

```

for (let num = 0; num <= 9; num++)
    if (isSafe(grid, col, row, num)) {
        grid[row][col] = num;
        if (solveSudokuHelper(grid))
            return true
        else // backtrack
            grid[row][col] = 0;
    }
return false;

```

• Arrays :-

• Majority Element (Moore's voting algorithm) :-

Time complexity  $\rightarrow O(N)$   $\rightarrow$  space  $O(1)$

1	4	3	2	4	4	4	2
---	---	---	---	---	---	---	---

pseudo code  $\longrightarrow$

Here we check the next element  
if it is not then we use  
decrement.

$\uparrow$   
 element = array[0]  
 count = 1;  
 if (element != array[i])  
 count --;  
 if (count == 0)

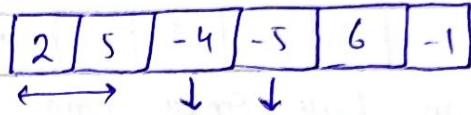
$\rightarrow$  This Problem can be solved using  
map.

But its time complexity is  
 $O(N) \rightarrow$  space  $O(N)$ .

count = 1;  
 element = array[i]  
 else-if (element == array[i])  
 count ++

## • Largest Sum Contiguous Subarray (Kadane's algorithm) :-

- if a negative number occurs we move to the next element and store sum.



function largestSum(array)

{

let maxSum = 0, curSum = 0;

for (let i = 0; i < array.length; i++)

{

curSum = curSum + array[i]

if (curSum > maxSum)

maxSum = curSum;

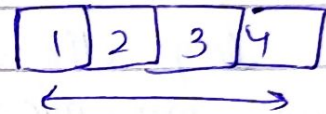
if (curSum <= 0)

cur = 0;

}

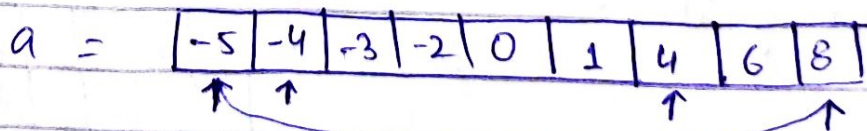
return maxSum;

}



Steps  
 → 3; → maxSum  
 → 1  
 → 6 → maxSum  
 → 12 → maxSum  
 → 9

## • Check sum Zero (array must be sorted) :-



Approach :-

Left = 0; Right = array.length - 1, sum;

sum = a[Left] + a[Right]

if sum > 0 right--

else left++;

sum == 0; console.log.



- Best time to Buy & sell stocks → LeetCode - 121.

3	5	1	7	4	9	3
---	---	---	---	---	---	---

you can buy only one time ∴

$min = \text{Math.min}(min, a[i]);$

$max = \text{Math.max}(max, a[i]);$

after loop the return

return  $max - min;$

- Best time to Buy & sell stocks → LeetCode 122 :-

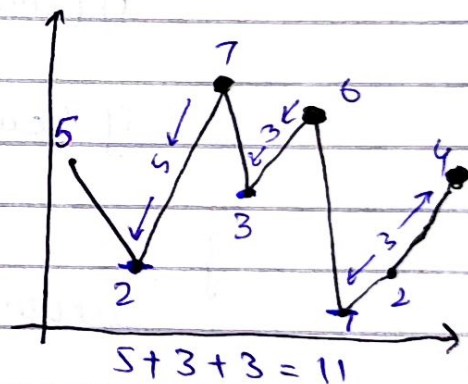
5	2	7	3	6	1	2	4
---	---	---	---	---	---	---	---

let profit = 0;

if (array[i] > array[i-1])

profit += array[i] - array[i-1];

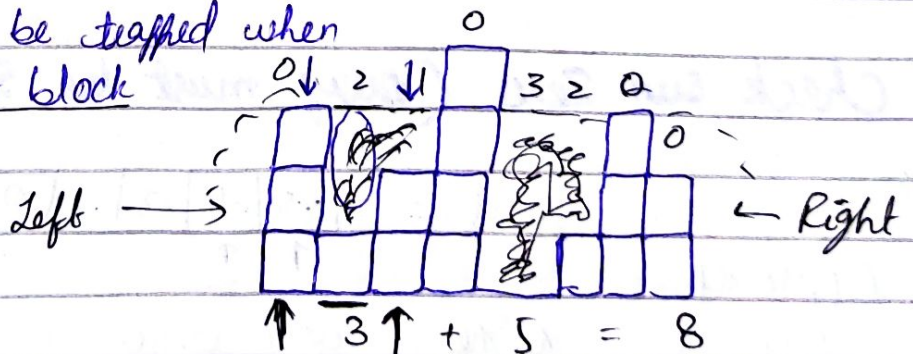
return profit;



