2020-2021 Spring Semester

CS201 Homework 2 - Report

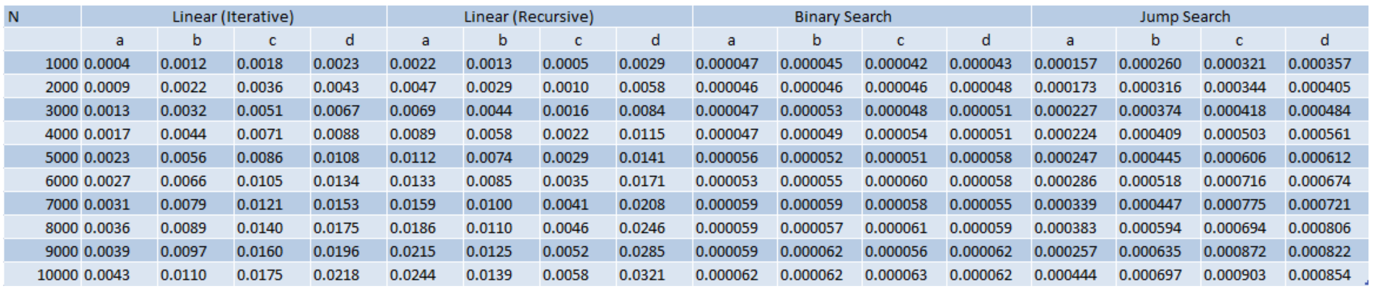
Name: Alper

Surname: Mumcular

Student ID: 21902740

Date: 10.04.2021

1) Table of all the values recorded



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
1000	0.0004	0.0012	0.0018	0.0023	0.0022	0.0013	0.0005	0.0029	0.000047	0.000045	0.000042	0.000043	0.000157	0.000260	0.000321	0.000357
2000	0.0009	0.0022	0.0036	0.0043	0.0047	0.0029	0.0010	0.0058	0.000046	0.000046	0.000046	0.000048	0.000173	0.000316	0.000344	0.000405
3000	0.0013	0.0032	0.0051	0.0067	0.0069	0.0044	0.0016	0.0084	0.000047	0.000053	0.000048	0.000051	0.000227	0.000374	0.000418	0.000484
4000	0.0017	0.0044	0.0071	0.0088	0.0089	0.0058	0.0022	0.0115	0.000047	0.000049	0.000054	0.000051	0.000224	0.000409	0.000503	0.000561
5000	0.0023	0.0056	0.0086	0.0108	0.0112	0.0074	0.0029	0.0141	0.000056	0.000052	0.000051	0.000058	0.000247	0.000445	0.000606	0.000612
6000	0.0027	0.0066	0.0105	0.0134	0.0133	0.0085	0.0035	0.0171	0.000053	0.000055	0.000060	0.000058	0.000286	0.000518	0.000716	0.000674
7000	0.0031	0.0079	0.0121	0.0153	0.0159	0.0100	0.0041	0.0208	0.000059	0.000059	0.000058	0.000055	0.000339	0.000447	0.000775	0.000721
8000	0.0036	0.0089	0.0140	0.0175	0.0186	0.0110	0.0046	0.0246	0.000059	0.000057	0.000061	0.000059	0.000383	0.000594	0.000694	0.000806
9000	0.0039	0.0097	0.0160	0.0196	0.0215	0.0125	0.0052	0.0285	0.000059	0.000062	0.000056	0.000062	0.000257	0.000635	0.000872	0.000822
10000	0.0043	0.0110	0.0175	0.0218	0.0244	0.0139	0.0058	0.0321	0.000062	0.000062	0.000063	0.000062	0.000444	0.000697	0.000903	0.000854

Figure 1

Important Note: In main.cpp file, to find the values of Binary Search and Jump Search, I used 1.000.000 times for loop and divide the duration to 10.000. After that I added two 0's to the values because If I divide 1.000.000, I will get a value which contains "e".

