1.1 O(n)
1.2 O(n)

3.1 Table

Sorting Method	**Best Case**	**Worst Case**
Bubble Sort Non-Recursive	O(n^2)	O(n^2)
Bubble Sort Recursive	O(n^2)	O(n^2)
Selection Sort (Non-recursive)	O(n^2)	O(n^2)
Insertion Sort (Non-recursive)	O(n)	O(n^2)
Merge Sort (Recursive)	O(nlog(n))	O(nlog(n))
Quick Sort (Recursive)	O(nlog(n))	O(n^2)

3.2 Step by step

4	77	98	30	20	50	77	22	49	2

Bubble Sort

for (i < length), for (j < length - i), push the biggest number to the right

4	77	30	20	50	77	22	49	2	98
4	30	20	50	77	22	49	2	77	98
4	20	30	50	22	49	2	77	77	98

... ...

apply same logic to the rest

Selection Sort (Non-recursive)

for (i < length), for (i + 1 < j < length - i), find min and swap with i

2	77	98	30	20	50	77	22	49	4
2	4	98	30	20	50	77	22	49	77
2	4	20	30	98	50	77	22	49	77

... ...

apply same logic to the rest

Insertion Sort (Non-recursive)

for (i < length), while (j > 0 && array[i] < array[j - 1]), insert array[i] and shift array[j - 1] to the left

4	77	98	30	20	50	77	22	49	2
4	77	98	30	20	50	77	22	49	2
4	77	98	30	20	50	77	22	49	2
4	30	77	98	20	50	77	22	49	2
4	20	30	77	98	50	77	22	49	2

... ...

apply same logic to the rest

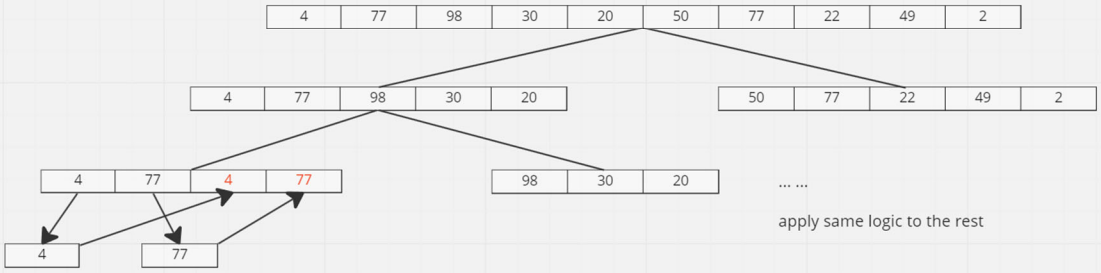
Merge Sort

Push: while (floor < ceiling), mergeSort(array, floor, middle), mergeSort(array, middle + 1, ceiling)

Pop: when base case is reached, merge(left sorted array, right sorted array)

***merge:** while (leftSortedArray.length > left Index && rightSortedArray.length > right Index), compare and put the smaller one to newArray[i]

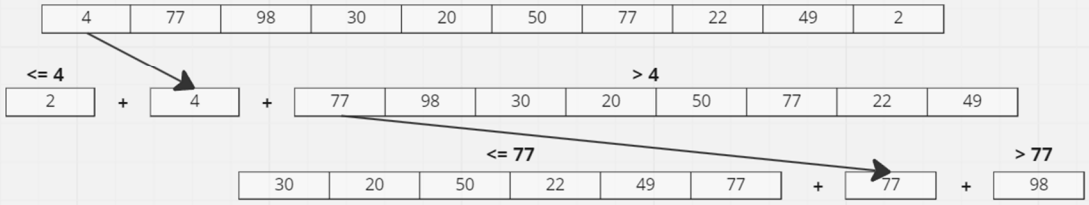
attach the rest of leftSortedArray/rightSortedArray to the end of newArray



Quick Sort

Push: index = Partition (array, low, high), quickSort(array, low, index - 1), quickSort(array, index + 1, high)

Pop: each stack push, the whole array is sorted in respective partitions



... ...

apply same logic to the rest