Let's say the input array to be sorted is [5, 2, 4, 7, 1, 3].

1. The first step is to choose a pivot element. In this case, we will choose the middle element, which is 4.
2. We then partition the array around the pivot such that all elements smaller than the pivot are on the left of the pivot and all elements greater than the pivot are on the right of the pivot. This gives us two subarrays [5, 2, 1] and [7, 3].
3. We then recursively sort the left and right subarrays. The left subarray will be sorted to [1, 2, 5], and the right subarray will be sorted to [3, 7].
4. Finally, we combine the two sorted subarrays to get the sorted array [1, 2, 3, 4, 5, 7].

Time complexity

The time complexity of quick sort is O(n log n) in the average case and O(n^2) in the worst case. The worst case occurs when the pivot element is always the smallest or the largest in the array.

Advantages

- Quick sort is a very efficient sorting algorithm for large data sets.
- Easy to understand and implement.
- A recursive algorithm, which makes it easy to parallelize.

Disadvantages

- The worst-case time complexity of quick sort is O(n^2).
- Not a stable sorting technique, (i.e.) the relative order of elements with equal keys is not preserved.

Not a good choice for sorting small data sets.