- Trapping Rain water - LeetCode 42 :-

3	1	2	4	0	1	3	2
---	---	---	---	---	---	---	---

Logic:- water will only be trapped when  
Left block and right block  
will be greater.



Left-Max = 3, 3, 3, 4, 4, 4, 4, 4

Right-Max = 4, 4, 4, 4, 3, 3, 3, 2

value =  $\text{Min}(\text{Left}[i], \text{right}[i]) - a[i]$

## • Sorting Algorithms:-

### → Bubble Sort :-

Swapping is used and the largest element will move to the end.

8	4	1	5	9	2
---	---	---	---	---	---

in first iteration  $\rightarrow i = n \rightarrow j = 0 ; j \leq i ; j++$   
 if ( $\text{array}[j] > \text{array}[j+1]$ )

swap ( $\text{array}[j]$ ,  $\text{array}[j+1]$ )

4	8	1	5	9	2
4	1	5	8	9	2

4	1	8	5	9	2
4	1	5	8	2	9

$\rightarrow$  largest at end.

### • insertion sort :-

Take array and split it in two parts

8	4	1	5	9	2
---	---	---	---	---	---

sorted

non-sorted.

store element in temp =

for (let  $i = 1$ ;  $i < n$ ;  $i++$ )

let  $\text{temp} = \text{a}[i]$ ;

let  $j = i - 1$ ;

while ( $j \geq 0$  &&  $\text{a}[j] > \text{temp}$ ) {

$\text{array}[j+1] = \text{array}[j]$

$j--$ ;

}

$\text{array}[j+1] = \text{temp}$

bcs of while loop we incremented one time.

8	4	1	5	9	2
---	---	---	---	---	---

4	8	1	5	9	2
---	---	---	---	---	---

1	4	8	5	9	2
---	---	---	---	---	---

1	4	5	8	9	2
---	---	---	---	---	---

1	4	5	8	9	2
---	---	---	---	---	---

1	2	4	5	8	9
---	---	---	---	---	---

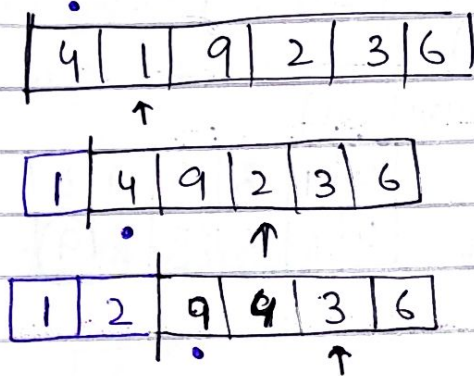


## • Insert Selection Sort :-

Take min value and allange it / swap it with first value.

