Alp Eren Köken
22202876
CS 201, Section: 3
HOMEWORK 2

TABLES:

Linear Search Algorithm - Elapsed Times (ms)

N	Scenario 1 (ms)	Scenario 2 (ms)	Scenario 3 (ms)	Scenario 4 (ms)
10	0.000005	0.00000857	0.0000115	0.00001459
30	0.00000857	0.00003113	0.00004956	0.00003361
60	0.00001484	0.00004077	0.00005846	0.00005662
100	0.00002787	0.00005697	0.00009423	0.00008929
300	0.00006344	0.00013772	0.00024271	0.00024596
600	0.00009284	0.00025053	0.00047986	0.00048160
1000	0.00015621	0.00041314	0.00083722	0.00079467
3000	0.00041644	0.00120236	0.00241521	0.00240836
6000	0.00080504	0.00235679	0.00470757	0.00470322
10000	0.00130953	0.00401949	0.00784509	0.00783577
30000	0.00391282	0.0120472	0.024037	0.0239559

Recursive Linear Search Algorithm - Elapsed Times (ms)

N	Scenario 1 (ms)	Scenario 2 (ms)	Scenario 3 (ms)	Scenario 4 (ms)
10	0.00000539	0.00000977	0.00001325	0.00001693
30	0.00000972	0.00002721	0.00007307	0.00007638
60	0.00001755	0.00007349	0.00015491	0.00016697
100	0.00004166	0.00013768	0.00027997	0.00030126
300	0.00014357	0.00044022	0.00083738	0.00084177
600	0.00028836	0.00089591	0.00164961	0.00171235
1000	0.00050203	0.00142354	0.00284831	0.00275650
3000	0.001381	0.00477746	0.00985498	0.00997535
6000	0.00293129	0.00987721	0.0198585	0.00997535
10000	0.00532846	0.0165419	0.0343467	0.0343163
30000	0.0166403	0.0525302	0.11199	0.116069

Binary Search Algorithm

N	Scenario 1 (ms)	Scenario 2 (ms)	Scenario 3 (ms)	Scenario 4 (ms)
10	0.00000591	0.00000739	0.00000405	0.00001241
30	0.00001002	0.00001283	0.00001374	0.00001604
60	0.00000865	0.00001143	0.00001610	0.00001808
100	0.00001792	0.00001570	0.00001920	0.00002209
300	0.00001097	0.00002159	0.00001911	0.00002697
600	0.00001169	0.00002543	0.00002218	0.00003128
1000	0.00001822	0.00002583	0.00003059	0.00003148
3000	0.00002323	0.00003192	0.00003018	0.00003904
6000	0.00001816	0.00003635	0.00003381	0.00004491
10000	0.00003807	0.00003010	0.00003393	0.00004258
30000	0.00002882	0.00002510	0.00004834	0.00004294

