CS 201-01

HW 02

Mehmet Onur Uysal

22002609

Algorithm Table

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
10^7	379	37465	0.242	3.020	36.74	37.80
$2 * 10^7$	742	74250	0.262	3.698	75.27	74.96
$3 * 10^7$	1122	110657	0.260	4.131	109.27	112.73
$4 * 10^7$	1510	149986	0.277	4.438	146.13	149.13
$5 * 10^7$	1864	187595	0.283	4.607	163.90	191.64
$6 * 10^7$	2240	227257	0.294	4.857	217.99	227.71
$7 * 10^7$	2643	258479	0.318	4.915	254.48	267.03
$8 * 10^7$	2972	289519	0.317	5.102	298.69	290.77
$9 * 10^7$	3335	331896	0.328	5.330	316.79	327.17
10^8	3725	374544	0.321	5.365	363.88	365.71

(Time values are in miliseconds)

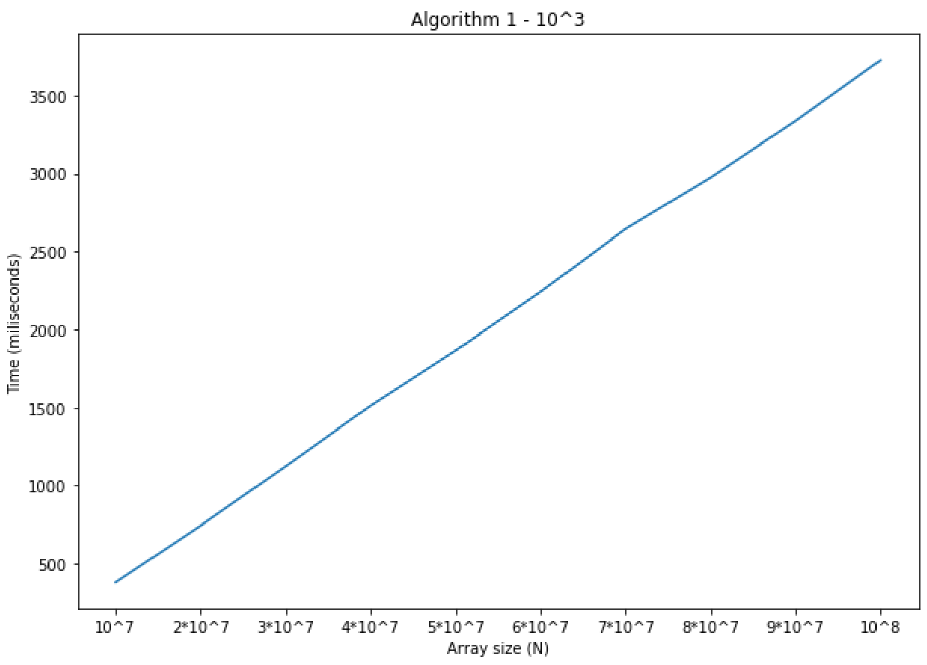
Computer Parameters

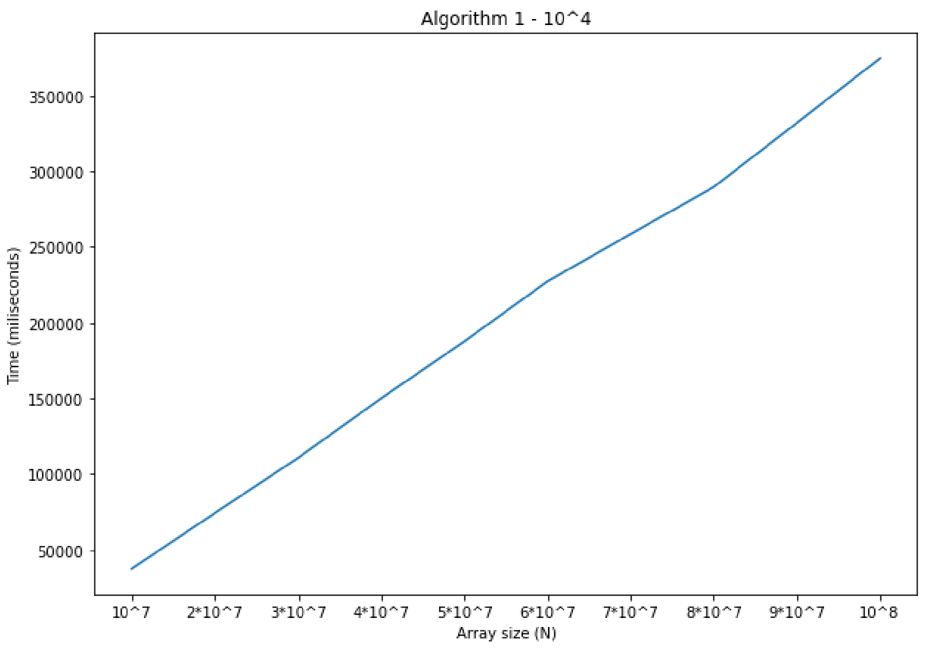
Processor: 7th Gen Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz   2.90 GHz

RAM: 8.00 GB   Speed: 2133MHz

Code was compiled and run with GNU GCC compiler in Codeblocks IDE.

