CS221 - Section 1 - Homework 2
Burak Öztürk - 21901841

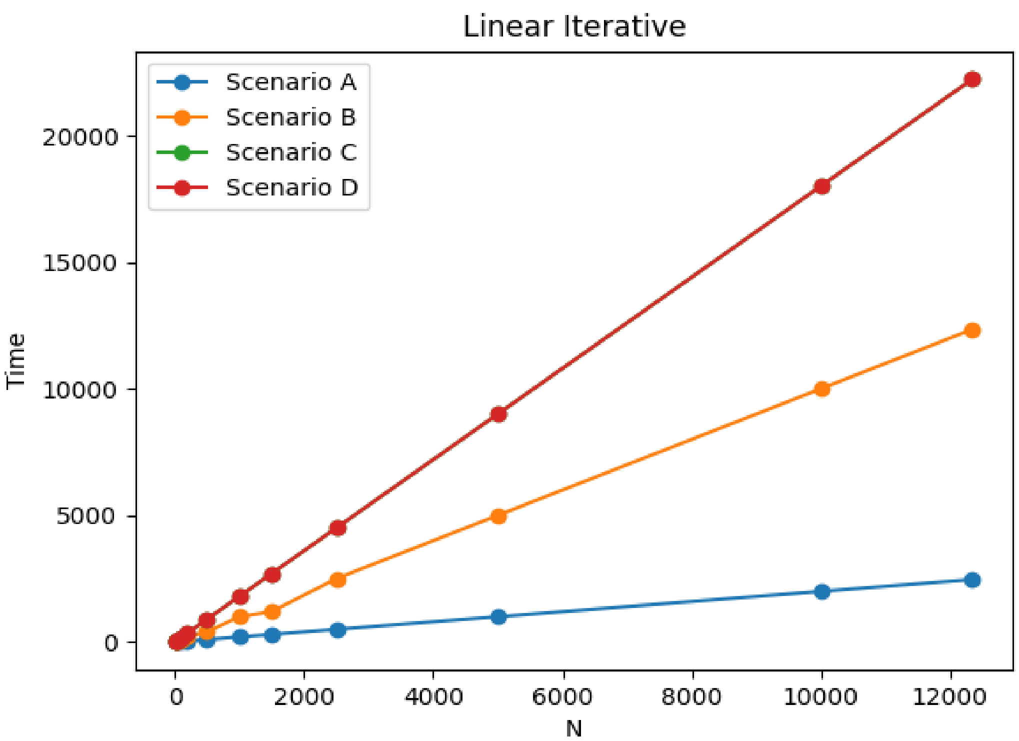
2.1.1. I tested 12 N values. 12345 is chosen to generate more fine outputs like 74074 instead of 58000 just above it.

