CS201: Fundamental Structures of Computer Science I

HOMEWORK - 2

Maximum Subsequence Sum ProblemAnalysis

Zeynep Begüm Kara
22003880

CS201 - 02

May 5, 2023

Results

Computer Spesification: Windows 11, Intel(R)Core(TM) i5-10210U CPU 1.60GHz 2.11 GHz

In this study, it is aimed to observe the time complexity of four solutions for the Maximum Subsequence Sum Problem. This experiment is conducted on the same array of randomly generated sequences of integers to get more precise results to avoid the possible effects of the different distribution of numbers.

Due to theoretical knowledge, it was expected that the growth rate of algorithm - 1 to be O(N^3), algorithm - 2 to be O(N^2), algorithm - 3 to be O(N logN), and algorithm - 4 to be O(N) where N is the input size. This means the running time of those algorithms at most requires the time which is specified by function f(n) where f(n) is O(f(n)). More formally,

*Algorithm A is the order of f(n) if it requires no more than c * f(n) time units to solve a problem of size n > n_0.*

Therefore, to show that an algorithm is order of f(n) one must find appropriate c and n_0 so that c * f(n) represents an upper bound for running time or visually compare experimental and theoretical values through graphs.

Since algorithm-1 is order of O(n^3), even in the relatively small input sizes it requires a lot of time to compute whereas in the same small input sizes, algorithm-4, which is order of O(n), terminates in a very short time so that one cannot even measure it properly.

Experiment Results for size N = 2500 between N= 25000

```
-------------------------------------------------------------------------
                            Algorithm Time
-------------------------------------------------------------------------
              1             2             3             4
Input Size    O(N^3)        O(N^2)        O(N logN)     O(N)
-------------------------------------------------------------------------
N=2500        5458.20       6.02          0.00          0.00
N=5000        42515.55      24.94         0.00          0.00
N=7500        142454.18     56.36         0.00          0.00
N=10000       348594.99     103.13        1.13          0.00
N=12500       666930.13     155.96        1.00          0.00
N=15000       1138738.63    227.77        1.00          0.00
N=17500       1810953.98    303.40        1.00          0.00
N=20000       2702069.56    399.35        0.50          0.00
N=22500       3846856.20    504.56        1.23          0.00
N=25000       5279840.33    619.23        0.50          0.00
-------------------------------------------------------------------------

Process returned 0 (0x0)   execution time : 15986.875 s
Press any key to continue.
```

As it was expected in each size, the running time of algorithm - 1 is way much worse than the running time of algorithm - 2, 3, 4. Also as the input size increases, the running time of algorithm-1 increases dramatically. Even though when size is 2500 Algorithm - 1 tooks takes approximately 5.5 seconds, when size is 25000 almost takes 1.5 hours to compute. Similarly the running time of algorithm - 2 is worse than the running time of algorithm - 3, 4. Also as the running time of algorithm - 2 is quadratically proportional to the input size. However, even the biggest value n = 25000 took almost 0.6 seconds to compute, which is better than the running time of algorithm - 1 of the smallest input size n = 2500. Such dramatic difference results from input size and It can be argued that for smaller input sizes, the running time difference between algorithm 1 and 2 would be less. In algorithm 3, even though the time measured is shorter than in the first two algorithms, we see a fluctuation in measured time as the input size increases, which was unexpected. This error might have resulted from due to significant figures and sensitivity of time measurement, and also the other operations performed by the computer at the measurement time can damage the pure elapsed time measurement [1]. Also, the last algorithm terminates so quickly that we couldn't even measure those significant figures in our setup. To observe more accurate results of these algorithms a second test is performed with much bigger sized inputs below:

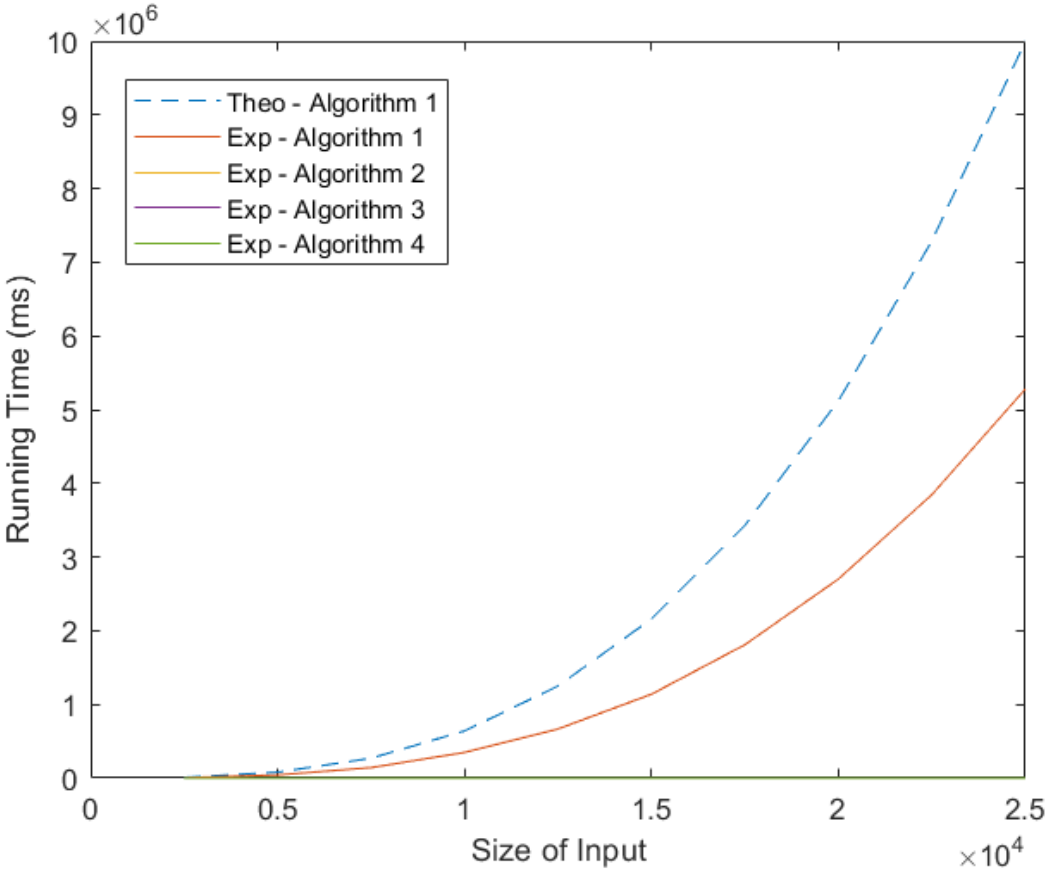
```
                              Algorithm Time
-------------------------------------------------------------------------
                     3                        4
Input Size           O(N logN)                O(N)
-------------------------------------------------------------------------
N=250000000          34948.8878               719.2839
N=300000000          41957.0627               799.7265
N=350000000          34458.9295               841.4474
N=400000000          36458.5980               943.0145
N=450000000          43925.0271               1038.1163
N=500000000          48796.0985               1190.3549
N=550000000          54760.9464               1320.1227
N=600000000          59019.9462               1444.2488
N=650000000          62043.0774               1332.3373
N=700000000          67806.0554               1702.0222
-------------------------------------------------------------------------

Process returned 0 (0x0)   execution time : 571.091 s
Press any key to continue.
```

Note that for this input size, the first two algorithms are not executed since it would require many hours (possibly even days). Even though there is a minor fluctuation in algorithm 3 which might again be caused by [1], the overall running times of the two algorithms increases as N increases as expected. For each input size, the fourth algorithm performed better than algorithm 3, which was consistent with our theoretical expectations.