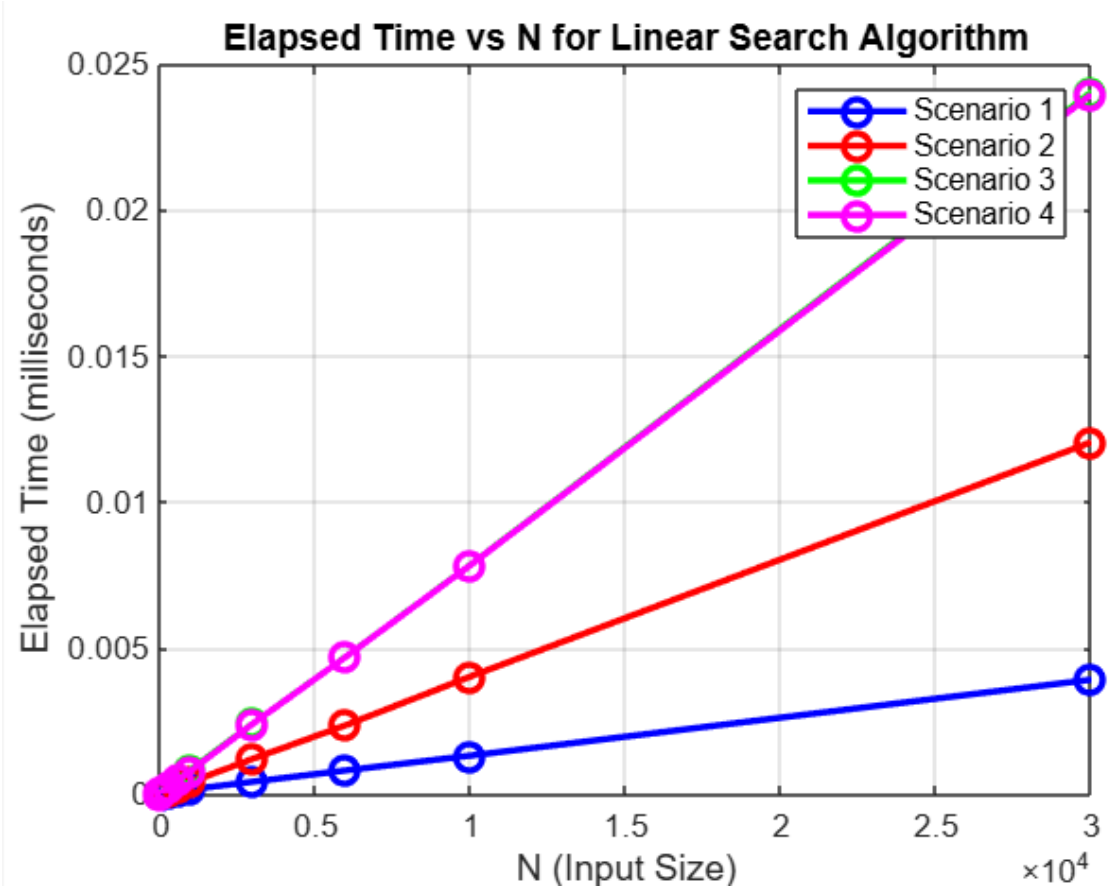
Jump Search Algorithm

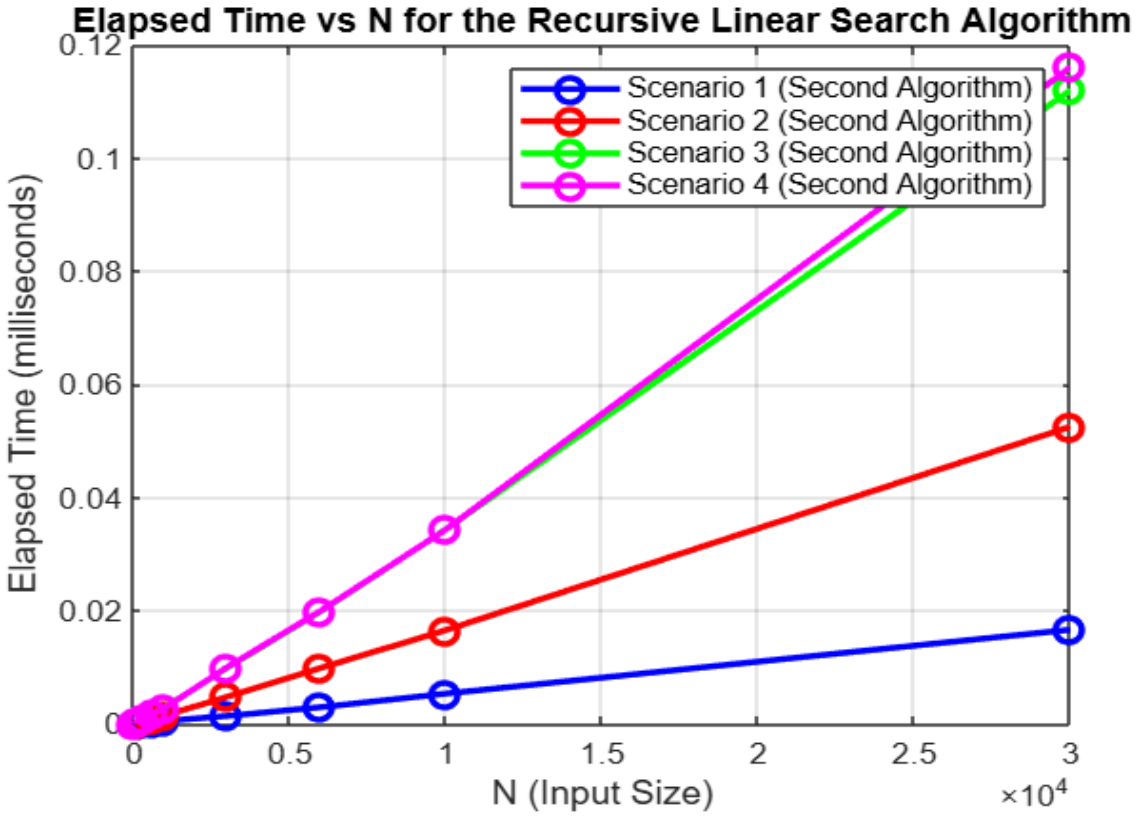
N	Scenario 1 (ms)	Scenario 2 (ms)	Scenario 3 (ms)	Scenario 4 (ms)
10	0.00001548	0.00002547	0.00003307	0.00004699
30	0.00002080	0.00004101	0.00006219	0.00006836
60	0.00003246	0.00005038	0.00008946	0.00011081
100	0.00003028	0.00005797	0.00010815	0.00010698
300	0.00006154	0.00011239	0.00020135	0.00018999
600	0.00005446	0.00015363	0.00029297	0.00024235
1000	0.00008356	0.00018427	0.00032579	0.00032800
3000	0.00011602	0.00035245	0.00056473	0.00053873
6000	0.00028395	0.00049042	0.00087823	0.00073068
10000	0.00029834	0.00066335	0.00111242	0.00091844
30000	0.00057522	0.00100523	0.00173493	0.00160820

