Programiz

Quicksort Algorithm

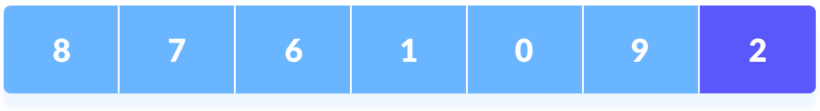
In this tutorial, you will learn how quicksort works. Also, you will find working examples of quicksort in C, C++ Python and Java.

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.

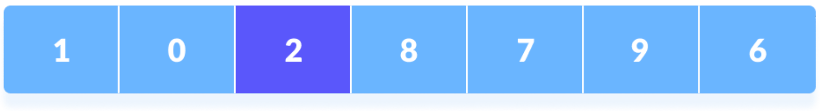
How QuickSort Works?

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.

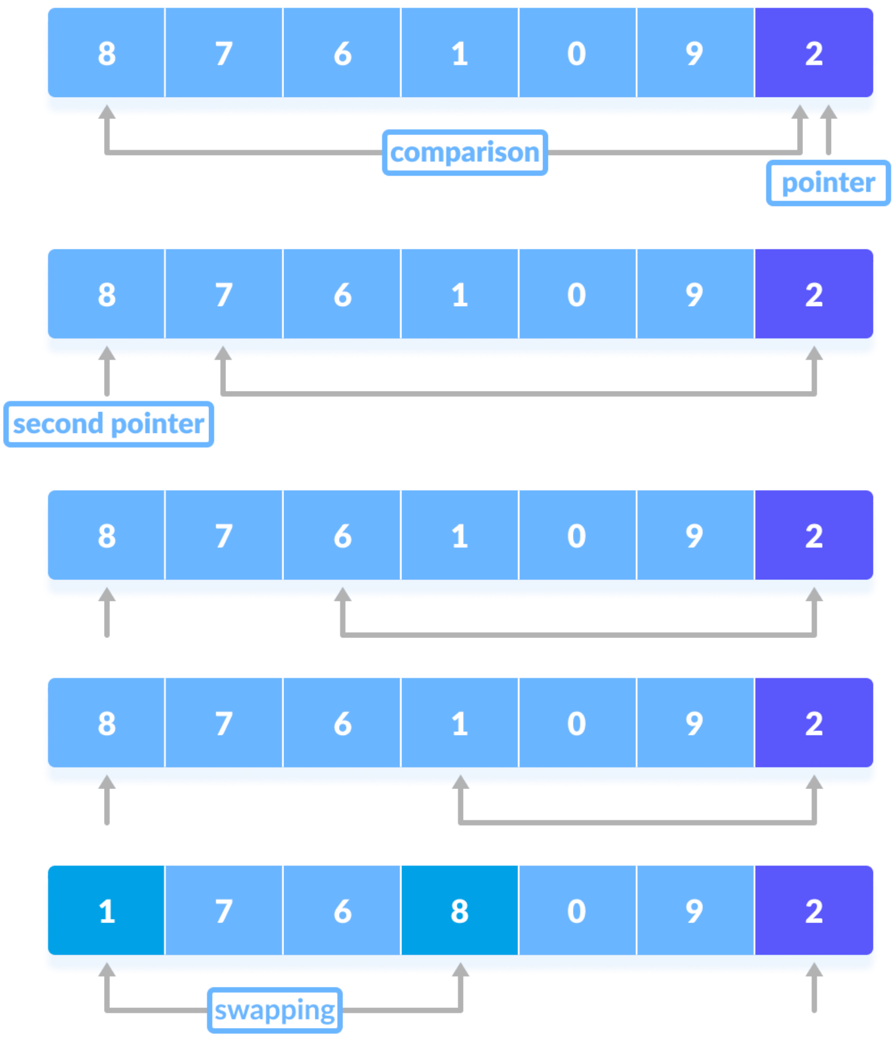


   The above arrangement is achieve ing steps.

