Assignment No. 3

Aim: Sort a given set of n integer elements using Quick Sort method and compute its time complexity. Run the program for varied values of n and record the time taken to sort. The elements can be read from a user or can be generated using the random number generator. Demonstrate using C++ how the divide and conquer method works along with its time complexity analysis: worst case, average case and best case.

Objectives:
1. To apply Quick sort method and compute its time complexity.
2. To apply Divide and conquer strategy and find Best case, Worst case and Average case complexity.

Theory:

- **Sorting:**

Sorting is the process of organizing elements in a structured manner. Quicksort is one of the most popular sorting algorithms that use **n*log n** comparisons to sort an array of n elements in a typical situation. Quicksort is based on the divide-and-conquer strategy. We'll take a look at the Quicksort algorithm in this tutorial and see how it works.

What is Quick Sorting technique?

Quick sort is a highly efficient sorting algorithm and is based on partitioning of array of data into smaller arrays. A large array is partitioned into two arrays one of which holds values smaller than the specified value, say pivot, based on which the partition is made and another array holds values greater than the pivot value. Quicksort partitions an array and then calls itself recursively twice to sort the two resulting subarrays. This algorithm is quite efficient for large-sized data sets as its average and worst-case complexity are **$O(n^2)$,** respectively.

The Idea of Quick sort :

Quicksort is a fast sorting algorithm that works by splitting a large array of data into smaller sub-arrays. This implies that each iteration works by splitting the input into two components, sorting them, and then recombining them. For big datasets, the technique is highly efficient since its average and best-case complexity is O(n*logn). It works by recursively sorting the sub-lists to either side of a given pivot and dynamically shifting elements inside the list around that pivot.

As a result, the Quick Sort method can be summarized in three steps:

Pick: Select an element.

Divide: Split the problem set, move smaller parts to the left of the pivot and larger items to the right.

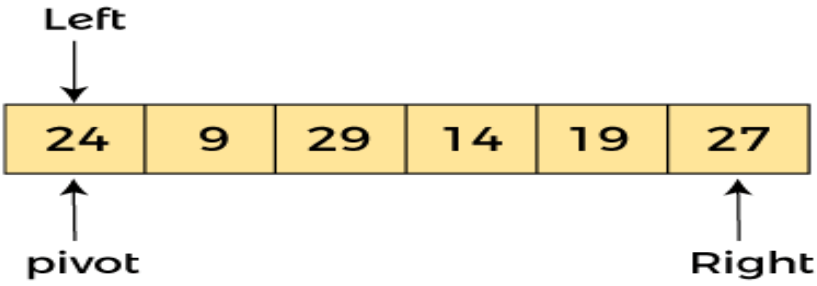
Repeat and combine: Repeat the steps and combine the arrays that have previously been sorted.

Quick Sorting Example:

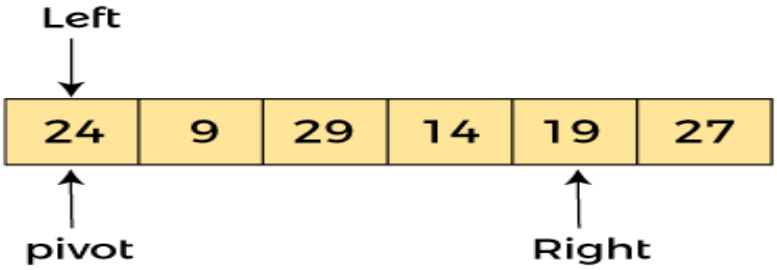
Let the elements of array are:

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.

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

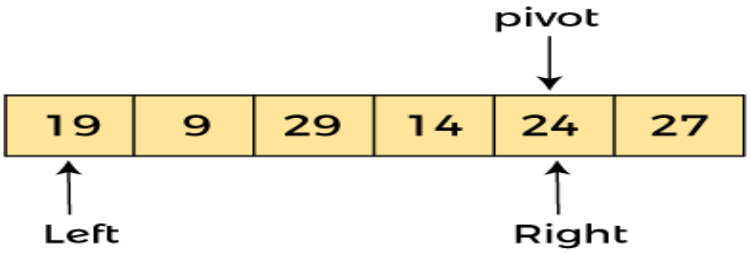


Now, a[pivot] < a[right], so algorithm moves forward one position towards left, i.e.