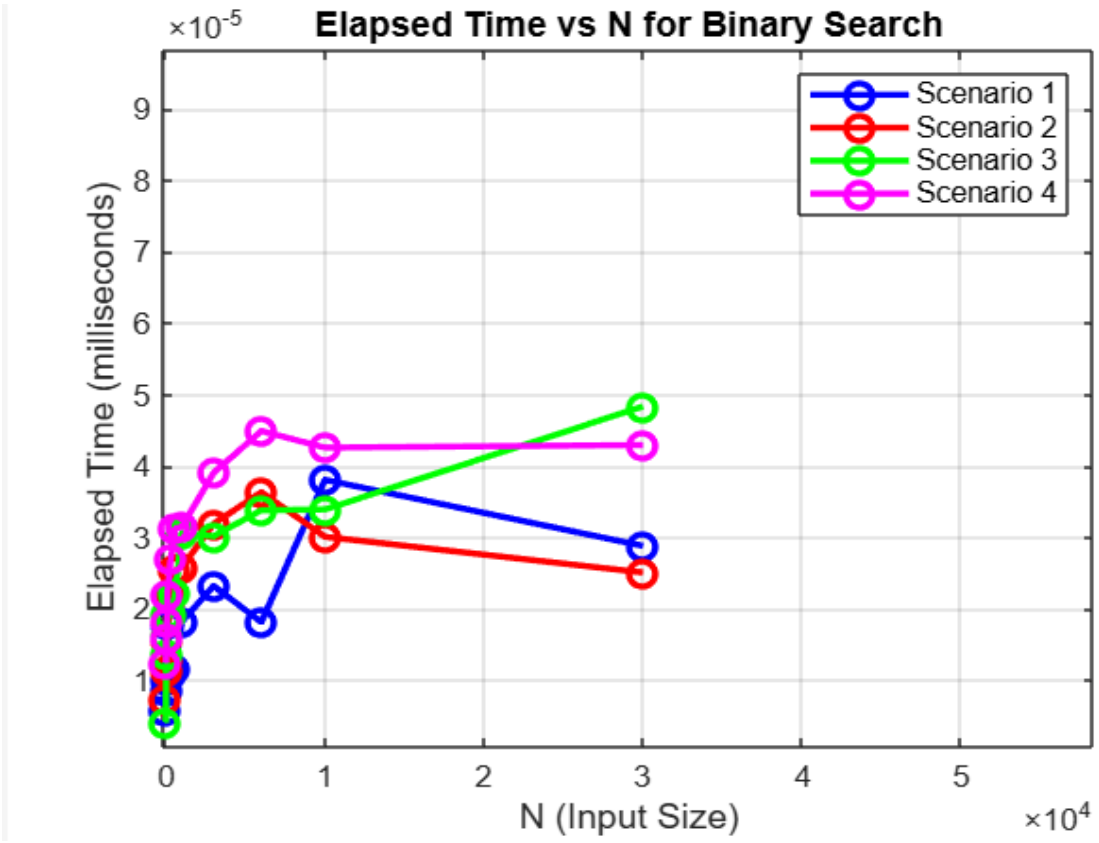
Random Search Algorithm

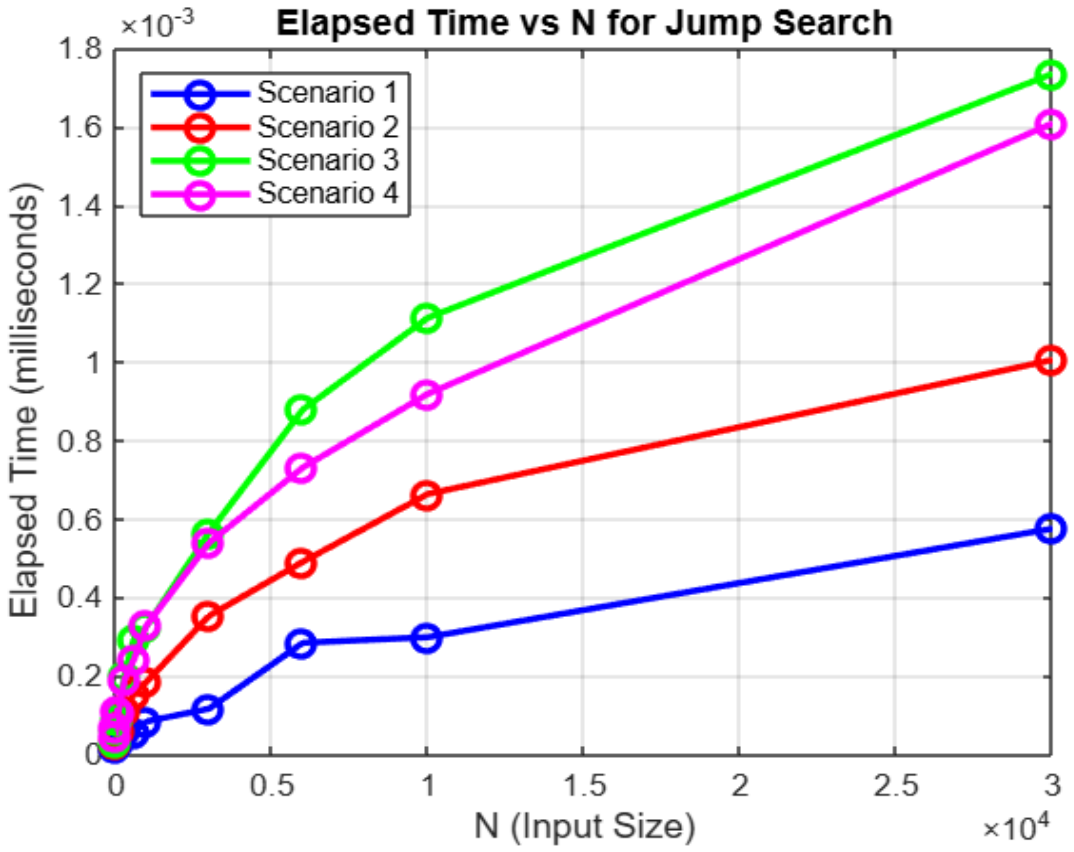
N	Scenario 1 (ms)	Scenario 2 (ms)	Scenario 3 (ms)	Scenario 4 (ms)
10	0.00009625	0.00007879	0.00004552	0.00013658
30	0.00040151	0.00025210	0.00008809	0.00042070
60	0.00010393	0.00041008	0.00011586	0.00076677
100	0.00078184	0.00035147	0.00015684	0.00131434
300	0.00052841	0.00207538	0.00033106	0.00376002
600	0.00095215	0.00198820	0.00060497	0.00764326
1000	0.01068670	0.00656377	0.00096722	0.01215570
3000	0.02476380	0.01970590	0.00282008	0.03679880
6000	0.01527360	0.03925230	0.00566325	0.07216610
10000	0.03195840	0.06509520	0.00922445	0.11983200
30000	0.06945080	0.12107900	0.02914820	0.36401900

