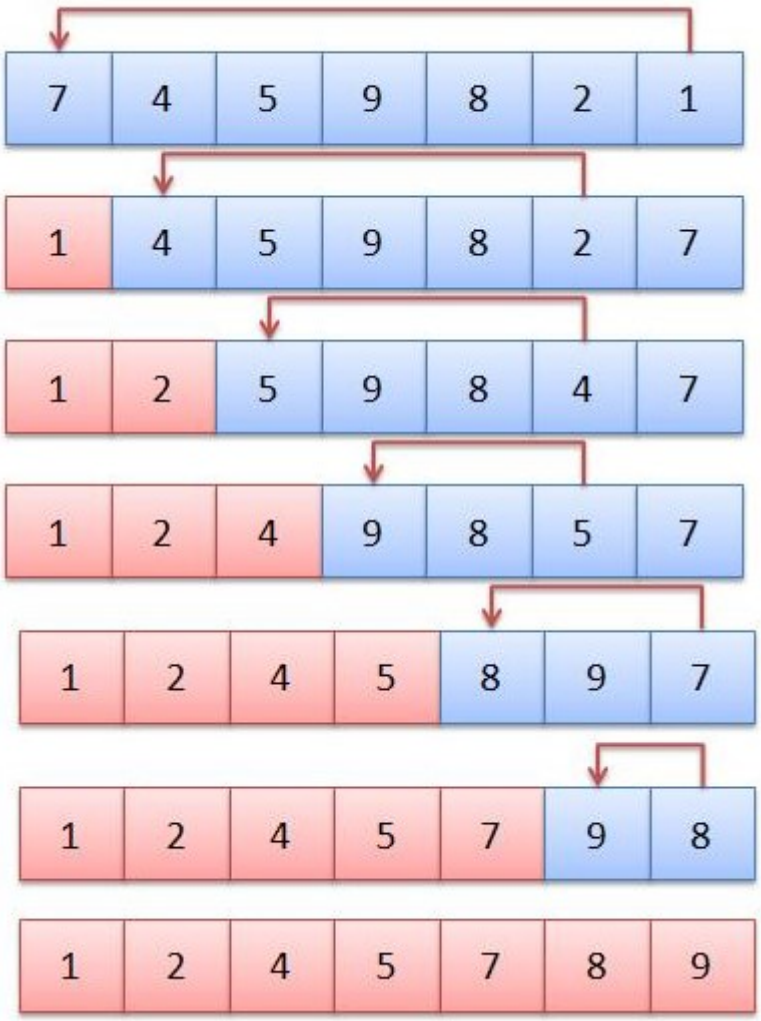
Selection Sort

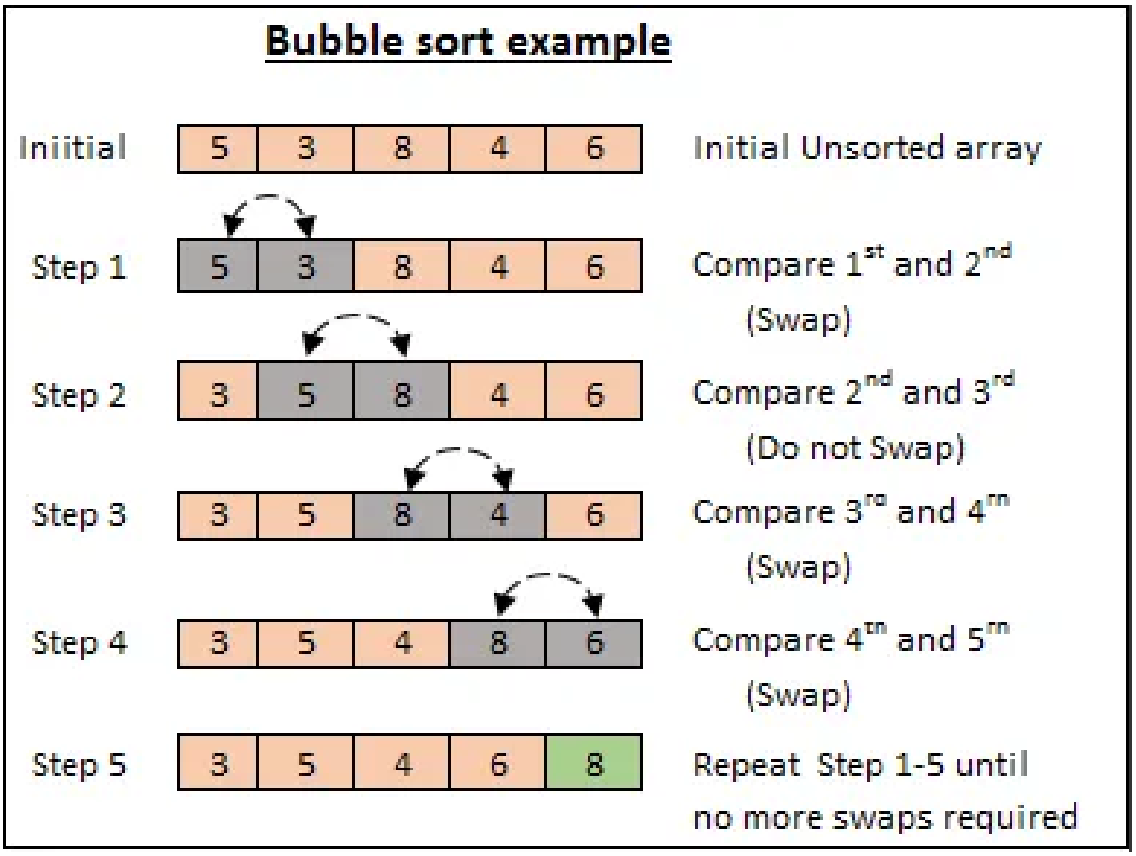
Selection sort is a sorting algorithm that selects the smallest element from an unsorted list in each iteration and places that element at