2021-2022 Spring Semester

CS201- HW2: Algorithm Analysis

Name: İdil Atmaca

ID: 22002491

Course: CS201 – Fundamental Structures of Computer Science 1

Section: 01

Date: 05/04/2022

Computer Specifications

ASUS Vivobook Pro 14 OLED

Processor	11th Gen Intel(R) Core(TM) i5-11300H @ 3.10GHz 3.11 GHz
Installed RAM	16.0 GB (15.7 GB usable)
System type	64-bit operating system, x64-based processor

TIME ELAPSED IN MILLISECONDS

N	ALGORITHM 1			ALGORITHM 2					
	p = 101	p = 1009	p = 10007	p = 101	p = 1009	p = 10007	p = 50	p = 100	p = 1000
10^8	6557.31	6734.78	6467.47	34.37	1666.82	88416.8	8973.84	9386.58	8829.03
2*10^8	12814.52	12728.81	12633.7	30.702	1185.69	86585.2	16801.1	16837.1	17225.2
3*10^8	19549.74	18499.8	19363.6	31.368	1213.97	86098.6	26447.8	25065.2	25455.8
4*10^8	25535.73	24573.2	26050.9	30.853	705.744	91396.3	34672.3	35397.9	33405.8
5*10^8	32210.43	31259.8	31482.7	31.109	841.197	121161	42521.1	41599.8	43445
6*10^8	40851.85	39118.51	37687.5	30.678	990.404	113392	50471.6	50037.8	50744.2
7*10^8	46085.32	46017.7	43568.4	31.804	1143.68	88955.7	58198.8	58736.5	58736.5
8*10^8	50001.74	51326.33	49530.6	34.01	759.986	92464.1	68109.4	67204.7	67615.8
9*10^8	55727.83	57154.1	56121.3	41.603	925.386	134515	76812.9	78042.7	75546.2
10*10^8	62432.31	62657.5	64730.87	55.445	1057.6	89910.8	86388.3	84625.4	87068.5

TIME ELAPSED IN MILLISECONDS			
N	ALGORITHM 3		
	p = 101	p = 1009	p = 10007
10^5	49803.51	50072.42	48934.2
2*10^5	52162.71	52045.31	53413.9
4*10^5	54949.73	55763.24	54781.61
8*10^5	58211.52	58260.52	57675.63
16*10^5	60686.35	60886.23	61120.3
32*10^5	63945.72	63854.42	65146.2
64*10^5	65819.13	69177.41	65841.6
128*10^5	69176.23	68600.66	70444.61
256*10^5	73231.64	72079.82	75683.62
512*10^5	75189.81	76436.95	79419.41

k values for each algorithm are given in the graphs

Blue, orange and green lines demonstrates for p = 101, 1009,10007 respectively in figure 1, 2 and 6.

ALGORITHM 1 (Naive algorithm)

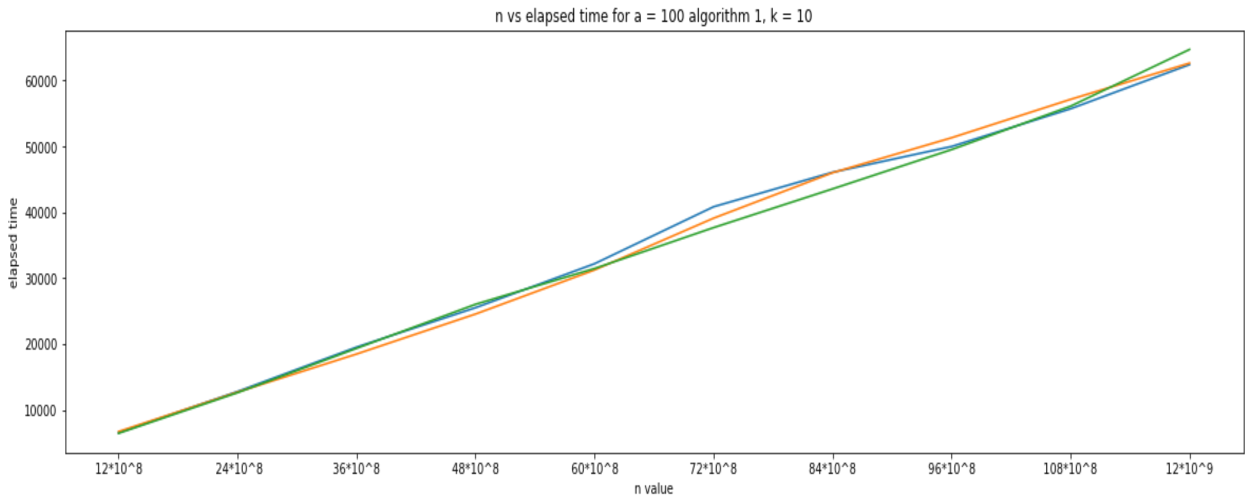

Figure1

ALGORITHM 2 (Naive algorithm with Cycle shortcut)

