Data Structures and Algorithms

Lecture 5: Arrays – Searching

Searching an element in an array

The method of searching for a specific value in an array is known as searching.

Types of search algorithms:

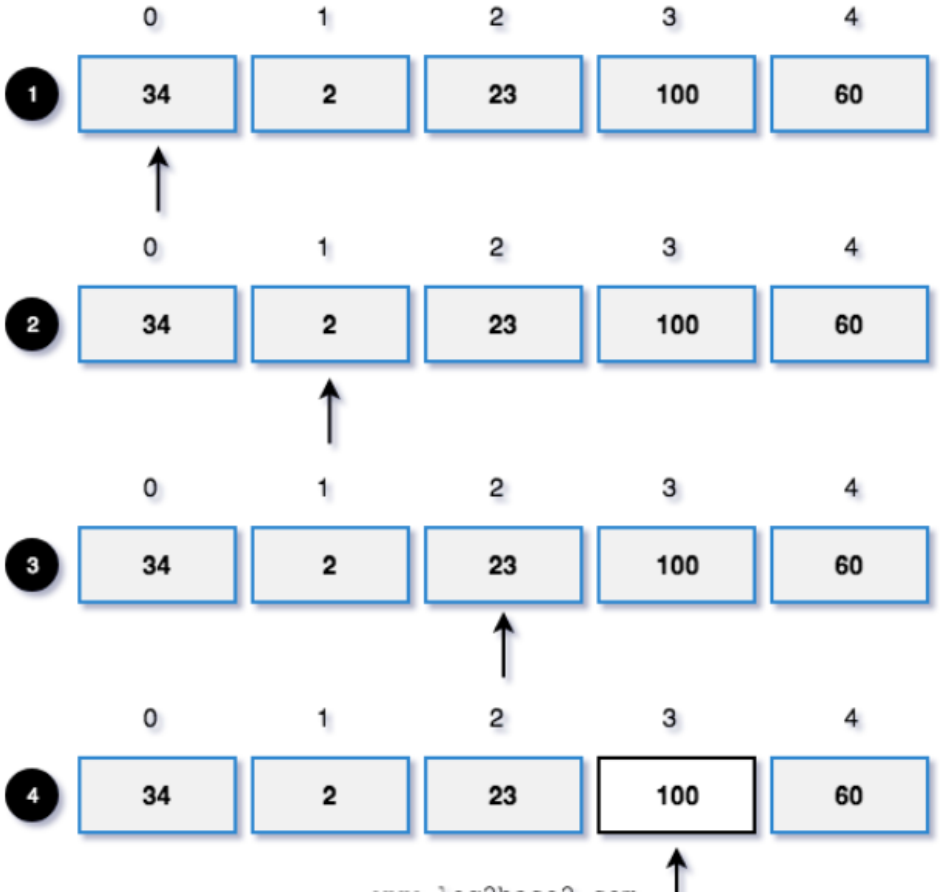
1. **Linear or Sequential Search:**
    - This algorithm works by sequentially iterating through the whole array or list from one end until the target element is found. If the element is found, it returns its index, else -1.
2. **Binary Search:**
    - This type of searching algorithm is used to find the position of a specific value contained in a sorted array. The binary search algorithm works on the principle of divide and conquer and it is considered the best searching algorithm because it's faster to run.

Linear Search

Let's take an array of 5 elements:

34, 2, 23, 100, 60.

Search element = 100



The execution will be:

