Gün Kaynar
22101351

Q1)

The first algorithm (simple) contains two for loops in a nested manner. The first loop will perform m times the inner loop which performs n times, where n is the size of the first array and m is the size of the second array. The time complexity will be O(m*n) in this case.

The second algorithm (sorting and binary search) has helper functions like quicksort and binary search. The quicksort function takes the first array as input and sorts it in O(n^2) in the worst case, but in average it is O(n*log(n)). The binary search then takes each element of the second array with size m and searches in the first array. Each step of the search will take O(log(n)) time, but overall function needs m steps, so that the overall complexity of the binary search will be O(m*log(n)). We know that n > m, so that O(n*log(n)) + O(m*log(n)) = O(n*log(n)), however in the worst case the quicksort will give O(n^2). So, the overall algorithm works in O(n^2).

The third algorithm (frequency map) will take the first array with size n and map its elements to the number of times its elements appear. This step has time complexity O(n). Afterwards, it will decrease the frequencies of the elements by looping through the second array with size m. This process will take O(m) time. Since n > m, O(n) + O(m) = O(n). This algorithm seems to work very fast compared to the first two algorithms. However, in C++, mapping process is overly inefficient, thus this algorithm will take much more time regardless of its good time complexity.

Q2)
The testing is done on MacBook Air 2020 with M1 chip. The system specifications are as follows:
CPU: Apple M1 chip 8-core CPU with 4 performance cores and 4 efficiency cores
GPU: 7-core GPU 16-core with Neural Engine
Memory: 8GB
Storage: 256GB SSD

Q3)

n	Algorithm 1		Algorithm 2		Algorithm 3	
	$m = 10^3$	$m = 10^4$	$m = 10^3$	$m = 10^4$	$m = 10^3$	$m = 10^4$
10^5	1 ms	1 ms	11 ms	11 ms	56 ms	51 ms
2×10^5	1 ms	1 ms	22 ms	23 ms	96 ms	96 ms
5×10^5	3 ms	20 ms	62 ms	61 ms	218 ms	222 ms
8×10^5	193 ms	1364 ms	104 ms	106 ms	378 ms	349 ms
10^6	220 ms	1787 ms	142 ms	138 ms	471 ms	427 ms
2×10^6	228 ms	2213 ms	307 ms	310 ms	829 ms	832 ms
3×10^6	230 ms	2191 ms	500 ms	508 ms	1353 ms	1242 ms
4×10^6	218 ms	2204 ms	750 ms	757 ms	1624 ms	1654 ms
5×10^6	211 ms	2196 ms	1019 ms	1036 ms	1999 ms	2041 ms
6×10^6	222 ms	2194 ms	1337 ms	1341 ms	2424 ms	2417 ms

Q4)

The plots are prepared.

