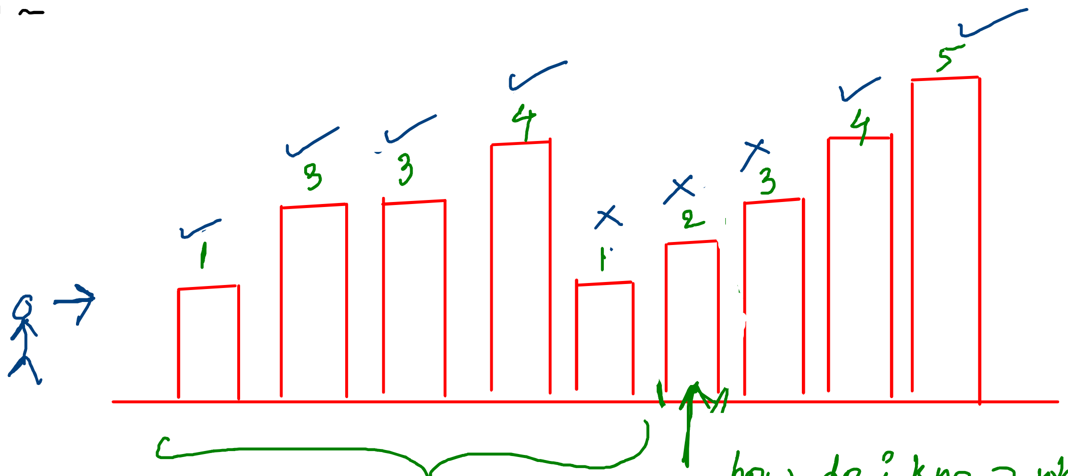
Q: Buildings



ip: [1, 3, 3, 4, 1, 2, 3, 4, 5]

op: 6

If there is any building with height > 2 then i cannot see

how do i know whether I can see this or not?

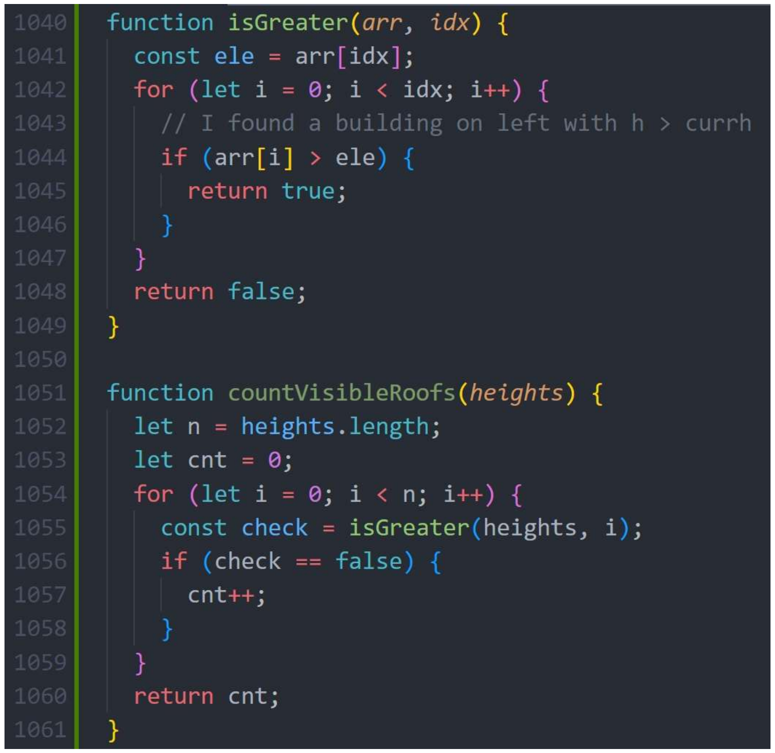
$$\begin{array}{ccccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ \underbrace{1, & 3, & 3, & 4, & \overset{}{\underset{\times}{1}}, & \underset{\times}{2}, & \underset{\times}{3}, & 4,}_{>5} & 5 \end{array}$$

