Yasemin Akın
22101782
Sec-02

CS 201, Spring 2023

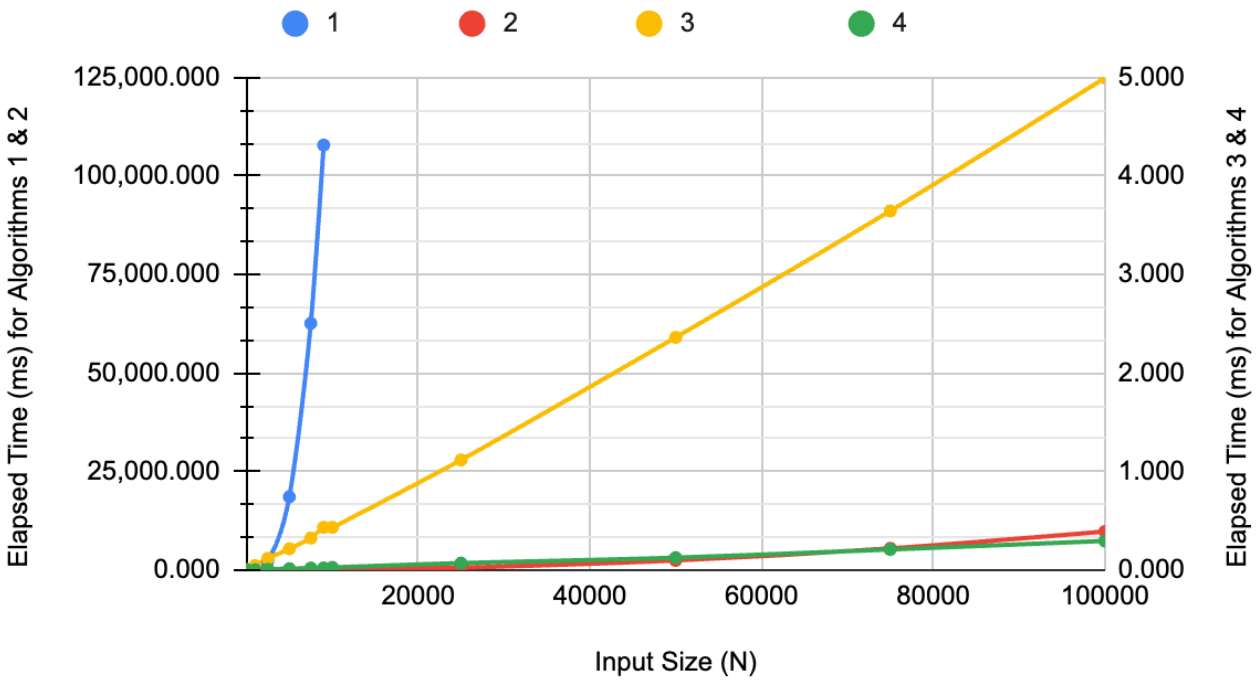
Homework Assignment 2

Comparison Table

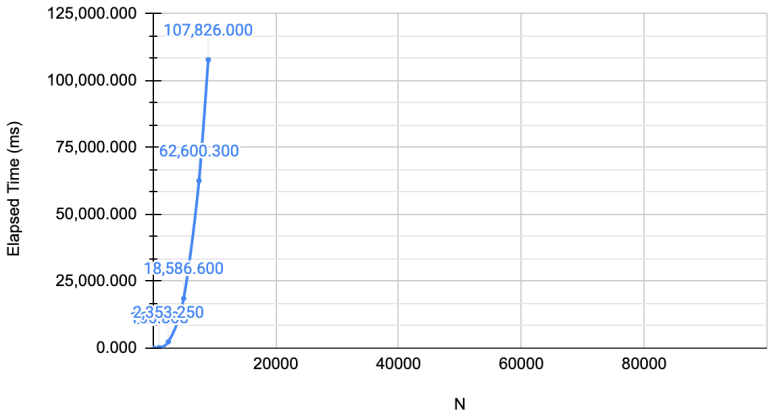
Algorithm-Run Time(ms) Table

Input Size \ Algorithm #	**1** $O(n^3)$	**2** $O(n^2)$	**3** $O(nlogn)$	**4** $O(n)$
10	0.001	0.001	0.001	0
50	0.036	0.004	0.003	0
100	0.240	0.015	0.005	0
1000	166.808	1.028	0.046	0.003
2500	2353.25	6.33	0.118	0.008
5000	18586.6	24.936	0.216	0.014
7500	62600.3	55.457	0.325	0.021
9000	107826.0	80.238	0.434	0.022
10000	NA	99.772	0.434	0.027
25000	NA	613.636	1.118	0.071
50000	NA	2446.09	2.362	0.126
75000	NA	5503.95	3.645	0.209
100000	NA	9788.48	4.995	0.296
500000	NA	NA	25.656	1.097
1000000	NA	NA	53.757	2.196
2000000	NA	NA	111.746	4.399
2500000	NA	NA	140.955	5.59
3000000	NA	NA	172.634	7.025
3500000	NA	NA	204.222	8.53

