https://www.javatpoint.com/quick-sort#:~:text=Time%20Complexity,-Case&text=The%20best%2Dcase%20time%20complexity,O(n*logn).

Quick Sort

Based on the **Divide-and-Conquer** approach, the quicksort algorithm can be explained as:

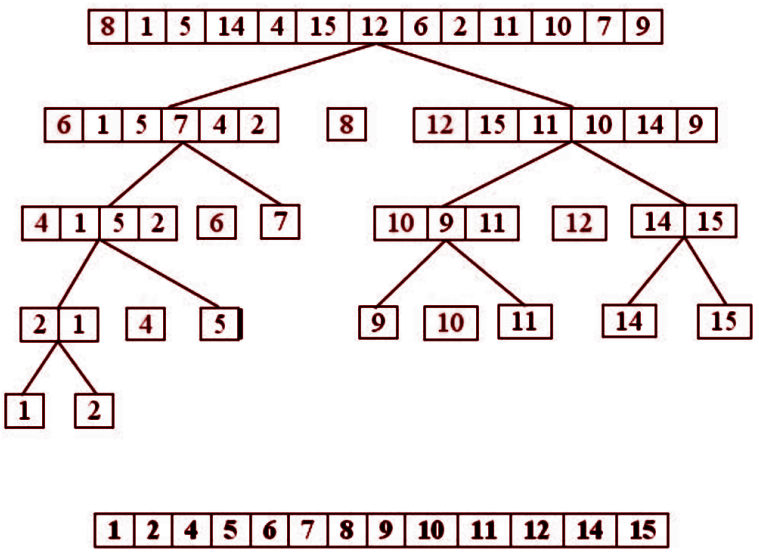
- **Divide:** The array is divided into subparts taking pivot as the partitioning point. The elements **smaller** than the pivot are placed to the **left** of the pivot and the elements **greater** than the pivot are placed to the **right** side.
- **Conquer:** The left and right sub-parts are again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

The advantage of quicksort over merge sort is that it does not require any extra space to sort the given array.

There are many ways to pick a pivot element:

1. Always pick the **first** element as the pivot.
2. Always pick the **last** element as the pivot.
3. Pick a **random** element as the pivot.
4. Pick the **middle** element as the pivot.

The following diagram shows the complete quick sort process, by considering the first element as the pivot element, for an example array [8,1,5,14,4,15,12,6,2,11,10,7,9].



- In step 1, 8 is taken as the pivot.
- In step 2, 6 and 12 are taken as pivots.
- In step 3, 4 and 10 are taken as pivots.
- We keep dividing the list about pivots till there are only 2 elements left in a sublist.

Pseudocode:

```
/*
      array from lo(0) to hi(arr.length-1) is considered
*/
function quickSort(arr, lo , hi )

     if (lo >= hi)
          return

     pivot = arr[lo]

     //   partitioning
     left = lo
     right = hi

     while left <= right

          //   move left to a problem
          while  arr[left] < pivot
                    left++


          //   move right to a problem
          while  arr[right] > pivot
                    right--


          //   problem solve : swap
          if  left <= right
                    temp = arr[left]
                    arr[left] = arr[right]
                    arr[right] = temp
                    left++
                    right--


      //   smaller problems
      quickSort(arr, lo, right)
      quickSort(arr, left, hi)
```

Time complexity: O(N^2), in the worst case.
But as this is a randomized algorithm, its time complexity fluctuates between $O(N^2)$ and O(N*log N) and mostly it comes out to be O(N*logN).

Space complexity: O(1), as no extra space is required.