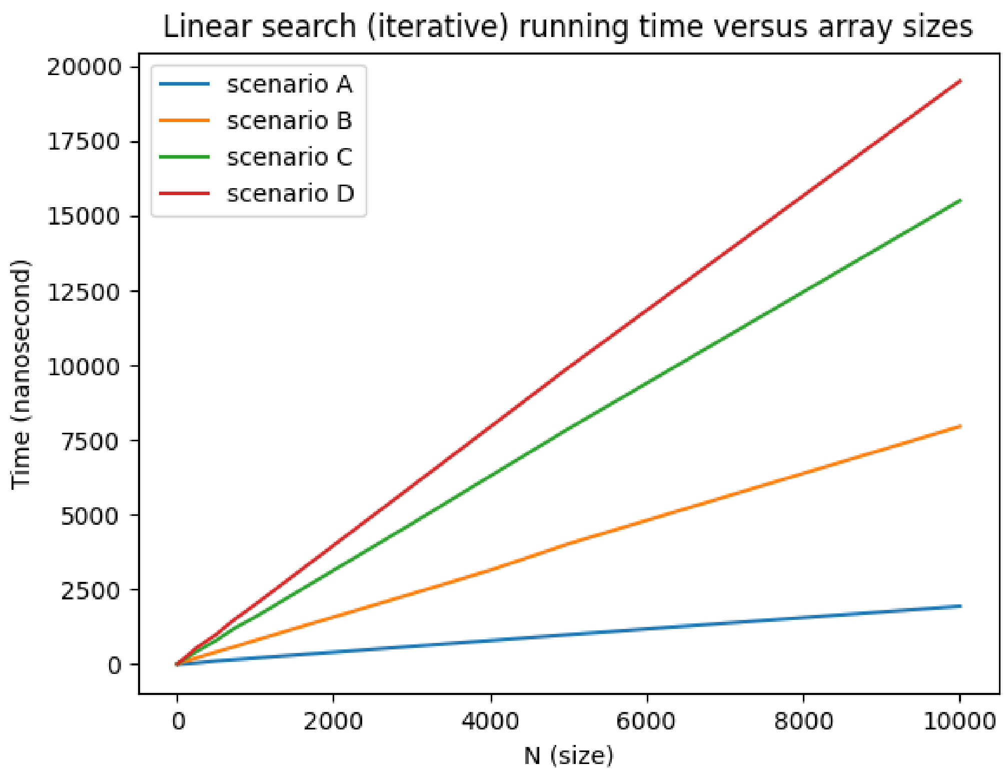
Furkan Güzelant

21901515 Sec: 1

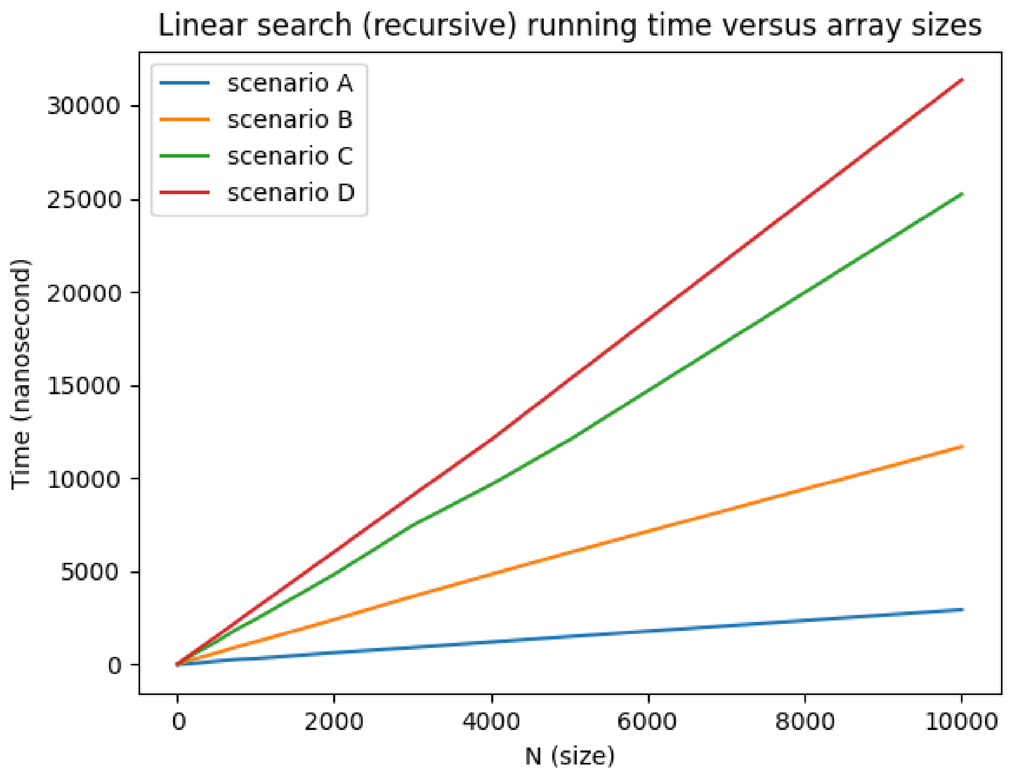
CS201 Homework 2

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	4.0	8.0	14.9	16.0	5.9	13.9	25.9	30.9	7.0	5.0	8.9	10.9	13.9	23.9	37.9	58.8
100	17.9	92.7	171.5	208.4	31.9	155.6	273.3	315.9	21.9	18.9	12.0	19.0	22.9	61.8	106.7	130.6
250	46.9	210.4	416.8	543.5	74.8	326.1	643.3	764.0	24.9	14.9	21.9	22.0	51.8	110.7	197.4	234.3
500	111.7	407.9	794.8	989.4	182.5	617.3	1211.7	1493.9	30.9	17.0	27.9	24.9	55.8	120.7	253.3	298.2
750	159.6	597.4	1221.7	1523.9	272.2	931.5	1866.0	2267.0	28.9	24.9	24.9	28.9	88.7	166.5	305.2	360.0
1000	209.4	797.9	1576.7	1992.6	320.1	1211.8	2420.5	3016.7	21.9	32.9	32.9	28.9	68.8	245.3	391.9	414.9
2000	401.9	1577.8	3137.6	3969.4	634.3	2407.6	4824.1	6034.8	34.9	36.9	24.9	32.9	118.7	262.3	490.7	576.4
3000	599.4	2353.7	4697.4	5948.0	910.6	3650.4	7455.1	9063.8	31.9	36.9	24.9	35.9	155.5	318.1	599.4	698.1
4000	794.8	3143.6	6287.2	7928.8	1202.7	4832.0	9639.2	12043.8	38.9	28.9	37.9	36.9	138.6	388.0	746.0	783.9
5000	991.3	4026.2	7870.9	9916.4	1499.0	6005.9	12041.8	15274.2	42.9	28.9	37.9	38.9	122.7	460.8	722.1	878.6
10000	1942.0	7952.7	15501.7	19501.6	2936.7	11675.3	25227.4	31355.8	48.6	40.8	40.6	48.7	142.6	488.9	946.4	1170.1

