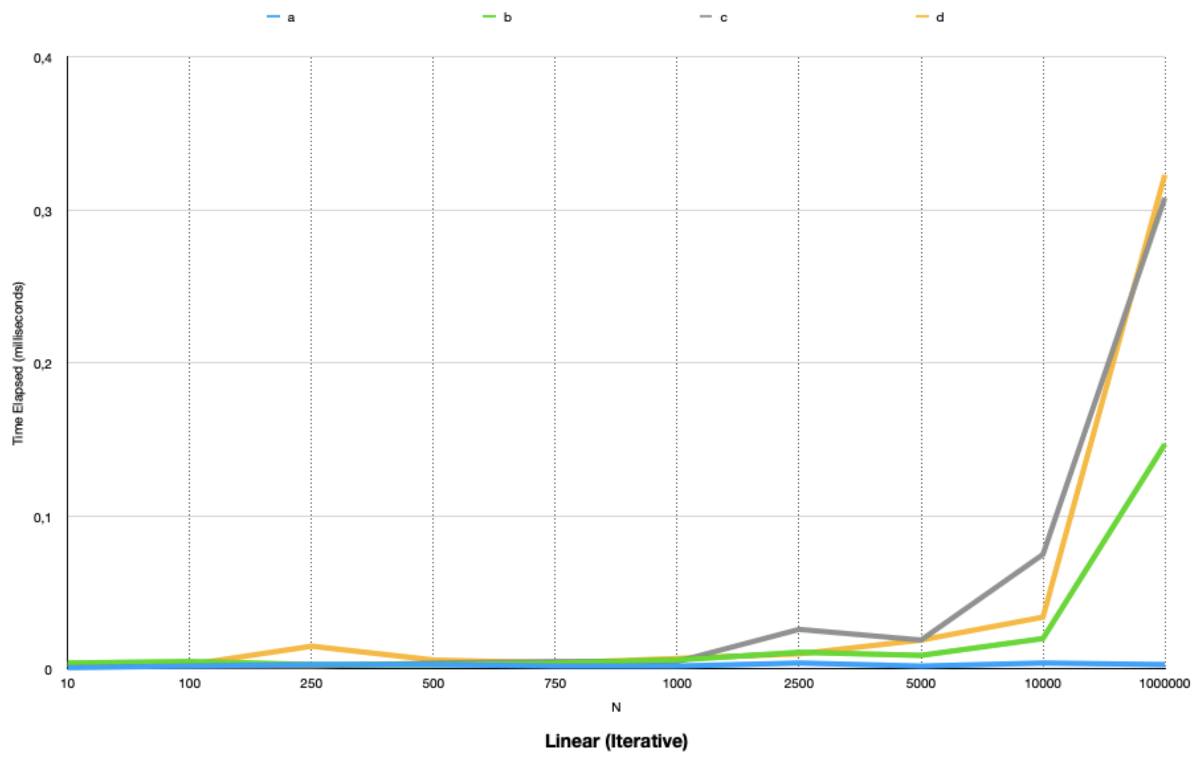
CS 201 HW2

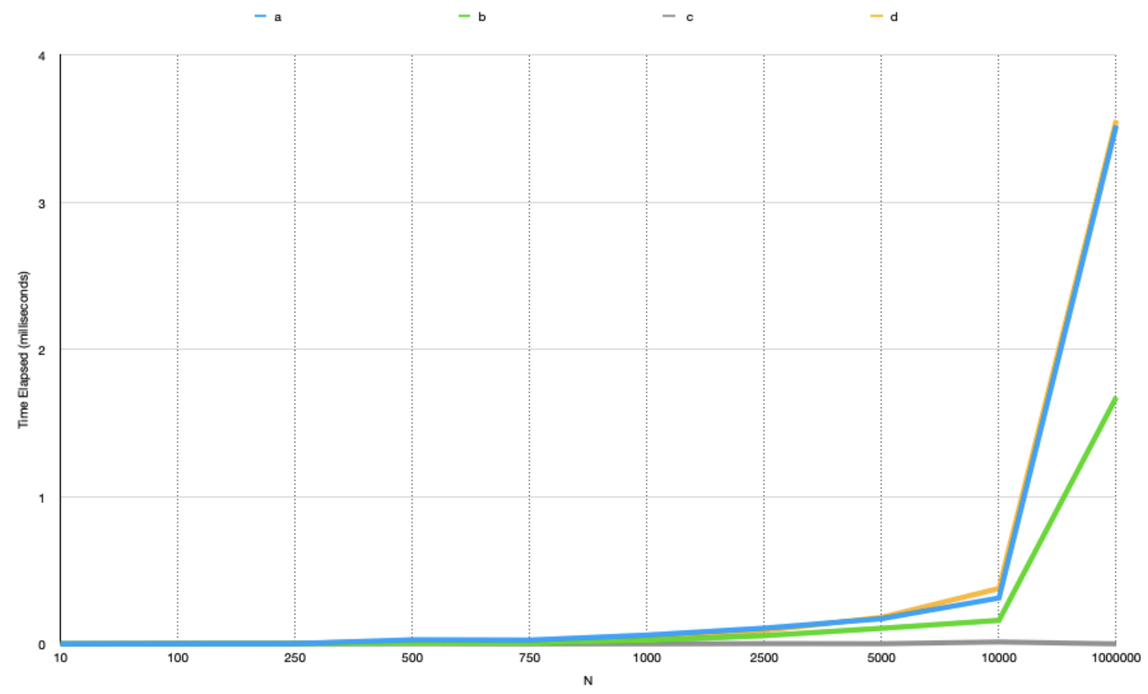
Ceyda Dereci
21602458

In the table below, execution time for arrays of different sizes are concluded for each application.

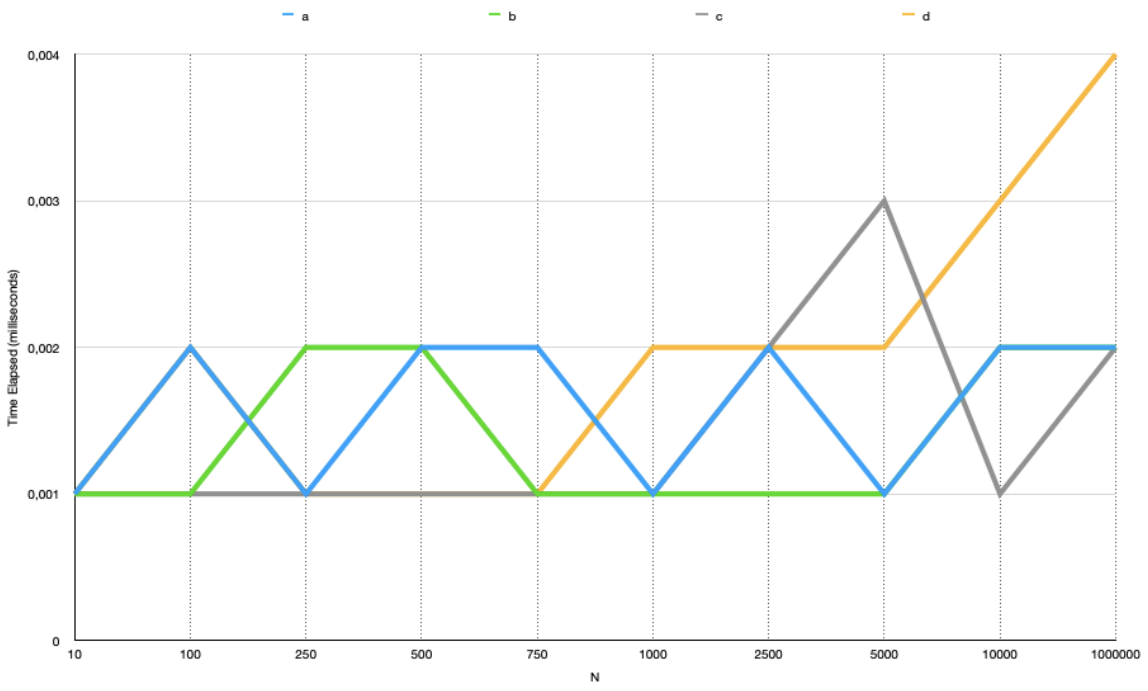
N	Linear (Iterative)				Linear (Recursive)				Binary Search				Jump Search			
	a	b	c	d	a	b	c	d	a	b	c	d	a	b	c	d
10	0,001	0,004	0,003	0,003	0,001	0,002	0,001	0,001	0,001	0,001	0,001	0,001	0,002	0,001	0,001	0,001
100	0,002	0,005	0,002	0,003	0,002	0,002	0,002	0,005	0,002	0,001	0,001	0,002	0,001	0,002	0,001	0,001
250	0,003	0,003	0,002	0,015	0,003	0,003	0,002	0,004	0,001	0,002	0,001	0,001	0,002	0,002	0,002	0,002
500	0,003	0,003	0,004	0,006	0,029	0,008	0,002	0,02	0,002	0,002	0,001	0,001	0,001	0,002	0,001	0,002
750	0,002	0,004	0,005	0,004	0,027	0,007	0,001	0,019	0,002	0,001	0,001	0,001	0,001	0,001	0,001	0,002
1000	0,002	0,006	0,005	0,007	0,061	0,024	0,001	0,035	0,001	0,001	0,001	0,002	0,001	0,002	0,001	0,001
2500	0,004	0,011	0,026	0,01	0,108	0,057	0,003	0,092	0,002	0,001	0,002	0,002	0,002	0,003	0,004	0,001
5000	0,002	0,009	0,019	0,019	0,172	0,107	0,002	0,181	0,001	0,001	0,003	0,002	0,001	0,002	0,005	0,002
10000	0,004	0,02	0,075	0,034	0,313	0,161	0,014	0,378	0,002	0,002	0,001	0,003	0,001	0,003	0,003	0,001
100000	0,003	0,147	0,308	0,323	3,521	1,676	0,001	3,559	0,002	0,002	0,002	0,004	0,001	0,008	0,01	0,001