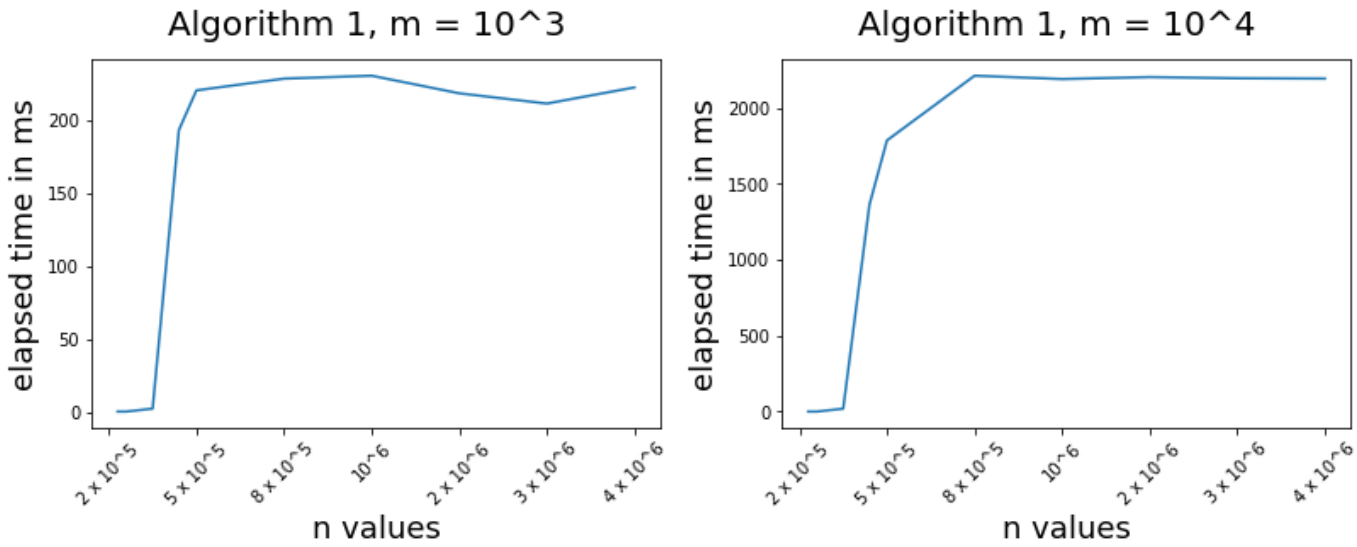

Figure 1 shows the time spent on finding whether array 2 with size m is a subset of array 1 with size n. The n values can be seen on x-axis, and the elapsed time for algorithm 1 in milliseconds can be seen on y-axis. Two different plots are prepared for two different m values.

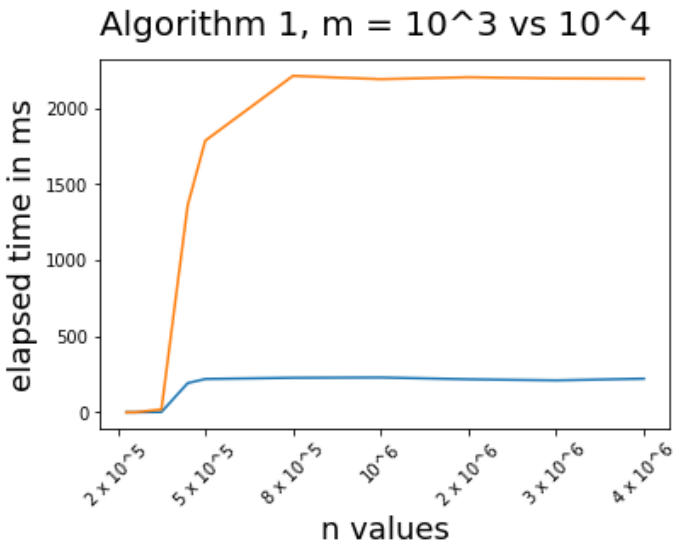

*Figure 2 compares the algorithm 1's elapsed time in milliseconds for m = 10^3 and m = 10^4. The time complexity of this algorithm is O(m*n), so that with m changing, it makes sense to draw the plot with both m values.*

