Linear Search

A presentation by Aryan Singh, 14200121176, Department of Computer Science and Engineering

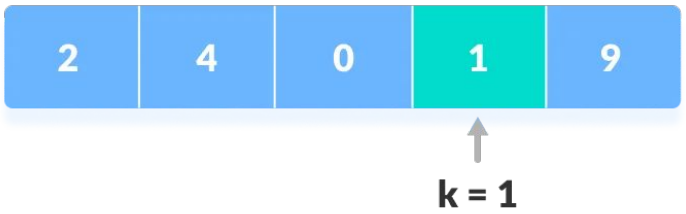
What is Linear Search and why should you care?

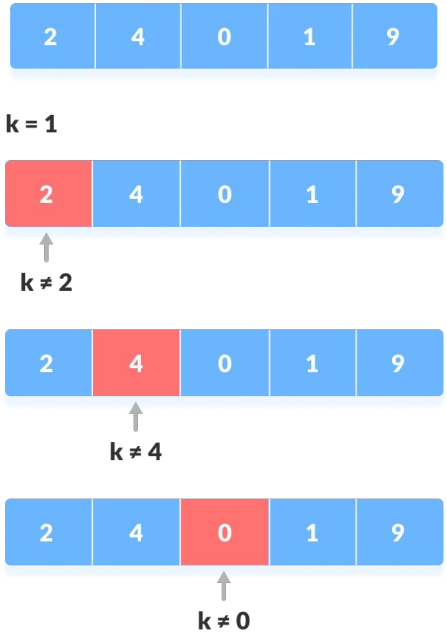
Some question are easier to answer than others, so let's begin with the easy one!

Is a method for finding an element within a list. It sequentially checks each element of the list until a match is found or the whole list has been searched.

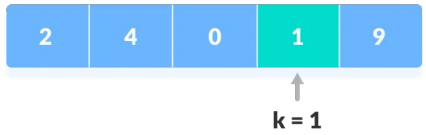
As to ***why you should care?***

One of the most straightforward and elementary searches is the sequential search. As a real world example, pickup the nearest phonebook and open it to the first page of names. It's something that we do many days of our lives. And we have been for a long time. And the reason for our adherence to it, is simplicity and effectiveness in small case problems. In a world of BIG data we do sometimes encounter problems that are small and that's where linear search shines brightest.





Say you want to find '1' here



You check each Index or box if you will in that array in a sequence till you find the one. (Pun intended!)

Diagrammatic Representation

Application and Analysis

When many values have to be searched in the same list, it often pays to pre-process the list in order to use a faster method. For example, one may sort the list and use binary search, or build an efficient search data structure from it. Should the content of the list change frequently, repeated re-organization may be more trouble than it is worth.

As a result, even though in theory other search algorithms may be faster than linear search (for instance binary search), in practice even on medium-sized arrays (around 100 items or less) it might be infeasible to use anything else. On larger arrays, it only makes sense to use other, faster search methods if the data is large enough, because the initial time to prepare (sort) the data is comparable to many linear searches

For a list with n items, the best case is when the value is equal to the first element of the list, in which case only one comparison is needed. The worst case is when the value is not in the list (or occurs only once at the end of the list), in which case n comparisons are needed.

If the value being sought occurs k times in the list, and all orderings of the list are equally likely, the expected number of comparisons.

For example, if the value being sought occurs once in the list, and all orderings of the list are equally likely, the expected number of comparisons is

$$\begin{cases} n & \text{if } k = 0 \\ \frac{n+1}{k+1} & \text{if } 1 \le k \le n. \end{cases}$$

However, if it is *known* that it occurs once, then at most n - 1 comparisons are needed, and the expected number of comparisons is [{(n+2)(n-1)}/2n]

Algorithm and function for linear search in C

```
0: START

1: SET i to 0

2: IF i > n THEN GOTO step 7

3: IF arr[i] = a then GOTO step 6

4: SET i to i + 1

5: GOTO step 2

6: PRINT ‘element a found at index
{i}’ and GOTO step 8

7: PRINT ‘element not found’

8: END
```

```c
int l_s(int arr[], int n, int x) {

 for (int i = 0; i < n; i++)

    if (arr[i] == x)

        return i;

  return -1;

}
```

Thank you!

A presentation by Aryan Singh, 14200121176, Department of Computer Science and Engineering

A special thanks to Wikipedia, For the Analysis of the algorithm. Made using libreoffice.