the beginning of the unsorted list.

Selection Sort:



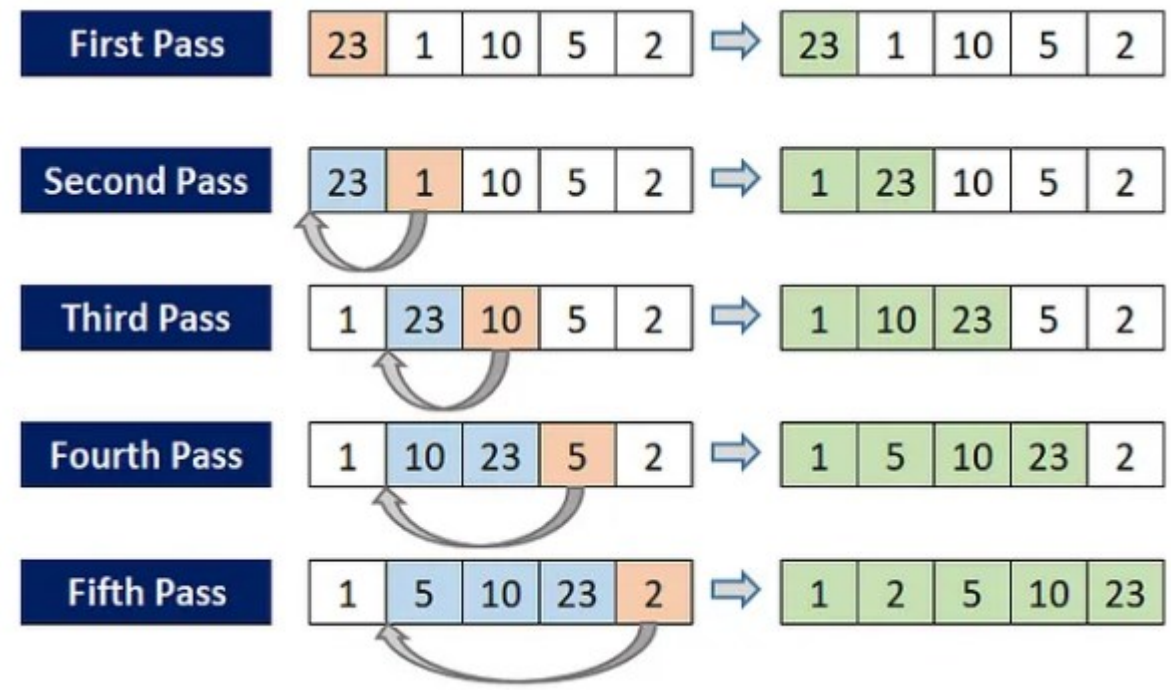
Bubble Sort

Bubble sort is a sorting algorithm that compares two adjacent elements and swaps them until they are in the intended order.



