**Linear Search**

In computer science, a search algorithm is any algorithm which solves the Search problem, namely, to retrieve information stored within some data structure, or calculated in the search space of a problem domain.

# Linear Search

Linear search or Sequential search is a method for finding a target value within a list.

Linear search sequentially checks each element of the list until it finds an element that matches the target value. If the algorithm reaches the end of the list, the search terminates unsuccessfully.

# Basic Algorithm

Given a list L of n elements with values or records L0 ... Ln−1, and target value T, the following subroutine uses linear search to find the index of the target T in L.[3]

1. Set i to 0.
2. If Li = T, the search terminates successfully; return i.
3. Increase i by 1.
4. If i < n, go to step 2. Otherwise, the search terminates unsuccessfully.

# Pseudocode

procedure linear\_search (list, value)

for each item in the list

if match item == value

return the item's location

end if

end for

end procedure

# C/C++ Implementation

// Linearly search x in arr[]. If x is present then return its location, otherwise return -1

int search(int arr[], int n, int x) {

int i;

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

if (arr[i] == x)

return i;

return -1;

}

## [Linear search for Unsorted list](http://nptel.ac.in/courses/106103069/Module_6/linear_search_unsorted.htm)

linear search(int x[],int n,int key)

{

int i,flag = 0;

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

if(x[i]==key) {

flag=1;

break;

}

}

if(flag==0)

return(-1);

else

return(1);

}

## [Linear search for sorted list](http://nptel.ac.in/courses/106103069/Module_6/linear_search_sorted1.htm)

Now in this case, the basic algorithm remains the same as we have done in case of an unsorted array but the only difference is we do not have to search the entire array every time. Whenever we encounter an element say y greater than the key to be searched, we conclude that there is no such element which is equal to the key, in the list. This is because all the elements in the list are in ascending order and all elements to the right of y will be greater or equal to y, ie greater than the key. So there is no point in continuing the search even if the end of the list has not been reached and the required element has not been found.

Linear search( int x[], int n, int key)

{

int i, flag=0;

for(i=0; i < n && x[i] <= key; i++) {

if(x[i]==key) {

flag=1;

break;

}

}

if(flag==1) /\* Unsuccessful Search\*/

return(-1);

else

return(1); /\*Successful search\*/

}

# Complexity

**Best case**: when the search term is in the first slot in the array. The number of comparisons in this case is 1.

**Average case**: when the search term is somewhere in the middle of the array. The number of comparisons in this case is approximately N/2.

**Worst case**: when the search term is in the last slot in the array, or is not in the array. The number of comparisons in this case is equal to the size of the array N.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Algorithm** | **Best case** | **Average case** | **Worst case** | **Space Complexity** |
| Linear Search | O(1) | O(n) | O(n) | O(1) |
| Binary Search | O(1) | O(logn) | O(logn) | O(1)\* |
| Jump Search | O(1) | O(√ n) | O(√ n) | O(1) |
| Interpolation Search | O(1) | O (log log n)) | O(n) | O(1) |
| Exponential Search | O(1) | O(log i) | O(log i) | O(1) |
| Fibonacci Search | O(1) | O(logn) | O(logn) | O(1) |

# Application

1. Practical when the list has only a few elements, or when performing a single search in an unordered list
2. 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.
3. Even though in theory other search algorithms may be faster than linear search, in practice even on medium-sized arrays (around 100 items or less) it might be infeasible to use anything else
4. Linear search is rarely used practically because other search algorithms such as the binary search algorithm and hash tables allow significantly faster searching comparison to Linear search

# Example:

#include<stdio.h>

int lineaSearch(int arr\_arg[], int n, int x)

{

int i;

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

{

if (arr\_arg[i] == x)

return i;

}

return -1;

}

int main(void)

{

int arr[] = {2, 3, 4, 10, 40};

int size = 0, sElement = 0, rindex = 0;

fprintf(stdout, "Enter the element to be search: ");

scanf("%d",&sElement);

size = sizeof(arr)/ sizeof(arr[0]);

rindex = lineaSearch(arr, size, sElement);

if(-1 == rindex)

fprintf(stdout, "Element not found in the array");

else

fprintf(stdout, "Element found at index : %d, element is : %d", rindex, arr[rindex]);

return 0;

}

Output:

Enter the element to be search: 4

Element found at index : 2, element is : 4

# References

<https://www.geeksforgeeks.org/searching-algorithms/>

<https://www.geeksforgeeks.org/linear-search/>