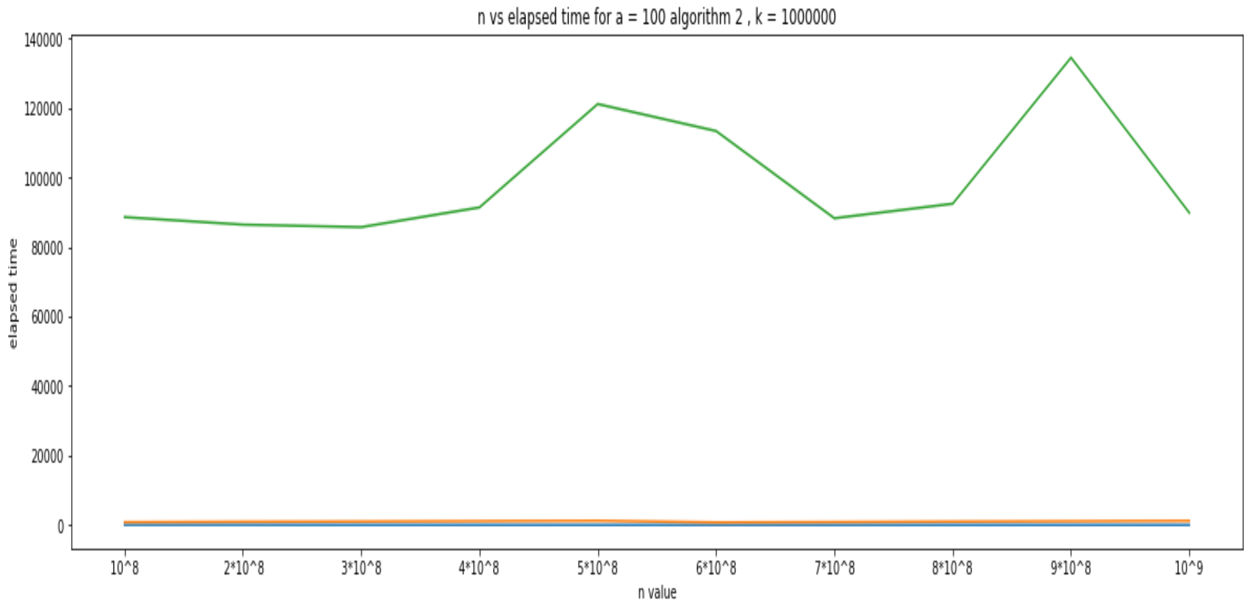

Figure2: shows the time complexity of the algorithm for a and p values that are relatively prime

