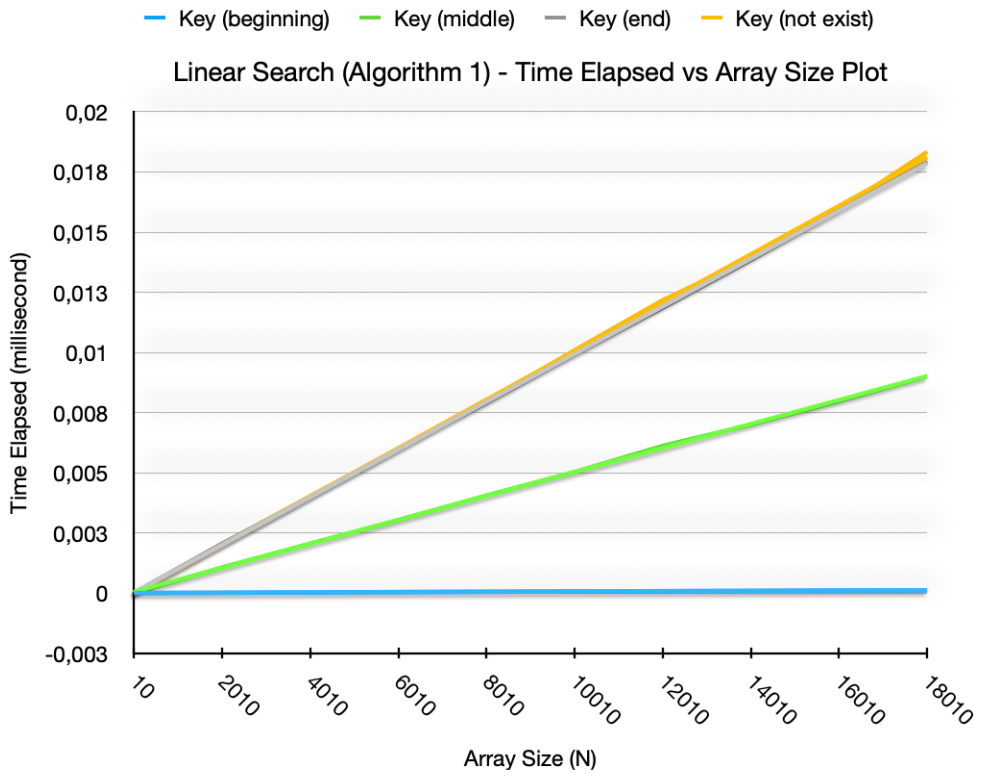
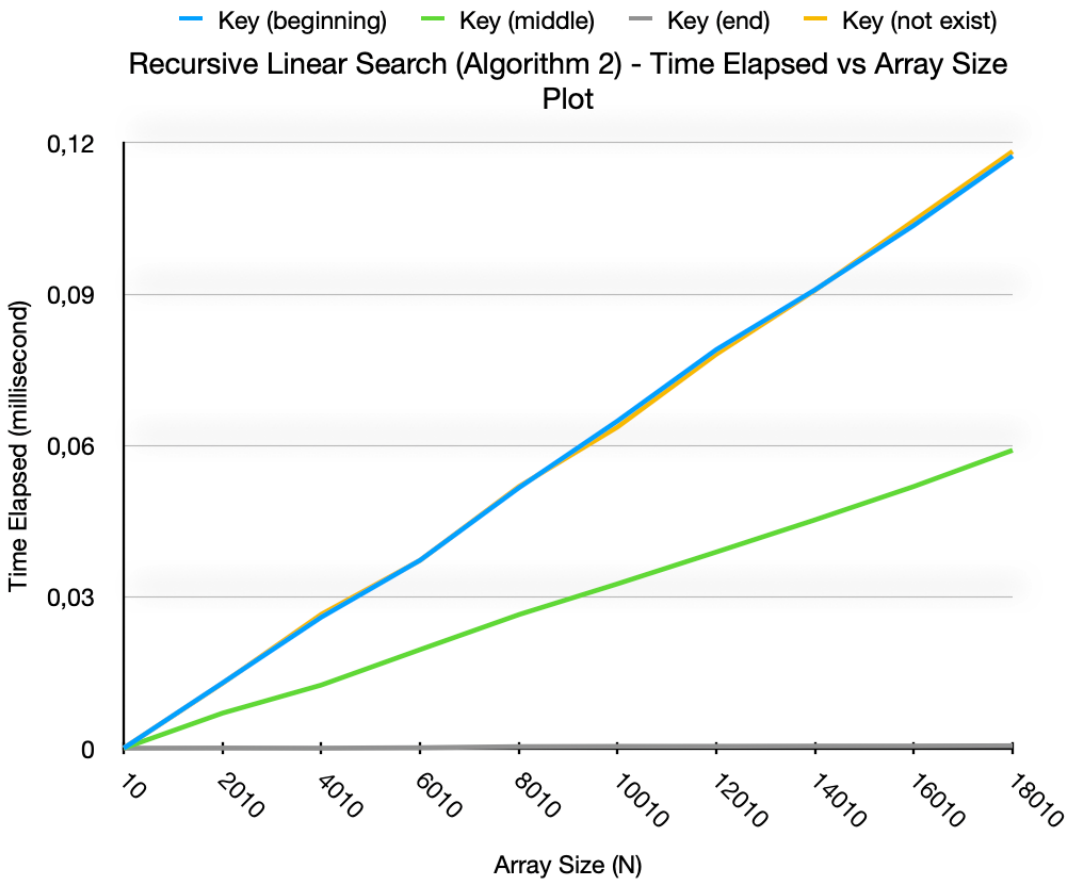
# CS 201 – Homework Assignment 2
## Report of Analysis of Search Algorithms

