Date: 06/04/2022

Course: CS201
Section: 2
Assignment: 2
Name: İlayda Zehra Yılmaz
ID: 22001769

Question 1:

The first algorithm has O(n) complexity:
The first algorithm has a for loop that loops until it reaches n and its step size is 1 (meaning it loops through n elements 1 by 1). The loops exucutes the statements inside itself n times. The statements inside the loop have O(1) complexity on their own so they don't affect function's complexity as adding constants to n does not affect the overall complexity. The statements outside for loop also have O(1) comlexisties so they don't affect the complexity too. Therefore the function's complexity is for loop's complexity which is O(n).

Second algorithm has O(n) complexity:
In this algorithm the worst case complexitry is O(n) because when it cannot find the index it loops through the whole array for the mod calculation. So in the worst case algorithm loops once to find index which is O(n) and it loops again to calculate which is O(n) sum of these two complexities gives O(n) and the other statements give O(1) which does not affect function's complexity (the last loop is never entered because we didn't find the index so it more be more than n. Therefor algorithym works just like Algorithm 1). Because of this my graphs are mostly linear.
However, when the index is found the algorithm works faster and I predict this is the case when my "Algorithm 2, p = 101" graph decreases instantly. So in this case the complexity will not be O(n) as this is the best case and its compexity will be calculated by the sum of the first for loop that gives O(n) while looking for the index, the second for loop that loops until it reaches index which gives O(i) complexity (i being the index we are looking for) and the last loop that loops until (n)mod(i) value. But we are not looking for the worst case which is the upper bound so this only explains some differences in the graph but does not affect the O(n) complexity which is the general complexity.

Third algorithm has O(log(n)) complexity:
The recurence relation of this algorithm is
$T(n) = T(n/2) + \Theta(1)$
$T(1) = \Theta(1)$
because every step of the recursion takes the n as n/2 so the value that it must recurance through gets smaler like this:
$T(n) = T(n / 2) + \Theta(1)$
$T(n) = T(n / 4) + \Theta(1) + \Theta(1)$
end eventually we react to the general case which is:
$T(n) = T(n / 2^k) + k\ \Theta(1)$
$T(n) = T(1) + \log n\ \Theta(1)$
$T(n) = \Theta(1) + \Theta(\log n)$
$T(n) = \Theta(\log n)$

This is because our algorithm uses a tempMod value which makes it rememeber the answer of recurence relations it calculates so it does not have to calculate two recurance relations in every recursion. So it acts like a for loop that divides its way to half in every loop.When if we assume it exacutes x times, x would be our complexity and x can be found by:

$1 = n/2^x$

$x = log_2(n)$

Because base 2 is not important this will give us O(log(n))

Question 2:

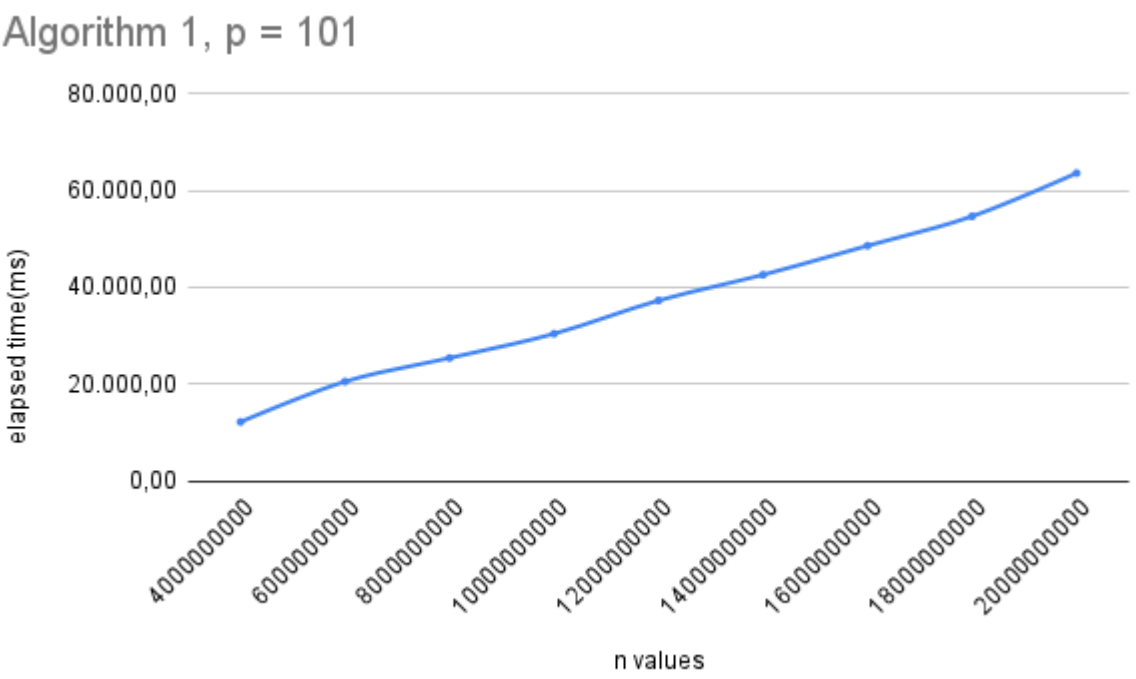
OS: macOS Catalina
Model: MacBook Air (Retina, 13-inch, 2020)
Model Identifier: MacBookAir9,1
Processor Name: Quad-Core Intel Core i5
Processor Speed: 1,1 GHz
Number of Processors: 1
Total Number of Cores: 4
L2 Cache (per Core): 512 KB
L3 Cache: 6 MB
Hyper-Threading Technology: Enabled
Memory: 8 GB 3733 MHz LPDDR4X
Boot ROM Version: 1715.81.2.0.0 (iBridge: 19.16.10744.0.0,0)
Serial Number (system): FVFDK4DTMNHX
Hardware UUID: 97BBBA82-49EF-5052-8B01-ABF4744FD8EC
Activation Lock Status: Enabled
Graphics: Intel Iris Plus Graphics 1536 MB
Storage: (For this homework Machintosh HD is used)
Machintosh HD: 75 GB available of the total 170 GB
BOOTCAMP: 13,93 GB available of total 80,68 GB

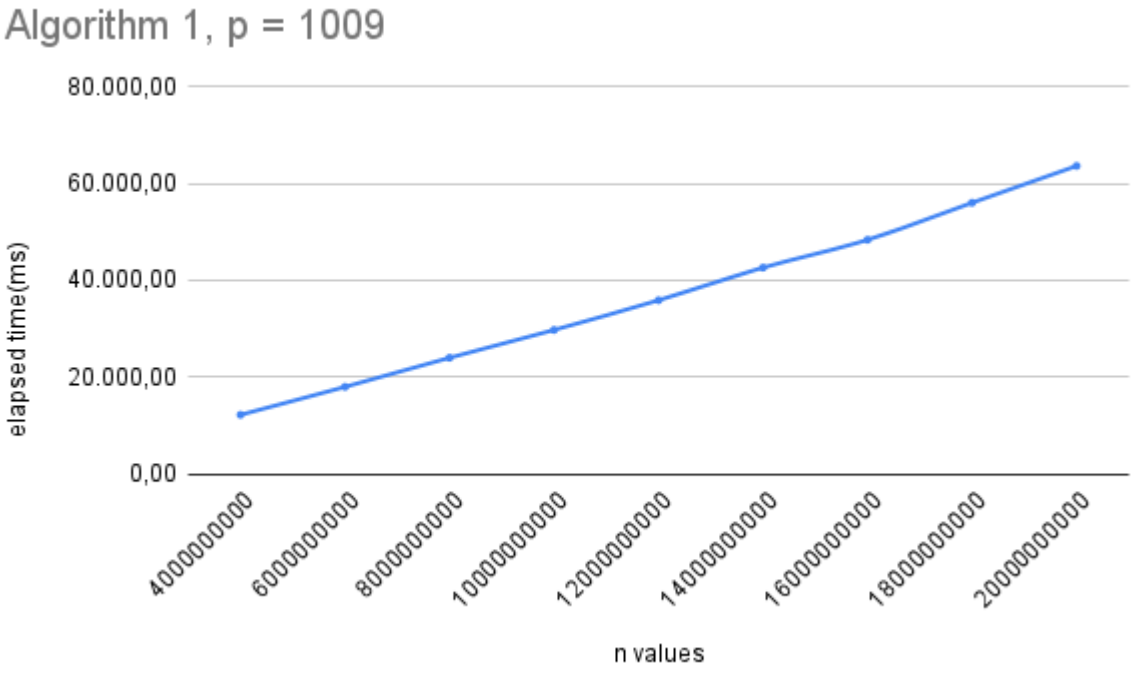
Question 3:

In the below table, x is the value of how many times the function is called to get that time value. I only called the recursive functions more than once as you can see.

