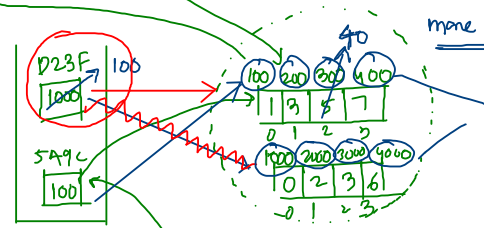
Example :

```
685  let arr = [1, 3, 5, 7];
686  console.log(arr);
687
688  let brr = [0, 2, 3, 6];
689  brr[2] = 30;
690
691  brr = arr;
692  arr[2] = 40;
693  console.log(arr);
694  console.log(brr);
```

100



address (hexadecimal)
0, 1, 2, 7, 4, 5, 6, 7, 8
A, B, C, D, E, F

name	address
arr	5A9C
brr	D23F

* 691. brr = arr
brr = 100
Now both arr and brr refers to the same array at 100 in the heap

* 692. arr[2] = 40
100[2] = 40

* 693. 1, 3, 40, 7
694. 1, 3, 40, 7

const arr = [1, 3, 5, 7];
arr[2] = 40;

why const is not giving error that cannot be changed?

→ we are changing the heap box not the stack box

→ But this fails
arr = brr; ×

```
696  function incrementNum(num) {
697    num++;
698  }
699
700  let num = 5;
701  incrementNum(num);
702  console.log(num);
```

num=5

5

5

PASS BY VALUE

```
704  function incrementArr(arr) {
705    arr[0]++;
706  }
707
708  let brr = [1, 3, 5, 7];
709  incrementArr(brr);
710  console.log(brr);
```

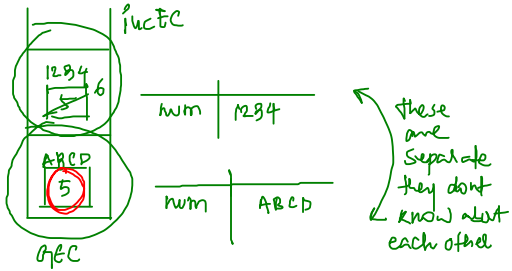
arr = 100

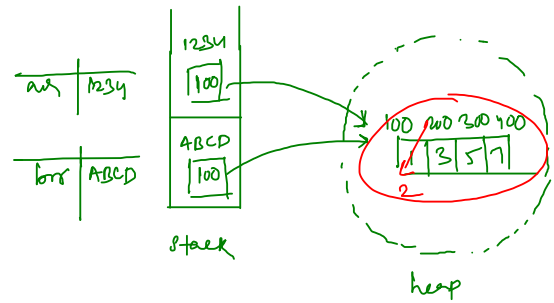
100

ref

2, 3, 5, 7

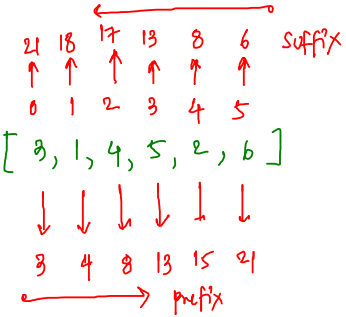
PASS BY Reference





* Every function has own EC that means it has own memory space.

Prefix Sum :



Eg: arr : [3, 1, 4, 5, 2, 6]

```
let sum = 0;
for (let i=0; i<n; i++) {
    sum += arr[i];
    cl(sum);
}
```

★ for suffix iterate reverse

→ acting like your prefix sum

i=0, sum += arr[0]
⇒ sum = 0+3 = (3)
i=1, sum += arr[1]
⇒ sum = 3+1 = (4)

Prefix Max :

0 1 2 3 4 5
arr : [3, 1, 4, 5, 2, 6]
3 3 4 5 5 6

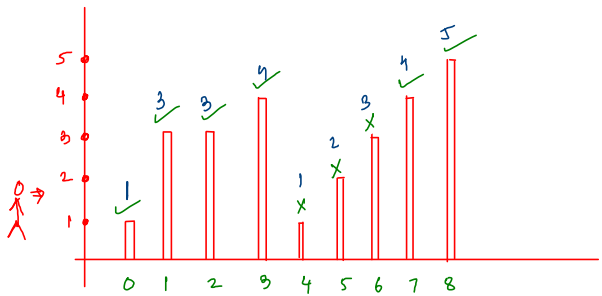
```
let max = -infinity;
for (let i=0; i<n; i++) {
    max = Math.max(arr[i], max);
    cl(max);
}
```

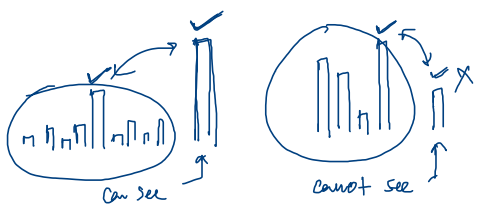
→ acting like your prefix max

i=0, max = max(3, -∞) = 3
i=1, max = max(1, 3) = 3
i=2, max = max(4, 3) = 4

★ Buildings :

[1, 3, 3, 4, 1, 2, 3, 4, 5] op: 6
0 1 2 3 4 5 6 7 8





★ How will you decide whether you can see an i^{th} building or not ?

