Empirical Analysis – Time Complexity

COMP 2210 Assignment

Assignment Overview

The focus of this assignment is on experimentation, analysis, critical thinking, and applying the scientific method. The provided resources for this assignment are the following.

- `provided.jar` — A jar file containing various classes you will need for this assignment. One class, `ProvidedClass`, has a method named `methodToTime` for which you must experimentally determine the time complexity.
- `ProvidedClient.java` — A Java class that illustrates basic interaction with `ProvidedClass`. *This class is for illustration only.* You can modify and use it for your own purposes or you can use it as an example to create a different but similar class of your own.

Experimental Discovery of Time Complexity

You must develop and perform a repeatable experimental procedure that will allow you to empirically discover the time complexity (in term of big-Oh) of the `methodToTime(int N)` method in `ProvidedClass`. The parameter N represents the *problem size*. Thus, by iteratively calling `methodToTime` with successively larger values of N, you can collect timing data that is useful for characterizing the method's time complexity.

The constructor `ProvidedClass(int key)` will create a `ProvidedClass` object in which `methodToTime` is tailored specifically to the given key value. You must use your Banner ID number as the key required by the constructor. For example, one student might invoke the constructor `ProvidedClass(903111111)` and another student might invoke the constructor `ProvidedClass(903222222)`. This will create two distinct objects in which `methodToTime` will (likely) have different time complexities. Note that you **must** use your own Banner ID since grading will be based on Banner ID rather than student name.

You are guaranteed that for any key value used, the associated time complexity will be proportional to N^k for some positive integer k. Thus, you can take advantage of the following property of polynomial time complexity functions $T(N)$.

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

This property tells us that as N is successively doubled, the ratio of the method's running time on the current problem size to the method's running time on the previous problem size (i.e., $T(2N)/T(N)$) converges to a numerical constant, call it R, that is equal to 2^k, and thus $k = \log_2 R$.

As an example, consider the data shown in Table 1. Each row in this table records data for a given run of some method being timed. The first column (N) records the problem size, the second column (Time) records the time taken for the method to run on this problem size, the third column (R) is the ratio discussed above (i.e., $Time_i/Time_{i-1}$), and the third column (k) is $\log_2 R$. From Property 1 and Table 1 we can hypothesize that the method being timed has $O(N^4)$ time complexity.

To document your work you must write a lab report that fully describes the experiment used to discover the big-Oh time complexity of the given method. The lab report must discuss the experimental procedure (what you did), data collection (all the data you gathered), data analysis (what you did with the

Table 1: Running-time data and calculations

N	Time	R	k
8	0.04	—	—
16	0.08	2.25	1.17
32	0.84	10.37	3.37
64	7.59	9.03	3.18
128	113.56	14.97	3.91
256	1829.28	16.11	4.01
512	29689.21	16.23	4.02

data), and interpretation of the data (what conclusions you drew from the data). You are required to use `System.nanoTime()`, as illustrated in `ProvidedClient.java`, to generate timing data like that shown in Table 1. Time values must be expressed in seconds. The lab report must be well written and exhibit a high degree of professionalism. Your experiment must be described in enough detail that it could be reproduced by someone else.

Using the provided `jar` file

To compile and run `ProvidedClient` and any of your own code that you might use for this assignment, you will need to include `provided.jar` on the classpath. You can do this from the command line or from within your IDE.

From the command line. In the following example, `>` represents the command line prompt and it is assumed that the current working directory contains both the source code and the jar file.

```
> javac -cp .:provided.jar ProvidedClient.java
> java -cp .:provided.jar ProvidedClient
```

From the jGRASP IDE.

1. Create a project for the assignment (suppose you call it ProjectX), and include `ProvidedClient.java` and any source code files that you create.
2. Click on *Settings* → *PATH/CLASSPATH* → *Project* → *<ProjectX>*.
3. Click on the *CLASSPATHS* button.
4. Click *New*.
5. In the "Path or JAR File" text box, use the *Browse* button to navigate to `provided.jar` and select it.
6. Click on *OK*.
7. Click on *Apply*.
8. Click on *OK*.