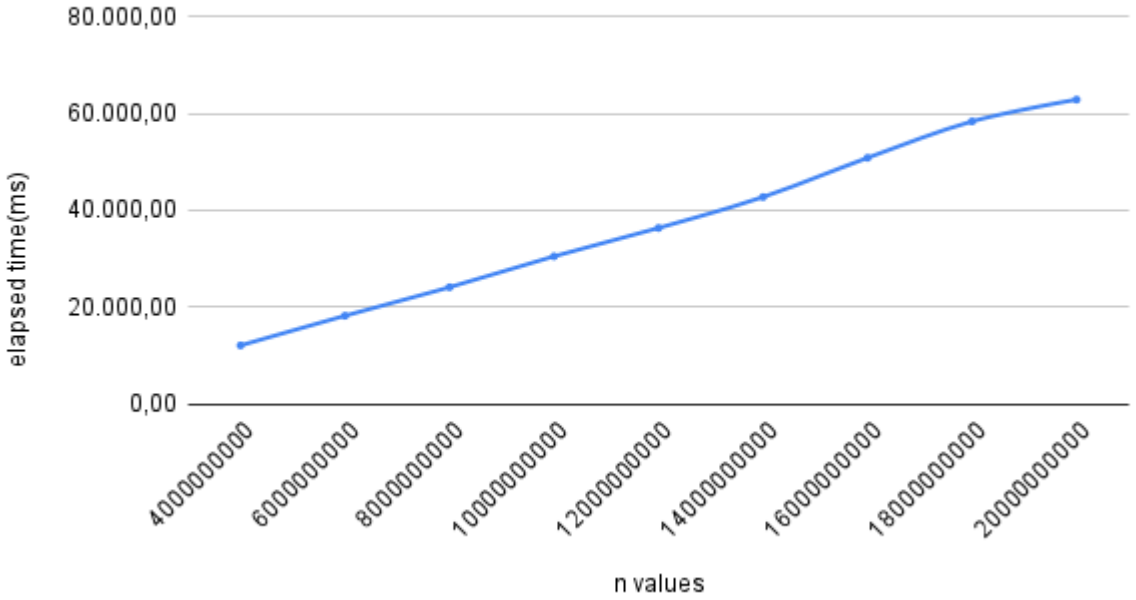
	Algorithm 1 (ms)	Algorithm 1 (ms)	Algorithm 1 (ms)	Algorithm 2 (ms)	Algorithm 2 (ms)	Algorithm 2 (ms)	Algorithm 3 (ms)	Algorithm 3 (ms)	Algorithm 3 (ms)
	p = 101	p = 1009	p = 10007	p = 101	p = 1009	p = 10007	p = 101	p = 1009	p = 10007
	(x =1)	(x =1)	(x =1)	(x =1)	(x =1)	(x =1)	$(x = 2 * 10^8)$	$(x = 2 * 10^8)$	$(x = 2 * 10)^8$
$n = 2 * 10^9$	6442.7	6037.81	6095.3	13163.50	12057.20	13078.1	56348.50	60904.00	59804.1
$n = 4 * 10^9$	12171.90	12195.80	12067.70	25289.60	24545.60	25933.90	75199.20	68580.40	72607.70
$n = 6 * 10^9$	20517.70	17976.30	18204.10	39190.00	36198.30	37576.90	74494.20	78325.90	79717.70
$n = 8 * 10^9$	25395.00	23990.70	24109.20	48549.90	48803.10	49084.30	78084.90	86629.10	78070.30
$n = 10 * 10^9$	30407.00	29750.00	30493.90	66682.60	61370.10	61719.40	79717.70	87462.40	80097.30
$n = 12 * 10^9$	37293.20	35892.60	36363.70	95643.80	73726.80	74111.60	83957.80	88573.00	81777.00
$n = 14 * 10^9$	42608.00	42639.60	42762.50	84379.40	85896.40	85137.30	8274.00	91994.10	83016.70
$n = 16 * 10^9$	48618.20	48396.10	50875.30	97352.10	97545.30	97493.00	84463.40	91499.50	82001.90
$n = 18 * 10^9$	54698.20	56055.90	58406.90	108965.00	110579.00	108496.00	86878.30	93904.00	85111.50
$n = 20 * 10^9$	63633.89	63670.90	62912.00	121891.00	122040.00	121609.00	87575.70	94350.50	88671.00

