Maryam Shahid

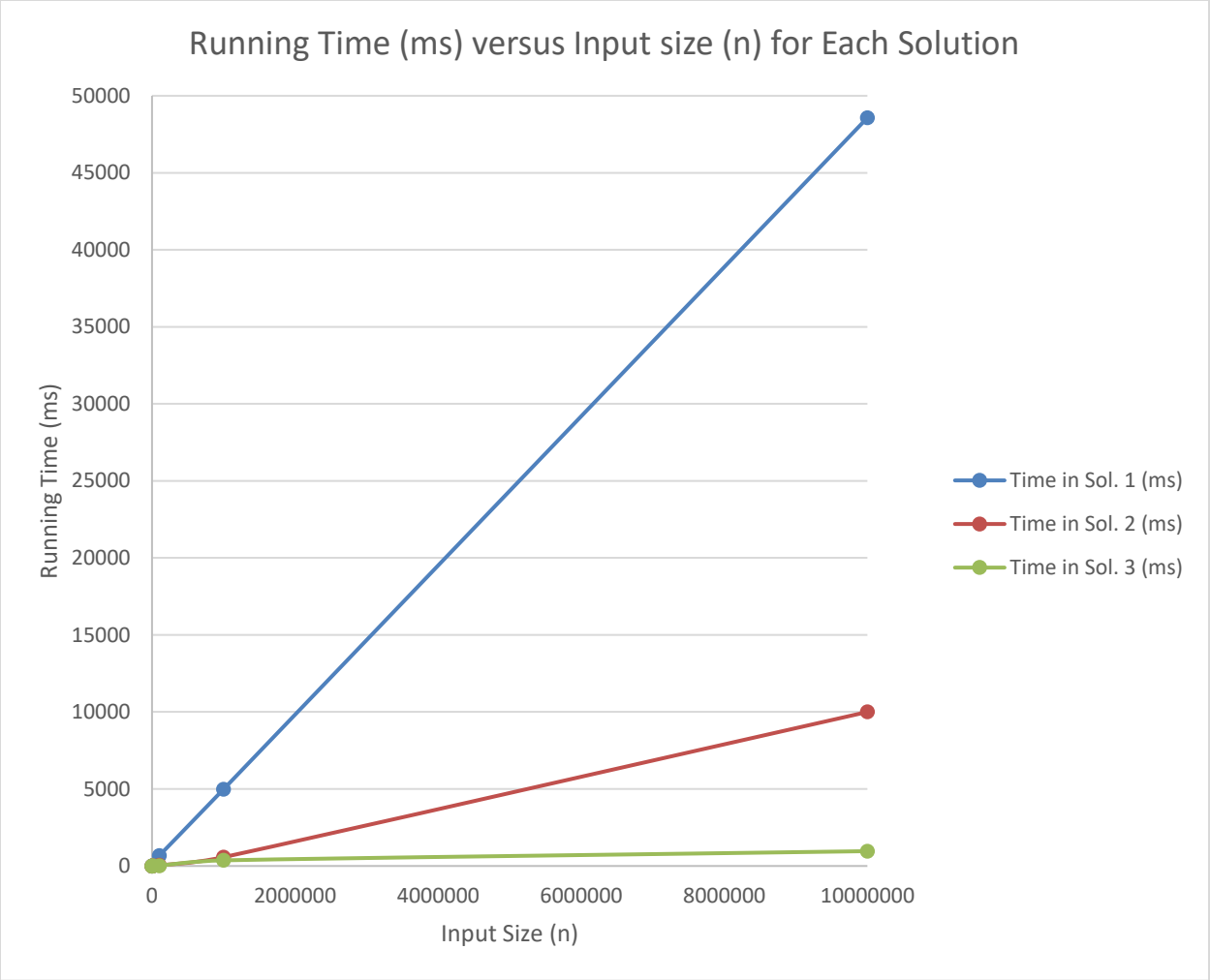
21801344

Date: 21/04/2020

CS 201

Input Size (n)	Time in Sol. 1 (ms)	Time in Sol. 2 (ms)	Time in Sol. 3 (ms)
100	0	0	0
1000	0	0	1
10000	33	6	2
100000	667	52	14
1000000	4984	573	365
10000000	48572	10000	962

