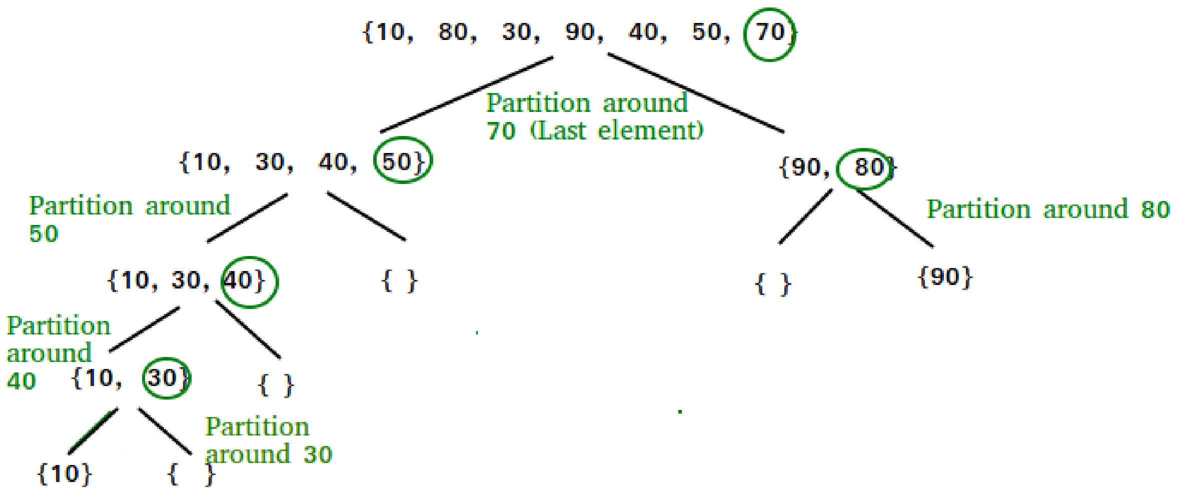
***QuickSort** is a sorting algorithm based on the [Divide and Conquer algorithm](#) that picks an element as a pivot and partitions the given array around the picked pivot by placing the pivot in its correct position in the sorted array.*

*The key process in **quickSort** is a **partition()**. The target of partitions is to place the pivot (any element can be chosen to be a pivot) at its correct position in the sorted array and put all smaller elements to the left of the pivot, and all greater elements to the right of the pivot.*
Partition is done recursively on each side of the pivot after the pivot is placed in its correct position and this finally sorts the array.



Partition Algorithm:
*The logic is simple, we start from the leftmost element and keep track of the index of smaller (or equal) elements as **i**. While traversing, if we find a smaller element, we swap the current element with arr[i]. Otherwise, we ignore the current element.*

Let us understand the working of partition and the Quick Sort algorithm with the help of the following example:
Consider: arr[] = {10, 80, 30, 90, 40}.

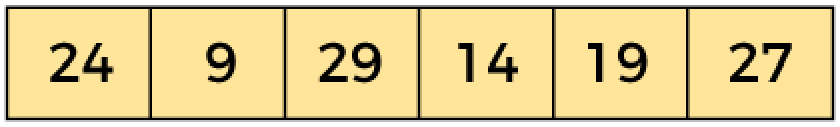
- *Compare 10 with the pivot and as it is less than pivot arrange it accrodingly.*

Working of Quick Sort Algorithm

Now, let's see the working of the Quicksort Algorithm.

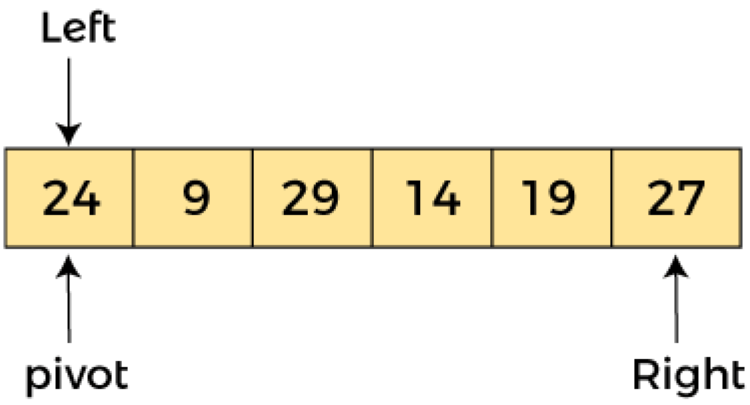
To understand the working of quick sort, let's take an unsorted array. It will make the concept more clear and understandable.

