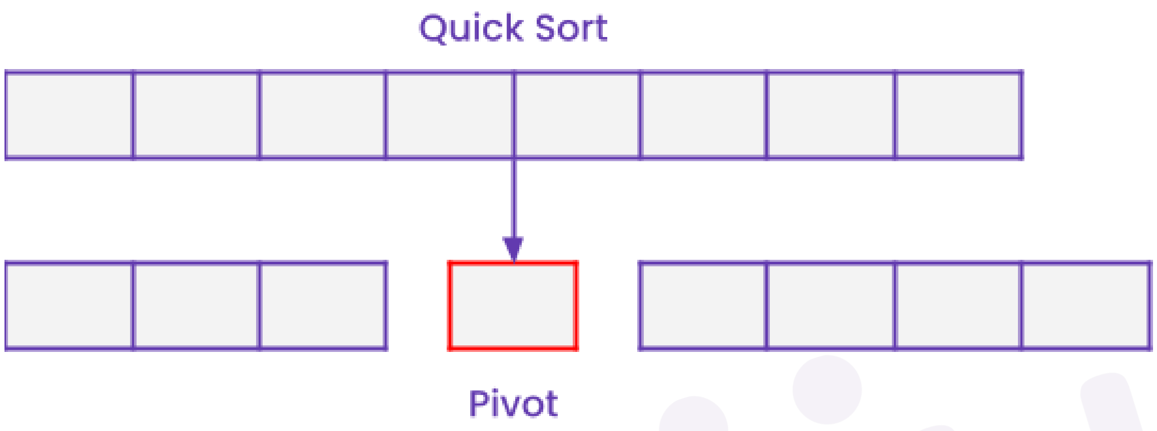
Quick sort Algorithm :

Quicksort picks an element as pivot, and then it partitions the given array around the picked pivot element. In quick sort, a large array is divided into two arrays in which one holds values that are smaller than the specified value (Pivot), and another array holds the values that are greater than the pivot.

After that, left and right sub-arrays are also partitioned using the same approach. It will continue until the single element remains in the sub-array.



Choosing the pivot

Picking a good pivot is necessary for the fast implementation of quicksort. However, it is typical to determine a good pivot. Some of the ways of choosing a pivot are as follows -

- Pivot can be random, i.e. select the random pivot from the given array.
- Pivot can either be the rightmost element of the leftmost element of the given array.
- Select median as the pivot element.

Algorithm:

```
QUICKSORT (array A, start, end)
 • if (start < end)
 • p = partition(A, start, end)
 • QUICKSORT (A, start, p - 1)
 • QUICKSORT (A, p + 1, end)
 • end if
```

Partition Algorithm:

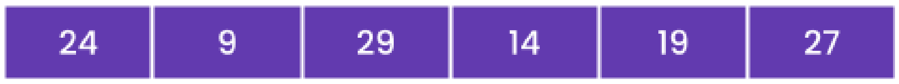
The partition algorithm rearranges the sub-arrays in a place.

```
PARTITION (array A, start, end){
 • pivot = A[end]
 • i = start-1
 • for j = start to end -1 {
 • do if (A[j] < pivot) {
 • then i = i + 1
 • swap A[i] with A[j]
 •  }}
 • swap A[i+1] with A[end]
 • return i+1
}
```

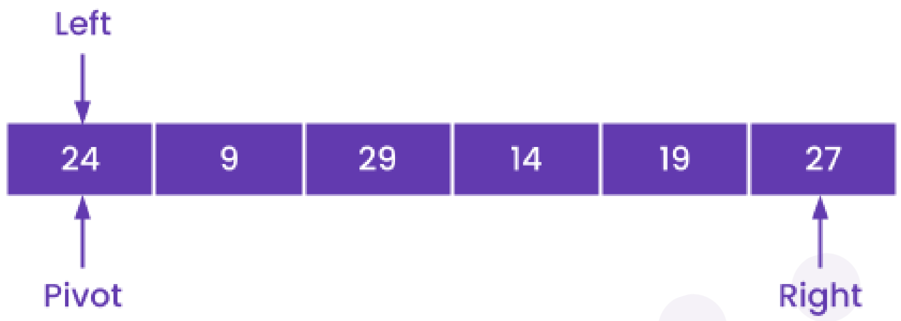
Working of quick sort algorithm :

let's understand the working of quicksort algorithm with the help of an example :