[1, 3, 3, 4, 1, 2, 3, 4, 5]
0 1 2 3 4 5 6 7 8

tallest

Can you see or not ?

→ you need to check is this hiding or not

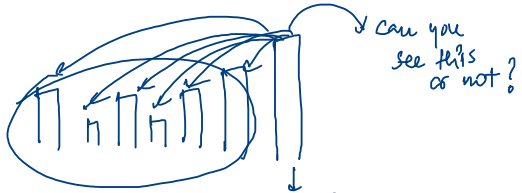
→ i.e, tallest building on the left <= ar(i)

⇒ you can see

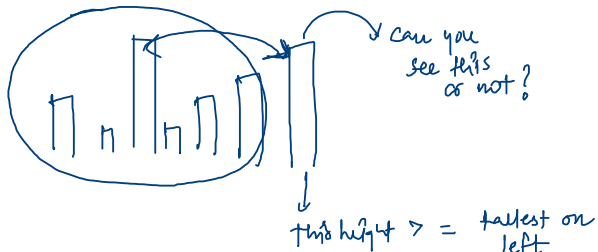
1, 3, 3, 4, 1, 2, 3, 4, 5
-∞ 1 3 3 4 4 4 4 4 (max on left)

↳ something like prefix

-∞ <= 1 ✓
1 <= 3 ✓
3 <= 3 ✓
3 <= 4 ✓
4 <= 1 ✗
4 <= 2 ✗
4 <= 3 ✗
4 <= 4 ✓
4 <= 5 ✓



* But comparing all elements is not efficient
⇒ more comparisons are required

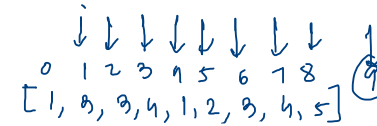


* Instead of comparing with all elements, just compare with the tallest building

⇒ Why? (Because only the tallest can hide the current building)

```
function countVisibleRoofs(heights) {
  // Write your code here
  const n = heights.length;
  let cnt = 0;
  let tallestOnLeft = -Infinity;
  for (let i = 0; i < n; i++) {
    if (heights[i] >= tallestOnLeft) {
      cnt++;
    }
    tallestOnLeft = Math.max(tallestOnLeft, heights[i]);
  }

  return cnt;
}
```



tL = ~~-∞~~ ~~1~~ ~~3~~ ~~4~~ 5

① i = 0, 1 >= -∞, cnt++

② i = 1, 3 >= 1, cnt++

③ i = 2, 3 >= 3, cnt++

④ i = 3, 4 >= 3, cnt++

⑤ i = 4, 1 >= 4, X

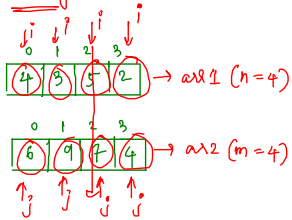
⋮

⑧ i = 7, 4 >= 4, cnt++

⑨ i = 8, 5 >= 4, cnt++

* Array Adding :

Eg :



res = [ ], carry = 0 ~~1~~

i = ~~3~~ 2 1

j = ~~3~~ 2 1

1. arr1[i] + arr2[j]

⇒ arr1[3] + arr2[3]

⇒ 2 + 4 = 6

res.push(6) ⇒ res ⇒ ~~[ ]~~ ⇒ [6]

i--; j--;

2. arr1[i] + arr2[j]

⇒ arr1[2] + arr2[2]

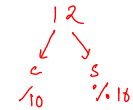
⇒ 5 + 7 = 12

⇒ res.push(12 % 10)

⇒ carry = 12/10 = 1

i--; j--;

* Carry Concept Rearhred



3. arr1[i] + arr2[j] + carry

⇒ arr1[1] + arr2[1] + carry

⇒ 2 + 9 + 1 = 13

⇒ res.push(13 % 10) ⇒ [6,2]

⇒ carry = 13/10 = 1 ⇒ [6,2,3]

i--; j--;

4. arr1[i] + arr2[j] + carry

⇒ arr1[0] + arr2[0] + carry

⇒ 4 + 6 + 1 = 11

⇒ res.push(11 % 10) ⇒ ~~[6,2,3]~~ ⇒ [6,2,3,1]

⇒ carry = 11/10 = 1

5. i = -1, j = -1 (stop)

* leftover carry,

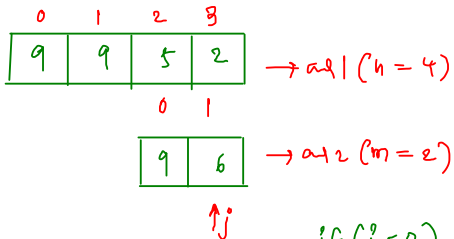
(5,2,3,1) carry = 1

(6,2,3,1,1)