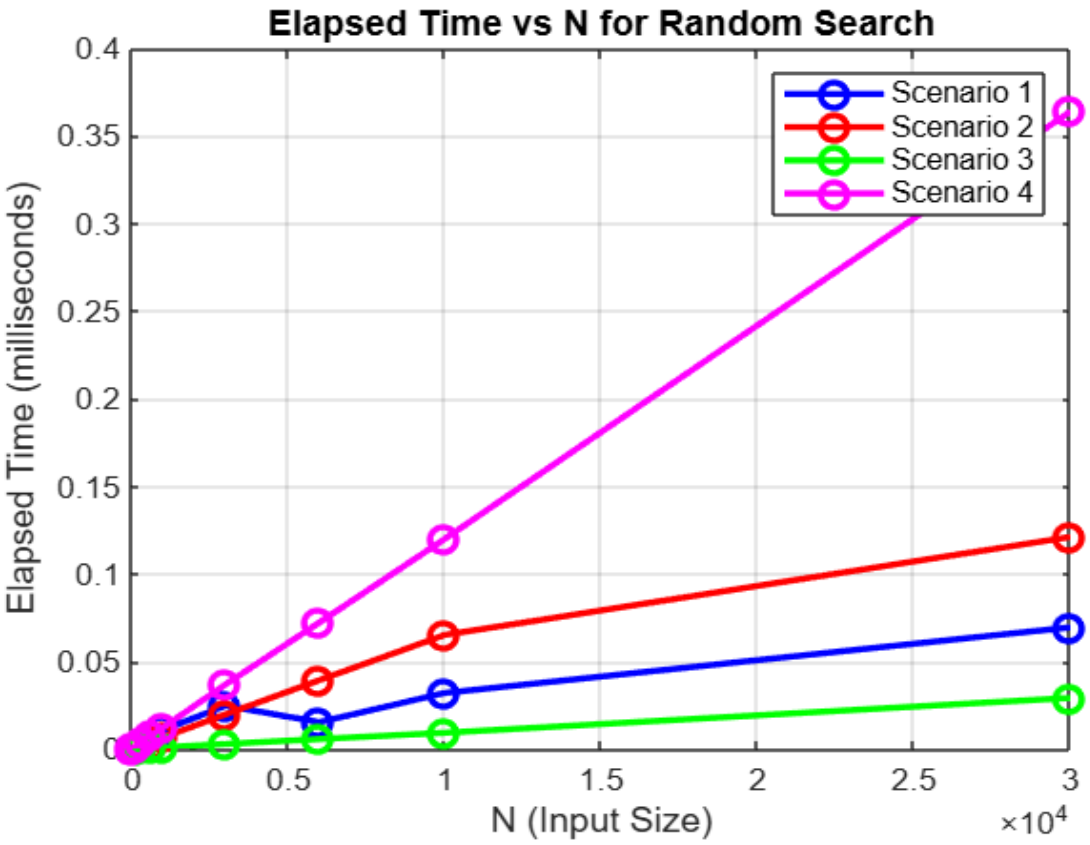
PLOTS:


Elapsed Time vs N for Linear Search Algorithm
Elapsed Time (milliseconds)
N (Input Size)
×10^4
Scenario 1
Scenario 2
Scenario 3
Scenario 4
0.025
0.02
0.015
0.01
0.005
0
0.5
1
1.5
2
2.5
3


Elapsed Time vs N for the Recursive Linear Search Algorithm
Elapsed Time (milliseconds)
N (Input Size)
×10^4
Scenario 1 (Second Algorithm)
Scenario 2 (Second Algorithm)
Scenario 3 (Second Algorithm)
Scenario 4 (Second Algorithm)
0.12
0.1
0.08
0.06
0.04
0.02
0
0.5
1
1.5
2
2.5
3


Elapsed Time vs N for Binary Search
×10-5
Elapsed Time (milliseconds)
N (Input Size)
×10^4
Scenario 1
Scenario 2
Scenario 3
Scenario 4


Elapsed Time vs N for Jump Search
×10-3
Elapsed Time (milliseconds)
N (Input Size)
×10^4
Scenario 1
Scenario 2
Scenario 3
Scenario 4



Computer Specification:

Computer: MONSTER ABRA 5 V18.1

Processor: Intel Core i7-11800H. It is an 11th generation "Tiger Lake" processor with 8 cores and 16 threads, with a base clock of 2.3 GHz and a turbo boost frequency up to 4.6 GHz.

RAM: 16 GB of RAM, configured as 2 x 8 GB DDR4 modules. This RAM operates at a frequency of 3200 MHz.

Operating System: Microsoft Windows 11 Home Single Language.

Discussion:

1)What are the theoretical worst, average, and best cases for each algorithm? What are the corresponding running times for each scenario for each algorithm?

(Scenario1: taken n/6th index, **Scenario2:** taken n/2th index, **Scenario3:** taken (n-3)th index, **Scenario 4:** does not exist.)

Algorithm 1 (Iterative Linear Search):

The iterative linear search algorithm begins with the first index and iterates with a for loop until it finds the same key. So:

The best case of this algorithm occurs when the key is in the beginning. For loop works for 1 case, then returns since it finds the desired key. Thus, the time complexity of the best case becomes O(1).

The average case of this algorithm occurs when the key is around the middle. For loop works for nearly n/2 cases, then returns since it finds the desired key. Thus, the time complexity of the average case becomes O(n/2)=O(n).

The worst case of this algorithm occurs when the key is in the last index or does not exist in the array. For loop work for n cases, then returns since it finds the desired key. Thus, the time complexity of the worst case becomes O(n).

Linear search is not that efficient for large arrays since time complexity becomes O(n) for worst and average cases.

Scenario 1: O(n), Scenario 2: O(n), Scenario 3: O(n), Scenario 4: O(n).

But even though the upper bounds are equal, their running time complexities are:

Scenario 1=1+3n/6, Scenario 2=1+3n/2, Scenario 3=1+3(n-3), Scenario 4=1+3n

Scenario 4>Scenario 3>Scenario 2>Scenario 1.

(Datas to be checked increase.)

Algorithm 2 (Recursive Linear Search):

The recursive linear search works similarly to the iterative linear search, but this time, it uses recursive function calls each time and checks for 1 value in each call.

The best case of this algorithm occurs when the key is in the beginning. Just 1 function call works, then returns since it finds the desired key. Thus, the time complexity of the best case becomes O(1).

The average case of this algorithm occurs when the key is around the middle. The function works for n/2 function calls and then returns since it finds the desired key. Thus, the time complexity of the average case becomes O(n/2)=O(n).

The worst case of this algorithm occurs when the key is in the last index or does not exist in the array. The function works for n function calls and then returns since it finds the desired key. Thus, the time complexity of the worst case becomes O(n).