Cnt = ~~0~~ ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ 6

is there > 1? NO

is there > 3? NO

$\underbrace{\textcircled{5}\ 3,}_{>4}\ 4 \times$



```javascript
function isGreater(arr, idx) {
  const ele = arr[idx];
  for (let i = 0; i < idx; i++) {
    // I found a building on left with h > currh
    if (arr[i] > ele) {
      return true;
    }
  }
  return false;
}

function countVisibleRoofs(heights) {
  let n = heights.length;
  let cnt = 0;
  for (let i = 0; i < n; i++) {
    const check = isGreater(heights, i);
    if (check == false) {
      cnt++;
    }
  }
  return cnt;
}
```

return → exit

$$h = \begin{array}{ccccccccc} 0 & 1 & 2 & 3 & 4 & \boxed{5} & 6 & 7 & 8 \\ [1, & 3, & 3, & 4, & 1, & 2, & 3, & 4, & 5] \end{array}$$

$>5$

cnt = ~~0~~ 1

(h, 5)

ele = arr[5]
= $\boxed{2}$

i = 0
arr[0] > 2
1 > 2 ✗

i = 1
arr[1] > 2
3 > 2

(h, 8)

ele = arr[8]
= 5

i = 0
arr[0] > 5
1 > 5

i = 1
arr[1] > 5
3 > 5

arr[2] > 5
3 > 5

arr[3] > 5
4 > 5

arr[4] > 6
1 > 5

arr[5] > 5
2 > 5

arr[6] > 5
3 > 5

arr[7] > 5
4 > 5

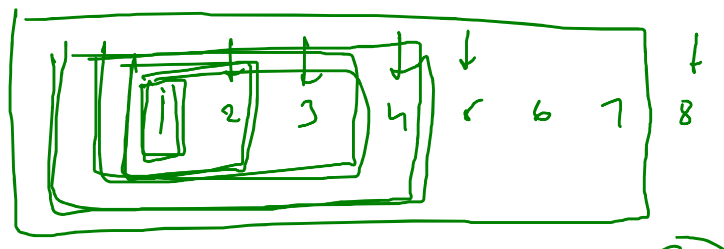
i = 8

i < idx
8 < 8 ✗

# Improving the logic

$i = 0 \rightarrow$ check on left part (0)

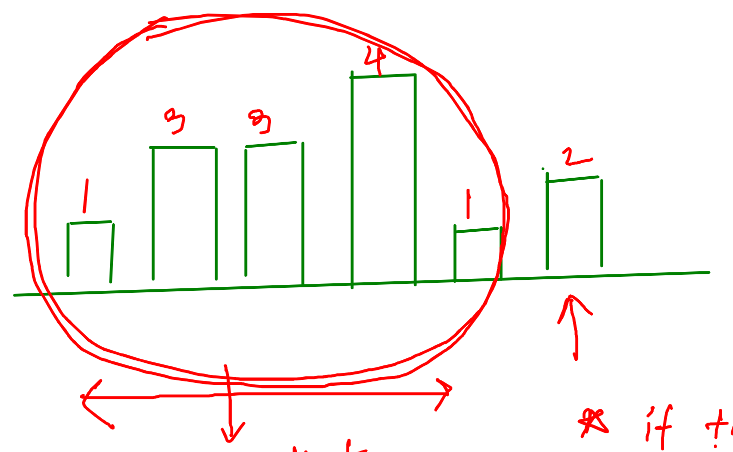
$i = 1 \rightarrow$ check on left part (1)



$1 + 2 + 3 + 4 + \cdots + \boxed{n-1} + n$

$\Rightarrow \dfrac{n(n-1)}{2} = \dfrac{n^2 - n}{2}$

$\Rightarrow \simeq n^2$ iterations



find tallest in this part

if tallest > curh → you cannot see

$[\overset{5}{\overbrace{1, 5, 2, 3,}}\ \boxed{4}\ \boxed{6}]$

max ele = 5

5 > 4 → Can't see 4

* if tallest building cannot block then other building cannot block.

