PART 4: ANALYSIS AND CONCLUSION

Table:

Search Algorithms Analysis worksheet

		Linear	Binary	Ternary	Exponential	Interpolation	Jump
Target Set	Search data	Time in Milliseconds					
100	23	0.0127	0.0101	0.0261	0.0691	0.0558	0.378
	34	0.0172	0.0043	0.0221	0.0079	0.0083	0.0235
	68	0.0178	0.0101	0.0052	0.0154	0.0084	0.0016
	80	0.0403	0.0049	0.0047	0.0086	0.0084	0.009
	99	0.0255	0.0074	0.0063	0.0131	0.0283	0.0093
	AVERAGE:	0.0227	0.00736	0.01288	0.02282	0.02184	0.08428
1000	12	0.0588	0.0132	0.0118	0.023	0.0059	0.0881
	34	0.0068	0.0047	0.0092	0.0182	0.0071	0.0369
	566	0.0649	0.013	0.0117	0.0254	0.0077	0.085
	899	0.0567	0.01	0.0194	0.0117	0.0086	0.0224
	987	0.0749	0.0195	0.0212	0.0346	0.0306	0.041
	AVERAGE:	0.05242	0.01208	0.01466	0.02258	0.01198	0.05468
10000	100	0.0406	0.0145	0.1525	0.0346	0.0059	0.0246
	3000	0.2204	0.0083	0.0301	0.0381	0.0164	0.1048
	6000	0.2099	0.0089	0.0155	0.0619	0.0058	0.0956
	7666	0.6306	0.0159	0.0615	0.0265	0.0258	0.0677
	9877	0.5129	0.0183	0.0427	0.0491	0.0082	0.0649
	AVERAGE:	0.32288	0.01318	0.06046	0.04204	0.01242	0.07152

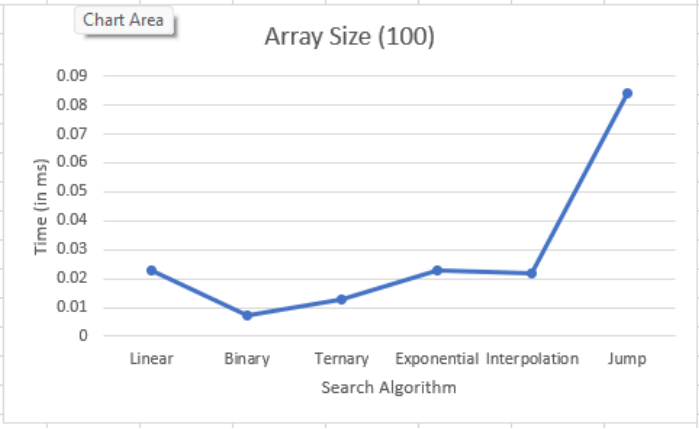
Average Time with respect to Array Size

	Linear	Binary	Ternary	Exponential	Interpolation	Jump
Array Size	Average Time in Milliseconds					
100	0.0227	0.0074	0.0129	0.0228	0.0218	0.0843
1000	0.0524	0.0121	0.0147	0.0226	0.012	0.0547
10000	0.3229	0.0132	0.0605	0.042	0.0124	0.0715

Averate time with respect to Index Position

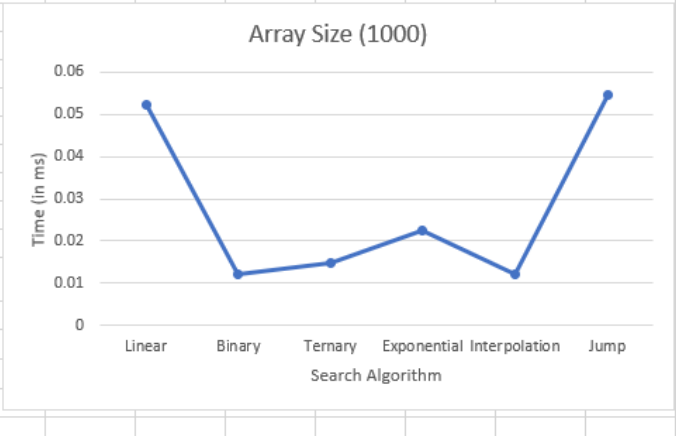
ndex Positio	Array Size			Linear	Binary	Ternary	Exponential	Interpolation	Jump
	100	1000	10000	Average Time in Milliseconds					
Start	23	12	100	0.0374	0.0126	0.0635	0.0422	0.0225	0.1636
Q1	34	34	3000	0.0815	0.0058	0.0205	0.0214	0.0106	0.0551
Middle	68	566	6000	0.0975	0.0107	0.0108	0.0342	0.0073	0.0607
Q3	80	899	7666	0.2425	0.0103	0.0285	0.0156	0.0143	0.033
End	99	987	9877	0.2044	0.0151	0.0234	0.0323	0.0224	0.0384

Graphs (Array Size):