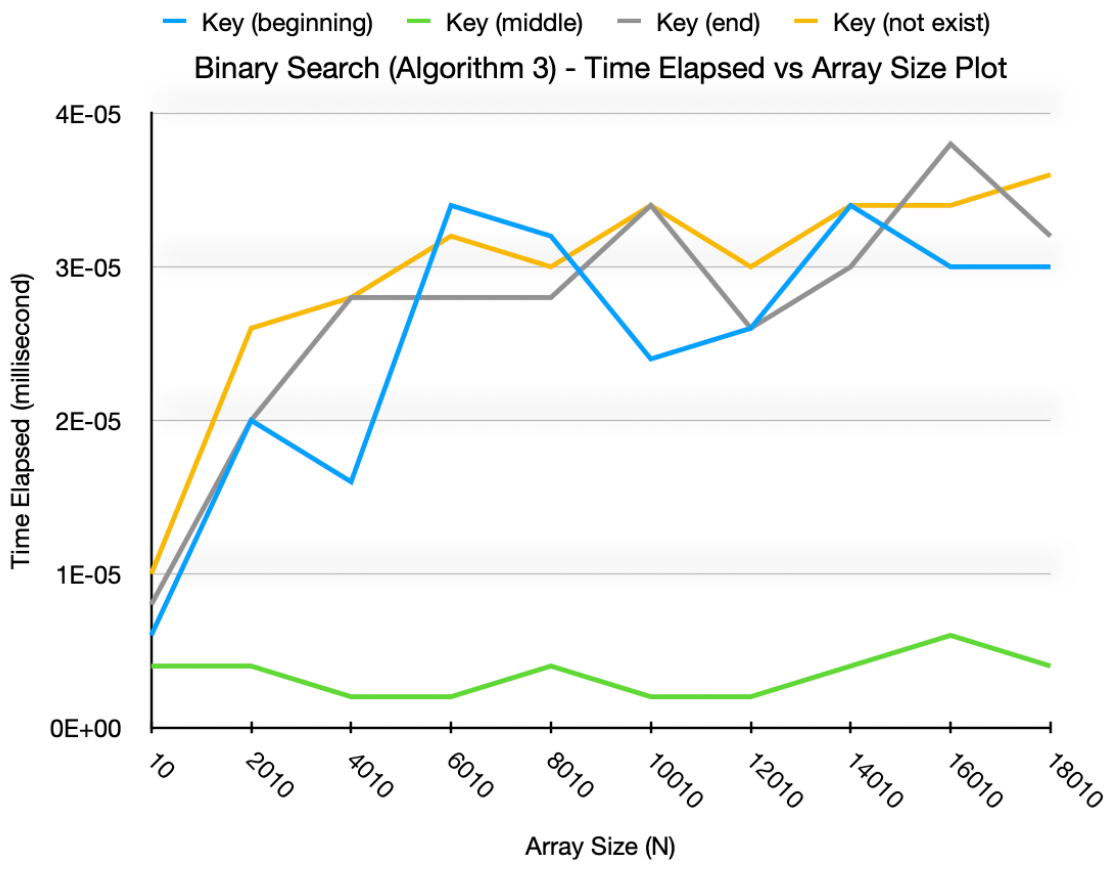
**Name Surname:** Işıl Ünveren
**Section:** 01
**ID:** 22202444

