UNIVERSITY INSTITUTE OF ENGINEERING & TECHNOLOGY

PANJAB UNIVERSITY, CHANDIGARH

ANALYSIS AND DESIGN OF ALGORITHMS

Name: Charanjit Singh(UE163031)

Bachelor of Computer Science and Engineering [4th Semester]

25 Jan, 2018.

RUNTIME ANALYSIS OF LINEAR SEARCH

Linear Search is a kind of search algorithm, which performs ‘search’ in simplest method available, i.e. by comparing the key one by one with all the members of the provided list.

To implement linear search on a provided list with a key, we simply set a loop which traverses and compares the list members one by one until the key is found or we reach at the end of the list. Therefore the time complexity of Linear search is of order of: O(n) (Worst Case) i.e. the time required for searching an element in list will not exceed the order of ‘n’.(or say it cannot be ‘n^2’ or greater).

We’ll analyse the Linear search algorithm by increasing number of inputs(n) in the list and then searching a number there. To accomplish this mission, we have to take care of these assumptions:

- CPU is single core.
- CPU is single threaded.
- CPU is taking care of only search process, and there is no other process that is disturbing it.
- CPU generates completely random numbers when needed.
- Each and every comparison is taking same amount of processing effort and hence time.

Methodology:

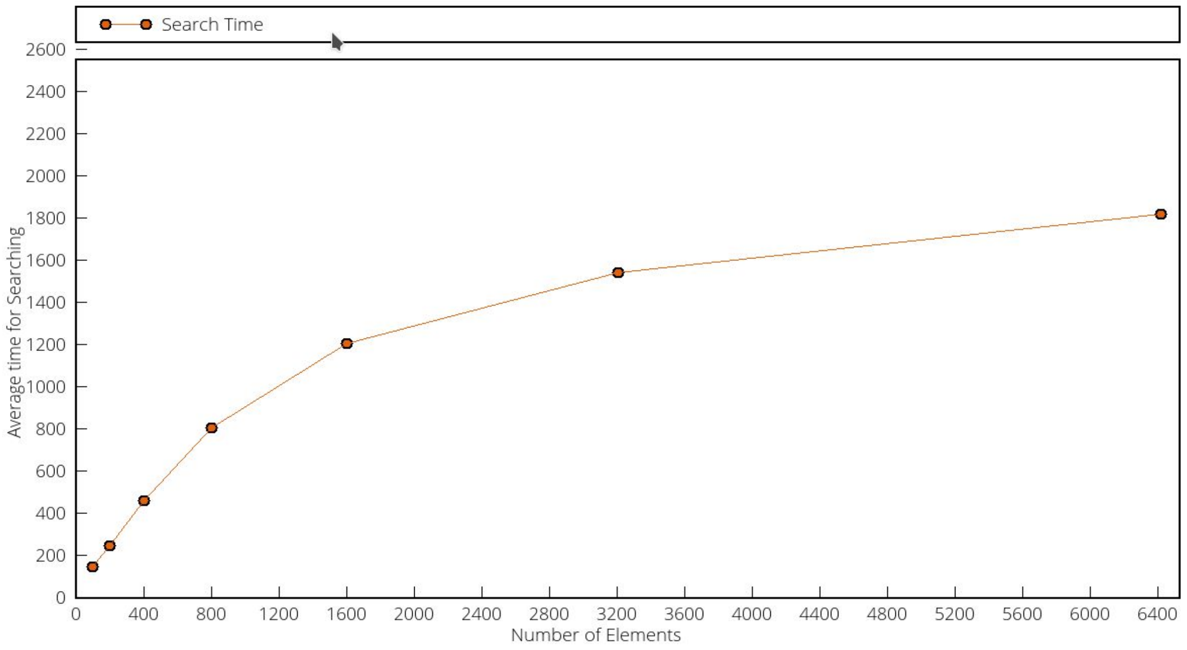
The steps I took during the course of this analysis were:

1. Initialise the list of (n) integers with some random data.
2. Store the CPU time at this instant.
3. Call Linear_Search(List(n),some_randomly_generated_key)
4. Now again Store the CPU time at this instant.
5. Note down the CPU time taken for Searching a key in list of (n) numbers, by subtracting data at step 4 and step 2.
6. Perform all these steps at least (m) times.
7. Take average of the total time consumed and (m).
8. Now increase the (n) and repeat all these steps again.
9. Plot the graph between Time consumed and (n).

I took (m)=1500 and (n)= 100, 200, 400, 800, 1600, 3200 and 6400.
The data structure I used as a list is a simple array. I have programmed this analytical program in ‘C++’.

