QuickSort Algorithm

Quicksort

- 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.

- **Like merge sort, this algorithm works by using a divide-and-conquer strategy to divide a single unsorted array into two smaller sub-arrays.**

The quick sort algorithm works as follows:

- 1. Select an element pivot from the array elements.
- 2. Rearrange the elements in the array in such a way that all elements that are less than the pivot appear before the pivot and all elements greater than the pivot element come after it (equal values can go either way). After such a partitioning, the pivot is placed in its final position. This is called the partition operation.
- 3. Recursively sort the two sub-arrays thus obtained. (One with sub-list of values smaller than that of the pivot element and the other having higher value elements.)

```
1.  quickSort(array, leftmostIndex, rightmostIndex)
2.    if (leftmostIndex < rightmostIndex)
3.       pivotIndex <- partition(array,leftmostIndex, rightmostIndex)
4.       quickSort(array, leftmostIndex, pivotIndex)
5.       quickSort(array, pivotIndex + 1, rightmostIndex)
6.
7.  partition(array, leftmostIndex, rightmostIndex)
8.    set rightmostIndex as pivotIndex
9.    storeIndex <- leftmostIndex - 1
10.   for i <- leftmostIndex + 1 to rightmostIndex
11.      if element[i] < pivotElement
12.         swap element[i] and element[storeIndex]
13.         storeIndex++
14.   swap pivotElement and element[storeIndex+1]
15. return storeIndex + 1
```

Complexity

- Time Complexities
- Worst Case Complexity [Big-O]: O(n2)
- 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).

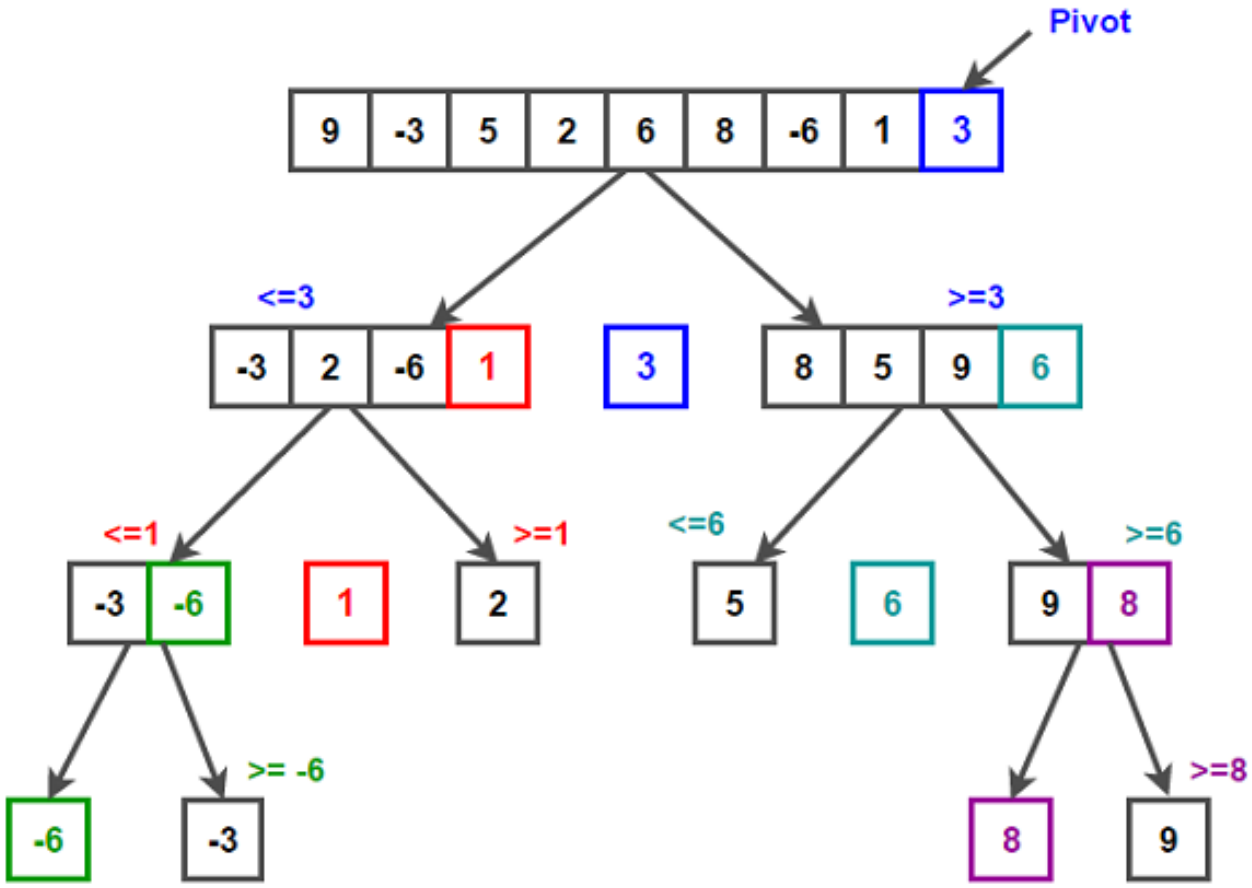
Quicksort Applications

- Quicksort is implemented when
- the programming language is good for recursion
- time complexity matters
- space complexity matters

How Quicksort works?

Quicksort is a divide and conquer algorithm. Like all divide and conquer algorithms, it first divides a large array into two smaller sub-arrays and then recursively sort the sub-arrays. Basically, three steps are involved in whole process –

- **Pivot selection:** *Pick an element, called a pivot, from the array (usually the leftmost or the rightmost element of the partition).*
- **Partitioning:** *Reorder the array so that all elements with values less than the pivot come before the pivot, while all elements with values greater than the pivot come after it (equal values can go either way). After this partitioning, the pivot is in its final position.*
- **Recur:** *Recursively apply the above steps to the sub-array of elements with smaller values than pivot and separately to the sub-array of elements with greater values than pivot.*


Pivot
9 -3 5 2 6 8 -6 1 3
<=3
-3 2 -6 1
3
>=3
8 5 9 6
<=1
-3 -6
1
2
>=1
<=6
5
6
9 8
>=6
>= -6
-6
-3
8
9
>=8