Najera Ericka
CS 2401
9 November 2018

Linear vs Binary Time Complexity

Finding the time complexity of the both linear search and binary search was interesting. I created both methods as iterative instead of recursion. I did iterative instead because since I was doing big double arrays, I didn't want my computer to overrun because of the extra storage it would use. No make the arrays fair I created each element with the Math.random and then sorted the array. After that I created two methods that would call the linear search and binary search separately and would record the time in Nano time. I used Nano instead because when I first ran the program it would give me zero for the time.

In the linear search I created a for loop to run through each element of the array and check if that number was the searchee. By the for loop I determined the time complexity to be O(n). The binary search gets the first index and last to determine the middle element splitting it in two arrays. It checks the middle element, then if it is that number is returns that index. If not, it checks if it is less than that value making that middle element the "last" and getting a new middle from that sub array. If it is greater than the middle then the first element is now the middle and it creates a new middle as well. It keeps dividing and going left or right until the number is found. Since it divided the array in two, the time complexity is O(logn).

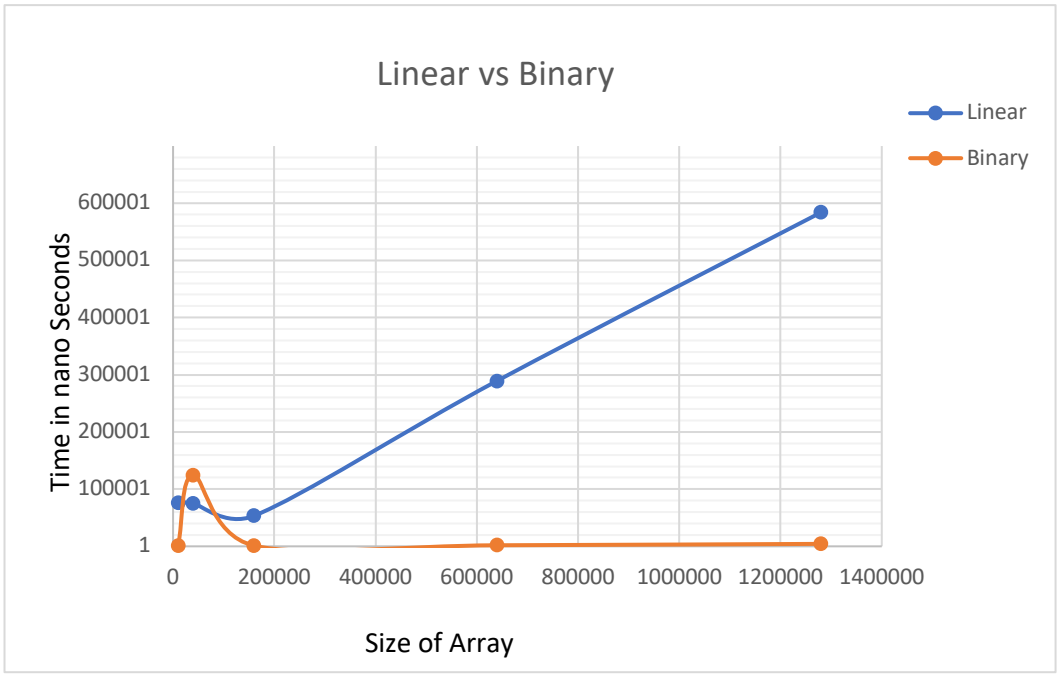
In order to get the time it took to run each algorithm, I created two separate methods called linearAverage and binaryAverage. For both algorithms, I saved the current Nano time then I called the searching algorithm to search for a random element. Then I saved the current time again. To get the average I subtracted the final time minus the initial time. Then, I created

another method that contained a for loop set to run 30 times. In each loop it creates a different a random index to be found and looks for that index in both linear and binary search. I did this, so it would have variety in the different indexes and not just find the same index for 30 times like I did at first.

I am using a MacBook Pro with a 2.3 GHz Intel Core i5 processor, memory of 8 GB 2133 MHz LPDDR3, along with a macOS High Sierra version 10.13.6 (17G65). I am currently running Java Version 10.0.2. As of right now I need some updates on Java and computer software updates.

Some factors that could have influenced my accuracy of my results would be I had various tabs open such as text files and internet tabs open. One thing I can't manage is if the random index was given with the worst cases and increasing my time complexity. Overall, binary search was faster except in the second text of 40,000 length array. The big O notation of linear search is O(n) and binary search is O(logn). According to the graph in the book of the time complexities my graph does look like O(n) and O(logn). Somehow due to the size of the arrays, the smaller the array was the faster it was for linear and as the array got bigger it was faster with binary.

Array Size	Linear	Binary
10000	76545	1535
40000	75321	124099
160000	53374	1198
640000	289362	1855
1280000	583913	4290


Linear vs Binary
Linear
Binary
Time in nano Seconds
600001
500001
400001
300001
200001
100001
1
0
200000
400000
600000
800000
1000000
1200000
1400000
Size of Array