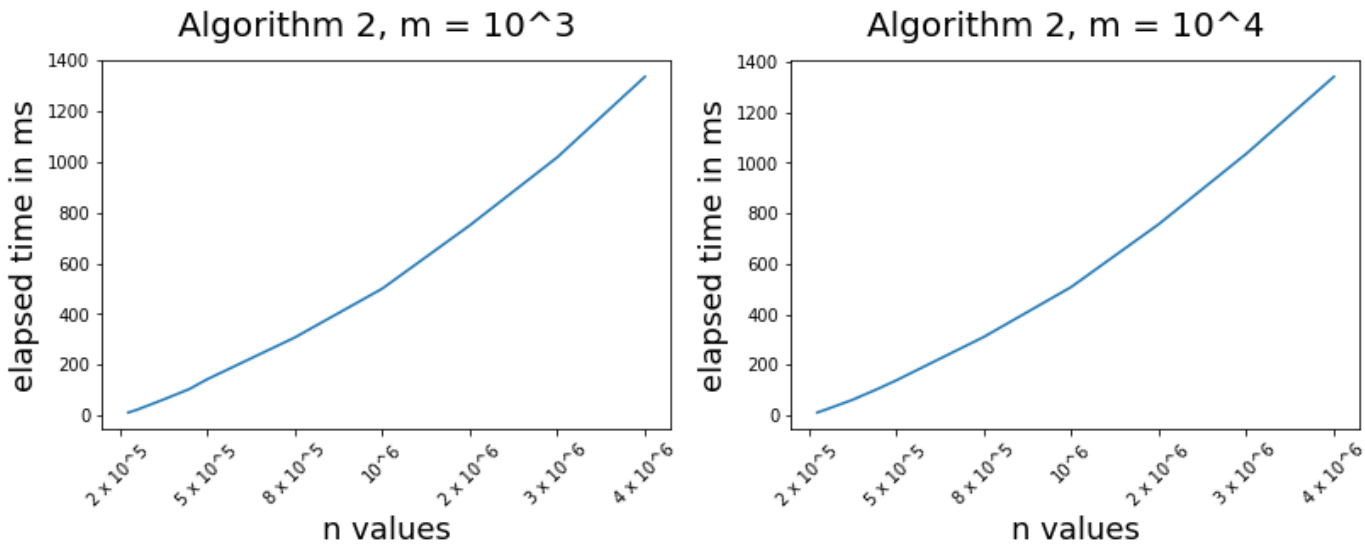

Figure 3 shows the time spent on finding whether array 2 with size m is a subset of array 1 with size n. The n values can be seen on x-axis, and the elapsed time for algorithm 2 in milliseconds can be seen on y-axis. Two different plots are prepared for two different m values.

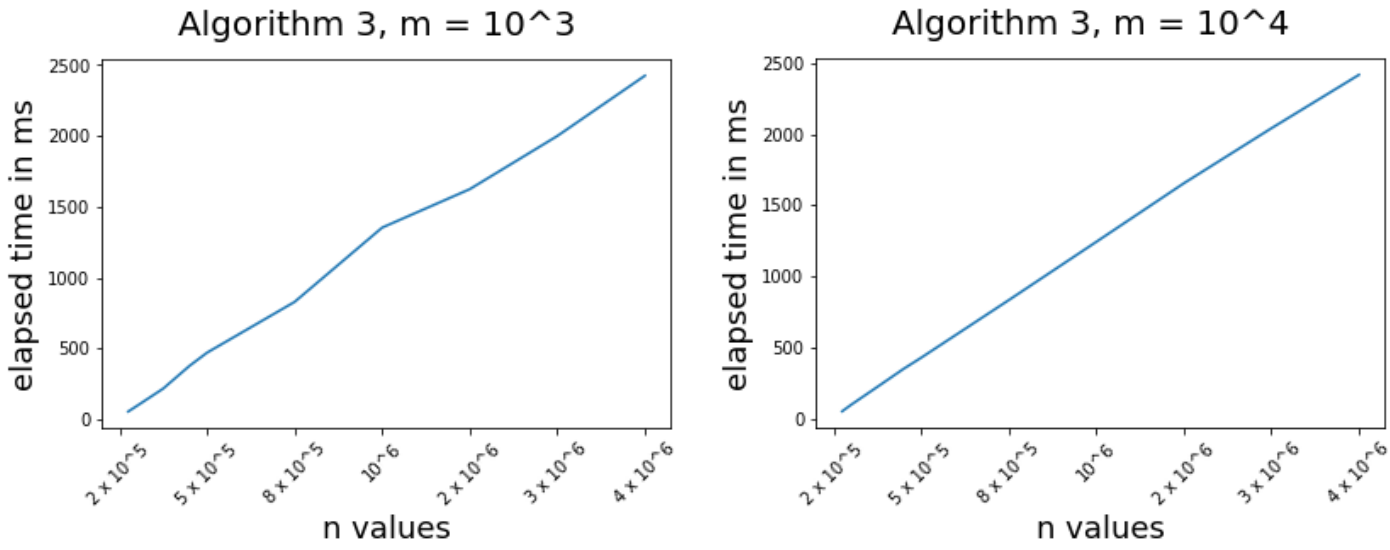

Figure 4 shows the time spent on finding whether array 2 with size m is a subset of array 1 with size n. The n values can be seen on x-axis, and the elapsed time for algorithm 3 in milliseconds can be seen on y-axis. Two different plots are prepared for two different m values.