Table 1 – with k fixed at 1000



Graph 1 – Running time (ms) versus input size (n) for each solution

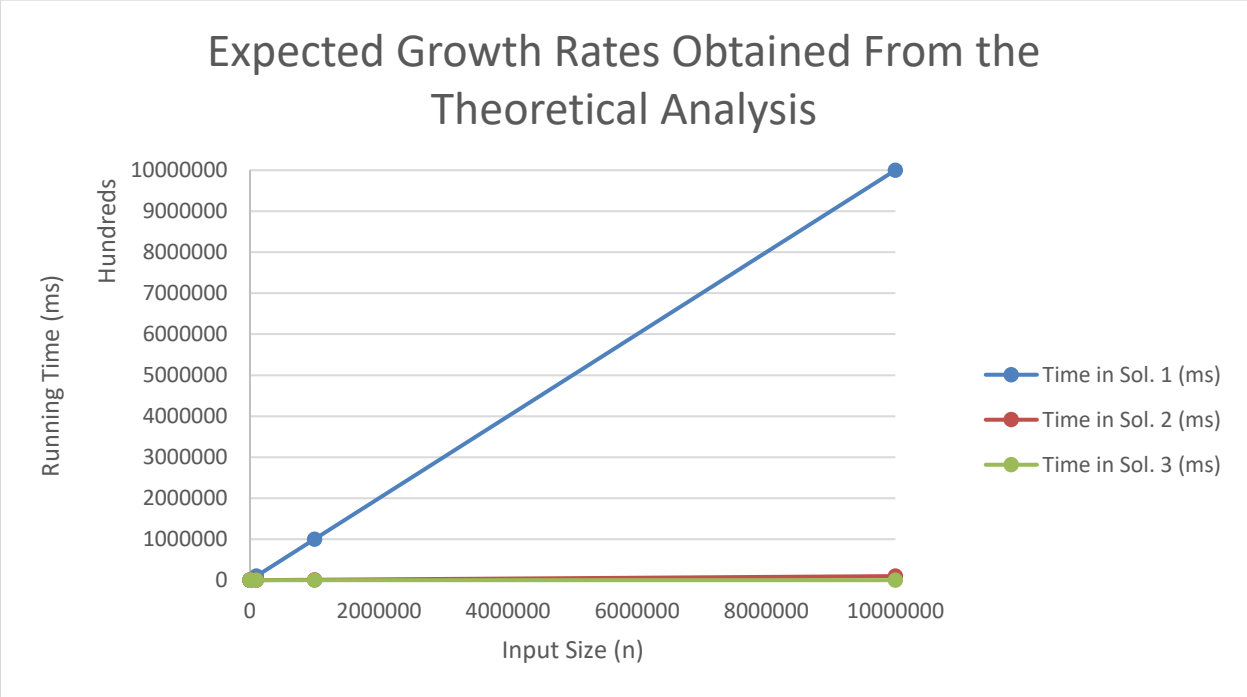
Computer Specifications:

Processor:	Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz 1.99 GHz
Installed memory (RAM):	16.0 GB (15.9 GB usable)
System type:	64-bit Operating System, x64-based processor
Pen and Touch:	Pen and Touch Support with 10 Touch Points

Theoretical Analysis:

Input Size (n)	Time in Sol. 1 (ms)	Time in Sol. 2 (ms)	Time in Sol. 3 (ms)
100	10000	100	2000
1000	100000	1000	3000
10000	1000000	10000	4000
100000	10000000	100000	5000
1000000	100000000	1000000	6000

Table 2 – Theoretical analysis (as given for each algorithm) by using the same k and n values



Graph 2 - Expected Growth Rates Obtained From the Theoretical Analysis

Conclusion

The time complexity for solution 1 is O(kn). This is the least efficient algorithm since it uses a binary approach. We observe the same pattern in our code as we expect in the theoretical analysis. Solution 2's time complexity is O(nlogn). This takes more time than expected in the start. This is due to the way the function is implemented. For example, the method used to sort elements in descending order dominates the time complexity of the function. Thus making it less efficient. Similarly, solution 3's time complexity is O(n). The graph showing the time taken to run my code fairly follows the theoretical analysis and matches the expected time to run. This is the most efficient algorithm since it searches for the largest numbers directly and does not take time to sort the array in any way. Whereas solution 1 and 2 take time to sort the arrays in their respective ways which hinders the runtime.