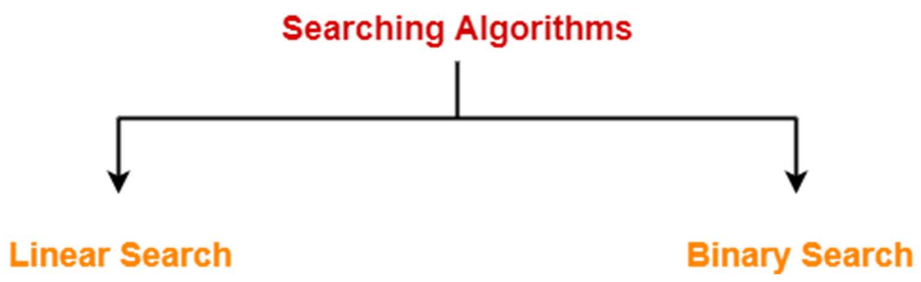
Searching

Searching Algorithms are designed to check for an element or retrieve an element from any data structure where it is stored. Based on the type of search operation, these algorithms are generally classified into two categories:

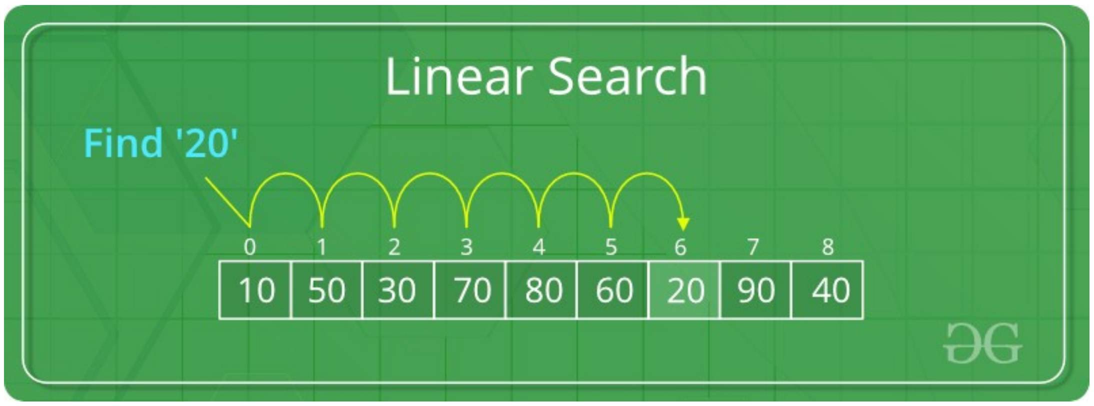
1. **Sequential Search**: In this, the list or array is traversed sequentially and every element is checked. For example: Linear Search.
2. **Interval Search**: These algorithms are specifically designed for searching in sorted data-structures. These type of searching algorithms are much more efficient than Linear Search as they repeatedly target the center of the search structure and divide the search space in half. For Example: Binary Search.



Linear Search

A simple approach of doing a **Linear Search** is

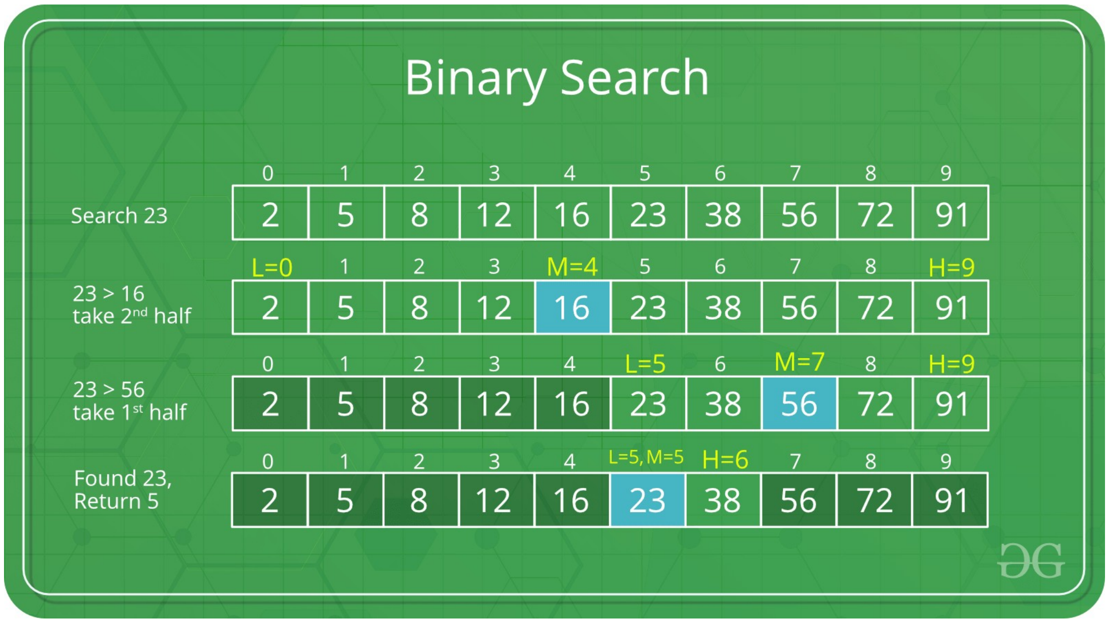
- Start from the leftmost element of the array and one by one compare the elect to be found (x) with each element of the array.
- If x matches with an element, return the index.
- If you reach the end of the array and x doesn't match with any of elements, return -1.



Binary Search

An important aspect of **Binary Search** is that the array must be sorted. A simple approach of doing this type of search is:

- Search a sorted array by repeatedly dividing the search interval in half. Begin with an interval covering the whole array.
- If the value of the search key is less than the item in the middle of the interval, narrow the interval to the lower half.
- Otherwise narrow it to the upper half.
- Repeatedly check until the value is found or the interval is empty.



Pseudo Code for Binary Search

```
function binary_search(A, T):
    L := 0
    R := len(A) - 1
    while L <= R:
        m := floor((L + R) / 2)
        if A[m] < T:
            L := m + 1
        else if A[m] > T:
            R := m - 1
        else:
            return m
    return unsuccessful
```

References:

- Searching Algorithms