Barış Tan Ünal | 22003617
Section: 02 | 07.12.2021

CS 201, FALL 2021
HOMEWORK ASSIGNMENT 2

DISCUSSION

All algorithms for finding out if an array is a subset of another given array does the same job when the elements of the smaller array are unique (otherwise only the third algorithm can work properly), but they differ in time complexities. It is all about the number of traversals through the given arrays and how much longer they take to execute when the size of the arrays increases.

Note that array 1 is the bigger and sorted array, whereas array 2 is the smaller one which consists random numbers. Size of the first array is n. Size of the second array is m.

Algorithm 1: The first algorithm which performs linear search, performs a single complete traversal over array 1 for each element in array 2. This means it goes over the size of the first array times the size of the second array, n*m elements. Consequently, time complexity of the first algorithm is O(n*m).

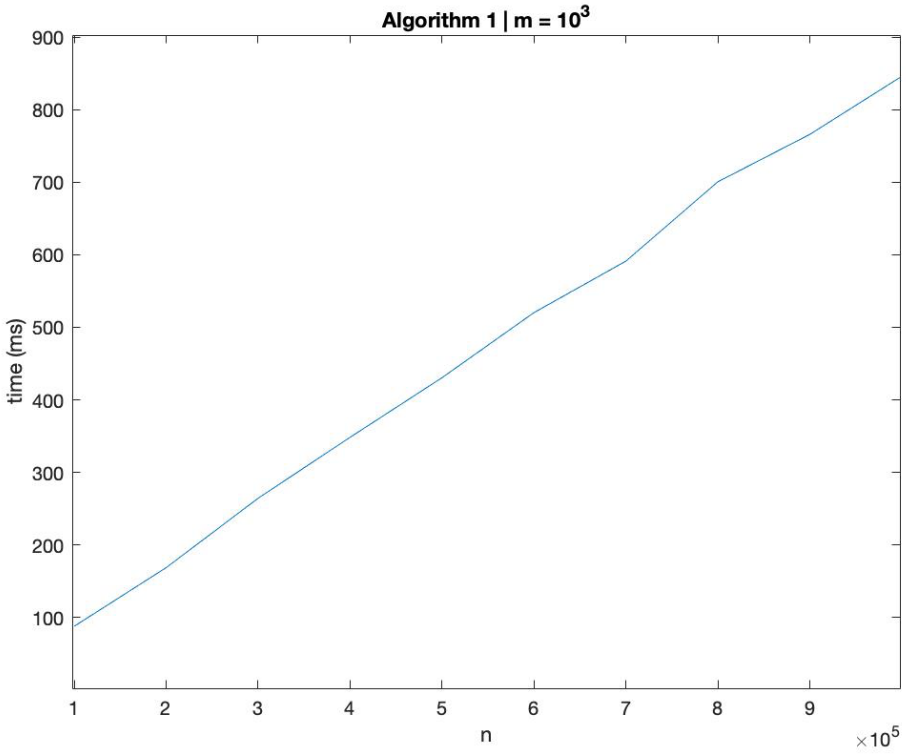
Algorithm 2: From the graphs and table below, it is clear that algorithm 2 is not only faster than the first algorithm but also when we give bigger arrays, the increase in time is much smaller too. Their diversity comes from their own ways of searching for a specific element in the bigger array, first one being linear and second one being binary search. For each element in array 2, it performs a binary search in sorted array 1 whose time complexity is O(log(n)). Since we pass through every element in array 2 linearly, its time complexity is O(m). Multiplying these two values gives us the total time complexity for algorithm 2, O(m*log(n)).