Question 4:

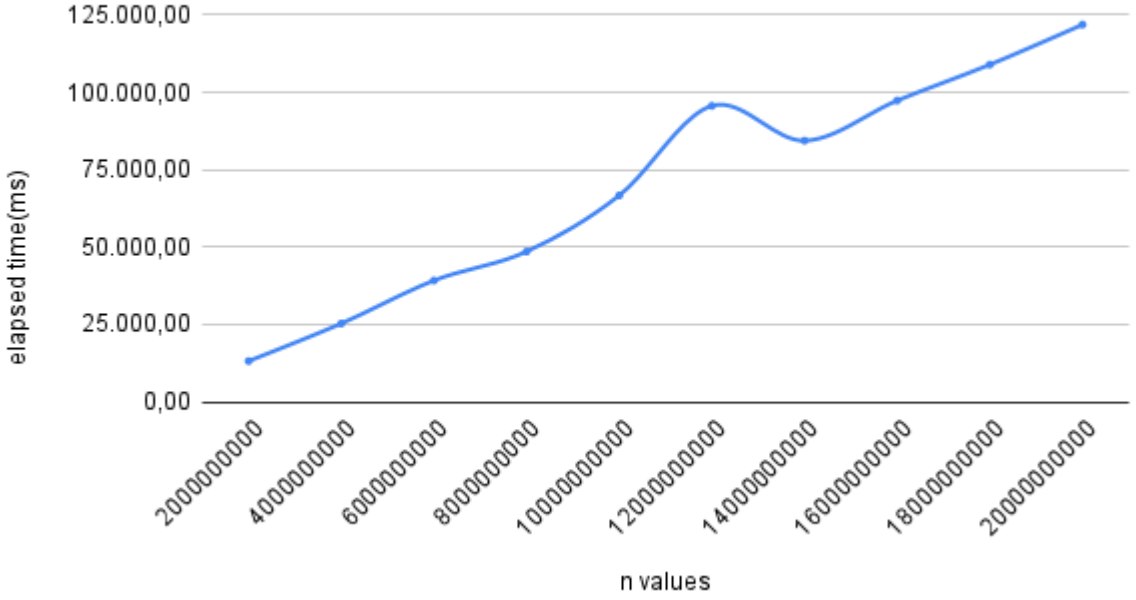

Algorithm 1, p = 101
80.000,00
60.000,00
40.000,00
20.000,00
0,00
elapsed time(ms)
4000000000
6000000000
8000000000
10000000000
12000000000
14000000000
16000000000
18000000000
20000000000
n values


Algorithm 1, p = 1009
80.000,00
60.000,00
40.000,00
20.000,00
0,00
elapsed time(ms)
4000000000
6000000000
8000000000
10000000000
12000000000
14000000000
16000000000
18000000000
20000000000
n values

