Searching Algorithms

Charles Aunkan Gomes
Lecturer, Dept. of CSE
United International University
charles@cse.uiu.ac.bd

Searching

The process of finding a particular element in an array is called searching. There two popular searching techniques:

- Linear search
- Binary search.

- The linear search compares each array element with the search key.
- If the search key is a member of the array, typically the location of the search key is reported to indicate the presence of the search key in the array. Otherwise, a sentinel value is reported to indicate the absence of the search key in the array.

Linear Search

Linear search is also called as **sequential search algorithm**. It is the simplest searching algorithm.

The steps used in the implementation of Linear Search are listed as follows:

- First, we have to traverse the array elements using a loop.
- In each iteration of **loop,** compare the search element with the current array element, and
  - If the element matches, then return the index of the corresponding array element.
  - If the element does not match, then move to the next element.
- If there is no match or the search element is not present in the given array, return **-1.**

Linear Search

Pseudocode for a linear search algorithm:

```
LinearSearch(array, key):
    for index from 0 to length(array) - 1:
        if array[index] == key:
            return index  // Key found
    return -1  // Key not found
```

Linear Search

Time Complexity:

Best Case: In the best case, the key might be present at the first index. So the best case complexity is O(1)

Worst Case: In the worst case, the key might be present at the last index i.e., opposite to the end from which the search has started in the list. So the worst-case complexity is O(N) where N is the size of the list.

Average Case: O(N)

Binary Search

Binary search is a search algorithm used to find the position of a target value within a sorted array. It works by **repeatedly dividing** the search interval in half until the target value is found or the interval is empty. The search interval is halved by comparing the target element with the middle value of the search space.

The steps used in the implementation of Binary Search are listed as follows:

- Divide the search space into two halves by finding the middle index "mid".
- Compare the middle element of the search space with the key.

  If the key is found at middle element, the process is terminated.

Binary Search

- If the key is not found at middle element, choose which half will be used as the next search space.
  - If the key is smaller than the middle element, then the left side is used for next search.
  - If the key is larger than the middle element, then the right side is used for next search.

This process is continued until the key is found or the total search space is exhausted.

Binary Search

```
BinarySearch(array, key):
    left = 0, right = length(array) – 1

    while left <= right:
        mid = (left + right)

        if array[mid] == key:
            return mid  // Key found
        else if array[mid] < key:
            left = mid + 1  // Narrow the search to the right half
        else:
            right = mid - 1  // Narrow the search to the left half

    return -1  // Key not found
```

Binary Search

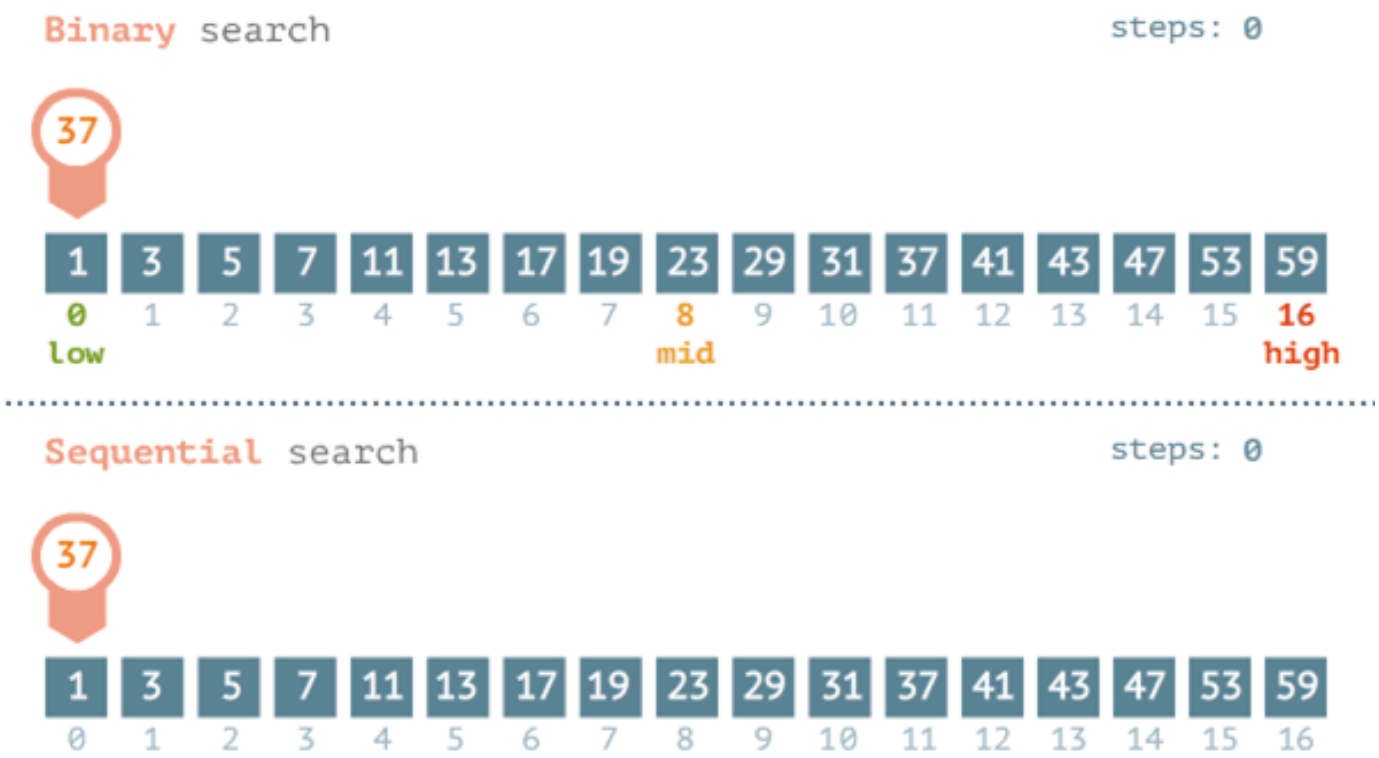
Time Complexity:

Best Case: In Binary search, best case occurs when the element to search is found in first comparison, i.e., when the first middle element itself is the element to be searched. The best-case time complexity of Binary search is O(1).

Worst Case: In Binary search, the worst case occurs, when we have to keep reducing the search space till it has only one element. The worst-case time complexity of Binary search is O(logn).

Average Case: O(logn)

Binary Search vs Linear Search



THANK YOU