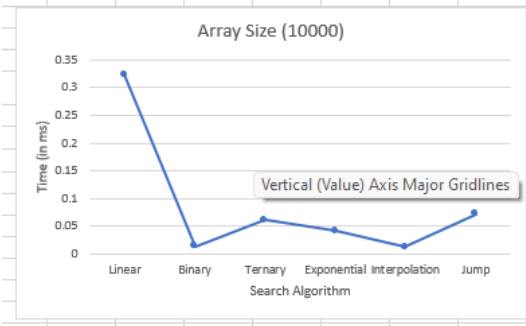


IN TERMS OF ARRAY SIZE

For Array Size (100):
- Fastest execution time is Binary Search
- Slowest execution time is Jump Search



For Array Size (1000):
- Fastest execution time is Interpolation Search
- Slowest execution time is Jump Search



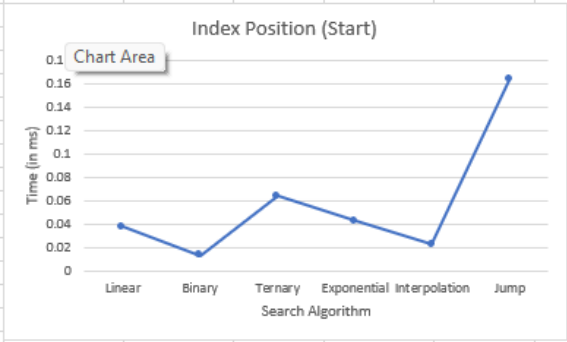
For Array Size (10000):
- Fastest execution time is Interpolation Search
- Slowest execution time is Linear Search

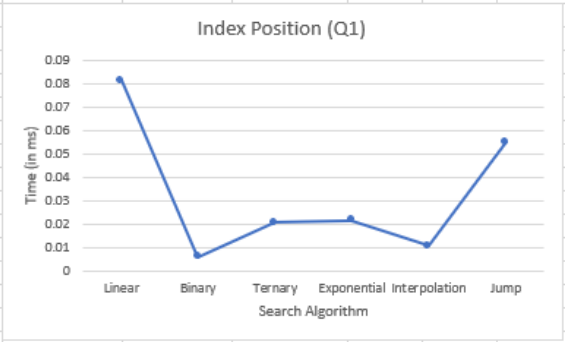
IN TERMS OF ARRAY SIZE

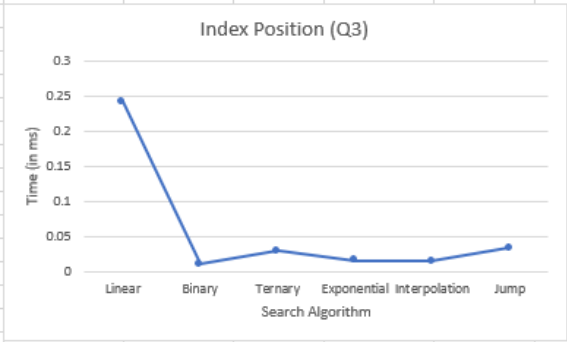
IN AVERAGE, INTERPOLATION SEARCH IS THE FASTEST ALGORITHM, SECOND IS BINARY SEARCH.
IN AVERAGE, JUMP SEARCH IS THE SLOWEST ALGORITHM, SECOND IS LINEAR SEARCH.

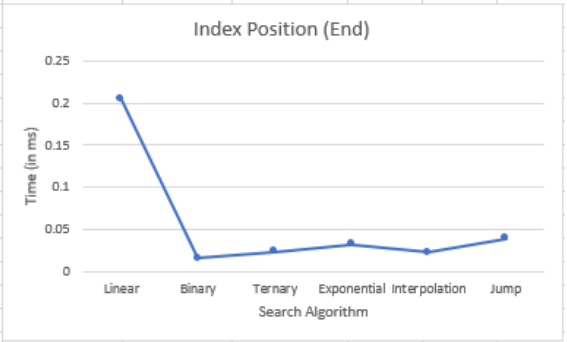
IT IS IMPORTANT TO CONSIDER THE SIZE OF YOUR DATASET IN ORDER TO KNOW THE BEST ALGORITHM TO USE

Graphs (Index Position):



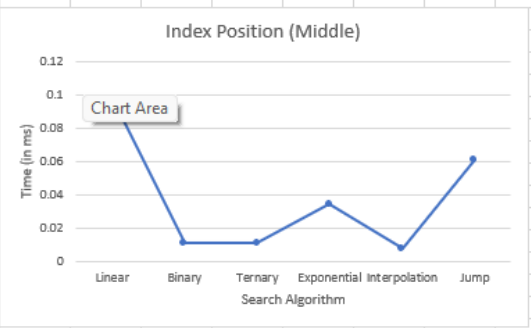






IN TERMS OF INDEX POSITION

For Index Position (Start):	For Index Position (Q1):	For Index Position (Middle):
• Fastest execution time is Binary Search	• Fastest execution time is Binary Search	• Fastest execution time is Interpolation Search
• Slowest execution time is Jump Search	• Slowest execution time is Linear Search	• Slowest execution time is Linear Search



For Index Position (Q3):	For Index Position (End):
• Fastest execution time is Binary Search	• Fastest execution time is Binary Search
• Slowest execution time is Linear Search	• Slowest execution time is Linear Search

IN TERMS OF INDEX POSITION

IN AVERAGE, BINARY SEARCH IS THE FASTEST ALGORITHM, SECOND IS INTERPOLATION SEARCH.
IN AVERAGE, LINEAR SEARCH IS THE SLOWEST ALGORITHM, SECOND IS JUMP SEARCH.

a. Which search algorithm performed the best overall?