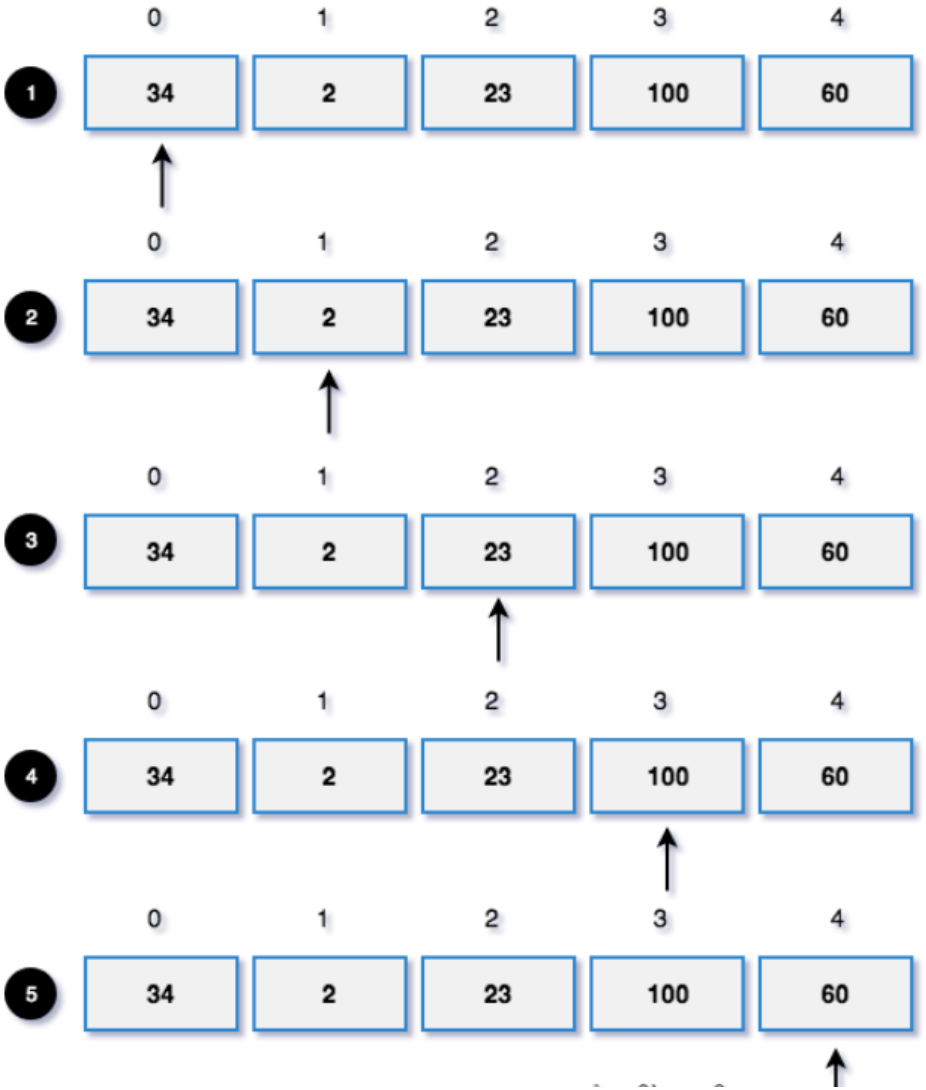
1. 34 != 100. Move to the next element
2. 2 != 100. Move to the next element
3. 23 != 100. Move to the next element
4. **100 == 100. Search Found.**

Linear Search

Let's take an array of 5 elements:

34, 2, 23, 100, 60.

Search element = 200



The execution will be:

1. 34 != 200. Move to the next element.
2. 2 != 200. Move to the next element.
3. 23 != 200. Move to the next element.
4. 100 != 200. Move to the next element.
5. **60 != 200. Search Not Found**

Linear Search

Time Complexity Analysis:

The **Best Case** occurs when the target element is the first element of the array. The number of comparisons, in this case, is 1. So, the time complexity is O(1).

The **Average Case**: On average, the target element will be somewhere in the middle of the array. The number of comparisons, in this case, will be N/2. So, the time complexity will be O(N) (the constant being ignored).

The **Worst Case** occurs when the target element is the last element in the array or not in the array. In this case, we have to traverse the entire array, and so the number of comparisons will be N. So, the time complexity will be O(N).