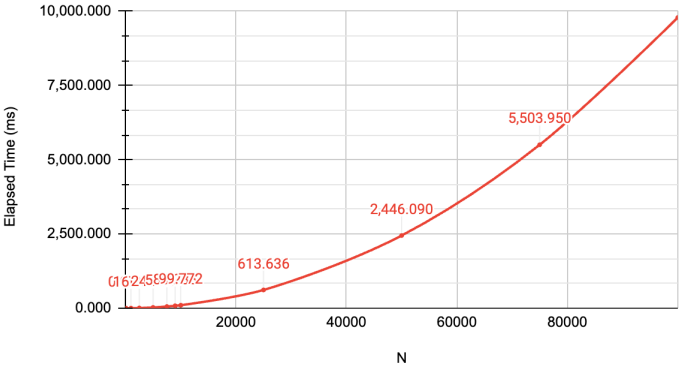
Various Algorithms vs. Experimental Run Time (ms) Plot



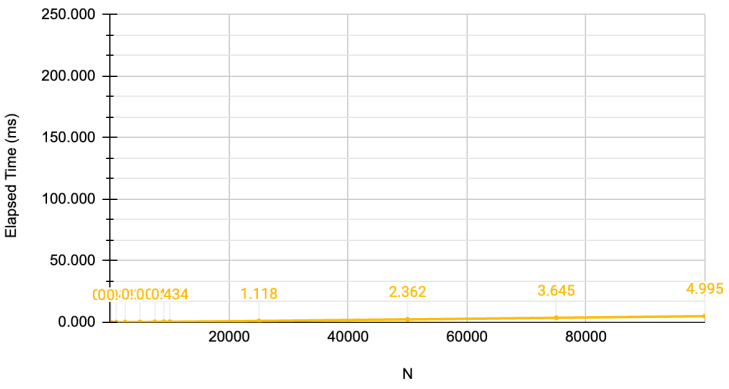
Maximum Subsequence Sum Algorithm 1 Plot



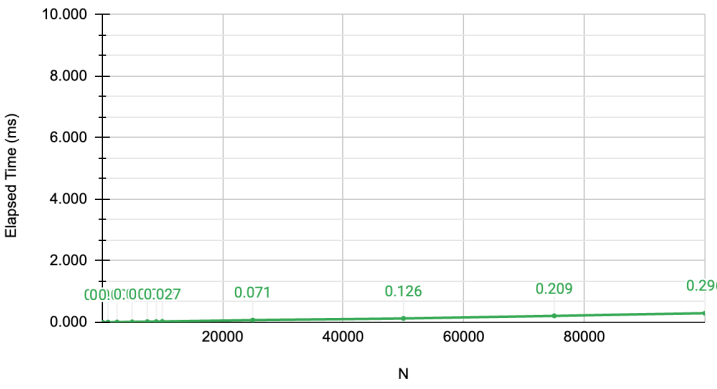
Maximum Subsequence Sum Algorithm 2 Plot



Maximum Subsequence Sum Algorithm 3 Plot



Maximum Subsequence Sum Algorithm 4 Plot



In the biggest plot above, the run time vs. input size curves of 4 different algorithms of the "Maximum Subsequence Sum Problem" are given with the help of the experimental data in the comparison table. In order to see the growth rates of each algorithm clearly, each algorithm's individual run time vs. input size graphics are also included above.

I would like to start by comparing the experimental results obtained by running the algorithms on my computer, whose specifications are given below, and the growth rates we see in the graphics. As can be seen from the first plot, the speed order of the algorithms is as follows: 4 > 3 > 2 > 1 and they have linear, logarithmic, quadratic and cubic growth rates, respectively. The growth patterns appearing in the graphs and the experimental data are in agreement with each other. In other words, if we look at the experimental data in the top comparison table, it will be seen that algorithm 4, which has a linear growth rate, gives much faster results than algorithm 1, which is seen to have a cubic growth rate run with the same input size.

