Merge sort:

function merge (left, right) {

let resultArray = [], leftIndex = 0, rightIndex = 0;

// We will concatenate values into the resultArray in order

while (leftIndex < left.length && rightIndex < right.length) {

if (left[leftIndex] < right[rightIndex]) {

resultArray.push(left[leftIndex]);

leftIndex++; // move left array cursor

} else {

resultArray.push(right[rightIndex]);

rightIndex++; // move right array cursor

}

}

// We need to concat here because there will be one element remaining

// from either left OR the right

return resultArray

.concat(left.slice(leftIndex))

.concat(right.slice(rightIndex));

}

// Merge Sort Implentation (Recursion)

function mergeSort (unsortedArray) {

// No need to sort the array if the array only has one element or empty

if (unsortedArray.length <= 1) {

return unsortedArray;

}

// In order to divide the array in half, we need to figure out the middle

const middle = Math.floor(unsortedArray.length / 2);

// This is where we will be dividing the array into left and right

const left = unsortedArray.slice(0, middle);

const right = unsortedArray.slice(middle);

// Using recursion to combine the left and right

return merge(

mergeSort(left), mergeSort(right)

);

}

Quick Sort :

function quick\_Sort(origArray) {

if (origArray.length <= 1) {

return origArray;

} else {

var left = [];

var right = [];

var newArray = [];

var pivot = origArray.pop();

var length = origArray.length;

for (var i = 0; i < length; i++) {

if (origArray[i] <= pivot) {

left.push(origArray[i]);

} else {

right.push(origArray[i]);

}

}

return newArray.concat(quick\_Sort(left), pivot, quick\_Sort(right));

}

}

var myArray = [3, 0, 2, 5, -1, 4, 1 ];

console.log("Original array: " + myArray);

var sortedArray = quick\_Sort(myArray);

console.log("Sorted array: " + sortedArray);