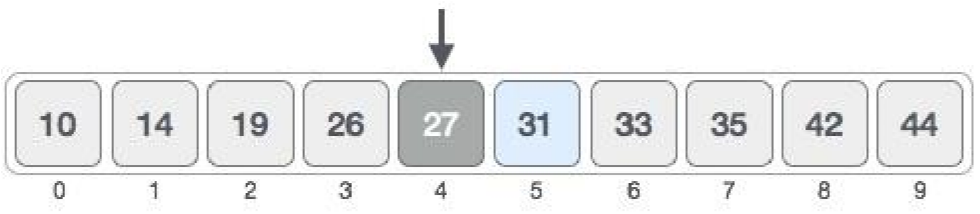
|  |
| --- |
| ***Searching techniques: Binary Search, Linear search,***  ***Sorting techniques: Bubble sort, insertion sort*** |
| **Binary Search**  Binary Search is used with sorted array or list. In binary search, we follow the following steps:   1. We start by comparing the element to be searched with the element in the **middle** of the list/array. 2. If we get a match, we **return the index of the middle element.** 3. If we do not get a match, we check whether the element to be searched is **less or greater than in value than the middle element**. 4. If the element/number to be searched is **greater in value** than the middle number, then we pick the elements on the right side of the middle element(as the list/array is sorted, hence on the right, we will have all the numbers greater than the middle number), and start again from the step 1. 5. If the element/number to be searched is **lesser in value** than the middle number, then we pick the elements on the left side of the middle element, and start again from the step 1.   Binary Search is useful when there are large number of elements in an array and they are **sorted.**  So a necessary condition for Binary search to work is that the list/array should be sorted.  **Features of Binary Search**   1. It is great to search through large sorted arrays. 2. It has a time complexity of **O(log n)** which is a very good time complexity. It has a simple implementation.   **How Binary Search Works?**  For a binary search to work, it is mandatory for the target array to be sorted. We shall learn the process of binary search with a pictorial example. 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

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

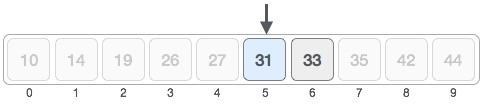


The value stored at location 7 is not a match, rather it is more than what we are looking for. So, the

value must be in the lower part from this location.

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

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.

# Linear search

**Linear search** is a searching algorithm which is used to detect the presence of a number in an array and if present, it locates its position in that array.

This algorithm compares each element of the array with the search query comparing every element until the number is found and located.

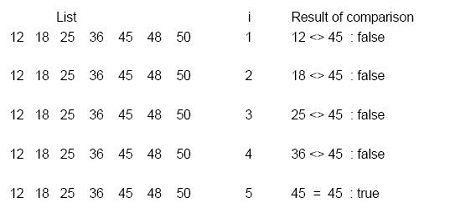
A simple approach is to do **linear search**, i.e

* 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.
* If x doesn’t match with any of elements, return -1.

Ex.

Assume the element 45 is searched from a sequence of sorted elements 12, 18, 25, 36, 45, 48, 50.

The Linear search starts from the first element 12, since the value to be searched is not 12 (value 45), the next element 18 is compared and is also not 45, by this way all the elements before 45 are compared and when the index is 5, the element 45 is compared with the search value and is equal, hence the element is found and the element position is 5.



Bubble sort:

Bubble sort is also known as sinking sort. This algorithm compares each pair of **adjacent** items and **swaps them** if they are in the wrong order, and this same process goes on until no swaps are needed.

Consider an array A of n elements whose elements are to be sorted by using Bubble sort.

The algorithm processes like following.

1. In Pass 1, A[0] is compared with A[1], A[1] is compared with A[2], A[2] is compared with A[3] and so on. At the end of pass 1, the largest element of the list is placed at the highest index of the list.
2. In Pass 2, A[0] is compared with A[1], A[1] is compared with A[2] and so on. At the end of Pass 2 the second largest element of the list is placed at the second highest index of the list.
3. In pass n-1, A[0] is compared with A[1], A[1] is compared with A[2] and so on. At the end of this pass. The smallest element of the list is placed at the first index of the list.

Example:

**I/P :**

**5**

**,3,1,9,8,2,4,**

**7**

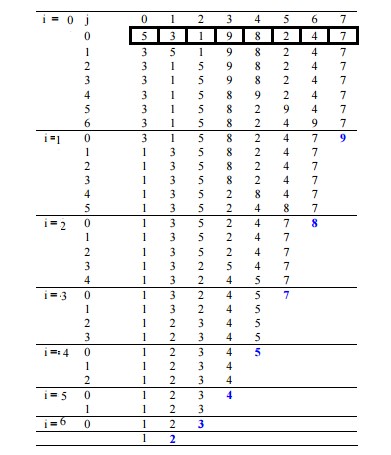
**O/P**

**:**

**1**

**,2,3,4,5,7,8,**

**9**



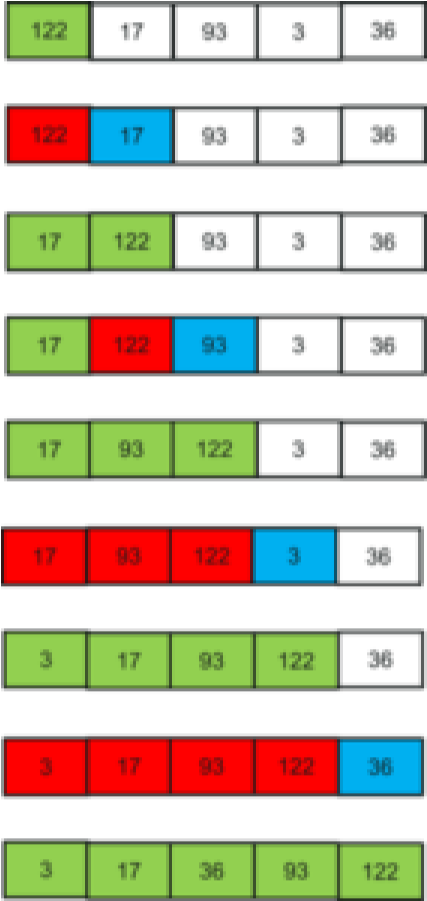
Insertion sort:

**Insertion sort is a sorting algorithm that places an unsorted element at its suitable place in each iteration.**

**Algorithm for Insertion Sort**

* **Step 1** − If the element is the first one, it is already sorted.
* **Step 2** – Move to next element
* **Step 3** − Compare the current element with all elements in the sorted array
* **Step 4** – If the element in the sorted array is smaller than the current element, iterate to the next element. Otherwise, shift all the greater element in the array by one position towards the right
* **Step 5** − Insert the value at the correct position
* **Step 6** − Repeat until the complete list is sorted

To understand how Insertion sort works, refer to the below image.



Let’s understand how insertion sort is working in the above image.

* **122**, 17, 93, 3, 36 for i = 1(2nd element) to 36 (last element)

i = 1. Since 17 is smaller than 122, move 122 and insert 17 before 122

* **17, 122**, 93, 3, 36 i = 2. Since 93 is smaller than 122, move 122 and insert 93 before 122

* **17, 93,122**, 3, 36 i = 3. 3 will move to the beginning and all other elements from 17 to 122 will move one position ahead of their current position.

* **3, 17, 93, 122,** 36

i = 4. 36 will move to position after 17, and elements from 93 to 122 will move one position ahead of their current position.

• **3, 17, 36, 93 ,122**