COMP 2210 Assignment 3, Part A

Group 77
Kelton McClantoc   Walker Wood
September 15, 2015

Part 1: Problem Overview

The intention of this project was for the group to empirically discover the big-Oh time complexity of *timeTrial (int N)* method from the given class *TimingLab*. The process for determining the running time of this method involves using the variable N to represent the problem size. It is assumed that the time complexity of the *timeTrial (int N)* method with respect to the problem size "T(N)" is proportional to the value N^k (where k is a positive integer) in such a way that:

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

(Eq. 1)

We assume from this, that as N is doubled, the ratio (R) of the running time on the current N to the last running time (T(2N)/T(N)) will converge to a numerical constant 2^k.

Part 2: Experimental Procedure

The group started by creating a testing class and instantiating the *TimingLab* class (public TimingLab(int key)) with key = 71. This key corresponds to the assigned group number and instantiates the class with unique methods to this particular key. Five variables of type double were declared:

double BILLION = 1000000000d;	// nanoseconds to seconds
double start = 0;	// start time of the current run
double elapsedTime = 0;	// elapsed time of current run
double prevTime = 0;	// elapsed time of previous run
double ratio = 1;	// elapsedTime / prevTime

The maximum problem size chosen for the experiment was N = 1024. This corresponds to 2^{10}. In order logically approach this superficial limit, the group decided to iterate through a for loop declared as “for (int i = 1, i <= 1024; i = i * 2)”. For timing accuracy, the group used System.nanoTime(), which returns a double representing the time, in nanoseconds, of the running system. By assigning the start variable to System.nanoTime() directly before calling the *timeTrial* (int N) method, and by assigning elapsedTime to System.nanoTime() - start, we can determine the time taken, in nanoseconds, to execute *timeTrial* (int N). Then, by dividing the time taken by BILLION, the time in seconds is hereby determined. This is represented by code as follows:

```
start = System.nanoTime();
tl.timeTrial(i);
elapsedTime = (System.nanoTime() - start) / BILLION;
```

where i is the current iteration value of the aforementioned for loop. Following, we printed the elapsed time via the System.out.printf() method, with the parameters “%4.3f” to display elapsedTime to the user. If prevTime != 0, then the ratio of the two times (elapesdTime / prevTime) is determined and printed, as well as the value k, which is equivalent to $\log_2 R$. This equivalency was determined from Eq. 1 above. At the end of the current iteration of the for loop, prevTime is set to the current value of elapsedTime. This ensures that the first elapsed time at i = 1 is not divided by zero, as prevTime initially has a value of zero. It also ensures that every iteration after i = 1 has a calculated ratio.

```
if(prevTime != 0) {
  ratio = elapsedTime/prevTime;
  System.out.println("R = " + ratio);
  System.out.println("K = " + (Math.log10(ratio)/Math.log10(2)));
  }
  prevTime = 0;
  System.out.println("");
```

Because Eq.1 states that T(N) is proportional to N^k the big-Oh (O(N)) running time can be found by noting what the printed k value converges to over time. Thus, the big-Oh running time can said to be ($O(N^k)$) where N is any problem size greater than zero and k is the non zero k value found through experimentation.

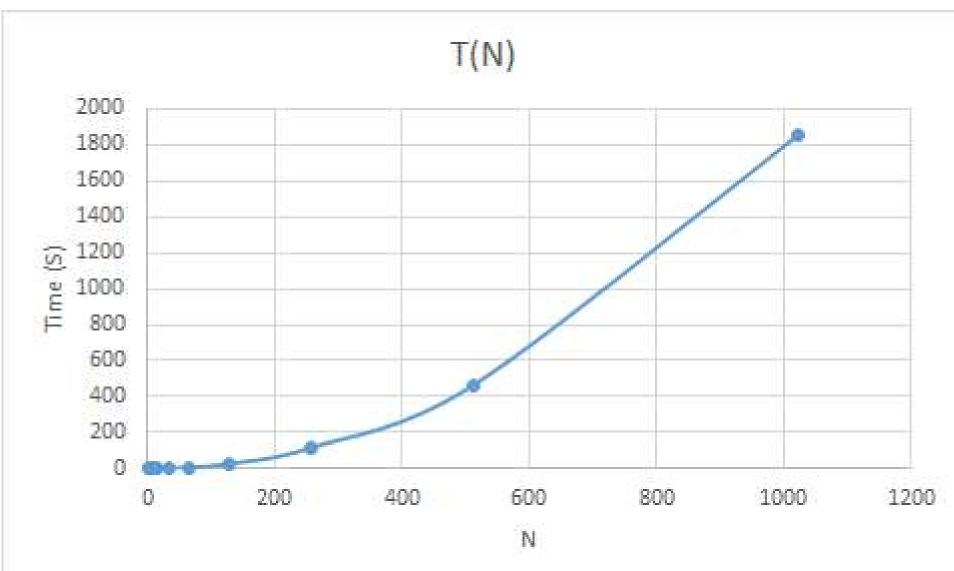
Part 3: Data Collection and Analysis

The data obtained from experimentation can aptly and succinctly be conveyed in a table and a graph. In the figures, N represents the size of the problem, Time (T) is the amount of time in seconds that the method ran for on a corresponding N, R is the ratio of the current N to the preceding N in the table, k is calculated from $\log_2 R$ and corresponds to the same k that is needed to find the big-Oh running time, and key is the key used on the TimingLab(int key) constructor.

	Key:	71	
N	Time(S)	R(t/t-1)	K
1	0.005	undef	undef
2	0.016	3.215	1.685
4	0.029	1.824	0.867
8	0.151	5.158	2.37
16	0.643	3.002	2.092
32	1.93	3.002	1.587
64	7.17	3.715	1.893
128	28.857	4.025	2.009
256	116.136	4.025	2.009
512	464.013	3.995	1.998
1024	1857.752	4.004	2.001

(Table 1)

Below is a graph of T as a function of N



(Figure 1)

As Table 1 shows, as N increases by double the previous value, time increases exponentially. Taking the ratio of the successive times reveals that the ratio eventually converges to a value of four, and by applying this information in Eq.1 we can determine that k converges to two.

Part 4: Interpretation

Equation one tells us, as stated in part 1, that as N is doubled, the ratio (R) of the running time on the current N to the last running time (T(2N)/T(N)) will converge to a numerical constant 2^k. By examining Table 1, we can see that the group's data proved to be consistent with this assumption in that the experimental value for K converged to two, giving the expected ratio. Additionally, the exponential curve shown by Figure 1 further demonstrates this expected consistency. This shows that the group's test class was very likely to have been properly implemented, and the big-oh time complexity is $O(N^2)$.