Recursive linear search is more inefficient than iterative linear search—even if both of them have time complexity O(n) for the worst and the average case—because of the overhead of function calls and potential stack overflow for very large N.

Scenario 1: O(n), Scenario 2: O(n), Scenario 3: O(n), Scenario 4: O(n).

But even though the time complexities are equal, their running time complexities are:

Scenario1=1+8n/6, Scenario2=1+8n/2, Scenario3=1+8(n-3), Scenario4=1+7n

Scenario 4>Scenario 3>Scenario 2>Scenario 1.

(Datas to be checked increase.)

Algorithm 3 (Binary Search):

The binary search decreases size logarithmically by dividing it by 2. It only works for sorted arrays.

The best case of this algorithm occurs when the key is in the middle. Just 1 function call works, then returns since it finds the desired key. Thus, the time complexity of the best case becomes O(1).

The average case of this algorithm occurs when the key is selected randomly. The loop decreases its size by dividing it by 2, so it works for log2(n) time, then returns since it finds the desired key. Thus, the time complexity of the average case becomes O(log2(n))=O(log(n)).

The worst case of this algorithm occurs when a maximum number of divides reaches the key or does not exist in the array. The function looks for all halving cases and returns after finding the desired key. Thus, the time complexity of the worst case becomes O(log2(n))=O(log(n)).

Binary search is actually very efficient, especially while working with large sizes. However, to be able to apply this algorithm, the array must be sorted first.

Scenario 1: O(logn), Scenario 2: O(logn), Scenario 3: O(logn), Scenario 4: O(logn).

But even though the upper bounds are equal, their running time complexities are:

Scenario1=2+4log2(n/6), Scenario2=2+4log2(n/2), Scenario3=2+4log2(n-3), Scenario4=2+4log2(n)

Scenario 4>Scenario 3>Scenario 1>Scenario 2.

(Datas to be checked increase.)

Algorithm 4 (Jump Search):

The jump search looks at blocks-sized √n Then, it looks at the relevant block with a linear search.

The best case of this algorithm occurs when the key is in the first element of the first block. It works with 1 jump and 1 comparison. Thus, the time complexity of the best case becomes O(1).

The average case of this algorithm occurs when the key is selected randomly. It works with √n jumps and √n comparison. Thus, the time complexity of the average case becomes O(2√n)=O(√n).

The worst case of this algorithm occurs when it reached index at the last index of last block.It works with √n jumps and √n comparison. Thus, the time complexity of the average case becomes O(2√n)=O(√n).

Jump search is more efficient when compared with linear searches since it works at O(√n) at average and worst case. However, the array must be sorted first.

Scenario 1: O(√n), Scenario 2: O(√n), Scenario 3: O(√n), Scenario 4: O(√n).

Scenario1=3+8√n/6, Scenario2=3+8 √n/2, Scenario3=3+8(√n-3), Scenario4=3+8√n

But even though the upper bounds are equal, their running time complexities are:

Scenario 3>Scenario 4>Scenario 2>Scenario 1.

(Datas to be checked increase.)

Algorithm 5 (Random Search):

Random search randomly looks at indexes and continues until it reaches the desired key.

The best case of this algorithm occurs when the key is in the first random check. It works with just 1 comparison. Thus, the time complexity of the best case becomes O(1).

The average case of this algorithm occurs when the key is found after N/2 random checks. It works with n/2 comparisons. Thus, the time complexity of the average case becomes O(n/2)=O(n).

The worst case of this algorithm occurs when it reaches the index at the last check of the random indexes or does not exist in the array. It works for n comparisons. Thus, the time complexity of the average case becomes O(n).

Even though its time complexity is O(n) for worst case,the algorithm is less efficient and practical than binary search or jump search.

Scenario 1: O(n), Scenario 2: O(n), Scenario 3: O(n), Scenario 4: O(n).

their running time complexities are:

Scenario1 a value between 12+9,15n+5,

Scenario2 a value between 12+9,15n+5,

Scenario3 a value between 12+9,15n+5,

Scenario4 a value between 15n+5,

Scenario 1=Scenario 2=Scenario 3<Scenario 4.

(Datas to be checked decided randomly.)

2)What are the worst, average, and best cases for each algorithm based on your table and your plots? Do theoretical and observed results agree? If not, give your best guess as to why that might have happened.

Algorithm 1 (Iterative Linear Search):

According to my table and plot the best case occurs when the key is in the scenario 1. The average case occurs in scenario2 and worst case occurs at scenario3 and scenario4.

In my tables i have taken scenerio1 close to begining not the first index. So the best case for my table seen as O(n) instead of O(1). However ,its because of index selecting. Other than that observed results agree with theorotical results.

Algorithm 2(Recursive Linear Search):

According to my table and plot the best case occurs scenario1.The average case occurs at scenario2,scenario3 and worst case occurs at scenario4.

In my tables i have taken scenerio1 close to begining not the first index. So the best case for my table seen as O(n) instead of O(1). However ,its because of index selecting. Other than that observed results agree with theorotical results.

Algorithm 3(Binary Search):

According to my table and plot the best case occurs scenario2. The average case occurs ar scenario1 and scenario3 and worst case occurs at scenario4.

In my tables i have taken middle+1th index as scenario2’s key not the middle index. So the best case for my table seen as O(logn) instead of O(1). However ,its because of index selecting. Other than that observed results agree with theorotical results.

Algorithm 4(Jump Search):

According to my table and plot the best case occurs at scenario1. The average case occurs at scenario2 and scenario4 and worst case occurs at scenario3.

In my tables i have taken scenerio1 close to begining not the first index. So the best case for my table seen as O(square roots of n) instead of O(1). However ,its because of index selecting. Other than that observed results agree with theorotical results.

Algorithm 5(Random Search):

According to my table and plot the best case occurs at scenario 3. The average case occurs at scenario1 and scenario2 and worst case occurs at scenario 4.

