Analyzing constraints

Max no. of consecutive 1s after replacing

Max no. of consecutive 1s after swapping

Majority Element

Row to column zero

21 Sep → 9 PM
Thurs

25 Sep
Monday → Adv
Modul

Analyzing Constraints

1. TLE
2. Data type

- Analyze the constraints → help you determine which data structure / algo to use for a given problem
- Look at constraints

Don't ask the interviewer constraints directly.

1. Brute Force : TC and SC
2. Should I optimize further?
3. Optimized algo

Constraints	TC
$n <= 10^6$	$O(n)$ / $O(n \log n)$
$n <= 20$	$O(n!)$ / $O(2^n)$

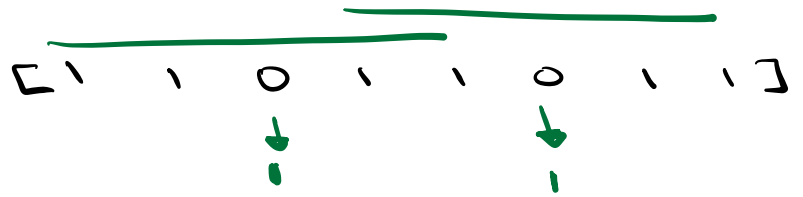
$n = 20$

$2^{10} \simeq 10^3$

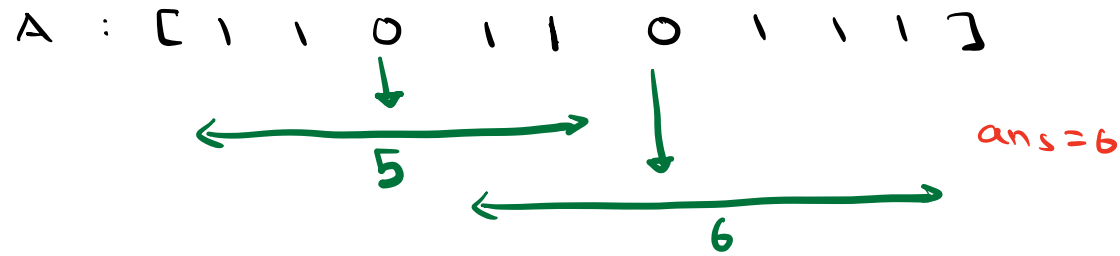
$2^{20} \simeq 10^6$

$n <= 10^{10}$ $O(\log n)$ / $O(\sqrt{n})$

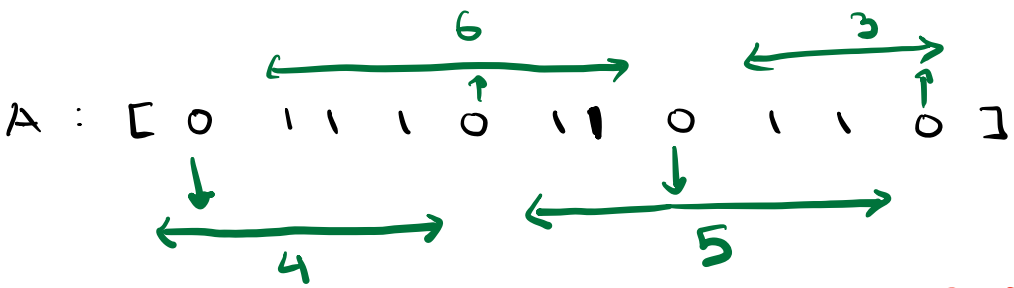
1. Given an array of 1's and 0's. You're allowed to replace only one 0 with 1. Find max. no. of consecutive 1s that can be obtained after making replacement.



ans = 5



ans = 6



ans = 6

BF : Go to every 0 and replace it with a 1.

↓

Find cnt of 1s on left side

Find cnt of 1s on right side

size of 1s = l + r + 1

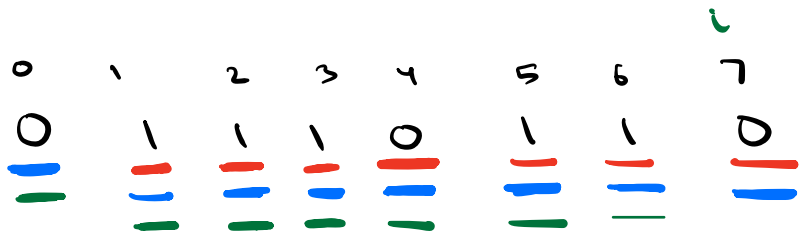
```
int findMaxOnes(int nums[]) {
    int n = nums.size()
    int ans = 0  → biggest group of 1s
    int cntOnes = 0
    for (i = 0; i < n; i++) {
        if (nums[i] == 1) cntOnes++
    }
    if (cntOnes == n) return n
    for (i = 0; i < n; i++) {
        if (nums[i] == 0) {
            int cntl = 0, cntr = 0
            int j = i-1
            while (j ≥ 0 && nums[j] == 1) {
                cntl++      j--
            }
            int j = i+1
            while (j < N && nums[j] == 1) {
                cntr++      j++
            }
            ans = max(ans, cntl + cntr + 1)
        }
    }
    return ans
}
```

A = [0 0 0 0 0] ans = 1

A = [1 1 1 1 1] ans = 5

If all array el are 1, ans = N

Lets count no. of times access each el



cnt[0] 3

cnt[3] 2

0+3+1 = 4

3+2+1 = 6

ans = 0 4 6

outer loop → i

→ j left

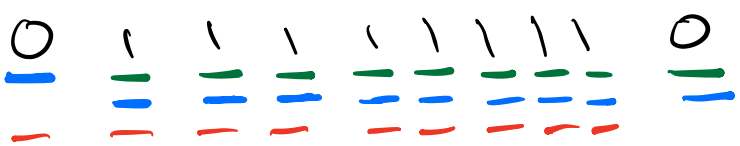
→ j right

TC : O(3n)

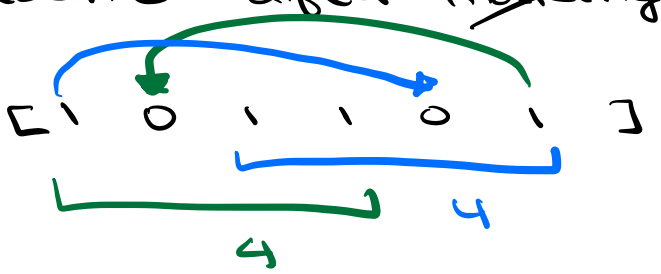
O(n)

N el → N^2

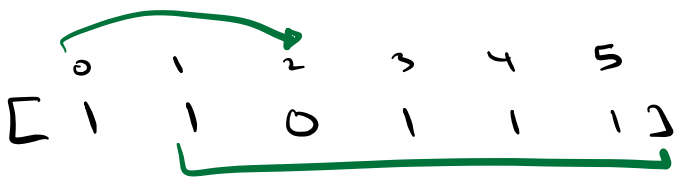
1 el → N



2. Given an array of 1's and 0's. You're allowed to swap at max one 0 with 1. Find max. no. of consecutive 1s that can be obtained after ~~making~~ replacement. swapping

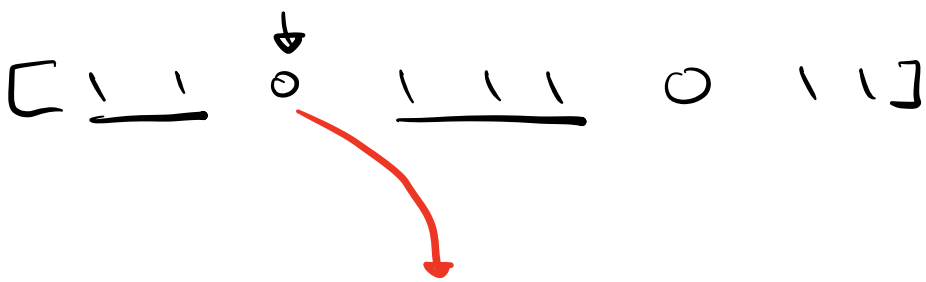


ans = 4



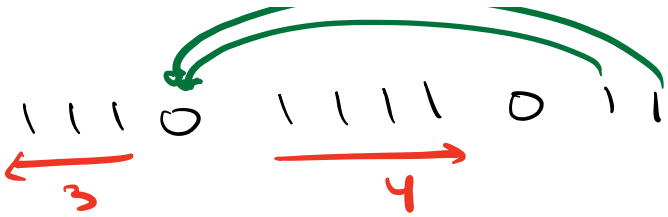
ans = 5

cntones == l+r ⇒ l+r

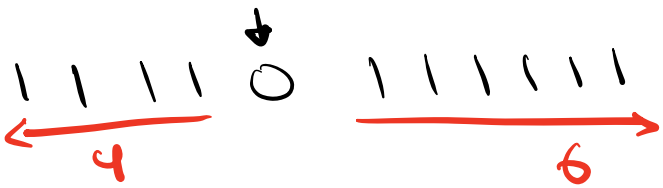


cntones > l+r → l+r+1

```
int findMaxOnes(int nums[]) {
    int n = nums.size()
    int ans = 0   → biggest group of 1s
    int cntOnes = 0
    for (i = 0; i < n; i++) {
        if (nums[i] == 1)  cntOnes++
    }
    if (cntOnes == n)  return n
    for (i = 0; i < n; i++) {
        if (nums[i] == 0) {
            int cntl = 0, cntr = 0
            int j = i-1
            while (j >= 0 && nums[j] == 1) {
                cntl++      j--
            }
            int j = i+1
            while (j < N && nums[j] == 1) {
                cntr++      j++
            }
            // swapped with extra 1
            if (cntOnes > cntl + cntr)
                ans = max(ans, cntl+cntr+1)
            else
                ans = max(ans, cntl+cntr)
        }
    }
    return ans
}
```



cntones = 9



cntones = 10

TC : O(N)

SC : O(1)

10:42

---

3. Given N elements, find majority element. Majority elements is the elements that occurs more than N/2 times where N is size of array.

freq > N/2

A = [2 1 4] — No majority element — freq > 1

ans = -1

A = [3 4 3 2 4 4 4 4]

ans = 4 — freq > 4

A = [3 3 4 2 4 4 2 4]

ans = -1 — freq > 4

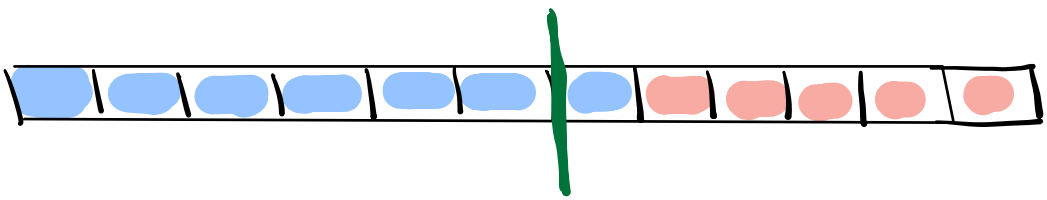
No majority element

Q1. A → 3 4 3 6 1 3 2 5 3 3 3

ans = 3 freq > 5 N = 11

At max how many majority elements can be there in an array? freq > N/2

ans = 1



Proof by contradiction

Let us say there are 2 ME ⇒ x, y

freq(x) > N/2 ⇒ freq(x) = N/2 + 1

freq(y) > N/2 ⇒ freq(y) = N/2 + 1

freq(x) + freq(y) = N + 2

BF : Check freq of each element

↙ Hashmap

TC : O(N)

SC : O(N)

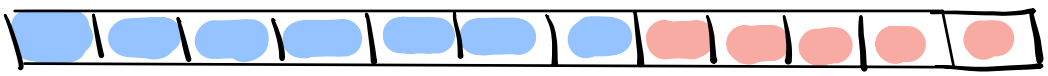
↘ Iterating in array

TC : O(N²)

SC : O(1)

Moore's Voting Algo

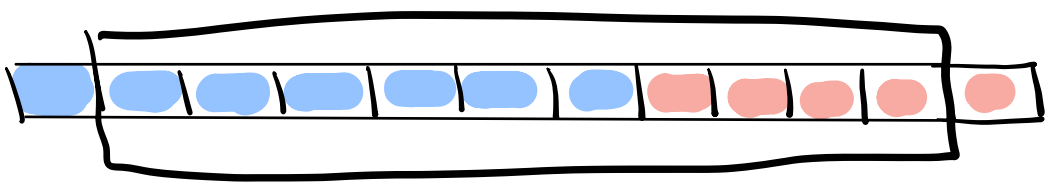
- There can only single ME
- If you remove 2 distinct ele then ME will remain same



Blue

N = 12

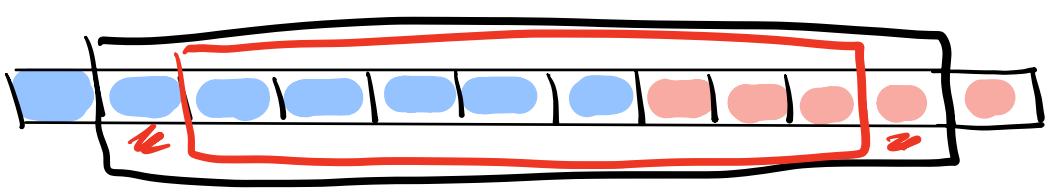
freq > 6



Blue

N = 10

freq > 5

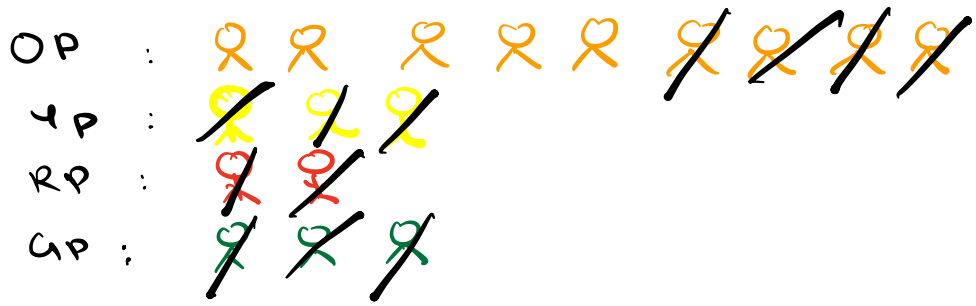


Blue

N = 8

freq > 4

4 parties in an election



OP	YP	RP	GP	Total
9	3	2	3	17

freq > 8
min freq = 9

Remove	O	Y	R	G	Winner	
OP and YP	8	2	2	3	OP	15
OP and GP	7	2	2	2	OP	13
OP and RP	6	2	1	2	OP	11
YP and RP	6	1	0	2	OP	9
YP and GP	6	0	0	1	OP	7
OP and GP	5	0	0	0	OP	5

ME = 3

ME = 1

cnt = 1

ME = 3

cnt = 1

- Iterate through each elc, keep track of ME and cnt.
- If i^th/cur elc == ME cnt++
  else cnt--
- At any point if cnt = 0
  i^th/cur elc → ME cnt = 1

```
int findCandidate(int a[], int n) {
    int majElc = a[0];
    int cnt = 1;

    for (i = 1; i < n; i++) {
        if (cnt == 0) {
            majElc = a[i];
            cnt = 1;
        }
        else if (majElc == a[i])
            cnt++;
        else
            cnt--;
    }
```

```
int cntmaj=0
for (i=0 ; i<n ; i++){
    if (a[i] == majele)
        cntmaj++
}
if ( cntmaj > N/2)
    return majele
else
    return -1
```

7 1 2 3 7 1

N = 5

freq > 2

ME 13

cnt 1 2 1

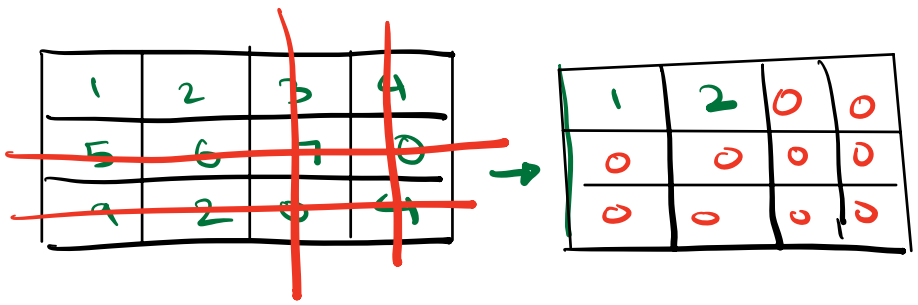
TC: O(N)

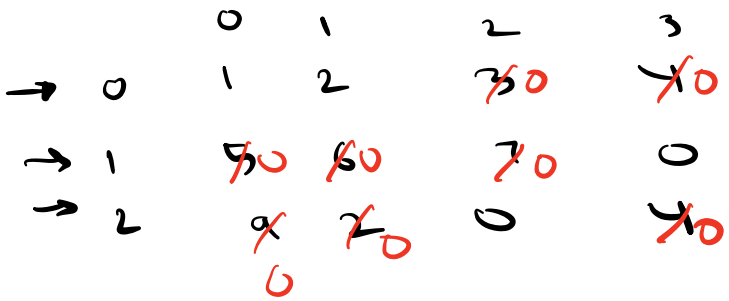
SC: O(1)

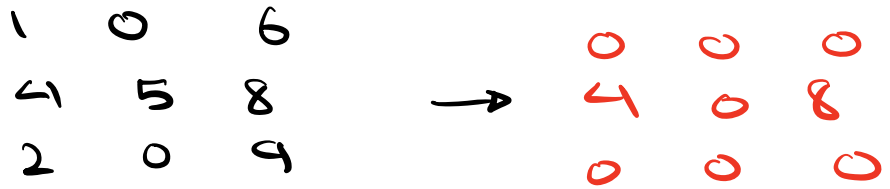
1 2 3 3 1 2

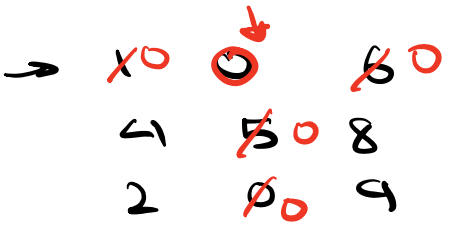
1 2 3 3 1 1

4. Given a 2D +ve integer matrix. Make all elements in ith row and jth col 0 is A[i][j] = 0



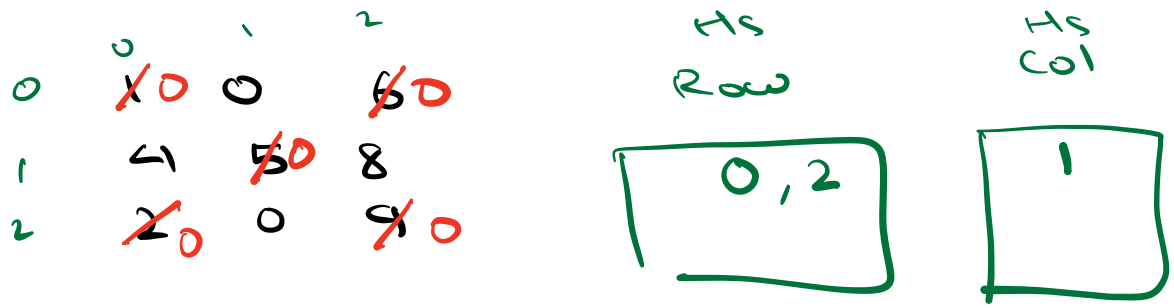






① Hashset of rows

② Hashset of cols



1. Iterate through mat
   find a cell mat[i][j]=0
   rowhs.insert(i)
   colhs.insert(j)

2. Iterate through mat
   mat[i][j]
   ↓
   rowhs.contains(i) or
   colhs.contains(j)
   T ↓ F ↘
   mat[i][j]=0 Leave as it is

TC: O(NM)
SC: O(N + M)
Row hs + col hs

Alternate For any cell = 0

Mark non-zero cells in row and col as -1

Doubts

- freq > N/2
- freq > 0

MLE → Memory Limit Exceeded

int arr [10^5 - 10^6]

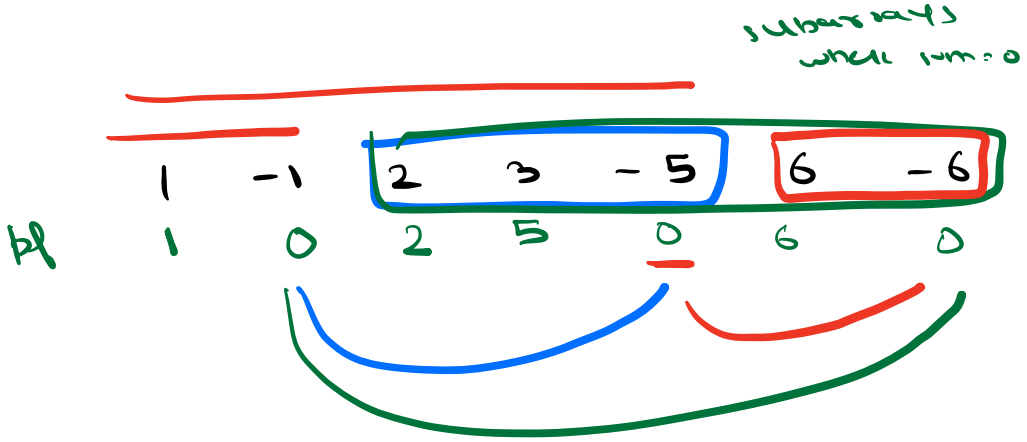
↓

4×10^6 Bytes

bool arr [10^6]

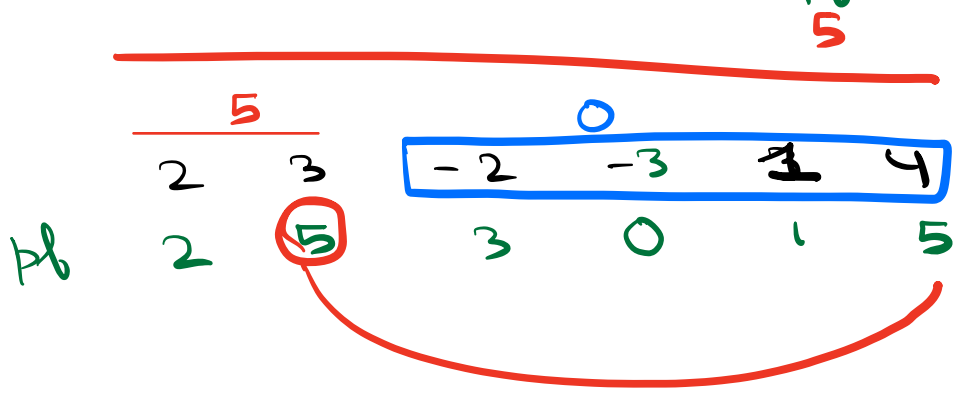
↓

10^6



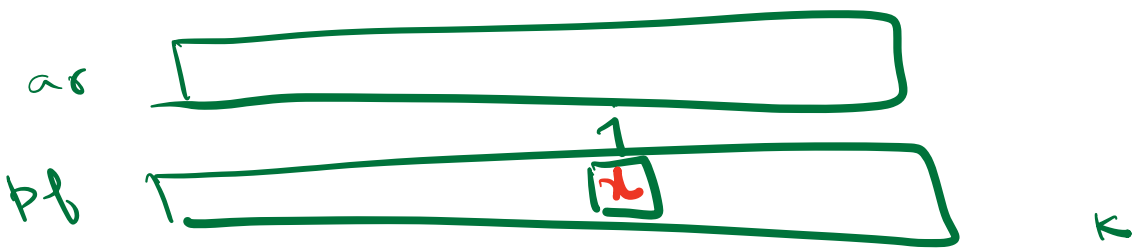
① pf[i] = 0
cnt++

② cnt pair of duplicate in pf



Map

pf, cnt



pf[i] == k

cnt++

sum(0-i)

i < j

sum(i-j) = k

pf[j] - pf[i-1] = k

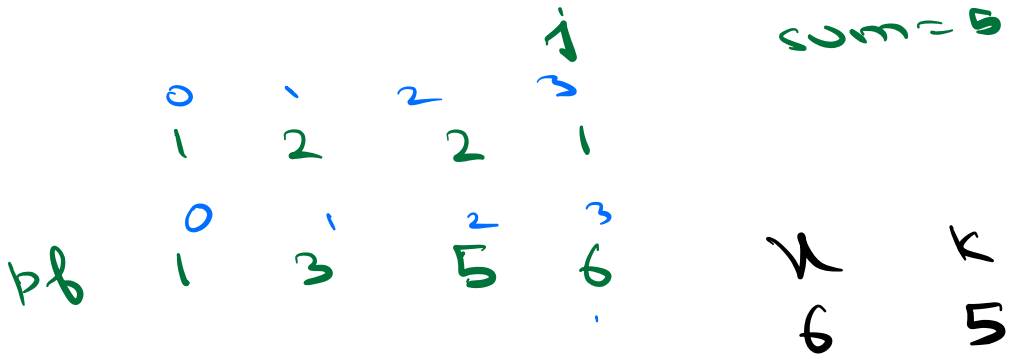
n - ⌊ ⌋ = k

pf[i-1] = n-k

j-i

i-1

n - (n-k) = k

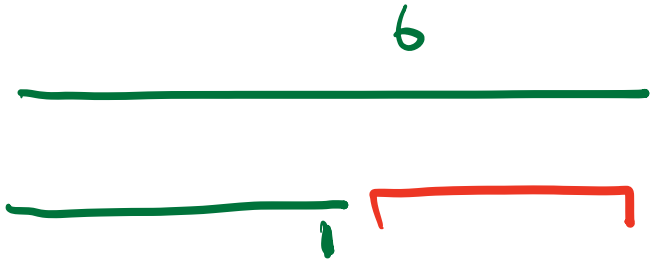


sum (i - j) = k

pf [j] - pf [i-1] = k

6 - 1 = 5

j = 3



pf [j] - pf [i-1] = k

n - ? = k

n - k = ?