* Adding 2 arrays :

Eg:

0	1	2	3	
4	3	5	2	→ arr1 (n=4)
6	9	7	4	→ arr2 (m=4)

```
  1 1
  4 3 5 2
+ 6 9 7 4
-------
1 1 3 2 6
```

5+7 = 1 | 2 (c, keep)

1+3+9 = 1 | 3 (c, keep)

1+4+6 = 11

```
let i = n-1;
let j = m-1;
let carry = 0;
let res = [];
while ( i>=0 || j>=0 ) {
    const sum = arr1[i] + arr2[j] + carry;
    res.push (sum % 10);
    carry = sum/10;
    i--;
    j--;
}
```

1+5+7 = 1 | 3 (carry, keep)

sum/10 = 1

sum%10 = 3

→ carry = 0 or 1, c > 1 is not possible

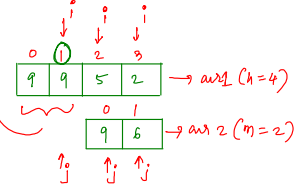
max possible addition,

⇒ 9+9+1 = 19 (c = 1, keep)

'j' is completed but 'i' is remaining

how to handle remaining elements

```
let i = n-1;
let j = m-1;
let carry = 0;
let res = [];
while (i >= 0 || j >= 0) {
    const sum = arr1[i] + arr2[j] + carry;
    res.push(sum % 10);
    carry = parseInt(sum/10);
    i--;
    j--;
}
while (i >= 0) {
    const sum = arr1[i] + carry;
    res.push(sum % 10);
    carry = parseInt(sum/10);
    i--;
}
if (carry == 1) {
    res.push(1);
}
res.reverse();
```



① sum = 2 + 6 + 0
= 8
carry = 8/10 = 0
i-- ⇒ 2, j-- ⇒ 0

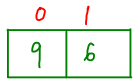
② sum = 5 + 9 + 0
= 14
carry = 14/10 = 1
i-- ⇒ 1, j-- ⇒ -1

③ sum = 9 + 1 = 10
carry = 10/10 = 1
i-- ; ⇒ 0

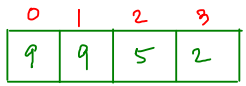
④ sum = 9 + 1 = 10
c = 10/10 = 1
i-- ; ⇒ -1

carry = 0 1
res = [] = [8] = [8,4] = [8,4,0] = [8,4,0,0]

eg:



→ arr1 (n = 2)



→ arr2 (m = 4)

* 'i' can be alone
  or 'j' can be alone
  any of these can be possible

→ handle 'j' in the same way as you handled 'i'.

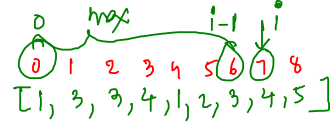
i is completed
but j is remaining

→ i, j run together
→ j is alone

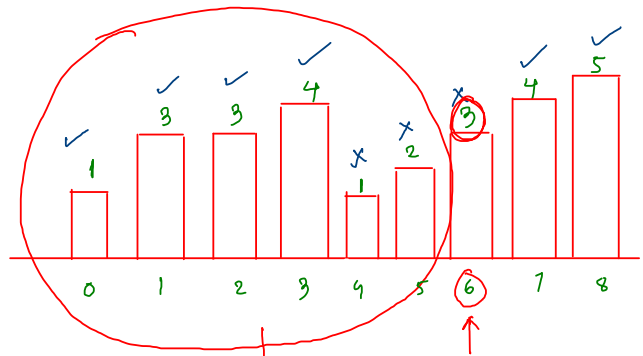
In prefix

→ i, j run together
→ i is alone

★ **Buildings :**



op = 6



Can you see this ?

tallest building = 4

★ How to decide wether you can see current building or not ?