Based on the table, here is the time-complexity graphs for each algorithm.

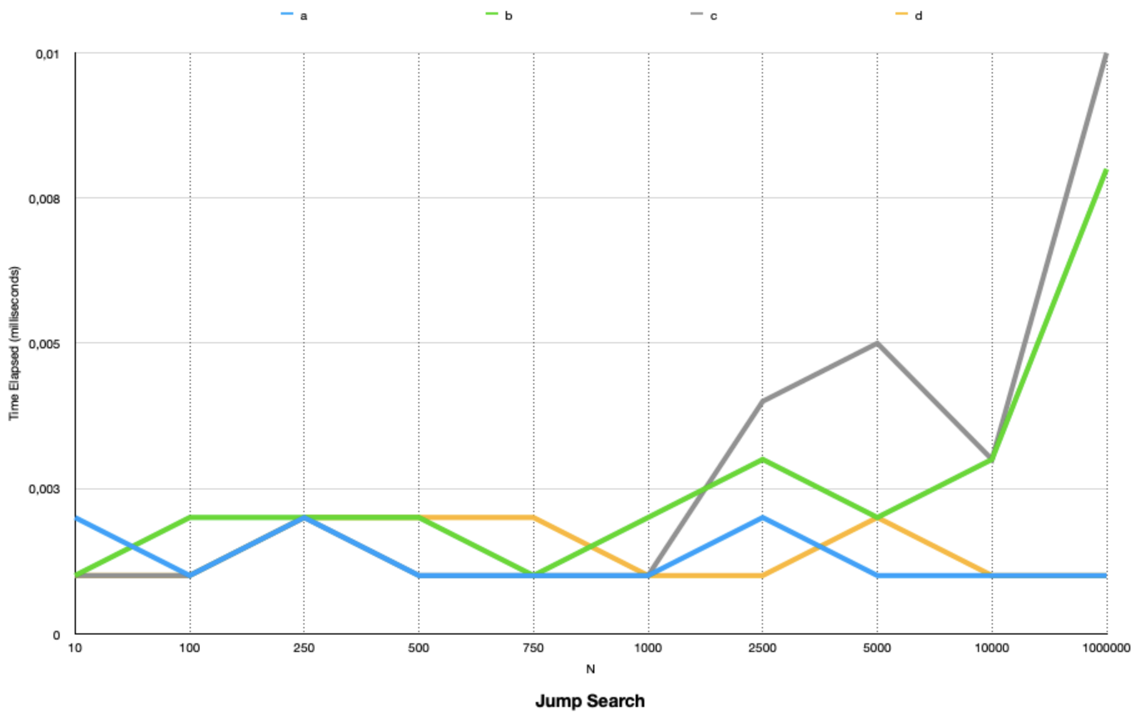




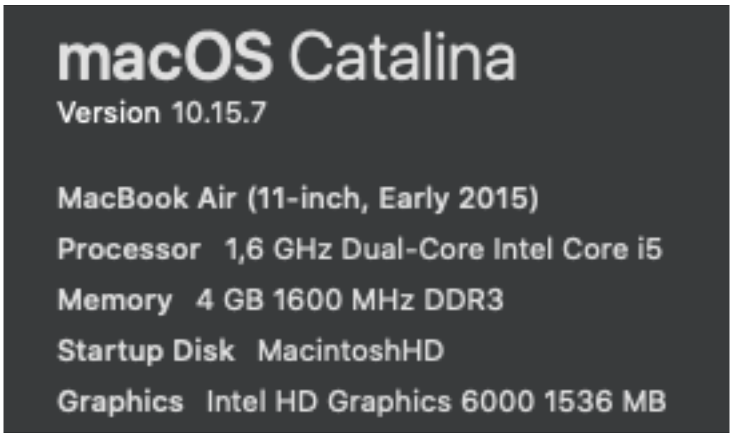
Linear (Recursive)



Binary Search



In below, there is specifications of my computer.



macOS Catalina
Version 10.15.7

MacBook Air (11-inch, Early 2015)
Processor 1,6 GHz Dual-Core Intel Core i5
Memory 4 GB 1600 MHz DDR3
Startup Disk MacintoshHD
Graphics Intel HD Graphics 6000 1536 MB

Theoretical Results

For Linear (Iterative) Search Algorithm,

- The worst case happens when the key is not present in the array. When the key is not in the array, the linearSearch() function compares it with all the elements in the array one by one. Hence, the worst case time complexity of linear search would be O(n).

- The average case, we need to calculate the computing time for every possible inputs. We also need to include the case when the key is not in the array. So, we have n + 1 case. O(1) + O(2) + …. + O(n + 1) = O((n +1) x (n+2)/2)/(n + 1) = O(n).

- The best case, occurs when the key is present at the first location. The number of operations in the best case is constant in other words, not dependent on n. So time complexity in the best case would be O(1).

For Linear (Recursive) Search Algorithm,

The algorithm works mostly same with iterative linear search, only from back to forward. Therefore, worst and the average case are same with the iterative linear search. In the best case, occurs when the key is present at the last location. And all the time complexity is same with iterative linear search.

- The worst case: O(n).
- The average case: O(n).
- The best case: O(1).

For Binary Search Algorithm,

- The worst case, occurs when the key is not present in the array. . And the time complexity is log2(n).