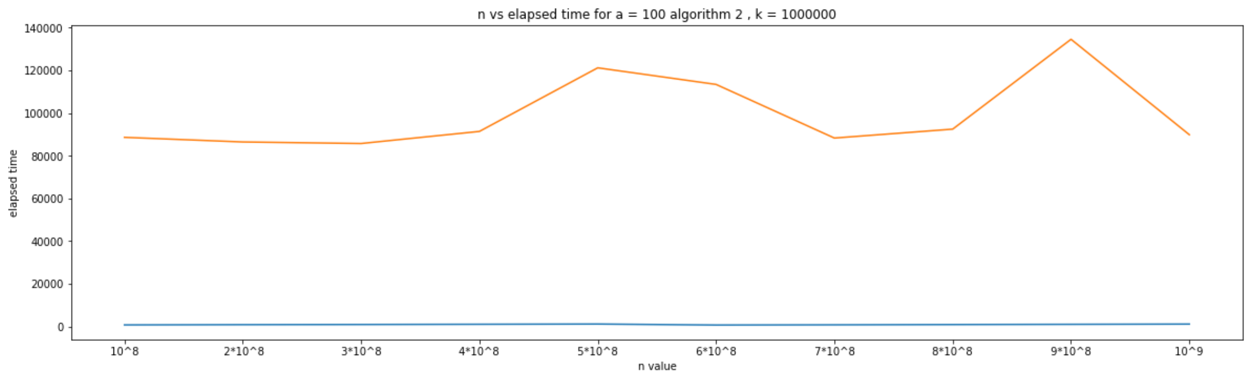

Figure 3: closer look for p = 1009 and p = 10007 for the figure 2

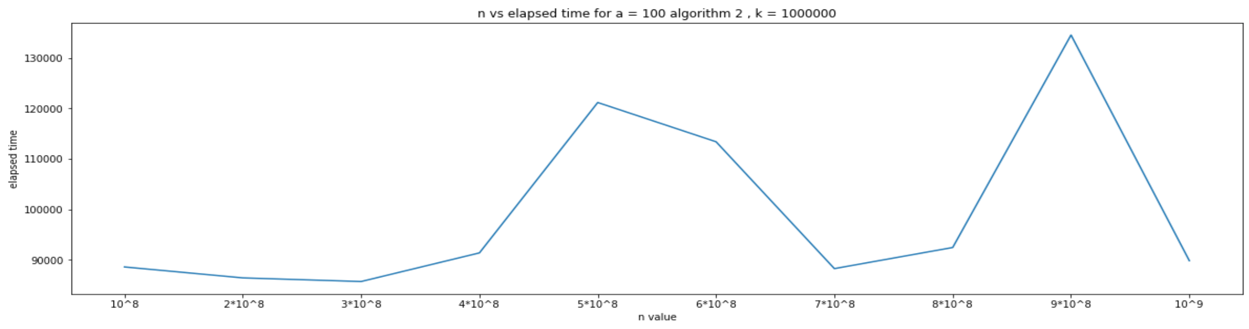

Figure 4: closer look for p = 10007 for figure 3

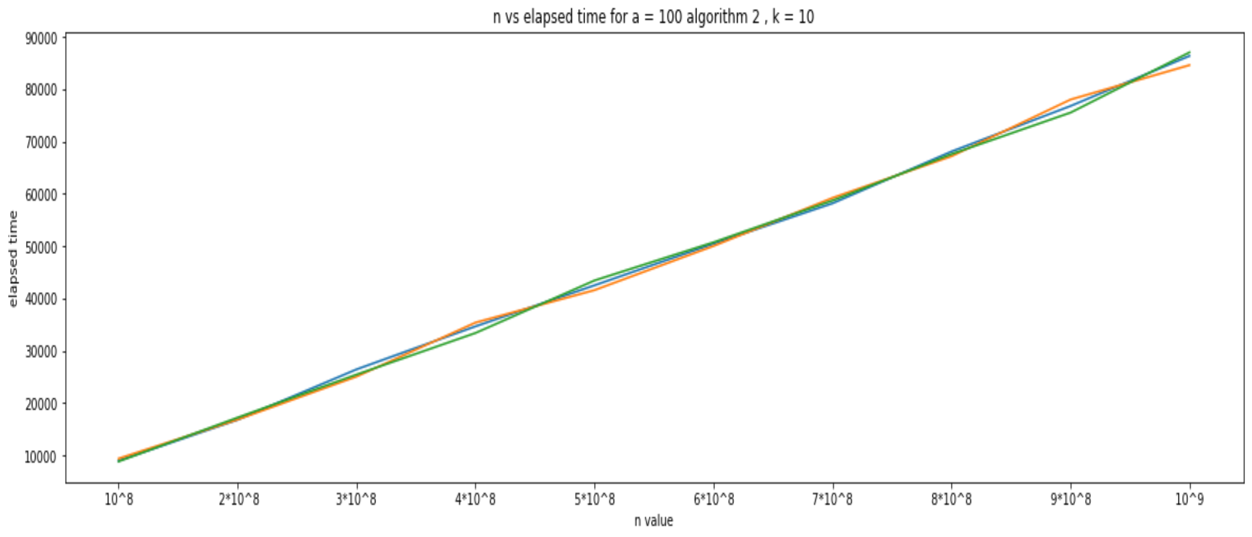

Figure5: shows the time complexity of the second algorithm for which a and p values are not relatively prime

Blue, orange and green lines demonstrate for p = 50, 100,1000 respectively for this figure