5 > 6 ✗

# running maximum :

cnt = $\cancel{0}$ $\cancel{1}$ $\cancel{2}$ 3

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

$\underbrace{1, 4, 2, 3}_{4}$

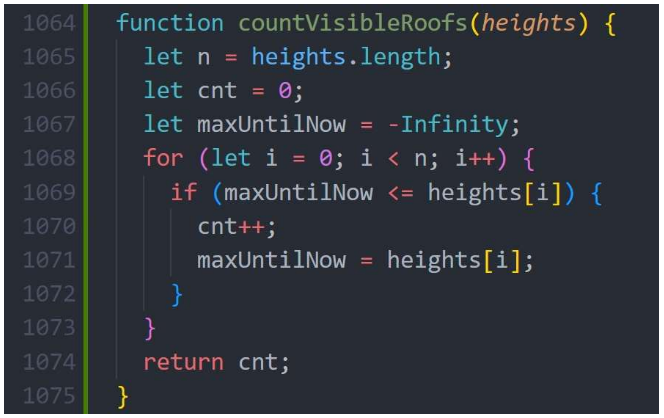
1. $-\infty > 1 \rightarrow$ no ✓
2. $1 > 4 \rightarrow$ no ✓
3. $4 > 2 \rightarrow$ yes ✗
4. $4 > 3 \rightarrow$ yes ✗
5. $4 > 6 \rightarrow$ no ✓
6. $6 > 5 \rightarrow$ yes ✗

max_until_now = $\cancel{-\infty}$ $\cancel{1}$ $\cancel{4}$ 6 $\rightarrow$ This will store the max on left part

$\Rightarrow$ n iterations

* running stream Technique

cnt = $\cancel{0}$ $\cancel{1}$ $\cancel{2}$ 3



```
function countVisibleRoofs(heights) {
  let n = heights.length;
  let cnt = 0;
  let maxUntilNow = -Infinity;
  for (let i = 0; i < n; i++) {
    if (maxUntilNow <= heights[i]) {
      cnt++;
      maxUntilNow = heights[i];
    }
  }
  return cnt;
}
```

1, 4, 2, 3, 6, 5

m = $\cancel{-\infty}$ $\cancel{1}$ $\cancel{4}$ 6

$-\infty <= 1$
$1 <= 4$
$4 <= 2$
$4 <= 3$
$4 <= 6$
$6 <= 5$ ✗

Q : Adding two Arrays :

Eg :

| 0 | 1 | 2 | 3 | 4 | |
|---|---|---|---|---|---|
| 2 | 1 | 3 | 5 | 4 | → arr1 |

| 0 | 1 | 2 | 3 | 4 | |
|---|---|---|---|---|---|
| 3 | 5 | 4 | 2 | 1 | → arr2 |

res[i] = arr1[i] + arr2[i]

| 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 5 | 6 | 7 | 7 | 5 |

res[0] = arr1[0] + arr2[0]

= 2+3 = 5

```
  21354
+ 35421
-------
  56775
```

Eg :

| 0 | 1 | 2 | 3 | |
|---|---|---|---|---|
| 4 | 3 | 5 | 2 | → arr1 |

| 0 | 1 | 2 | 3 | |
|---|---|---|---|---|
| 6 | 9 | 7 | 4 | → arr 2 |

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

~~carry~~

res :

| 0 | 1 | 2 | 3 |
|---|---|---|---|
| 11 | 3 | 2 | 6 |

$\boxed{1}$ $\boxed{1}$

6 % 10 = 6
6/10 = 0

2 + 4 + 0
= 6

5 + 7 + 0 = 12
12 % 10 = 2
12/10 = 1

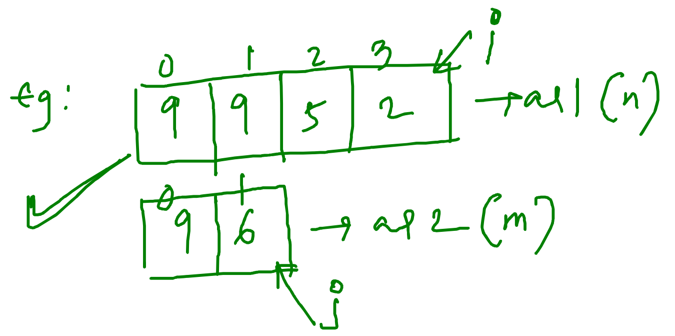
3 + 9 + 1 = 13
13 % 10 = 3
13/10 = 1

4 + 6 + 1 = 11
~~11 % 10 = 1~~
~~11/10 = 1~~

```
   ①  ①
    4  8  5  2
 +  6  9  7  4
 --------------
   11  8  2  6
```