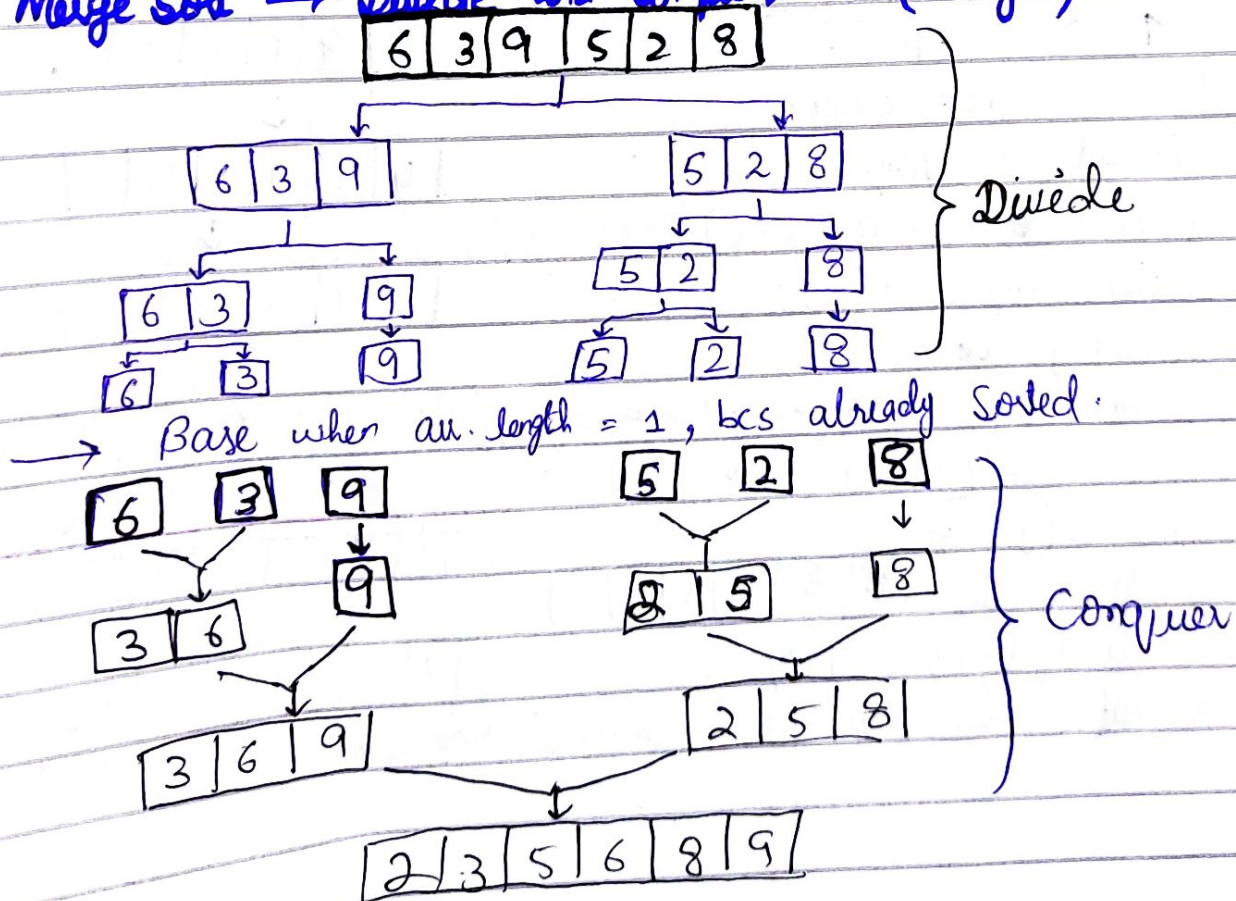
Outer loop runs  $< n-1$  times.

Inner loop runs  $< n$  times.



Dot represents min value and arrow  $j^{\text{th}}$  value.

## • Merge Sort $\rightarrow$ Divide and Conquer $\rightarrow (n \log n)$



## • Quick Sort :-

Pivot and Partition

start = 0

end = n-1

6	3	9	5	2	8
---	---	---	---	---	---

i = start - 1;

↑  
Pivot

Pivot

- random
- mid
- first
- last

Traversing

for (let j = start; j < end; j++) {

if (array[j] < pivot)

i++

swap;

}

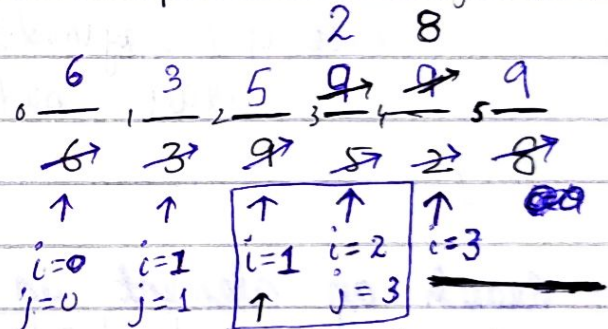
i++

swap (array[end], array[i]);

return i;

}

Loop will end at j = 4



bcs condition is false.  
swapped

function quicksort (array)

{ pivotIndex = partition(array, start, end);  
when i reached at the index - 1 times from higher position

quicksort (array, start, pivotIndex - 1);

quicksort (array, pivotIndex + 1, end);

}