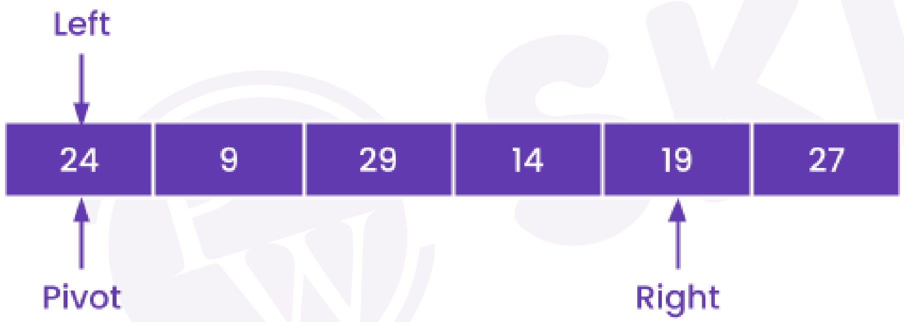
Let the elements of array are -



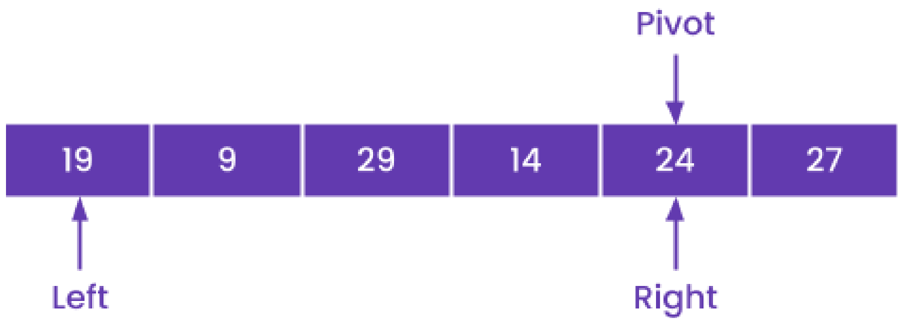
In the given array, we consider the leftmost element as pivot. So, in this case, a[left] = 24, a[right] = 27 and a[pivot] = 24.
The pivot is at left, so the algorithm starts from right and moves towards left.



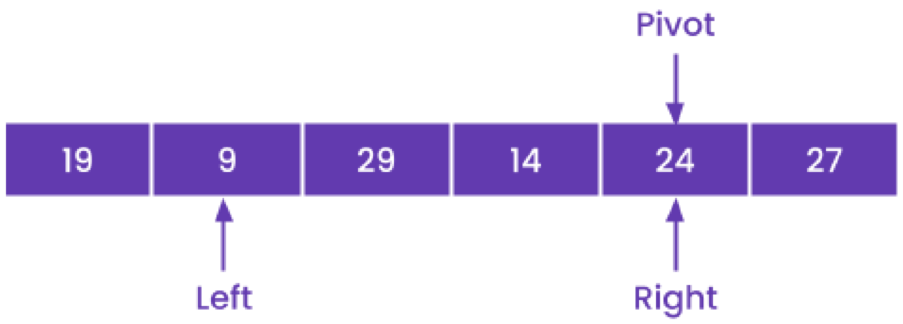
Now, a[pivot] < a[right], so algorithm moves forward one position towards left, i.e. -



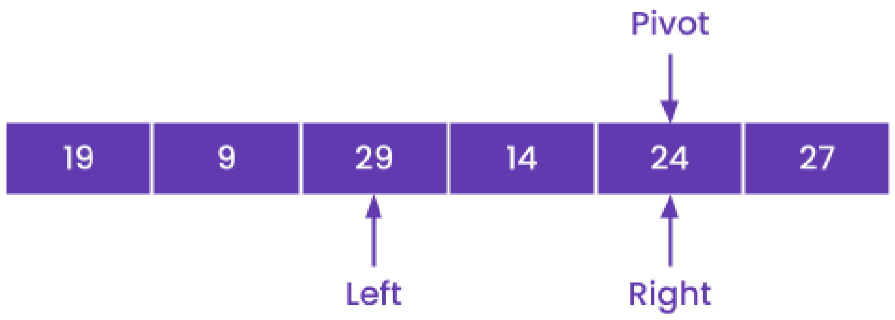
Now, a[left] = 24, a[right] = 19, and a[pivot] = 24.
Because, a[pivot] > a[right], so, algorithm will swap a[pivot] with a[right], and pivot moves to right, as -



