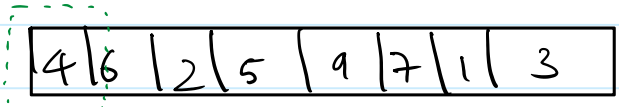
Quick Sort

- a divide and conquer sorting algorithm, it doesn't use any extra space.
- it uses an auxiliary stack space.

- This algorithm is a repetition of these two steps:
  - pick a pivot element and place it in its correct order in sorted array.
  - shift elements < pivot to the left, and > pivot element to the right.

// Approach:

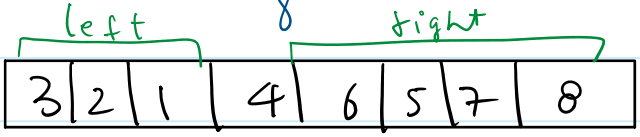
1: choose a pivot element. A pivot element can be any element you choose.



pivot = 4

- place the pivot in correct position, i.e in the 4th position.

2: Shift smaller elements to the left of pivot element and larger element to the right.



Now apply these two steps on the left and right subarray recursively, until the size of sorted array becomes 1.

// Algorithm :

- quickSort() function :

- low points to first index, high points to the last.
- get the index while shifting smaller elements to the left and larger elements to the right using a partition() function.

this index can be called partitioning index.

- after placing pivot at partition index, call the quickSort() for the left and right unsorted sub-array.

```
quickSort (low -> partition-1)   // left part
quickSort (partition+1 -> high)  // right part
```

```
quickSort (arr, low, high) {
    if (low < high) {
        pIndex = partition(arr, low, high);

        quickSort(arr, low, pIndex-1);
        quickSort(arr, pIndex+1, high);
    }
}
```

- partition() function :

- select the pivot element for the range
- take i and j , i as low, j = high.
- i moves forward and finds element > pivot element, j moves backward and find element < pivot element.

  i <= high, j >= low+1
- once we find such element i.e arr[i] > pivot, arr[j] < pivot.

  swap(arr[i], arr[j])

- continue 3 and 4 until $j < i$.
- finally swap pivot element with arr[j], and return j, i.e partition index.

```
partition(arr, low, high)
 pivot = arr[low];
   i = low;
   j = high;
 while (i < j) {
   while (arr[i] <= pivot && i <= high-1) i++;
   while (arr[j] > pivot && j >= low+1) j--;
   if (i < j) swap(arr[i], arr[j]);
 }
 swap(arr[low], arr[j]);
  return j;
```

// Complexities:

time complexity: O(nlogn)
space complexity: O(1) + O(N)