Let the elements of array are -

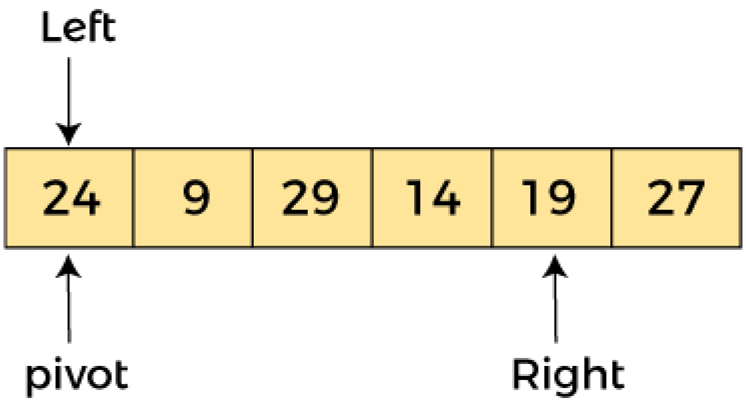


In the given array, we consider the leftmost element as pivot. So, in this case, a[left] = 24, a[right] = 27 and a[pivot] = 24.

Since, pivot is at left, so algorithm starts from right and move towards left.

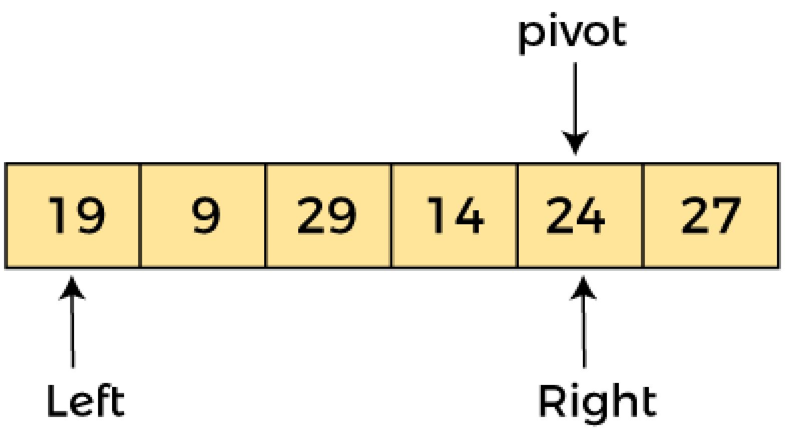


Now, a[pivot] < a[right], so algorithm moves forward one position towards left, i.e. -



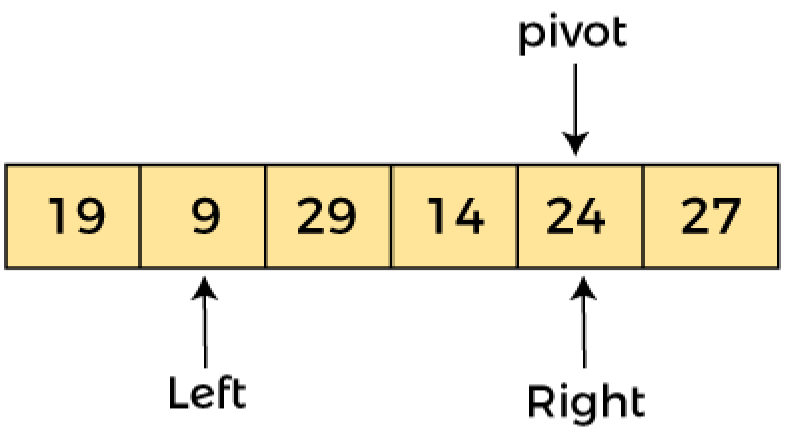
Now, a[left] = 24, a[right] = 19, and a[pivot] = 24.

