Oğuz Kuyucu

21902683

Section-2

The first algorithm controls each element of arr1(number of items = n) for each element of arr2(number of items = m) until it cannot find the item of arr2 in arr1. If n doubles, the time will double as well because for each item of arr2, algorithm controls 2n items. The same method applies for m as well. If m doubles, the algorithm passes for 2m items. Therefore, time required will double. Therefore, its complexity is O(m*n)

The second algorithm search items in arr1 with binary search for each item in arr2. If n doubles, number of pass will not get double, it will increase much slower.The number of pass doubles if n gets n^2. However, algorithm 2 uses linear search while passing over arr2's items. Therefore, algorithm two's complexity is O(m*logn). In order to use that algorithm, arr1 should be sorted already.

The third algorithm detects the max number = k of arr1 first. Then It creates an array which contains k element and initialize all items to 0. Then it visits all element of arr1 and it modifies array created which is frequency table. After that, for each element of arr2, it controls and modifies the frequency table. If the element exist in frequency table, algorithm decreases it. If one of the elements of frequency table gets -1, it returns 0. Independent from m, algorithm visits each element of arr1 2 times (while looking for max number and while creating frequency table). Independent from n, the algorithm visits all elements of arr2 only 1 times. Additionally, the algorithm won't visit all elements of frequency table for each element of arr2. Because controlling arr1 and arr2 are independent, their complexity should be added which is O(m+n).

Computer:

Lenovo Y700

OS:Windows 10 Home 64 bit

Ram: 16384 MB DDR 4

Processor: Intel core i7 6700HQ CPU @ 2.60GHz(8 CPUs),~2.6 GHz

Screen

Intel HD Graphics 530

Total memory: 8233 MB

VRAM : 128MB

Shared memory: 8105 MB

GPU

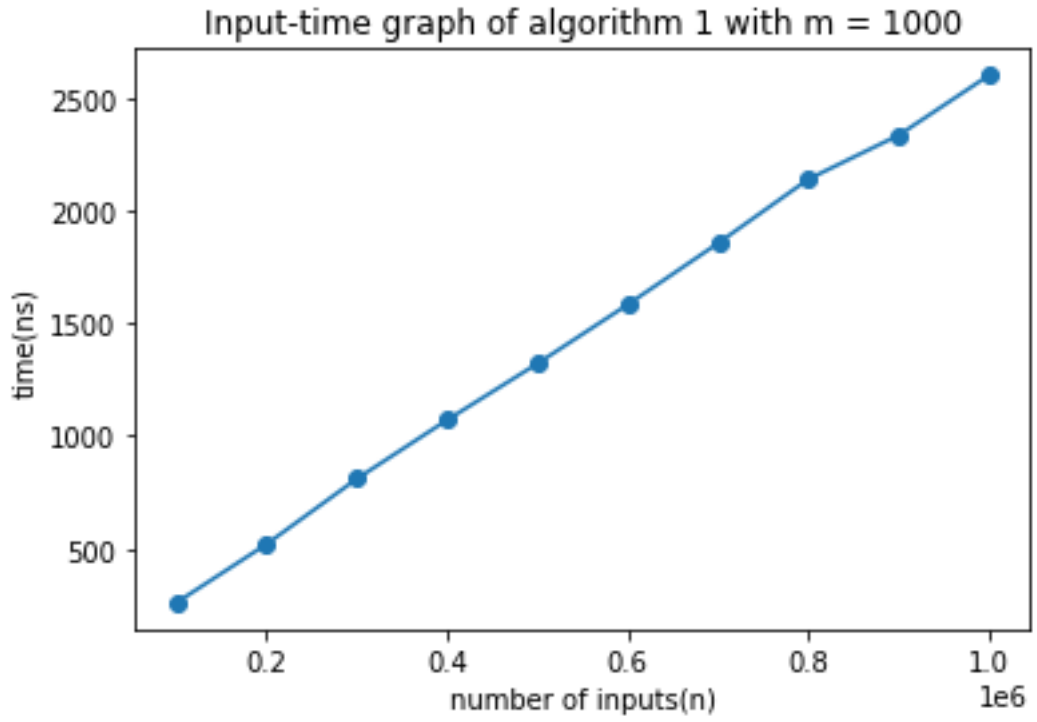
NVIDIA GeForce GTX 960 M

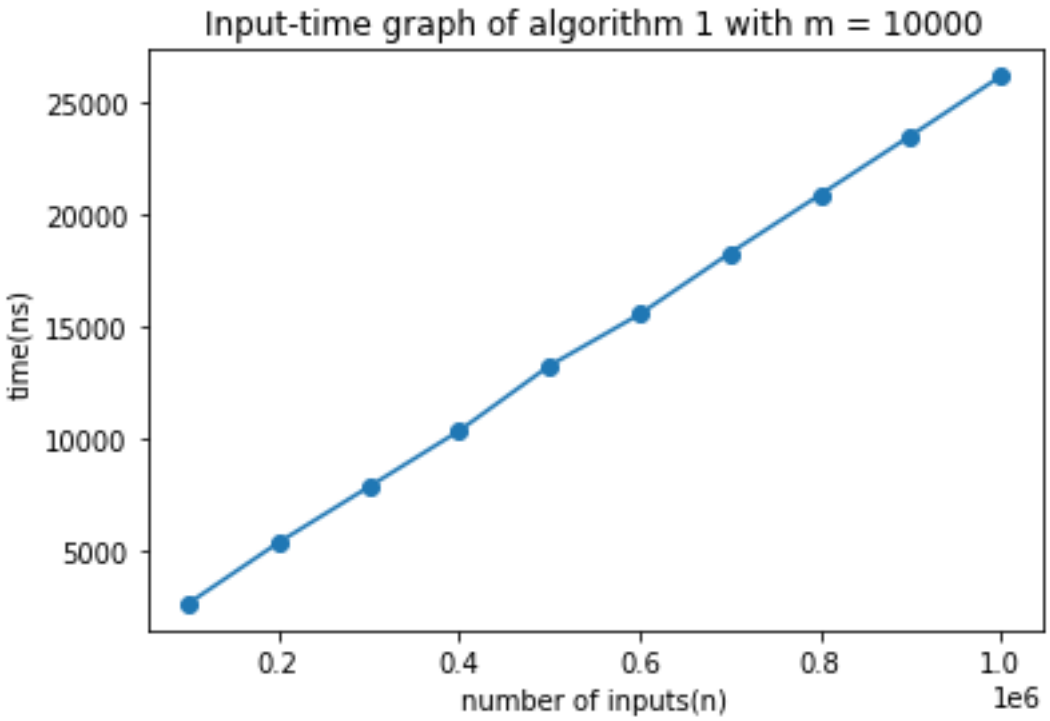
Total Memory: 12171 MB

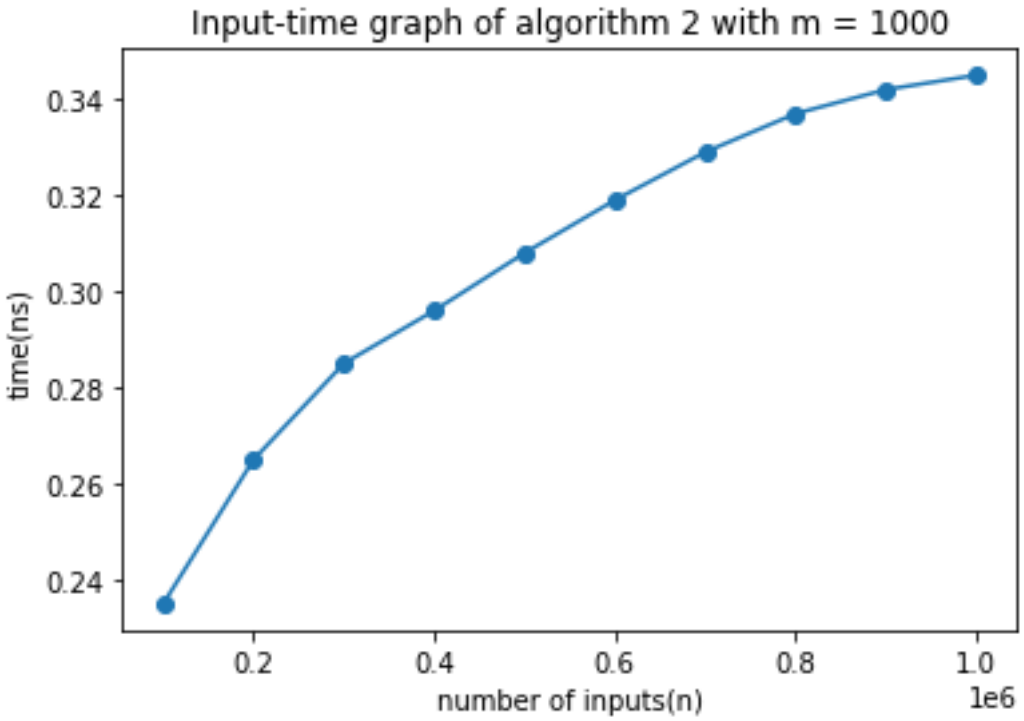
VRAM: 4065MB

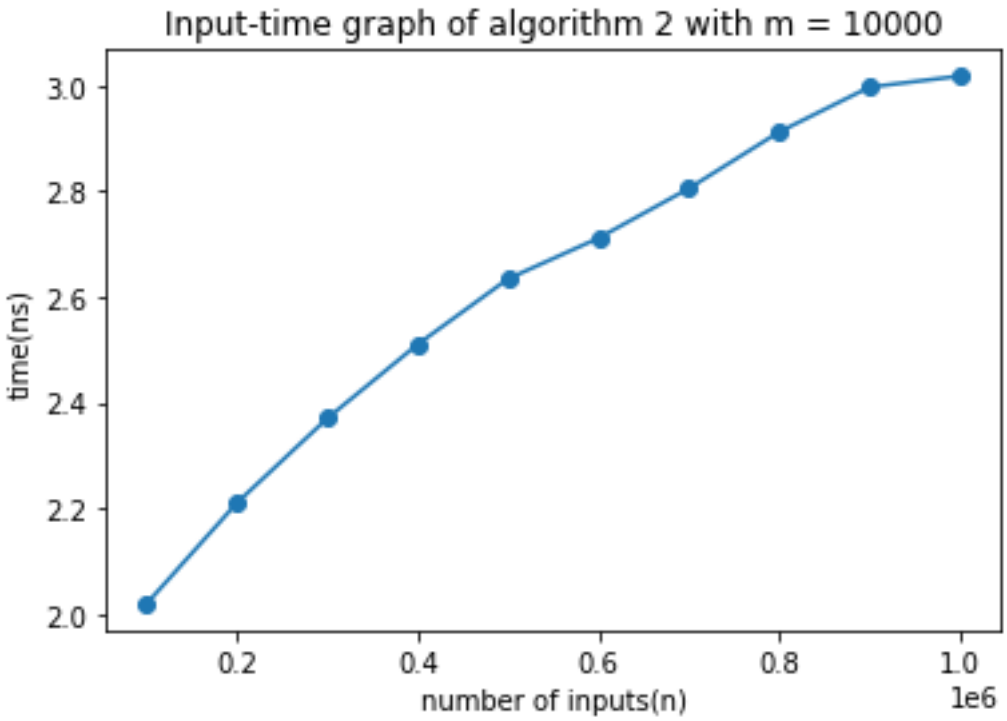
Shared memory:8105 MB

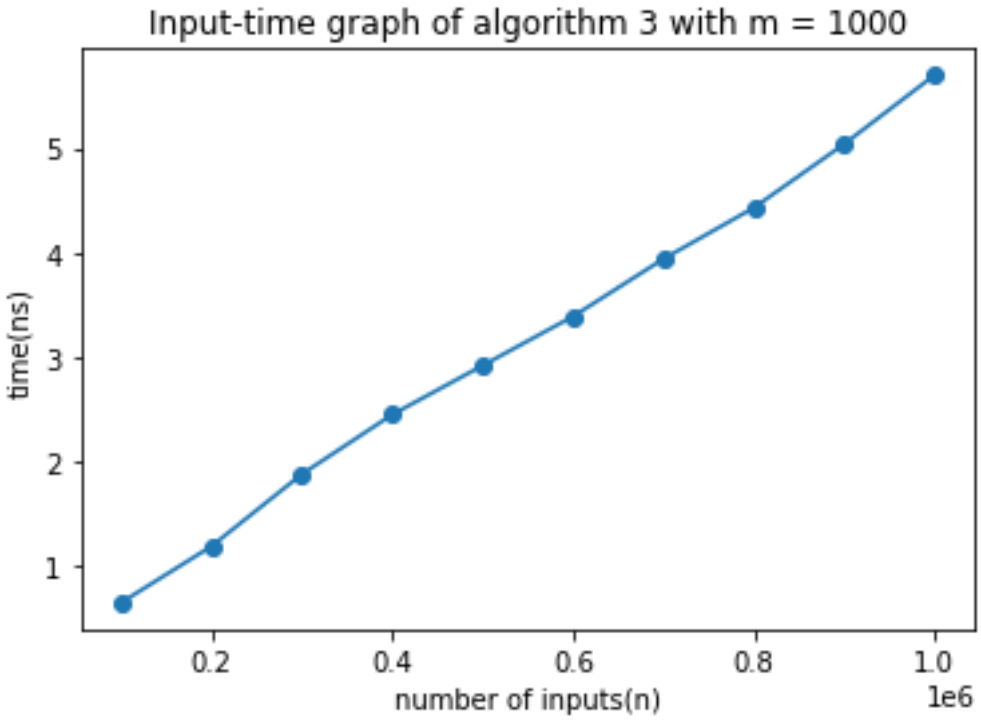
	algorithm1		algorithm2		algorithm3	
n	m =1000	m=10000	m =1000	m=10000	m = 1000	m=10000
100000	261	2606	0.235	2.019	0.65	0.80
200000	521	5333	0.265	2.110	1.19	1.26
300000	811	7822	0.285	2.370	1.88	1.98
400000	1072	10324	0.296	2.510	2.45	2.56
500000	1322	13233	0.308	2.634	2.92	2.99
600000	1584	15536	0.319	2.711	3.39	3.62
700000	1856	18269	0.329	2.805	3.94	3.99
800000	2140	20881	0.337	2.912	4.43	4.53
900000	2335	23510	0.342	2.997	5.04	5.25
1000000	2601	26172	0.345	3.018	5.70	5.78

