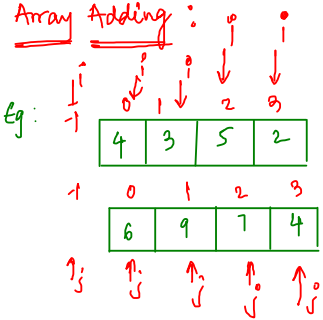
★ Array Adding :

Eg:



i = 1 j = 1

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

⇒ 3 + 9 + 1 = 13

⇒ res.push(13 % 10)

⇒ carry = 13/10 = 1

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

⇒ 4 + 6 + 1 = 11

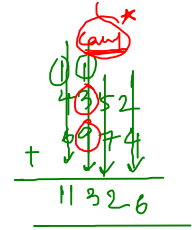
⇒ res.push(11 % 10), carry = 1/10 = 1

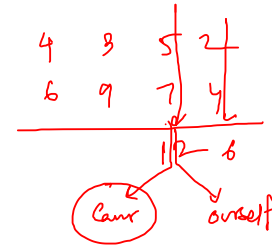
★ If there is any carry left over, push to res

⇒ [6, 2, 3, 1, 1]

★ reverse res

⇒ [1, 1, 3, 2, 6]





res = [] [6] [6,2] [6,2,3]

1. arr1[i] + arr2[j]

⇒ 4 + 2 = 6

⇒ res.push(6)

[6, 2, 3, 1]

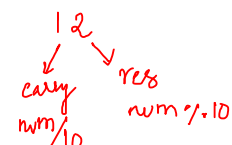
[6, 2, 3, 1, 1]

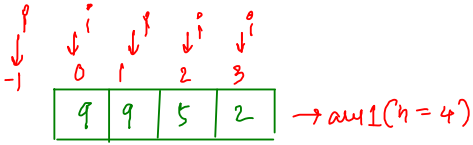
2. arr1[i] + arr2[j]

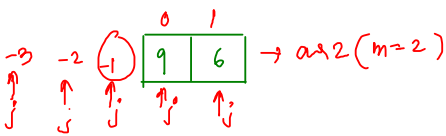
⇒ 5 + 7 = 12

⇒ res.push(12 % 10)

⇒ carry = 12/10 = 1







* leftover carry,

⇒ res = [8,4,0,0,1]

* reverse,

⇒ res = [1,0,0,4,8]

res = ~~[]~~ ~~[8]~~ ~~[8,4]~~ ~~[8,4,0]~~ [8,4,0,0]

carry = 0

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

⇒ 2 + 6 + 0 = 8

⇒ res.push(8 % 10)

⇒ carry = 8/10 = 0

⇒ i--; j--;

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

⇒ 5 + 9 + 0 = 14

⇒ res.push(14 % 10)

⇒ carry = 14/10 = 1

⇒ i--; j--;

3. arr1[i] + carry

⇒ 9 + 1 = 10

⇒ res.push(10 % 10)

⇒ carry = 10/10 = 1

⇒ i--; j--;

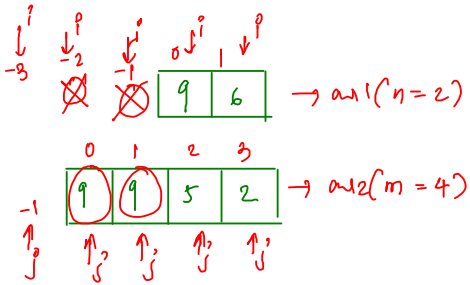
4. arr1[i] + carry

⇒ 9 + 1 = 10

⇒ res.push(10 % 10)

⇒ carry = 10/10 = 1

⇒ i--; j--;



res = ~~[]~~ ~~[8]~~ ~~[8,4]~~ ~~[8,4,0]~~ [8,4,0,0]

carry = 0 0 ✗ ✗ 1

1. 6+2+0 = 8 → 8%10, 8/10

8. 9+1 = 10 → 10%10, 10/10

9. 9+5+0 = 14 → 14%10, 14/10

4. 9+1 = 10 → 10%10, 10/10

res = [8,4,0,0,1] → leftover carry

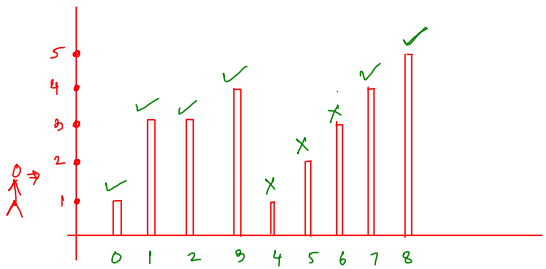
res = [1,0,0,4,8] → reverse

★ repeat the process only when element from i or j is present, i>=0 || j>=0

* Buildings :

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

* you can only when tallest on left <= curr[i]



op: 6 (no of buildings you can see)

* How to decide whether you can see a building or not ?

Can you see this ?