2) Plot for each search algorithm

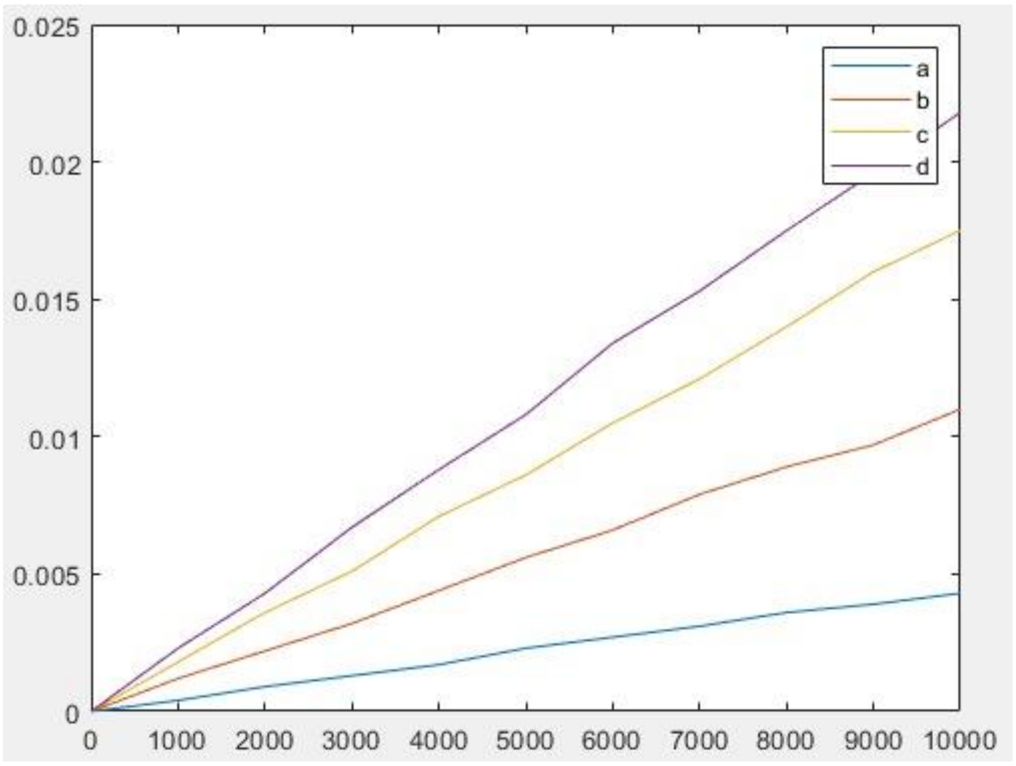


Figure 2: Plot for algorithm 1 according to table

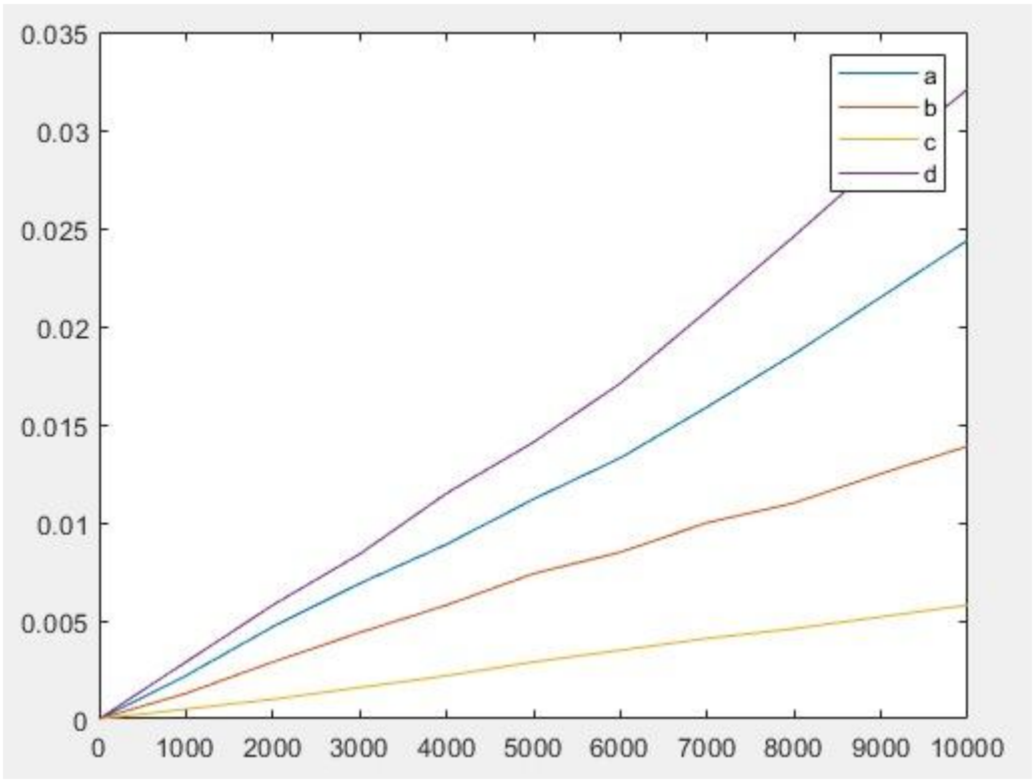


Figure 3: Plot for algorithm 2 according to table

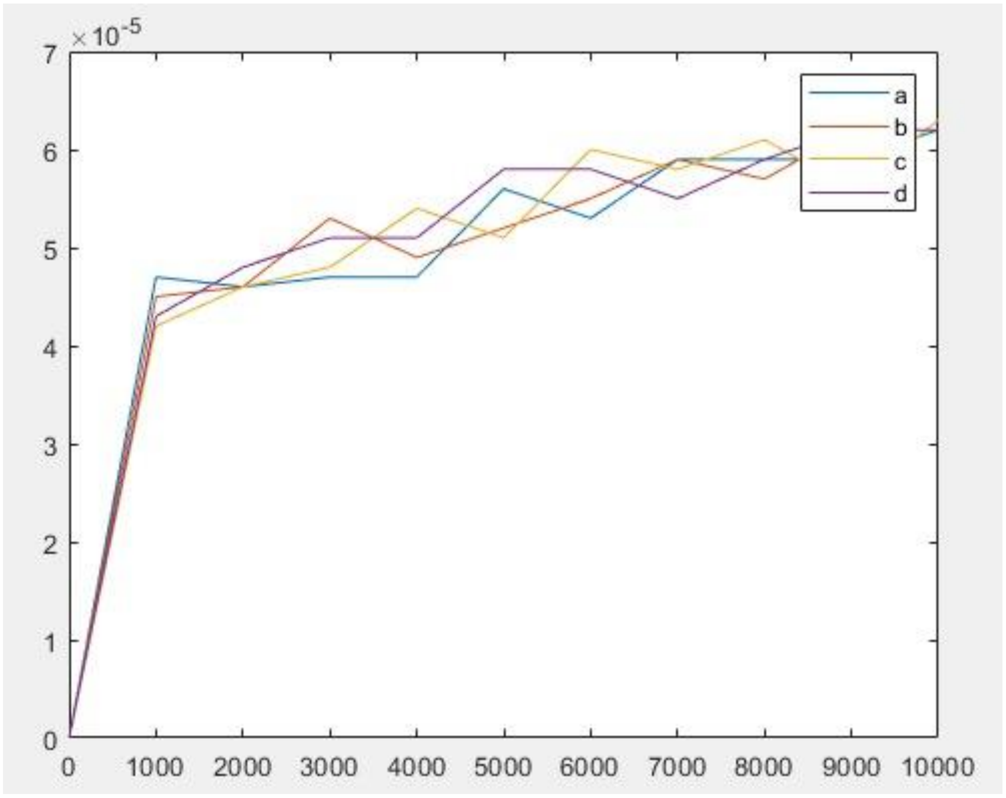


Figure 4: Plot for algorithm 3 according to table

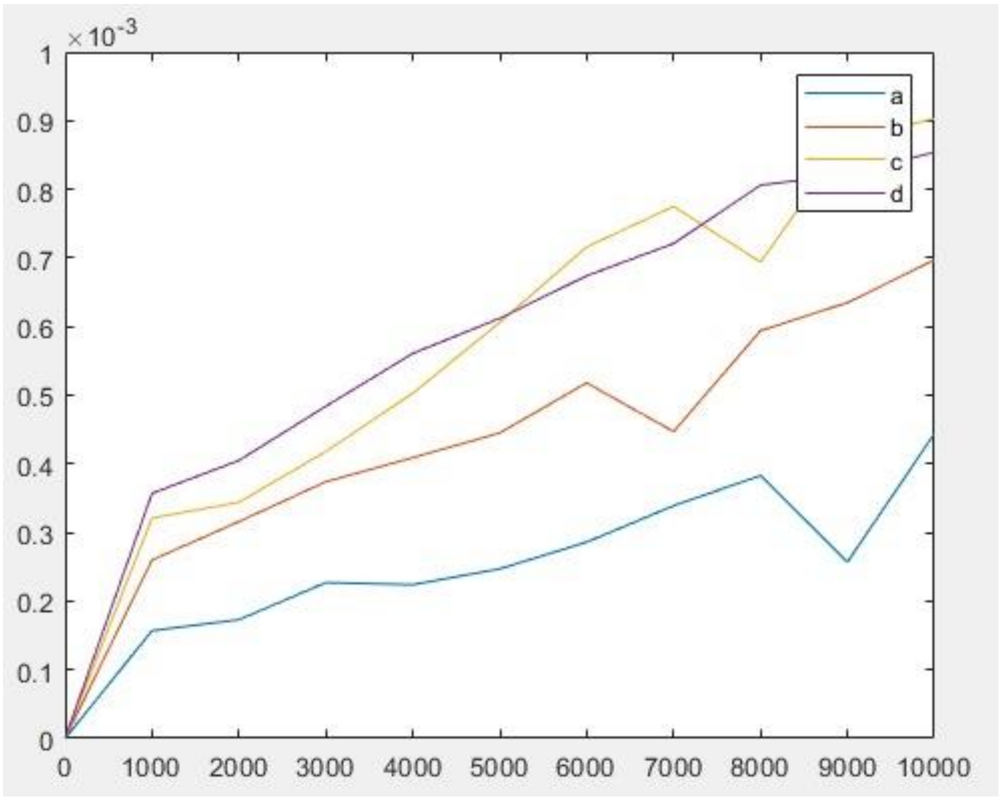


Figure 5: Plot for algorithm 4 according to table

3) Comments on Result

1) Specifications of my computer:

Processor: Intel Core i7-4790 CPU @ 3.60GHz

GPU: NVIDIA GeForce GTX970

RAM: 16 GB

Operating System: Windows 8.1 x64 bit

HDD: 1 TB

2) Theoretical worst, average, and best cases for each algorithm

For a successful search:

Best cases for each algorithm:

