


# Theory: Linear search in Java

 15 minutes

0 / 4 problems solved

Skip this topic

Start practicing

1593 users solved this topic. Latest completion was about 6 hours ago.

**Linear (sequential) search** is a simple algorithm for searching a value in arrays or similar collections. The algorithm checks each element of an array until it finds one that matches the target value. If the algorithm reaches the end of an array, it terminates unsuccessfully. In the worst case, it performs exactly  $n$  comparisons where  $n$  is the length of the input array containing data.

This algorithm can be also modified to check if an array contains a target element, to search for the first/last occurrence of an element, to search a value in a certain range of indexes, or counting all occurrences of a target element.

## §1. Implementation in Java

Let's consider an implementation of the algorithm in Java as a regular static method. The method takes an array of int's and a number to search in the array. It returns the index of the found element or, if nothing was found, `-1`.

```
1 public static int search(int[] array, int value) {
2     int index = -1;
3     for (int i = 0; i < array.length; i++) {
4         if (array[i] == value) {
5             index = i;
6             break;
7         }
8     }
9     return index;
10 }
```

The implementation compares each element of an input array with the passed value in a loop. If an element is equal to the value, the loop stops and the method returns the found index. If no value is found in the array, the method returns `-1`.

Let's call the method passing different arrays and values:

```
1 int[] numbers = {1, 4, 7, 2, 3, 5};
2
3 search(numbers, 1); // 0
4 search(numbers, 2); // 3
5 search(numbers, 3); // 4
6 search(numbers, 4); // 1
7 search(numbers, 5); // 5
8 search(numbers, 6); // -1, no value found
```

In the last sample, the algorithm returns `-1` after it has checked all elements of the array and found none that equals `6`.

## §2. Searching in sorted arrays

Note that this algorithm works both for **sorted** and **unsorted** arrays. We can make the algorithm more efficient at searching for an element in a sorted array; here is a modified version of the algorithm:

Current topic:

[Linear search in Java](#) ...

Topic depends on:

✓ [Linear search](#) ...

✗ [Algorithms in Java](#) ...

Topic is required for:

[Binary search in Java](#) ...

Table of contents:

[1 Linear search in Java](#)

[§1. Implementation in Java](#)

[§2. Searching in sorted arrays](#)

[Feedback & Comments](#)

```
1 public static int searchInSortedArray(int[] array, int value) {
2     int index = -1;
3     for (int i = 0; i < array.length; i++) {
4         if (array[i] == value) {
5             index = i;
6             break;
7         } else if (array[i] > value) {
8             break;
9         }
10    }
11    return index;
12 }
```

If the next element checked is greater than the passed value, it means we will not find the value in the array and we can stop.

Below is an example of how this can be applied:

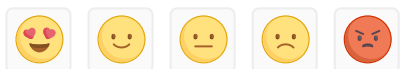
```
1 |
searchInSortedArray(new int[] {8, 15, 19, 20, 21}, 10); // -1, the element 10 is not found
```

The algorithm performs only two comparisons to confirm that the element is not contained in the array. Thus, we check a smaller number of elements when the passed value is not contained in the array and is smaller than the max element. Otherwise, the algorithm still makes  $n$  comparisons and the time complexity is  $O(n)$

There are more efficient algorithms for searching in sorted sequences – namely, **binary search**. But this is a whole other story.

 Report a typo

151 users liked this theory. 1 didn't like it. What about you?



Start practicing

[Comments \(0\)](#)

[Hints \(0\)](#)

[Useful links \(0\)](#)

[Show discussion](#)