not sending carry

eg:



```
 ①  ①
  8
  9  9  5  2
  +  9  6
 -----------
 10  0  4  8
```

```
i = n-1        c = 0
j = m-1

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

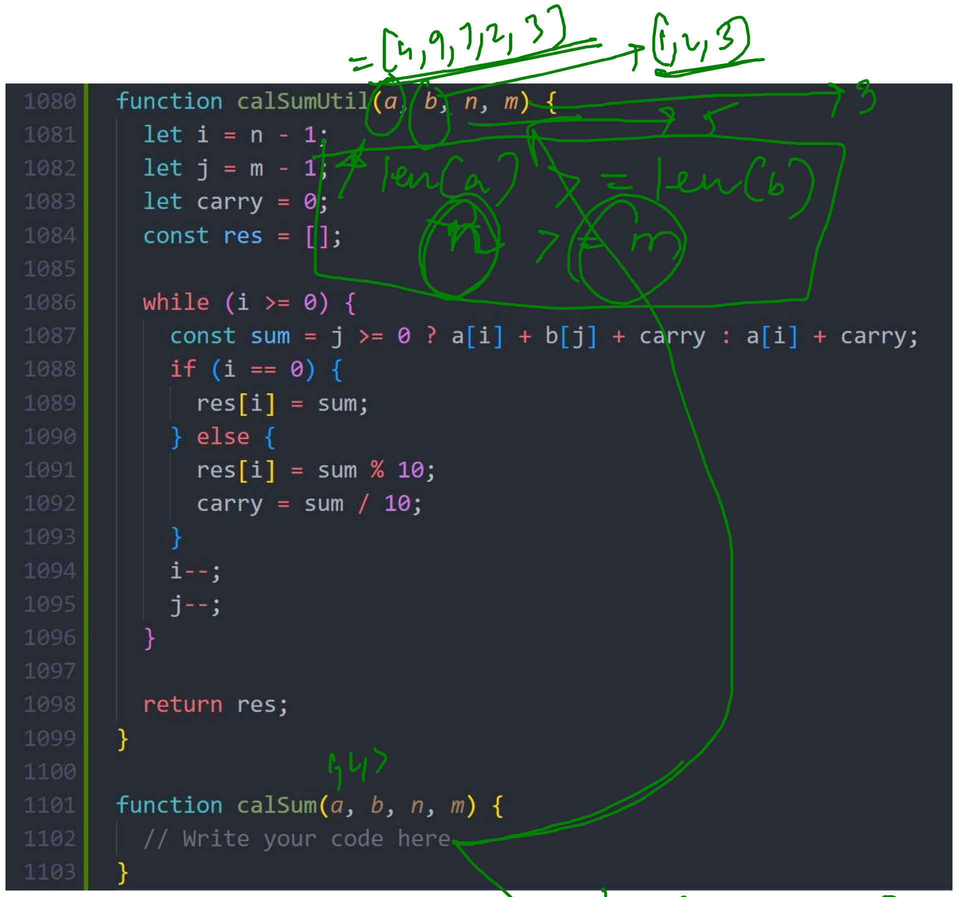
$i = \cancel{3}\ \cancel{2}\ \cancel{1}\ 0$

$j = \cancel{1}\ \cancel{0}\ \cancel{-1}\ -2$

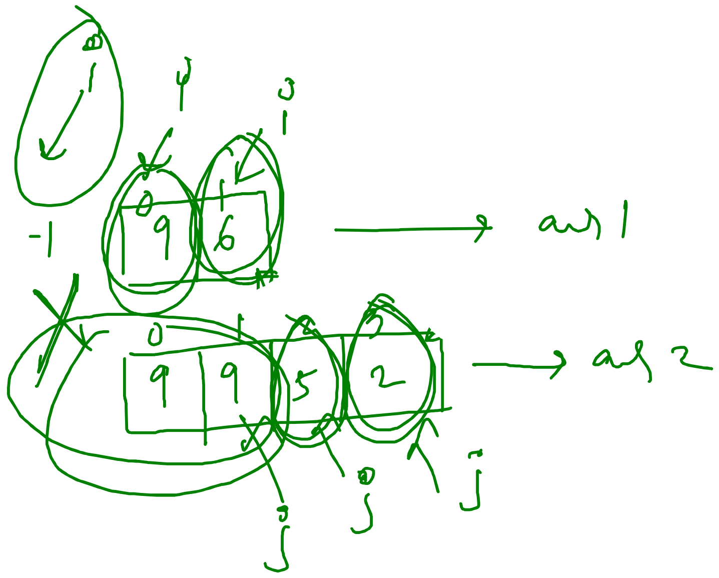
sum = 9 + 1
= 10

10 % 10 = 0

10 / 10 = 1

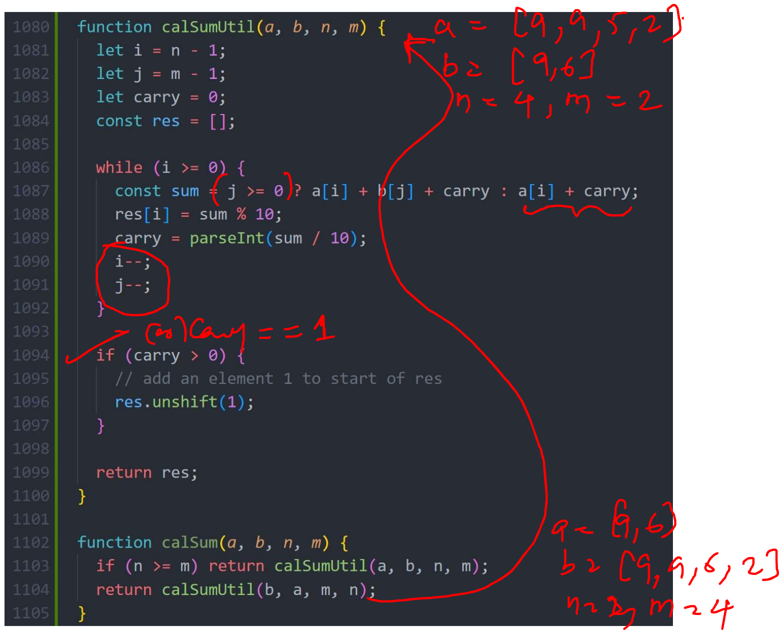