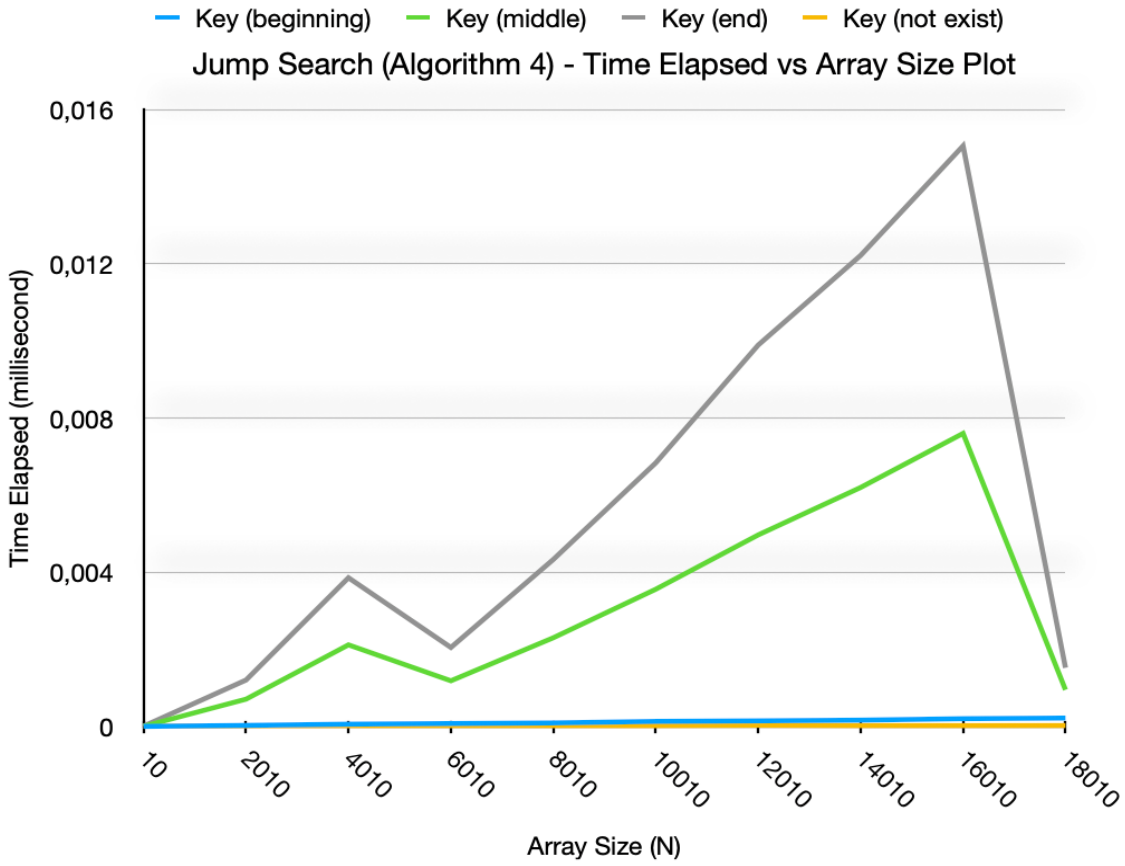
| TABLE OF RUNNING TIMES (ms) OF DIFFERENT SEARCH ALGORITHMS IN DIFFERENT SCENARIOS AND WITH DIFFERENT ARRAY SIZES | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **N** | **Linear (Iterative)** | | | | **Recursive Linear** | | | | **Binary** | | | | **Jump** | | | | **Random** | | | |
| | **A** | **B** | **C** | **D** | **A** | **B** | **C** | **D** | **A** | **B** | **C** | **D** | **A** | **B** | **C** | **D** | **A** | **B** | **C** | **D** |
| **10** | 0.000008 | 0.000008 | 0.002066 | 0.000016 | 0.000018 | 0.000012 | 0.000004 | 0.000026 | 0.000006 | 0.000002 | 0.000008 | 0.00001 | 0.00001 | 0.000012 | 0.000016 | 0.000012 | 0.000098 | 0.000088 | 0.000102 | 0.000116 |
| **2010** | 0.000018 | 0.001074 | 0.000016 | 0.002038 | 0.012938 | 0.00695 | 0.000034 | 0.01287 | 0.00002 | 0.000024 | 0.00002 | 0.000026 | 0.000036 | 0.000716 | 0.001212 | 0.000026 | 0.013258 | 0.016436 | 0.014542 | 0.01685 |
| **4010** | 0.000032 | 0.002064 | 0.003994 | 0.004042 | 0.025964 | 0.012516 | 0.000008 | 0.026542 | 0.000016 | 0.000024 | 0.000028 | 0.000028 | 0.000068 | 0.002124 | 0.003864 | 0.000028 | 0.023514 | 0.025416 | 0.024146 | 0.026556 |
| **6010** | 0.00005 | 0.003042 | 0.006008 | 0.006004 | 0.037252 | 0.019538 | 0.000094 | 0.037234 | 0.000034 | 0.000024 | 0.000028 | 0.000032 | 0.000084 | 0.001192 | 0.002056 | 0.000028 | 0.0397 | 0.03668 | 0.039834 | 0.040056 |
| **8010** | 0.000064 | 0.00406 | 0.00792 | 0.007962 | 0.051608 | 0.026474 | 0.000306 | 0.051846 | 0.000032 | 0.000024 | 0.000028 | 0.00003 | 0.0001 | 0.002308 | 0.004338 | 0.000028 | 0.046036 | 0.051906 | 0.051042 | 0.053758 |
| **10010** | 0.000074 | 0.005036 | 0.00992 | 0.010096 | 0.064876 | 0.032582 | 0.000368 | 0.063668 | 0.000024 | 0.000028 | 0.000034 | 0.000034 | 0.00014 | 0.003566 | 0.006842 | 0.000028 | 0.064772 | 0.064006 | 0.063586 | 0.069126 |
| **12010** | 0.000086 | 0.00612 | 0.011856 | 0.012176 | 0.079008 | 0.038886 | 0.000376 | 0.078094 | 0.000026 | 0.000026 | 0.000026 | 0.00003 | 0.000154 | 0.004974 | 0.009896 | 0.000038 | 0.07957 | 0.073092 | 0.073492 | 0.07971 |
| **14010** | 0.000098 | 0.007004 | 0.013842 | 0.013906 | 0.090874 | 0.045262 | 0.00042 | 0.090766 | 0.000034 | 0.000028 | 0.00003 | 0.000034 | 0.000172 | 0.006204 | 0.01222 | 0.00003 | 0.093008 | 0.082736 | 0.085246 | 0.093826 |
| **16010** | 0.000112 | 0.007982 | 0.01588 | 0.01593 | 0.103626 | 0.051874 | 0.000432 | 0.104578 | 0.00003 | 0.000028 | 0.000038 | 0.000034 | 0.00021 | 0.007606 | 0.01506 | 0.00003 | 0.09316 | 0.099372 | 0.099326 | 0.106148 |
| **18010** | 0.00012 | 0.00901 | 0.017954 | 0.018328 | 0.117396 | 0.059054 | 0.000482 | 0.118294 | 0.00003 | 0.000032 | 0.000032 | 0.000036 | 0.000226 | 0.000958 | 0.00153 | 0.000032 | 0.113014 | 0.106848 | 0.10191 | 0.117628 |