Linear(Iterative) = O(1) (if the key is at the index 0)

Linear(Recursive) = O(1) (if the key is at the index n-1 (last))

Binary Search = O(1) (if the key is at arr[mid])

Jump Search = O(1) (if the key is at the index 0)

Average cases for each algorithm:

Linear(Iterative) = O(N)

Linear(Recursive) = O(N)

Binary Search = O(log(N))

Jump Search = O(√N)

Worst cases for each algorithm:

Linear(Iterative) = O(N) (if the key is at the index n-1 (last))

Linear(Recursive) = O(N) (if the key is at the index 0)

Binary Search = O(log(N))

Jump Search = O(2√N - 1) = O(√N)

For an unsuccessful search:

Linear(Iterative) = Best case, worst case, and average case are all same = O(N)

Linear(Recursive) = Best case, worst case, and average case are all same = O(N)

Binary Search = Best case, worst case, and average case are all same = O(log(N))

Jump Search = Best case, worst case, and average case are all same = O(√N)

3) Worst, average, and best cases for each algorithm based on table

Linear(Iterative):

Best case = scenario a, O(0.2N) = O(N)

Average case = scenario b, O(0.5N) = O(N)

Worst case = scenario d (if unsuccessful) O(N)

scenario c (if successful) O(0.8N) = O(N)

Theoretical and observed results agree in Linear (Iterative) Search.

Linear(Recursive):

Best case = scenario c, O(0.2N) = O(N)

Average case = scenario b, O(0.5N) = O(N)

Worst case = scenario d (if unsuccessful) O(N)

scenario a (if successful) O(0.8N) = O(N)

Theoretical and observed results agree in Linear (Recursive) Search.

