COMP 2210 Assignment 3

Perry Bunn

Abstract

Efficiency of algorithms is important for large scale applications like search and sort of very large data sets. Calculation may vary by speed for every computer but the base equation of the time takes should fall within a margin of error of each other.

1. **Problem Review**

We 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*).

$$T(N) \propto N_k \Longrightarrow R = \frac{T(2N)}{T(N)} = \frac{(2N)^k}{N^k} = 2^k \times \frac{N^k}{N^k} = 2^k$$

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*(2*N*)/*T* (*N*)) converges to a numerical constant, call it *R*, that is equal to 2*k*, and thus *k* = log2 *R*.

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., $Time_i/Time_{i-1}$)

2. **Experimental Procedures**

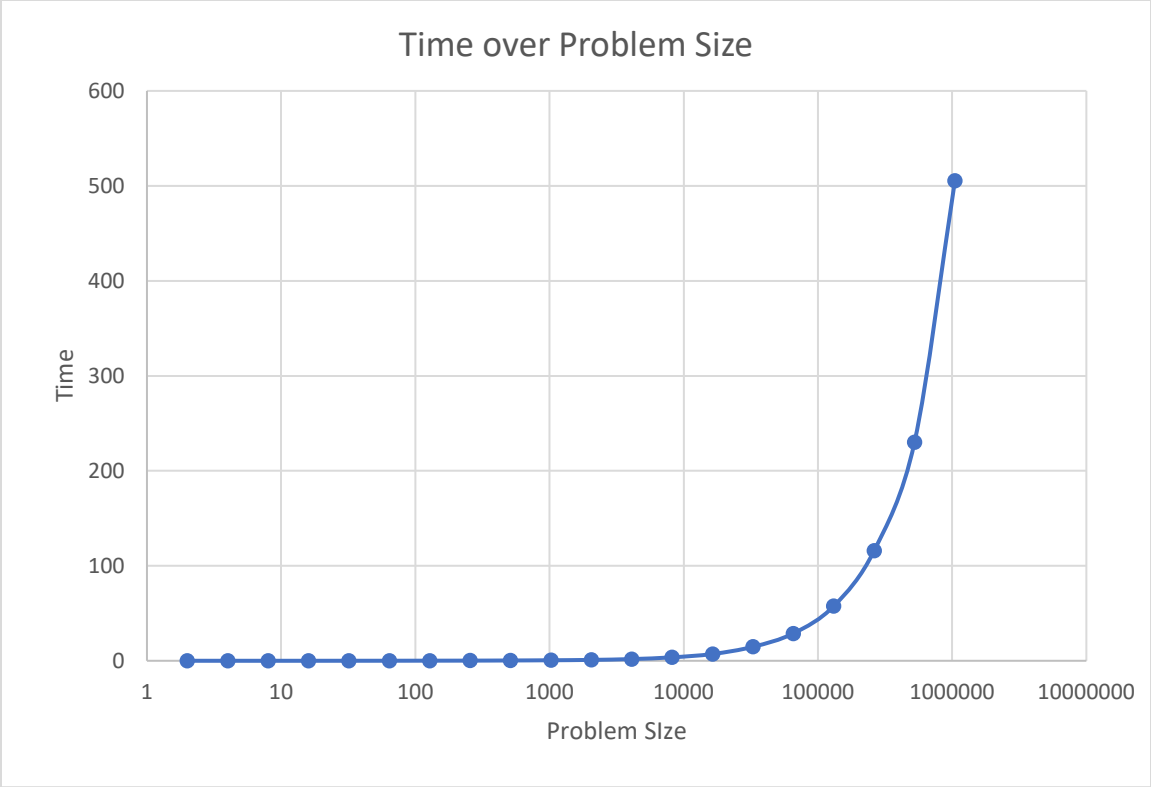
All calculations were performed in Intelij Idea Ultimate 171 on a computer with an Intel i7-4770k 3.4HGz, 8GBs of RAM, 64bit Windows 10.

I left my N at 2 for the entirety of the analysis, however I changed the amount of time that the program ran from 10 to 20 times. This gave me enough data points to confidently determine the big-Oh notation.

3. **Data Collection and Analysis**

The program would double N in each later run. As you can see, when the N size grows to 1048576, the elapsed time for that single run became 505.4 seconds. Since the running time becomes longer.

Problem Size	**Time**	Ratio	k
2	0.02101	-	-
4	0.007465	0.355307	-1.49286
8	0.004966	0.665252	-0.58803
16	0.009943	2.002127	1.001534
32	0.028341	2.850442	1.511186
64	0.055856	1.970837	0.978808
128	0.116334	2.082753	1.058492
256	0.169316	1.455423	0.541438
512	0.326471	1.928178	0.947238
1024	0.557663	1.708155	0.772439
2048	0.940201	1.685967	0.753577
4096	1.812993	1.928303	0.947332
8192	3.625098	1.99951	0.999646
16384	7.152134	1.972949	0.980354
32768	14.72049	2.058195	1.04138
65536	28.78363	1.955345	0.967423
131072	57.51942	1.998338	0.998801
262144	115.8664	2.014388	1.010342
524288	230.166	1.986477	0.990212
1048576	505.4362	2.195964	1.134855



The graph x-axis is logarithmic to show the scale of the problem size

4. Interpretation

To Find the big-Oh notation you must look at column K in the table, this column represents the logarithm or the result in the Ratio column. The Ratio column can be used to multiply the T(N) column which is in terms of seconds with three significant decimals, the result of such calculation is the expected result of the next iteration.

As we can see, after the first a few runs, the K value seems to converge to 1, also that the ratio converges around 2. We can hypothesize that the method being timed has $O(2N)$ time complexity. The plot of the running time versus size N seems to be a demonstration of my hypothesis.

Reference

1. Dr. Heandrix Dean, *Algorithm_Analysis.pptx*.