Quicksort Algorithm

Quicksort is an algorithm based on divide and conquer approach in which the array is split into subarrays and these sub-arrays are recursively called to sort the elements.

How QuickSort Works?

1. A pivot element is chosen from the array. You can choose any element from the array as the pviot element.

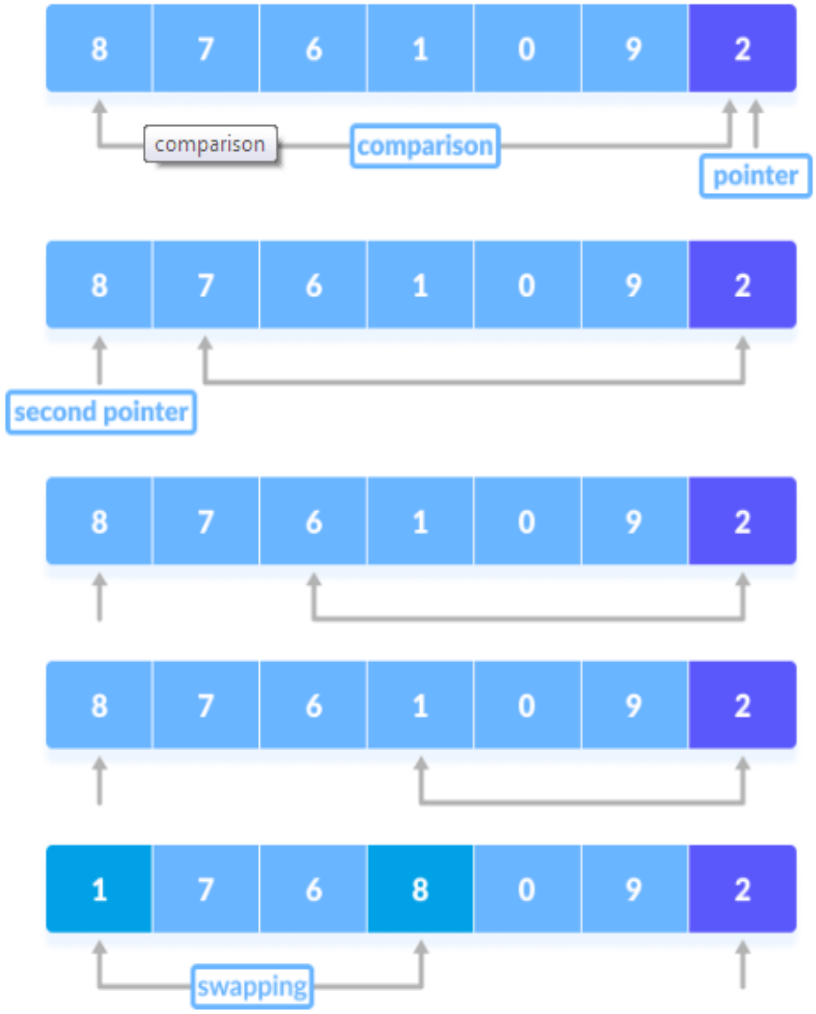
   Here, we have taken the rightmost (ie. the last element) of the array as the pivot element.

2. The elements smaller than the pivot element are put on the left and the elements greater than the pivot element are put on the right.

