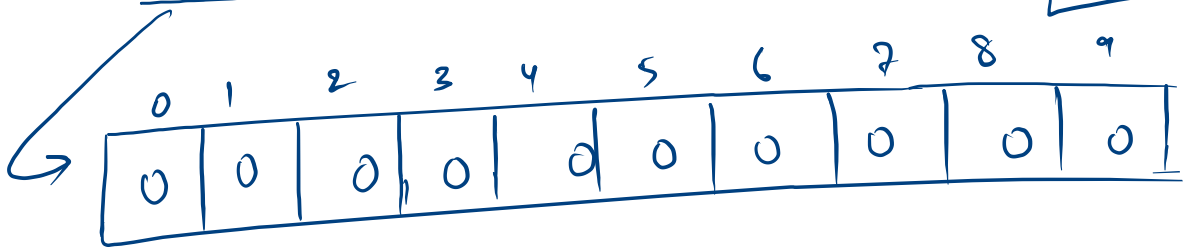
Array → → Collection of similar data
→ Contg. memory allocation

Declaration

```
int arr[] = new int [10];
```

(Index → Pos)

Range → [0 → len]



arr.length → size / no. of blocks

→ Find

→ Span

min
max

Identity

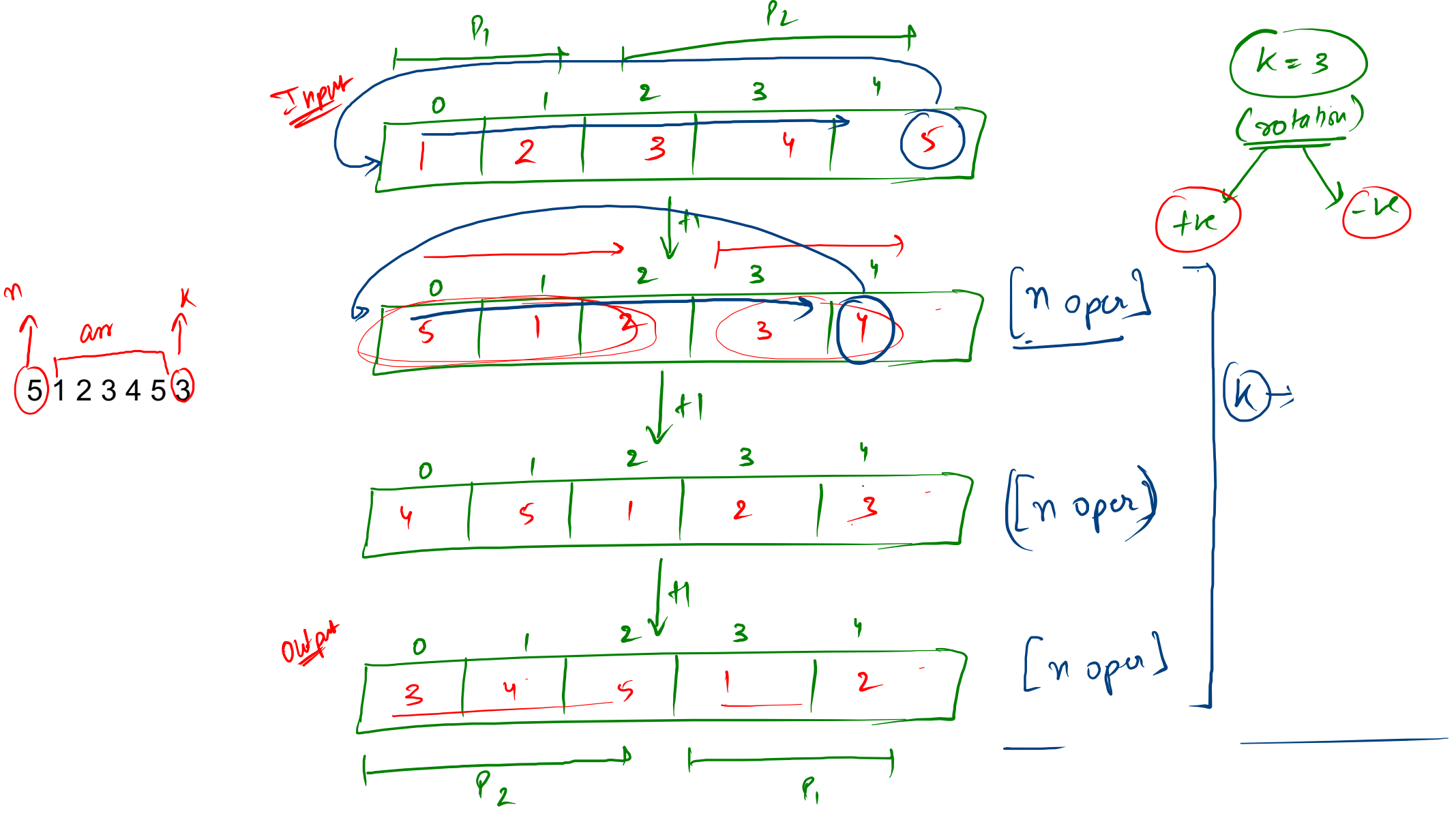
Default

+ → 0

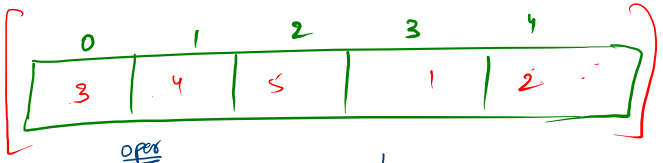
num + 0 = num

→ Reverse an array

↳ 2 ptr / [lo | hi] → Swap


n
arr
k
5 1 2 3 4 5 3
P1
P2
Input
0 1 2 3 4
1 2 3 4 5
+1
5 1 2 3 4
+1
4 5 1 2 3
+1
Output
3 4 5 1 2
P2
P1
k = 3
(rotation)
+ve
-ve
[n oper]
[n oper]
[n oper]
k

k=3



rotate using reverse

arr = p1 + p2

oper

① reverse P_2 [n]

② = P_1 [n]

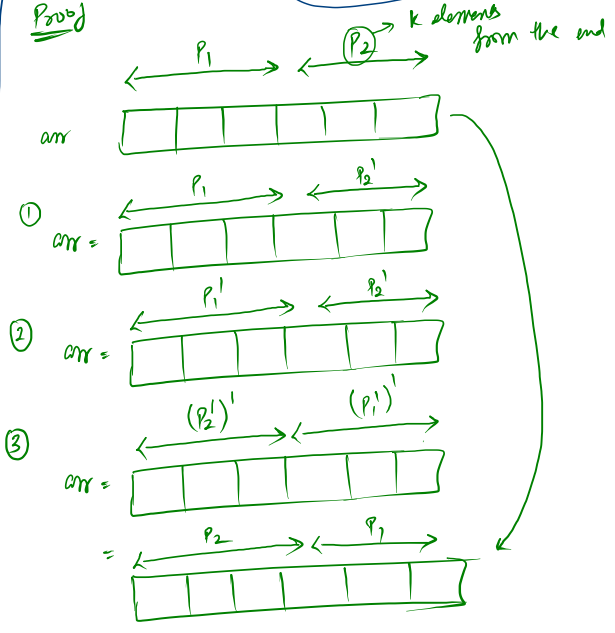
③ reverse arr [n]

3n

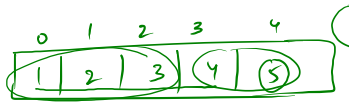
reverse / Inplace

Algo → T.C. = 3n → O(n) → Complexity

Proof



(a) arr =



K=11

(-5) = 1 2 3 4 5

(+1) = 5 1 2 3 4 (-4) = 5 1 2 3 4

(+2) = 4 5 1 2 3 (-3) = 4 5 1 2 3

(+3) = 3 4 5 1 2 (-2) = 3 4 5 1 2

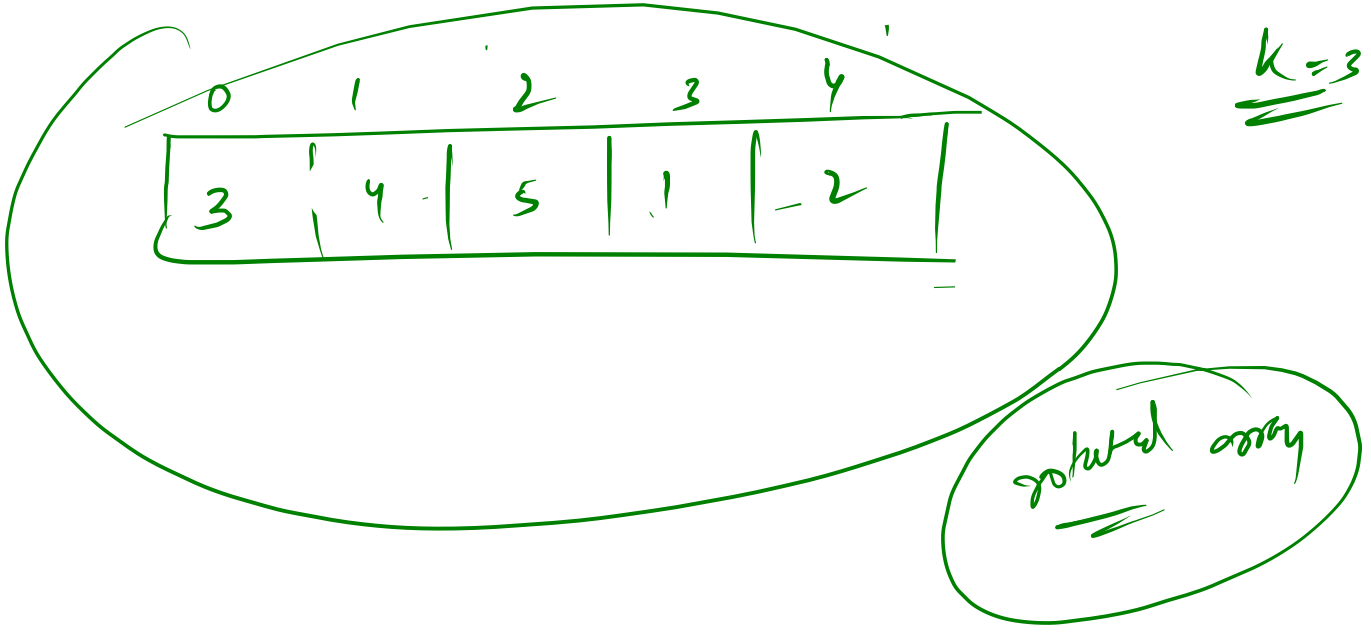
(+4) = 2 3 4 5 1 (-1) = 2 3 4 5 1

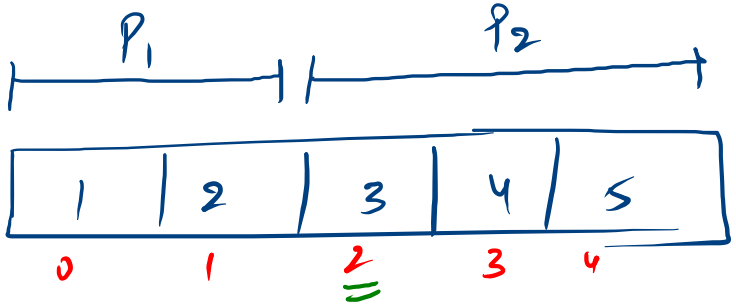
(+5) = 1 2 3 4 5

① rotation is cyclic → K = (K) % arr.length

② -ve rotation → +ve rotation

→ K = K + arr.length


0 1 2 3 4
3 4 5 1 2
k = 3
sorted array



K = 3

P1.sp = 0

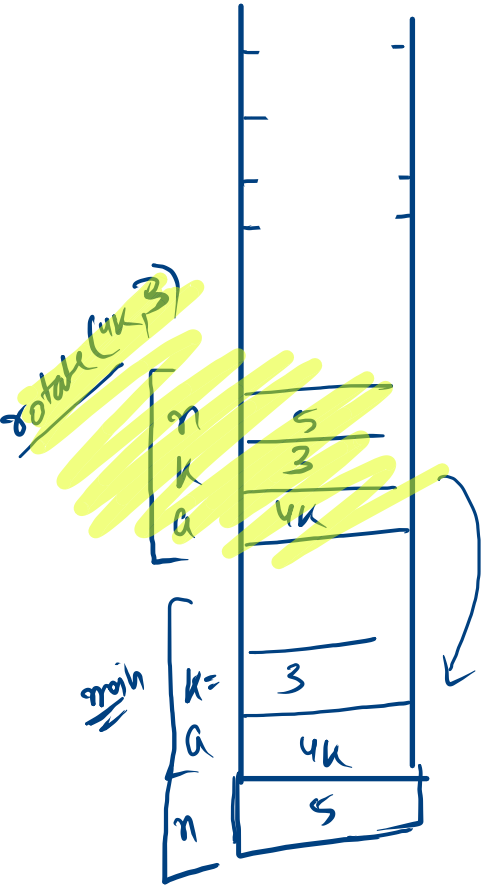
P1.ep = a.length - k - 1

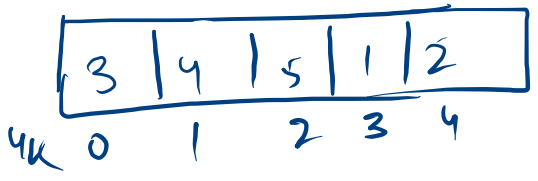
5 - 3 - 1 = 1

P2.sp = a.length - k = 5 - 3 = 2

P2.ep = a.length - 1 = 5 - 1 = 4

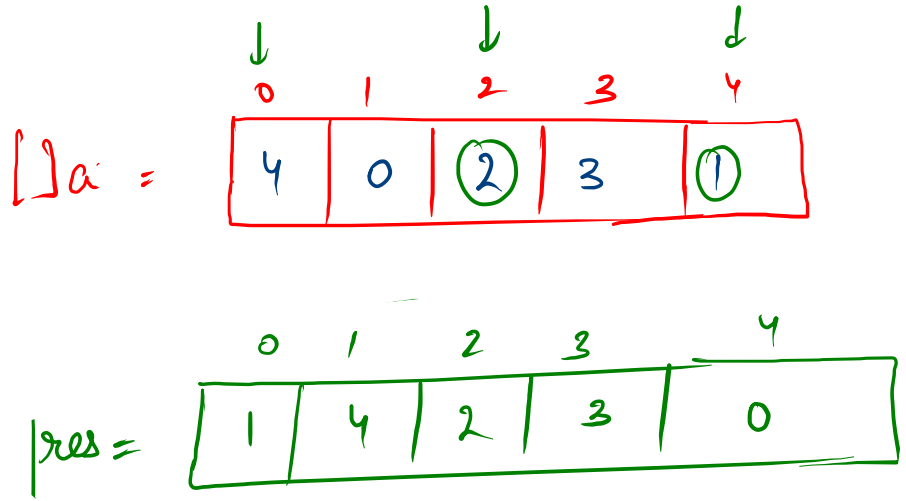
P1.ep + 1 = P2.sp ⇒ P1.ep = P2.sp - 1






```java
public static void rotate(int[] a, int k){
  int n = a.length;
  k = k % n; // true rotation number

  if(k < 0){
      // -ve rotation -> +ve rotation number
      k = k + n;
  }

  // arr = p1 + p2

  reverse(a,n-k,n-1);// reverse part2
  reverse(a,0,n-k-1);// reverse part1
  reverse(a,0,n-1);// reverse a
}

// reverse(a,lo,hi) : reverse part of array from [lo->hi]
public static void reverse(int a[],int lo,int hi){
    while(lo < hi){
        int tmp = a[lo];
        a[lo] = a[hi];
        a[hi] = tmp;

        lo++;
        hi--;
    }
}
```

```
5
4
0
2
3
1
```




```java
public static int[] inverse(int[] a){
  int res[] = new int[a.length];

  for(int idx = 0 ; idx < a.length ; idx++){
      int val = a[idx];
      res[val] = idx;
  }

  return res;
}
```

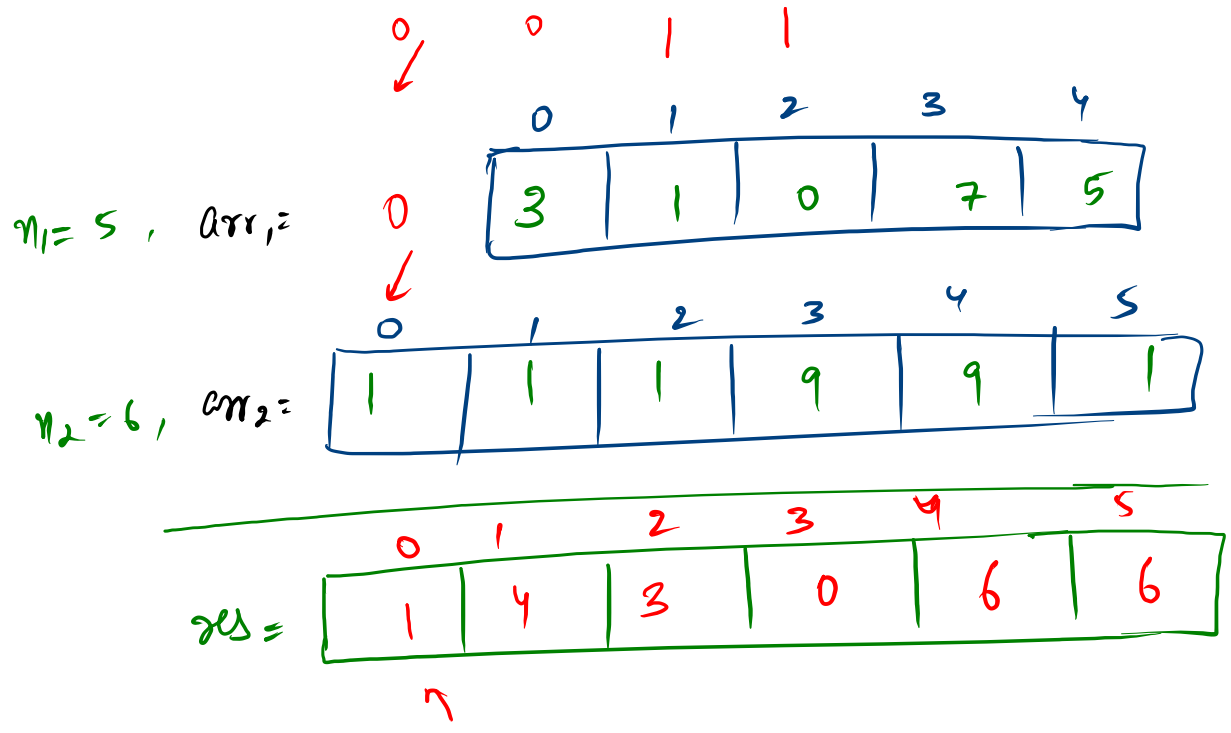
idx = 0, 1, 2, 3, 4

val = 4 0 2 3 1

What ✓

How

~~Why~~



0 0 1 1

Carry = 0

$p_1 = -2$, $d_1 = 0$

$p_2 = -1$, $d_2 = 1$

Sum = $d_1 + d_2 + carry$

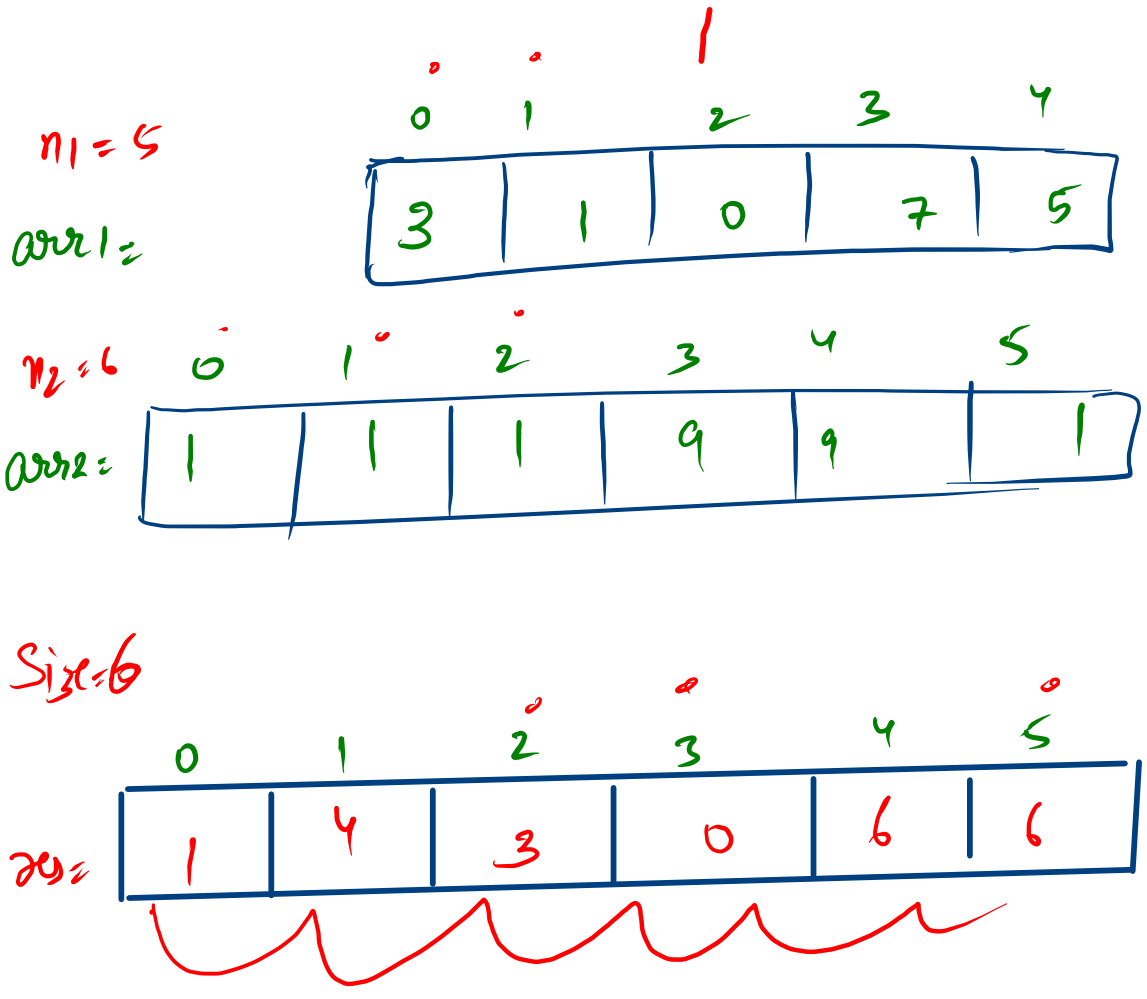
= 0 + 1 + 0

= 1

$p_3 = -1$

digit = Sum % 10 = ~~1%10 = 1~~

Carry = Sum / 10 = ~~1/10 = 0~~




```java
int size = n1 > n2 ? n1 : n2;
int res[] = new int[size];

int p1=n1-1 , p2=n2-1 , p3 = size-1 , carry = 0;

while(p1 >= 0 || p2 >= 0){
    int d1 = p1 < 0 ? 0 : arr1[p1];
    int d2 = p2 < 0 ? 0 : arr2[p2];

    int sum = d1 + d2  + carry;
    int digit = sum % 10;
    carry = sum / 10;

    res[p3] = digit;
    p1--;
    p2--;
    p3--;
}
```

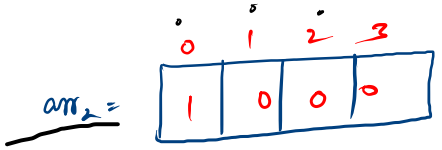
Sum	Digit	p_1	p_2	p_3	carry
1	1	d_1 : 0	d_2 : 1		

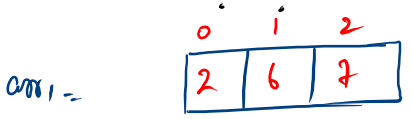
```java
boolean flag = false;
for(int idx = 0 ; idx < size ; idx++){
    if(res[idx] != 0){
        flag = true;
    }
    if(flag){
        System.out.println(res[idx]);
    }
}
```

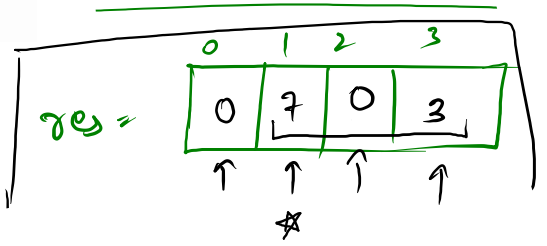
flag = false;
true;

idx = 0
X
1
3

7
0
3







borrow = 0 1 0

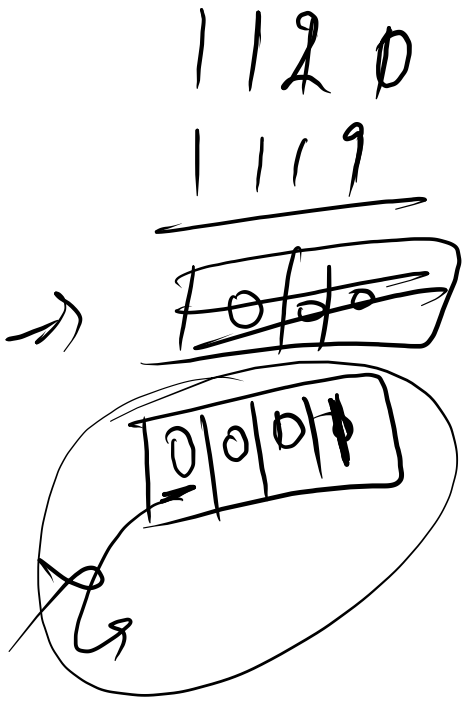
$p_2 = 3 \quad d_2 = 1$

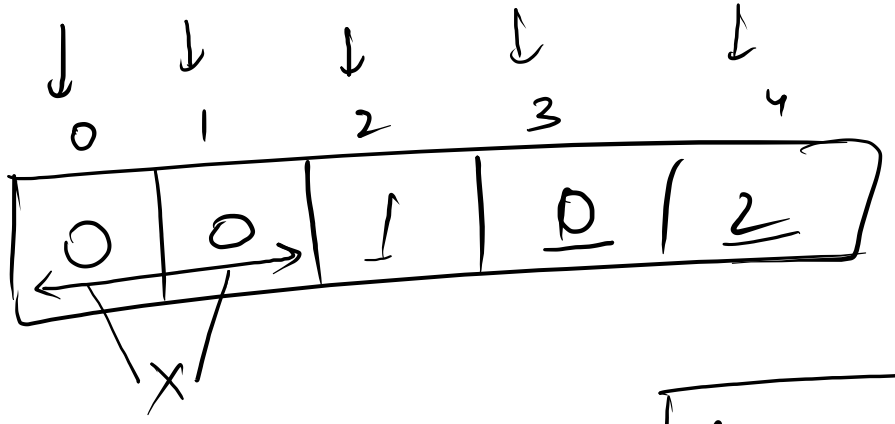
$p_1 = 2 \quad d_1 = 0$

$p_3 = 3$

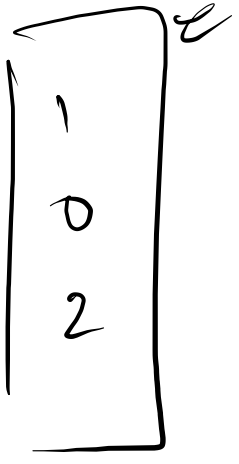
diff = d2 - d1 - borrow;
1 - 0 - 1
= 0

```java
if(diff < 0) {
    diff += 10;
    borrow = 1;
}else{
    borrow = 0;
}
```





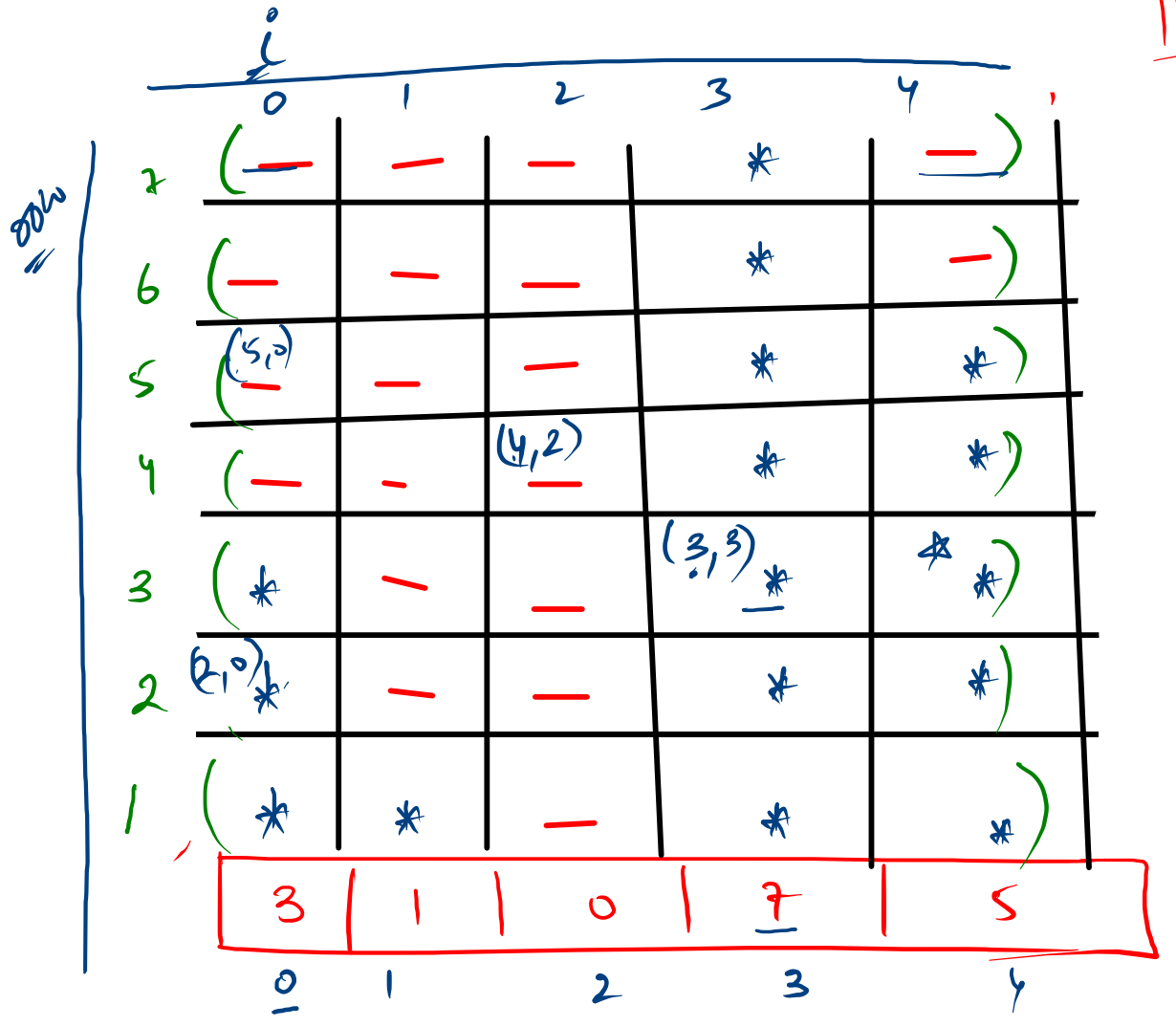
flag = ~~false~~
true




```
boolean flag = false;
for(int idx = 0 ; idx < size ; idx++){
    if(res[idx] != 0){
        flag = true;
    }
    if(flag){
        System.out.println(res[idx]);
    }
}
```

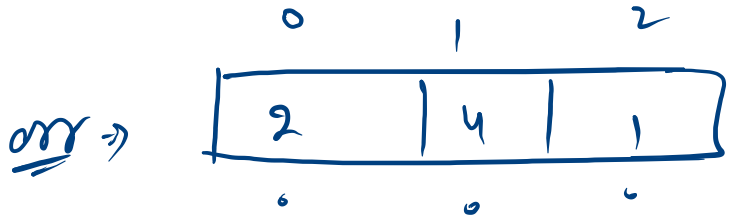
nRows =

nCol = n




```
if(arr[i] >= row){
    print(*);
}else{
    print(_);
}
```

arr[i] → stars will be printed till arr[i]




```java
int nrow = max;
int ncol = n;

for(int row = nrow ; row >= 1 ; row--){
    for(int i = 0 ; i < ncol ; i++){
        if(arr[i] >= row){
            System.out.print("*\t");
        }else{
            System.out.print("\t");
        }
    }
    System.out.println();
}
```

nrow = 4

ncol = 3

row	i	arr[i]
4	0 1 2	2 4 1
3	0 1 2	2 4 1
2	0 1 2	2 4 1
1	0 1 2	2 4 1