```
function calSumUtil(a, b, n, m) {
  let i = n - 1;
  let j = m - 1;
  let carry = 0;
  const res = [];

  while (i >= 0) {
    const sum = j >= 0 ? a[i] + b[j] + carry : a[i] + carry;
    if (i == 0) {
      res[i] = sum;
    } else {
      res[i] = sum % 10;
      carry = sum / 10;
    }
    i--;
    j--;
  }

  return res;
}

function calSum(a, b, n, m) {
  // Write your code here
}
```



if (n >= m)
calsumUtil(a, b, n, m)
else
calsumUtil(b, a, m, n)

a = [1, 2, 3]  n = 3
b = [4, 9, 7, 2, 3]  m = 5

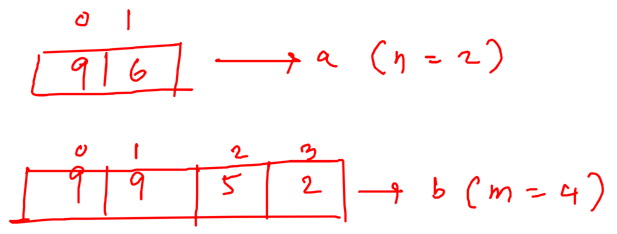


```javascript
function calSumUtil(a, b, n, m) {
  let i = n - 1;
  let j = m - 1;
  let carry = 0;
  const res = [];

  while (i >= 0) {
    const sum = j >= 0 ? a[i] + b[j] + carry : a[i] + carry;
    res[i] = sum % 10;
    carry = parseInt(sum / 10);
    i--;
    j--;
  }