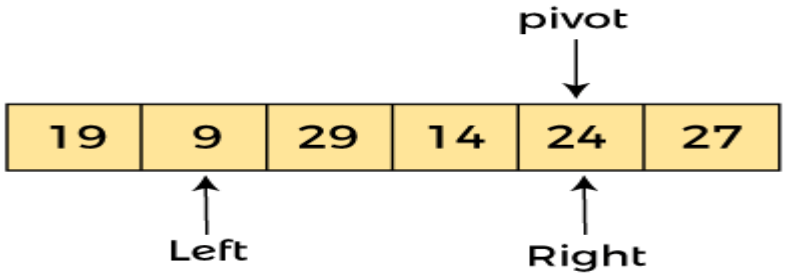
Now, a[left] = 24, a[right] = 19, and a[pivot] = 24.

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

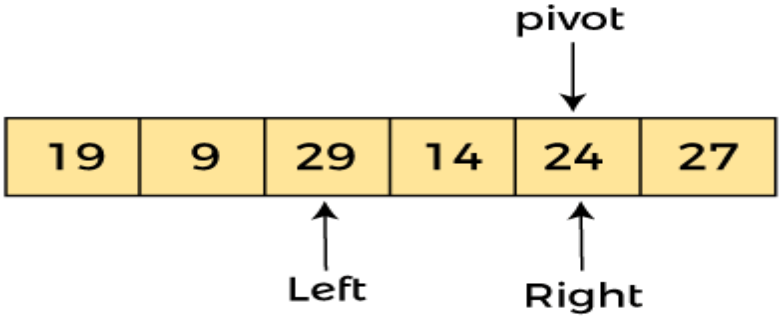


Now, a[left] = 19, a[right] = 24, and a[pivot] = 24. Since, pivot is at right, so algorithm starts from left and moves to right.

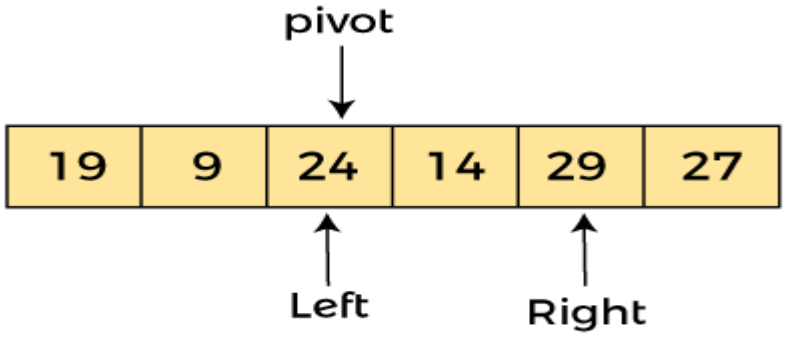
As a[pivot] > a[left], so algorithm moves one position to right as -



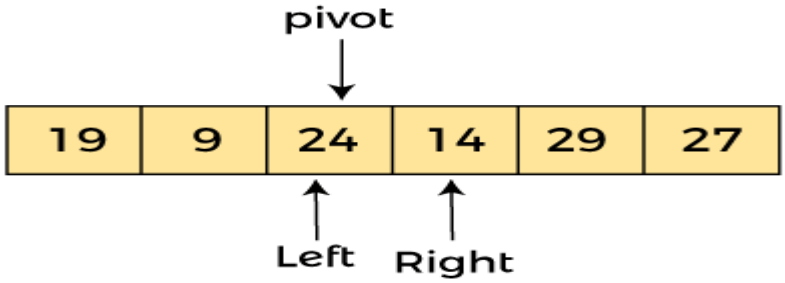
Now, a[left] = 9, a[right] = 24, and a[pivot] = 24. As a[pivot] > a[left], so algorithm moves one position to right as -



