Alper Bozkurt
21802766

Question 1:

The first algorithm (Naive algorithm) contains one loop. “n” is the number that indicates how many times “a” multiplies itself. Every time it multiplies itself taking modulus “p” as a result. After n times of looping the result returned. Thus, the time complexity will be O(n) for the worst case. It is also O(n) for average and best case because the loop in all circumstances cycling “n” times.

The second algorithm (Naive algorithm with Cycle shortcut) contains two loops. However, time complexity totally depends on the “p” value for both loops. The first loop transfers to the if statement when p - 1 equals the i (i is the index value going to the n from 1) according to Fermat’s Theorem which states a^(p-1)=1(modp). From the if statement, function is called again but this time n is equal to p-1 thus the complexity again becomes p-1 but these two loops are not nested but the second one is another one where we quit the first loop. So the time complexity will be the sum of these two loops which will be O(2(p-1)) which is equal to O(1).

The third algorithm (Recursive Algorithm) contains itself. Every time it calls itself, there will be O(1) time complexity. But searching in an algorithm happens like binary search in this situation. It divides the n value into 2 so it will search the function with the speed of power of two. It means the time complexity will be O(logn).

Question 2:
Macbook Pro 2020 (Apple M1) chip is used.
CPU: Apple M1 chip 8-core(4 performance cores and 4 efficiency cores)
Memory: 8GB
Storage: 256 GB SSD

Question 3:

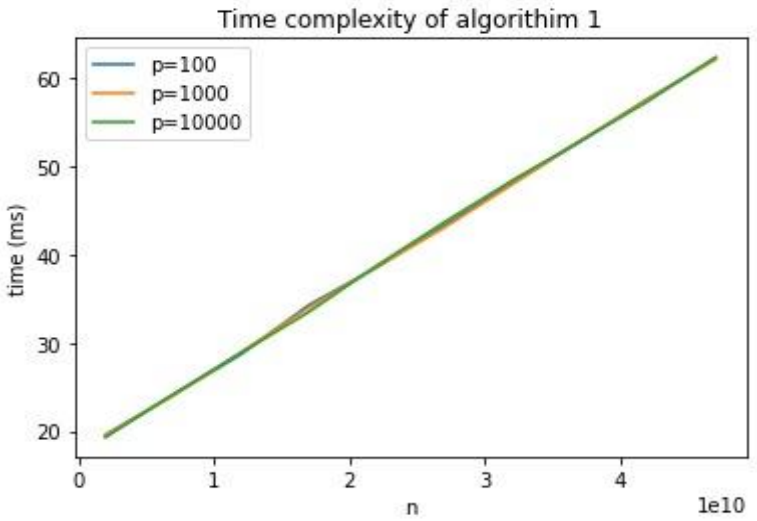
(“a” is constant, and it is 10005)
(Values are milliseconds(ms))
(decimal points are ignored for the values > 10.000 ms (19.35 ms = 19ms))
(“.” between algorithm 1 symbolizes the “thousand” values for better read in algorithm 1)

	Algorithm 1			Algorithm 2			Algorithm 3		
n	p = 100	p = 1000	p = 10000	p = 100	p = 1000	p = 10000	p = 100	p = 1000	p = 10000
2.000.000.000	19.571	19.529	19.279	0.001	0.008	0.007	0.001	0.001	0.001
2.500.000.000	24.081	24.002	24.164	0.001	0.01	0.013	0.001	0.001	0.001
3.000.000.000	28.688	28.923	28.867	0.001	0.006	0.011	0.001	0.001	0.001
3.500.000.000	34.259	33.980	33.490	0.001	0.008	0.009	0.001	0.001	0.002
4.000.000.000	38.522	38.577	38.722	0.001	0.011	0.008	0.001	0.001	0.001
4.500.000.000	43.311	43.057	43.663	0.001	0.007	0.013	0.001	0.001	0.001
5.000.000.000	48.077	47.929	48.381	0.001	0.01	0.011	0.001	0.002	0.002
5.500.000.000	52.774	52.885	52.817	0.001	0.012	0.009	0.002	0.002	0.001
6.000.000.000	57.618	57.566	57.346	0.001	0.007	0.007	0.002	0.003	0.002
6.500.000.000	62.128	62.117	62.356	0.001	0.009	0.014	0.001	0.002	0.002

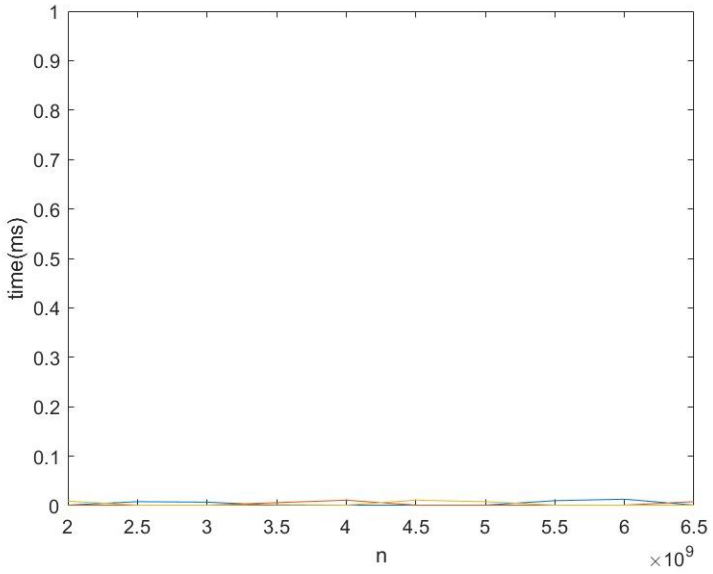
Question 4:

