in the data below do not begin registering until problem size N=64. From there, time is observed to have exponential growth with a doubled work load of N.

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

This call to method foo() took 0.012 seconds.
This call to method TimingLab.timeTrial(64 took 0.012 seconds.)
   Run       N       Time      Ratio    lg Ratio
     1       2     0.000s      0.000  -Infinity
     2       4     0.000s        NaN         NaN
     3       8     0.000s        NaN         NaN
     4      16     0.000s        NaN         NaN
     5      32     0.000s        NaN         NaN
     6      64     0.000s        NaN         NaN
     7     128     0.004s   Infinity   Infinity
     8     256     0.029s      7.250      2.858
     9     512     0.227s      7.828      2.969
    10    1024     1.841s      8.110      3.020
    11    2048    14.351s      7.795      2.963
    12    4096   114.618s      7.987      2.998
    13    8192   920.325s      8.029      3.005

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

Screenshot 2 - Directly from jGrasp output

Table 1 – TimingLab.class data

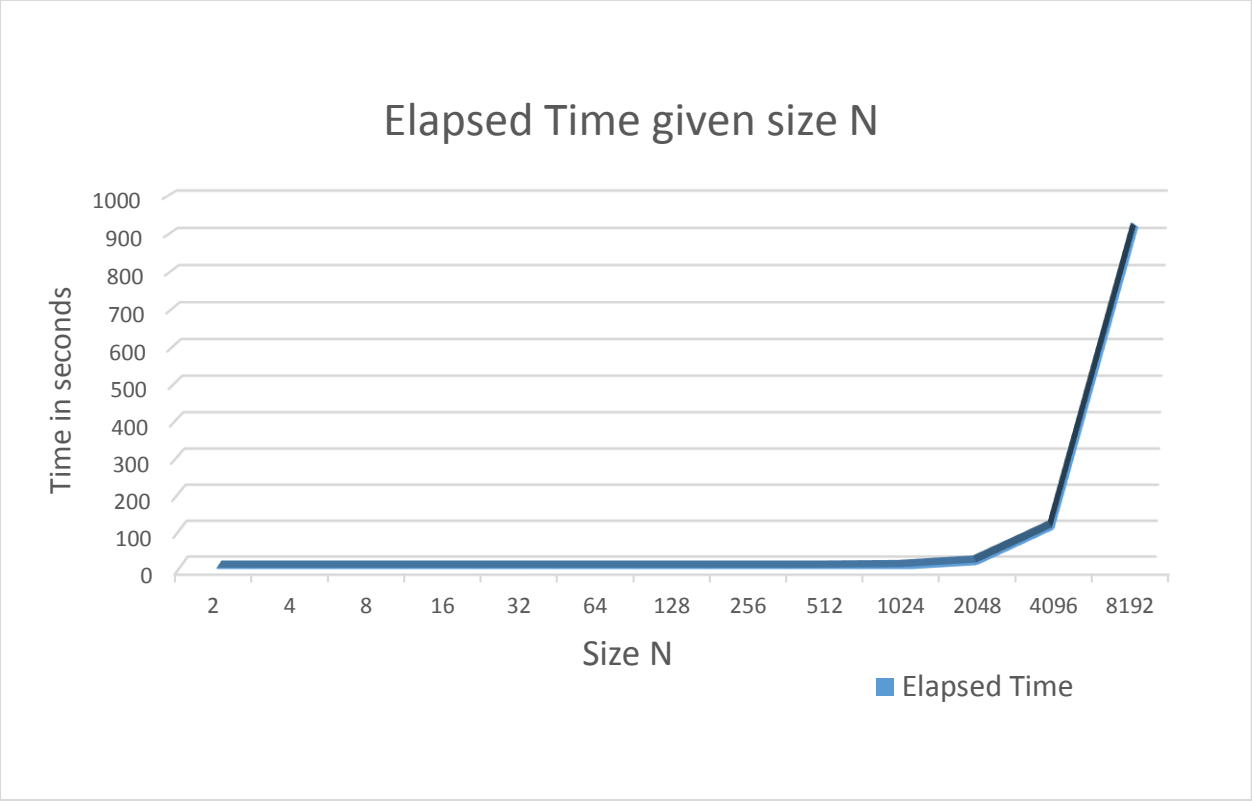
This table is a repeated representation of the data results provided from Screenshot (2).

