CS-201

Fundamental Structures of Computer Science I

HOMEWORK-2

Mehmet Hasat Serinkan

21901649

Question 1

Algorithm 1: The time complexity of the first algorithm is O(N). There are 2 statements before the loop. The for loop will be executed for N+1 times and the statements in the for loop will be executed for N times. Lastly, the return statement will be executed for only one time. Therefore, when this function is called, the total amount of execution is 2+(N+1)+N+N+1 = 3N+4. Hence, its upper bound is O(3N+4), which is simply O(N).

Algorithm 2: In all cases, 2 statements will be executed before the loop. There are 3 different cases for algorithm 2:

- Worst Case: If the condition is satisfied on the last step of the loop, it would be the worst case. Loop will be executed for N times, the statements in the loop will run for N-1 times. After that, the "else" part will be executed. There are 3 statements before the loop in the "else" part. Then, the loop will execute for N times the condition of "n % i" equals n. The statements in the loop will execute for N-1 times. In the end, it will return the function which will be executed only one time. Therefore, the entire time of execution equals to 2+N+2(N-1)+3+N+2(N-1)+1 = 6N+2 which is equal to O(N). So, it can be easily seen that the time complexity of the worst case of algorithm 2 is O(N).
- Best Case: If the loop executes only one time, then it will be the best case. Therefore, the time complexity of the best case of algorithm 2 is O(1).
- Average Case: Average case equals to all possible case times divided by the number of cases. Since the best case of algorithm 2 is O(1) and the worst case of algorithm 2 is O(N), all possible case times can be simplified as 1+2+3+4+…+N = $\frac{N(N+1)}{2}$, then the total number of cases is N. Therefore, the average case will be equal to O($\frac{(N+1)}{2}$), which is simplified as O(N).

Algorithm 3: The base case is n=1 which means when n=1, function will return. The recurrence relation when n is even is T(N) = T($\frac{N}{2}$) + 1, and when n is odd T(N) = T($\frac{N-1}{2}$) + 1 which can simplified as T(N) = T($\frac{N}{2}$) + 1. Therefore, the general recurrence relation is T(N) =

T($\frac{N}{2}$) + 1. Now, assume that the function will return when it makes k times execution. Then, the recurrence relation becomes T(N) = T($\frac{N}{2^k}$) + k. To reach the base case, 2^k should be N which means k = logN. Then equation becomes T(N) = T(1) + logN = 1 + logN. Its time complexity O(1 + logN) can simplified as O(logN). Therefore, the time complexity of the third algorithm is O(logN).

Question 2

Specifications of my computer:

- System Manufacturer and Model: MONSTER-TULPAR T7 V19.5
- Processor: Intel® Core™ i7-9750H CPU @ 2.60 GHz
- Installed RAM: 32.0 GB
- System type: 64-bit operating system
- Windows specification: Windows 10 Home Single Language
- Graphic Card: NVIDIA GeForce RTX 2070

Question 3

Running times of algorithm 1,2, and 3 with different n and p values. Execution times are reported as milliseconds. Value of parameter "a" is 3.

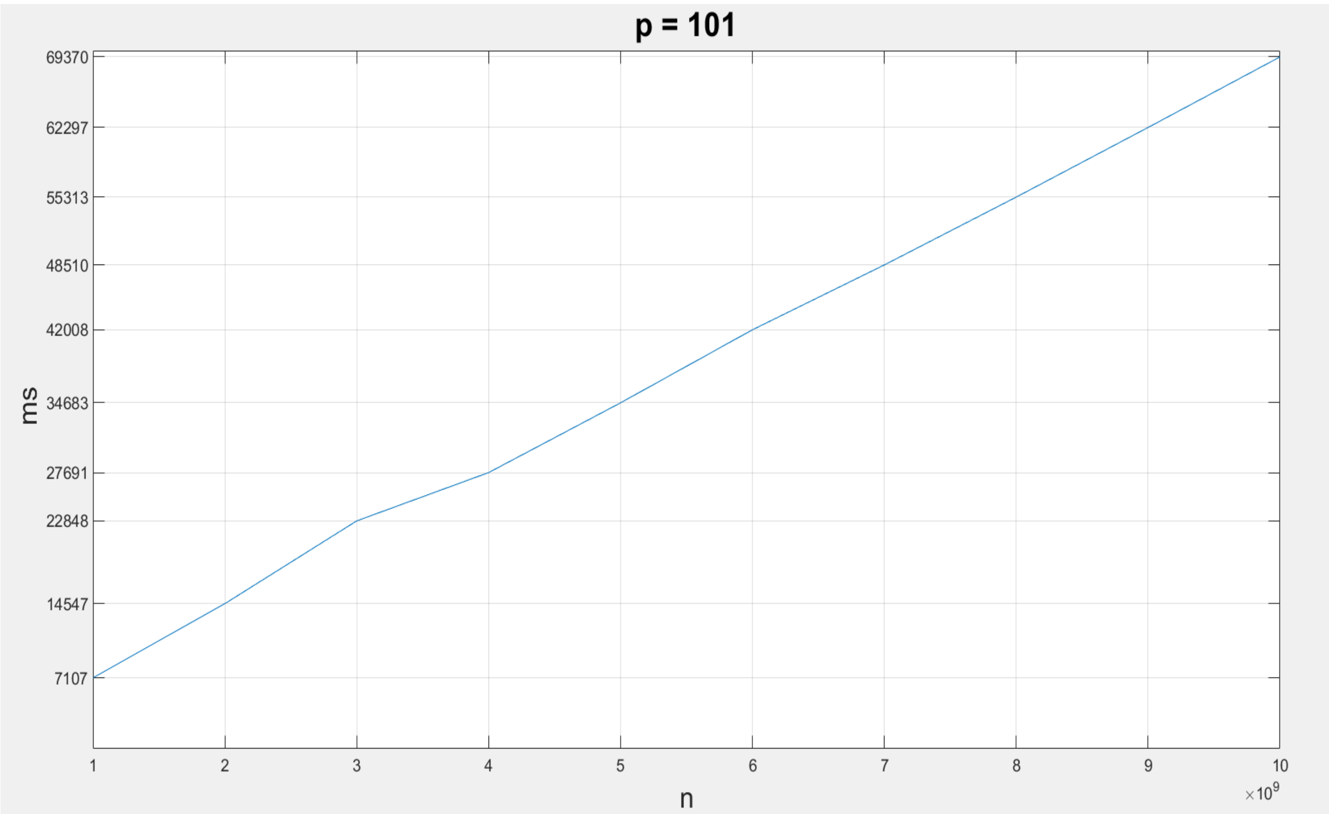
N	Algorithm 1			Algorithm 2			Algorithm 3		
	P = 101	P=1009	P=10007	P = 101	P=1009	P=10007	P = 101	P=1009	P=10007
10^9	7107	7041	7170	0.000094	0.001354	0.001026	0.0000356	0.0000351	0.0000380
$2 x 10^9$	14547	13836	14705	0.000085	0.001549	0.000967	0.0000372	0.0000371	0.0000433
$3 x 10^9$	22848	20824	21467	0.000099	0.001264	0.000929	0.0000386	0.0000412	0.0000399
$4 x 10^9$	27691	27840	27761	0.000087	0.001425	0.000828	0.0000399	0.0000418	0.0000389
$5 x 10^9$	34683	34879	34590	0.000085	0.001578	0.001031	0.0000418	0.0000434	0.0000450
$6 x 10^9$	42008	41730	41360	0.000119	0.001380	0.000972	0.0000435	0.0000455	0.0000456
$7 x 10^9$	48510	48360	48515	0.000084	0.001483	0.000902	0.0000450	0.0000458	0.0000461
$8 x 10^9$	55313	55451	55548	0.000108	0.001282	0.000858	0.0000489	0.0000470	0.0000476
$9 x 10^9$	62297	62284	62291	0.000093	0.001400	0.001046	0.0000500	0.0000495	0.0000499
10^{10}	69370	68724	69230	0.000092	0.001559	0.001041	0.0000505	0.0000515	0.0000519

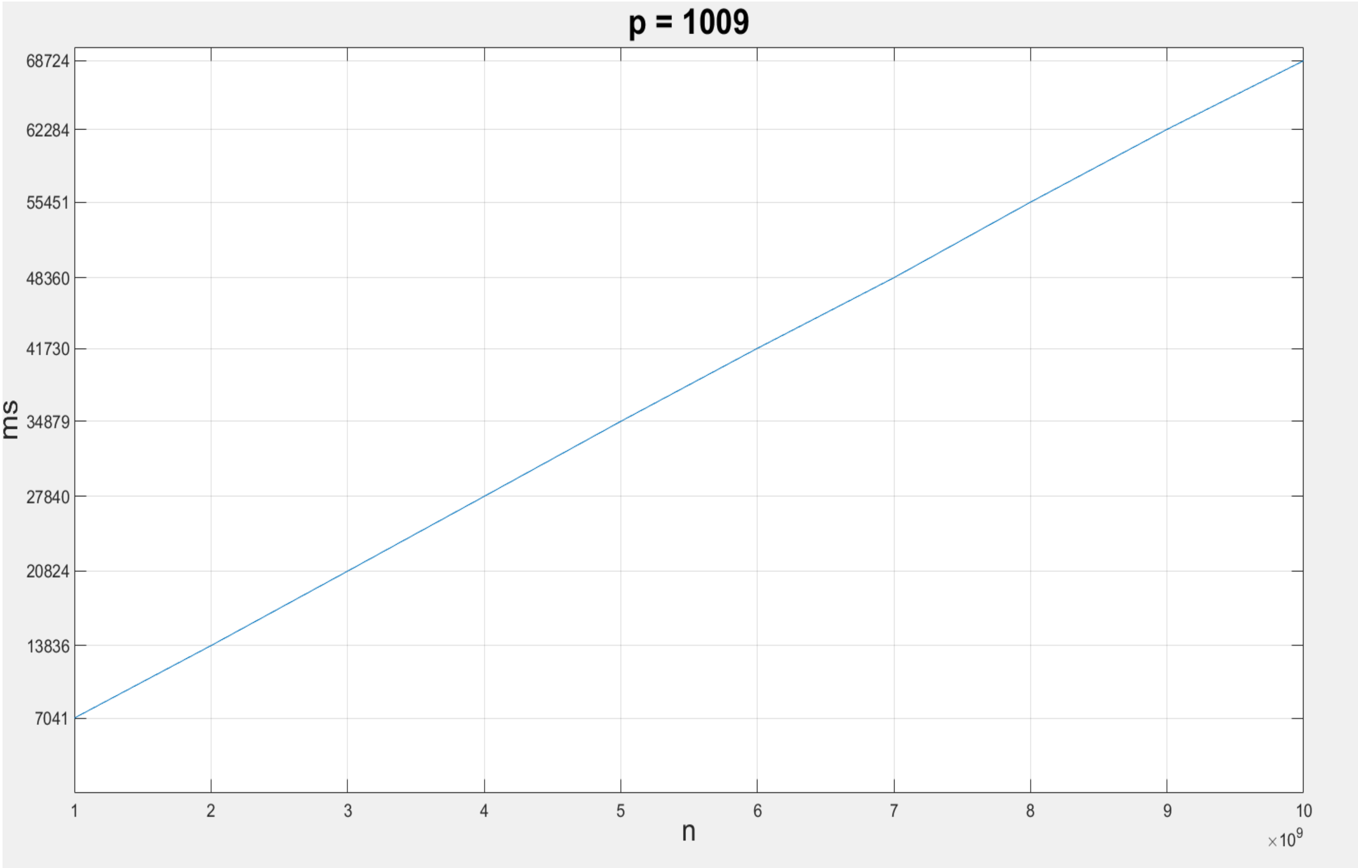
Comments:

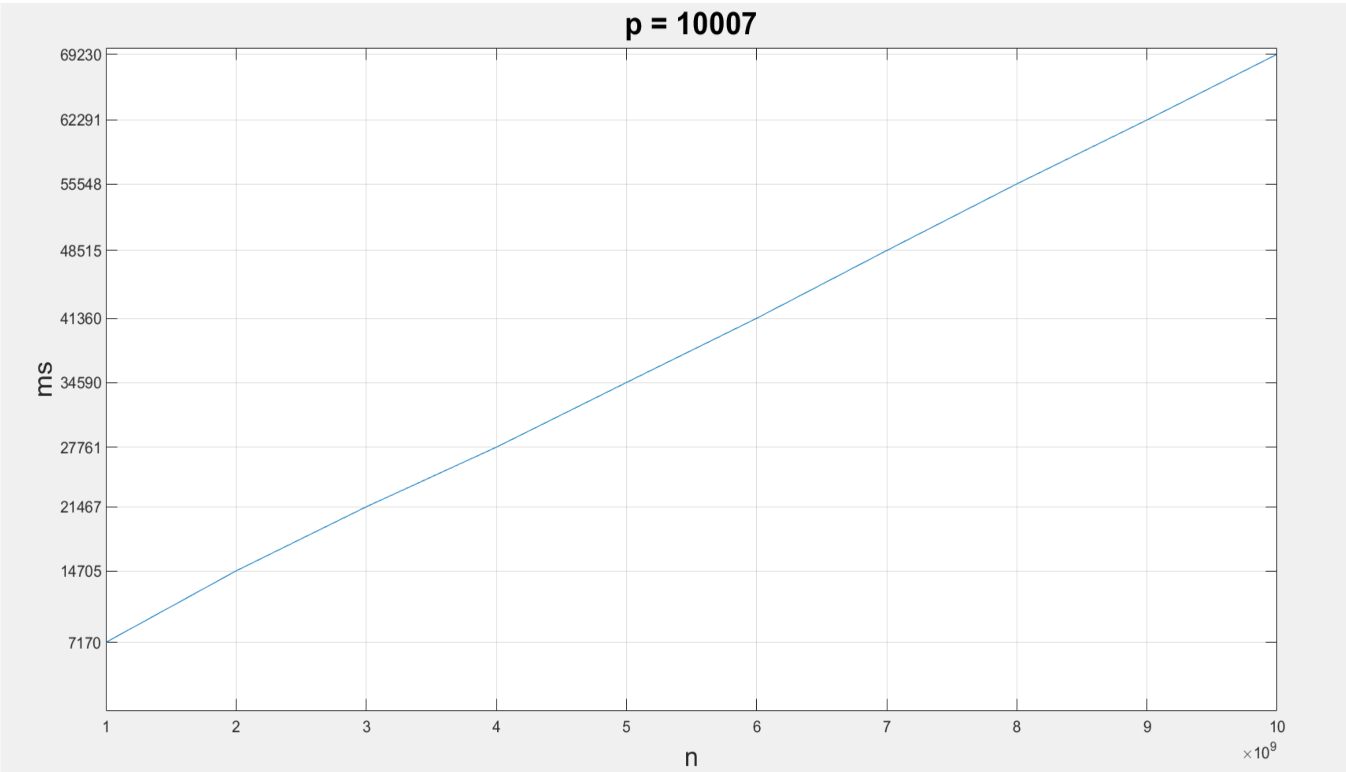
- It can be easily seen that Algorithm 1 is not efficient. It waits almost a minute when p is a huge number.
- Algorithm 2-3 is much more efficient than Algorithm 1. They execute the function very rapidly.
- The p values do not crucially affect run times in all algorithms.
- When p is a small value, it is best to use Algorithm 2.
- The time complexity of Algorithm 2 is almost always O(1). As we mentioned before, it is best case for Algorithm 2. The main reason for this is the numbers are huge; hence, it reaches the else part of algorithm 2 almost everytime.
- These values are not very accurate because of non-ideal experimental conditions. The conditions of observations are not the same. For instance, my computer gets heater in Algorithm 1; therefore, the execution time got much larger.

Question 4

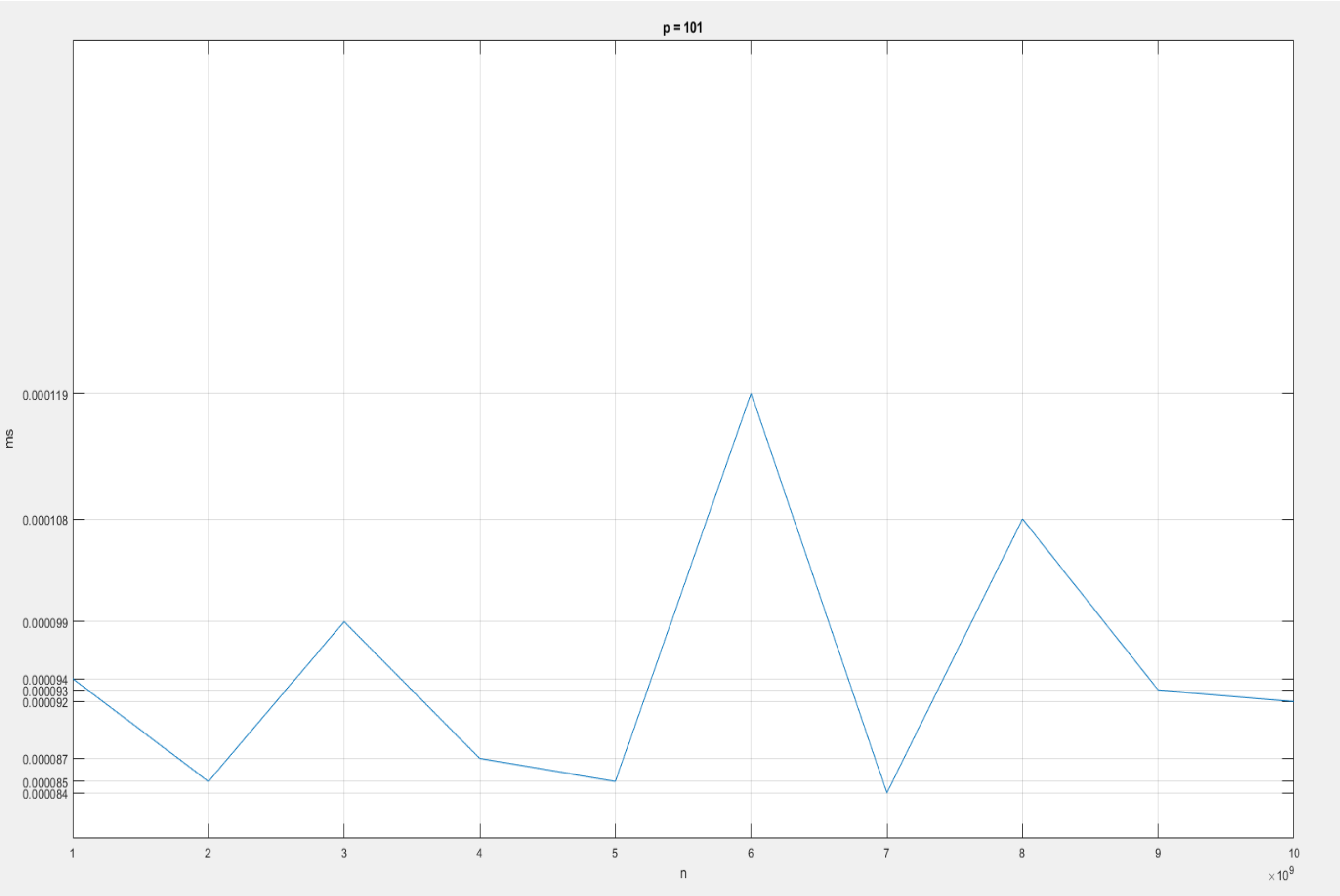
Plots of Algorithm 1:

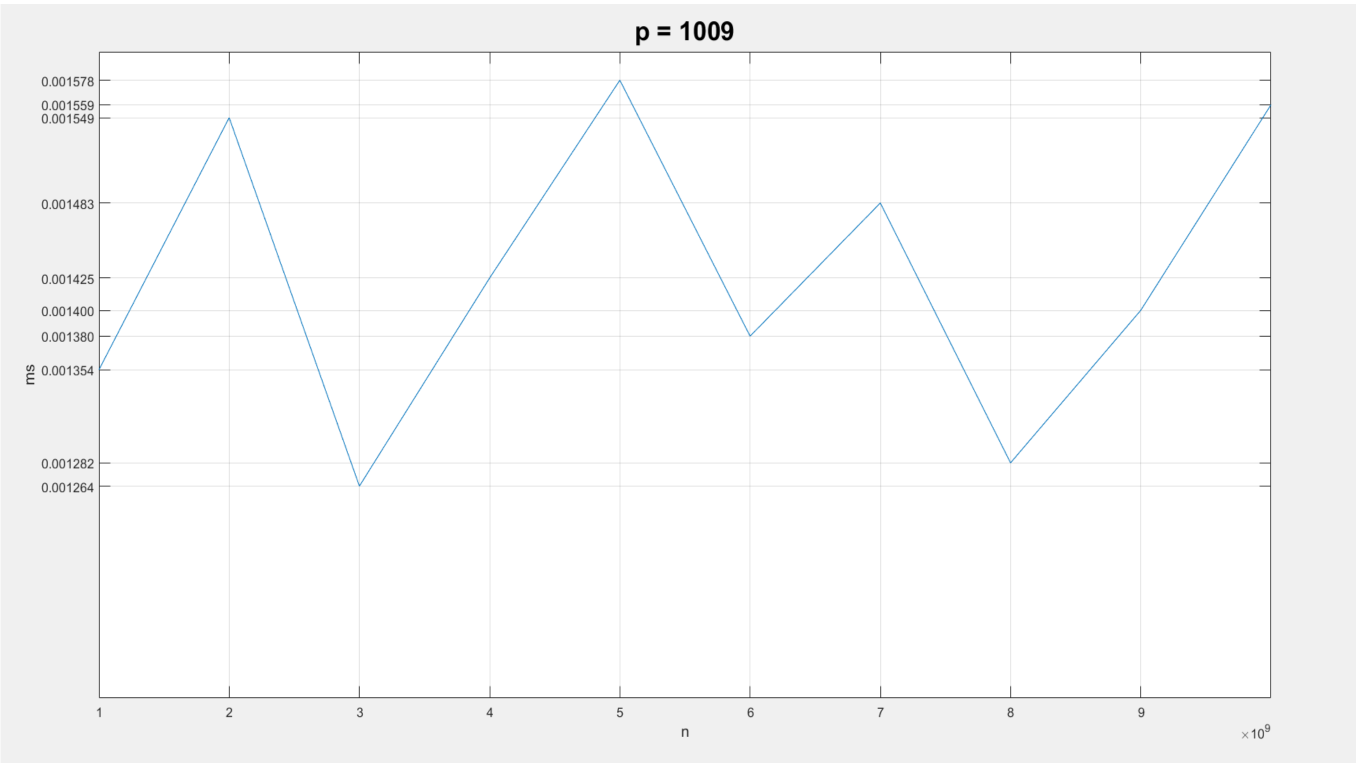


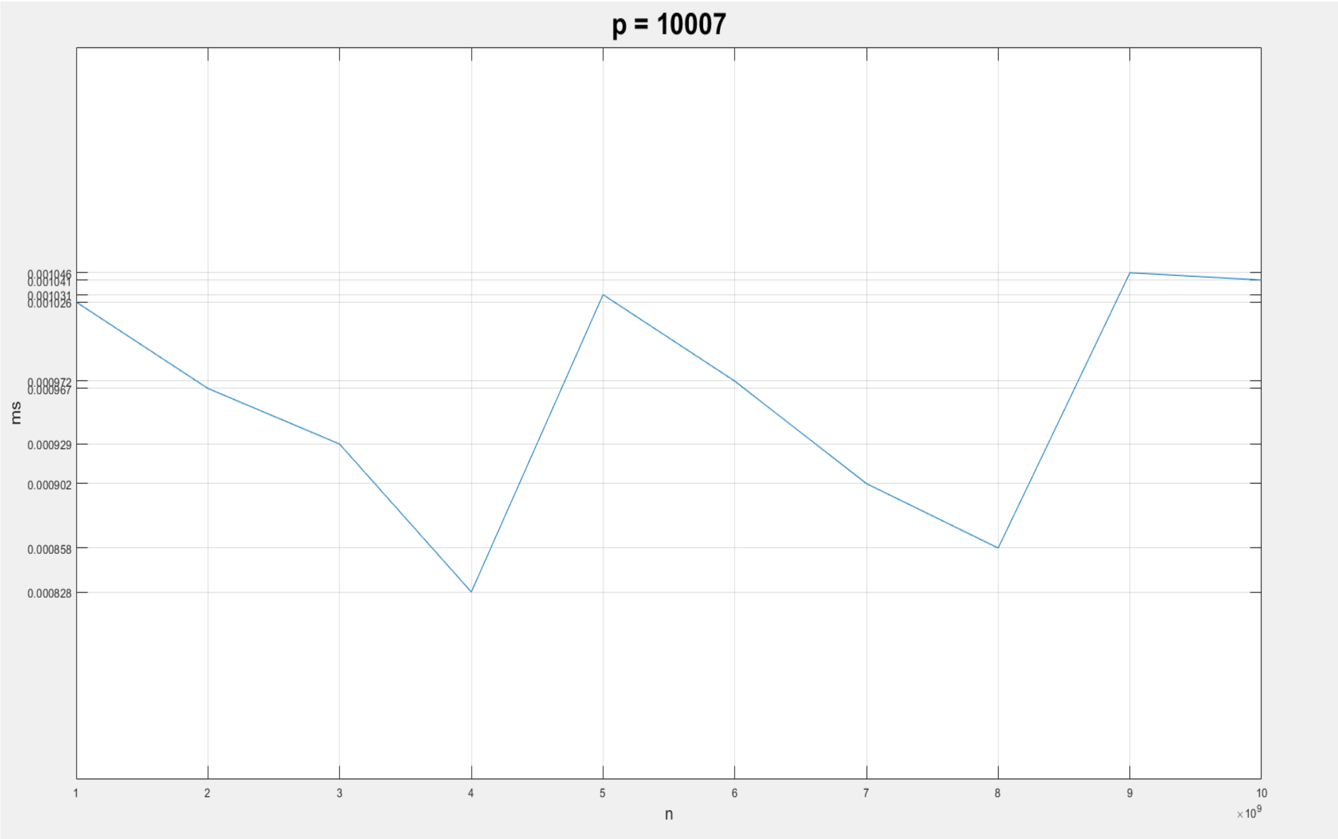

p = 1009
68724
62284
55451
48360
41730
34879
27840
20824
13836
7041
ms
1
2
3
4
5
6
7
8
9
10