Binary Search:

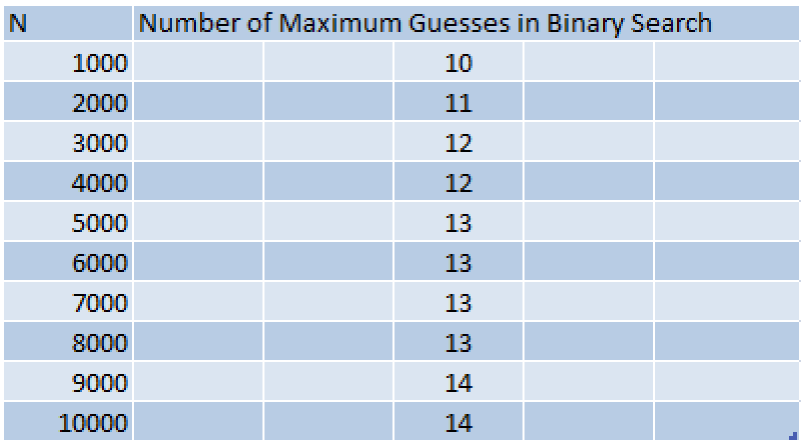
Best case = None of the scenarios is the best case because being at the beginning, middle, or end is not important for binary search because binary search checks the middle element of the array.

Worst case = None of the scenarios seems the worst case according to the table.

Theoretical and observed results do not agree with this part. The reason for this might be the elements I am looking for in each scenario found at the maximum guesses or the computer might have been speeding up while computing the running time in scenario d. Even though I tried to change the key value in each scenario, and re-run, again and again, scenario d did not seem like the worst case.

Average case = scenario a, b, c, d O(log(N))

Scenario d should not have been in this part because the computer always needs the number of maximum guesses to understand that the element is not in the array. However, according to Figure 4, scenario d does not seem like the worst case.



N	Number of Maximum Guesses in Binary Search
1000	10
2000	11
3000	12
4000	12
5000	13
6000	13
7000	13
8000	13
9000	14
10000	14

Figure 6: Number of maximum guesses in selected array sizes

Jump Search:

Best case = scenario a, O(0.2√N + √N - 1) = O(√N)

Average case = scenario b, O(0.5√N + √N - 1) = O(√N)

Worst case = scenario d (if unsuccessful) O(2√N-1) = O(√N)

scenario c (if successful), O(0.8√N + √N - 1) = O(√N)

Theoretical and observed results generally agree in Jump Search. Sometimes scenario c took longer than scenario d. Also, array[7000] in scenario b and array[9000] in scenario a seem bad in Figure 5. The reason for this might be the index of the element I am looking for in the array is at arr[k(√N)] where k is an arbitrary integer. For that reason, I changed the elements but the running times do not differ too much. Hence, the most possible reason for that could be the computer might be slow down while computing the scenarios I mentioned above.

3) Plotting theoretical expected running times for each algorithm

Linear(Iterative):

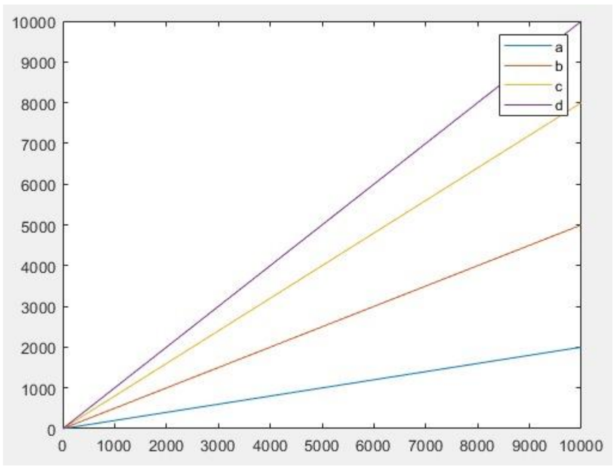


Figure 7: Plot for algorithm 1

This plot and the plot according to the table are consistent. On each array size, scenario d took longer, and scenario a took less time. The plot according to the table is not as smooth as this one. This could be due to the computer while computing the running times.

