



Quick Sort

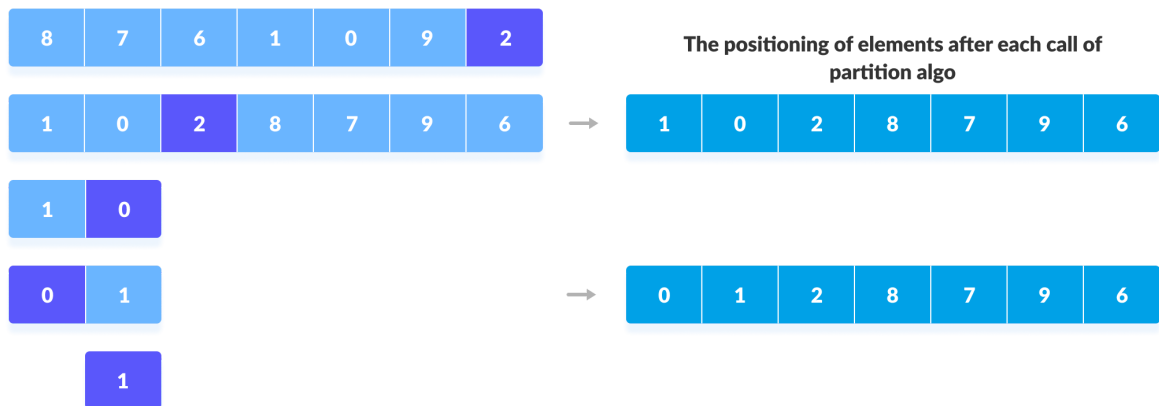


QuickSort is a sorting algorithm based on the Divide and Conquer that picks an element as a pivot and partitions the given array around the picked pivot by placing the pivot in its correct position in the sorted array.

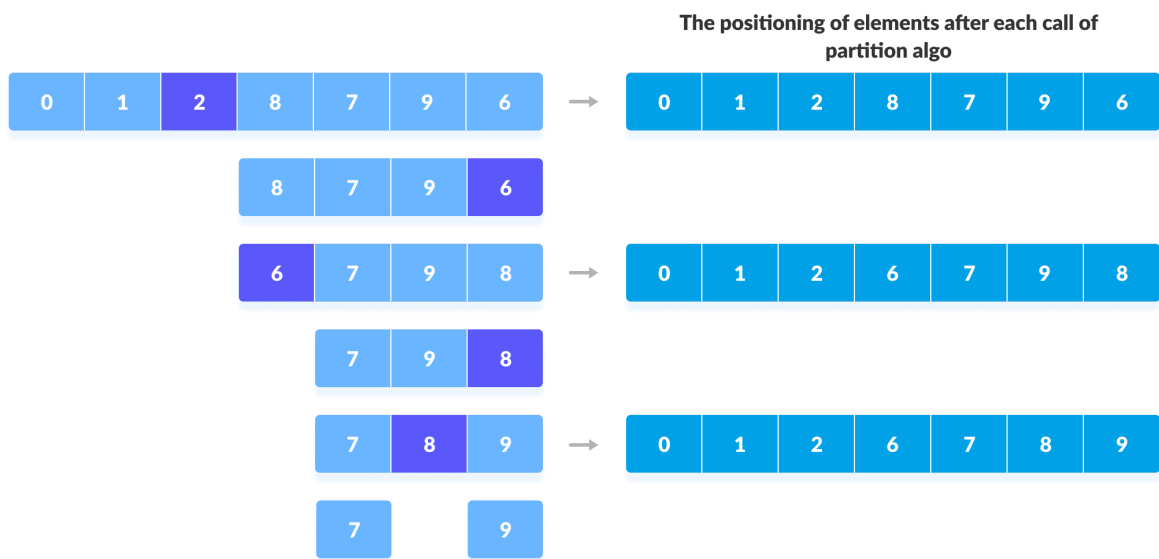
Main Steps of The Algorithm

1. **Choose a Pivot:** Select an element from the array as the pivot. The choice of pivot can vary (e.g., first element, last element, random element, or median).
2. **Partition the Array:** Rearrange the array around the pivot. After partitioning, all elements smaller than the pivot will be on its left, and all elements greater than the pivot will be on its right. The pivot is then in its correct position, and we obtain the index of the pivot.
3. **Recursively Call:** Recursively apply the same process to the two partitioned sub-arrays (left and right of the pivot).
4. **Base Case:** The recursion stops when there is only one element left in the sub-array, as a single element is already sorted.

quicksort(arr, low, pi-1)



quicksort(arr, pi+1, high)



Choice of Pivot

- Always pick the first or last element as a pivot. The below implementation picks the last element as pivot. The problem with this approach is it ends up in the worst case when array is already sorted.
- Pick a random element as a pivot. This is a preferred approach because it does not have a pattern for which the worst case happens.
- Pick the median element as pivot. This is an ideal approach in terms of time complexity as we can find median in linear time and the partition function will always divide the input array into two halves. But it takes more time on average as median finding has high constants.