Because, a[pivot] > a[right], so, algorithm will swap a[pivot] with a[right], and pivot moves to right, as -

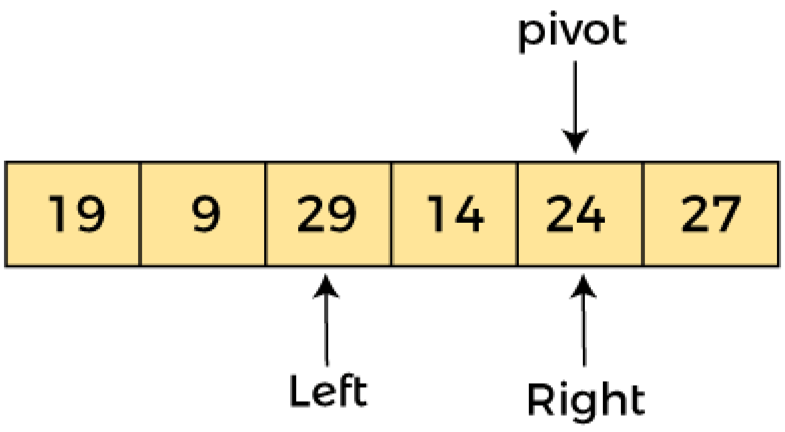


Now, a[left] = 19, a[right] = 24, and a[pivot] = 24. Since, pivot is at right, so algorithm starts from left and moves to right.

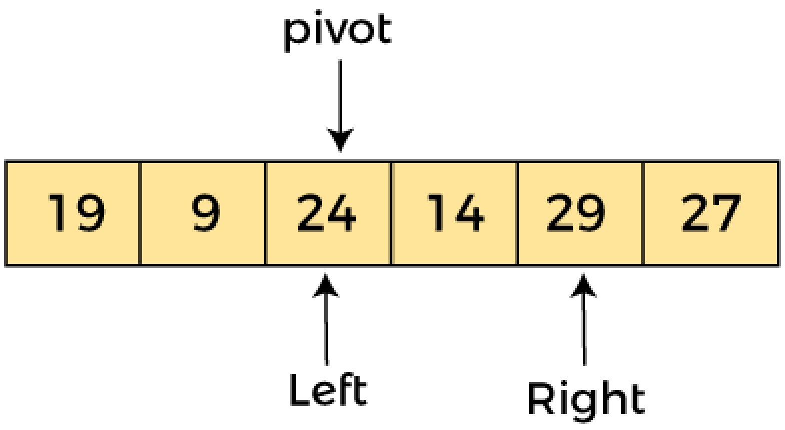
As a[pivot] > a[left], so algorithm moves one position to right as -



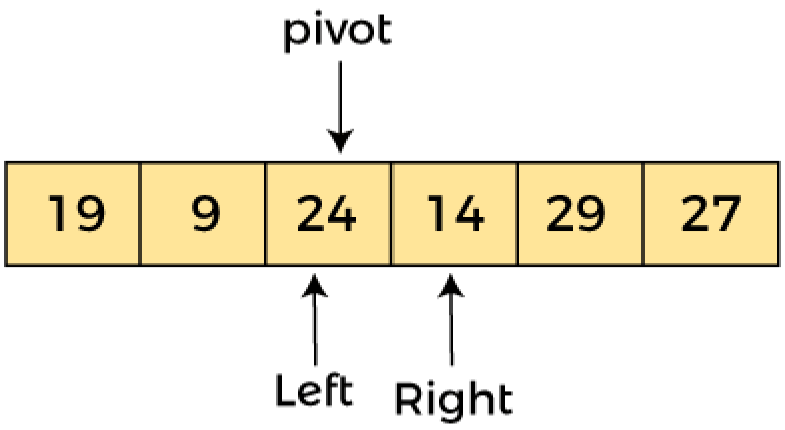
Now, a[left] = 9, a[right] = 24, and a[pivot] = 24. As a[pivot] > a[left], so algorithm moves one position to right as -



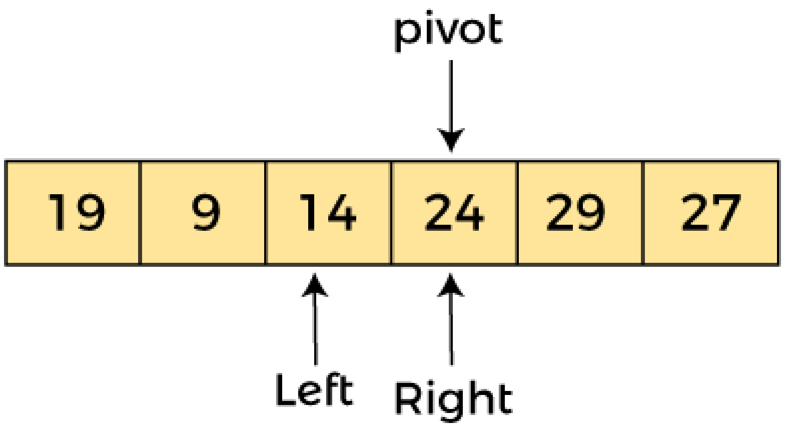
Now, a[left] = 29, a[right] = 24, and a[pivot] = 24. As a[pivot] < a[left], so, swap a[pivot] and a[left], now pivot is at left, i.e. -