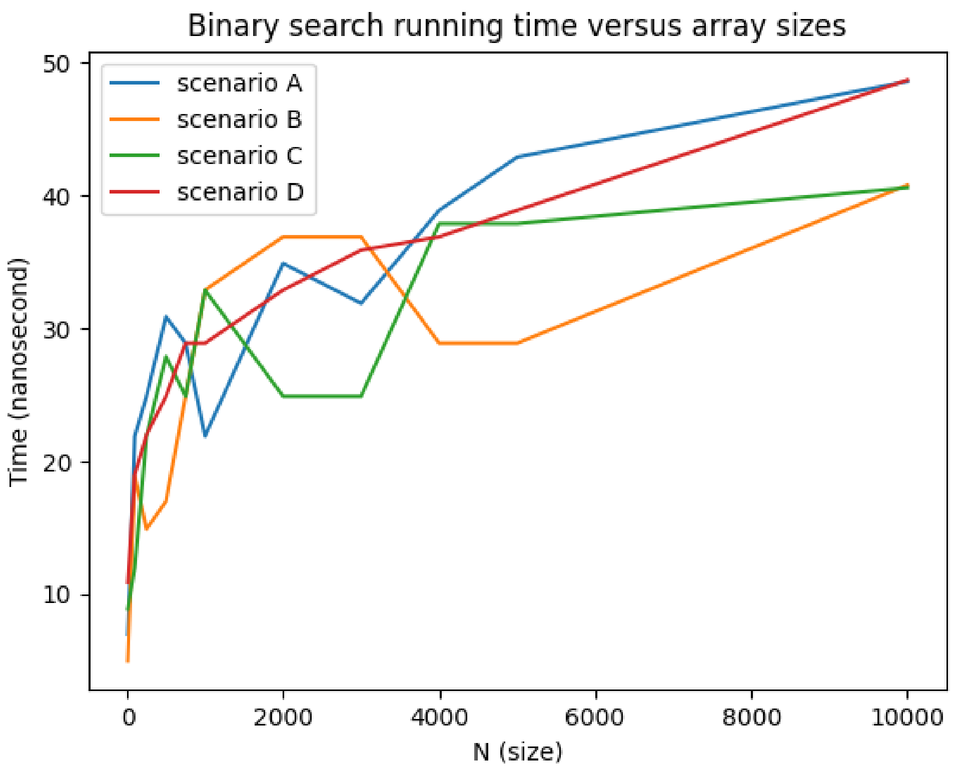
Table 1: The running time of algorithms (in nanoseconds) with different array sizes



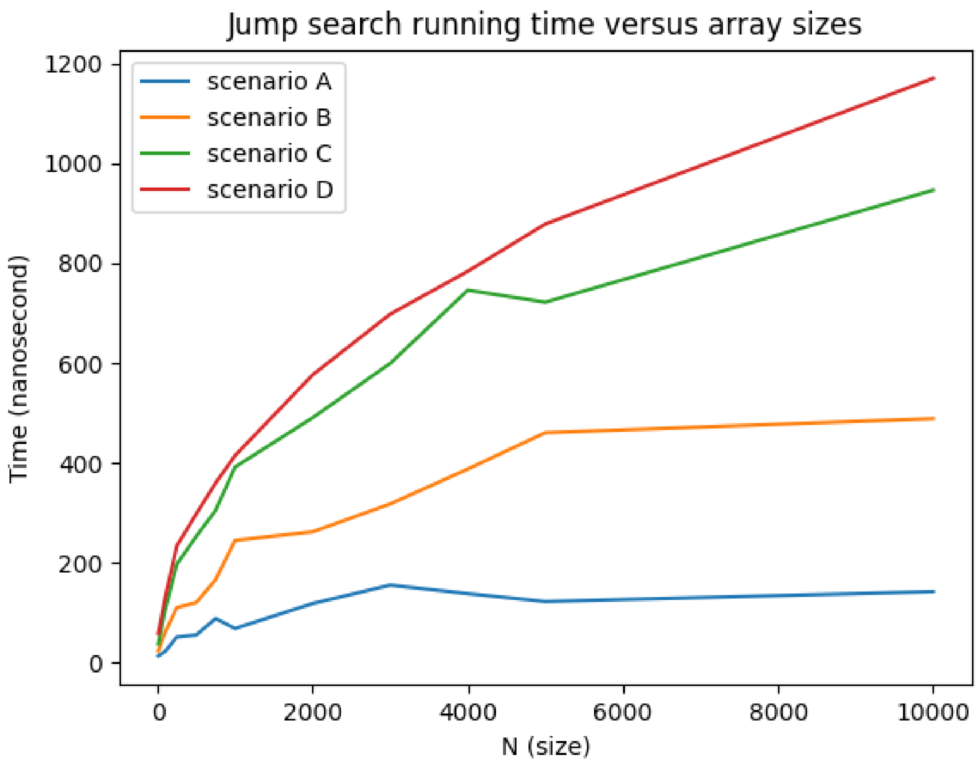
Plot 1: Linear search (iterative) running time with respect to array size (N)