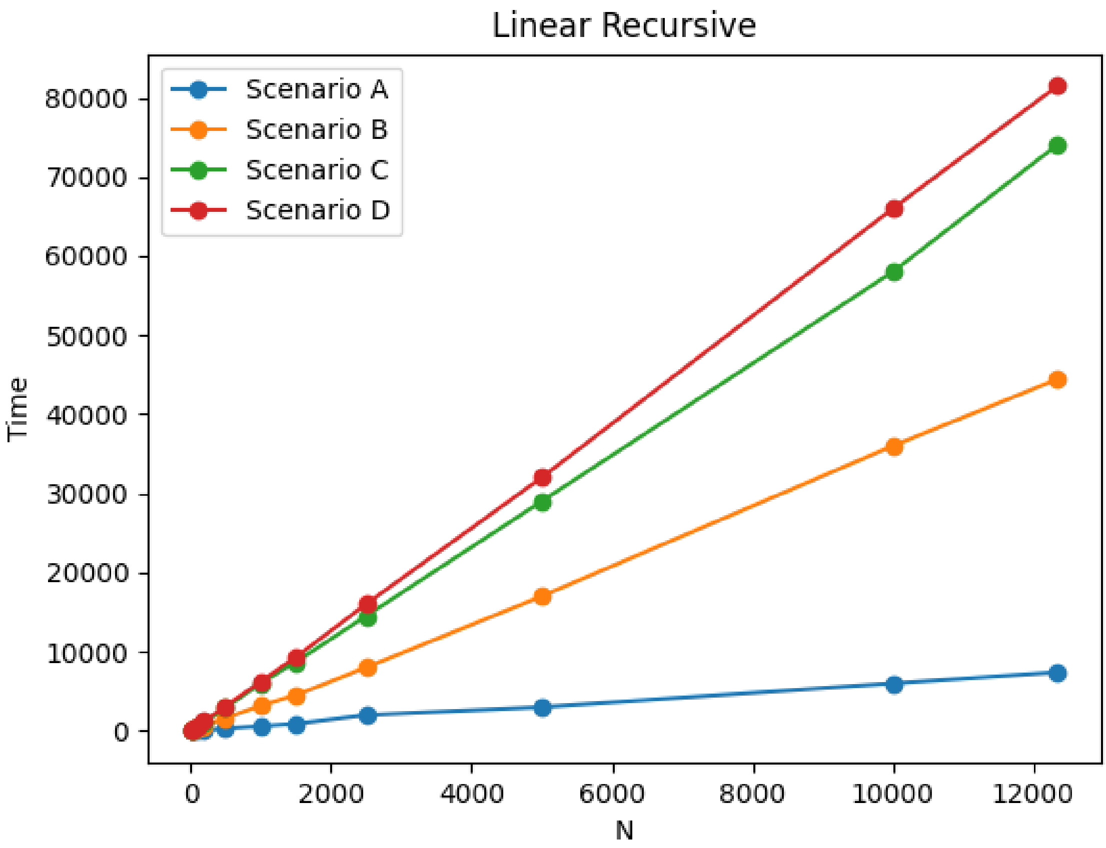
2. I have written the code and gather resulting output.

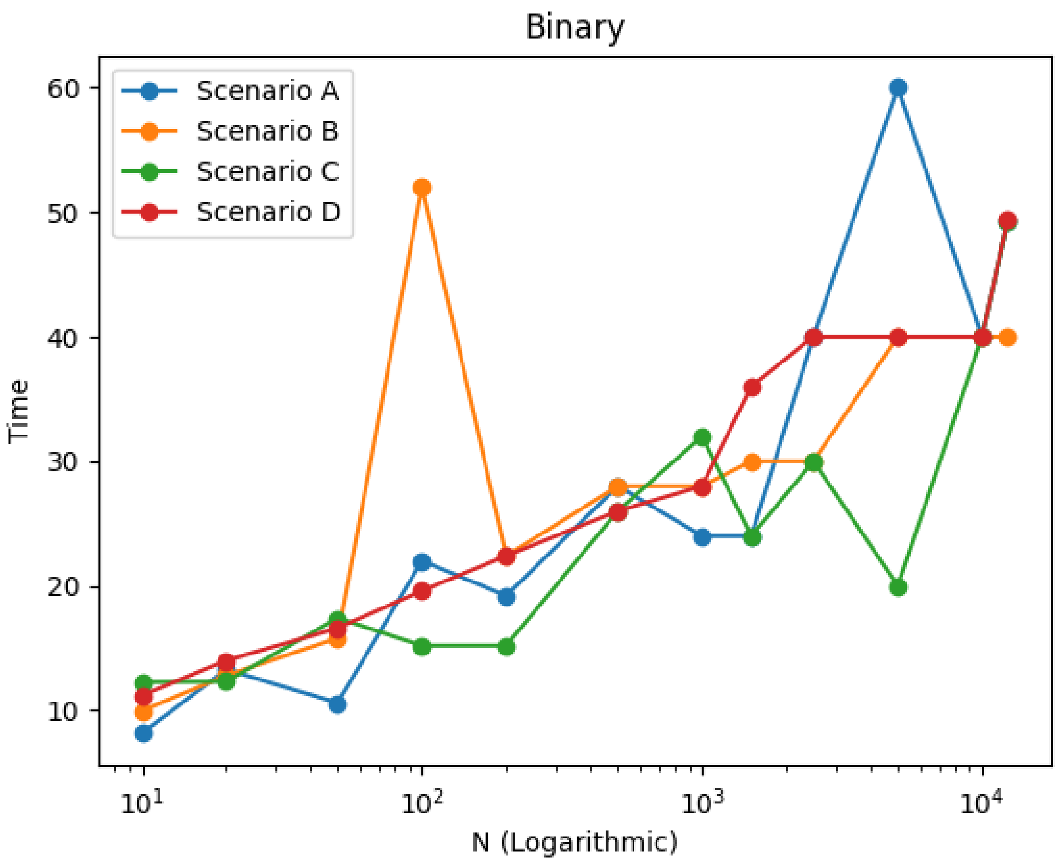
3. Table (Time unit is 10^(-6) milliseconds.):