Theorotical results and practical results agree in most of the algorithms except the Random Search algorithm. This difference stem from my implementation of algorithm. In my algorithm i look for the random index and look for the last index of the array if both of them doesn't equal to desired key ,i swap them and decrease size by 1 and continue doing this by incrementing array by 1 each time. This algorithm favors scenario 3 since it also examines is the last index equal to key or not. Thus, speed of algorithms are like this:

Scenario3>Scenario1>Scenario2>Scenario4.

General reasons why practical datas might not agree with theorotical predictions:

Due to other processes computer makes at the same time, some of datas makes small differenceses with the theorotical datas.

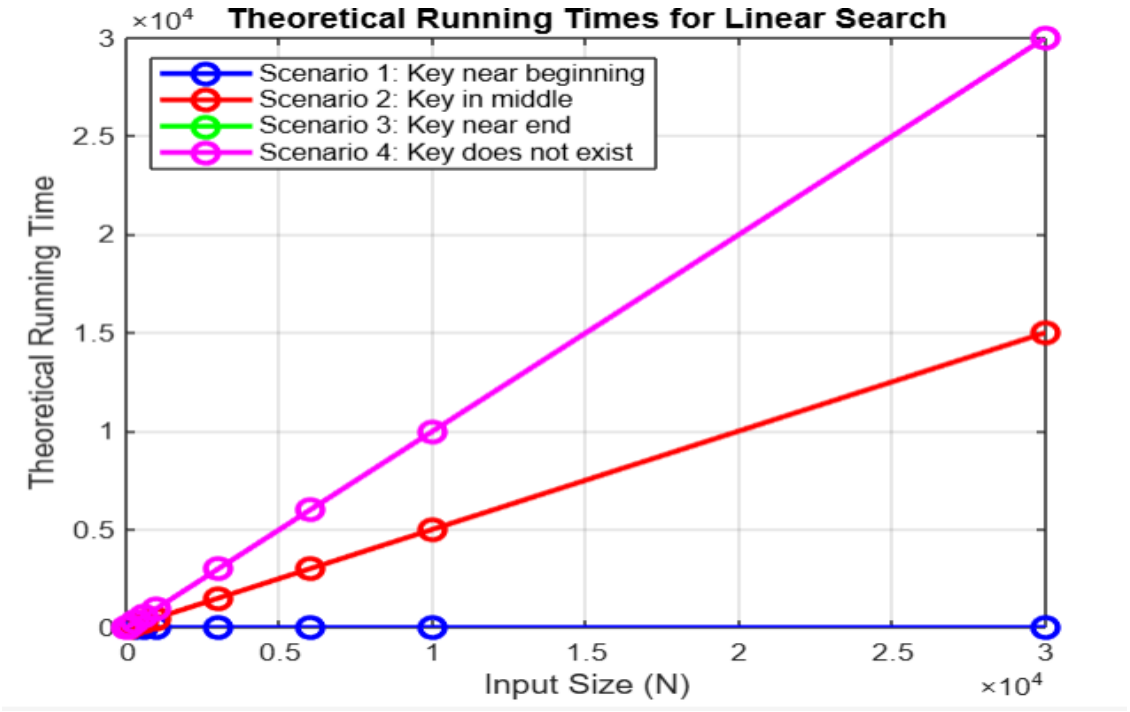
1)Variability caused by system noise (specific background processes) might lead to anomalies in reported runtimes.

2) Large arrays that fit in the cache could have a faster runtime per element than arrays that require frequent access to the RAM. The larger variations in the values might be the result of stack operations.

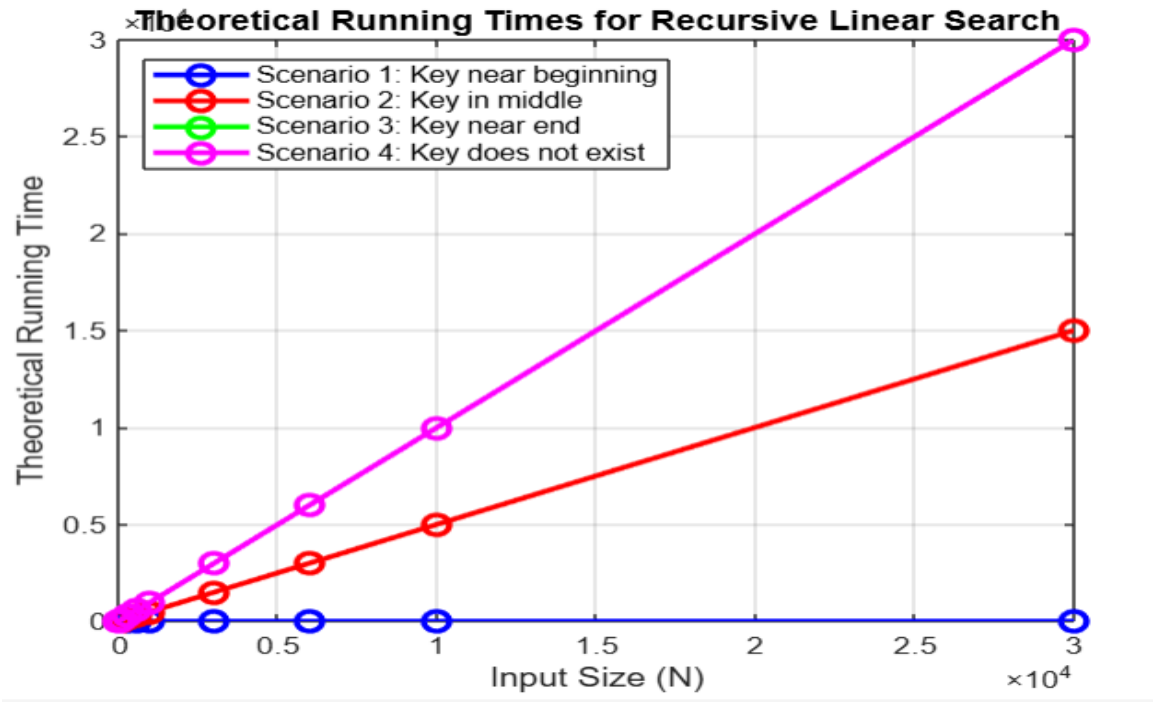
3)Every recursive call pushes function parameters and return addresses. This adds additional overhead compared to an iterative technique, which does not require such stack management. Additionally, when stack frames surpass cache limits and result in cache misses that might induce oscillations, recursive calls may lead to scattered memory access patterns.