Run	N	Time	Ratio	log Ratio
1	2	0.000s	N/A	N/A
2	4	0.000s	N/A	N/A
3	8	0.000s	N/A	N/A
4	16	0.000s	N/A	N/A
5	32	0.000s	N/A	N/A
6	64	0.000s	N/A	N/A
7	128	00.004s	N/A	N/A
8	256	0.029s	7.250	2.858
9	512	0.227s	7.828	2.969
10	1024	1.841s	8.110	3.020
11	2048	14.351s	7.795	2.963
12	4096	114.618s	7.987	2.998
13	8192	920.325s	8.029	3.005

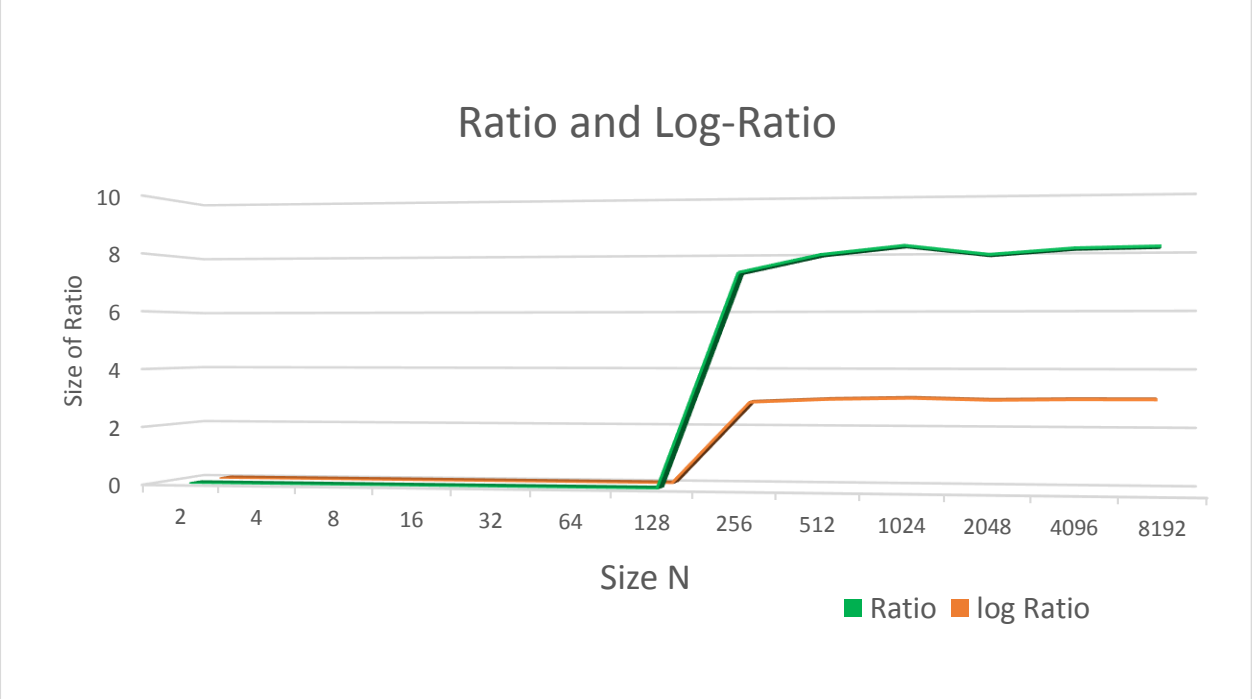
As you can see from the table above, and charts below, the $\log_2$ of the ratio(log Ratio) is approaching a number: 3. Using the property of polynomial time complexity functions *T(N)*,

$$T(N) \propto N^k \implies \frac{T(2N)}{T(N)} \propto \frac{(2N)^k}{N^k} = \frac{2^k N^k}{N^k} = 2^k \quad (1)$$

it is observed that the value of k = 3. From this test procedure, the big-Oh time complexity has been identified. That value can safely be said to be $O(N^3)$.



Graph 1 - This 3-D line graph plots the increase of the size of N (x-axis) against the increase in time (y-axis) in seconds. Data is from Table 1.



Graph 2 - This 3-D line graph plots the differences of the two ratios, given an increase of N (x-axis). Size of the ratio is on the y-axis. Data is from Table 1.

4. Interpretation

This experiment was to test if it was possible to determine the big-Oh time complexity of an unknown method and be able to discern its efficiency. By way of running trials on the given class and doubling the amount of data to process with each iteration, it was possible to record data and make an astute analysis on the time complexity without seeing the source code. Through these test results and analysis, the big-Oh time complexity has been determined. That value can safely be said to be $O(N^3)$.