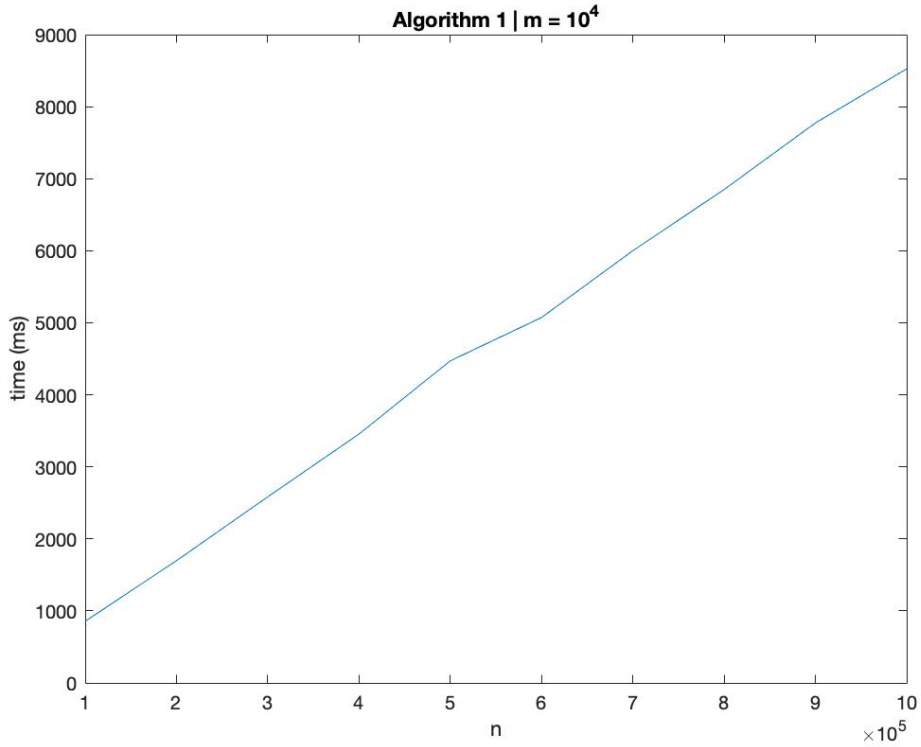
Algorithm 3: In algorithm 3, a third array is created, a frequency array for the first array. In order to that the greatest element of the first array must be found. This algorithm does a single traversal on array 1 for this operation with a time complexity of O(n). The creation of the frequency array is in constant time or O(1). After creating, all values of the frequency array should be set to 0 which has a time complexity of O(n). Then, the real frequency values of array 1 should be set to frequency array, O(n) again. After initializing the frequency array, algorithm performs a single complete traversal on array 2, checking whether that element is present in array 1 by using the frequency table. It decreases the desired index in the frequency array each time it passes to a new value in array 1, which is in constant time, O(1). In total there are three n-iterations and one m-iteration which means time complexity of O(m + n) since O(3n) the same thing as O(n).

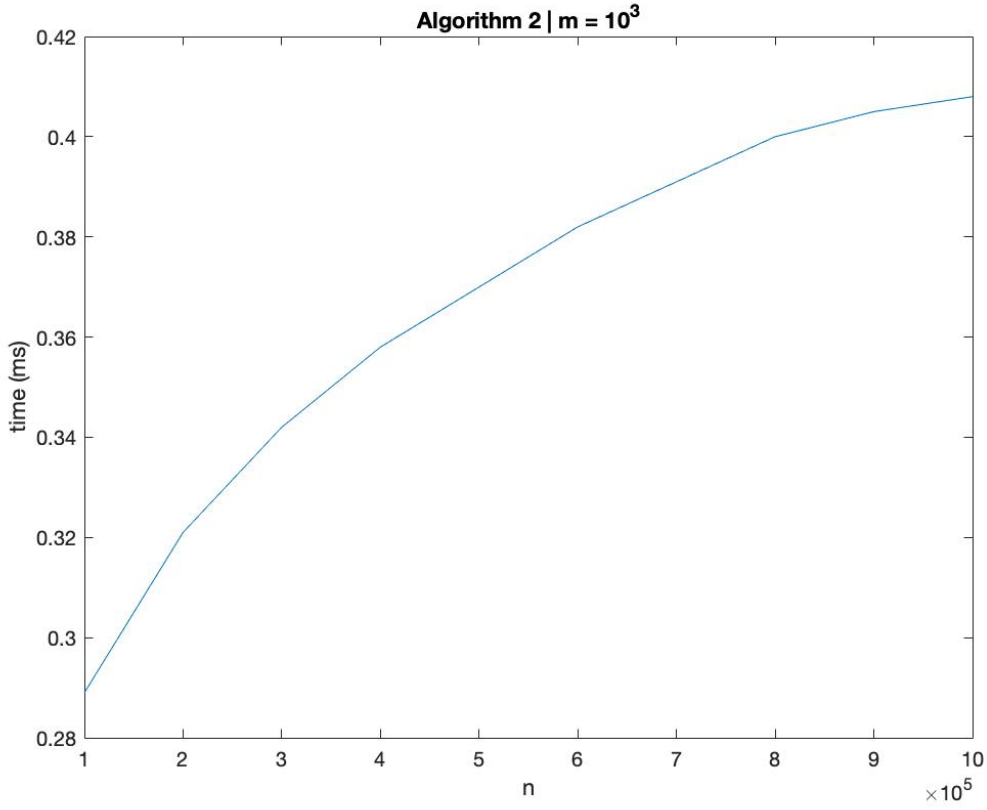
TABLE

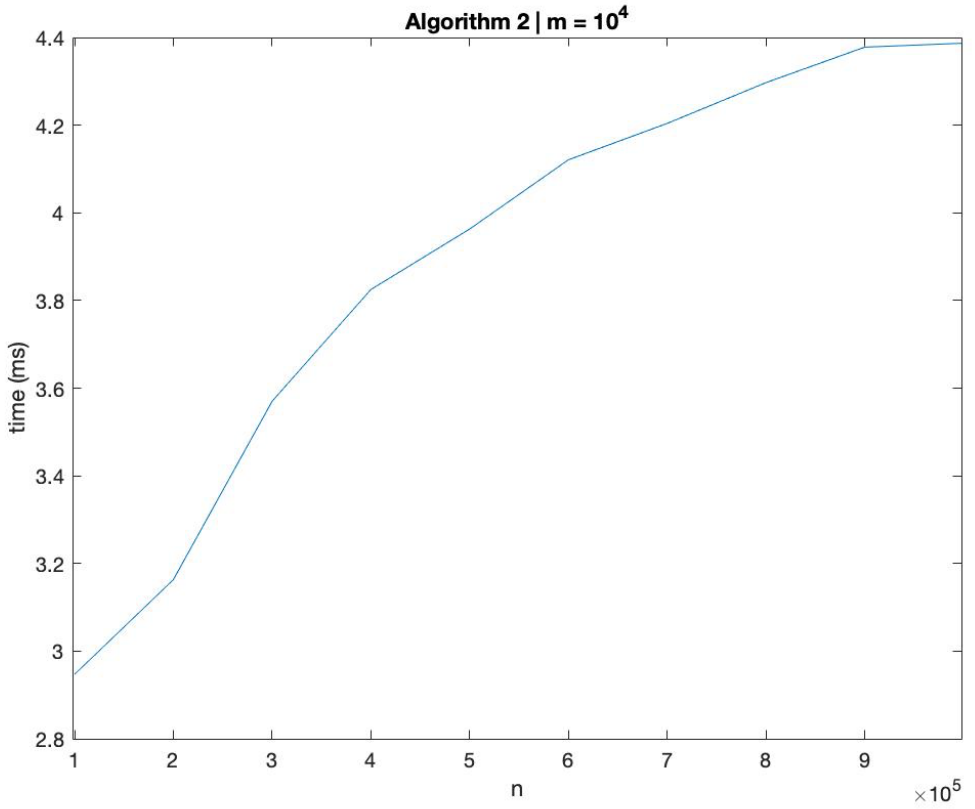
Average Algorithm Execution Time Table (milliseconds)						
n	*Algorithm 1*		*Algorithm 2*		*Algorithm 3*	
	m = 10^3	**m = 10^4**	**m = 10^3**	**m = 10^4**	**m = 10^3**	**m = 10^4**
1 * 10^5	87.959	856.685	0.289	2.947	0.747	0.777
2 * 10^5	168.770	1696.90	0.321	3.163	1.467	1.547
3 * 10^5	264.370	2583.11	0.342	3.570	2.312	2.406
4 * 10^5	348.373	3455.20	0.358	3.825	3.177	3.078
5 * 10^5	430.511	4471.35	0.370	3.963	3.939	3.920
6 * 10^5	520.238	5072.97	0.382	4.121	4.790	4.719
7 * 10^5	591.199	6000.25	0.391	4.204	5.720	5.614
8 * 10^5	700.522	6849.43	0.400	4.297	6.912	6.381
9 * 10^5	765.824	7774.01	0.405	4.378	7.362	7.294
10 * 10^5	845.870	8524.58	0.408	4.387	8.194	8.135