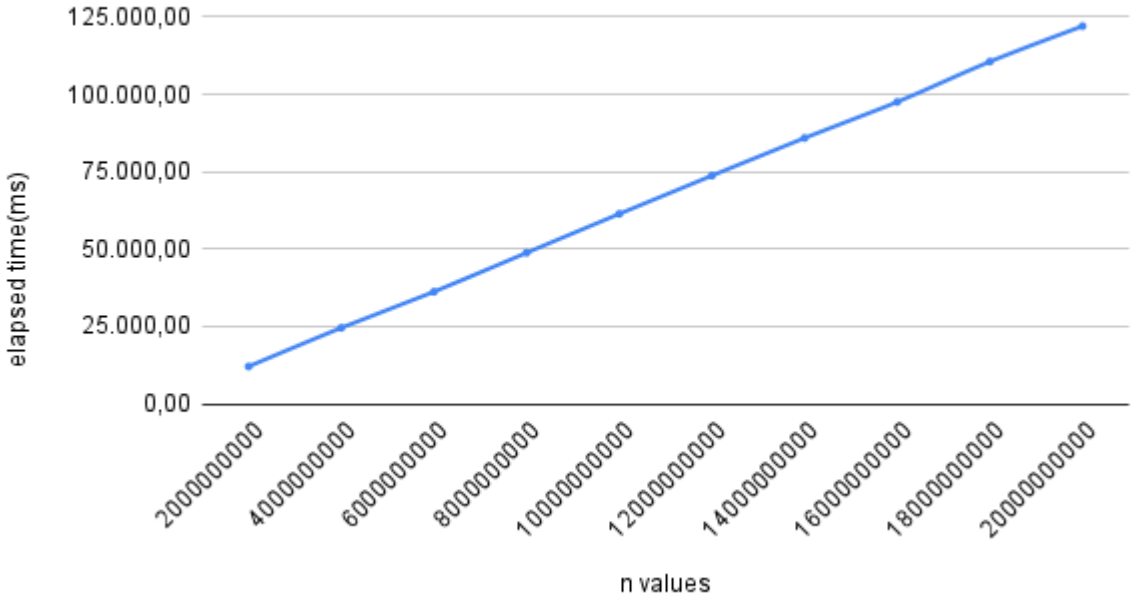
Algorithm 1, p = 10007



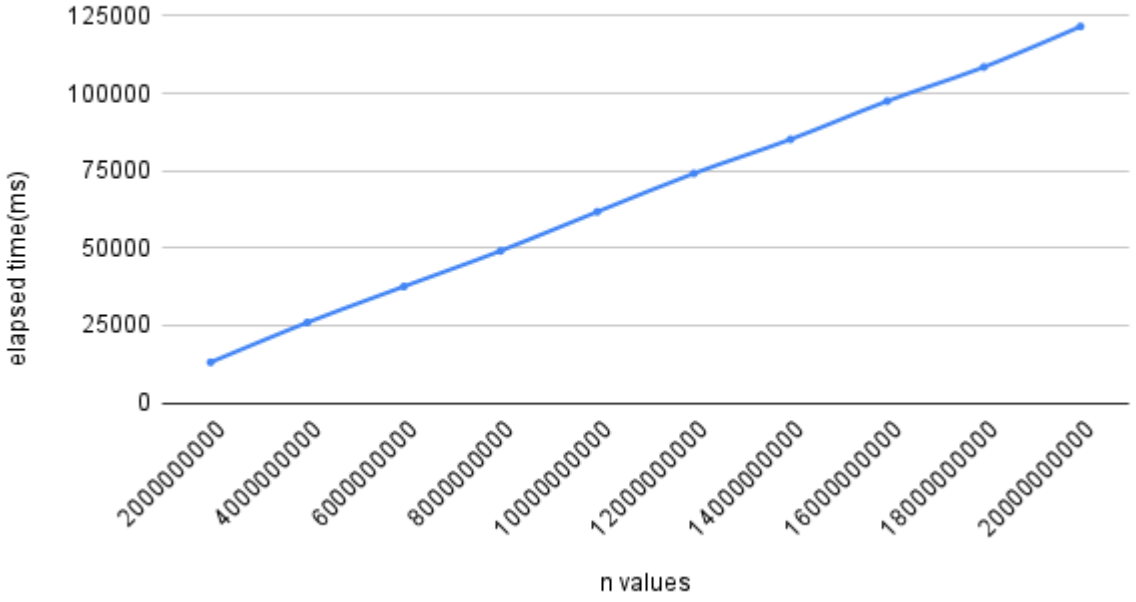
Algorithm 2, p = 101

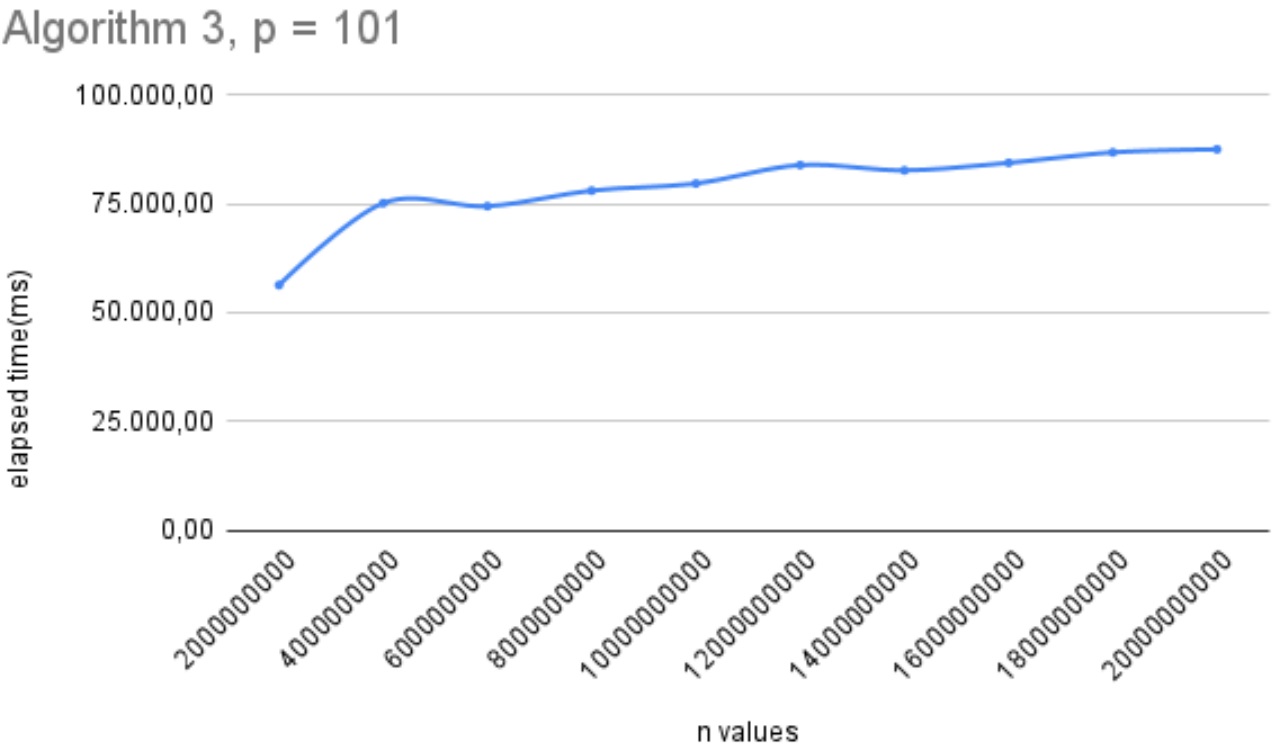


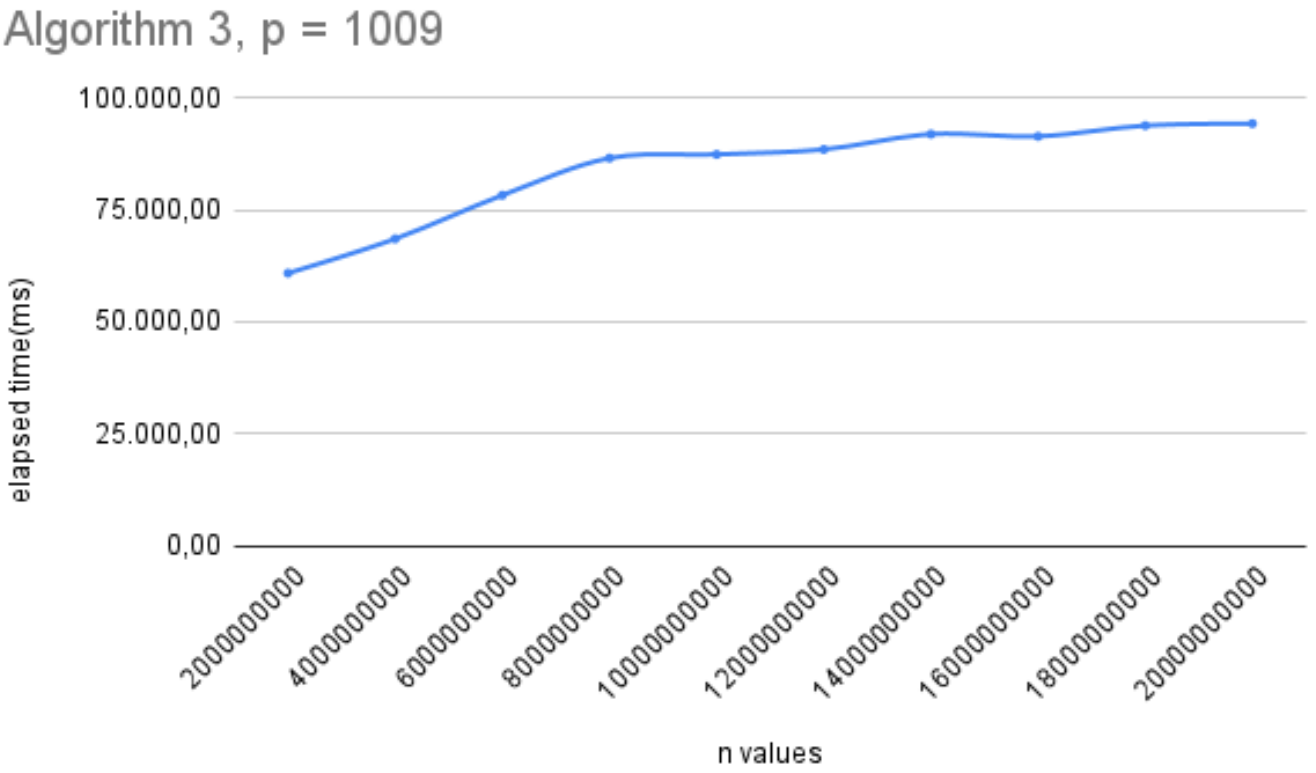
Algorithm 2, p = 1009

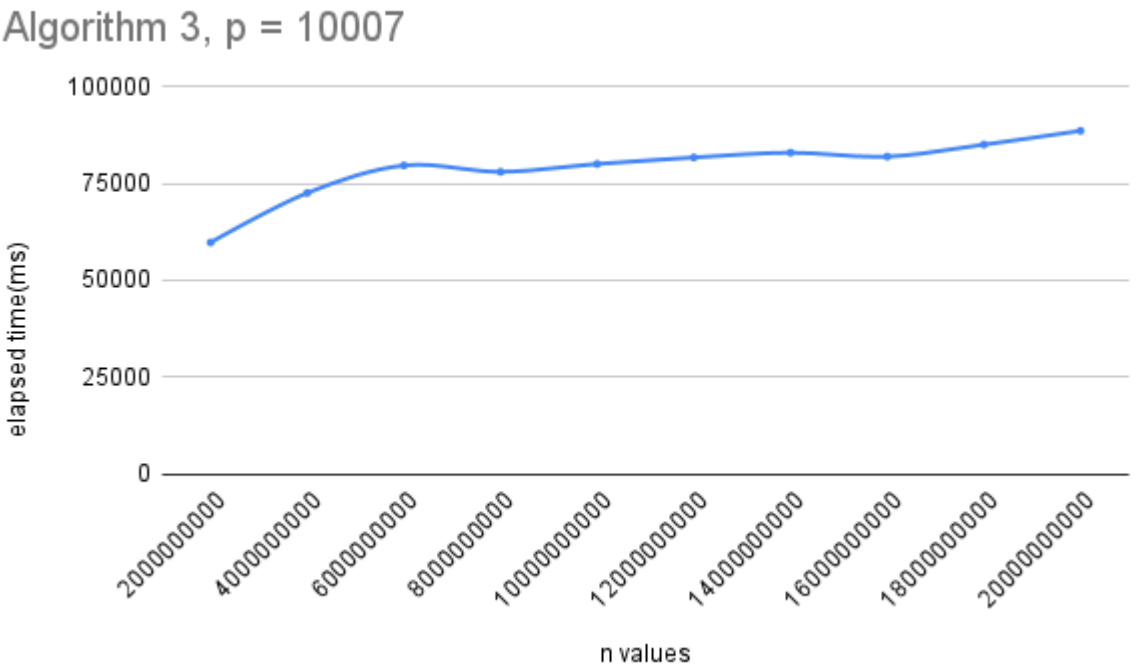


Algorithm 2, p = 10007




Algorithm 3, p = 101
100.000,00
75.000,00
50.000,00
25.000,00
0,00
elapsed time(ms)
2000000000
4000000000
6000000000
8000000000
10000000000
12000000000
14000000000
16000000000
18000000000
20000000000
n values


Algorithm 3, p = 1009
100.000,00
75.000,00
50.000,00
25.000,00
0,00
elapsed time(ms)
2000000000
4000000000
6000000000
8000000000
10000000000
12000000000
14000000000
16000000000
18000000000
20000000000
n values


Algorithm 3, p = 10007
100000
75000
50000
25000
0
elapsed time(ms)
2000000000
4000000000
6000000000
8000000000
10000000000
12000000000
14000000000
16000000000
18000000000
20000000000
n values