Contents

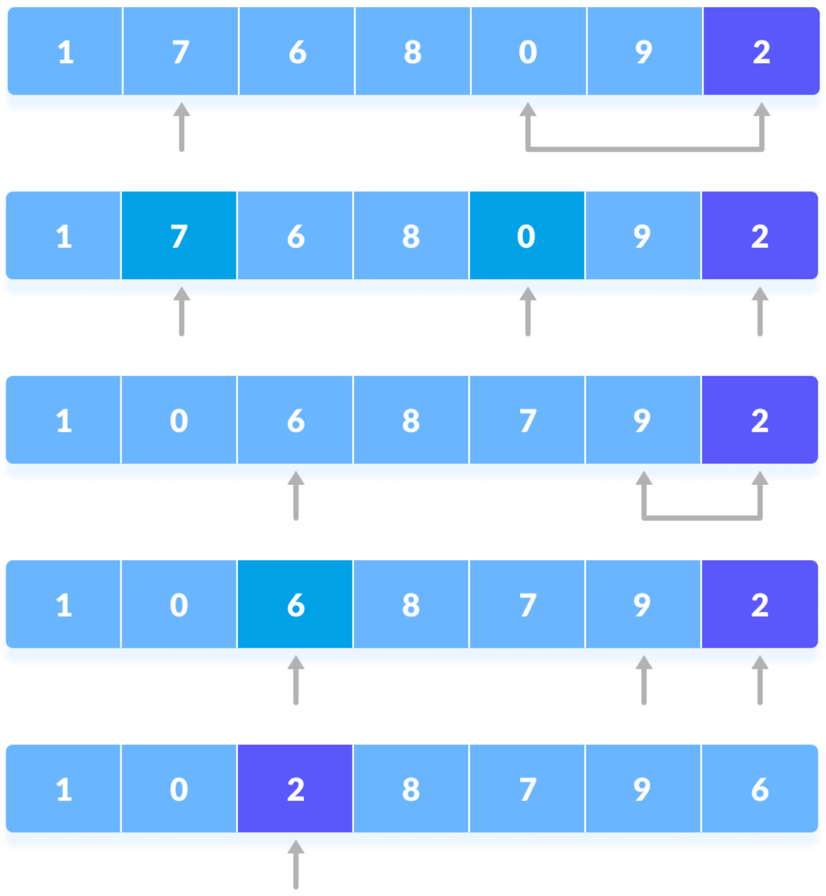
a. A pointer is fixed at the pivot element. The pivot element is compared with the elements beginning from the first index. If the element greater than the pivot element is reached, a second pointer is set for that element.

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.



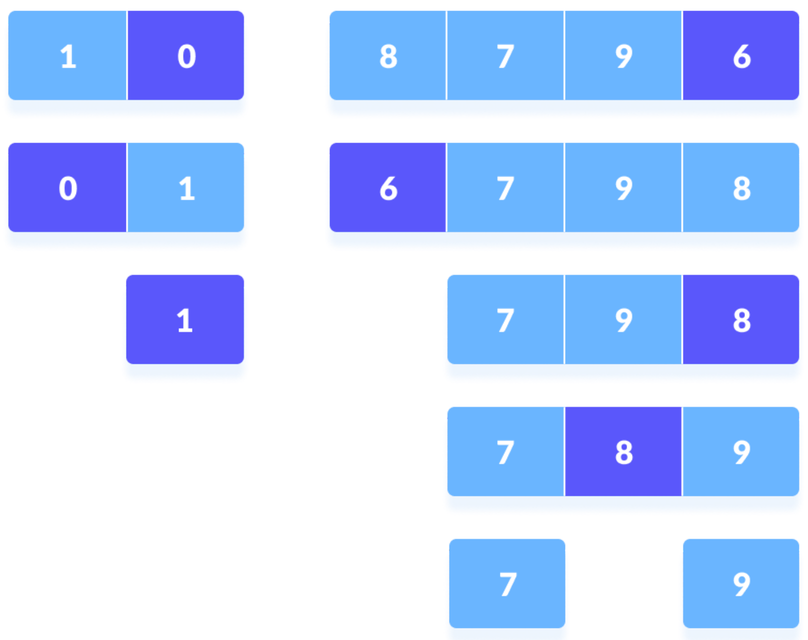
c. The process goes on until the second last element is reached.

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.

