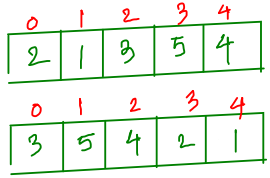
* Adding two Arrays :

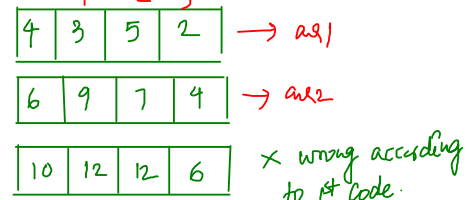
Eg:



#1: Carry not handled

```
const res = [];
for(let i = n-1; i >=0; i--) {
   res[i] = arr1[i] + arr2[i];
}
```

Eg:




```
const res = [];
let carry = 0;
for(let i = n-1; i>=0; i--) {
   const sum = arr1[i] + arr2[i] + carry;
   res[i] = sum % 10;
   carry = sum/10;
}
```

#2: Handling carry

```
const res = [];
let carry = 0;
for(let i = n-1; i>=0; i--) {
    const sum = arr1[i] + arr2[i] + carry;
    res[i] = sum % 10;
    carry = sum/10;
}
if (carry == 1) {
    add "1" to beginning/start
    of the array
}
```

0	1	2	3	
1	3	2	6	→ res

0	1	2	3	4	
1	3	2	6		→ res

0	1	2	3	
4	3	5	2	→ arr1
6	9	7	4	→ arr2
1	3	2	6	→ res

```
const sum
= arr1[i] + arr2[i]
    + carry;
= 9 + 9 + 1
= 19
```

carry > 1 not possible

carry = 0 1

① i = 3

Sum = 2 + 4 + 0
= 6
res[3] = 6 % 10 = 6
Carry = 6/10 = 0

② i = 2

Sum = 5 + 7 + 0
= 12
res[2] = 12 % 10 = 2
Carry = 12/10 = 1

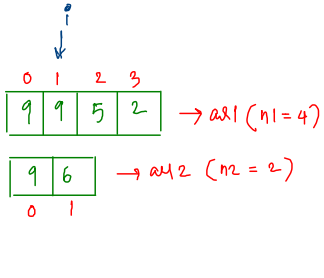
③ i = 1

Sum = 3 + 9 + 1
= 13
res[1] = 13 % 10 = 3
Carry = 13/10 = 1

④ i = 0

Sum = 4 + 6 + 1
= 11
res[0] = 11 % 10 = 1
Carry = 11/10 = 1

Eg:




```
let i = n1 - 1;
let j = n2 - 1;
let carry = 0;
while (i >= 0) {
    let sum;
    if (j >= 0) sum = arr1[i] + arr2[j] + carry;
    else sum = arr1[i] + carry;
    res[i] = sum % 10;
    carry = sum/10
    i--;
    j--;
}
```

if (carry == 1) {
add "1" to beginning/start of the array
}

* Iterate based on bigger array, (we assume n1 >= n2)

① i = 3, j = 1
Sum = 2 + 6 + 0 = 8
res[3] = 8 % 10 = 8
Carry = 8/10 = 0

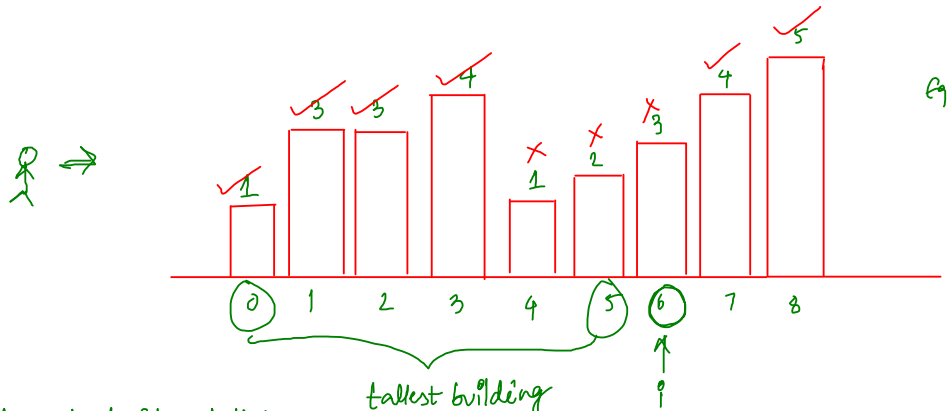
② i = 2, j = 0
Sum = 5 + 9 + 0 = 14
res[2] = 14 % 10 = 4
Carry = 14/10 = 1

③ i = 1, j = -1
Sum = 9 + 1 = 10
res[1] = 10 % 10 = 0
Carry = 10/10 = 1

④ i = 0, j = -2
Sum = 9 + 1 = 10
res[0] = 10 % 10 = 0
Carry = 10/10 = 1

⑤ i = -1, j = -3 ~~(i >= 0)~~

★ Buildings :



→ Checking with just previous element is not sufficient.