Partition Algorithm

The key process in **quickSort** is a **partition()**. There are three common algorithms to partition. All these algorithms have $O(n)$ time complexity.

- Naive Partition: Here we create copy of the array. First put all smaller elements and then all greater. Finally we copy the temporary array back to original array. This requires $O(n)$ extra space.
- Lomuto Partition: We have used this partition in this article. This is a simple algorithm, we keep track of index of smaller elements and keep swapping. We have used it here in this article because of its simplicity.
- Hoare's Partition: This is the fastest of all. Here we traverse array from both sides and keep swapping greater element on left with smaller on right while the array is not partitioned.

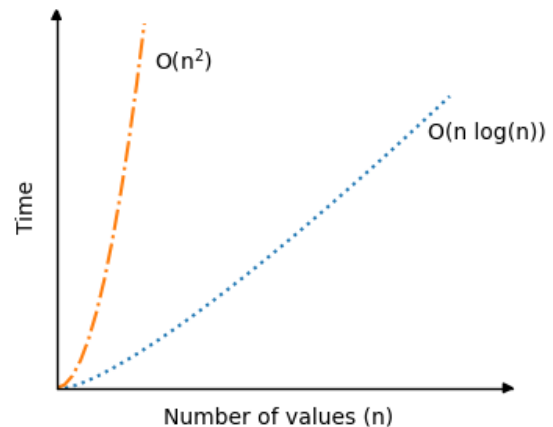
Pseudocode

```
quickSort(array, leftmostIndex, rightmostIndex)
  if (leftmostIndex < rightmostIndex)
    pivotIndex <- partition(array, leftmostIndex, rightmostIndex)
    quickSort(array, leftmostIndex, pivotIndex - 1)
    quickSort(array, pivotIndex, 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
```

Analysis

- **Best Case:** ($\Omega(n \log n)$), Occurs when the pivot element divides the array into two equal halves.
- **Average Case** ($\theta(n \log n)$), On average, the pivot divides the array into two parts, but not necessarily equal.
- **Worst Case:** ($O(n^2)$), Occurs when the smallest or largest element is always chosen as the pivot (e.g., sorted arrays).



Advantages of Quick Sort

- It is a divide-and-conquer algorithm that makes it easier to solve problems.
- It is efficient on large data sets.
- It has a low overhead, as it only requires a small amount of memory to function.
- It is Cache Friendly as we work on the same array to sort and do not copy data to any auxiliary array.
- Fastest general purpose algorithm for large data when stability is not required.
- It is tail recursive and hence all the tail call optimization can be done.

Disadvantages of Quick Sort

- It has a worst-case time complexity of $O(n^2)$, which occurs when the pivot is chosen poorly.
- It is not a good choice for small data sets.
- It is not a stable sort, meaning that if two elements have the same key, their relative order will not be preserved in the sorted output in case of quick sort, because here we are swapping elements according to the pivot's position (without considering their original positions).

Applications of Quick Sort

- Efficient for sorting large datasets with $O(n \log n)$ average-case time complexity.
- Used in partitioning problems like finding the kth smallest element or dividing arrays by pivot.
- Integral to randomized algorithms, offering better performance than deterministic approaches.
- Applied in cryptography for generating random permutations and unpredictable encryption keys.
- Partitioning step can be parallelized for improved performance in multi-core or distributed systems.
- Important in theoretical computer science for analyzing average-case complexity and developing new techniques.



Java implementation can be found under Implementation_Java folder

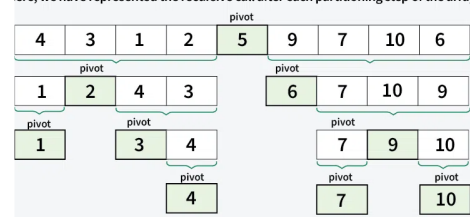
References

Quick Sort - GeeksforGeeks

QuickSort is a divide-and-conquer sorting algorithm that selects a pivot, partitions the array around it, and recursively sorts the resulting sub-arrays.

🔗 <https://www.geeksforgeeks.org/quick-sort-algorithm/>

Here, we have represented the recursive call after each partitioning step of the array.




QuickSort (With Code in Python/C++/Java/C)

Quicksort is an algorithm based on divide and conquer approach in which an array is split into sub-arrays and these sub arrays are recursively sorted to get a sorted array. In this tutorial, you will understand the working of quickSort with working code in C, C++, Java, and Python.

🔗 <https://www.programiz.com/dsa/quick-sort>

W3Schools.com

W3Schools offers free online tutorials, references and exercises in all the major languages of the web. Covering popular subjects like HTML, CSS, JavaScript, Python, SQL,

 https://www.w3schools.com/dsa/dsa_algo_quicksort.php



 **Author → Serhat Kumas**

<https://www.linkedin.com/in/serhatkumas/>

SerhatKumas - Overview

Computer engineering student who loves coding in different fields instead of focusing on a one specific area. -

SerhatKumas

 <https://github.com/SerhatKumas>