* queue,

(1,1,3,2,6) ✓

eg:



if (i < 0)
★ if (j < 0)
handle in code

3. arr1(1) + carry

⇒ 9 + 1 = 10
⇒ res.push (10 % 10) ⇒ ~~[3,4]~~ ⇒ [8,4,0]
⇒ carry = 10/10 = 1

4. arr1(0) + carry

⇒ 9 + 1 = 10
⇒ res.push (10 % 10) ⇒ ~~[8,4,0]~~ ⇒ [8,4,0,0]
⇒ carry = 10/10 = 1

res = [ ], carry = 0 1
i = 3 2 1 0 -1
j = 1 0 -1 -2 -3

1. arr1(3) + arr2(1) + 0

⇒ 2 + 6 + 0 = 8
⇒ res.push (8 % 10) ⇒ ~~[ ]~~ ⇒ [8]
⇒ carry = 8/10 = 0

2. arr1(2) + arr2(0) + carry

⇒ 5 + 9 + 0 = 14
⇒ res.push (14 % 10) ⇒ ~~[8]~~ ⇒ [8,4]
⇒ carry = 14/10 = 1

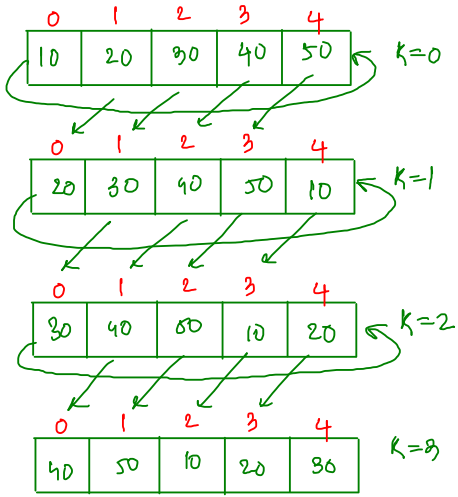
★ left over carry
(8,4,0,0) ⇒ (8,4,0,0,1)

★ reverse ⇒ (1,0,0,4,8)

* Rotate Array :

Eg :

(arr, K = 3)



op :

K=0	10, 20, 30, 40, 50
K=1	20, 30, 40, 50, 10
K=2	30, 40, 50, 10, 20
K=3	40, 50, 10, 20, 30
K=4	50, 10, 20, 30, 40
K=5	10, 20, 30, 40, 50
K=6	20, 30, 40, 50, 10
K=7	30, 40, 50, 10, 20
K=8	40, 50, 10, 20, 30
K=9	50, 10, 20, 30, 40
K=10	10, 20, 30, 40, 50
K=11	20, 30, 40, 50, 10

* Imp observation,

K = 0, 5, 10, 15, 20, 25, ... 100
K = 1, 6, 11, 16, 21, 26, ... 1001
K = 2, 7, 12, 17, 22, ... 1098
K = 3, 8, 13, 18, 23, ...
K = 4, 9, 14, 19, 24, ...

Eg: 1098^{th} rotation?

⇒ 1098 % 5 = 3
⇒ find K = 3
⇒ 40, 50, 10, 20, 30

Eg: 100^{th} rotation?

K = 100
↘ find K = 0
⇒ 10, 20, 30, 40, 50

Eg: 1001^{st} rotation?

K = 1001
1001 % 5 = 1
↘ find K = 1
⇒ 20, 30, 40, 50, 10

✱ How to change the array ?

arr = [10, 20, 30, 40, 50]

K = 28

✱ K = K % N = 28 % 5 = 3

① reverse the entire arr

[50, 40, 30, 20, 10]
0 1 2 3 4

② reverse 1st n-K elements, n-K = 5 - 3 = 2

0 1 2 3 4
[50, 40, 30, 20, 10]

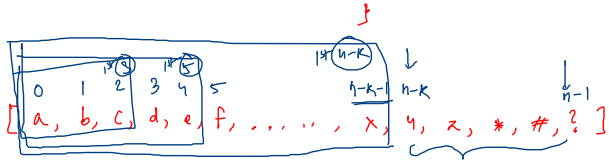
[40, 50, 30, 20, 10]

③ reverse remaining elements

[40, 50, 30, 20, 10]

[40, 50, 10, 20, 30]

```
function reverseArr(arr, start, end) {
    /* code
    */
}
function rotateArr(arr, n, k) {
    k = k % n
    reverseArr(arr, 0, n-1);
    reverseArr(arr, 0, n-k-1);
    reverseArr(arr, n-k, n-1);
}
```



1st 3 elements = (0, 2) → (0, 3-1)

1st 5 elements = (0, 4) → (0, 5-1)

1st n-K elements = (0, (n-k)-1)