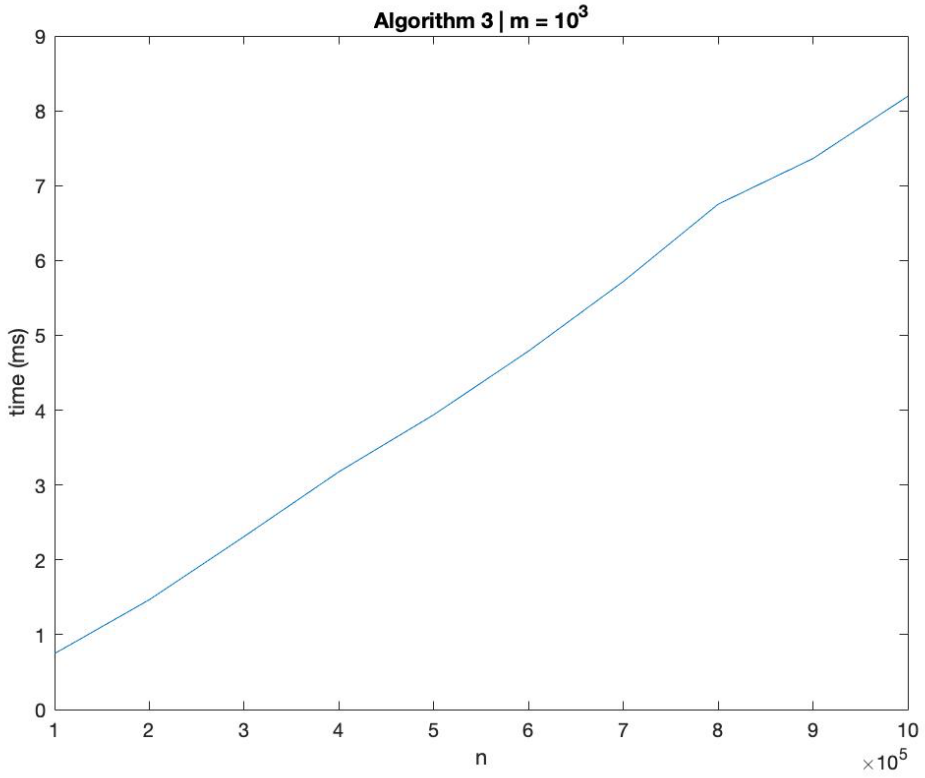
PLOTS

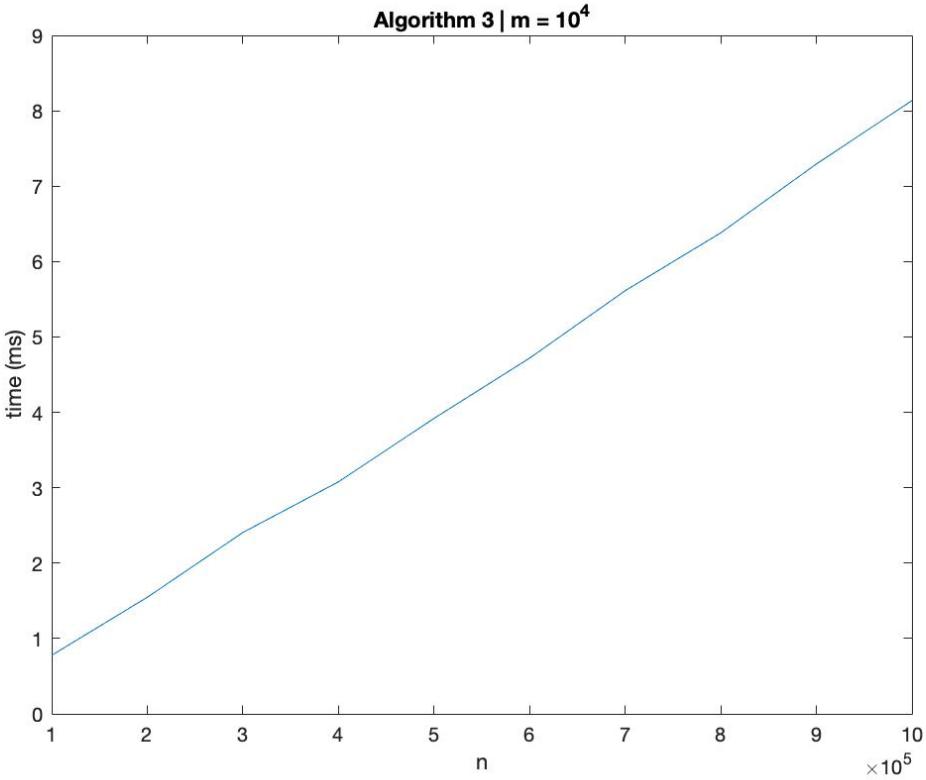

Algorithm 1 | m = 10^3
time (ms)
900
800
700
600
500
400
300
200
100
1
2
3
4
5
6
7
8
9
n
×10^5


Algorithm 1 | m = 10^4
time (ms)
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
1
2
3
4
5
6
7
8
9
10
n
×10^5


Algorithm 2 | m = 10^3
time (ms)
n
×10^5
0.42
0.4
0.38
0.36
0.34
0.32
0.3
0.28
1
2
3
4
5
6
7
8
9
10


Algorithm 2 | m = 10^4
time (ms)
n
×10^5
4.4
4.2
4
3.8
3.6
3.4
3.2
3
2.8
1
2
3
4
5
6
7
8
9


Algorithm 3 | m = 10^3
time (ms)
n
×10^5
0
1
2
3
4
5
6
7
8
9
10


Algorithm 3 | m = 10^4
time (ms)
n
×10^5
0
1
2
3
4
5
6
7
8
9
10

COMPUTER SPECIFICATIONS

Model Name:	MacBook Pro
Model Identifier:	MacBookPro16,3
Processor Name:	Quad-Core Intel Core i5
Processor Speed:	1,4 GHz
Number of Processors:	1
Total Number of Cores:	4
L2 Cache (per Core):	256 KB
L3 Cache:	6 MB
Memory:	8 GB 2133Mhz LPDDR3