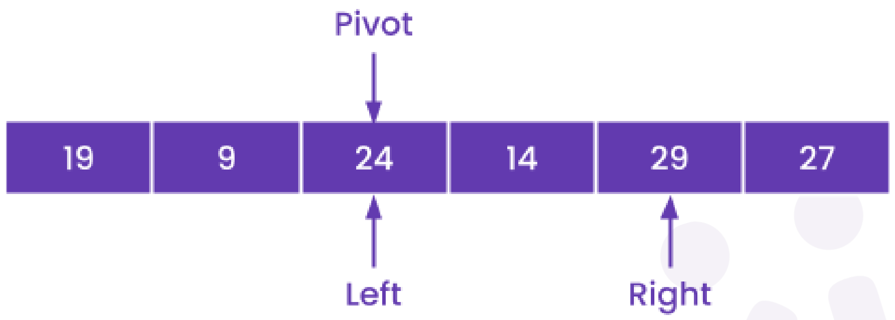
Now, a[left] = 19, a[right] = 24, and a[pivot] = 24. Since, the pivot is at right, so the algorithm starts from left and moves to right.
As a[pivot] > a[left], so algorithm moves one position to right as -



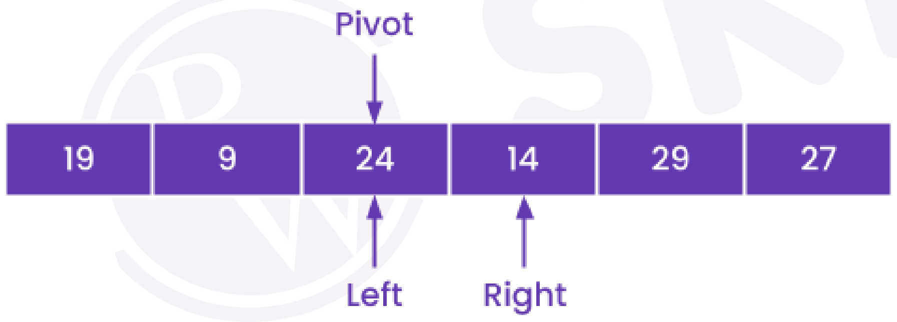
Now, a[left] = 9, a[right] = 24, and a[pivot] = 24. As a[pivot] > a[left], so algorithm moves one position to right as



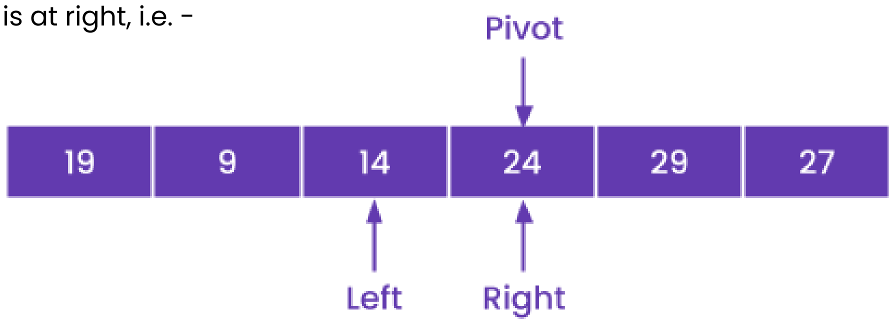
Now, a[left] = 29, a[right] = 24, and a[pivot] = 24. As a[pivot] < a[left], so, swap a[pivot] and a[left], now pivot is at left, i.e. -