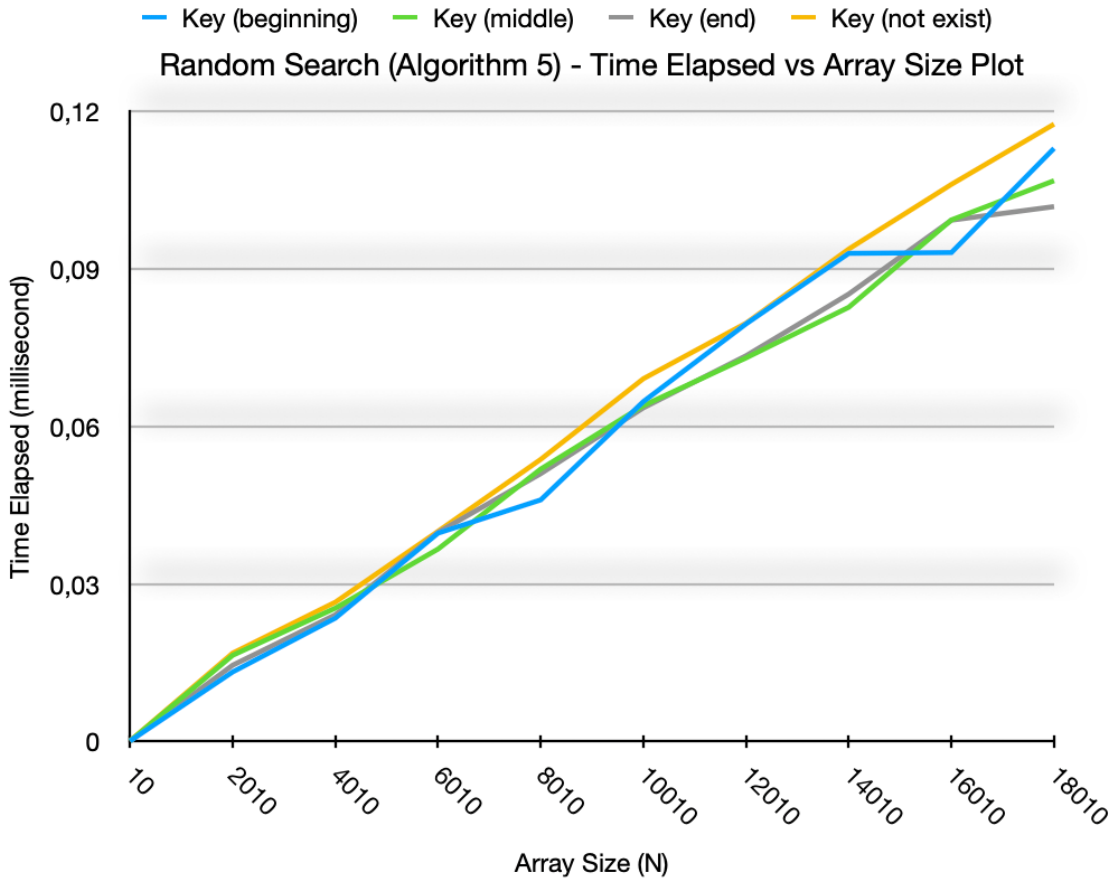
## Graphs of Observation




Key (beginning)
Key (middle)
Key (end)
Key (not exist)
Recursive Linear Search (Algorithm 2) - Time Elapsed vs Array Size Plot
Time Elapsed (millisecond)
0,12
0,09
0,06
0,03
0
10
2010
4010
6010
8010
10010
12010
14010
16010
18010
Array Size (N)


Key (beginning)
Key (middle)
Key (end)
Key (not exist)
Binary Search (Algorithm 3) - Time Elapsed vs Array Size Plot
Time Elapsed (millisecond)
4E-05
3E-05
2E-05
1E-05
0E+00
10
2010
4010
6010
8010
10010
12010
14010
16010
18010
Array Size (N)


Key (beginning)
Key (middle)
Key (end)
Key (not exist)
Jump Search (Algorithm 4) - Time Elapsed vs Array Size Plot
Time Elapsed (millisecond)
0,016
0,012
0,008
0,004
0
10
2010
4010
6010
8010
10010
12010
14010
16010
18010
Array Size (N)


Key (beginning)
Key (middle)
Key (end)
Key (not exist)
Random Search (Algorithm 5) - Time Elapsed vs Array Size Plot
Time Elapsed (millisecond)
0,12
0,09
0,06
0,03
0
10
2010
4010
6010
8010
10010
12010
14010
16010
18010
Array Size (N)

**Specifications of My Computer**

Processor: 8 CPU - 10 GPU
RAM: 16 GB
Operating System: macOS

**Theoretical Worst, Average, and Best Cases**

- Algorithm 1 (Linear Search)

**Best Case**: Key is found in the first iteration which means key is at the beginning of the array.
Corresponding running time: O(1)

**Worst Case**: Key is found in the last iteration or not found which means key is at the end of the array or key does not exist in the array.
Corresponding running time: O(N)

**Average Case**: Key is found in the middle of the expected number of iterations - as average, N/2 iterations-.
Corresponding running time: O(N)

- Algorithm 2 (Recursive Linear Search)

**Best Case**: Key is found in the first iteration which means key is at the end of the array.
Corresponding running time: O(1)

**Worst Case**: Key is found in the last iteration or not found.
Corresponding running time: O(N)

**Average Case**: Key is found in the middle of the expected number of iterations - as average, N/2 iterations-.
Corresponding running time: O(N)

- Algorithm 3 (Binary Search)

**Best Case**: Key is found in the first iteration which means key is in the middle.
Corresponding running time: O(1)

**Worst Case**: Key is found in the last iteration or not found which means key is at the end or beginning.
Corresponding running time: O(logN)

**Average Case**: Key is found in the middle of the expected number of iterations - as average (logN)/2 iterations -.
Corresponding running time: O(logN)

- Algorithm 4 (Jump Search) (assuming block size = N)

**Best Case**: Key is found in the first iteration which means key is at the beginning of the array or key does not exist in the array.
Corresponding running time: O(1)

**Worst Case**: Key is found in the last iteration or not found.
Corresponding running time: O(N)

**Average Case**: Key is found in the middle of the expected number of iterations - as average, N/2 iterations-.

Corresponding running time: O(N)

- Algorithm 5 (Random Search)

**Best Case**: Key is found in the first random index.
Corresponding running time: O(1)

**Worst Case**: Key is found in the last random index or not found.
Corresponding running time: O(N)

**Average Case**: Key is found in the middle of the expected number of iterations - as average, N/2 iterations-.
Corresponding running time: O(N)

**Observed Worst, Average, and Best Cases**

- Algorithm 1 (Linear Search)

**Best Case**: Key is at the beginning of the array which means key is found in first iteration. This result agrees with the theoretical information.

**Worst Case**: Key is at the end of the array which means key is found in last few iterations, or key does not exist. This result agrees with the theoretical information.

**Average Case**: Key is in the middle of the array which means key is found in half of the iterations of the worst case. This result agrees with the theoretical information.