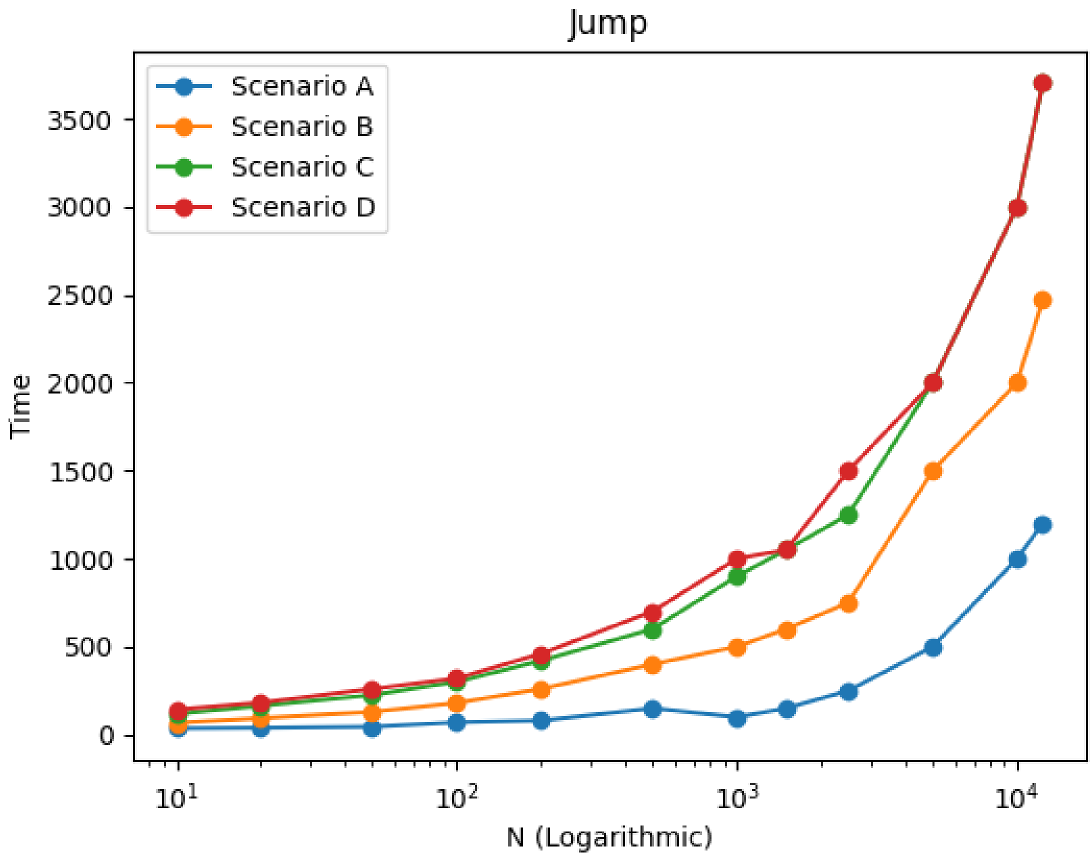
N	Linear (Iterative)				Linear (Recursive)				Binary Search				Jump Search			
	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
10	4	10	18	16	6	20	30	30	8.16	9.96	12.28	11.2	36	67	118	143
20	8	16	40	40	8	36	64	72	13.28	12.8	12.32	14	39	94	164	182
50	10	50	90	100	20	140	280	300	10.6	15.8	17.4	16.6	45	130	225	260
100	20	100	180	180	40	320	540	580	22	52	15.2	19.6	70	180	300	320
200	40	200	360	360	80	600	1120	1240	19.2	22.4	15.2	22.4	80	260	420	460
500	100	400	900	900	300	1600	2900	3100	28	28	26	26	150	400	600	700
1000	200	1000	1800	1800	600	3200	6000	6200	24	28	32	28	100	500	900	1000
1500	300	1200	2700	2700	900	4500	8700	9300	24	30	24	36	150	600	1050	1050
2500	500	2500	4500	4500	2000	8000	14500	16000	40	30	30	40	250	750	1250	1500
5000	1000	5000	9000	9000	3000	17000	29000	32000	60	40	20	40	500	1500	2000	2000
10000	2000	10000	18000	18000	6000	36000	58000	66000	40	40	40	40	1000	2000	3000	3000
12345	2469	12345	22222	22222	7407	44444	74074	81481	49.3	40	49.3	49.38	1200	2469	3703	3703

4. Plots:




Linear Recursive
Scenario A
Scenario B
Scenario C
Scenario D
Time
N
0
2000
4000
6000
8000
10000
12000
0
10000
20000
30000
40000
50000
60000
70000
80000


Binary
Scenario A
Scenario B
Scenario C
Scenario D
Time
N (Logarithmic)
10
20
30
40
50
60



5.1. Specifications of my computer:

Processor: Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz (12 CPUs), ~2.6GHz
RAM: 16384MB
Operating System: Windows 10 Pro 64-bit (10.0, Build 19041) (19041.vb_release.191206-1406)
Model: MONSTER ABRA A5 V15.8
C++ Compiler: gcc version 6.3.0 (MinGW.org GCC-6.3.0-1)

5.2:

Iterative Linear Search:

Best: value = first element of array | time = 1 comparison time

Average: value = mid element of array | time = N / 2 comparisons (N: array size)

Worst: value = last element of array or value is not on the array
time = N comparisons

Recursive Linear Search:

Best: value = first element of array | time = 1 comparison time

Average: value = middle element of array | time = 3N / 2 comparisons and 3 N / 2 function calls

Worst: value = last element of array or value is not on the array
time = 3N comparisons and 3N function calls

Binary Search:

Best: value = middle element of array | time = 1 search iteration

Average: Value can be anywhere for average case | time = log2(n) search iterations

Worst: value = $\lfloor$log2(n) + 1$\rfloor$th element or value is not on the array| time = log2(n) search iterations

Jump Search:

Best: value = first element of array | time = 1 comparison time

Average: value = middle element of array | time = $\sqrt{n}$ / 2 jumps and $\sqrt{n}$ / 2 linear search comparisons

Worst: value = last element of array or value is not on the array | time = $\sqrt{n}$ jumps and $\sqrt{n}$ linear search comparisons

5.3:

Iterative / Recursive Linear Search:

Best case is always Scenario A for my data for both algorithm. This is expected because linear search will always be faster if target is close to start and that is the case for A.

Average case is B and worst case is C and D for the same reason. In B, target is close to middle and in C, target is close to the end. D is worst case too because algorithm haves to search all the way to the end in that case which is as bad as C. C is little bit faster in my data because Scenario C is causing to search until somewhere close to end and D is causing to search to the very end.

Also recursive implementation is little bit slower than iterative because of all the extra function calls and data transfers that recursion requires.

Binary Search:

Best, average and worst cases are fluctuating for this algorithm because binary search's speed is determined by being middle element of the array or sub arrays which log2 function but my target selections are based on 10% steps. This makes the results unstable and pseudo-random.

But Scenario D is stable because in that case, algorithm always checks log2(n) terms and that stabilizes measurements. It can be seen in the plot that Scenario D line is the least fluctuatative. Others huge highs and lows.

Jump Search:

Jump search is like linear search++. Best, average and worst cases are same with the linear search but a lot faster. Because of jumps it has O($\sqrt{n}$) complexity. Everything else works like linear search and my results are suitable to that.