After analyzing our table and graphs, we found out that in terms of array size, Interpolation Search performed best overall since it has a unique and faster way of finding numbers in a list that makes it really shine. On the other hand, in terms of index position, Binary Search performed the best.

However, since our focus is our array size and data sets which is sorted uniformly, Interpolation Search performed the best overall since it locates the exact position using interpolation formula; and as it tries to find exact location every time, so the searching time is reduced greatly. Since our data is sorted uniformly and evenly, this search algorithm is super handy because it's so good at making clever guesses using the nearby numbers.

Another cool thing about Interpolation Search is how fast it can be, especially when the numbers follow a uniform pattern. It is like having a shortcut in a game that helps you finish levels quicker. When the numbers are evenly spread out, Interpolation Search can move through the list faster, finding what we are looking for in fewer steps. But here is the catch: if the numbers are not evenly spread, this method might get confused and struggle a bit. Sometimes it might go too far or not far enough in finding the right number, like guessing too high or too low. But, for now, let us just disregard that fact considering that our data set is evenly spread and followed a uniform pattern.

b. Did any search algorithms perform better on specific data sets?

In sets of numbers up to 100, the straightforward way of Linear Search works pretty well. Checking each number one by one isn't too hard when there are only 100 numbers to look at. But Binary Search might not be as great here because it needs the numbers to be in order, and sorting such a small bunch might not be worth the hassle.

As the numbers go up to 1000, Binary Search starts to shine. It's really good with a bunch this size because it's faster, especially if the numbers are in order. Another method, Interpolation Search, also starts doing well here, especially if the numbers are spread out evenly. It's good at guessing where a number might be, needing fewer guesses than Binary Search in certain cases.

When the numbers reach up to 10000, Binary Search still works great, especially if they're in order. But there's also Exponential Search, which is good at quickly figuring out where to look next. Interpolation Search stays strong too, especially if the numbers are evenly spread out. It's pretty good at guessing where a number might be hiding based on nearby ones, needing fewer steps to find it.

So, these ways of searching for numbers work differently depending on how many numbers there are, whether they're sorted, and how they're spread out. Linear Search is good for smaller bunches, while Binary Search, Exponential Search, and Interpolation Search do better with bigger ones, each having their own strengths in certain situations.

c. How did the size of the data set affect the performance of the search algorithms?

The dataset size mattered a lot for the search algorithms. It's important to consider the size of your data set for us to know which search algorithm is best to use. Smaller sets, like up to 100 numbers, suited simpler methods like Linear Search. But as the dataset grew to 1000 numbers, Binary Search stood out for quickly finding sorted numbers. Meanwhile, Interpolation Search worked well with evenly spread numbers, needing fewer guesses.

In larger sets, like up to 10000 numbers, certain algorithms stood out. Binary Search remained efficient with its sorted list advantage. Also, Exponential and Interpolation Searches managed these larger sets effectively. Exponential Search narrowed down the search fast, while Interpolation Search's adaptability reduced steps needed.

All in all, as the dataset size increased, more sophisticated algorithms, especially those with logarithmic time complexities and adaptability to different distributions, displayed better efficiency and reduced search times compared to simpler methods. The performance of the search algorithms was notably influenced by the dataset size, showcasing the strengths of various algorithms in handling datasets of different magnitudes.

d. Write a brief conclusion summarizing your findings

Interpolation Search proved to be the standout performer among the search algorithms analyzed in terms of array size, while in terms of index position the Binary Search Stood out. Its unique approach using an interpolation formula to precisely locate numbers in a list, especially in uniformly distributed data, significantly reduced search times compared to Binary Search. While Linear Search suited smaller datasets up to 100 numbers, Binary Search excelled with sorted data up to 1000. As the dataset reached 10000, Binary Search maintained efficiency with sorted lists, while Exponential and Interpolation Searches effectively managed larger sets. Furthermore, we also found out that jump search and linear search are slower compared to other search algorithms.

In essence, the dataset size greatly impacted algorithm performance. It's important to consider the size of your data set for us to know which search algorithm is best to use. Smaller datasets favored simpler methods, while larger datasets showcased the efficiency of algorithms with logarithmic time complexities and adaptability. This analysis highlighted the diverse strengths of various algorithms in handling datasets of varying sizes, emphasizing their adaptability and efficiency in different scenarios.