PRACTICAL 1

Javier Escribano Salgado (javier.escribano)
Martín Azpilcueta Rabuñal (m.azpilcueta)
Group 6.1

INTRODUCTION

The purpose of this practice is to find the sum of the maximum subsequence of a given set of numbers using two algorithms. It is asked to make sure that both algorithms return the same values for the same set of input numbers, even though the algorithms work differently and have different performance.

Besides, it is also asked to perform an analysis of the obtained results by means of empirical verification of the theoretical complexity.

Moreover, perform an empirical check using an underestimated and overestimated bound for each algorithm.

Time Units:

Microseconds (μs)

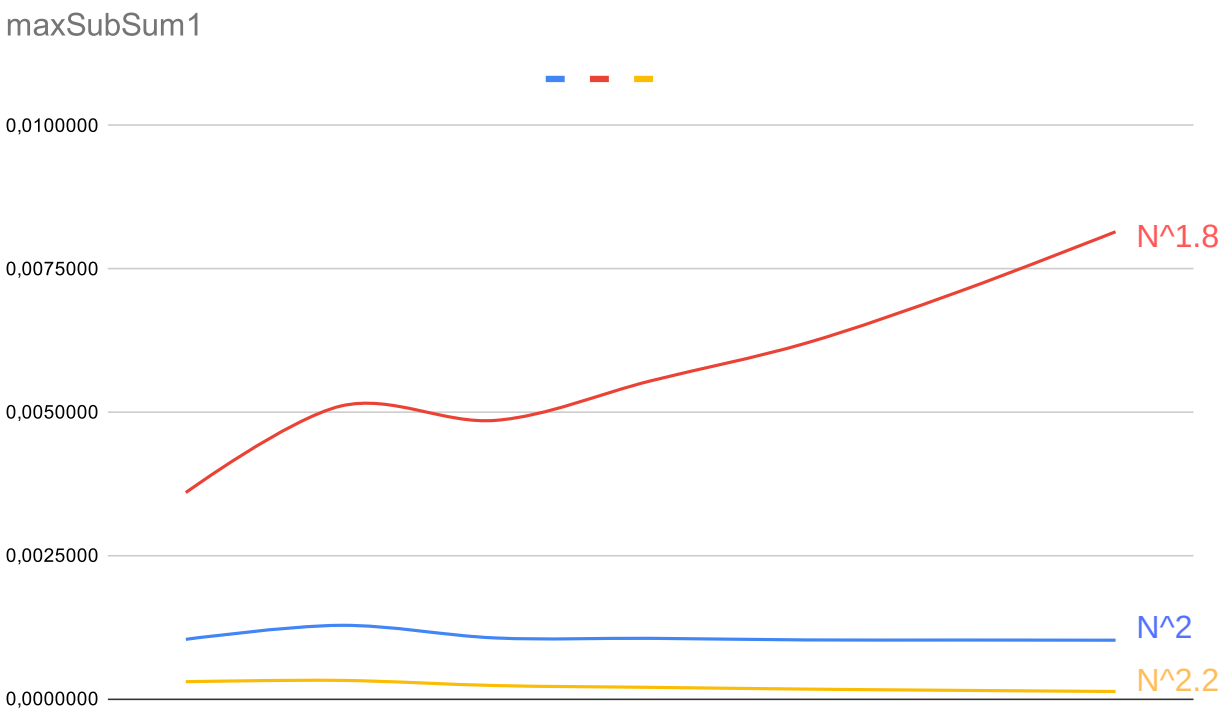
Machine Used:

HP ENVY - 13-aq0001ns:

- Intel® Core™ i5 8265U (1,6 GHz Base Frequency, max 3,9 GHz with Intel® Turbo Boost technology, 6 MB Cache, 4 cores)
- SDRAM DDR4-2400 (8 GB)
- SSD 512 GB PCIe® NVMe™ M.2
- Running on Windows 10. File compiled with Ubuntu 18.04 Subsystem