Eg: Can you see arr[6] = 3

3 < 2 ⇒ NO

⇒ I can see ✗

★ How to decide whether you can see a building or not !

→ arr[i] < tallest building on its left [max(arr[0...i-1])]

→ 3 < 4 ⇒ you cannot see.

```
let cnt = 0;
for (let i = 0; i < n; i++) {
    if (arr[i] >= tallest building on [0...i-1]) {
        cnt++;  // I can see
    }
}
```

how to get this efficiently

Running stream / carry forward technique :

→ keep track of max element until the current index, runningMax / mxm.

* can be applied in many problems, variations might include : runningMin, runningSum, runningProduct.

```
  0  1  2  3  4  5
[1, 3, 3, 4, 1, 2]
```

runningMax = ~~-∞~~ / 8 4

① i = 0,
* what is the the max-tle on my left? ⇒ runningMax ⇒ (-∞)
follow findMax logic,
1 > -∞, rM = 1

② i = 1,
* max on left ⇒ rM ⇒ (1)
3 > 1 ⇒ rm = 3

③ i = 2,
*max on left ⇒ rm ⇒ (3)
3 > 3 X

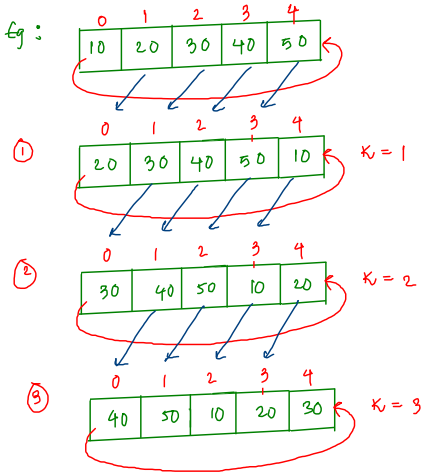
④ i = 3
* max on left ⇒ rm ⇒ (3)
4 > 3 ⇒ rm = 4

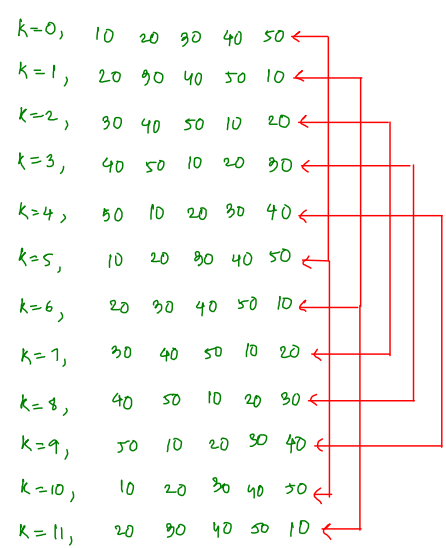
⑤ i = 4
*max on left ⇒ rm ⇒ (4)
1 > 4 X

⑥ i = 5
* max on left ⇒ rm ⇒ (4)
2 > 4 X

★ Rotate Array :

Eg :





★ Find K = 100 ?

10	20	30	40	50
0	1	2	3	4

→ you need not rotate one by one.

→ think smart,

K % n

K = 0, 5, 10, 15, 20, 25 → 0

K = 1, 6, 11, 16, 21, 26 → 1

K = 2, 7, 12, 17, 22, 27 → 2

K = 3, 8, 13, 18, 23, 28 → 3

K = 4, 9, 14, 19, 24, 29 → 4

⇒ given K, answer for K % n

100 % 5 = 0

⇒ K = 9298, 9298 % 5

= 3

⇒ Now you know that
given K, we find K % n.

arr = [10, 20, 30, 40, 50], k = 3

① reverse entire array,

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

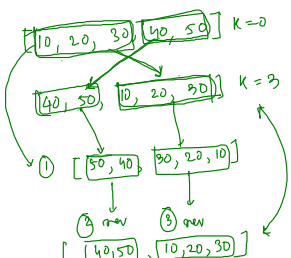
② reverse 1st n-k elements

n-k = 5-3 = 2

[40, 50, 30, 20, 10]

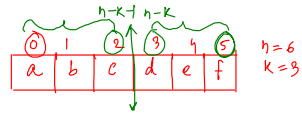
③ reverse last k elements (k=3)

[40, 50, 10, 20, 30]




```
function reverseArr(arr, s, e) {
    /*
      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);
}
```



① s=0, e=5 ⇒ s=0, e=n-1

② n-k = 6-3 = 3 ⇒ s=0, e=n-k-1

③ s=n-k, e=n-1