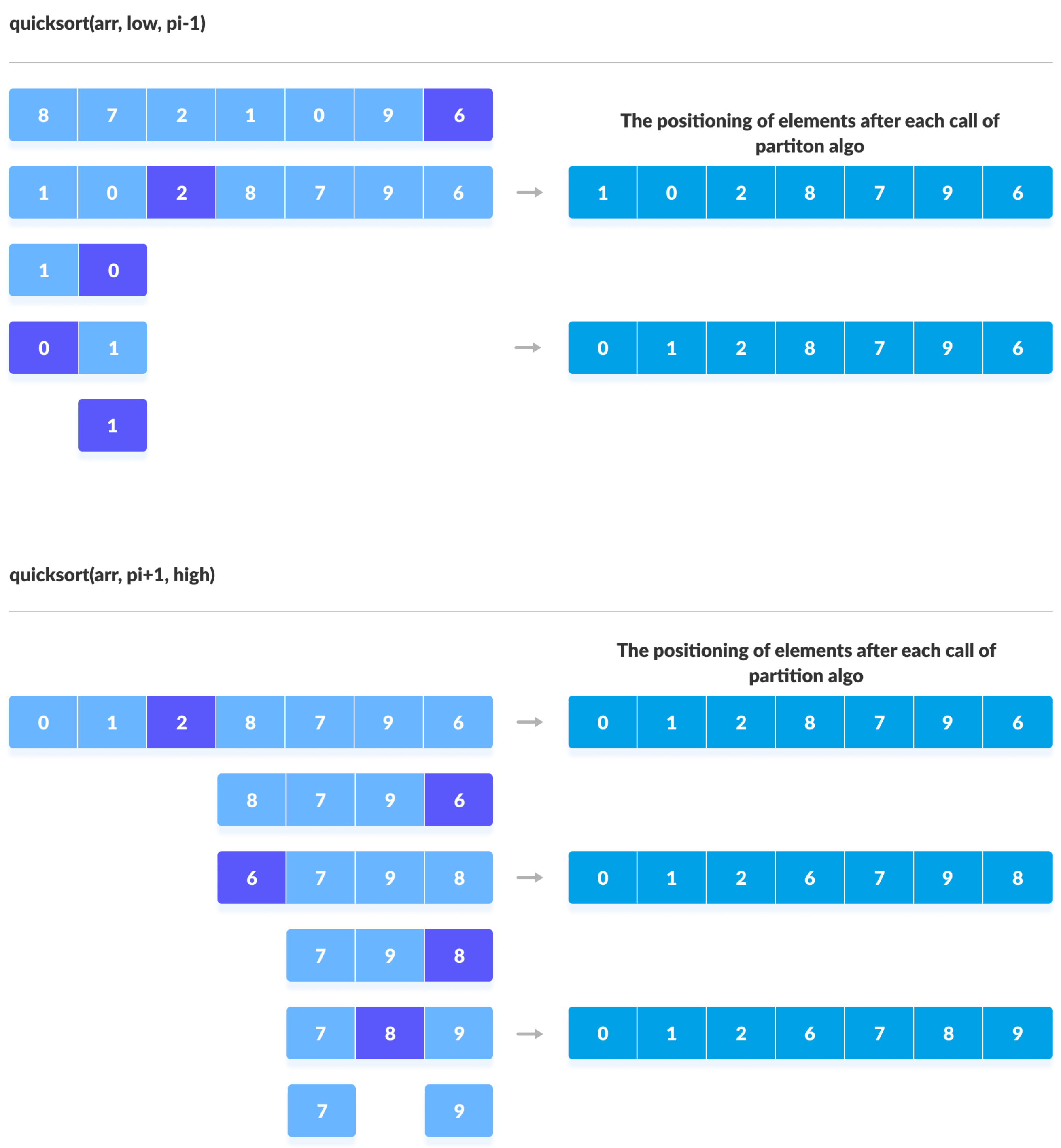
5. At this point, the array is already sorted.

Quicksort uses recursion for sorting the sub-parts.

On the basis of Divide and conquer approach, quicksort algorithm can be explained as:

- **Divide**
  The array is divided into subparts taking pivot as the partitioning point. The elements smaller than the pivot are placed to the left of the pivot and the elements greater than the pivot are placed to the right.
- **Conquer**
  The left and the right subparts are again partitioned using the by selecting pivot elements for them. This can be achieved by recursively passing the subparts into the algorithm.
- **Combine**
  This step does not play a significant role in quicksort. The array is already sorted at the end of the conquer step.

You can understand the working of quicksort with the help of an example/illustration below.


quicksort(arr, low, pi-1)
The positioning of elements after each call of partiton algo
quicksort(arr, pi+1, high)
The positioning of elements after each call of partition algo

Quick Sort Algorithm

```
quickSort(array, leftmostIndex, rightmostIndex)
  if (leftmostIndex < rightmostInd
    pivotIndex <- partition(array,l                ightmostIndex)
    quickSort(array, leftmostIndex, pivotIndex)
```

Contents

```
    quickSort(array, pivotIndex + 1, rightmostIndex)

partition(array, leftmostIndex, rightmostIndex)
  set rightmostIndex as pivotIndex
  storeIndex <- leftmostIndex - 1
  for i <- leftmostIndex + 1 to rightmostIndex
    if element[i] < pivotElement
      swap element[i] and element[storeIndex]
      storeIndex++
  swap pivotElement and element[storeIndex+1]
return storeIndex + 1
```

Python, Java and C/C++ Examples

Python Java C C+

```python
# Quick sort in Python

def partition(array, low, high):
  pivot = array[high]
  i = low - 1

  for j in range(low, high):
    if array[j] <= pivot:
      i = i + 1
      (array[i], array[j]) = (array[j], array[i])

  (array[i + 1], array[high]) = (array[high], array[i + 1])
  return i + 1


def quickSort(array, low, high):
  if low < high:
    pi = partition(array, low, high)
    quickSort(array, low, pi - 1)
    quickSort(array, pi + 1, high)


data = [8, 7, 2, 1, 0, 9, 6]
size = len(data)
quickSort(data, 0, size - 1)
print('Sorted Array in Ascending Order:')
print(data)
```

Complexity

Contents

Time Complexities

- **Worst Case Complexity [Big-O]**: `O(n`^{2}`)`

  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

Data Structure & Algorithms

AVL Tree

Spanning Tree and Minimum Spanning Tree

Huffman Coding

Longest Common Subsequence

Master Theorem

Divide and Conquer Algorithm

Binary Search

Floyd-Warshall Algorithm

Strongly Connected Components

TUTORIALS

Python Tutorials

C Tutorials

Java Tutorials

Kotlin Tutorials

C++ Tutorials

Swift Tutorials

R Tutorials

DSA

Contents