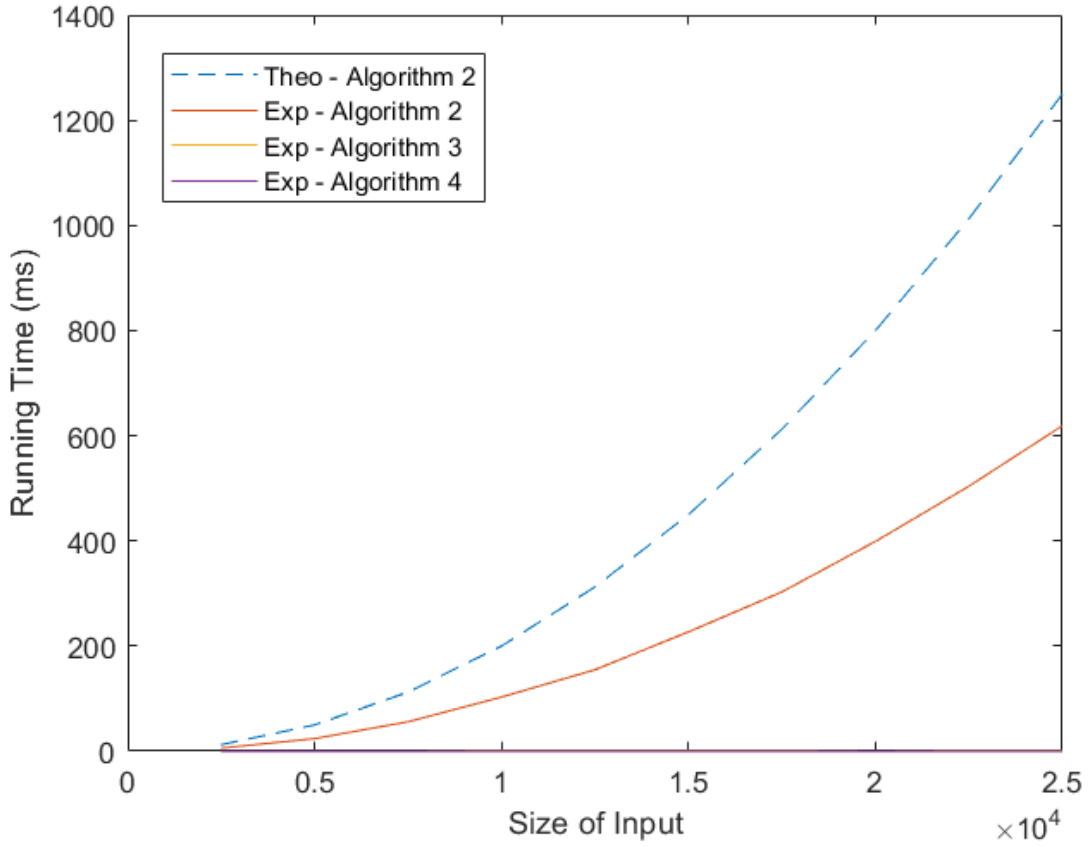
Note that in each algorithm, if we had examined much bigger sized arrays, the results would be more consistent with and closer to the theoretical values.

Plot 1: Theoretical vs Experimental Comparison of Algorithm 1



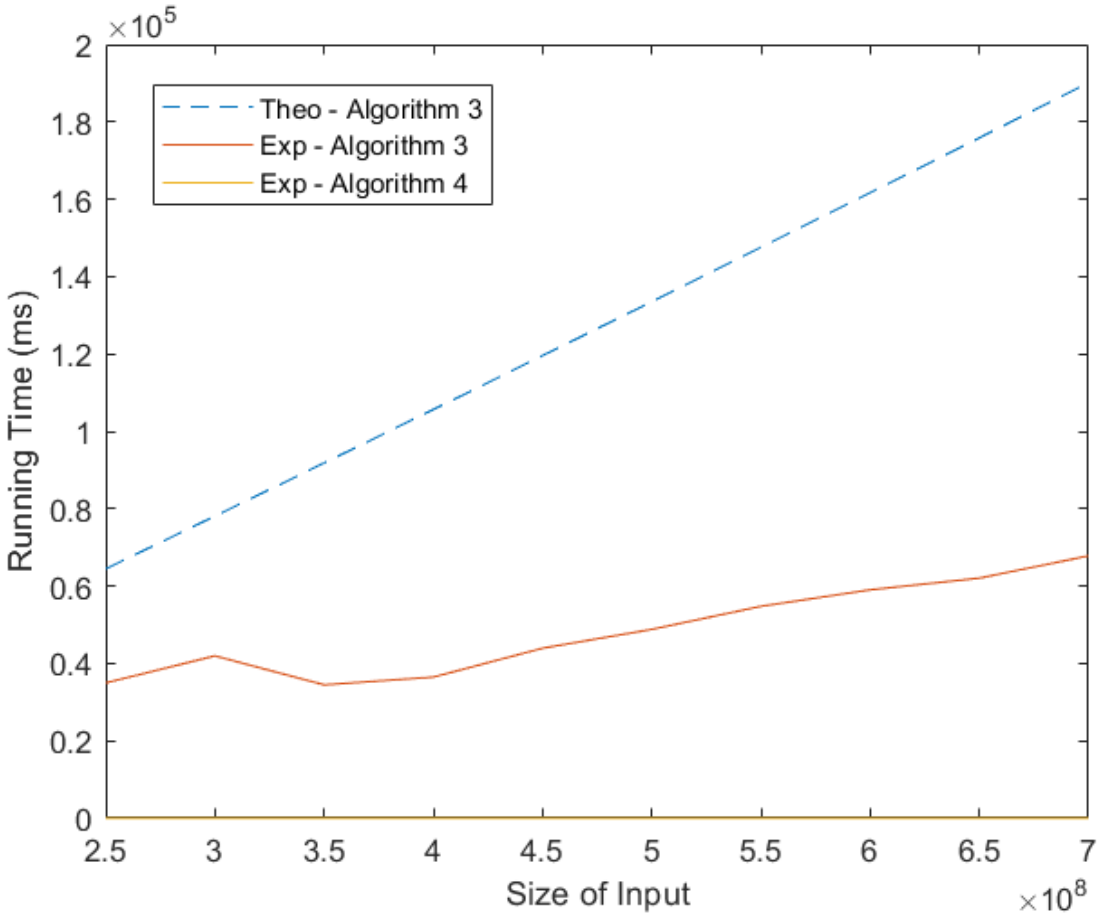
In plot-1, the elapsed time vs size graph is shown for 4 algorithms. Continuous lines represent experimental values whereas dashed line represent a function that has the complexity O(N^3), which indicates the worst-case growth rate for algorithm 1. For our graphical purposes, the theoretical expectation is chosen to be as f(n) = n^3 / 1562500, c = 1/1562500. We clearly see that the experimental results function behaves like a cubic function and lies below this line n > 5000. Hence, one can conclude that algorithm 1 is the order of N^3, as expected.

Plot 2: Theoretical vs Experimental Comparison of Algorithm 2



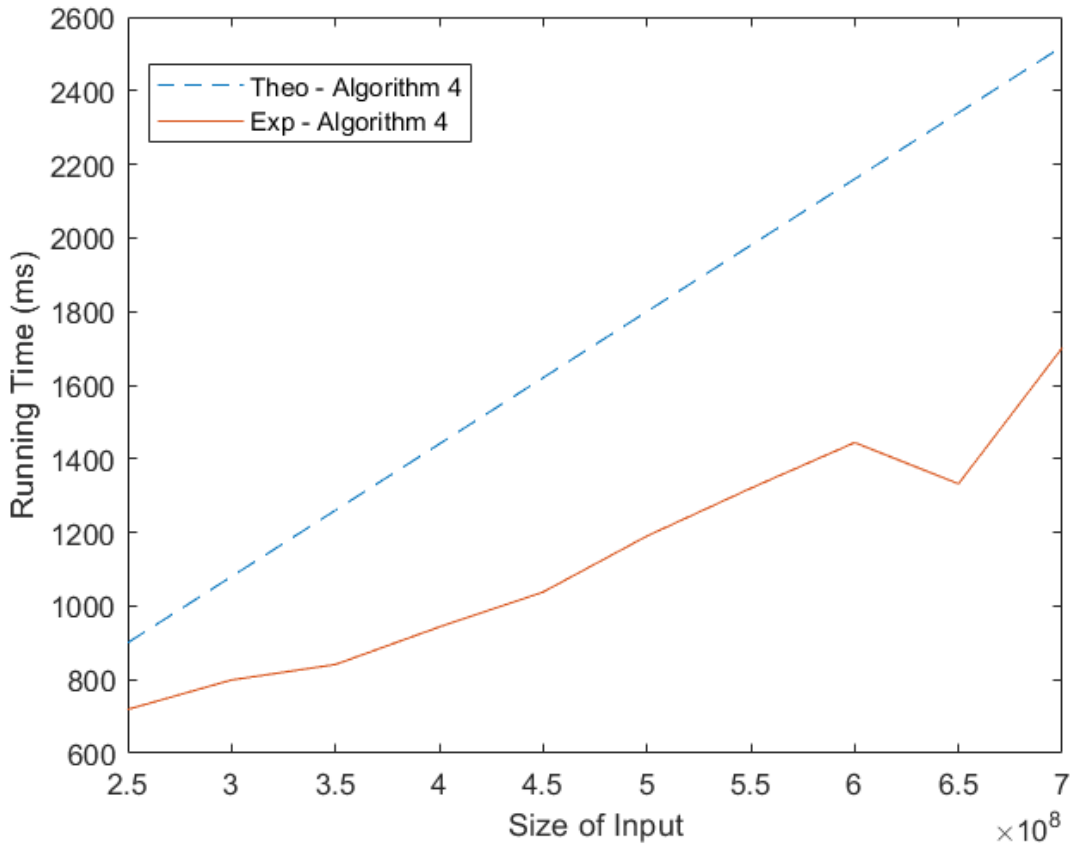
In the plot-2, the elapsed time vs size graph is shown for 3 algorithms. Continuous lines represent experimental values whereas dashed lines represent a function that has the complexity O(N^2), which indicates the growth rate for algorithm 2. For our graphical purposes, the theoretical expectation function is chosen to be as f(n) = n^2 / 97656, c = 1 / 97656. We clearly see that the experimental results function behaves like a quadratic function and lies below this line n > 5000. Hence, one can conclude that algorithm 2 is order of N^2, as expected.

Plot 3: Theoretical vs Experimental Comparison of Algorithm 3



In the plot-3, the elapsed time vs size graph is shown for 2 algorithms. Continuous lines represent experimental values whereas dashed lines represent a function that has the complexity O(N*logN), which indicates the growth rate for algorithm 3. For our graphical purposes, the theoretical expectation function is chosen to be as f(n) = c * n * logn, c = 50862727. One can get a more appropriate, i.e. strict upper bound by choosing a different c. The overall structure of the experimental results function looks like directly proportional to n* logn and lies below the theoretical line when n > 250000000. Hence, one can conclude that algorithm 2 is order of N * log N, as expected. See [1] for reasons of fluctuations in experimental values. To reduce such fluctuations one can work with bigger-sized inputs and stop other activities in the computer, which will end up with closer values to theoretical ones.

Plot 4: Theoretical vs Experimental Comparison of Algorithm 4



In plot-4, the elapsed time vs size graph is shown. Continuous lines represent experimental values whereas dashed lines represent a function that has the complexity O(N), which indicates the growth rate for algorithm 4. For our graphical purposes, the theoretical expectation function is chosen to be f(n) = c * n, c = 1 / 277777. Except for the error which may be caused by [1], we see that size of the input is linearly proportional to running time. Also, the experimental results graph lies below this line where n > 250000000, which represents that it is better than the theoretical worst-case expectation. Hence, one can conclude that algorithm 2 is order of N, as expected. See [1] for reasons of fluctuations in experimental values. To reduce such fluctuation one can work with bigger-sized inputs and stop other activities in the computer, which will end up with closer values to theoretical ones.