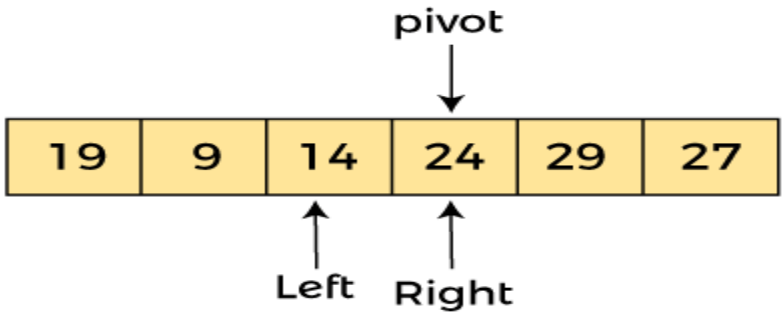
Now, a[left] = 29, a[right] = 24, and a[pivot] = 24. As a[pivot] < a[left], so, swap a[pivot] and a[left], now pivot is at left, i.e. -



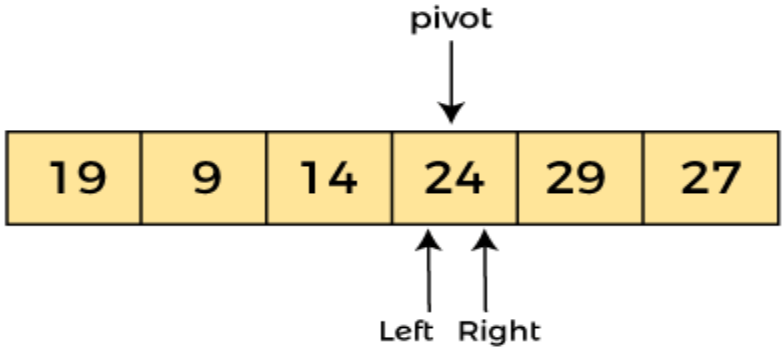
Since, pivot is at left, so algorithm starts from right, and move to left. Now, a[left] = 24, a[right] = 29, and a[pivot] = 24. As a[pivot] < a[right], so algorithm moves one position to left, as -



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. -



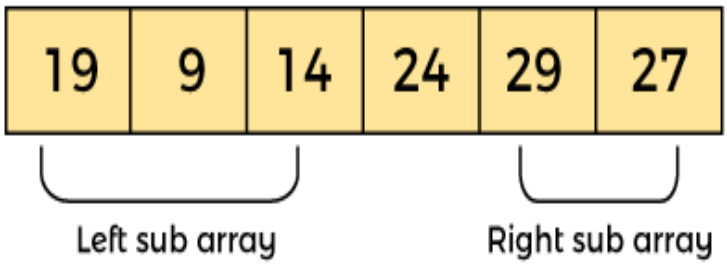
Now, a[pivot] = 24, a[left] = 14, and a[right] = 24. Pivot is at right, so the algorithm starts from left and move to right.



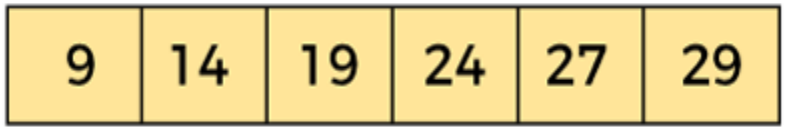
Now, a[pivot] = 24, a[left] = 24, and a[right] = 24. So, pivot, left and right are pointing the same element. It represents the termination of procedure.

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



Time Complexity for the Algorithm:

- **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*log n).**
- **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*log n).**
- **Worst Case Complexity:** In quick sort, worst case occurs when the pivot element is either 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 **$O(n^2)$.**

Program:

Output:

Conclusion: