Q.1 Google Given an Array of +ve no's. find the max. length (K), such that there exists NO subarray of length k with sum >= B.

* All subarrays of length (K) have sum < B.

A :	3	2	5	4	6	3	7	2
	0	1	2	3	4	5	6	7

B = 20

× K=5 ⇒ [0-4] : sum = 20 = B.

× k=4 ⇒ [0-3] : 14 < B
[1-4] : 17 < B
[2-5] : 18 < B
[3-6] : 20 = B ×

K=3 ✓ : All subarrays of length = 3 have sum < B.

Brute force

⇒ Iterate over k = N to 0 :
⇒ Check if K_i satisfies the condition.
(All subarrays of len = K_i have sum < B)

$$\boxed{\begin{aligned} K_{max} &= N \\ K_{min} &= 1 \end{aligned}}$$

⇒ find the sum of every subarray of length (K).

Ex

$$A : \overset{0}{3},\ \overset{1}{2},\ \overset{2}{5},\ \overset{3}{4},\ \overset{4}{6},\ \overset{5}{3},\ \overset{6}{7},\ \overset{7}{2}$$

K = 4

$Sum_1 = 14$

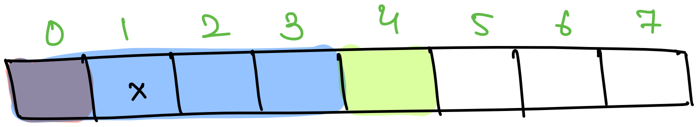
$Sum_2 = Sum_1 - A[0] + A[4]$

$= 14 - 3 + 6 = 17$

⇒ Sliding Window | Prefix Sum.

⇒ Write a fun^n which returns true if a given length K satisfies the condition

All subarrays of length K have sum < B.



N = 8
K = 4

1^{st} window : [0-3] ⇒ S

2^{nd} window : [1-4] ⇒ $S - \overset{i-K}{A[0]} + \overset{i}{A[4]} = S_1$

3^{rd} window : [2-5] ⇒ $S_1 - A[1] + A[5] = S_2$

4^{th} window : [3-6] ⇒ $S_2 - A[2] + A[6] = S_3$

5^{th} window : [4-7] ⇒ $S_3 - A[3] + A[7]$

```
bool  check ( A[] , K , B ) {
      // Get the sum of first window
      // of size K => sum.
      if( sum >= B)
          return false;
      for( i=K; i<N ; i++ ) {
            sum += A[i];
            sum -= A[i-K];
            if( sum >= B)
                return false;
      }
      return true;
}
```

Iterate over all possible values Ⓚ : [N,1]

← linear iteration →

```
for( K= N; K >= 1; K--) {
      if( check( A, K, B))
            return K;
}
```

TC : O(N^2)

SC : O(1)

Using PS

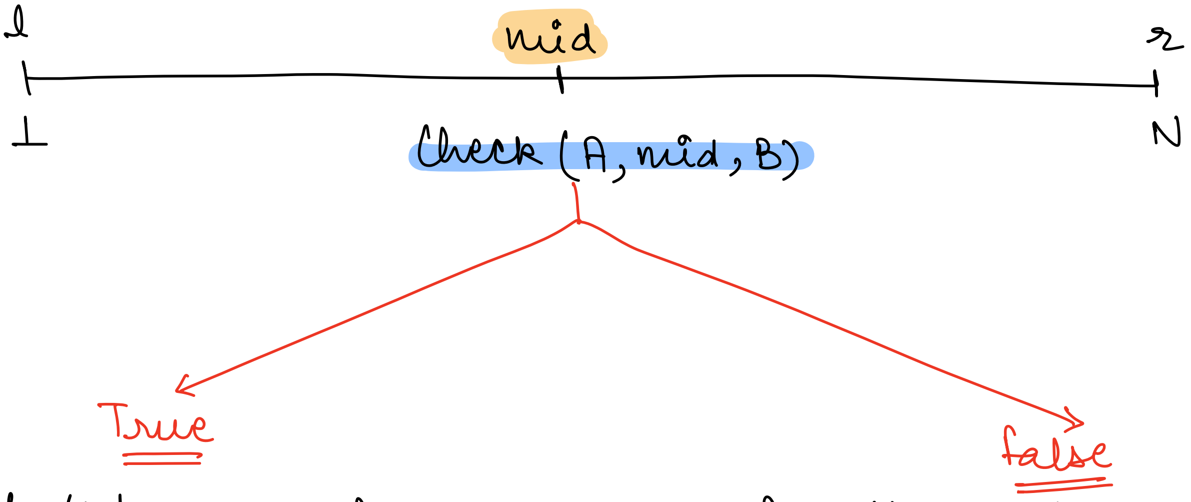
TC : O(N^2)

SC : O(N)

⇒ **Target** : K_{max} which satisfies the condition.

