Sorting Algorithm

Selection Sort
Simple Comparison based algorithm.
Diving an array into 2 parts sorted and unsorted.

Steps:

Start with first element
find smallest element in the unsorted part of the array
swap it with first unsorted element
move the boundaries between sorted and unsorted one by one
Repeat until array is sorted

Complexity

Case	Time Complexity
Best Case	O(n^2)
Average Case	O(n^2)
Worst Case	O(n^2)

Space Complexity: No extra memory Used O(1)

How Space Complexity is calculated:

refers to the amount of some extra memory required by algorithm to run, also if you take some inputs from user.

Fixed space for variables and constants
if you are using some extra data structure like (array, objects, stack, queue etc)

```
function selectionSort(arr) {
let n = arr.length
for (let i = 0; i < n - 1; i++) {
let minIndex = i; //smallest element index first
for (let j = i + 1; j < n; j++) {
if (arr[j] < arr[minIndex])
minIndex = j
}
if (minIndex !== i)
[arr[i], arr[minIndex]] = [arr[minIndex], arr[i]]
}
}

const arr = [20, 12, 10, 15, 2]
selectionSort(arr)
console.log('Result: ',arr)
```

in our code:

- n: single constant number
- i,j, minIndex: all loop variables (constant space)
- swapping using temp variable constant space
- No array used extra no ds used

complexity: O(1)

Merge Sort:

it is using divide and conquer approach. Divides array into 2 parts, sort each part recursively and merge those sorted parts.

Steps:

- divide array into 2 parts (partition)
- recursively sort each part
- merge the sorted parts to produce sorted array

Merge Sort is stable algorithm and work properly with big datasets due to its time complexity.
time Complexity for Merge Sort
log n : number of each time the array is divided (depth of recursion)
n : time to merge elements at each level.

Case	Time Complexity
Best Case	O(n log n)
Average Case	O(n log n)
Worst Case	O(n log n)

Space Complexity: O(n)- requires addition space to store temporary array during merging.

```
function mergeSort(arr){
if(arr.length<=1)
return arr; //array is already sorted as only 1 or less element

const mid = Math.floor(arr.length/2);
const left = mergeSort(arr.slice(0,mid));
const right = mergeSort(arr.slice(mid));
return merge(left,right)
}
function merge(left,right){
console.log("left",left,"right",right)
const result= []
let i=0, j=0;
while(i<left.length && j<right.length){
if(left[i]<=right[j]){
result.push(left[i]);
i++;
}else{
result.push(right[j]);
j++;
}
}
return result.concat(left.slice(i)).concat(right.slice(j));
}

console.log(mergeSort([38,27,43,3,9,82,10]))
```