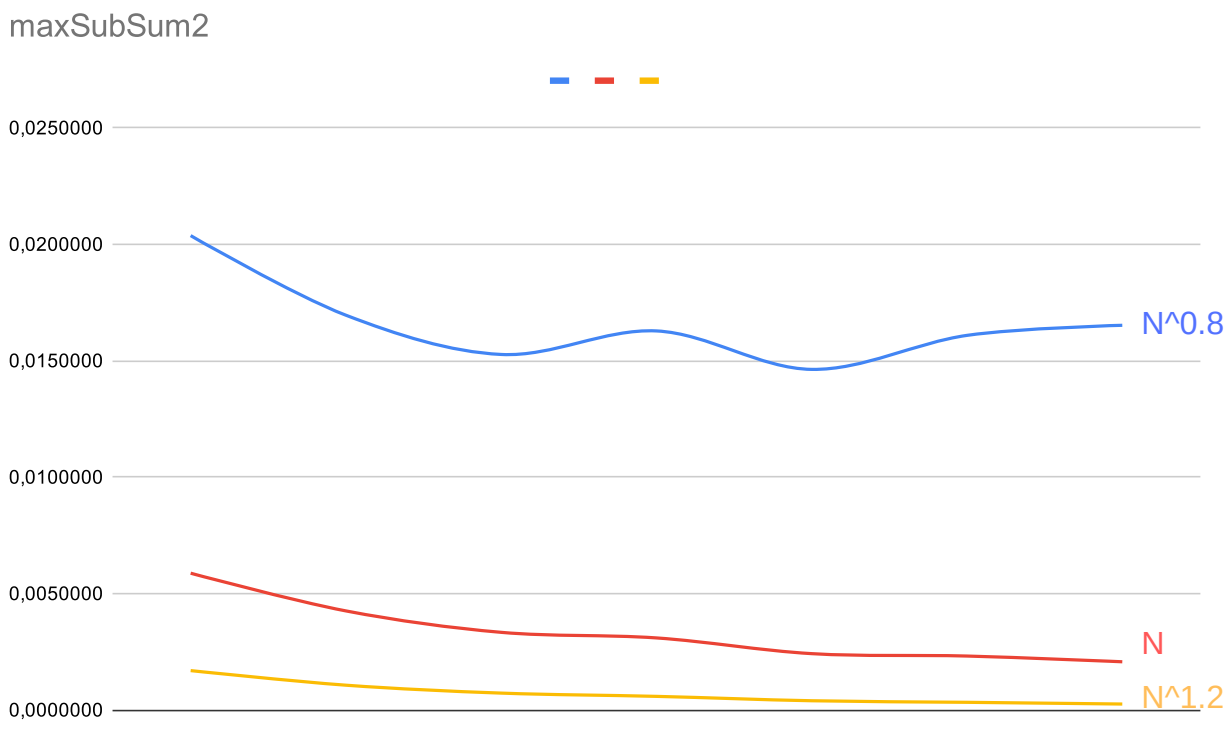
maxSubSum1

N	T(N)	T(N)/N^1.8	T(N)/N^2	T(N)/N^2.2
500	259.53 *	0.0035978	0.0010381	0.0002995
1000	1283.00	0.0051077	0.0012830	0.0003223
2000	4249.00	0.0048577	0.0010623	0.0002323
4000	16895.00	0.0055469	0.0010559	0.0002010
8000	65793.00	0.0062032	0.0010280	0.0001704
16000	262728.00	0.0071136	0.0010263	0.0001481
32000	1047856.00	0.0081476	0.0010233	0.0001285
		Underestimated	Tight	Overestimated
			C = 0.0010738	
			K = 1000	



maxSubSum2

N	T(N)	T(N)/N^0.8	T(N)/N	T(N)/N^1.2
500	2.94*	0.0203785	0.0058800	0.0016966
1000	4.26*	0.0169594	0.0042600	0.0010701
2000	6.68*	0.0152740	0.0033400	0.0007304
4000	12.40*	0.0162845	0.0031000	0.0005901
8000	19.40*	0.0146329	0.0024250	0.0004019
16000	37.15*	0.0160940	0.0023219	0.0003350
32000	66.42*	0.0165264	0.0020756	0.0002607
		Underestimated	Tight	Overestimated
			C = 0.0033432	
			K = 1000	



CONCLUSIONS:

After the empirical analysis of both algorithms, we have found out a great performance difference between them. This is:

Regarding the first algorithm, we estimate it is of the type O(n^2) since it traverses a matrix n*n and depends on n*n values. Consequently, we assess the functions t(n)/n^1.8 and t(n)/n^2.2 as underestimated and overestimated bounds, respectively.

As we can see in the graphs and tables, the value of the measurements for t(n)/n^2 tends to a constant (C = 0.0010738) when increasing the number of iterations.

On the other hand, we evaluate the second algorithm as O(n), since it depends on n values traversing an array of n numbers. As a result, we estimate the functions t(n)/n^0.8 and t(n)/n^1.2 as underestimated and overestimated bounds, respectively.

Nevertheless, the practice statement itself already provides the notation of both algorithms.

Notes:

After executing the program several times we have discovered that at certain times the algorithms can return a negative value as the time of execution. From all the non-negative time executions, which were the majority of the cases, we have selected these examples.

This file has been compiled using the command:
gcc nameofthefile.c -Wall -lm -o outputname