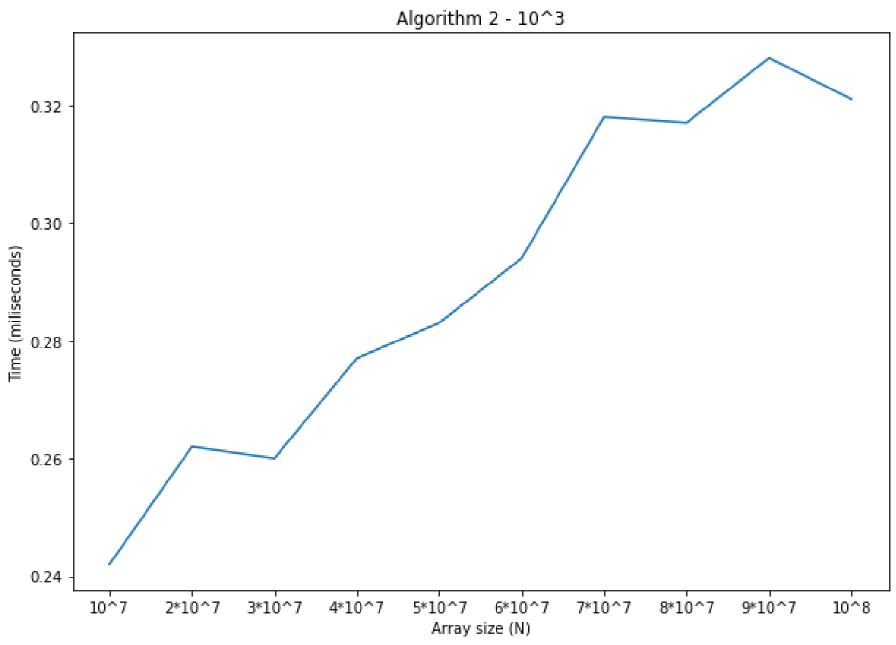
1. **<u>Algorithm 1 (Linear Search)</u>**
   Linear search is the slowest algorithm among these three. For every item in the second array, algorithm iterates over all the items in the first array using two nested for loops. So, the complexity is O(M*N) for this algorithm. Even if the first item on the second array is not in the first array, algorithm checks all elements of the first array. So, the time complexity is N for the best-case scenario. Graph of this algorithm grows in a linear manner when M is constant.