Insertion Sort

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

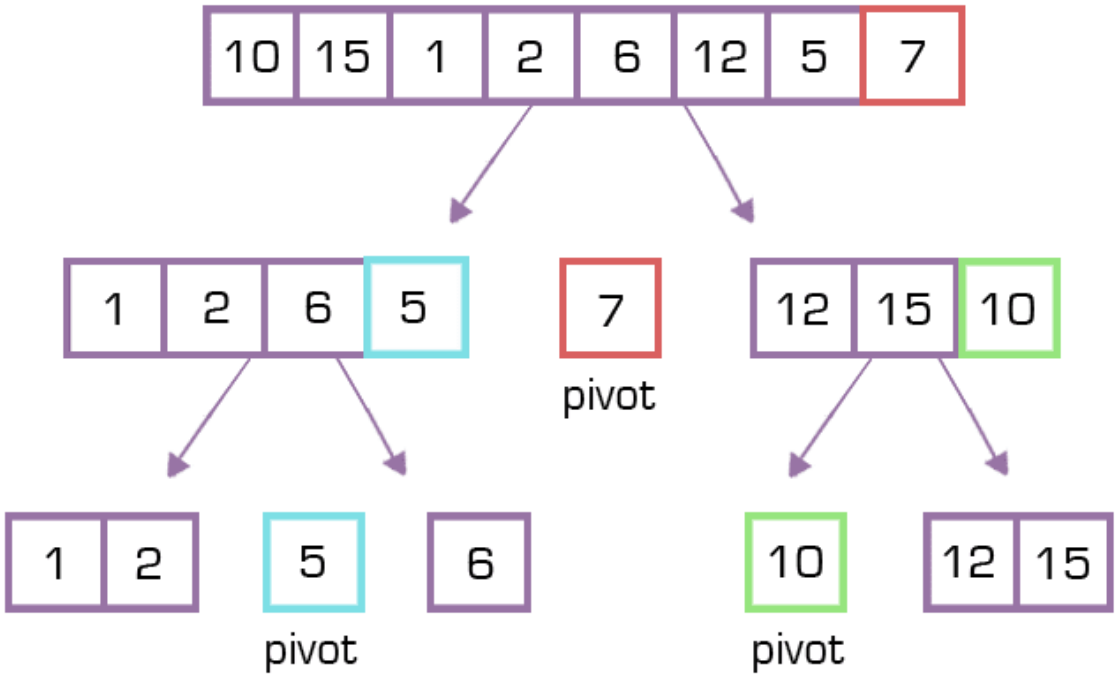


Quick Sort

Quicksort is a sorting algorithm based on the **divide and conquer approach** where