Curr H < tallest of left part

⇒ This means (3) will be hidden due to tallest (4) on the left

→ only count when curr H >= tallest on left

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

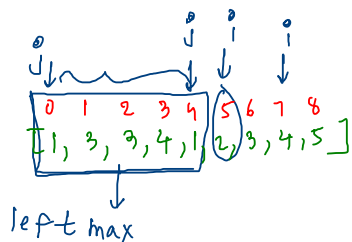
i=0 → 1 ≥ -∞ → cnt = 1
i=1 → 3 ≥ 1 → cnt = 2
i=2 → 3 ≥ 3 → cnt = 3
i=3 → 4 ≥ 3 → cnt = 4
i=4 → 1 ≥ 4 → hidden
i=5 → 2 ≥ 4 → hidden
i=6 → 3 ≥ 4 → hidden
i=7 → 4 ≥ 4 → cnt = 5
i=8 → 5 ≥ 4 → cnt = 6

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

how to get this ?

(leftMax) — iterate and find max

```
let leftMax = -∞
for (let j = 0; j < i; j++) {
    leftMax = Math.max(leftMax, arr[j]);
}
```



```javascript
function countVisibleRoofs(heights) {
  // Write your code here
  const n = heights.length;
  let cnt = 0;

  for (let i = 0; i < n; i++) {
    let leftMax = -Infinity;
    for (let j = 0; j < i; j++) {
      leftMax = Math.max(leftMax, heights[j]);
    }

    if (heights[i] >= leftMax) {
      cnt++;
    }
  }

  return cnt;
}
```

0 1 2 3 4

[a, b, c, d, e]

n = 5

1 + 2 + 3 + 4 = total

$= \frac{n(n-1)}{2}$

$\simeq n^2$ iterations

↓

n iterations

(Optimization needed)

① i = 0

j = 0 ✗

② i = 1

j = 0 ✗

③ i = 2

j = 0, 1 ✗

④ i = 3

j = 0, 1, 2 ✗

⑤ i = 4

j = 0, 1, 2, 3 ✗

* Running stream / Carry Forward :

→ keep track of max element until the crrent Index

→ here running max is indicating the left Max

* runningMin, runningSum, runningProduct, runningMax → Applications

0	1	2	3	4	5
1,	3,	3,	4,	1,	2

runningMax = $-\infty$ ~~1~~ ~~3~~ 4

ans = ~~3~~ ~~1~~ ~~2~~ ~~3~~ 4

① i = 0,
1 >= $-\infty$ → yes
rm = max(1, $-\infty$) = 1

② i = 1,
3 >= 1 → yes
rm = max(1, 3) = 3

③ i = 2,
3 >= 3 → yes
rm = max(3, 3) = 3

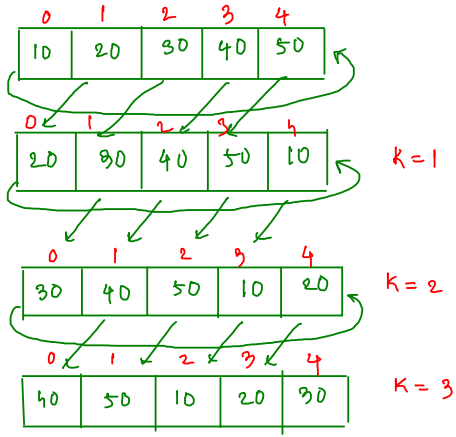
④ i = 3,
4 >= 3 → yes
rm = max(4, 3) = 4

⑤ i = 4
1 >= 4 → no
rm = max(4, 1) = 4

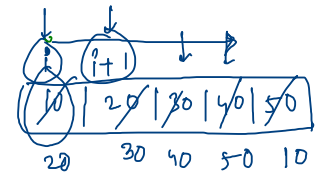
⑥ i = 5
2 >= 4 → no
rm = max(4, 2) = 4

* Rotate Array : [In place]

Eg:



1:



X = 10

arr[i] = arr[i+1]

repeat this k times

```
902   for (let time = 0; time < k; time++) {
903     const firstEle = arr[0];
904     for (let i = 0; i < n - 1; i++) {
905       arr[i] = arr[i + 1];
906     }
907     arr[n - 1] = firstEle;
908   }
```