- Algorithm 2 (Recursive Linear Search)

**Best Case**: Key is at the end of the array which means it is found in first few recursions. This result agrees with the theoretical information.

**Worst Case**: Key is at the beginning of the array which means it is found in the last few recursions, or key is not found. This result agrees with the theoretical information.

**Average Case**: Key is in the middle of the array which means key is found in half of the iterations of the worst case. This result agrees with the theoretical information.

- Algorithm 3 (Binary Search)

**Best Case**: Key is in the middle which means key is found in the first iteration. This result agrees with the theoretical information.

**Worst Case**: Key is at the beginning or at the end of each space reduction (half of previous part) which means key is at the beginning or at the end of the array, or key does not exist but there is not a clear distinction between them in observation which can be caused by limited number of running times or background processes of the computer which can lead instant changes in running time behavior of the computer. This result agrees with the theoretical information.

**Average Case**: I cannot observe an average case in the plots that I created because to be able to observe an average case I need a key element that is placed between head and middle of the array but I have only four cases which consist keys placed at the beginning, in the middle, at the end, and not placed in the array. Therefore, I cannot make a comparison between theoretical average case and observed case.

- Algorithm 4 (Jump Search) (assuming block size = N)

**Best Case**: Key is at the beginning of the array or key is out of bounds of the array which means in sorted array, key is larger than array's last element. In these conditions key is found in first few iterations. This result agrees with the theoretical information.

**Worst Case**: Key is at the end of the array which means it is found in last few iterations. This result agrees with the theoretical information.

**Average Case**: Key is in the middle of the array which means when searching according to assumption that array is sorted it is found almost in the iterations that is half of the worst case. This result agrees with the theoretical information.

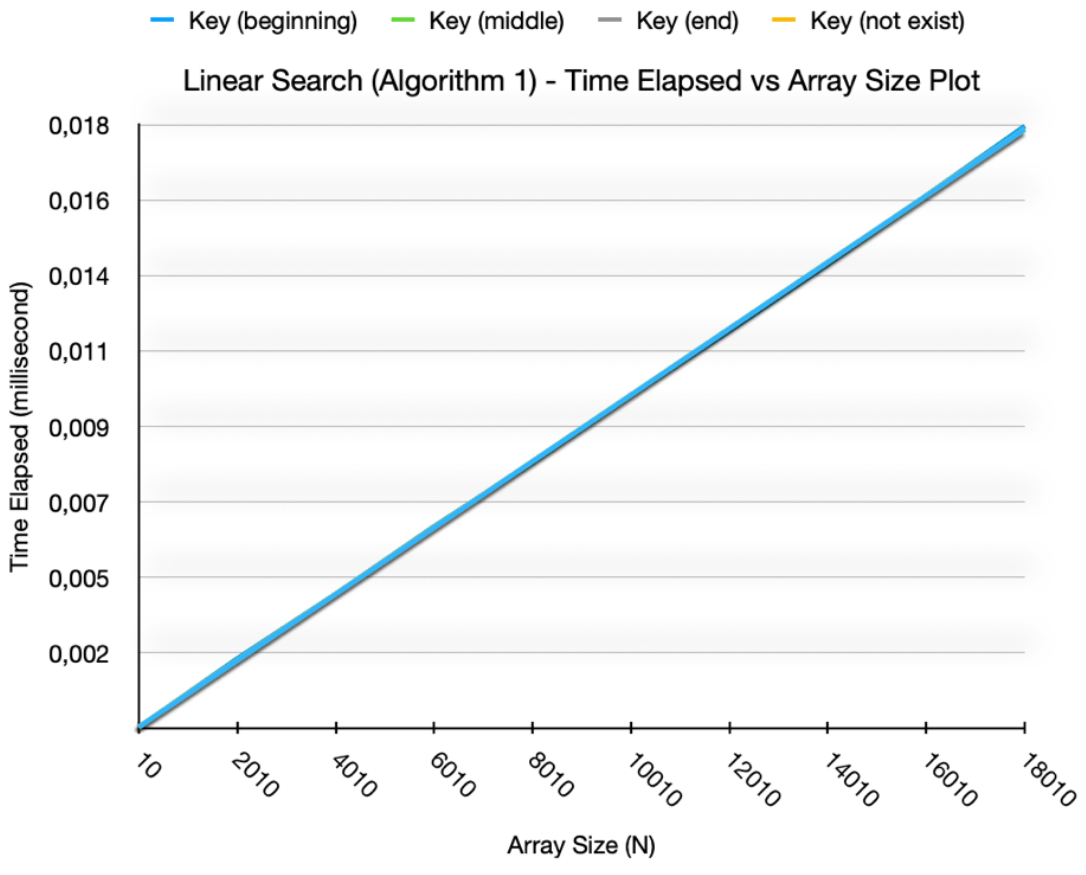
- Algorithm 5 (Random Search)