5.4: Take N = 100 and consistent time requirement for every step of algorithm as 1 ms.

Iterative Linear Search:

Cases:

A = 1 ms | B = 50 ms | C = 100 ms | D = 100 ms

Recursive Linear Search:

Cases:

A = 2 ms | B = 75 ms | C = 150 ms | D = 150 ms

(Slower because of the extra work that recursion requires)

Binary Search:

Cases:

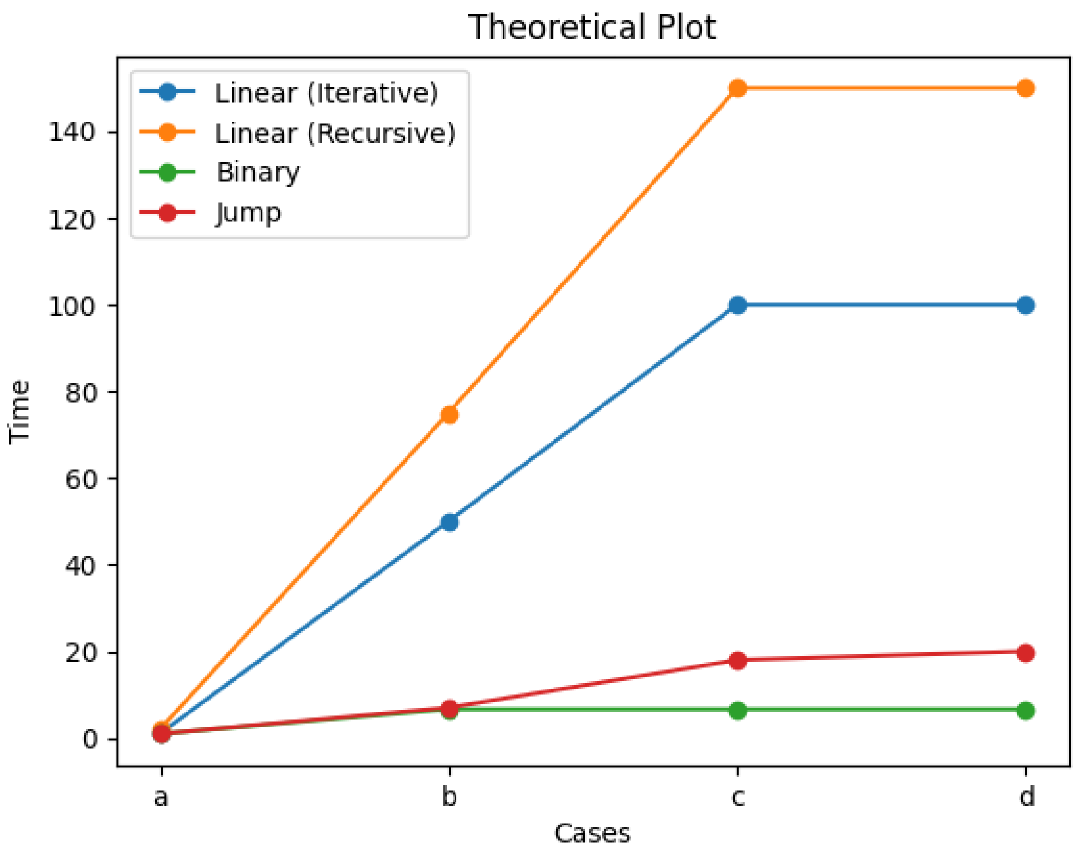
A = 1 ms | B = C = D = log2(100) ≈ 6.64 ms

Jump Search:

Cases:

A = 1 ms | B ≈ 5-8 ms | C ≈ 18 ms | D ≈ 20 ms

Plots:



As seen in the theoretical plot above, all algorithms except binary search behaving proportional to my data. This plot is true according to other plots from different sources too besides being much more coarse.

By this theoretical results, I can say my measurements are not too different from real world applications besides binary search's results with the reason being constant and randomly chosen targets' effect on binary search algorithm's way of working.