Linear Search

Linear Search

Linear search is a straightforward algorithm used to find the position of a target element within a list. It checks each element in the list sequentially until it finds the target element or reaches the end of the list.

How Linear Search Works

1. **Start from the beginning:** Begin with the first element of the array or list.
2. **Check each element:** Compare the current element with the target element.
3. **Move to the next element:** If the current element does not match the target, move to the next element.
4. **Stop if found or at the end:** If a match is found, return the index of the element. If the end of the list is reached and no match is found, return an indication (e.g., -1) that the element is not present in the list.

Example

Suppose we have an array:

```
css
Copy code
arr = [5, 8, 3, 12, 15]
```

And we want to find the position of `12`.

1. Start with the first element (`5`) – does not match.
2. Move to the second element (`8`) – does not match.
3. Move to the third element (`3`) – does not match.
4. Move to the fourth element (`12`) – it matches.
5. Return the index of `12`, which is `3`.

Algorithm

Here's a step-by-step algorithm:

1. **Input:** An array `arr` of size `n`, and a target element `x`.
2. **For each element** `arr[i]` in `arr`:
    - If `arr[i] == x`, return `i`.
3. If the loop completes without finding `x`, return `1`.

Code Example (Java)

```
java
Copy code
public class LinearSearch {
    public static int linearSearch(int[] arr, int x) {
        for (int i = 0; i < arr.length; i++) {
            if (arr[i] == x) {
                return i;
            }
        }
        return -1;
    }

    public static void main(String[] args) {
        int[] arr = {5, 8, 3, 12, 15};
        int x = 12;
        int result = linearSearch(arr, x);
        if (result != -1) {
            System.out.println("Element found at index: " + r
esult);
        } else {
            System.out.println("Element not found");
        }
    }
```

```
}
```

Time Complexity

1. **Best Case (`O(1)`):** The element is found at the first position.
2. **Worst Case (`O(n)`):** The element is at the last position or not present in the list at all. In this case, the algorithm will iterate over all `n` elements.
3. **Average Case (`O(n)`):** On average, the algorithm may need to search through half of the list, but the complexity still scales linearly.