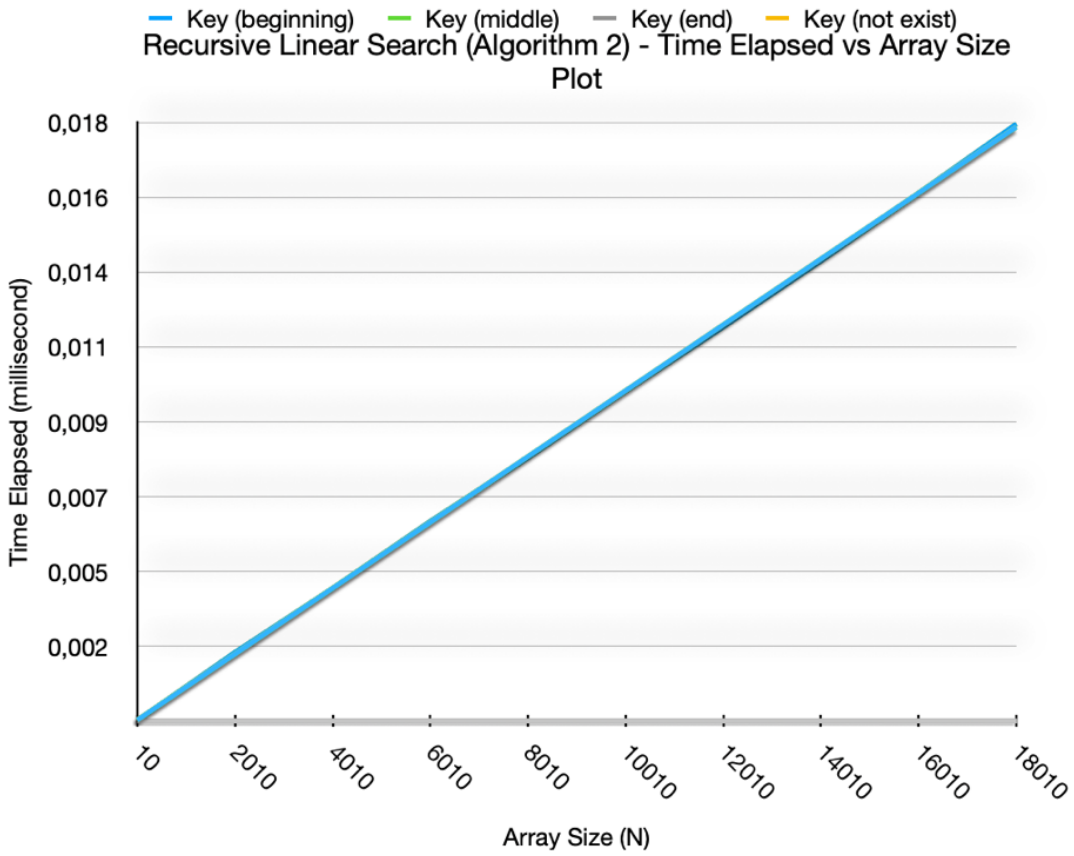
**Best Case**: Since this algorithm does not have certain rule for searching and it only searches by checking random indexes, it cannot be said something like the best case is the condition that key is at the beginning or in the middle of the array. The best case is the situation that first random index matches with the index of the key element by chance. In that condition key is found in one iteration which is the best case. This result agrees with the theoretical information.

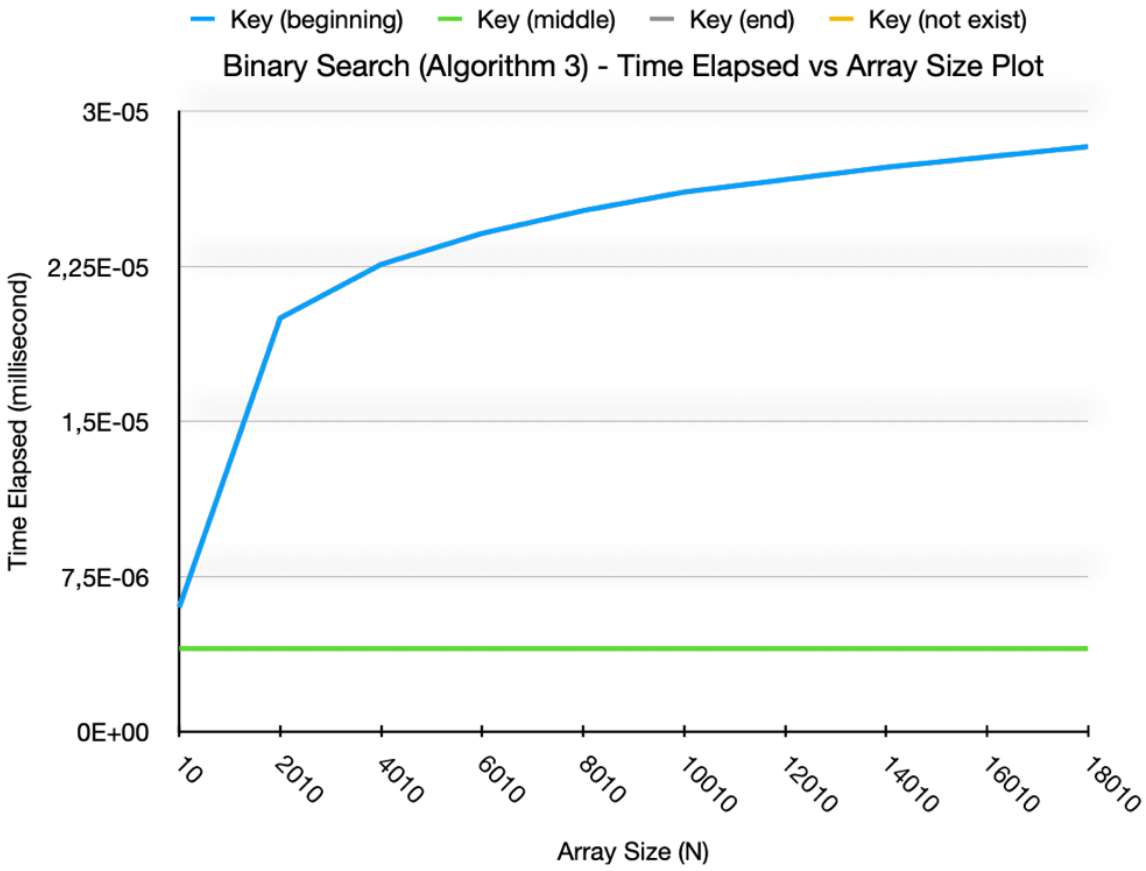
**Worst Case**: Key does not exist in the array. In that condition, all of the indices are tried, but search becomes unsuccessful. So, all iterations are done. This result agrees with the theoretical information.