① time = 0 → n iterations

② time = 1 → n iterations

③ time = 2 → n iterations

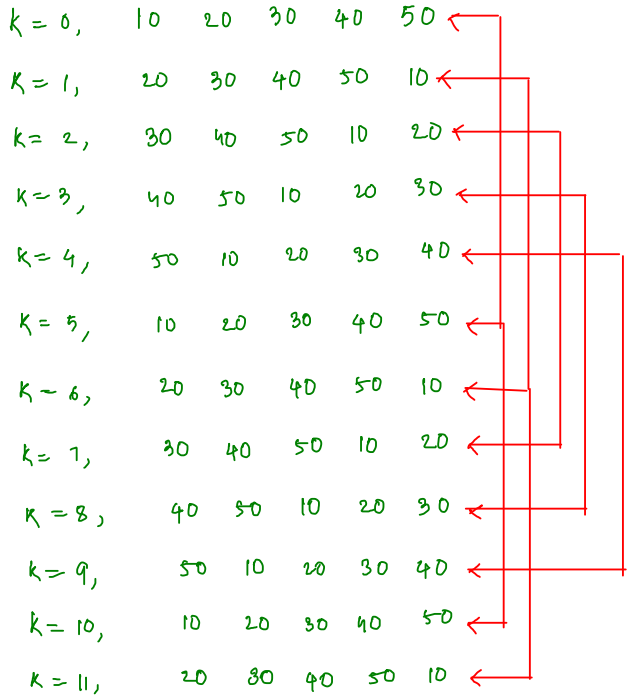
⋮

time = k-1 → n iterations

⇒ total = k * n

optimise to (n)

#2 :



★ Cycle ⇒ "%" (help) k%5 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, ... → 3

k = 4, 9, 14, 19, 24, 29, ... → 4

⇒ given k, answer for k = k%n

let k = 9298

→ 9298 % 5 = 3

k = 3 → 40 50 10 20 30

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

(0) (1) (2) (3) (4)

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]

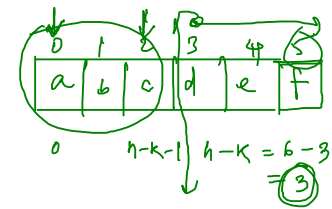
(0) (1) (2) (3) (4)

③ reverse last k elements

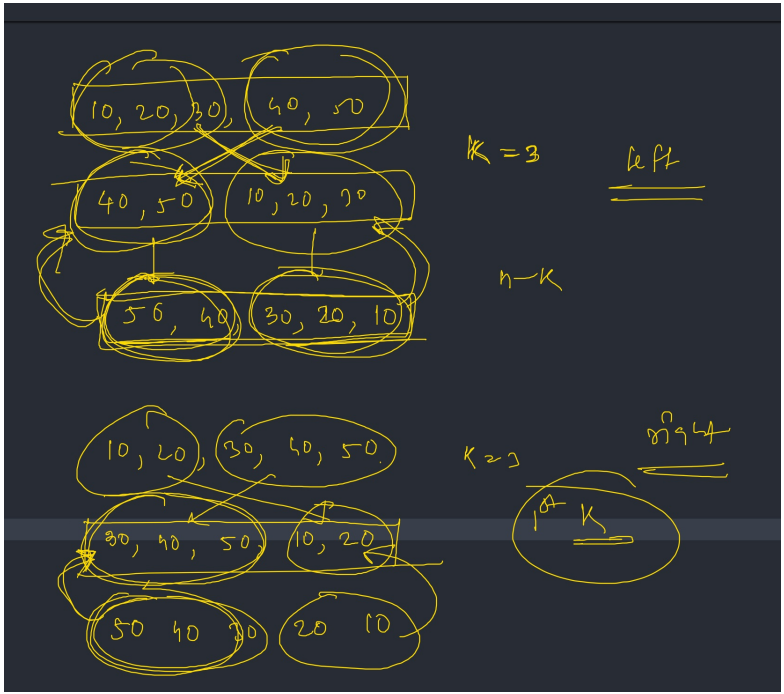
[40, 50, 10, 20, 30]

(0) (1) (2) (3) (4)

```
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);
}
```



* running stream (v. imp)
* cycle ⇒ % helps
* examples are imp to correct the code