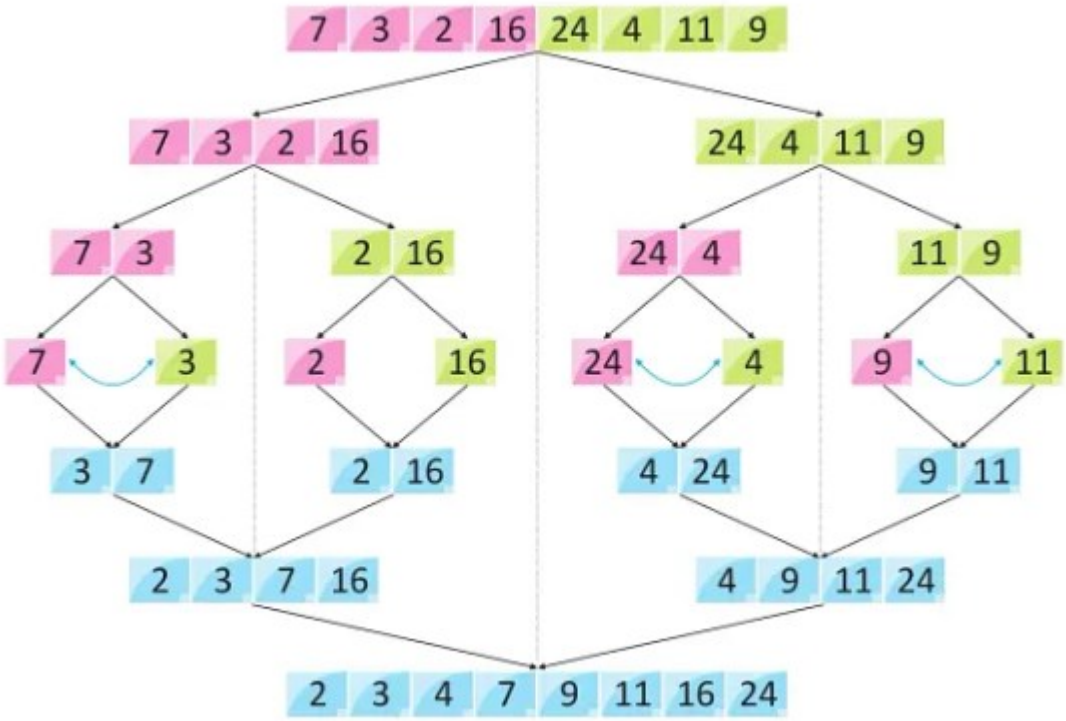
1. An array is divided into subarrays by selecting a **pivot element** (element selected from the array).

   While dividing the array, the pivot element should be positioned in such a way that elements less than pivot are kept on the left side and elements greater than pivot are on the right side of the pivot.
2. The left and right subarrays are also divided using the same approach. This process continues until each subarray contains a single element.
3. At this point, elements are already sorted. Finally, elements are combined to form a sorted array.



Merge Sort

Merge Sort Algorithm is a classic ‘Divide and Conquer’ algorithm where the list to be sorted is divided into smaller parts, sorted individually, and then merged back together.

