COMP 2210 Empirical Analysis Assignment – Part A

Maxwell Heeschen

September 24, 2016

**Abstract**

The purpose of this experiment was to find the time complexity of a method given to me called methodToTime. I wasn’t able to see the code, as it was in a .jar file, so I had to measure the time that different problem sizes would take. I recorded the time of doubling problem sizes, and calculated the ratio of the time between each problem size and the problem size before it. Because this ratio was approaching 4, this meant that the time complexity of the method was N2 due to log2(R) approaching 2.

1. Problem Overview

The problem looked over in this study was determining the time complexity in an unknown method given by an instructor. The method is located in a .jar file and I was provided a sample file that ran several methods in the .jar file using a key which was my TigerID. The time complexity of an algorithm can be found if I can collect the data for several runs using twice as many inputs as the last data run, and then measuring the time that each run takes. The time complexity can then be found with the following equation:

T(N)∝ Nk =⇒ T(2N)/T(N) ∝ (2N)­k/Nk = 2kNk/Nk = 2k

Where k is used as the exponent for N in the notation O(Nk).

1. Experimental Procedure

The system specs that I used were OS: Windows 10 Home 64-bit, Processor: AMD FX-8150 Eight-Core Processor ~3.6GHz, and Memory: 8192 MB. The starting number of elements (N) that I chose was 1000 because the default number was yielding times that were too small. My TigerID that I used for the key was 903532323. First a single method was timed “to increase the accuracy of later timing”.

start = System.nanoTime();  
46 methodToTime();  
47 elapsedTime = (System.nanoTime() - start) / BILLION;  
48 System.out.print("This call to method methodToTime() took ");  
49 System.out.printf("%4.3f", elapsedTime);  
50 System.out.println(" seconds.");

Here is the method methodToTime that is being called:

private static void methodToTime() {  
83 for (int i = 0; i < 100000; i++) {  
84 String s1 = "War";  
85 String s2 = "Eagle";  
86 String s3 = s1 + s2;  
87 s1 = null;  
88 s2 = null;  
89 s3 = null;  
90 }

After that, there were multiple calls made to the method timeTrial where the value of N doubled each time:

TimingLab tl = new TimingLab(key);  
54 start = System.nanoTime();  
55 tl.timeTrial(N);  
56 elapsedTime = (System.nanoTime() - start) / BILLION;  
57 System.out.print("This call to method TimingLab.timeTrial("  
58 + N + ") took ");  
59 System.out.printf("%4.3f", elapsedTime);  
60 System.out.println(" seconds.");  
61   
62  // measure elapsed time for multiple calls to timeTrial  
63  // with doubling N values  
64 System.out.print("Timing multiple calls to timeTrial(N) ");  
65 System.out.println("with increasing N values.");  
66 System.out.println("N\tElapsed Time (sec)");  
67 for (int i = 0; i < 20; i++) {  
68 start = System.nanoTime();  
69 tl.timeTrial(N);  
70 elapsedTime = (System.nanoTime() - start) / BILLION;  
71 System.out.print(N + "\t");  
72 System.out.printf("%4.3f\n", elapsedTime);  
73 N = N \* 2;  
74 }

This method output several values into an easily readable table format.

1. Data Collection and Analysis

Here is a table of the values displayed by the method described above where N is the problem size passed to methodToTime, Time is the time that it took, and R is the current row’s time divided by the previous row’s time which is equal to 2k:

|  |  |  |
| --- | --- | --- |
| Table 1 | | |
| N | Time | R (rounded to nearest hundredth) |
| 1000 | .003 | -- |
| 2000 | .011 | 3.67 |
| 4000 | .047 | 4.27 |
| 8000 | .183 | 3.89 |
| 16000 | .723 | 3.95 |
| 32000 | 2.876 | 3.98 |
| 64000 | 11.505 | 4.00 |
| 128000 | 46.043 | 4.00 |
| 256000 | 184.021 | 4.00 |

When graphing the time versus N, it is easy to see that the time complexity is quadratic, so I had to find out how it was quadratic. The graph of R shows the line approaching 4.

1. Interpretation

Because Graph 1 was quadratic, I needed to know how it was quadratic. Using Graph 2, and how it approaches 4, I can set 4 = 2k. When I solve for k I get k = 2, which implies that the time complexity for the algorithm in the method methodToTime is O(N2).