Plot 2: Linear search (recursive) running time with respect to array size (N)



Plot 3: Binary search running time with respect to array size (N)



Plot 4: Jump search running time with respect to array size (N)

<u>Specifications of the system:</u>

Excalibur G770

Processor: Intel® Core™ i7-9750H CPU @ 2.60 GHZ

RAM: 8 GB

Operating system: Windows 10 Home 64 bit

Algorithm 1 – Linear Search (Iterative)

In successful search:

Theoretical worst case: When the item is in the last place on the array, the worst case occurs. N iterations are necessary thus its time complexity is O(N). It is likely to be the scenario that we named as C in our discussion.

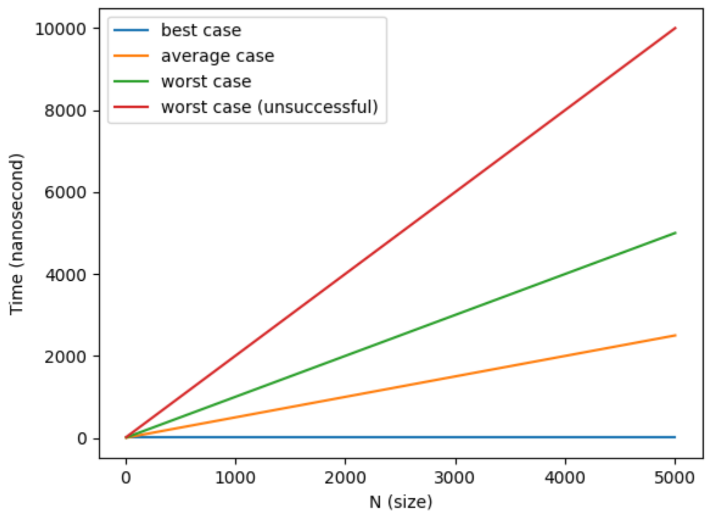
Theoretical average case: It generally happens when the item is around the middle of the array. (N + 1) / 2 iteration is necessary thus its time complexity is O(N). It is likely to be the scenario that we named as B in our discussion.

Theoretical best case: When the item is in the first place, the best case occurs. Its time complexity is simply O(1). It is likely to be the scenario that we named as A in our discussion.

In unsuccessful search:

The worst, average, and best case are same when the search is unsuccessful, that is element is not in the array. N iteration is made therefore, its time complexity is O(N).

In general, we expect a linearly increasing graph in this algorithm. In theory, the graph should be like plot below. (Running times are arbitrary to show the growth rate of the cases).



If we compare our results with theoretical plot, we see that they are essentially consistent, especially for scenarios b, c, d. Running times are increasing linearly as the array size increases as we expected. For scenario a (best case), a constant line is expected however, we searched for around first 10% element (like index 10 in 100 element sized array, not the first item), to make it proportional to other arrays. That's why, we see a small increase as n increases. But otherwise, we see that running times increase with same factor with N, for instance we see 2 time increase in running time as N goes 5000 to 10000, which agrees theoretical result.

Algorithm 2 – Linear Search (Recursive)

In successful search:

Theoretical worst case: Similarly, worst case occurs when the item is in the last place of the array. The function calls itself N times, thus its time complexity is O(N). It is likely to be the scenario that we named as C in our discussion.

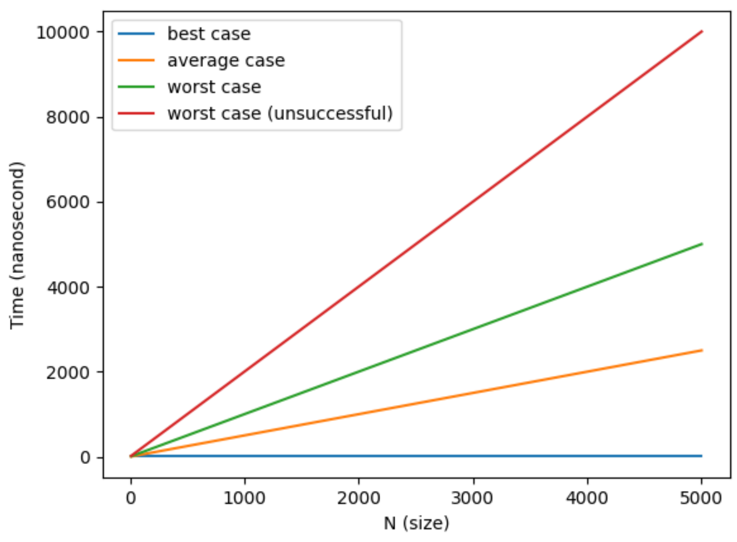
Theoretical average case: On average the function call itself (N+1) / 2 times. Generally, it happens when element is in the middle of the array. The time complexity of this scenario is O(N). It is likely to be the scenario that we named as B in our discussion.

Theoretical best case: When the item is in the first place, the best case occurs. It needs only one call to find the item, so its time complexity is simply O(1). It is likely to be the scenario that we named as A in our discussion.

In unsuccessful search:

The worst, average, and best case are same when the search is unsuccessful, that is element is not in the array. The function is called N times therefore, its time complexity is O(N).

In general, we expect a linearly increasing graph in this algorithm. In theory, the graph should be like this:



If we compare our results with theoretical plot, we see that they mostly agree. Running times are increasing linearly as the array size increases as we expected. Again, we see a small increase in scenario a (constant line is expected), however as we stated above, because we didn't use the very first index, we didn't get a constant line.

Algorithm 3 – Binary Search

In successful search:

Theoretical worst case: The worst case is when the item searched is at the first or last place of the array. As algorithm divides array in two in every iteration, assuming k iterations are made, approximately we have:

$N = 2^k$ or $k = \log_2 n$

So, time complexity of binary search is O(logn) in the worst case scenario.

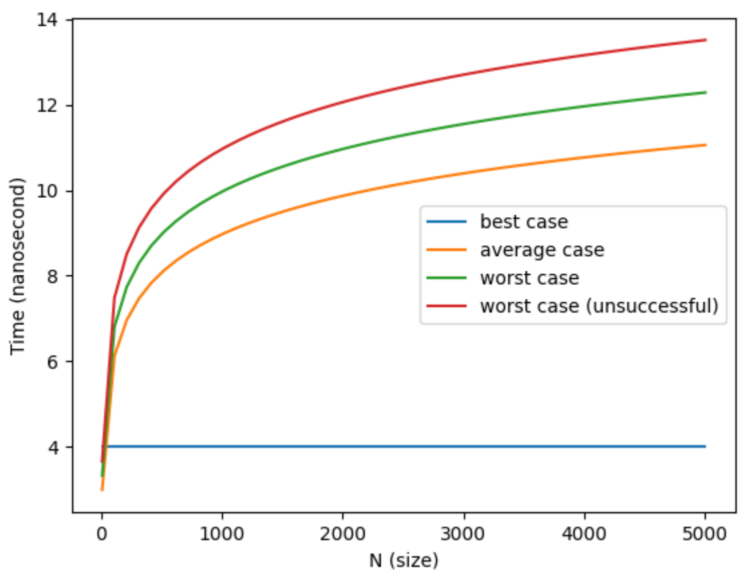
Theoretical average case: The upper bound for average case is logn, that is, its time complexity is O(logn) for this case.