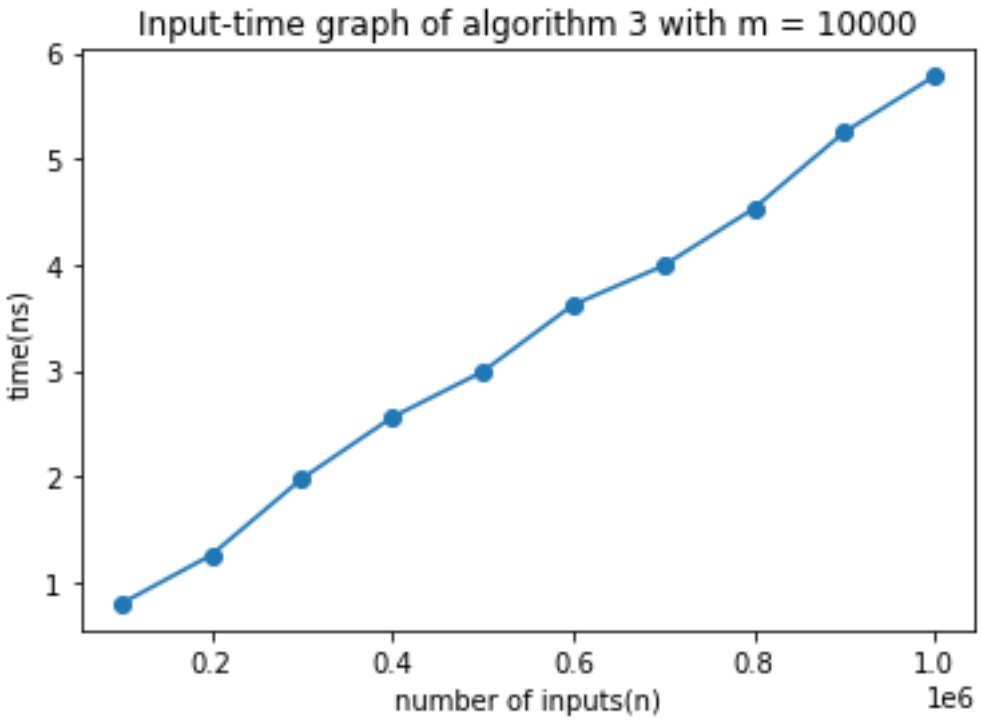



Input-time graph of algorithm 1 with m = 10000
time(ns)
25000
20000
15000
10000
5000
0.2
0.4
0.6
0.8
1.0
1e6
number of inputs(n)


Input-time graph of algorithm 2 with m = 1000
time(ns)
0.34
0.32
0.30
0.28
0.26
0.24
0.2
0.4
0.6
0.8
1.0
1e6
number of inputs(n)


Input-time graph of algorithm 2 with m = 10000
time(ns)
3.0
2.8
2.6
2.4
2.2
2.0
0.2
0.4
0.6
0.8
1.0
1e6
number of inputs(n)


Input-time graph of algorithm 3 with m = 1000
time(ns)
5
4
3
2
1
0.2
0.4
0.6
0.8
1.0
1e6
number of inputs(n)


Input-time graph of algorithm 3 with m = 10000
time(ns)
number of inputs(n)
1e6
1
2
3
4
5
6
0.2
0.4
0.6
0.8
1.0