Here’s the data I got from the analysis on my machine.
{ 4 X Intel® Core i5-5200 CPU @2.20 GHz 3.8 GB Main Memory OS:Linux x64 Debian }

Experiment I:



graph (a)

[100] B: 4 | A: 96 | W: 1400 | TC: 144829	[n] B: Best Case
[200] B: 2 | A: 180 | W: 1318 | TC: 282106	A: Average Case
[400] B: 2 | A: 361 | W: 1137 | TC: 524646	W: Worst Case
[800] B: 3 | A: 622 | W: 875 | TC: 922606	TC: Total Comparisons made
[1600] B: 1 | A: 963 | W: 536 | TC: 1511927	m = 1500
[3200] B: 2 | A: 1337 | W: 161 | TC: 1900436	
[6400] B: 1 | A: 1486 | W: 13 | TC: 217431	

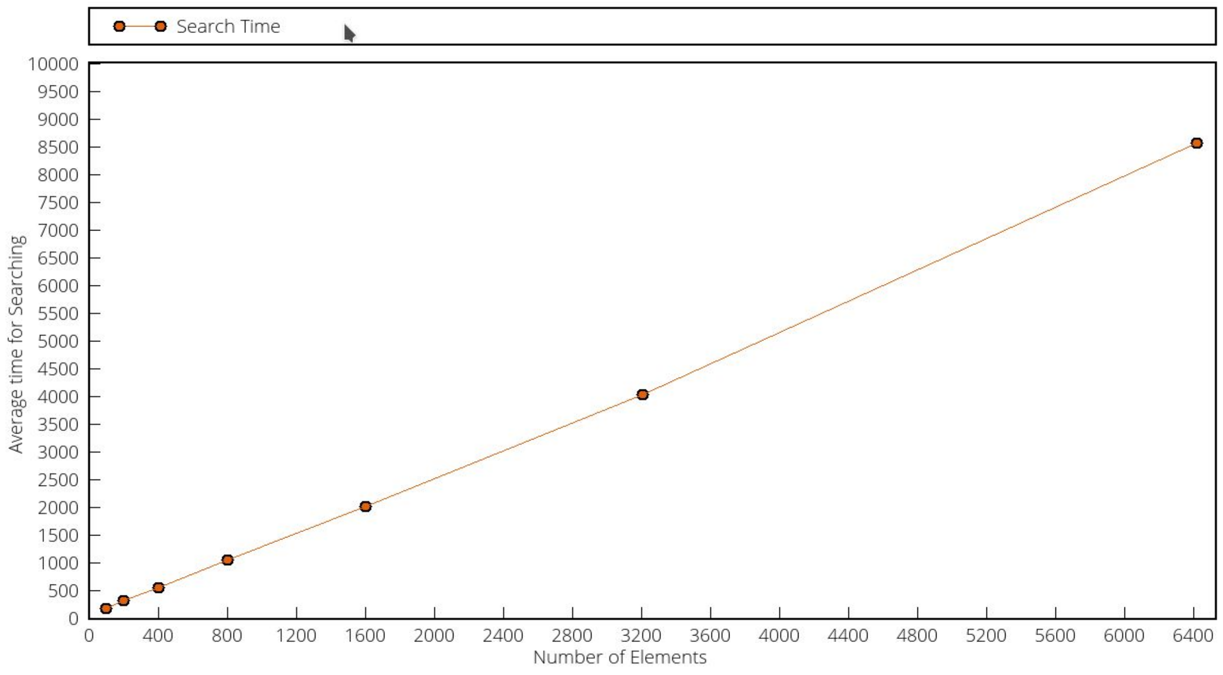
Table (a)

As we can see in the graph(a) that the graph is sloping upwards, which is showing us that the time required to search a key in a list increases directly with the number of elements in the list. The TC(Total Comparisons Made) explains all the ups and downs of the graph. As it shows the total number of comparisons made for searching a random key in a list of (n) random numbers, (m) times. Another point to note down here is that the worst cases of (n*2) takes 2X more time if we compare it with worst cases of list of (n) numbers, Whereas, the best cases take same time, irrespective of (n) as it has to make only 1 comparison.
Greater the number of average cases and worst cases, more the ‘search’ procedure will take time, and it is clear from graph(a) and table(a). I have made this observation on an multicore,

multiprocessing and multithreading system, which again can cause a significant effect on the readings.

Experiment II:

Let’s make the observations more meaningful by setting all the keys, in such a way that they are not present in the list. Now the CPU will traverse all the members of the list, for all the values of (n). And thus, will treat all the (n)s in same conditions.



graph(b)