- The average case: The size of the array is n in the beginning. After every iteration, array size will be divided 2. Then after k iteration, the array size is n / (2)^k. Also we now that after k divisions array length will be 1. So, n / (2^k) = 1. Then n = 2^k. Applying logarithm both sides, log2(n) = log2(2^k). We find k = log2(n). Therefore, the time complexity of binary search is log2(n).
- The best case: when the first comparison is correct (the key item is equal to the mid of array). It means, regardless of the size of the list/array, we'll always get the result in constant time. So the best case complexity is O(1).

For Jump Search Algorithm,

- The worst case: n/m jumps, and m-1 comparisons is needed. Hence, the total number of comparisons will be (n/m+(m-1)). This expression has to be minimum, so that we get the smallest value of m. By differentiating this expression we get m = √n.
- The average case: Since the optimal value of m= √n , thus, n/m=√n resulting in a time complexity of O(√n).
- The best case: the key at the edge of the very first block it searches in. Therefore, time complexity is O(1).

Experimental Results

For Linear (Iterative) Search Algorithm,

- The worst case: The larger N, the more accurate we can see. The worst case occurred in the case d when n was the greatest. Therefore it agree with the theoretical results.
- The average case: 0.028 milliseconds according to the table.
- The best case: The blue line (case a) is almost always below the other lines. So, case a is the best case as in the theoretical results.

For Linear (Recursive) Search Algorithm,

- The worst case: The blue and yellow lines (case a and d) are the worst cases for the table mostly. And also see that some places d is above all of the lines. It almost accurate with the theoretical results. Since n is relatively small, also case a is really close to the case d.
- The average case: 0,265175 milliseconds.
- The best case: The grey line (case c) is always below the other cases. So it’s the best case as in the theoretical results.