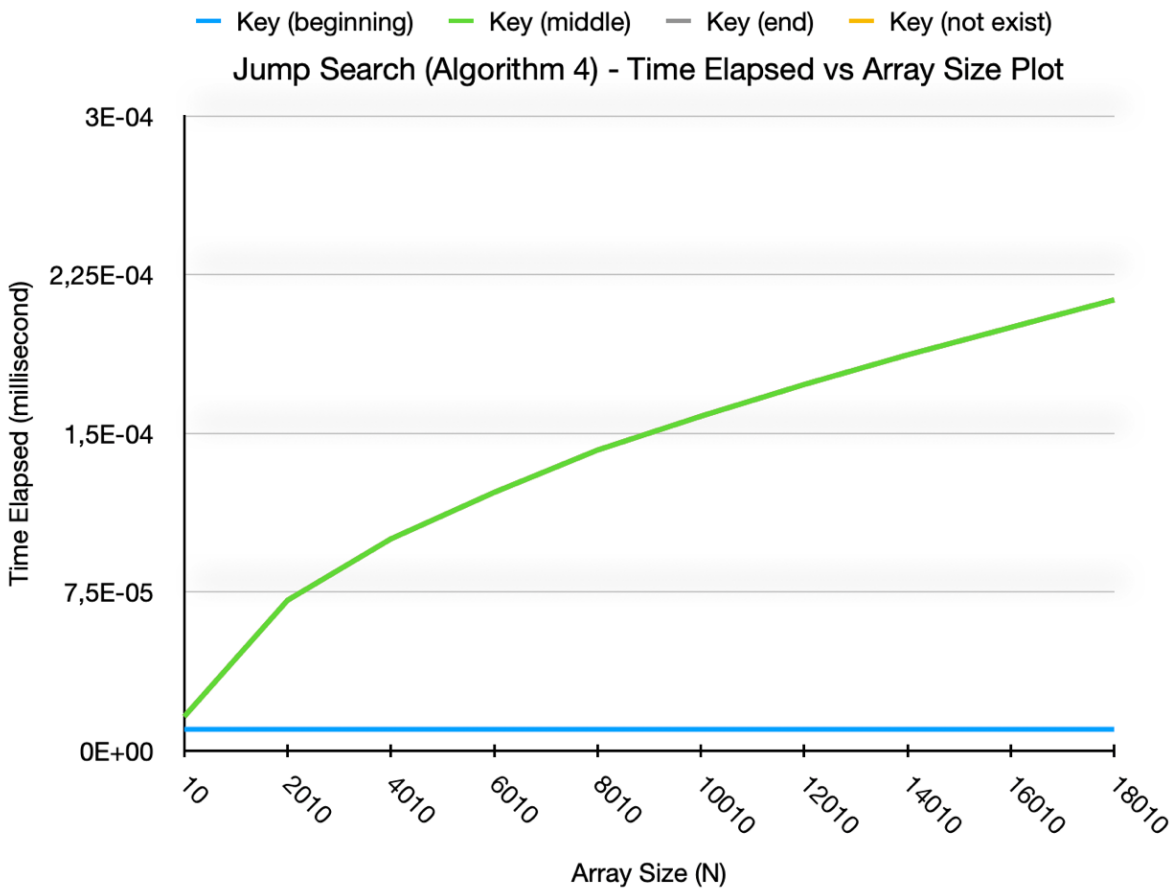
**Average Case**: Key is found in the middle of the expected number of iterations which means in the half of the way random index matches the key's index. This result agrees with the theoretical information.

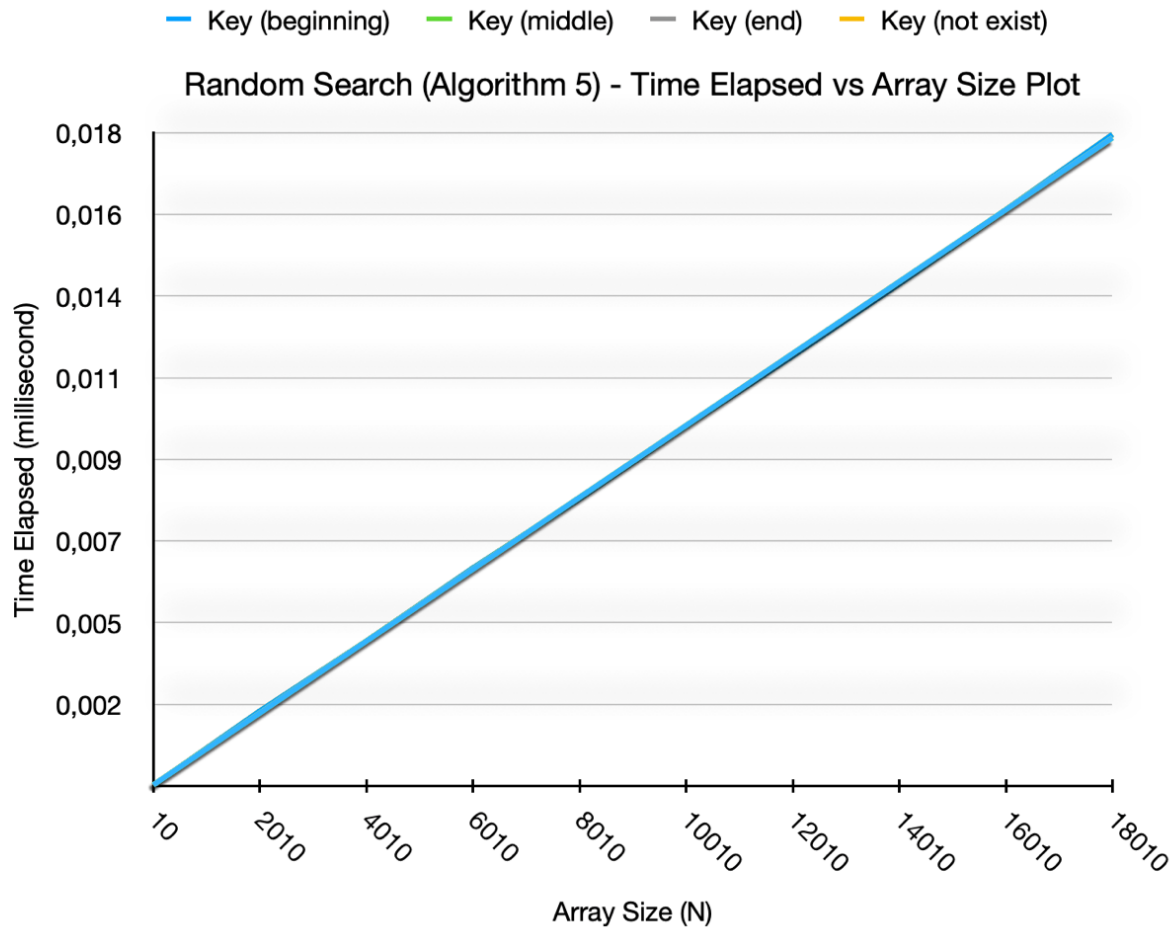
## Graphs of Theoretical Information




Key (beginning)
Key (middle)
Key (end)
Key (not exist)
Recursive Linear Search (Algorithm 2) - Time Elapsed vs Array Size Plot
Time Elapsed (millisecond)
0,018
0,016
0,014
0,011
0,009
0,007
0,005
0,002
10
2010
4010
6010
8010
10010
12010
14010
16010
18010
Array Size (N)


Key (beginning)
Key (middle)
Key (end)
Key (not exist)
Binary Search (Algorithm 3) - Time Elapsed vs Array Size Plot
Time Elapsed (millisecond)
3E-05
2,25E-05
1,5E-05
7,5E-06
0E+00
10
2010
4010
6010
8010
10010
12010
14010
16010
18010
Array Size (N)


Key (beginning)
Key (middle)
Key (end)
Key (not exist)
Jump Search (Algorithm 4) - Time Elapsed vs Array Size Plot
3E-04
2,25E-04
1,5E-04
7,5E-05
0E+00
Time Elapsed (millisecond)
10
2010
4010
6010
8010
10010
12010
14010
16010
18010
Array Size (N)


Key (beginning)
Key (middle)
Key (end)
Key (not exist)
Random Search (Algorithm 5) - Time Elapsed vs Array Size Plot
0,018
0,016
0,014
0,011
0,009
0,007
0,005
0,002
Time Elapsed (millisecond)
10
2010
4010
6010
8010
10010
12010
14010
16010
18010
Array Size (N)

**Comparison of Theoretical Information and Observed Results**

## Algorithm 1 – Linear Search

Observed results of linear search algorithm consistent with the theoretical information which is time complexity of linear search for the best case is O(1), for the worst case is O(N), for the average case is O(N). Since we ignore constant numbers in theoretical values - for the average case observed result is O(N/2) - it can be said that observation of linear search almost totally matches with the theoretical data.

## Algorithm 2 – Recursive Linear Search

Observed results of recursive linear search algorithm consistent with the theoretical information which is time complexity of recursive linear search for the best case is O(1), for the worst case is O(N), for the average case is O(N). Since we ignore constant numbers in theoretical values - for the average case observed result is O(N/2) - it can be said that observation of recursive linear search almost matches with the theoretical data except minor deviations which can be caused by many reasons like changes in runtime behavior of the computer or background processes which can impact speed of the system.

## Algorithm 3 – Binary Search

Observed results of binary search algorithm have consistencies and inconsistencies with the theoretical information which is time complexity of binary search for the best case is O(1), for the worst and average cases is O(logN) -actual value of the time complexity of the average case is $O(\log_2(N))$ bu constants are ignored-. Observation results for the best case almost matches with the theoretical result except minor deviations which can be caused by many reasons like the index of the key which is assumed to be close to the beginning but is not exactly in the index zero, changes in runtime behavior of the computer, background processes which can impact speed of the system or inadequate running time for different scenarios. However, in observed results average case and the worst cannot be distinguished clearly. There can be many reasons of that. For instance, it may happen because of the other applications and software running in the background, limited repetitions of observation, or compiler optimization. However, when the deviations of the observed graph are ignored, when looking at the general appearance of the graph, it can be said that it has a similar curve -which is the logN curve- to the theoretical graph.

## Algorithm 4 – Jump Search

Since jump size is taken as $\sqrt{N}$ in the observation, its theoretical time complexity is $O(\sqrt{N})$, if we assume jump size is equal to array size theoretical time complexity is O(N) but in the sake of more efficient algorithm jump size is taken as $\sqrt{N}$, for the average and the worst cases, and for the best case it is O(1) since it is expected to find the key in first iteration when key is at the beginning of the array or key does not exist in the array. This condition, which is the best case, is consistent with the observed results and theoretical information. However, same thing could not be observed for the average and the worst cases because graph of the observation and theory do not match. This

can be because of the linear search in algorithm used after deciding the block that may possibly have target element. In that part if key is close to the beginning of the block, time consumed will be relatively shorter than the condition that key is close to the end of the block. Additionally this may be caused because of the background effects which are running aplications at that time, compiler optimization, or inadequate number of trials.

**Algorithm 5 – Random Search**

Observed results of random search algorithm consistent with the theoretical information which is time complexity of random search for the best case is O(1), for the worst case is O(N), for the average case is O(N). The best case can be observed only in a condition that first random index matches with the index of key in the array but since there are limited repetitions for the observation such a condition could not be encountered. Therefore, in all scenarios, which are key is at the beginning, in the middle, at the end, and key does not exist, random search algorithm has similar time consumption both among four different scenories in the observation and between observed results and theoretical information. There are minor deviations in observed results which may be caused because of chance of the random index, the background effects which are running aplications at that time or compiler optimization.