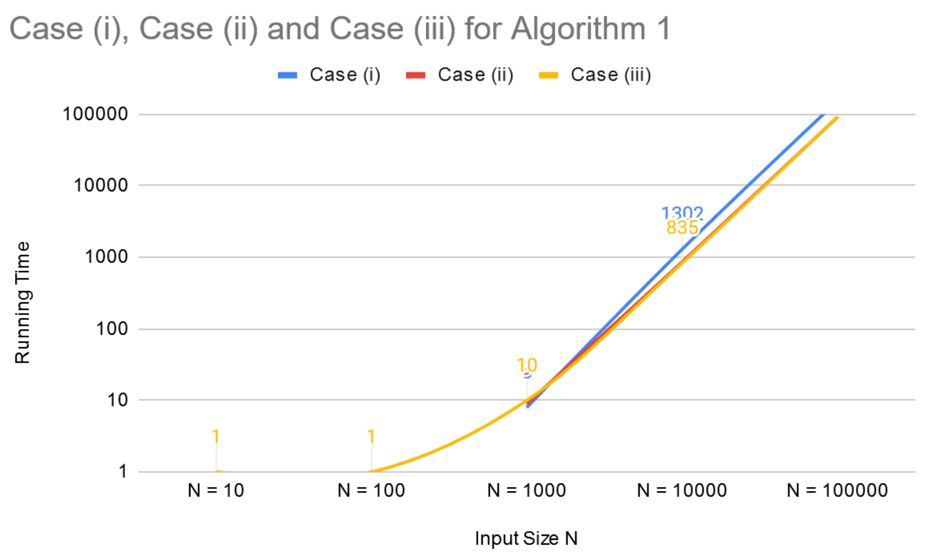
Algorithm 1 Time (ms)

Input Size	$O(n^2)$	$O(n^2)$	$O(n^2)$
N	Case (i)	Case (ii)	Case (iii)
N = 10	0	0	1
N = 100	0	0	1
N = 1000	8	8	9
N = 10000	1302	866	835
N = 100000	153757	91717	92621



Best case: Case (iii)

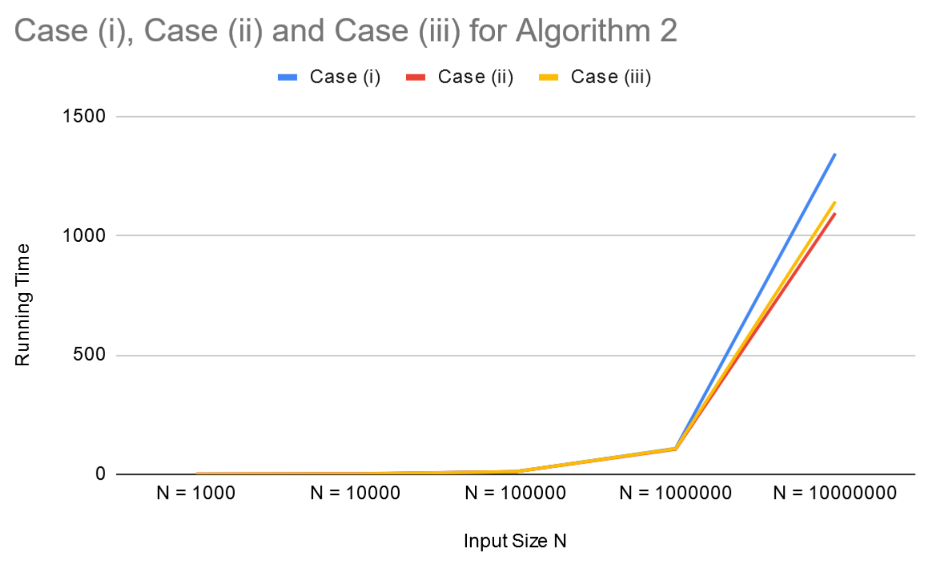
Average case: Case (ii)

Worst case: Case (i)

Worst case time complexity: $O(n^2)$

Algorithm 2 Time (ms)

Input Size	$O(n)$	$O(n)$	$O(n)$
N	Case (i)	Case (ii)	Case (iii)
N = 1000	0	1	0
N = 10000	1	2	1
N = 100000	11	11	11
N = 1000000	108	105	107
N = 10000000	1344	1095	1143



Best case: Case (ii)

Average case: Case (iii)

Worst case: Case (i)

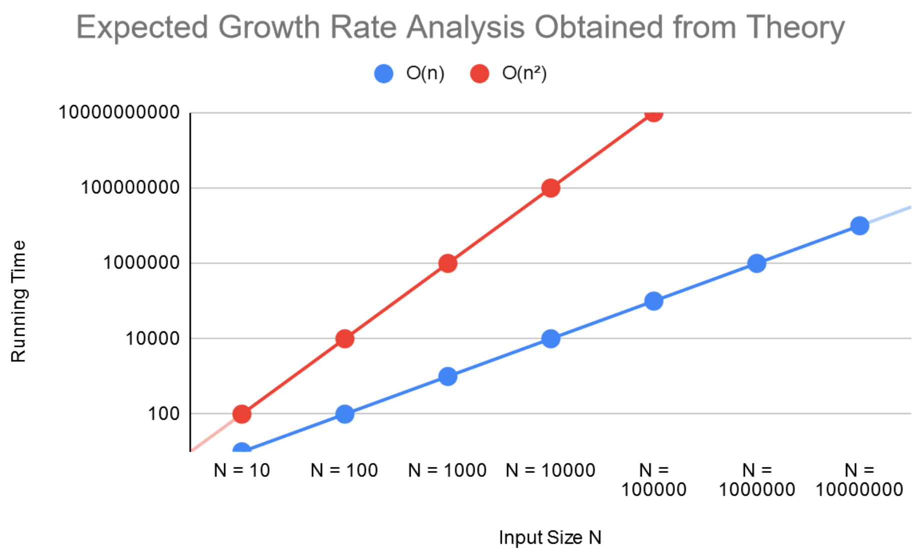
Worst case time complexity: O(n)

Computer Specifications

Processor: Intel(R) Core(TM) i7-2630QM CPU @ 2.00GHz 2.00 GHz

RAM: 8 GB

OS: Microsoft Windows 10 Home



In both cases, the trends of experimental and theoretical results were similar. In the experiment, randomly generated numbers were assigned to arrays for two algorithms with a difference in run time and tested. The results were written separately for each case, with the input numbers on the x-axis and the run times in milliseconds on the y-axis. Expected growth rates vary according to Big O complexities. In this case, the working time of O (n^2) increases much faster than that of O (n). O (n) shows a linear increase compared to O (n^2). Although there is no significant difference initially for small input sizes, the difference between cases and algorithms becomes clear as the worst-case scenario is approached. In terms of cases, case i runs a click slower than other cases. Case ii and iii move close to each other. When evaluated in terms of algorithms, Algorithm 1 works much slower than Algorithm 2, which includes a while loop, since it includes a nested for loop. In this case, it is much more logical for a programmer to use algorithm 2, as he will use big data, considering the running time. However, it should not be forgotten that while gaining, on the one hand, there will always be losses on the other. For example, in terms of runtime and space.