Difficulty: Red **Category:** Sorting **Successful Submissions:** 12,009+

Quick Sort

Write a function that takes in an array of integers and returns a sorted version of that array. Use the Quick Sort algorithm to sort the array.

If you're unfamiliar with Quick Sort, we recommend watching the Conceptual Overview section of this question's video explanation before starting to code.

Sample Input

```
array = [8, 5, 2, 9, 5, 6, 3]
```

Sample Output

```
[2, 3, 5, 5, 6, 8, 9]
```

Hints

Hint 1

Quick Sort works by picking a "pivot" number from an array, positioning every other number in the array in sorted order with respect to the pivot (all smaller numbers to the pivot's left; all bigger numbers to the pivot's right), and then repeating the same two steps on both sides of the pivot until the entire array is sorted.

Hint 2

Pick a random number from the input array (the first number, for instance) and let that number be the pivot. Iterate through the rest of the array using two pointers, one starting at the left extremity of the array and progressively moving to the right, and the other one starting at the right extremity of the array and progressively moving to the left. As you iterate through the array, compare the left and right pointer numbers to the pivot. If the left number is greater than the pivot and the right number is less than the pivot, swap them; this will effectively sort these numbers with respect to the pivot at the end of the iteration. If the left number is ever less than or equal to the pivot, increment the left pointer; similarly, if the right number is ever greater than or equal to the pivot, decrement the right pointer. Do this until the pointers pass each other, at which point swapping the pivot with the right number should position the pivot in its final, sorted position, where every number to its left is smaller and every number to its right is greater.

Hint 3

Repeat the process mentioned in Hint #2 on the respective subarrays located to the left and right of your pivot, and keep on repeating the process thereafter until the input array is fully sorted.

Optimal Space & Time Complexity

Best: O(nlog(n)) time | O(log(n)) space - where n is the length of the input array Average: O(nlog(n)) time | O(log(n)) space - where n is the length of the input array Worst: O(n^2) time | O(log(n)) space - where n is the length of the input array