Binary Search

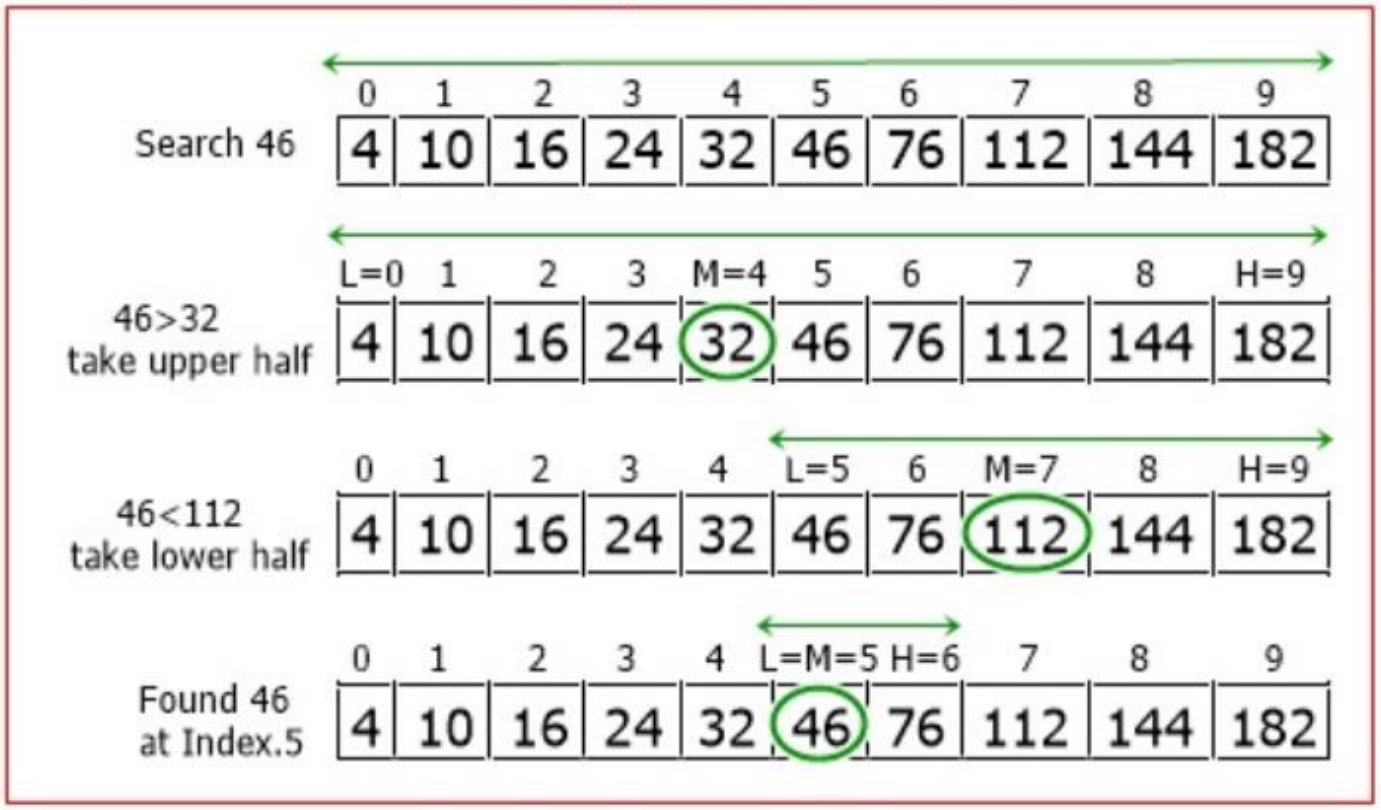
Approach:

- Compare the target element with the middle element of the array.
- If the target element is greater than the middle element, then the search continues in the right half.
- Else if the target element is less than the middle value, the search continues in the left half.
- This process is repeated until the middle element is equal to the target element, or the target element is not in the array
- If the target element is found, its index is returned, else -1 is returned.

Binary Search

Example: Array = 4, 10, 16, 24, 32, 46, 76, 112, 144, and 182

Search element = 46



46 is found at index number 5.

Binary Search

Time Complexity Analysis:

The **Best Case** occurs when the target element is the middle element of the array. The number of comparisons, in this case, is 1. So, the time complexity is O(1).

The **Average Case**: On average, the target element will be somewhere in the array. So, the time complexity will be O(logN).

The **Worst Case** occurs when the target element is not in the list or it is away from the middle element. So, the time complexity will be O(logN).

Recall:

- Array Searching
- Linear search
- Binary search