Search space :- $[\overset{K_{min}}{1}, \overset{K_{max}}{N}]$

$$\boxed{\begin{aligned} ans_{min} &= 1 \\ ans_{max} &= N. \end{aligned}}$$



* All subarrays of length = mid have sum < B.

* ans = mid

* Move to right

* l = mid + 1

* All the subarrays of len = mid are NOT satisfying the condition.

* Move to left.

* r = mid-1;

Ex A : $\overset{0}{3}, \overset{1}{2}, \overset{2}{1}, \overset{3}{5}, \overset{4}{4}, \overset{5}{7}$

K ∈ [1, 6] , B=6

[1-6]

mid = 3 = K ⇒ Check(A, mid, B)

↳ false

[1-2]

mid = 1 ⇒ Check(A, mid, B)

↳ false;

Iterate over values of (K) in a BS way.

```
int BSonAns( A[], B) {
    l = 0, r = N; ans;
    while( l <= r) {
        mid = (l+r)/2;
        if ( Check( A, mid, B)) {
            ans = mid
            // move to right side for
            // better ans.
            l = mid + 1;
        }
        else{
            r = mid-1;
        }
    }
    return ans;
}
```

$$\begin{array}{lcccccccc} & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ A: & 3, & 2, & 5, & 4, & 6, & 3, & 7, & 2 \end{array}$$

B = 20

$K \in [0, 8]$

mid = 4 ⇒ check(A, 4, 20)

S1 = 20 >= B ⇒ false

[0, 3]

mid = 1 ⇒ Check(A, 1, 20)

↳ true

ans = 1

[2, 3]

mid = 2 ⇒ Check(A, 2, 20)

↳ true

ans = 2

[3, 3]

mid = 3 ⇒ check(A, 3, 20)

true

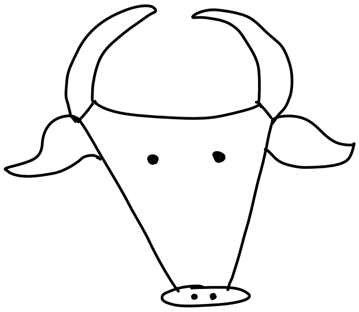
ans = 3

[4, 3] ⇒ l > r ⇒ Break

$$\boxed{\begin{array}{l} TC : O(N\log N) \\ SC : O(1) \end{array}}$$

Q.2 Aggressive Cows

Google



Cow | Bull

Given :-

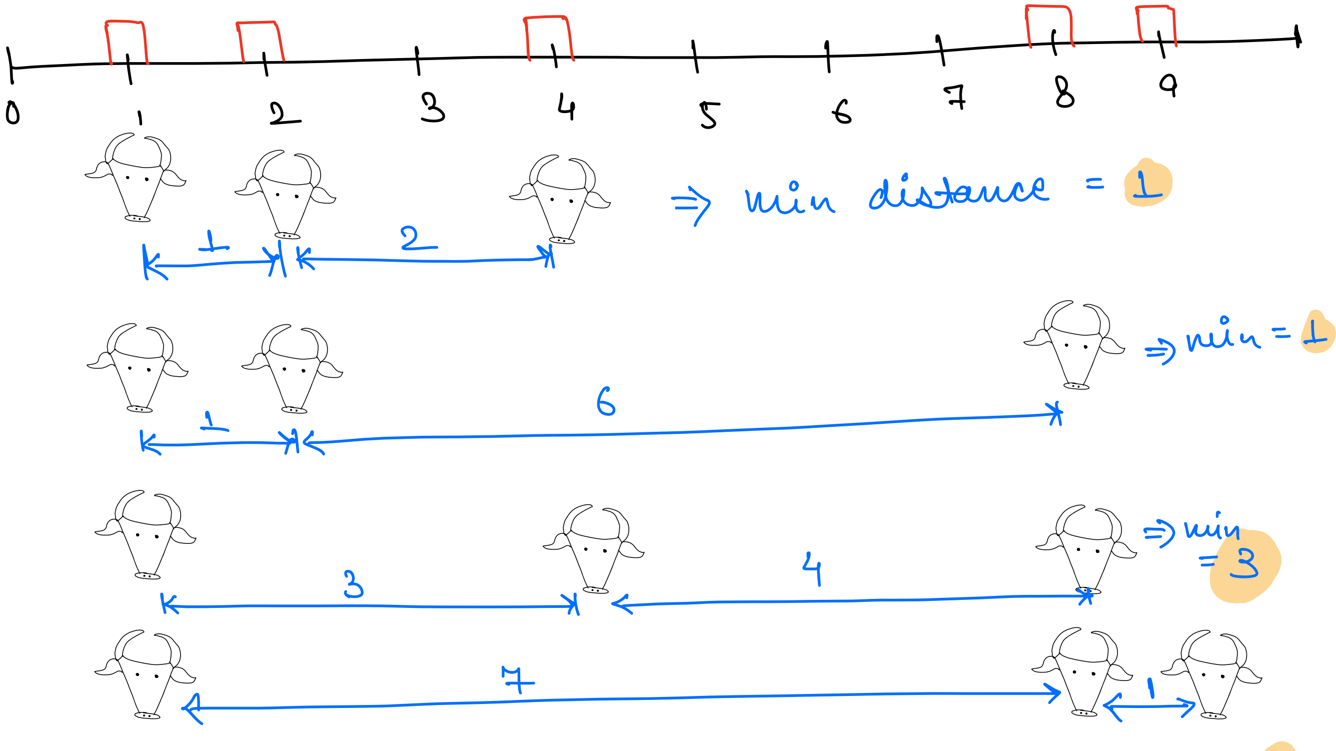
1) A sorted array of +ve no's having the positions of rooms we can keep a cow.