n
×10^9


p = 10007
69230
62291
55548
48515
41360
34590
27761
21467
14705
7170
ms
1
2
3
4
5
6
7
8
9
10
n
×10^9

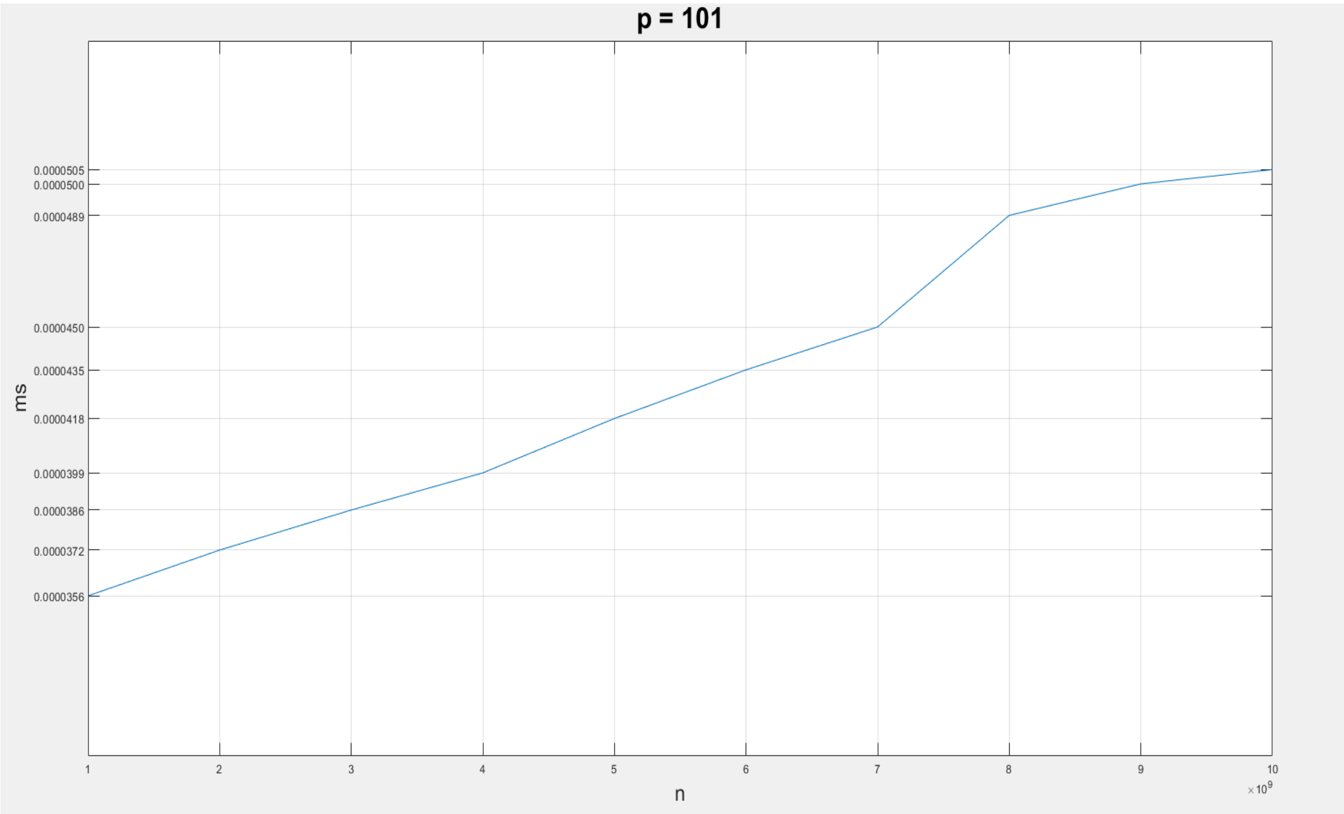
Plots of Algorithm 2:

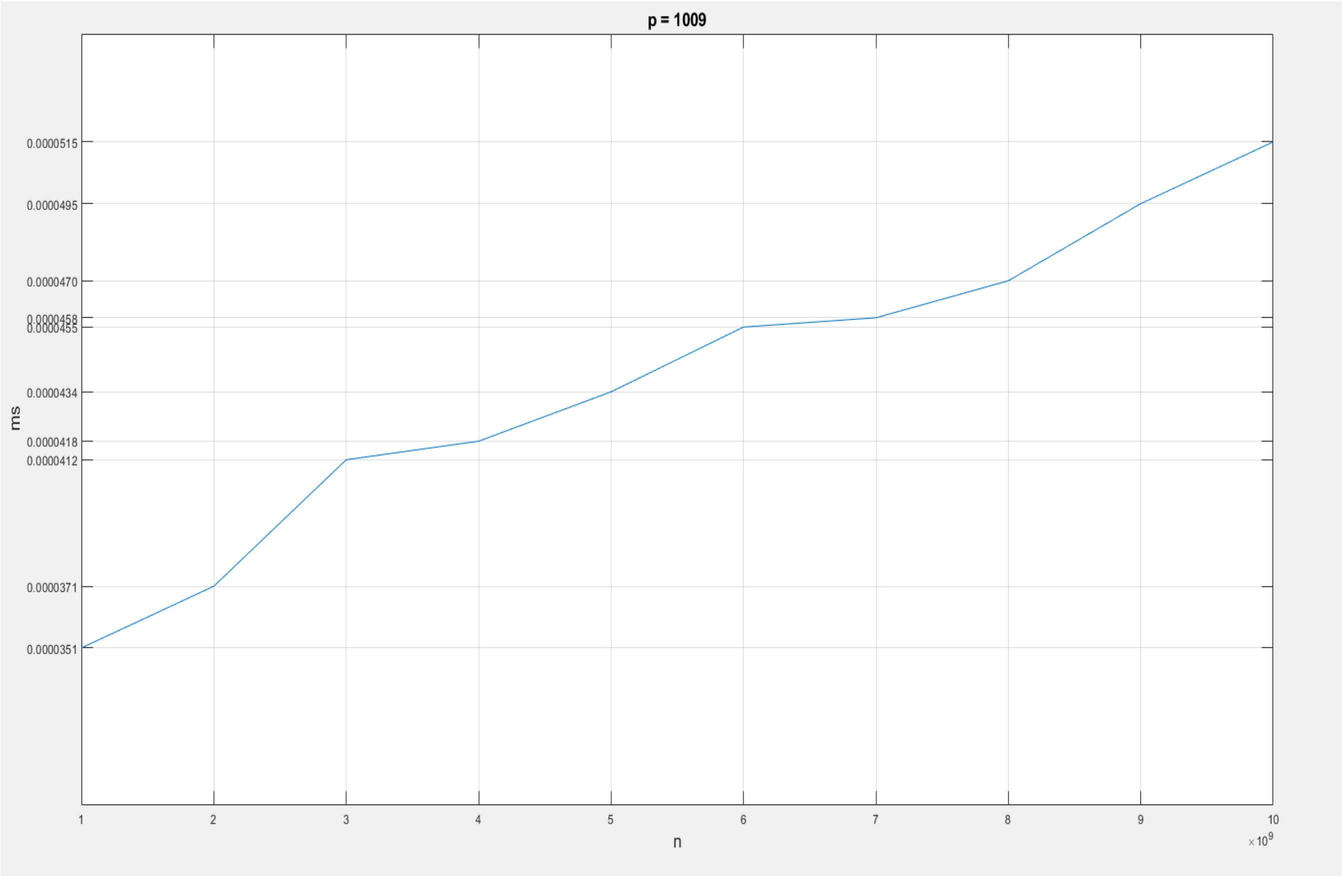


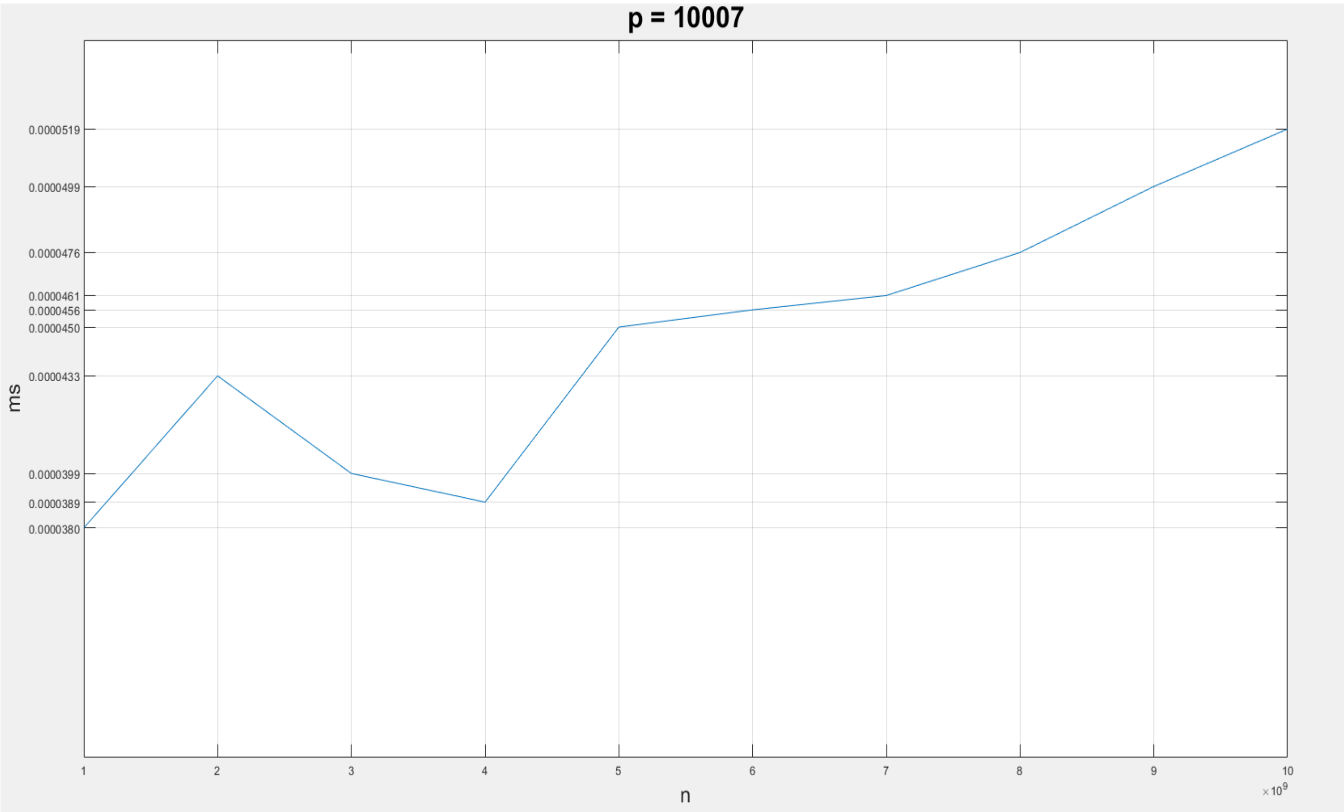




Plots of Algorithm 3:




p = 1009
0.0000515
0.0000495
0.0000470
0.0000458
0.0000455
0.0000434
0.0000418
0.0000412
0.0000371
0.0000351
ms
1
2
3
4
5
6
7
8
9
10
n
×10^9


p = 10007
0.0000519
0.0000499
0.0000476
0.0000461
0.0000456
0.0000450
0.0000433
0.0000399
0.0000389
0.0000380
ms
1
2
3
4
5
6
7
8
9
10
n
×10^9

Discussion:

- Since the numbers are large, it can be easily seen from the graph that Algorithm 1's time complexity is linear, which is O(N). If the numbers were smaller, it would be not easy to see the linear relation.
- If there were much more n values in the plots of Algorithm 2, it would easier to see it has O(1) time complexity. Moreover, if the range of y-axis were larger, it would make easier as well. However, to have better observation, less range of y-axis and fewer n values were used. Therefore, the plots of Algorithm 2 are not very accurate. By looking only at the plots, it is very hard to see Algorithm 2 has O(1) time complexity.
- Approximately every plot of Algorithm 3 has increased when n gets greater. However, the third plot is not defining the time complexity of Algorithm 3 explicitly. It gets less run time when n is converted from $2\ x\ 10^9$to $3\ x\ 10^9$. Nevertheless, if the n values were much more, then plots would be more accurate. It would be easier to see Algorithm 3 has O(logN) time complexity.
- In brief, the plots would show the time complexities of algorithms more accurately if we had used much more n values. Moreover, with a better computer and more ideal conditions, the time complexities would be easier to see from the plots.