Linear(Recursive):

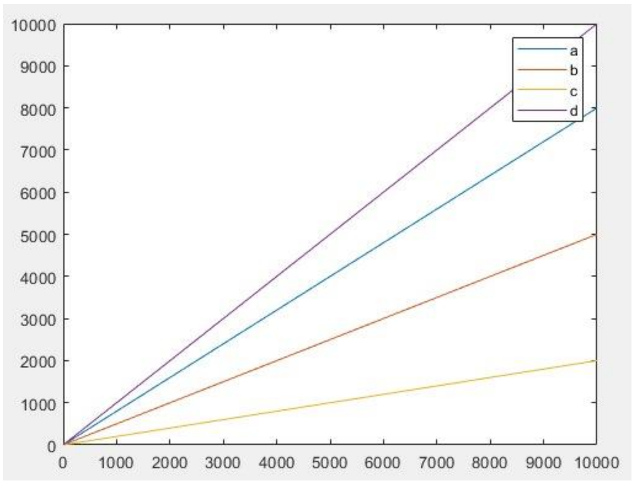


Figure 8: Plot for algorithm 2

This plot and the plot according to the table are consistent. On each array size, scenario d took longer, and scenario c took less time. The difference between algorithm 1 and this, recursive linear search starts checking from the end of the array whereas algorithm 1 starts checking from the index 0. The plot according to the table is not as smooth as this one. This could be due to the computer while computing the running times.

Binary Search:

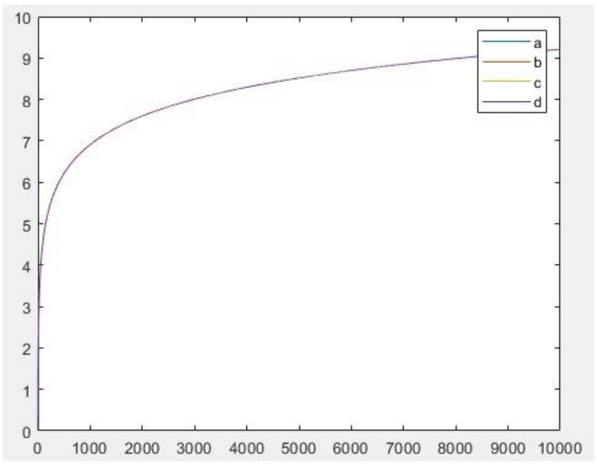


Figure 9: Plot for algorithm 3

This plot and the plot according to the table are similar, but there are some inconsistencies. In the plot according to the table, In each N, the worst case changes. For example, N = 1000 worst case is scenario a, N = 2000 worst case is scenario d (which is expected), N = 3000 worst case is scenario b, and N = 4000 worst case is scenario c. All these examples make it difficult to comment on which scenario is the worst case or best case.

Jump Search:

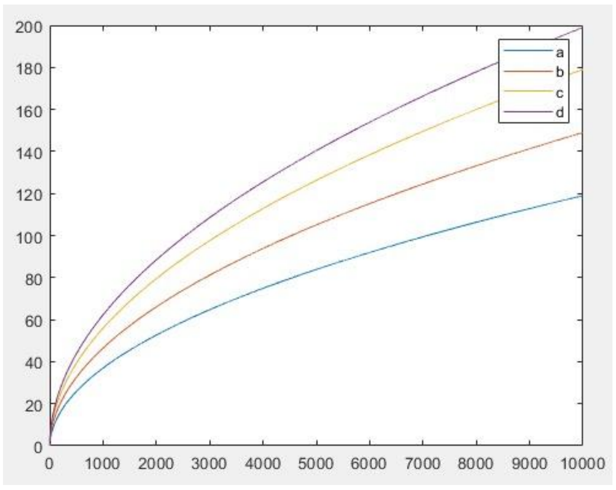


Figure 10: Plot for algorithm 4

This plot and the plot according to the table are mostly consistent, but In some N's there are some inconsistencies. Except N = 6000, and N = 7000 worst, average, and best scenarios are all same (a is the best scenario, d is the worst scenario). In N = 7000 scenario b and N = 9000 scenario a lower the quality of the plot in Figure 5 because these 2 inconsistencies make the plot sharper, but the plot should be smooth.