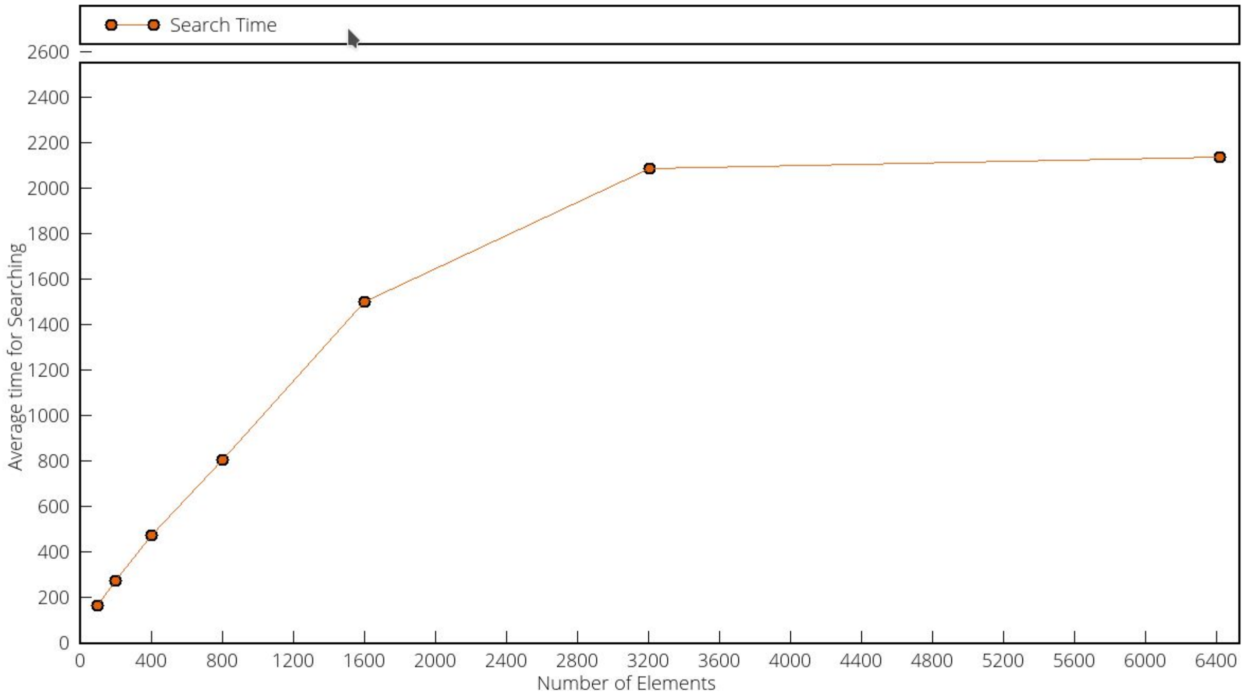
[100] B: 0 | A: 0 | W: 1500 | TC: 150000	[n] B: Best Case
[200] B: 0 | A: 0 | W: 1500 | TC: 300000	A: Average Case
[400] B: 0 | A: 0 | W: 1500 | TC: 600000	W: Worst Case
[800] B: 0 | A: 0 | W: 1500 | TC: 1200000	TC: Total Comparisons made
[1600] B: 0 | A: 0 | W: 1500 | TC: 2400000	m = 1500
[3200] B: 0 | A: 0 | W: 1500 | TC: 4800000	
[6400] B: 0 | A: 0 | W: 1500 | TC: 9600000	

table(b)

Now, the things might be clear that using this graph that the time complexity of the linear search is of order of O(n) as we can see from graph(b) ‘s Y-axis and compare it with Y-axis of graph(a), we can see the differences and maximum values if graph(b) are much greater than graph(a) because in this experiment we can see that there are all the worst cases, i.e. we are traversing the whole list every time and taking more time. Total number of comparisons made explains the graph. Even in this experiment, we can see the slight deviation at (n)=3200, this is obviously due to multiprocessing and multicore system architecture.This experiment gives the peak values of the linear search algorithm. No value can exceed the values observed by this experiment until and unless the CPU is not multiprocessing and multicore.

Experiment III:

Let's have a look at another experiment:



graph (c)

[100] B: 3 | A: 93 | W: 1404 | TC: 144430 [200] B: 5 | A: 195 | W: 1300 | TC: 278695 [400] B: 2 | A: 362 | W: 1136 | TC: 522710 [800] B: 4 | A: 649 | W: 847 | TC: 915651 [1600] B: 3 | A: 1011 | W: 486 | TC: 1436511 [3200] B: 3 | A: 1327 | W: 170 | TC: 1983314 [6400] B: 1 | A: 1487 | W: 12 | TC: 2079354	[n] B: Best Case A: Average Case W: Worst Case TC: Total Comparisons made m = 1500

table(c)

Again, TC is giving explanation of the graph(c). The tables [(a) and (c)], I derived was from completely random data, whereas table(b) was forced to traverse all the elements of the lists. Thus, all the tables and graphs provided above and below are showing us behaviour of simple Linear search algorithm.

<u>Conclusion</u>:

Thus, coming to conclusion, we can say that, it is very irrelevant to compare the algorithms on the basis of average cases. It is the worst case, that describes the algorithm's efficiency and limits. Though average cases gives us the idea of the working of an algorithm in a real life scenario. So, from the above analysis, we conclude that:

- The time complexity of linear search algorithm is of order of **O**(n).
- The ups and downs in graphs which came during the experiments were because of the difference between the number of comparisons made.
- The best case of search algorithm has time complexity **Ω**(1).

X-X-X-X-X