Todays Content

1. Given ar[N] & k.
   Calculate length of smallest subarray with sum >= k.

2. Given a binary arr[] & k.
   Return maximum number of consecutive 1's in the array
   if you can flip at most k 0's.

Given ar[N] return length of smallest subarray with sum >=k

Constraints:

$1 <= N <= 10^6$

$1 <= ar[i] <= 10^9$

k=15   0 1 2 3 4 5 6 7

#Ex1 ar[8] = {3 2 4 5 2 6 5 6} ans=3

k=20   0 1 2 3 4 5 6 7 8 9

#Ex1 ar[8] = {3 2 4 5 2 6 8 4 5 3} ans=4

Idea1: Generate all subarrays TC: $O(N^2 * N) = O(N3)$ SC: O(1)

```
For every subarray iterate & cal sum >= k.
if sum >= k
    ans = min(ans, #subarray length)
```

Idea2: Generate all subarrays TC: $O(N^2 * 1 + N) = O(N2)$ SC: O(N)

```
For every subarray cal sum using pf[] >= k.
if sum >= k
    ans = min(ans, #subarray length)
```

Ideas:

Target: length of smallest subarray with sum >= k

Search Space: l h
{0 N}

```
            0  1  2  3  4  5  6  7  8  9
ar[10] = {  3  2  4  5  2  6  8  4  5  3 }
k = 20
```

```
l   h   m   # Subarray len:  F .. F  F  F  T  T T . .  T
0   10  5   Is there a subarray of len = m with sum >= k.
            m   m+1 m+2 ..     # If subarray of len : m, with sum>=k
            5   T T T T          exists, then subarray of
            ans = m; h = m-1;    len > m+1, m+2 .. with sum >= k
                                 also exists.

0   4   2   Is there a subarray of len = m with sum >= k.
            m-2 m-1  m         # If subarray of len : m, with sum>=k
             F   F   2           doesn't exist, then subarray of
            l = m+2              len = m-1, m-2.. , with sum >= k
                                 also doesn't exist
```

```
int smallest(vector<int> &arr, int k){   TC: O(Nlog N)  SC: O(1)
    int l=0, h= arr.size(), ans=0;
    while(l<=h){
        int m = (l+h)/2;          TODO: Sliding Window
        if(check(arr,m,k)){  # Check if there exists a subarray of
        }    ans=m; h=m-1;         len = m, with sum>=k.
        else{
    } }  l=m+1;
    return ans;
}
```

#ideas:

k = 20   0 1 2 3 4 5 6 7

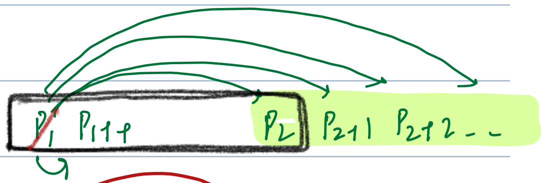
arr[10] = { ~~3~~ ~~2~~ ~~4~~ ~~5~~ 2 6 8 3 }

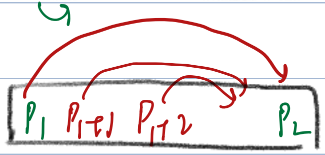
P_1 P_2

Valid   ans = INT_MAX   sum = ~~0~~ ~~3~~ ~~5~~ ~~9~~ ~~14~~ ~~16~~ ~~22~~ ~~19~~ ~~27~~ ~~25~~ ~~21~~

~~16~~ 19

P_1	P_2	sum>=k		
0	-1	0>=20		Inc P2++ update sum;
0	0	3>=20		Inc P2++ update sum;
0	1	5>=20		Inc P2++ update sum;
0	2	9>=20		Inc P2++ update sum;
0	3	14>=20		Inc P2++ update sum;
0	4	16>=20		Inc P2++ update sum;
0	5	22>=20	ans = 6	remove arr[P1] then P1++
1	5	19>=20		Inc P2++ update sum;
1	6	27>=20	ans = 6	remove arr[P1] then P1++
2	6	25>=20	ans = 5	remove arr[P1] then P1++
3	6	21>=20	ans = 4	remove arr[P1] then P1++
4	6	16>=20		Inc P2++ update sum;
4	7	19>=20		Inc P2++ if (P2 == N) { break }





```
int smallest( vector<int> &arr, int k){
    int p1=0, p2=-1, ans=INT_MAX, sum=0;
    while( p2 < N){
        if(sum >= k){
            ans = min(ans, p2-p1+1)
            sum = sum - arr[p1];
        }   p1++;
        else{
            p2++;
            if( p2 == N){ break;}
            sum = sum + arr[p2];
        }
    }
    return ans;
}
```

#2 Pointers on subarrays:

Type 1: 2 Pointers:

If [i..j] is valid
    and we can prove that {i j j+1 j+2 ... }
Implies
If [i..j] is Invalid
    and we can prove that {i i+1 i+2 j}

We can apply below 2 pointers logic.

Pseudo Code:

```
int P1=0, P2=-1;
int ans=0, N=arr.size();
while(P2<N){
    if( valid[P1..P2] ){
        update ans;
        remove arr[P1] in subarray
    } P1++;
    else{
        # [i..j] invalid
        P2++; ——→ # goto next sub array [i..j+1]
        if( P2==N){ break}
        Add arr[P2] in subarray → # New ele in subarray.
    }
}
```

If a subarray shows any one of below pattern we can apply 2 pointers

Type

Case : 1

Type1:

2 Pointers:

If [i..j] is valid implies that
{i i+1 i+2 ... j} Valid

or

If [i..j] is invalid implies that
{i... j j+1 j+2} Invalid

```
i=0, j=-1;
while(j<N){

    if(valid(i..j)){
        update ans;
        j++;
        if(j==N){break;}
    } Add ar[j] in subarray
    else{ # [i..j] invalid
        remove ar[i] in subarray
        i++;
    }
}
```

Case : 2

Type1:

2 Pointers:

If [i..j] is valid implies that
{i   j j+1 j+2...}

or

If [i..j] is invalid implies that
{i......, j-3 j-2 j-1 j } invalid

```
i=0, j=-1;
while(j<N){

    if(valid(i..j)){
        update ans;
        remove ar[i] in subarray
    } i++;
    else{ # [i..j] invalid
        j++;
        if(j==N){break;}
        Add ar[j] in subarray
    }
}
```

28 Max Consecutive Ones:

Given a binary arr[] and an integer k.
Return maximum number of consective 1s in the arr[],
If you flip at most k 0's.

0 1 2 3 4 5 6 7 8 9 10 11

Ex1: arr[] {1 ~~0~~ 1 1 ~~0~~ 1 0 0 1 1 1 0} ans = 6.

k=2 1 1 1 1 1 1

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Ex2: arr[] {1 0 1 1 ~~0~~ 1 ~~0~~ 1 1 ~~0~~ 1 1 0 1 0 0} ans = 10

k=3 1 1 1 1 1 1 1 1 1 1

Rewrite above Question:

Given a binary arr[] and an integer k.
Return length of longest subarray with all 1s.
If you flip at most k 0's.

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Ex2: arr[] {1 0 1 1 0 1 0 1 1 0 1 1 0 1 0 0} ans = 10

k=3

#2 pointer:

Entire subarray can make 1

Assume sub [i i+1 i+2 j] is valid

{

Assume sub [i j j+1 j+2..] is invalid

0 1 2 3 4 5 6 7 8 9 10

Ex2: arr[ ] {1 0 1 0 0 1 1 0 1 1 0}

k = 2

Valid ans co

P_1 P_2 co <= k

(0 -1)

```
int consecutiveSwaps (vector<int> row, int k){
}
```