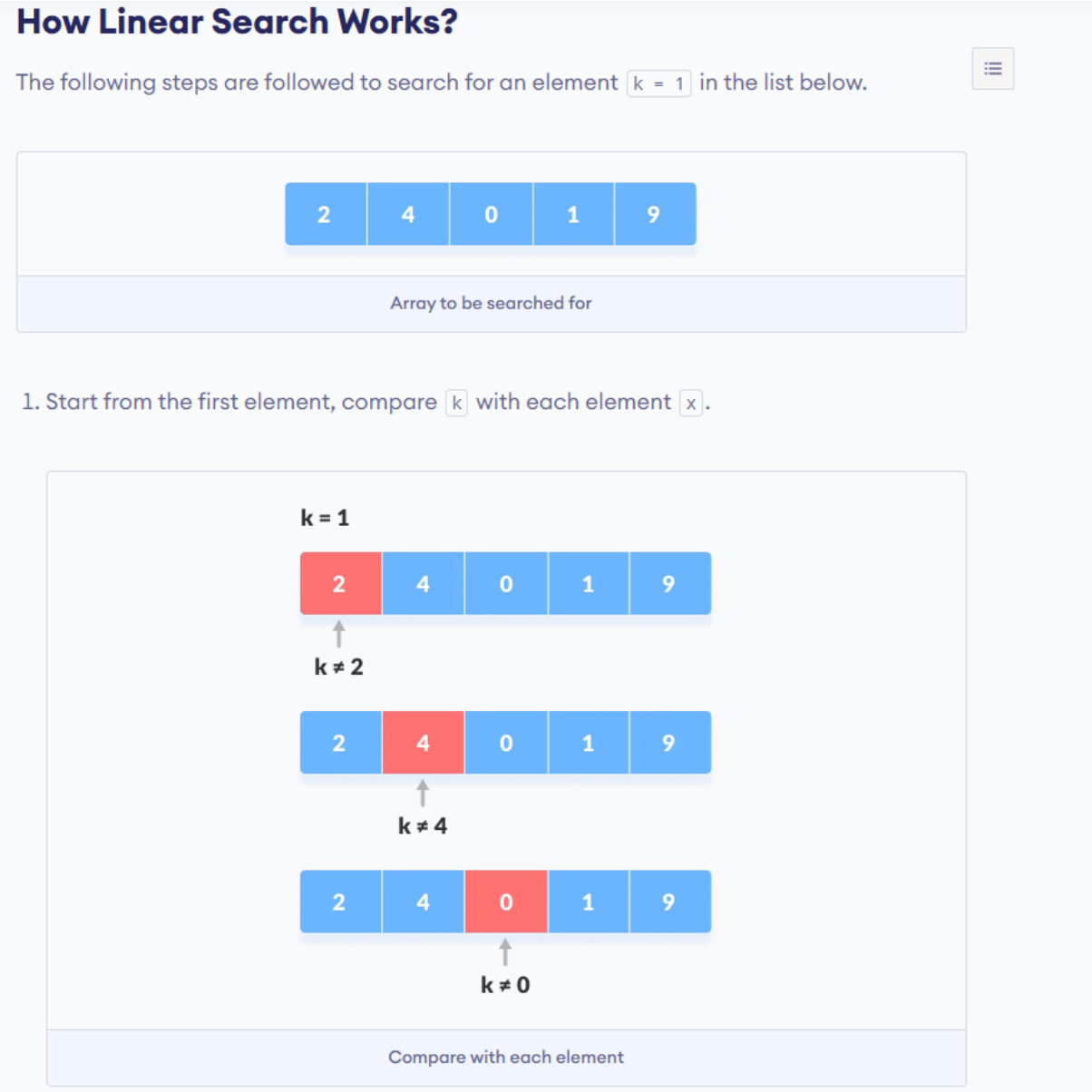


Searching Alorithms

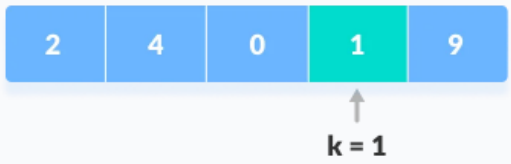
Linear Search

Linear Search

Linear search is a sequential searching algorithm where we start from one end and check every element of the list until the desired element is found



2. If `x == k`, return the index.



Element found

3. Else, return `not found`.

Binary Search

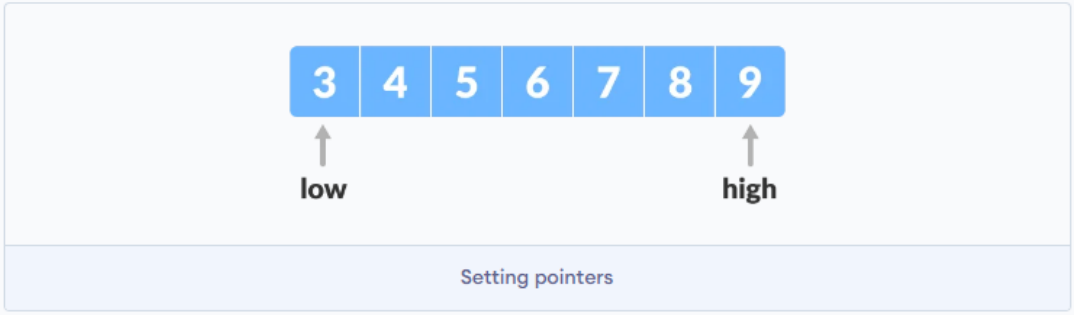
Binary Search is a searching algorithm for finding an element's position in a sorted array.

In this approach, the element is always searched in the middle of a portion of an array.

The search element will be either on the left part of the array or on the right part of the array.

Let `x = 4` be the element to be searched.

2. Set two pointers low and high at the lowest and the highest positions respectively.



Setting pointers

3. Find the middle element `mid` of the array ie. `arr[(low + high)/2] = 6`.

Mid element

4. If x == mid, then return mid.Else, compare the element to be searched with m.
5. If `x > mid`, compare `x` with the middle element of the elements on the right side of `mid`. This is done by setting `low` to `low = mid + 1`.
6. Else, compare `x` with the middle element of the elements on the left side of `mid`. This is done by setting `high` to `high = mid - 1`.

Finding mid element

7. Repeat steps 3 to 6 until low meets high.

Mid element

8. `x = 4` is found.