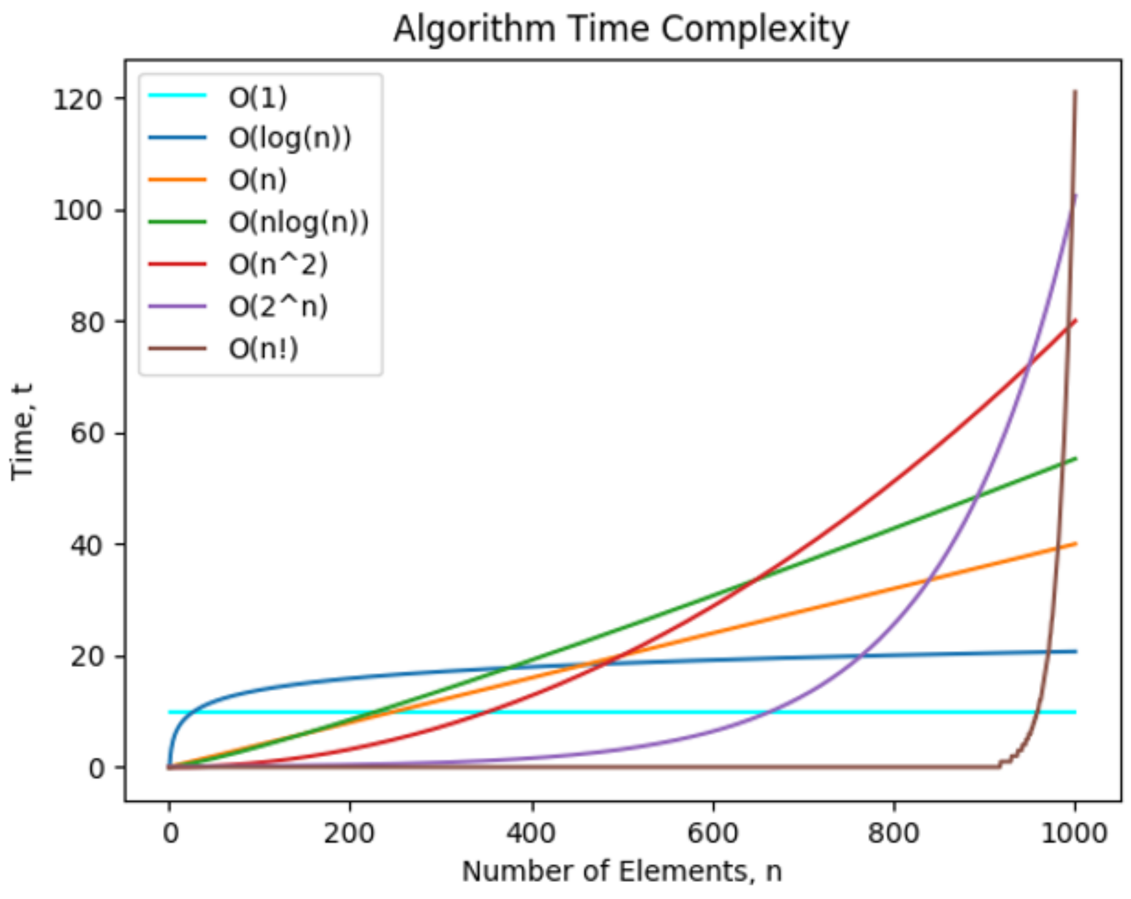
For Binary Search Algorithm,

- The worst case: According the graph the worst case occurred in the yellow line (case d).
- The average case: 0.001575 milliseconds.
- The best case: The green line (case b) is one which mostly below the other lines. However, the results are not quite presice because the results were too close to each others.

For Jump Search Algorithm,

- The worst case: according the graph it occurred at the case c.
- The average case: 0.00205 milliseconds.
- The best case: the blue line (case a) is mostly below the other lines. Therefore, according the graph the best case occurred at case a.

Theoretical time complexity graph is at the below.


Algorithm Time Complexity
O(1)
O(log(n))
O(n)
O(nlog(n))
O(n^2)
O(2^n)
O(n!)
Time, t
Number of Elements, n