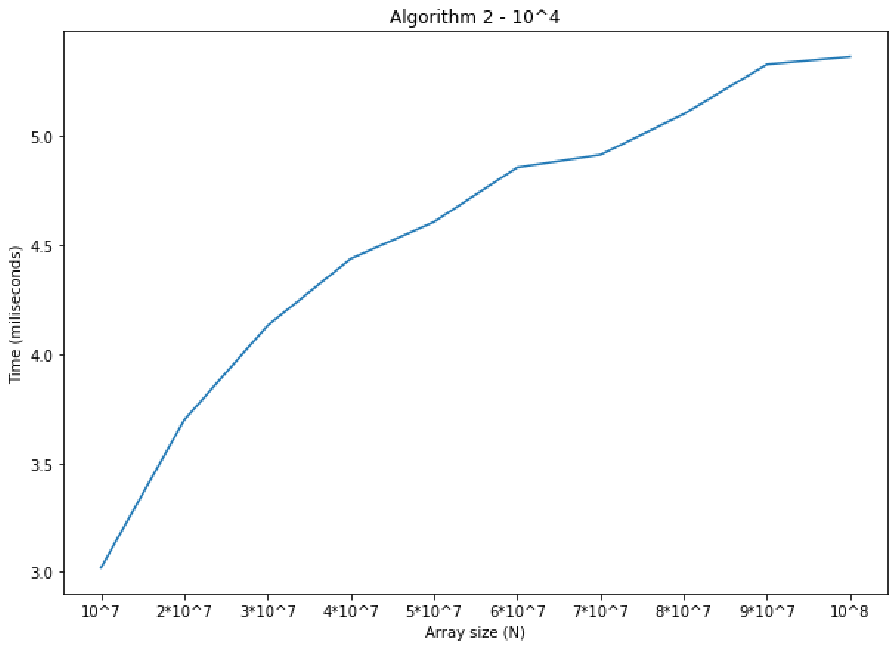




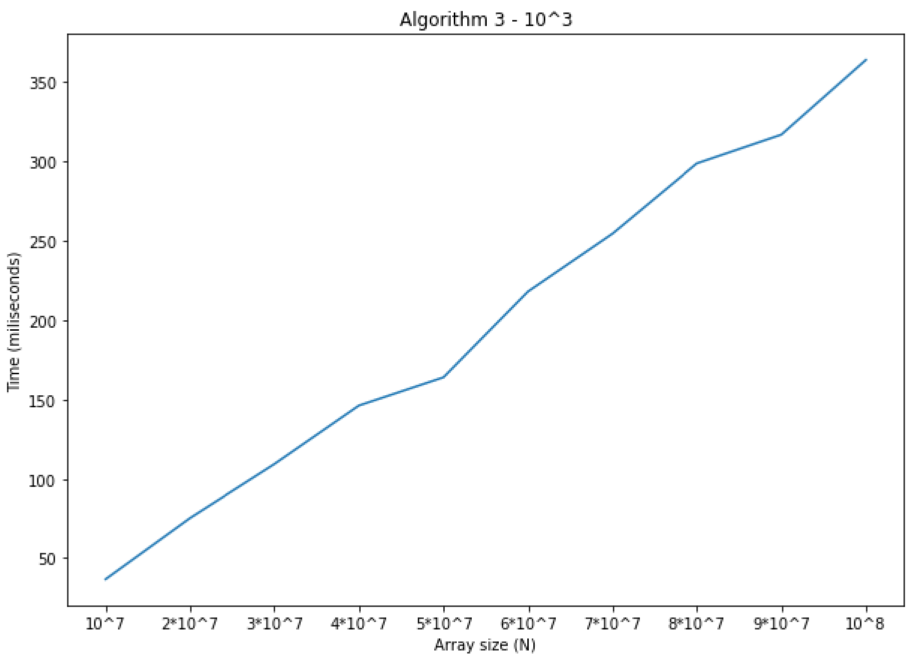
2. **<u>Algorithm 2 (Binary Search)</u>**

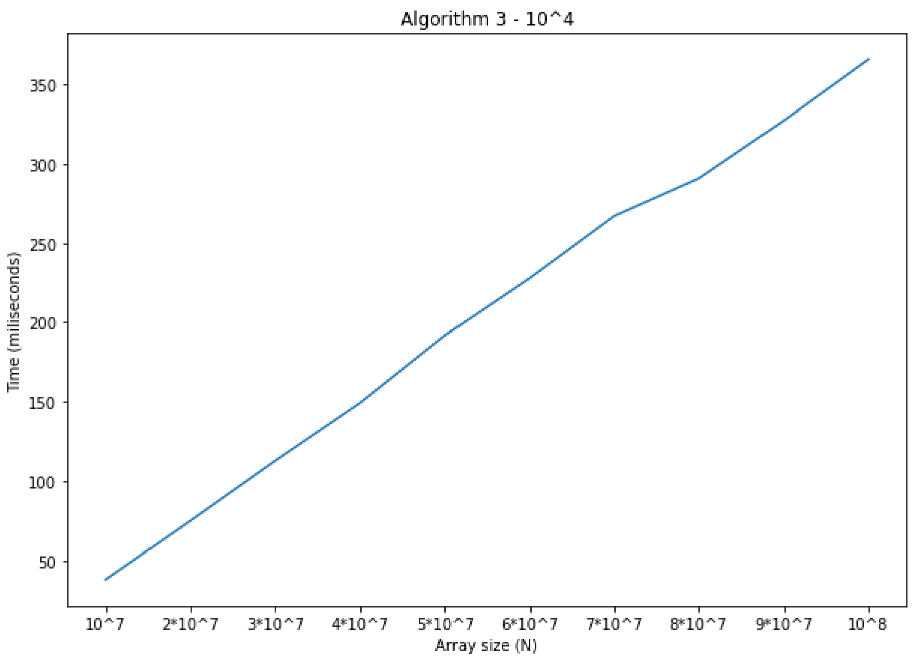
   For these values of N and M, binary search algorithm performs best among others. Unlike the first one, binary search algorithm requires the first array to be sorted. This way, according to the value to be searched in the first array, algorithm can determine which part of the array to search. Again, the first array is searched for every item in the second array but searching time is significantly lower than the first algorithm. Time complexity of this algorithm is O(M*log(N)) because all M items are searched each of whom takes log(N) time. When M is constant and 10^3 values seem arbitrary but for larger values, graph of the algorithm is clearly logarithmic. For the best-case scenario (first item does not exist in the first array), time complexity is O(log(N)). So, it is clear that this algorithm is superior to the linear search algorithm when out first array is sorted.





3. **<u>Algorithm 3 (Frequency Map)</u>**
   Frequency map algorithm works differently compared to others. While the other two searches each item in the array, this algorithm initially creates a map of the first array which makes it easier to determine whether an item with a specific value exists in the array. Creation and initialization of the frequency map is O(N) and searching every item in the map is O(M). So, time complexity of this algorithm is O(M+N). Although this algorithm did not operate as fast as the second one, it is way more efficient than the first linear search algorithm.


Algorithm 3 - 10^3
Time (miliseconds)
10^7
2*10^7
3*10^7
4*10^7
5*10^7
6*10^7
7*10^7
8*10^7
9*10^7
10^8
Array size (N)


Algorithm 3 - 10^4
Time (miliseconds)
10^7
2*10^7
3*10^7
4*10^7
5*10^7
6*10^7
7*10^7
8*10^7
9*10^7
10^8
Array size (N)