  if (carry > 0) {
    // add an element 1 to start of res
    res.unshift(1);
  }

  return res;
}

function calSum(a, b, n, m) {
  if (n >= m) return calSumUtil(a, b, n, m);
  return calSumUtil(b, a, m, n);
}
```

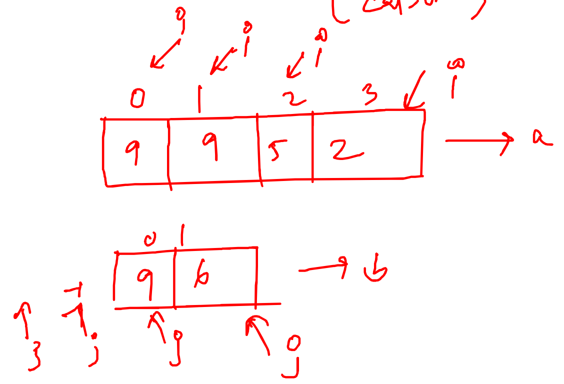
a = [9, 9, 5, 2]
b = [9, 6]
n = 4, m = 2

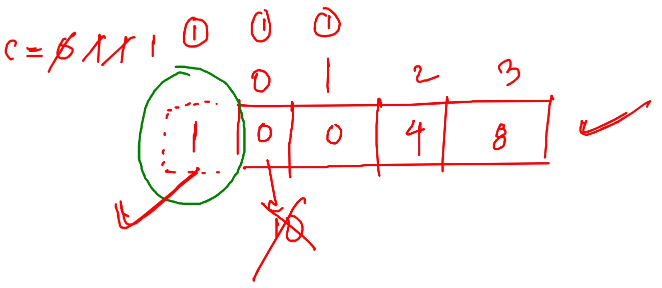
(or) Carry == 1

a = [9, 6]
b = [9, 9, 6, 2]
n = 2, m = 4



* len(a) >= len(b)
(calsum)





① Sum = 2 + 6 + 0 = 8
8 % 10, 8/10

② Sum = 5 + 9 + 0 = 14
14 % 10, 14/10

③ Sum = 9 + 1 = 10
10 % 10, 10/10

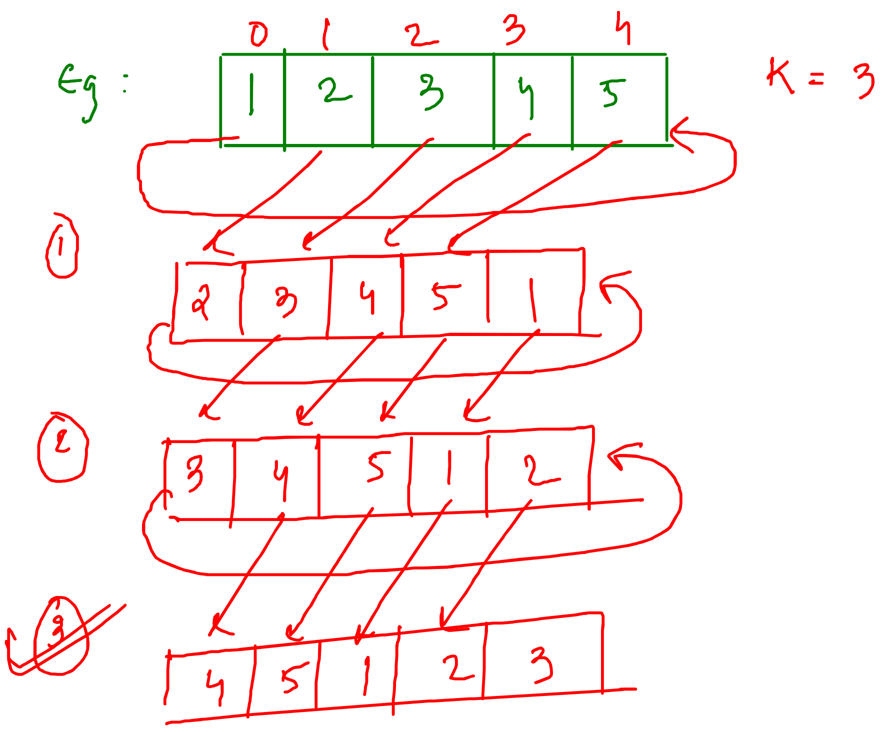
④ Sum = 9 + 1 = 10
10 % 10, 10/10

Carry = 0 or 1

* Carry = sum of two digits
= 9 + 9 + 1 = 19

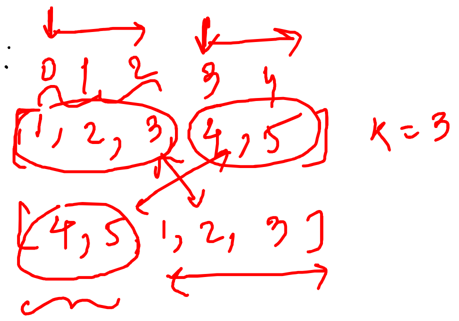
Q: Rotate Array :

Eg :



→ n iterations
→ extra array

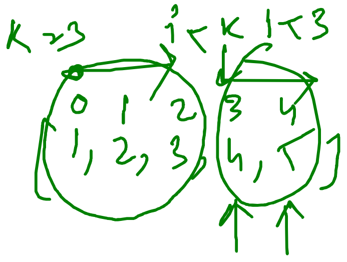
#1 :



① $(K \to n-1) \to (3 \to 4)$

② $(0 \to K-1) \to (0 \to 2)$