Theoretical best case: The best case is when the item is in the middle of the array. Item is found in only one iteration, so its time complexity is O(1) for the best case.

In unsuccessful search:

The best, average, and worst case are the same in the unsuccessful search, when element is not found. Time complexity is O(logn) for this scenario. We examined this in scenario d, in our discussion.

We also may consider scenario a and c as worst case (successful) and b as best case, however, as the positions searched changes for different arrays, cases may change as well.



When we compare our result with the theoretical plot, we can see some inconsistencies. It is probably because the position of the item searched is arbitrarily chosen for scenario a, b, and c and it may cause a shift in running time. However, if we look at scenario d, which is the case that item searched doesn't exist in the array, we can see that it is more consistent. Furthermore, because the position chosen for scenario b is not exactly the middle of the array (around first 40% of the array), it doesn't show the best case scenario.

Algorithm 4 – Jump search

In successful search:

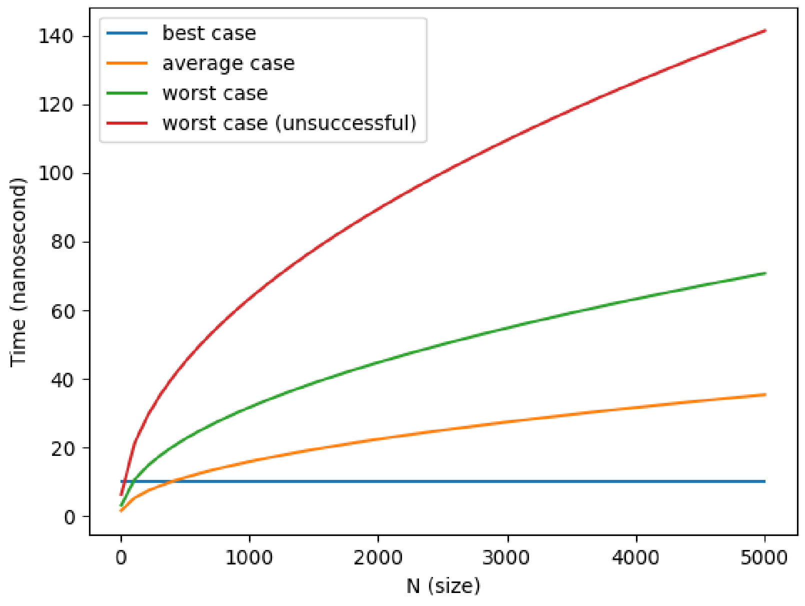
Theoretical worst case: The worst case is when the item is in the last place of the array. (N + m)/ m iterations are necessary in this algorithm. If we take $m = \sqrt{n}$, the time complexity we get is O($\sqrt{n}$) for the worst case. We examined this in scenario C.

Theoretical average case: The algorithm works in O($\sqrt{n}$) time, for the average case. This is when the item is around the middle of the array. We examined this in scenario B.

Theoretical best case: Best case is when the item is in the first place of the array. Time complexity of this scenario is O(1). We examined this in scenario A.

In unsuccessful search:

When element is not in the array, algorithm makes $\sqrt{n}$ iteration, that is it works in O($\sqrt{n}$). We examined this in scenario D.



If we compare our result with the expected result, it can be seen that they are similar. The graph of jump search grows slower than linear search, but faster than binary search, as we expected. For example, we see that as n increases from 1000 to 4000 (increases 4 times), we see roughly 2 time increase in the lines, which agrees expected result. Similar to other algorithms, as we didn't search for the first element in scenario a, we didn't get a constant line for that case.

Note:

Max size for array is made N = 10000 because higher sized arrays caused stack overflow.