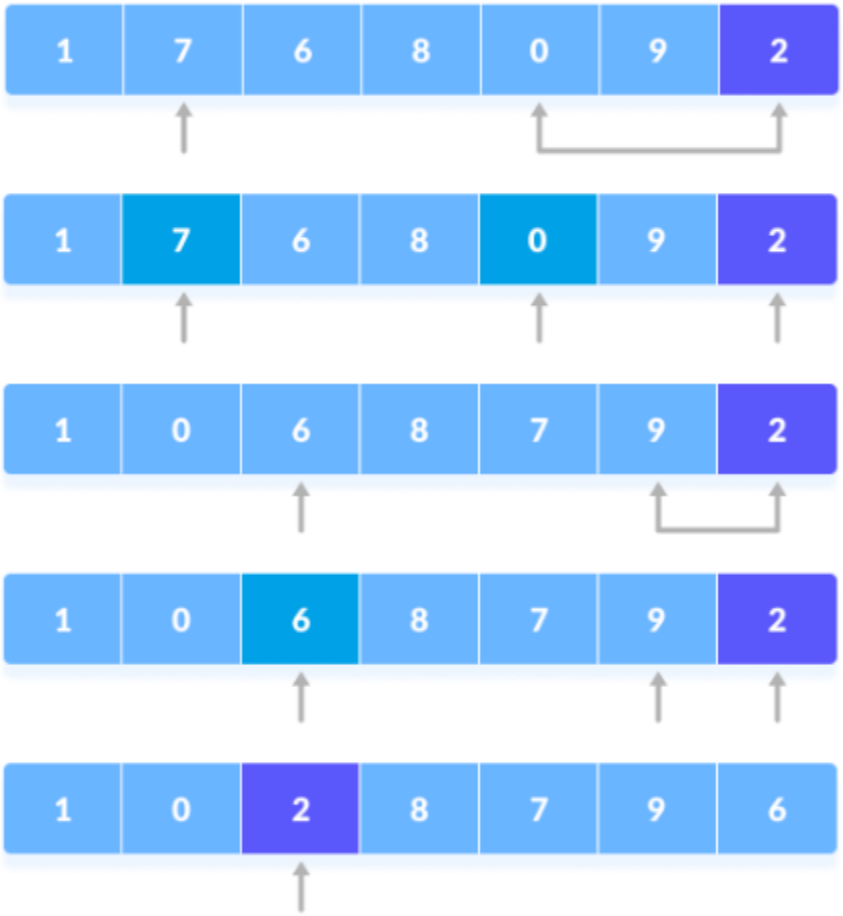
The above arrangement is achieved by the following steps.

a) A pointer is fixed at the pivot element. The pivot element is compared with the elements beginning from the first index. If the element greater than the pivot element is reached, a second pointer is set for that element.

b) Now, the pivot element is compared with the other elements. If element smaller than the pivot element is reached, the smaller element is swapped with the greater element found earlier.



c) The process goes on until the second last element is reached.
d) Finally, the pivot element is swapped with the second pointer.



3. Pivot elements are again chosen for the left and the right sub-parts separately. Within these sub-parts, the pivot elements are placed at their right position. Then, step 2 is repeated

4. The sub-parts are again divided into smallest sub-parts until each subpart is formed of a single element.
5. At this point, the array is already sorted.

Quick Sort Algorithm

```
1.  quickSort(array, leftmostIndex, rightmostIndex)
2.    if (leftmostIndex < rightmostIndex)
3.       pivotIndex <- partition(array,leftmostIndex, rightmostIndex)
4.       quickSort(array, leftmostIndex, pivotIndex)
5.       quickSort(array, pivotIndex + 1, rightmostIndex)
6.
7.  partition(array, leftmostIndex, rightmostIndex)
8.    set rightmostIndex as pivotIndex
9.    storeIndex <- leftmostIndex - 1
10.   for i <- leftmostIndex + 1 to rightmostIndex
11.      if element[i] < pivotElement
12.         swap element[i] and element[storeIndex]
13.         storeIndex++
14.   swap pivotElement and element[storeIndex+1]
15. return storeIndex + 1
```

Complexity

Time Complexities

Worst Case Complexity [Big-O]: O(n2)

It occurs when the pivot element picked is always either the greatest or the smallest element.
In the above algorithm, if the array is in descending order, the partition algorithm always picks the smallest element as a pivot element.

Best Case Complexity [Big-omega]: O(n*log n)

It occurs when the pivot element is always the middle element or near to the middle element.

Average Case Complexity [Big-theta]: O(n*log n)

It occurs when the above conditions do not occur.

Space Complexity

The space complexity for quicksort is O(n*log n).

Quicksort Applications

Quicksort is implemented when

- the programming language is good for recursion
- time complexity matters
- space complexity matters