CS201 - Section 3
Homework 2

Aral Müftüoğlu
22201566

Specifications of the computer (Macbook Air):

Processor:

Chip: Apple M1

CPU: 8-core

Architecture: ARM-based, built on a 5nm process.

Clock Speed: Not officially specified but operates dynamically for efficiency.

Ram:

Capacity: 16GB

Type: LPDDR4X

Operating System:

Comes pre-installed with macOS

Fully supports macOS Monterey, macOS Ventura, and later versions

Linear Search (Iterative)

For the iterative linear search algorithm worst case is for the key which is not present or is the last item. For that case time complexity is O(n). Average case is for the key that is at a random position in the array. For that case time complexity is also O(n). Lastly, for the best case key should be the first element in the array. In that case, time complexity is O(1). However, in the observation there is no key which is in the first index instead of that, there is a key close the beginning at the index "size/6". So, best case is that key and time complexity for this key is O(n).

For the observed results,

- a (key close the beginning) is the element of array which is at the index "size/6".
- b (key is around the middle) is the is the element of array which is at the index "size/2".
- c (key is close the end) is the element of the array which is at the index "size-size/6".
- d (key does not exist) is the key equal to "size*10+1" because array consist of elements which have max value "size*10".

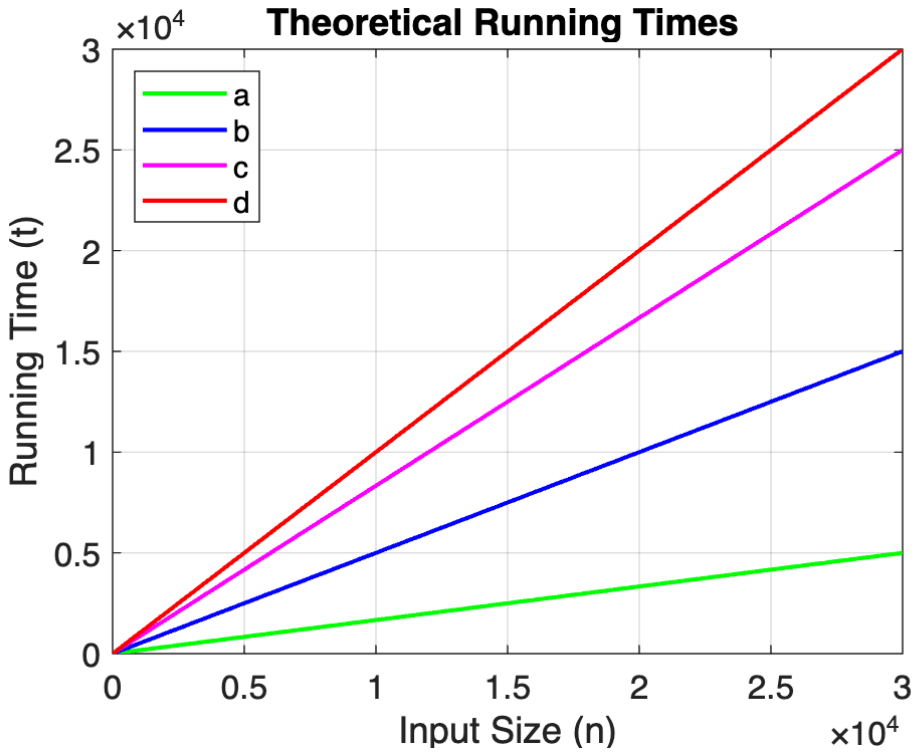
In these scenarios corresponding running times are

For scenario a, (size/6).t

For scenario b, (size/2).t

For scenario c, (size-size/6).t

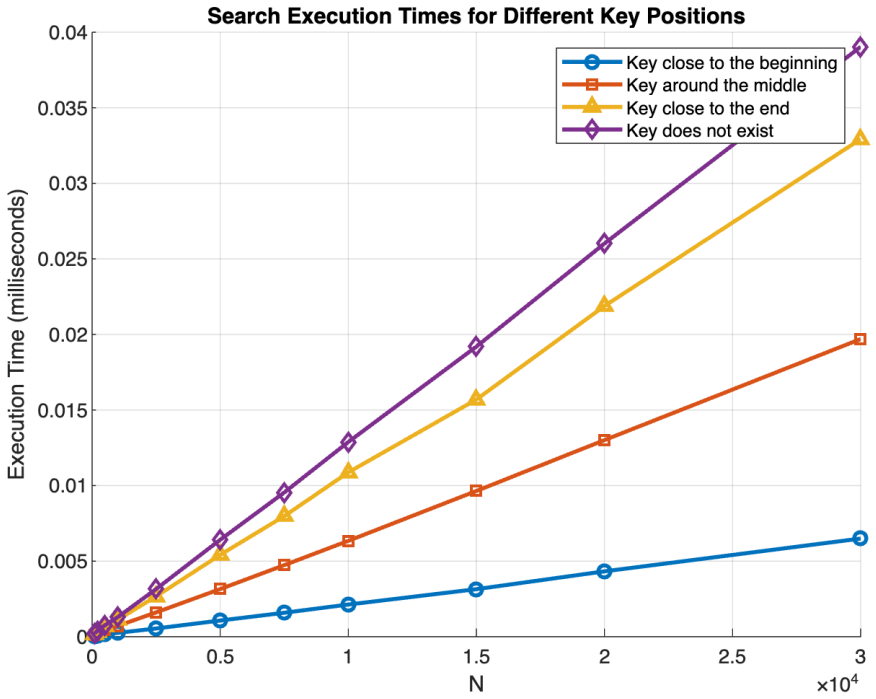
For scenario d, size.t



Based on the plot the best case in algorithm is the scenario “a” because it needs relatively small number of comparisons among the other scenarios. After that, scenario “b” represents the average case because in the scenario nearly half of the array is being searched. In the algorithm, scenario “d” is the worst case because in this one it is required that all of the elements to be checked. Although scenario “d” is the worst case, scenario “c” is also nearly worst case because it needs big part of the array to be checked (until index “size-size/6”). However, d needs more checking so it is the worst case. So, in the algorithm:

a, is the best case
b, is the average case
d, is the worst case

N	Linear (Iterative)			
	a	b	c	d
100	0.0000385	0.0001204	0.0001893	0.0002215
200	0.0000843	0.0002216	0.0003277	0.000379
500	0.0001668	0.0004356	0.0006839	0.0007601
1000	0.0002541	0.0007124	0.0011215	0.0012941
2500	0.0005447	0.0016091	0.0026716	0.0031566
5000	0.0010703	0.0031505	0.0054172	0.0064142
7500	0.0015845	0.0047357	0.0079793	0.0095434
10000	0.0021283	0.0063435	0.0108735	0.012883
15000	0.0031449	0.0096436	0.0156947	0.0191968
20000	0.0043244	0.0130101	0.0218839	0.0260179
30000	0.0064937	0.0196969	0.0329274	0.0390374



a (key close to beginning) is the best case
b (key around the middle) is the average case
d (key does not exist) is the worst case

For the execution speed of algorithm:

Scenario a > Scenario b > Scenario c > Scenario d

Theoretical results and observed results agree. Scenario “a” is the fastest as searching starts from beginning and key in the scenario is close the beginning. However there is a important point for the best cases in both theoretical results and observed results. It is assumed that we chose scenario “a” as a key close the beginning not at at the beginning and the key is the element in the array at the index size/6. So, while it can be O(1) for the best case in the condition that element near beginning is the first element of the array, but observed result is O(n) because it is not the first element it is the element near the beginning.

Plots of theoretical result and observed result agree. These plots generally consistent. Only in a few values they seems a little bit different. That can stem from the effect of the current operations taking place in the background on the processor while the computer is running code.

Linear Search (Recursive)

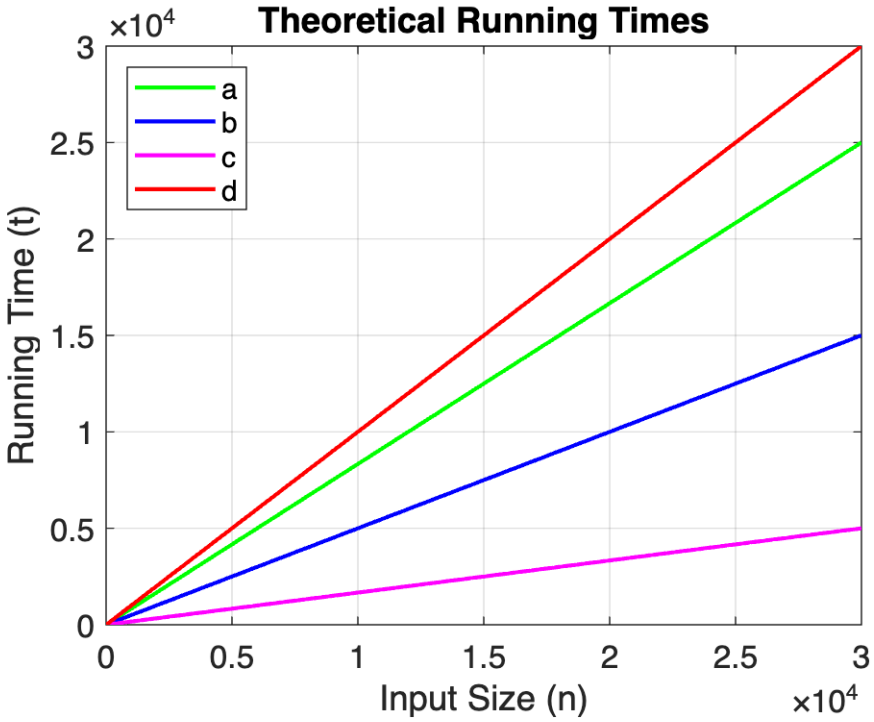
For the recursive linear search algorithm average case is the condition that algorithm search half of the array and it has time complexity O(n). Worst case has also time complexity O(n) because it searches whole array to find non-present value. Lastly, best case occurs when algorithm find the key first index it search and it has time complexity O(1). However, in the observation there is no key for first index of array instead of that, there is a key close the beginning for the best case. So, in the best case time complexity is O(n).

For the observed results,

- a (key close the beginning) is the element of array which is at the index “size/6”.
- b (key is around the middle) is the is the element of array which is at the index “size/2”.
- c (key is close the end) is the element of the array which is at the index “size-size/6”.
- d (key does not exist) is the key equal to “size*10+1” because array consist of elements which have max value “size*10”.

In these scenarios corresponding running times are

For scenario a, (size−size/6)·t
For scenario b, (size-size/2).t
For scenario c, (size/6).t
For scenario d, size.t



Based on the graph scenario “c” requires least time because algorithm starts to search from last index and in the scenario key is close the end so, it needs least comparisons among the other scenarios. It is the best case. Moreover, for scenario “d” algorithm needs to check all elements in the array so it is the worst case. Although, scenario “a” needs to check values until the element at index size/6 (search starts from last index), amount of comparisons in the scenario can not exceed scenario “d”. Lastly, in scenario “b” it checks half of the array so it is the average case in this algorithm.

c is the best case
b is the average case
d is the worst case

N	Linear (Recursive)			
	a	b	c	d
100	0.0004601	0.0003766	3.39e-05	0.000745
200	0.0011415	0.0007239	6.71e-05	0.0014508
500	0.0028999	0.001794	0.0004418	0.0035863
1000	0.0057901	0.0037218	0.0011388	0.0072768
2500	0.0147895	0.008958	0.0028955	0.0185248
5000	0.0312248	0.0178886	0.005831	0.0370781
7500	0.0466366	0.0277575	0.0089427	0.0561661
10000	0.0624701	0.0371282	0.0118085	0.0754487
15000	0.0945126	0.056305	0.0179836	0.113857
20000	0.127065	0.0780352	0.0256601	0.155583
30000	0.20161	0.11394	0.037092	0.230858

c (key close to end) is the best case
b (key around the middle) is the average case
d (key does not exist) is the worst case

For the execution speed of algorithm:

Scenario c > Scenario b > Scenario a > Scenario d

Theoretical results and observed results agree. However there is a important point for the best cases in both theoretical results and observed results. It is assumed that we chose scenario “c” as a key close the end not at at the end and the key is the element in the array at the index size-size/6. So, while it can be O(1) for the best case in the condition that element near end is the last element of the array, but observed result is O(n) because it is not the last element it is the element near the end.

The theoretical result and the observed result plots match. These plots are generally consistent. They only look slightly different in a few values. This may be due to the current load on the computer components (RAM, CPU...) due to applications running in the background or the processes the operating system runs in the background.

Binary Search

For a binary search algorithm worst case is the case that key is does not exist which is have time complexity O(logn). Also in the binary search, average case is represented by keys randomly located in the array that has time complexity O(log n) and lastly for the best case key should be the value of the element in the array which is index is (size/2) since searching algorithm firstly checks this index.

For the observed results,

- a (key close the beginning) is the element of array which is at the index “size/6”.
- b (key is around the middle) is the is the element of array which is at the index “size/2”.
- c (key is close the end) is the element of the array which is at the index “size-size/6”.
- d (key does not exist) is the key equal to “size*10+1” because array consist of elements which have max value “size*10”.

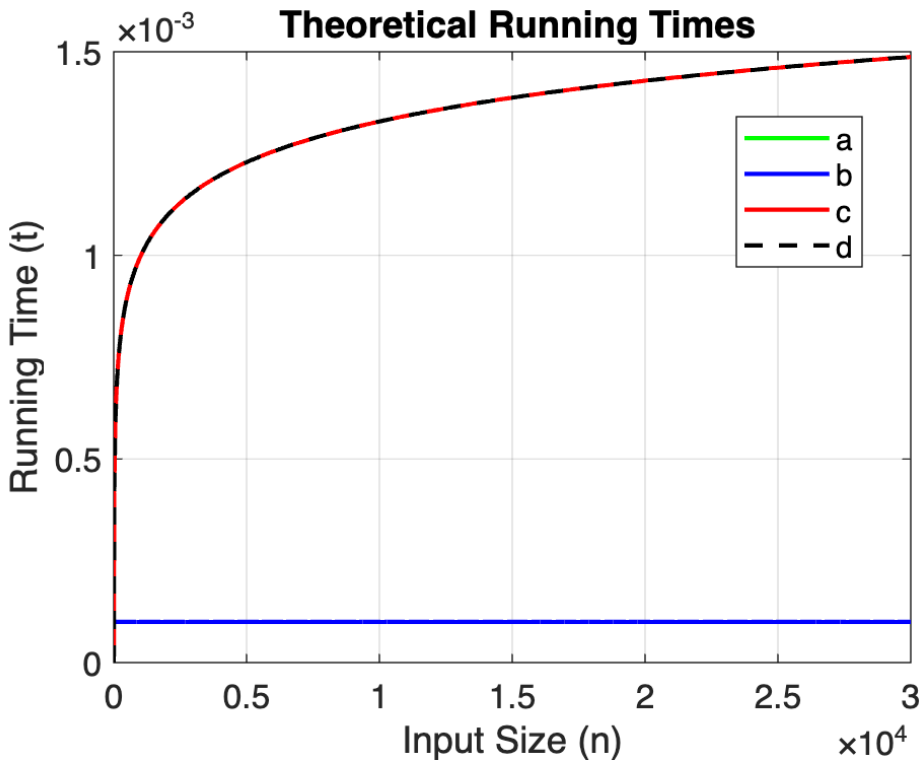
In these scenarios corresponding running times are