ALGORTIHM 3 (Recursive Algorithm)

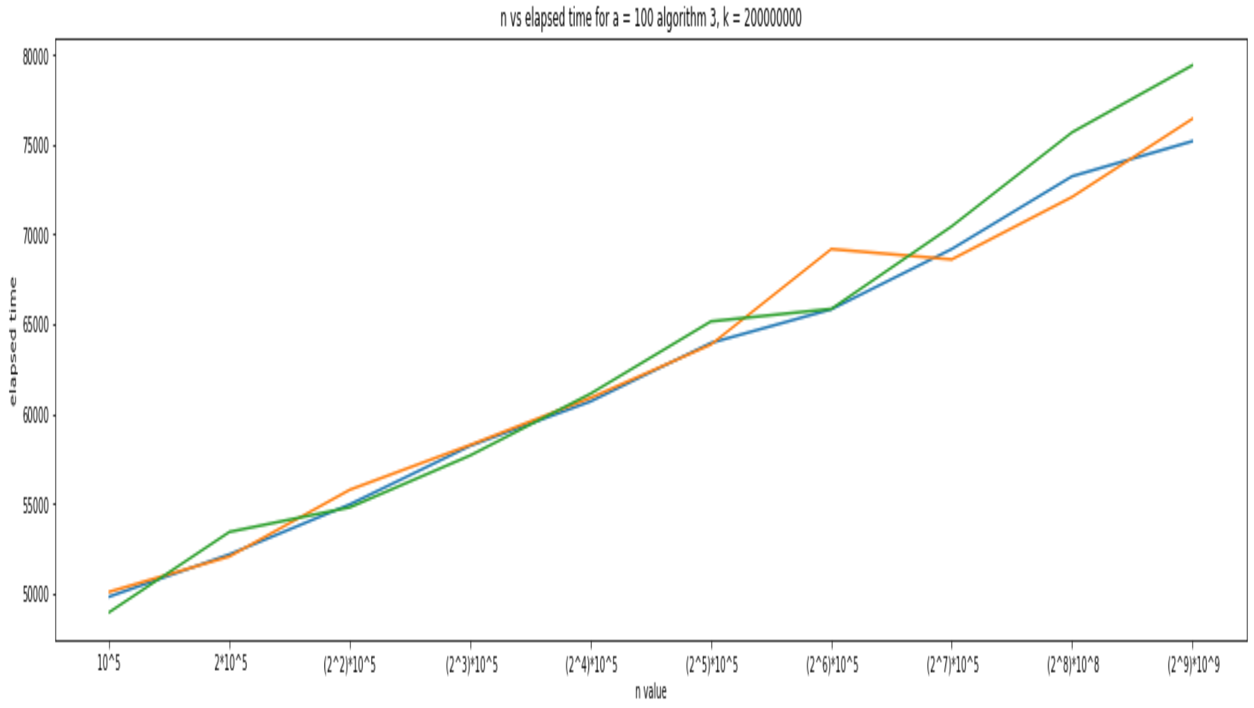

Figure6

N values' increase by 2^n in order to demonstrate a better plot

Results

The first algorithm is the slowest. Its time complexity is O(n), and its plots are linear. This happens because we iterate a for loop n times and update the result every time.

The second algorithms' time complexity depends on the relation between a and p. If they are relatively prime, algorithm's time complexity becomes O(i) because we iterate through the for loop until find i where a^i = p + 1. However, if a and p are not relatively prime, a^i mod p is never equal to 1 inside the scope. Since there are no shortcuts available in that scope, the time complexity for the second algorithm becomes the same as the first one. Thus, the upper bound for the second algorithm's time complexity is O(n) and lower bound is O(i) or O(logp) where log is based a. Some fluctuations happen in the plots because of how close a^i value gets to p + 1. It is clear from the figures that p = 10007 takes much more time than p = 1009 and p = 101. This occurs because until the algorithm finds the required i value, it keeps iterating through the for loop like the first algorithm. Furthermore, for value 10007 it iterates through the loop more relative to 1009 and 101. However, even if the p value is very big, it is still way more efficient than the first algorithm if a and p are relatively prime.

The third algorithms' time complexity is O(logN) since it is recursive. Some fluctuations happen because time complexity can change between odd and even n values. Other than that, the algorithm continuously calls itself until we are left with number a itself and perform the required operations.