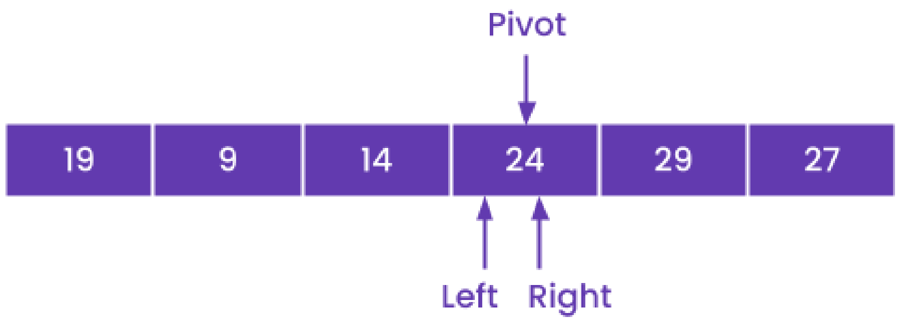
Since, the pivot is at left, so the algorithm starts from right, and moves to left. Now, a[left] = 24, a[right] = 29, and a[pivot] = 24. As a[pivot] < a[right], so algorithm moves one position to left, as -



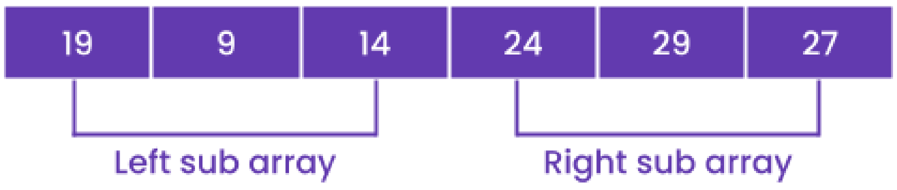
Now, a[pivot] = 24, a[left] = 24, and a[right] = 14. As a[pivot] > a[right], so, swap a[pivot] and a[right], now pivot is at right, i.e. -



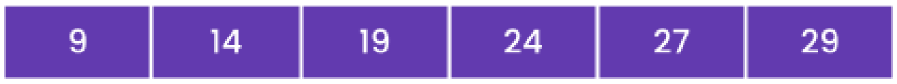
Now, a[pivot] = 24, a[left] = 14, and a[right] = 24. Pivot is at right, so the algorithm starts from left and moves to right.



Now, a[pivot] = 24, a[left] = 24, and a[right] = 24. So, pivot, left and right are pointing to 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 on the right side of element 24 are greater than it, and the elements that are on the 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 -



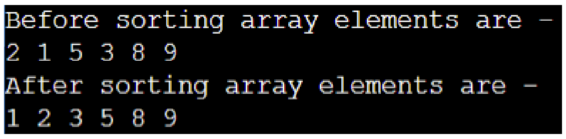
Quick sort time and space complexity analysis :

- **Best Case Complexity -** In Quicksort, the best-case occurs when the pivot element is the middle element or near to the middle element. The best-case time complexity of quicksort is O(n*logn).
- **Average Case Complexity -** It occurs when the array elements are in jumbled order that is not properly ascending and not properly descending. The average case time complexity of quicksort is O(n*logn).
- **Worst Case Complexity -** In quicksort, the worst case occurs when the pivot element is either the greatest or smallest element. Suppose, if the pivot element is always the last element of the array, the worst case would occur when the given array is sorted already in ascending or descending order. The worst-case time complexity of quicksort is O(n2).

NOTE : Worst case in quicksort can be avoided by choosing the right pivot element.
The space complexity of quicksort is O(n*logn).

Code : LP_Code2.java

output :



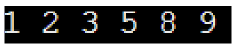
```
Before sorting array elements are -
2 1 5 3 8 9
After sorting array elements are -
1 2 3 5 8 9
```

Randomized quicksort algorithm :

The worst case time complexity comes out to be O(n2) which happens because of wrongly chosen pivot elements in the worst case scenario, which fails to partition the array in a balanced way. If we somehow choose a pivot element that tends to keep the array partitioning balanced, then we can claim the worst case time complexity to be O(n * log n). So we try to randomly choose our pivot element.
We have a built-in pseudo random number method in java which would help in selecting the random index, which gives us the element about which the partition is to be done.

Code : LP_Code3.java

output :



```
1 2 3 5 8 9
```

Next class teaser:

- Greedy Algorithms