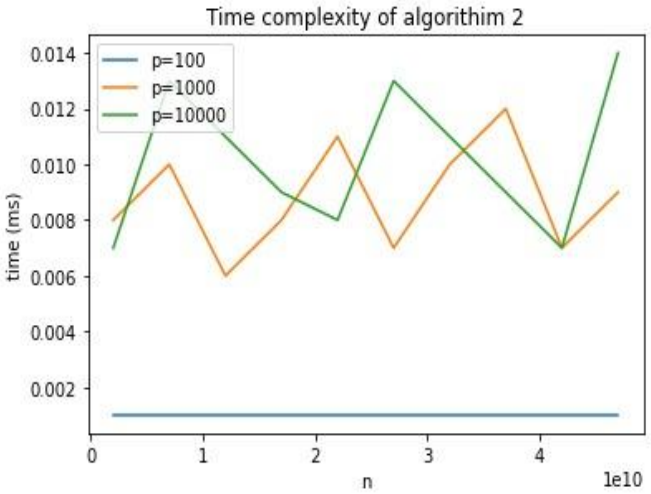
For the second and third algorithms, I increase the scale to the detailed observation of the increase and decrease to understand the mistake rate clearly. It is too much fluctuates because the scale is too much compared to the first graph. But also, other versions are added to be understood how little the increase is.

Algorithm1:

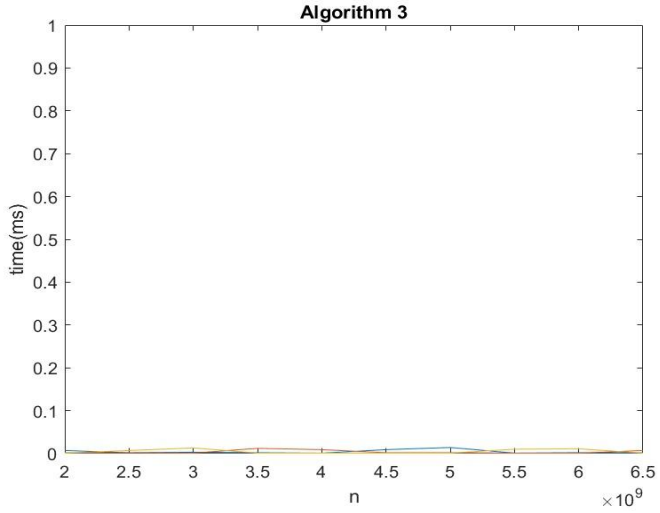


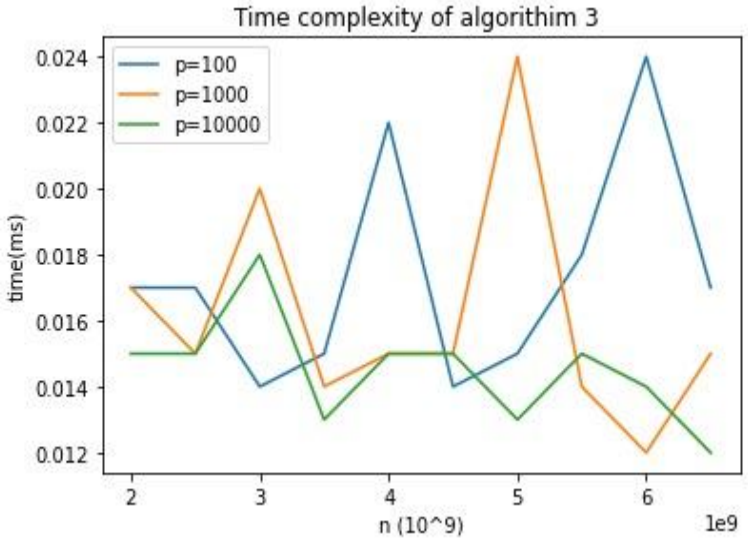
Algorithm2:




Time complexity of algorithim 2
p=100
p=1000
p=10000
time (ms)
0.014
0.012
0.010
0.008
0.006
0.004
0.002
0
1
2
3
4
n
1e10

Algorithm 3:


Algorithm 3
time(ms)
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
2
2.5
3
3.5
4
4.5
5
5.5
6
6.5
n
×10^9


Time complexity of algorithim 3
p=100
p=1000
p=10000
time(ms)
0.024
0.022
0.020
0.018
0.016
0.014
0.012
2
3
4
5
6
n (10^9)
1e9