COMP 2210 Empirical Analysis Assignment – Part A

Zejian Zhong

July 3, 2017

Abstract

The efficiency of an algorithm can be determined by calculating the time complexity (in term of big-Oh) of it. This assignment is to find the time complexity of an algorithm that is stored in a .jar file that is invisible to users. However, the result can be subjective because of different methods and different machines. In my case, the ratio R was approaching to 4, and the k, which is $\log_2(R)$, approaching to 2.

1 Problem Overview

The assignment is a departure from previous assignments insomuch as its focus is not on program construction, but is instead on experimentation, analysis, critical thinking, and applying the scientific method. To find the time complexity, I need to run the method that is stored in a provided .jar file and record the time complexity based on the problem size (int N), which represents the time trial. And I used my Banner ID number as the key required by the constructor. From the following property:

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

We can know that, the time complexity should be proportional to N_k for int k, and the ratio R should converge to a constant 2^k, so $k = \log_2(R)$.

2 Experimental Procedure

Below are some specs of the used machine:

- System: Windows 10 Pro 64-bit
- Processor: Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz
- RAM: 16.0 GB

- Java version: Java 8 update 131

I did not change the given number for N since the result time was easy to test repeatedly in a short time. My key was my Banner ID: 903907977. Below is the code to generate timing data:

```
/** generate timing data */

// time a single method in this class
// to increase accuracy of later timing
start = System.nanoTime();
methodToTime();
elapsedTime = (System.nanoTime() - start) / BILLION;
System.out.print("This call to method methodToTime() took ");
System.out.printf("%4.3f", elapsedTime);
System.out.println(" seconds.");

// measure elapsed time for a single call to timeTrial
TimingLab tl = new TimingLab(key);
start = System.nanoTime();
tl.timeTrial(N);
elapsedTime = (System.nanoTime() - start) / BILLION;
System.out.print("This call to method TimingLab.timeTrial("
   + N + ") took ");
System.out.printf("%4.3f", elapsedTime);
System.out.println(" seconds.");

// measure elapsed time for multiple calls to timeTrial
// with doubling N values
System.out.print("Timing multiple calls to timeTrial(N) ");
System.out.println("with increasing N values.");
System.out.println("N\tElapsed Time (sec)");
for (int i = 0; i < 7; i++) {
   start = System.nanoTime();
   tl.timeTrial(N);
   elapsedTime = (System.nanoTime() - start) / BILLION;
   System.out.print(N + "\t");
   System.out.printf("%4.3f\n", elapsedTime);
   N = N * 2;
}
```