Hence, there might be some cases that computer wants to fool us by fluctuating datas and unsatisfying theorotical results. However, these exceptions doesnt affect the general trends of plots, Still tables and plots agrees with the theorotical best, average and worst cases.

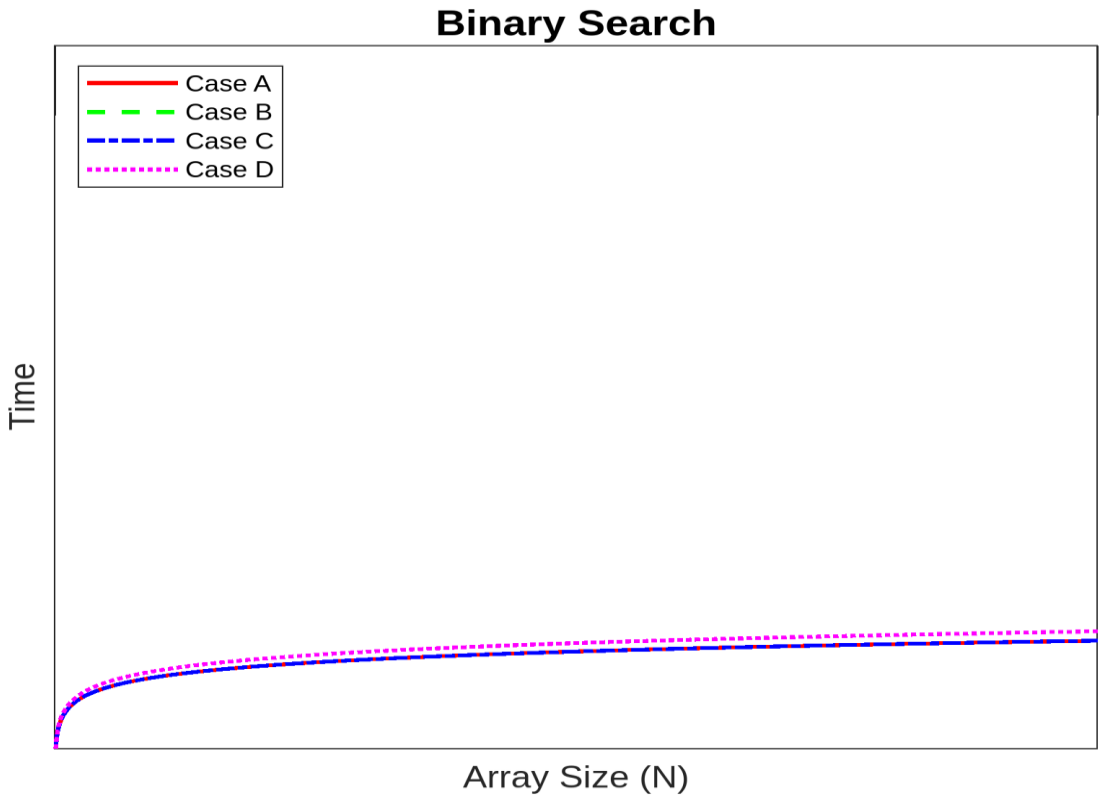
3)Plot theoretical running times for each algorithm across different scenarios and compare them to the plots at step 4. Comment on consistencies and inconsistencies.



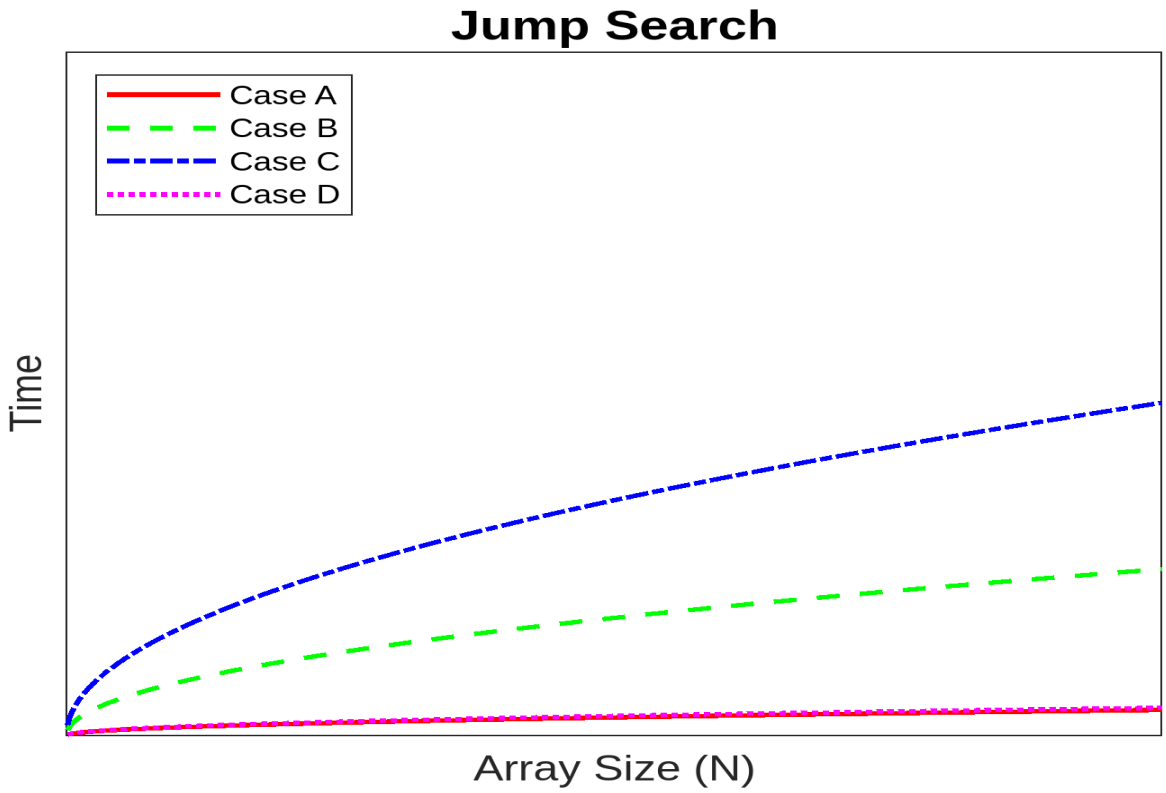
When i compare this with my graph, its clear that graphs looks very similar. Only difference is that i have chosen indexes close to begining in scenario 1 instead of first index and the count of my selectable indexes increase with size too, because being close must be relative to the size. That is why my scenario 1 makes an small angle with x axis and is O(n), not O(1).