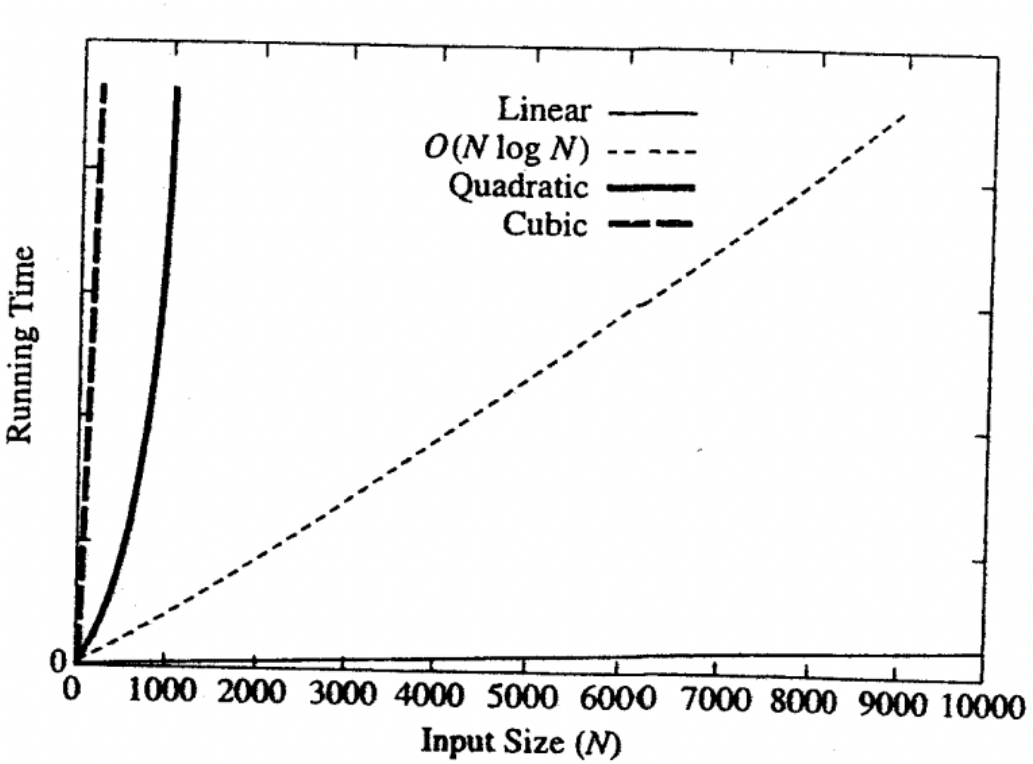


Figure 2.4 Plot (N vs. time) of various algorithms

I got the above black-and-white graphic from the distributed handout. This chart visualizes theoretical, expected growth rates. The algorithm numbers represented by the left-to-right curves are: 1, 2, 3, and 4, respectively. The expected

theoretical results in the handout and the experimental results agree. The growth rates specified for each algorithm in the handout are in line with the growth rates obtained from experimental runs of these algorithms. This fact can be understood by looking at the graphs formed from the experimental are actually very similar to the graph above. Experimental data, unlike theoretically calculated data, are not considered perfect and can be affected by many different criteria. There are many criterias for this, but I can explain a few of them as follows: Theoretical running times are calculated through mathematical studies of algorithms, which makes ideal assumptions like boundless memory and perfect hardware performance. Therefore, they are typically faster than empirical execution times. Empirical (real) running times, on the other hand, are based on real measurements of the algorithm's performance on a particular hardware and software setup, which might have restricted memory and other resources, and further delays. Because of this, empirical running times are both slower and more realistic than theoretical ones. The general conclusion to be drawn from these statements is that theoretical and empirical running times should both be taken into account when evaluating an algorithm's performance in real-world applications since they offer complimentary insights into the algorithm's strengths and weaknesses.

To summarize, the theoretical and experimental results are in an agreement with each other. Algorithms with cubic and quadratic growth curves are very slow or even do not work with large input sizes(this size varies, for example, as small as 10000 for cubic, 100000 for a quadratic algorithm in our example), while a logarithmic or a linear algorithm can reach results very quickly even with input sizes that are very very large (I am talking about millions). The elapsed time difference between the worst algorithm and the best algorithm is really big, so using the right algorithm when implementing is probably life-saving.

Computer Specifications

MacBook Pro
M2 Chip
8-Core CPU
16 GB RAM
macOS Monterey 12.5.1