- you will check is this hiding
- tallest building on left greater than (2) ⇒ hiding

tallest <= 2 ⇒ see

cnt = 0

1. i = 0 ⇒ -∞ <= 1 → cnt++ 1
2. i = 1 ⇒ 1 <= 8 → cnt++ 2
3. i = 2 ⇒ 8 <= 3 → cnt++ 3
4. i = 8 ⇒ 8 <= 4 → cnt++ 4
5. i = 4 ⇒ 4 <= 1 → X
6. i = 5 ⇒ 4 <= 2 → X
7. i = 6 ⇒ 4 <= 3 → X
8. i = 7 ⇒ 4 <= 4 → cnt++ 5
9. i = 8 ⇒ 4 <= 5 → cnt++ (6)

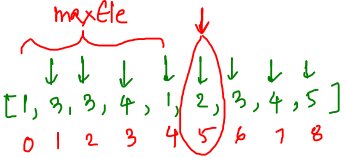
Running stream :

```
maxEle = -∞
for(let i=0; i<n; i++) {
    console.log(maxEle);
    if (arr[i] > maxEle) {
        maxEle = arr[i];
    }
}
```

or

```
maxEle = Math.max(arr[i], maxEle)
```

* tallest building on left
  = current running maxEle (maxEle until current i)



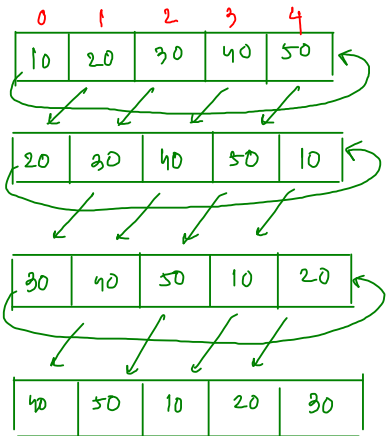
maxEle = ~~-∞~~ 1 3 4 5

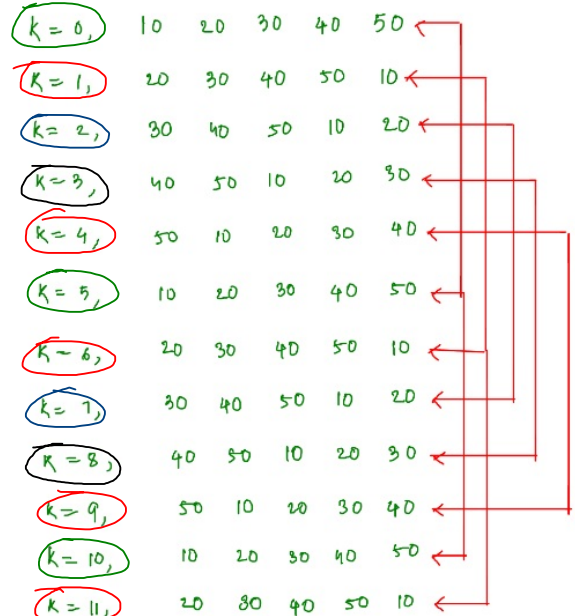
1. i=0 → -∞, 1 > -∞
2. i=1 → 1, 3 > 1
3. i=2 → 3, 3 > 3
4. i=3 → 3, 4 > 3
5. i=4 → 4, 1 > 4
6. i=5 → 4, 2 > 4
7. i=6 → 4, 3 > 4
8. i=7 → 4, 4 > 4
9. i=8 → 4, 5 > 4

* running Min,
  running Sum,
  running product,
  running Max

★ Rotate Array : [In place] make changes in given array only do not use extra space / extra arrays

eg:





* Imp observation,

$k \% N \Rightarrow k \% 5$

$k = 0, 5, 10, 15, 20, 25, \ldots \rightarrow 0$

$k = 1, 6, 11, 16, 21, 26, \ldots \rightarrow 1$

$k = 2, 7, 12, 17, 22, 27, \ldots \rightarrow 2$

$k = 3, 8, 13, 18, 23, 28, \ldots \rightarrow 3$

$k = 4, 9, 14, 19, 24, 29, \ldots \rightarrow 4$

Eg: 100^{th} rotation,

$\Rightarrow$ 100 falls in which group

$\Rightarrow k \% 5 \Rightarrow 100 \% 5 = 0$

1001^{th} rotation

$\rightarrow 1001 \% 5 \Rightarrow 1$

$k = 1$

arr = [10, 20, 30, 40, 50] k = 28

* k = k % N ⇒ k = 28 % 5 = 3

① reverse entire array,

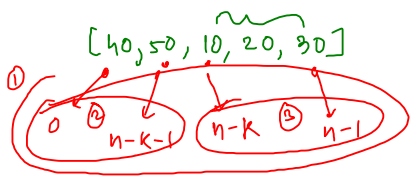
[50, 40, 30, 20, 10]
 0 1 2 3 4

② reverse 1st n-k elements,

n - K = 5 - 3 = 2

[40, 50, 30, 20, 10]
 0 1

③ reverse remaining elements




```
function reverseArr(arr, start, end) {
    /* code
    */
}

function rotateArr(arr, k) {
    k = k % n;
    reverseArr(arr, 0, n-1);
    reverseArr(arr, 0, n-k-1);
    reverseArr(arr, n-k, n-1);
}
```