For scenario a, $\log_2(size).t$

For scenario b, 1.t

For scenario c, $\log_2(size).t$

For scenario d, $\log_2(size).t$

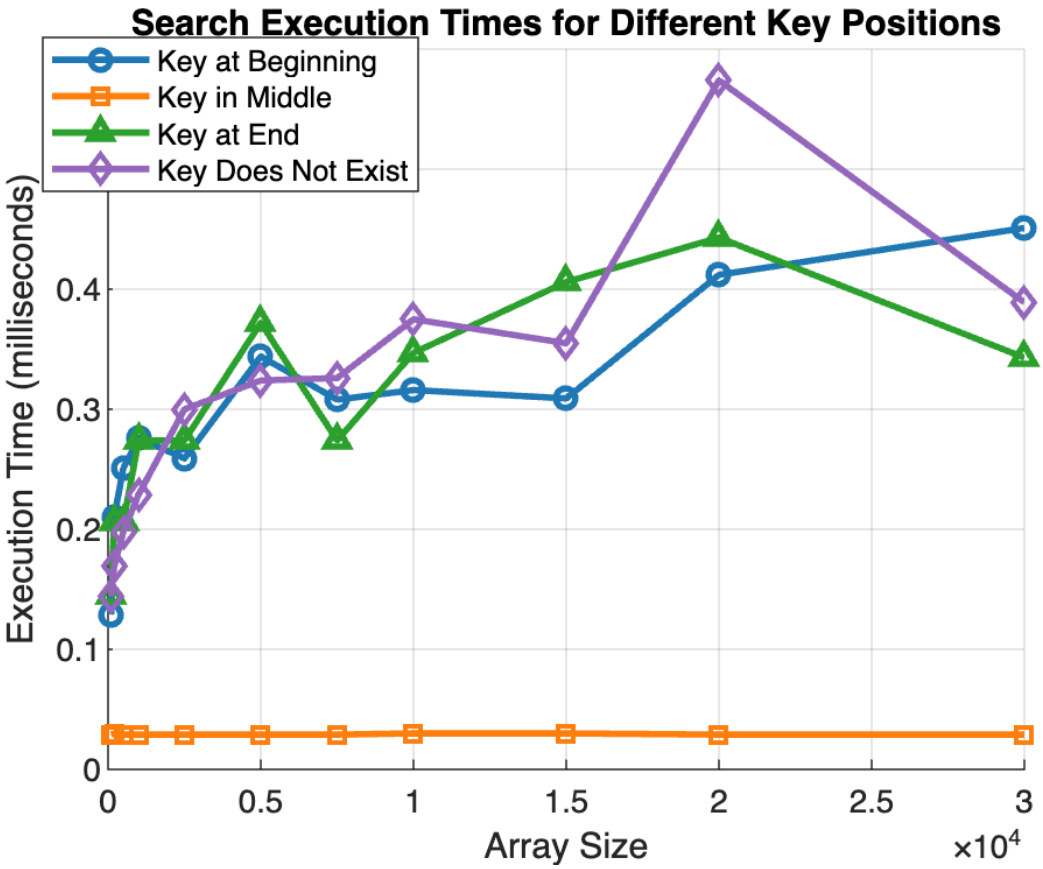


When theoretical graph is examined the best case is scenario "b" which has key at "size/2". For other cases all of them almost has same running time so all of them can be average case or worst case together. In notation, scenario "a" and "c" are average case together and scenario "d" is the worst case.

b is the best case
a, c are average case together
d is the worst case

N	Binary			
	a	b	c	d
100	0.0000129	0.0000029	0.0000145	0.0000144
200	0.000021	0.000003	0.0000206	0.0000169
500	0.0000251	0.0000029	0.0000206	0.0000198
1000	0.0000276	0.0000029	0.0000274	0.0000229

2500	0.0000259	0.0000029	0.0000274	0.0000300
5000	0.0000344	0.0000029	0.0000372	0.0000324
7500	0.0000308	0.0000029	0.0000274	0.0000326
10000	0.0000316	0.000003	0.0000347	0.0000375
15000	0.0000309	0.000003	0.0000406	0.0000355
20000	0.0000412	0.0000029	0.0000443	0.0000574
30000	0.0000451	0.0000029	0.0000343	0.0000389



b (key is around the middle) is the best case
a (key is close the beginning), c (key is close the end) are average case together
d (key does not exist) is the worst case