```
res = []
for (let i = K; i < n; i++){
    res.push(arr[i]);
}
for (let i = 0; i < k; i++){
    res.push(arr[i]);
}
```



ans = []
= [4]
= [4, 5]
= [4, 5, 1]
= [4, 5, 1, 2]
= [4, 5, 1, 2, 3]

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| $K=0$ | 1 | 2 | 3 | 4 | 5 |

$n=5$

$K = 11$

| | | | | | | |
|---|---|---|---|---|---|---|
| $K=1 \rightarrow$ | 2 | 3 | 4 | 5 | 1 | $1\%5=1$ |
| $K=2 \rightarrow$ | 3 | 4 | 5 | 1 | 2 | $2\%5=2$ |
| $K=3 \rightarrow$ | 4 | 5 | 1 | 2 | 3 | $3\%5=3$ |
| $K=4 \rightarrow$ | 5 | 1 | 2 | 3 | 4 | $4\%5=4$ |
| $K=\underline{\underline{5}} \Rightarrow$ | 1 | 2 | 3 | 4 | 5 | $5\%5=\textcircled{0}$ |
| $K=\underline{\underline{6}} \rightarrow$ | 2 | 3 | 4 | 5 | 1 | $6\%5=\textcircled{1}$ |
| $K=\underline{\underline{7}} \rightarrow$ | 3 | 4 | 5 | 1 | 2 | $7\%5=\textcircled{2}$ |
| $K=\underline{\underline{8}} \rightarrow$ | 4 | 5 | 1 | 2 | 3 | $8\%5=\textcircled{3}$ |
| $K=9 \rightarrow$ | 5 | 1 | 2 | 3 | 4 | $9\%5=\textcircled{4}$ |
| $K=10 \rightarrow$ | 1 | 2 | 3 | 4 | 5 | $10\%5=\textcircled{0}$ |
| $K=11 \rightarrow$ | 2 | 3 | 4 | 5 | 1 | $11\%5=\textcircled{1}$ |

$8\%5=\textcircled{3}$

★ ① $(K \rightarrow n-1) \rightarrow (\overset{0}{11} \rightarrow 4)$ ✗ $\rightarrow (1 \rightarrow 4)$

② $(0 \rightarrow K-1) \rightarrow (0 \rightarrow \overset{0}{10})$ ✗ $\rightarrow (0 \rightarrow 0)$

(This will fail)

ans = [2, 3, 4, 5, 1]

$K = 11$

$\Rightarrow K = K \% n$

$= 11 \% 5$

$= \textcircled{1}$

# Improve : (In-place)

arr =

| 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |

$n = 5$
$K = 3$

① reverse the entire array,

| 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 5 | 4 | 3 | 2 | 1 |

② reverse 1st n-k elements (5-3=2)

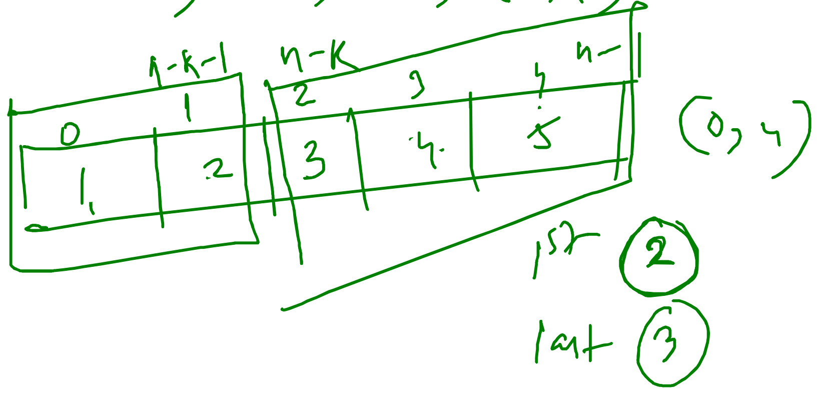
| 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 4 | 5 | 3 | 2 | 1 |

③ reverse last k elements (3)

| | | | | |
|---|---|---|---|---|
| 4 | 5 | 1 | 2 | 3 |

```
function reverse(arr, s, e) {
    b/w [s, e]
}
```

reverse (arr , 0, n-1) (0, 4)
reverse (arr, 0, n-k-1) (0, 1)
reverse(arr, n-k, n-1) (2,4)



0
1
n-k-1
n-k
2
3
4
n-
1
2
3
4
5
(0, 4)
1st ②
last ③