NIELIT GORAKHPUR

Course Name: **A Level (2nd Sem)**
Topic: **Linear Search in C++**
Subject: **Data Structure using C++**
Date: **23-04-2020**

Linear search (known as sequential search) is an algorithm for finding a target value within a list. It sequentially checks each element of the list for the target value until a match is found or until all the elements have been searched. This is one of the most basic search algorithms and is directly, inspired by real-life events.

Steps involved in this algorithm are:

- **Step 1**: Select the **first element** as the **current element**.
- **Step 2**: Compare the **current element** with the **target element**. If matches, then go to *step 5*.
- **Step 3**: If there is a **next element**, then set **current element** to **next element** and go to *Step 2*.
- **Step 4**: **Target element** not found. Go to *Step 6*.
- **Step 5**: **Target element** found and return **location**.
- **Step 6**: Exit process.

Example

Consider the following sorted list in which we want to find 6:

```
-1 3 3 6 11 12 16 17 18 19
```

We chose to compare with the first element -1 with 6. As -1 != 6, we move on.
The potential numbers are:

```
3 3 6 11 12 16 17 18 19
```

We compare 6 with the second element 3. As 3 != 6, we remove 3.
The potential numbers are:

```
3 6 11 12 16 17 18 19
```

We compare our target 6 with the first (3rd) element 3. As 3 != 6, we move on.
The potential numbers are:

```
6 11 12 16 17 18 19
```

We compare our target 6 with the first (3rd) element 3. As 6 == 6, we found our target and terminate the search process.

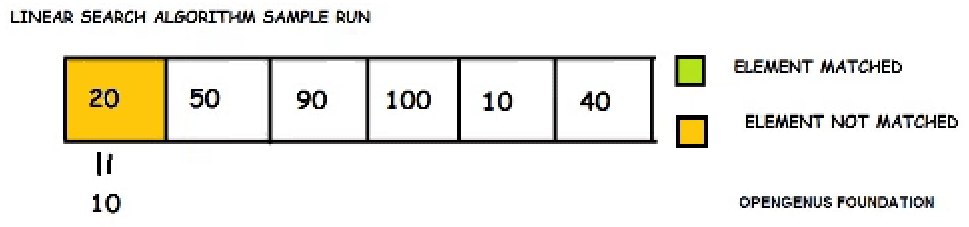
Complexity

- Worst case time complexity: **O(N)**
- Average case time complexity: **O(N)**
- Best case time complexity: **O(1)**
- Space complexity: **O(1)**

Linear search in average makes n/2 comparisons where n is the number of elements in the set. At the most, linear search algorithm takes n comparisons.

In terms of **implementation**, linear search algorithm takes 2n+1 comparisons (n to check if target element is found and n+1 comparisons to check if end of list is reached) in the worst case. With optimizations, we can make n+1 comparisons in the worst case.

Visual run



```
#include <stdio.h>
/*
 * Part of Cosmos by OpenGenus Foundation
 * Input: an integer array with size in index 0, the element to be searched
 * Output: if found, returns the index of the element else -1
*/
int search(int arr[], int size, int x)
{
   int i=0;
   for (i=0; i<size; i++)
       if (arr[i] == x)
           return i;
   return -1;
}
int main()
{
   int arr[] = {2,3,1,5}; // Index 0 stores the size of the array (initially 0)
   int size = sizeof(arr)/sizeof(arr[0]);
   int find = 1;
   printf("Position of %d is %d\n", find, search(arr,size,find));
   return 0;
}
```

Optimizations

The simple Linear search algorithm can be modified in small respects to get an optimized results:

Optimization 1: **Linear search with a sentinel**

The basic algorithm above makes *two comparisons per iteration* By adding an extra record w to the list (a sentinel value) that equals the target, the second comparison can be eliminated until the end of the search, making the algorithm faster. The search will reach the sentinel if the target is not contained within the list.

This optimization makes n+1 comparisons in the worst case.

Steps:

- **Step 1**: Set i to 0.
- **Step 2**: If Li = T, go to *step 4*.

- **Step 3**: Increase i by 1 and go to *step 2*.
- **Step 4**: If i < n, the search terminates successfully; return i else, the search terminates unsuccessfully.

Optimization 2: **Linear search in an ordered table**
If the list is ordered such that $L0 \leq L1 \ldots \leq Ln-1$, the search can establish the absence of the target more quickly by concluding the search once Li exceeds the target. This variation requires a sentinel that is greater than the target.

Steps:

- **Step 1**: Set i to 0.
- **Step 2**: If $Li \geq T$, go to *step 4*.
- **Step 3**: Increase i by 1 and go to *step 2*.
- **Step 4**: If Li = T, the search terminates successfully; return i else, the search terminates unsuccessfully.

Applications
Linked List is the preferred search algorithm in the following cases:

- List contains few elements (exact number depends on usage factors).
- List is unordered.
- If the content of the list change frequently, repeated re-organization can be an overload. In such a case, linear search is the preferred search algorithm.

Conclusion
In theory other search algorithms may be faster than linear search but in practice even on medium-sized arrays (around 120 items or less) it might be infeasible to use anything else. On larger arrays, it is advised to use faster search methods as if the data is large enough, the initial time to prepare the data is comparable to many linear searches.