For the execution speed of algorithm:

Scenario b > Scenario a ≈ Scenario c > Scenario d

Theoretical results and observed results agree in general but there is some issues in scenarios other than scenario “b”. Scenario “b” is the fastest as in the theoretical one because it is in the middle of the array. While scenario “b” is similar with theoretical one, others also have general shape of function “logn”. It shows that time complexities of these scenarios O(long) is clear in the observed results. However, disruptions in the shape of logn function for all of the scenario “c”, “a” and “d” can stem from too small execution times of the searches. Since, execution times of the algorithm is to small it can leads to different lines or graphs than theoretical ones because being too small include danger of miscalculation or understatement in representation in output, which can stem from computer.

Plots of theoretical result and observed result generally agree but reasons that are mentioned above their shape are not exactly same as expected. Moreover, background activities in the computer can affect the instant breaking points in the observed result graph.

Jump Search

For the jump search algorithm worst case is for the key which is not present or is the last item. For that case time complexity is $O(\sqrt{n})$. Average case is for the key that is at a random position in the array. For that case time complexity is also $O(\sqrt{n})$. Lastly, for the best case key should be the first element in the array. In that case, time complexity is O(1). However, in the observation there is no key which is in the first index instead of that, there is a key close the beginning at the index “size/6”. So, best case is that key and time complexity for this key is $O(\sqrt{n})$.

For the observed results,

- a (key close the beginning) is the element of array which is at the index “size/6”.
- b (key is around the middle) is the is the element of array which is at the index “size/2”.
- c (key is close the end) is the element of the array which is at the index “size-size/6”.
- d (key does not exist) is the key equal to “size*10+1” because array consist of elements which have max value “size*10”.

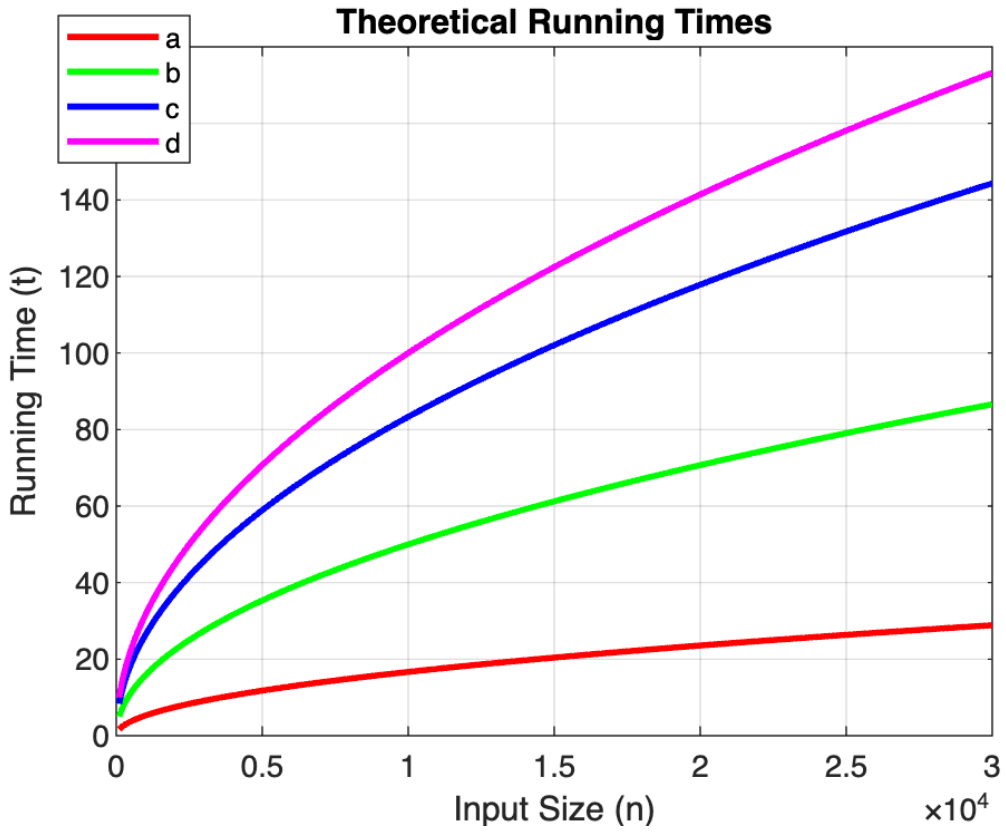
In these scenarios corresponding running times are