In recursive linear search we can say the exact same things that we said in linear search part. It is mostly agrees with my results.



Scenario4, the worst scenario, is worse than the others. The remaining three scenarios are all the same and reflect the typical case. If we look at the thorotical and our practical plots we can see that the binary search performs better in practice than in theory, as seen by the fact that the actual values are lower than the theoretical values. This is because of technical advancements as the CPU and cache become more efficient. At the end of the day even if datas fluctuate, in general theorotical results agree with tables and plots.

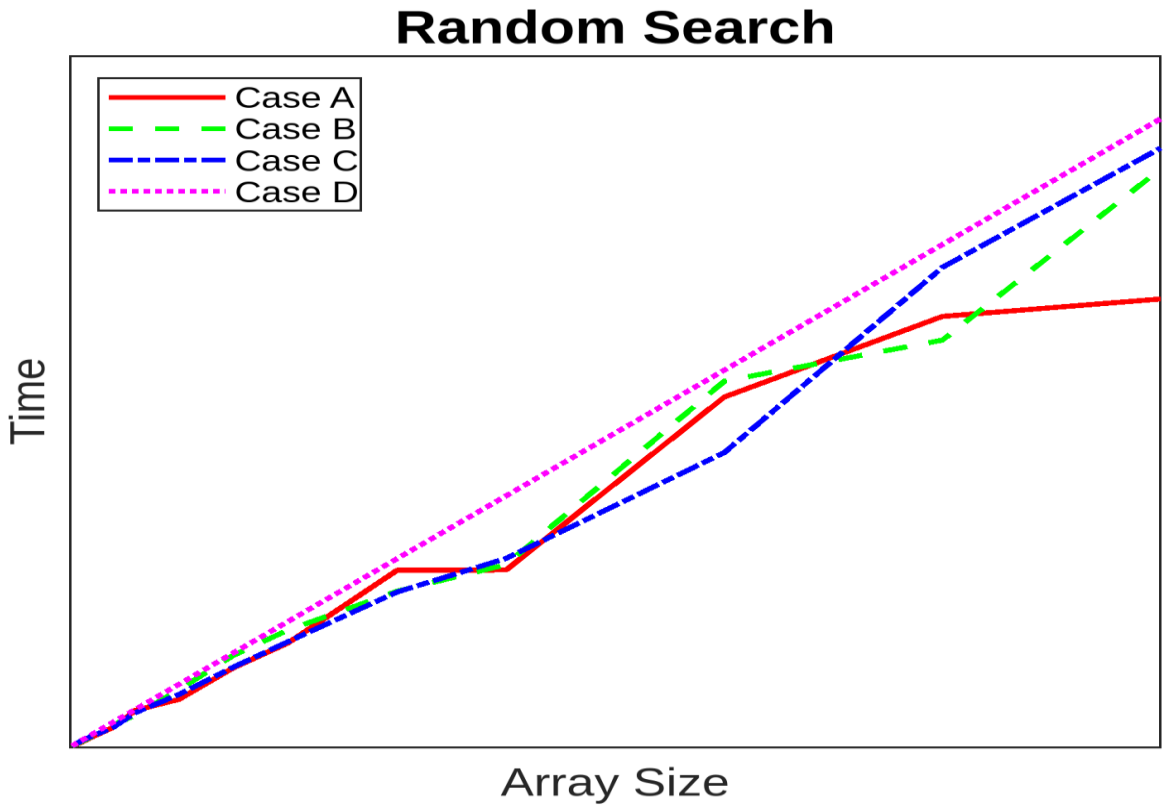


In terms of jump search we can say that our plots agree with theorotical result except for Scenario 1 and 4. The difference stems from different selecting styles of doesnt existing index and beginning index.

For Scenario4: In this theorotical graph selects its index as a number smaller than all possible values, so that it doesnt go through any loop it just returns-1 after first comparison. However, in my practical results i have selected my index bigger than all possible values, so that it goes through all of the loops and at the end returns –1.

For Scenario1: In this theorotical graph selects its index as first index.However, i have selected it as size/6 to make it close to beginning, not at the beginning.

After looking reasons of exceptions we can say that they are just because of selecting different indexes. Thus we can clearly state that our practical plot results agree with theorotical results.



In random search plots agree with each other in most cases. They just have difference at Scenario 3 which is caused by different method implementation style. As i have explained previously my method algorithm favors scenario3 because of its extra checking condition for last index. So my practical plot shows scenario3 as less time taker case. Furthermore this extra checking conditions makes faster other scenarios too, because advanced compilers nowadays saving small datas about all check conditions even if it wont use at that step. Compiler can use that small datas when there is a need for it. Hence, thats why my scenario1, scenario2 and scenario3 seems a bit far from scenario4. Other than that, this graph actually matches exactly with my plot espacially for small n's. In real life when n's get bigger theorotical plot and observed plots begins to differantiate. However still most of datas agrees. We can say that theorotical and practical results matches generally.