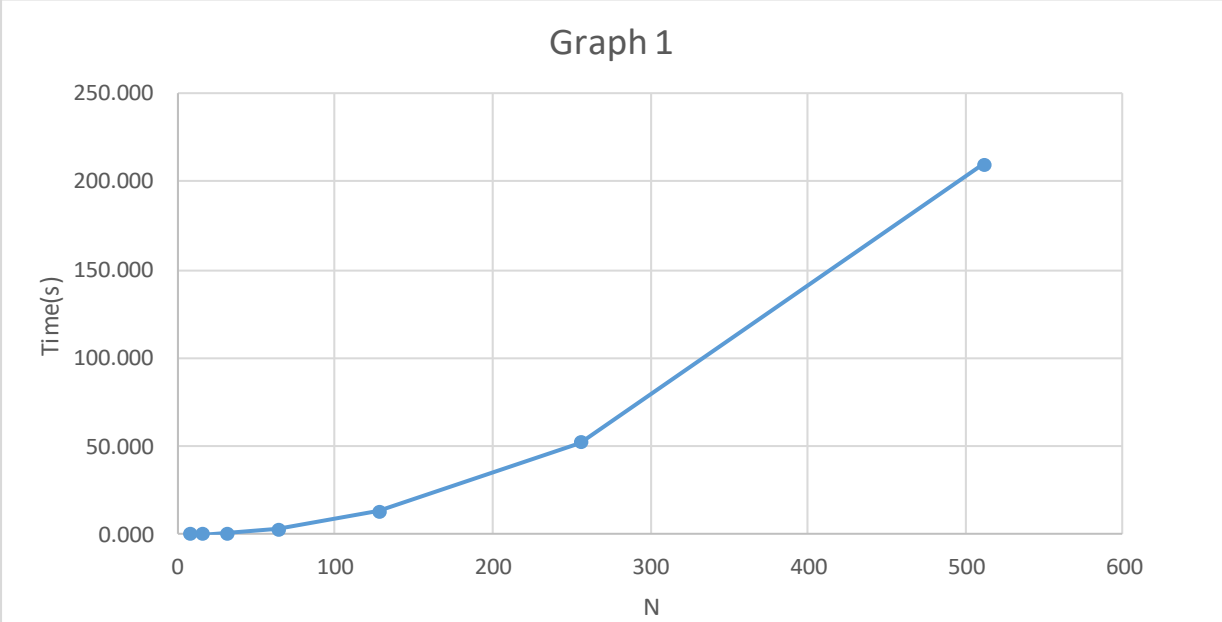
3 Data Collection and Analysis

By type the output of the code into Microsoft Excel, I got the table as shown below:

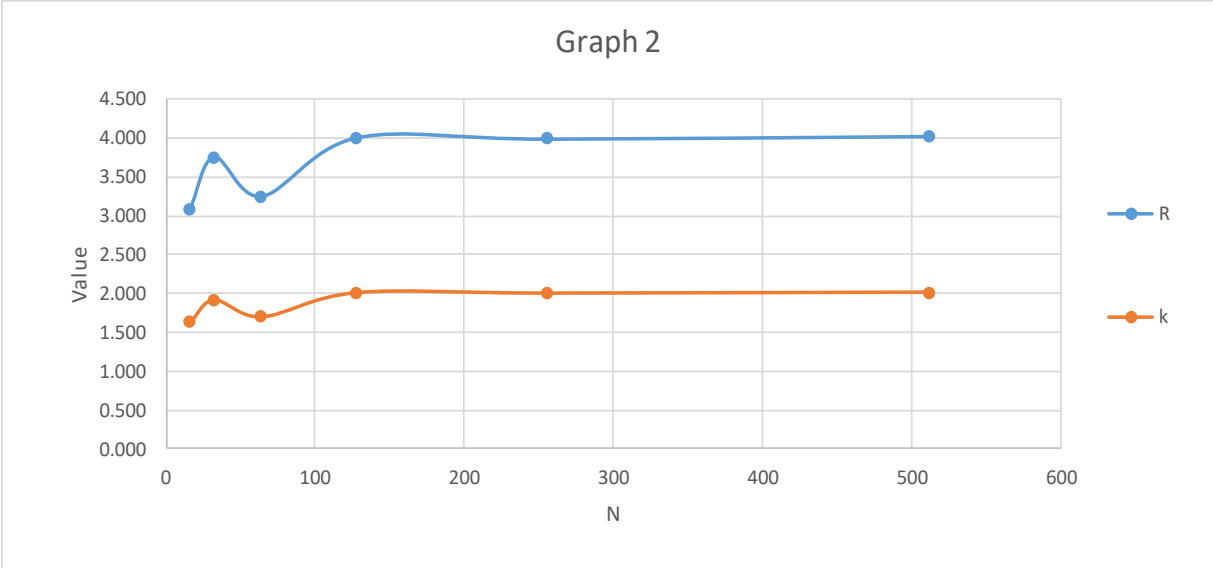
N	Time(s)	R	k
8	0.086	-	-
16	0.266	3.093	1.629
32	1.000	3.759	1.911
64	3.250	3.250	1.700
128	13.031	4.010	2.003
256	52.013	3.991	1.997
512	209.393	4.026	2.009

N is the time trial, Time is the running time that it took, R is the ratio (current row's time/previous row's time) and k is $\log_2(R)$.

Then I created two graphs as shown below to get a clearer look of the data.



Graph 1 shows the relation between the problem size N and the running time. This graph looks like a quadratic function.



Then I created graph 2 and it shows that the R is approaching to 4 and k is approaching to 2.

4 Interpretation

From the 2 graphs I created and the given property of N and k, we can calculate the k by $4 = 2^k$, so the k = 2, which means that the time complexity for algorithm in the .jar file is $O(N^2)$.