For scenario a, $(\sqrt{size})$ / 6 .t

For scenario b, $(\sqrt{size}$ / 2).t

For scenario c, $(\sqrt{size}$ - $\sqrt{size}$ / 6).t

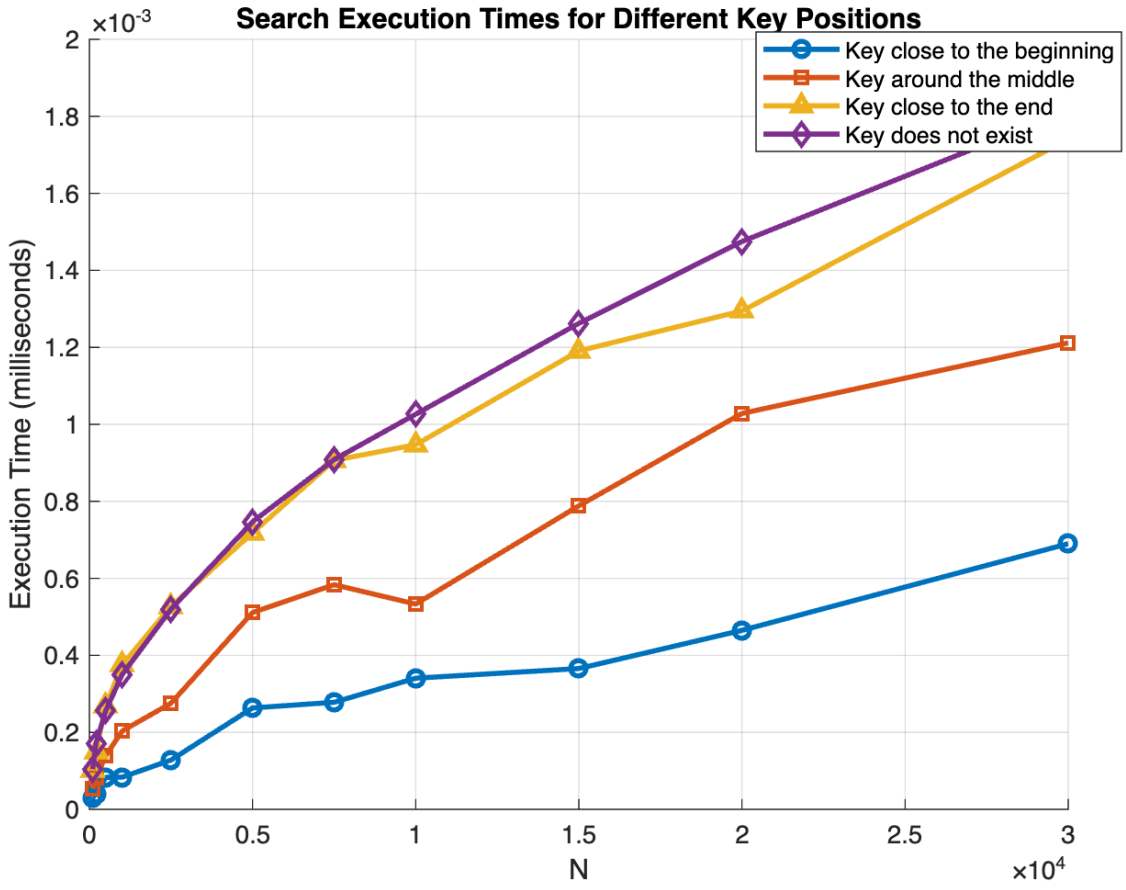
For scenario d, $(\sqrt{size})$.t



For the theoretical running times best case its the scenario “a” is the best case because in the scenario minimum amount of comparison is required when it is compared with other scenarios. Additionally, scenario “d” and “c” have a close theoretic running times but since scenario “d” is a bit more slow that is why it is worst case for sum search algorithm. Finally, when the scenario “b” is examined it is clear that it is the average case for the algorithm.

a is the best case
b is the average case
d is the worst case

N	Jump Search			
	a	b	c	d
100	0.0000309	0.0000532	0.0001006	0.0001045
200	0.0000387	0.0000818	0.0001502	0.0001718
500	0.0000832	0.000139	0.0002688	0.0002561
1000	0.0000834	0.0002033	0.000375	0.0003497
2500	0.0001281	0.0002756	0.0005274	0.0005198
5000	0.0002634	0.0005112	0.0007183	0.0007468
7500	0.0002781	0.0005835	0.0009065	0.0009085
10000	0.0003407	0.0005332	0.000947	0.0010263
15000	0.0003657	0.0007884	0.0011908	0.0012623
20000	0.000465	0.0010283	0.0012945	0.0014748
30000	0.0006899	0.0012117	0.0017406	0.0018142



a (key is close the beginning) is the best case
b (key is around the middle) is the average case
d (key does not exist) is the worst case

For the execution speed of algorithm:

Scenario a > Scenario b > Scenario c > Scenario d

Theoretical results and observed results are agreeing with each other. For the scenario “d” which has a key does not exist in the array both of them have biggest running times so the worst case. Also, for the scenario that key close the beginning again both of them have best case. However for both of the results, since in scenario “a” key is regarded as the element of the array at the index “size/6” time complexity of the best case can not be O(1) it is $O(\sqrt{n})$. If the key is just fixed a number in the array like first element array that time complexity could be possible. Moreover, for the scenarios “c” and “d”, because key at the scenario “c” selected as a element of the array at the index (size-size/6) it is not very close the scenario “d” which is the worst case. It requires less comparison especially for relatively big arrays.