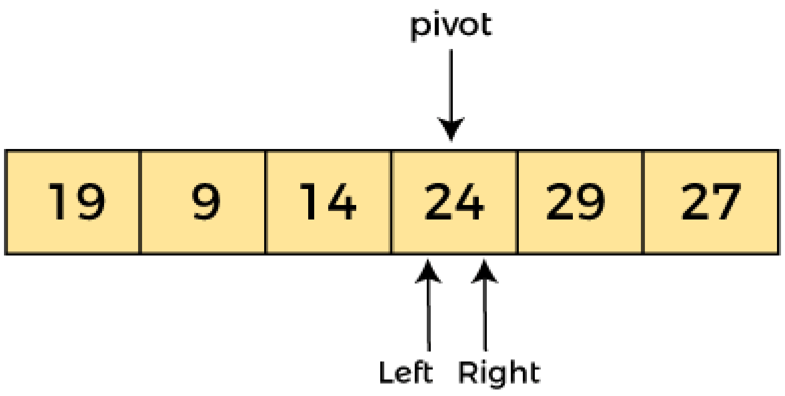
Since, pivot is at left, so algorithm starts from right, and move to left. Now, a[left] = 24, a[right] = 29, and a[pivot] = 24. As a[pivot] < a[right], so algorithm moves one position to left, as -



Now, a[pivot] = 24, a[left] = 24, and a[right] = 14. As a[pivot] > a[right], so, swap a[pivot] and a[right], now pivot is at right, i.e. -



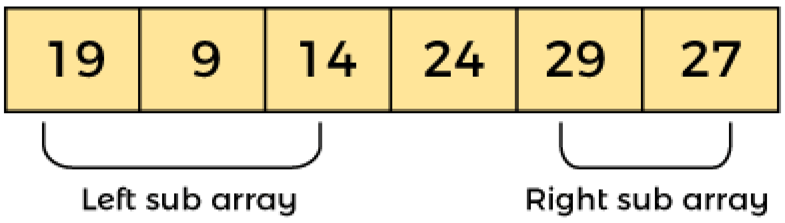
Now, a[pivot] = 24, a[left] = 14, and a[right] = 24. Pivot is at right, so the algorithm starts from left and move to right.



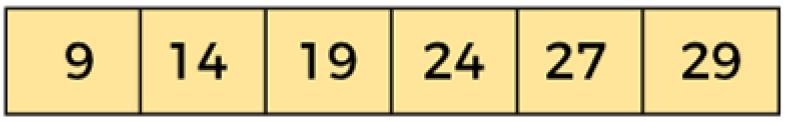
Now, a[pivot] = 24, a[left] = 24, and a[right] = 24. So, pivot, left and right are pointing the same element. It represents the termination of procedure.

Element 24, which is the pivot element is placed at its exact position.

Elements that are right side of element 24 are greater than it, and the elements that are left side of element 24 are smaller than it.



Now, in a similar manner, quick sort algorithm is separately applied to the left and right sub-arrays. After sorting gets done, the array will be -



Partition in QuickSort: Compare pivot with 10

- *Compare 80 with the pivot. It is greater than pivot.*

Partition in QuickSort: Compare pivot with 80

- *Compare 30 with pivot. It is less than pivot so arrange it accordingly.*

Partition in QuickSort: Compare pivot with 30

- *Compare 90 with the pivot. It is greater than the pivot.*

Partition in QuickSort: Compare pivot with 90

- *Arrange the pivot in its correct position.*

Partition in QuickSort: Place pivot in its correct position

Illustration of Quicksort:

As the partition process is done recursively, it keeps on putting the pivot in its actual position in the sorted array. Repeatedly putting pivots in their actual position makes the array sorted.

Follow the below images to understand how the recursive implementation of the partition algorithm helps to sort the array.

- *Initial partition on the main array:*

Quicksort: Performing the partition

- *Partitioning of the subarrays:*