COMP 2210 Empirical Analysis Assignment – Part A

George P. Burdell

September 18, 2016

Abstract

We identify through empirical evidence the big-Oh time complexity of the **`timeTrial`** method in the **`TimingLab`** class. Timing data relative to this method was collected for different problem sizes, and observations of growth rate were made using a *power law* assumption. Analysis of the data determined that **`timeTrial`** has $O(N^3)$ time complexity.

1 Problem Overview

Part A of the assignment required us to empirically discover the big-Oh time complexity of the **`timeTrial`** method in the **`TimingLab`** class. This method takes a single `int` parameter, **`N`**, that represents the *problem size* of the underlying algorithm. The constructor of the **`TimingLab`** class takes a single `int` parameter (the *key*) that is used to map the **`timeTrial`** method to one of various algorithms, each with a different time complexity.

Regardless of the value of *key*, however, the underlying algorithm that is associated with the **`timeTrial`** method has *polynomial* time complexity. Specifically, **`timeTrial`** is guaranteed to have time complexity $T(N)$ that is $O(N^k)$ for some $k > 1$. Thus, the following "power law" property holds.

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

This property allows us to record running times for m calls to **`timeTrial(N)`** with the value of **`N`** doubling on each successive call (i.e., $N_i = 2N_{i-1}$), and then calculate the $m - 1$ running time ratios. This data is described in the table below, where column N records the problem size, column $T(N)$ records the time required by the **`timeTrial`** method on the current problem size, and column R is the ratio of the time required for current run to the time required for the previous run (i.e., $T(N_i)/T(N_{i-1})$). Per the power law property above, the values of R will converge to a value $\hat{R} = 2^k$, making the time complexity of the **`timeTrial`** method $O(N^{\log_2 \hat{R}})$.

Table 1: Description of running time data

N	$T(N)$	R
N_0	$T(N_0)$	–
N_1	$T(N_1)$	$T(N_1)/T(N_0)$
N_2	$T(N_2)$	$T(N_2)/T(N_1)$
...	...	...
N_i	$T(N_i)$	$T(N_i)/T(N_{i-1})$
...	...	...
N_m	$T(N_m)$	$T(N_m)/T(N_{m-1})$

2 Experimental Procedure

The following is an outline of our experimental procedure, which follows directly from the preceding discussion.

1. Collect running time data for the **timeTrial** method, as described in Table 1.
2. Analyze the timing data to identify $\hat{R}$.

The following subsections discuss the experimental materials and context and elaborate on each step of the experimental procedure.

2.1 Experimental materials

Two Java classes constitute the experimental materials: **TimingLab**, which contains the **timeTrial** method under study, and **TimingLabSolution**, which generates timing data for the **timeTrial** method. We wrote **TimingLabSolution** to implement step 1 of the experimental procedure.

We were provided with a jar file (**resources.jar**) that contained the **TimingLab** class. The method under study (**timeTrial**) is a static method of this class. The **TimingLab** constructor takes an `int` parameter (the *key*) that is used to map the **timeTrial** to a particular polynomial-time algorithm. We were required to use our Banner ID number as the key when instantiating a **TimingLab** object.

2.2 Collecting running time data

We wrote **TimingLabSolution** to generate timing data as shown in Table 1. Specifically, **TimingLabSolution** records the running time required by the **timeTrial** method on problem sizes that are successively doubled.

The code that generated the timing data is shown below. (The complete source code for **TimingLabSolution** is given at the end of this report.) The variables **N** and **runs** are set via command-line arguments, as is the constructor's key value. To execute **TimingLabSolution** for our group (23) with an initial problem size of 16 and 10 timing runs, we would issue the following command (assuming **A3.jar** is in the current working directory): `% java -cp .:A3.jar TimingLabSolution 23 16 10`.

Note that, per the assignment requirements, **System.nanoTime()** was used to measure elapsed time. The identifier **BILLION** is a double with value 1,000,000 and is used to convert nanoseconds to seconds.

```
// call the method a few times before collecting data
for (int i = 0; i < 5; i++) {
  tl.timeTrial(N);
}

// gather the timing data
double startTime, elapsedTime;
System.out.println(String.format("%-5s%-5s%-8s", "Run", "N", "Time"));
for (int i = 1; i <= runs; i++) {
  startTime = System.nanoTime();
  tl.timeTrial(N);
  elapsedTime = (System.nanoTime() - startTime) / BILLION;
  System.out.println(String.format("%-5d%-5d%-8.2f", i, N, elapsedTime));
  N *= 2;
}
```

2.3 Analyzing running time data

Analysis of the running time data produced by **TimingLabSolution** involves computing the ratio of the running time for the current run to the running time of the previous run across all runs, and then estimating

the "steady state" value of this ratio ($\hat{R}$). The power law property above tells us that $\log_2 \hat{R}$ is the exponent k in the big-Oh expression $O(N^k)$ for the method **`timeTrial`**.

3 Data Collection and Analysis

Timing data was generated by **`TimingLabSolution`**. The computer environment in which it was run is described below.

- Computer: iMac (27-inch Mid 2011), 2.7GHz Intel Core i5 processor, 8GB 1333 MHz DDR3 memory
- Operating system: OS X 10.9.4
- Java: 1.8.0
  - **`javac -version`**: javac 1.8.0
  - **`java -version`**: java version "1.8.0" Java(TM) SE Runtime Environment (build 1.8.0-b132) Java HotSpot(TM) 64-Bit Server VM (build 25.0-b70, mixed mode)
  - **`System.nanoTime()`** used to measure elapsed time

3.1 Timing Data

Timing data was generated by running **`TimingLabSolution`** with problem sizes 16 to 256 inclusive. The command issued to run the program and the resulting program output (both copied from a terminal window) are shown below. All times are reported in seconds. (Note that due to the rapid growth rate in running times, the program was aborted after only half of the planned runs (10) were executed, giving data for **`N`** = 16 to 256 inclusive.)

```
$ java -cp .:A3.jar TimingLabSolution 23 16 10
Run  N    Time
1    16   4.55
2    32   34.48
3    64   276.53
4    128  2204.73
5    256  17531.89
```

The table below shows the raw data along with computations of each running time ratio (column R).

Table 2: Running time data with ratio calculations

N	*Time*	R
16	4.55	–
32	34.48	7.58
64	276.53	8.02
128	2204.73	7.97
256	17531.89	7.95

Although the excessive time requirements (almost five hours for the last run) required us to cut the planned data collection in half, it was clear that $\hat{R} \approx 8$.

4 Interpretation

With $\hat{R} = 8$, $k = \log_2 \hat{R} = 3$, and thus the method **`timeTrial`** has time complexity $O(N^3)$.