Plots of theoretical result and observed result agree in general structure. However, shape of function $y = \sqrt{x}$ (since time complexities of the cases $O(\sqrt{n})$ not exactly same with observed results plot even it is same with theoretical plot. This situation can stem from running times is too small to get precise results which is a result of jump search being very fast. Also, reasons of breaking points in the graph might be background activities of computer in while it was running the code and be the operating system .

Random Search

For the iterative linear search algorithm worst case is for the key which is not present or is the last item. For that case time complexity is O(n). Average case is for the key that is at a random position in the array. For that case time complexity is also O(n). Lastly, for the best case key should be the first element in the array. In that case, time complexity is O(1). However, in the observation there is no key which is in the first index instead of that, there is a key close the beginning at the index “size/6”. So, best case is that key and time complexity for this key is O(n).

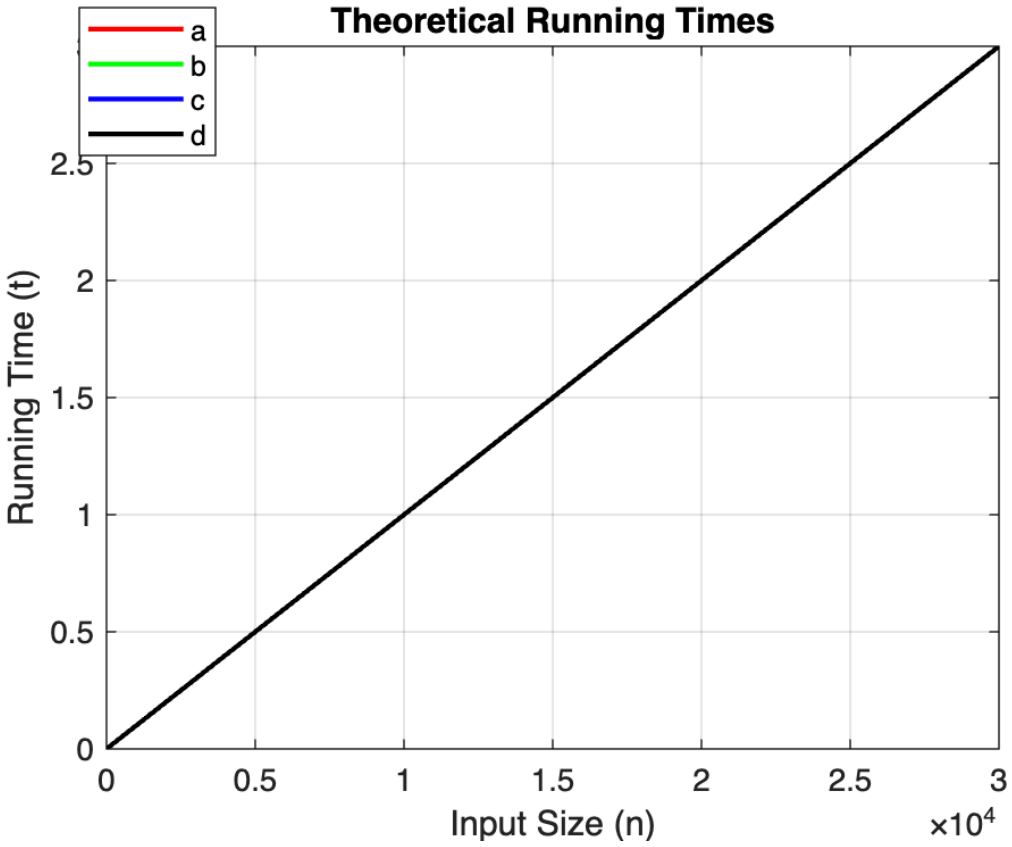
For the observed results,

- a (key close the beginning) is the element of array which is at the index “size/6”.
- b (key is around the middle) is the is the element of array which is at the index “size/2”.

- c (key is close the end) is the element of the array which is at the index “size-size/6”.
- d (key does not exist) is the key equal to “size*10+1” because array consist of elements which have max value “size*10”.

