

Step by Step Process

In Quick sort algorithm, partitioning of the list is performed using following steps...

- **Step 1** - Consider the first element of the list as **pivot** (i.e., Element at first position in the list).
- **Step 2** - Define two variables i and j. Set i and j to first and last elements of the list respectively.
- **Step 3** - Increment i until $\text{list}[i] \leq \text{pivot}$ then stop.
- **Step 4** - Decrement j until $\text{list}[j] > \text{pivot}$ then stop.
- **Step 5** - If $i < j$ then exchange $\text{list}[i]$ and $\text{list}[j]$.
- **Step 6** - Repeat steps 3,4 & 5 until $i > j$.
- **Step 7** - Exchange the pivot element with $\text{list}[j]$ element.

Following is the sample code for Quick sort...

Quick Sort Logic

```
//Quick Sort Logic
void quickSort(int list[10],int first,int last){
    int pivot,i,j,temp;

    if(first < last){
        pivot = first;
        i = first;
        j = last;

        while(i < j){
            while(list[i] <= list[pivot] && i < last)
                i++;
            while(j>pivot && list[j] > list[pivot])
                j--;
            if(i < j){
                temp = list[i];
                list[i] = list[j];
                list[j] = temp;
            }
        }

        temp = list[pivot];
        list[pivot] = list[j];
```

}

Consider the following unsorted list of elements...

List 5 3 8 1 4 6 2 7

Define pivot, left & right. Set pivot = 0, left = 1 & right = 7. Here '7' indicates 'size-1'.

Diagram illustrating the partitioning step of Quicksort. The list is [5, 3, 8, 1, 4, 6, 2, 7]. The pivot is 5. Elements less than the pivot (3, 1, 2) are moved to its left, and elements greater than the pivot (8, 4, 6, 7) are moved to its right.

Compare List[left] with List[pivot]. If **List[left]** is greater than **List[pivot]** then stop left otherwise move left to the next.

Compare List[right] with List[pivot]. If List[right] is smaller than List[pivot] then stop right otherwise move right to the previous.

Repeat the same until **left** >= **right**.

If both left & right are stopped but $left < right$ then swap $List[left]$ with $List[right]$ and continue the process.

If $\text{left} \geq \text{right}$ then swap $\text{List}[\text{pivot}]$ with $\text{List}[\text{right}]$.

Diagram illustrating the initial partitioning step of the Quick Sort algorithm. The list is [5, 3, 8, 1, 4, 6, 2, 7]. The pivot is 5. Elements less than the pivot (3, 2) are in the left partition. Elements greater than the pivot (8, 1, 4, 6, 7) are in the right partition. The pivot 5 is in its final sorted position.

Compare `List[left] < List[pivot]` as it is true increment left by one and repeat the same, left will stop at 8.

Compare List[right]>List[pivot] as it is true decrement right by one and repeat the same, right will stop at 2.

Diagram illustrating the initial partitioning step of the Quick Sort algorithm. The list is [5, 3, 8, 1, 4, 6, 2, 7]. The pivot is 5. Elements less than the pivot (3, 1, 2) are in the left partition, and elements greater than the pivot (8, 4, 6, 7) are in the right partition.

Here left & right both are stoped and left is not greater than right so we need to swap List[left] and List[right]

Diagram illustrating the partitioning step of Quicksort. The list is [5, 3, 2, 1, 4, 6, 8, 7]. The pivot is 5. Elements less than the pivot (3, 2, 1) are placed to the left, and elements greater than the pivot (4, 6, 8, 7) are placed to the right.

Compare `List[left] < List[pivot]` as it is true increment left by one and repeat the same, left will stop at 6.

Compare `List[right]>List[pivot]` as it is true decrement right by one and repeat the same, right will stop at 4.

Diagram illustrating the initial state of the array [5, 3, 2, 1, 4, 6, 8, 7] for the partitioning step. The pivot is 5. The right pointer is at index 4 (value 4), and the left pointer is at index 6 (value 8).

Here left & right both are stopped and left is greater than right so we need to swap List[pivot] and List[right]

List	4	3	2	1	5	6	8	7
------	---	---	---	---	---	---	---	---

Here we can observe that all the numbers to the left side of 5 are smaller and right side are greater. That means 5 is placed in its correct position.

Repeat the same process on the left sublist and right sublist to the number 5.

Diagram illustrating the partitioning step of Quicksort. The list is [4, 3, 2, 1, 5, 6, 8, 7]. The pivot is 5. Elements less than the pivot (4, 3, 2, 1) are moved to its left, and elements greater than the pivot (6, 8, 7) are moved to its right.

In the left sublist as there are no smaller number than the pivot left will keep on moving to the next and stops at last number. As the List[right] is smaller, right stops at same position. Now left and right both are equal so we swap pivot with right.

Diagram illustrating the partitioning step of Quicksort. The list is [1, 3, 2, 4, 5, 6, 8, 7]. The pivot is 5. Elements less than the pivot (1, 3, 2) are to its left, and elements greater than the pivot (6, 8, 7) are to its right.

In the right sublist left is greater than the pivot, left will stop at same position.

As the List[right] is greater than List[pivot], right moves towards left and stops at pivot number position.

Now left > right so we swap pivot with right. (6 is swap by itself).

Diagram illustrating the initial state of the list [1, 3, 2, 4, 5, 6, 8, 7] with pivot 4. The pivot is highlighted in yellow. Elements less than the pivot (1, 3, 2) are in the left partition, and elements greater than the pivot (5, 6, 8, 7) are in the right partition.

Repeat the same recursively on both left and right sublists until all the numbers are sorted.

The final sorted list will be as follows

List	1	2	3	4	5	6	7	8
------	---	---	---	---	---	---	---	---

Complexity of the Quick Sort Algorithm

To sort an unsorted list with '**n**' number of elements, we need to make $((n-1)+(n-2)+(n-3)+\dots+1) = (n(n-1))/2$ number of comparisons in the worst case. If the list is already sorted, then it requires '**n**' number of comparisons.

Worst-Case: $O(n^2)$

Best-Case: $O(n \log n)$

Average Case : $O(n \log n)$