## 4.1 SEARCHING: BINARY SEARCH AND VARIANTS

Searching is an operation or a technique that helps finds the place of a given element.

**Searching Techniques**

To search an element in a given array, it can be done in following ways:

1. Sequential Search
2. Binary Search

## Linear Search

- Sequential search is also called as Linear Search.
- Sequential search starts at the beginning of the list and checks every element of the list.
- It is a basic and simple search algorithm.
- Sequential search compares the element with all the other elements given in the list. If the element is matched, it returns the value index, else it returns -1.

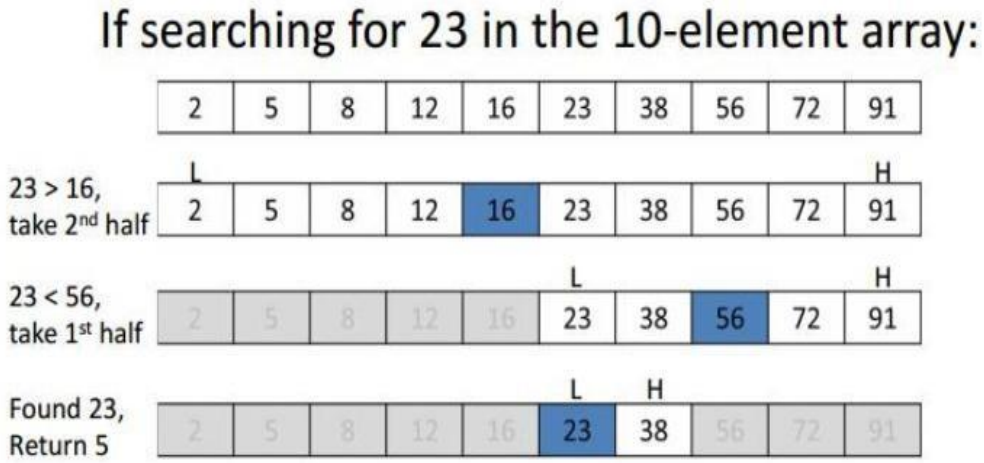
A simple approach is to do linear search, i.e

- Start from the leftmost element of arr[] and one by one compare x with each element of arr[] If x matches with an element, return the index.

## BINARY SEARCH AND VARIANTS

**Definition:** A binary search or half-interval search algorithm finds the position of a specified value within a sorted array**.**

- 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.




**Algorithm**

- Compare x with the middle element
- If x matches with middle element, we return the mid index.
- Else If x is greater than the mid element, then x can only lie in right half subarray after the mid element. So we recur for right half.
- Else (x is smaller) recur for the left half.

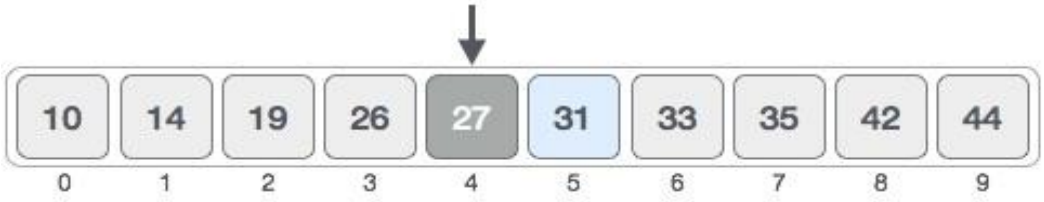
**How Binary Search Works?**

- For a binary search to work, it is mandatory for the target array to be sorted.
- The following is our sorted array and let us assume that we need to search the location of value 31 using binary search


First, we shall determine half of the array by using this formula −

mid = low + (high - low) / 2

Here it is, 0 + (9 - 0 ) / 2 = 4 (integer value of 4.5). So, 4 is the mid of the array.




Now we compare the value stored at location 4, with the value being searched, i.e. 31. We find that the value at location 4 is 27, which is not a match.

As the value is greater than 27 and we have a sorted array, so we also know that the target value must be in the upper portion of the array.


We change our low to mid + 1 and find the new mid value again.

low = mid + 1

mid = low + (high - low) / 2 =5+(9-5)/2 = 5+2 = 7

Our new mid is 7 now. We compare the value stored at location 7 with our target value 31.


The value stored at location 7 is not a match, rather it is more than what we are looking for. So, the value must be in the lower part from this location

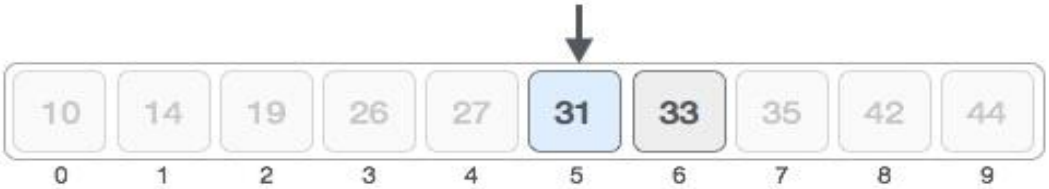

Hence, we calculate the mid again. This time it is 5

high=mid-1 high=7-1=6

Low=5

mid= low + (high - low) / 2 = 5+(6-5)/2

=5+0.5=5.5 =5




We compare the value stored at location 5 with our target value. We find that it is a match.

We conclude that the target value 31 is stored at location 5.

**Time Complexity**

- Linear Search O(n)
- Binary Search O(log n)

**Program**

```
def binary_search_recursive(arr, target, low, high):
    if low > high:
        return -1  # element not found
    mid = (low + high) // 2
    if arr[mid] == target:
        return mid
    elif arr[mid] < target:
        return binary_search_recursive(arr, target, mid + 1, high)
    else:
        return binary_search_recursive(arr, target, low, mid - 1)
# Example usage
arr = [1, 3, 4, 6, 8, 10]
target = 4
result = binary_search_recursive(arr, target, 0, len(arr) - 1)
if result != -1:
    print(f"Element {target} found at index {result}")
else:
    print(f"Element {target} not found")
```

**Variants**

1. Standard Binary Search

- Finds whether an element exists in a sorted array.
- Returns any index of the target if found.
- Can be iterative or recursive.
- Time Complexity: O(log n)

2. Lower Bound (Leftmost)

- Finds the first occurrence of a target (or the first element ≥ target).
- Useful when duplicates exist

3. Upper Bound (Rightmost)

- Finds the last occurrence of a target

4. First True / Predicate Binary Search

- Used when you are searching for the first index where a condition becomes true.

5. Last True / Predicate Binary Search

- Finds the last index where a condition is true.