In these scenarios corresponding running times are

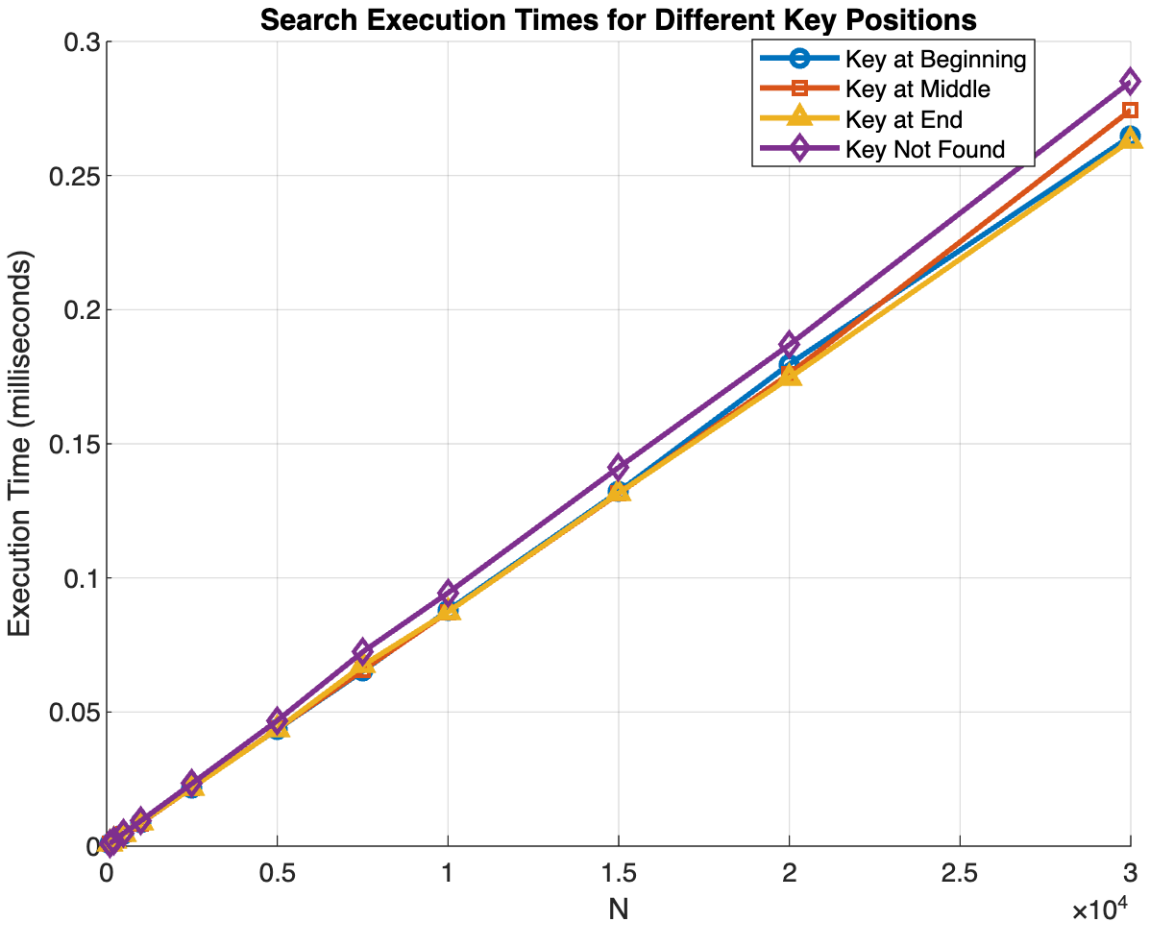
For scenario a, size.t
For scenario b, size.t
For scenario c, size.t
For scenario d, size.t



For the theoretical running times it seems like all of the scenarios have the same time complexity O(n) for the algorithm but there is a important point for scenario “d”, even it is not clear in the plot it notated to be worst case because in that scenario it sneed to be passed all elements of array. However, for the best case even if it is a very rare possibility because it needs algorithm to first generated random value is equal to key. While array is growing, this possibility is being less possible. If this situation can happen best case for algorithm is going to be O(1).

a, b, and c are average cases together
d is the worst case

N	Random			
	a	b	c	d
100	0.0008969	0.0008998	0.0009044	0.0009971
200	0.0018373	0.0018192	0.0018325	0.0019507
500	0.0044339	0.0043815	0.004386	0.0048084
1000	0.0088309	0.0085599	0.0087597	0.0094078
2500	0.0218045	0.0217741	0.0217723	0.0233414
5000	0.0435945	0.0436881	0.0436399	0.0468198
7500	0.0651947	0.0655442	0.0673523	0.072386
10000	0.0878836	0.0873208	0.0872939	0.0944046
15000	0.132158	0.131449	0.131659	0.141206
20000	0.179705	0.176082	0.174596	0.186935
30000	0.264652	0.274432	0.263112	0.285025



a (key is close the beginning), b (key is around the middle) and c (key is close the end) are the average case
d (key does not exist) is the worst case

For the execution speed of algorithm:

$$\text{Scenario a} \approx \text{Scenario b} \approx \text{Scenario c} > \text{Scenario d}$$

Theoretical and observed results agree. In the both of the results, all scenarios have same time complexity O(n). However, there is no best case O(1) since the rare possibly of the case. Moreover, scenario “d” is the worst case as mentioned in theoretical results. The scenario have a bit more running times than other three scenarios because it require more comparison.

Plots of theoretical and observed results almost same. There is difference for scenario “d”. In this scenario, even it is not clear in theoretical running times graph in observed results it’s running time is more than other especially y for bigger arrays. Moreover, observed results are not reaching function $y = x$ as it is expected from theoretical plot. This issue can stem from the fact that computer has different activities in background while running the code so that might affect the running times of the algorithm or that could be just stem from algorithm reaches the key before expected. (it is random search so that situation might be happened)