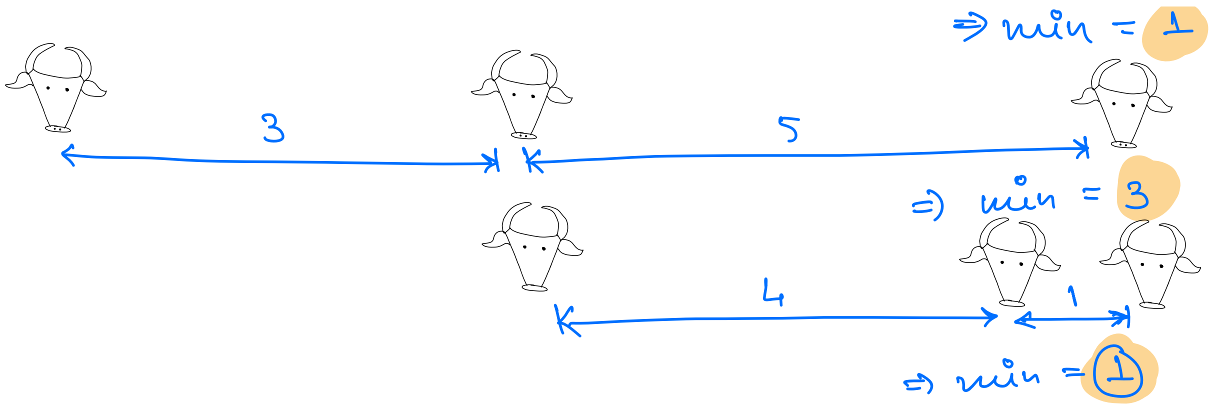
2) K ⇒ No. of cows. (K <= N)

Return the maximum value of minimum possible distance b/w any two cows.

	0	1	2	3	4	
A :	1,	2,	4,	8,	9	] Room positions.

K = 3





$\boxed{ans = 3}$

⇒ Place K cows in N rooms such that the distance b/w any two closest cows is maximum.

Brute force :-

⇒ Try all the combinations :-

→ No. of ways to place K cows in N positions :

$$\Rightarrow {}^{N}C_{K} = \frac{N!}{K!\,(N-K)!} \approx N!$$

* Iterate over all the ${}^{N}C_{K}$ combinations & keep updating the minimum distance.

distance b/w the closest cows.

TC: $O(N! * N)$

③ ans = x

Sqrt(N) ?

$\sqrt{N}$: 1 × 1 = N ✗

2 × 2 = N ✗

3 × 3 = N ✗

⋮

$i * i = N$

N=100

1×1 = 100 ✗

2×2 = 100 ✗

3×3 = 100 ✗

⋮

9×9 = 81 ✗

⑩×10 = 100 ✓

Target :- max value of i such that i × i <= N.

# Target :- distance b/w the closest cows.

$ans_{max} = A[N-1] - A[0]$ } when we have two cows.

$ans_{min} = 1$

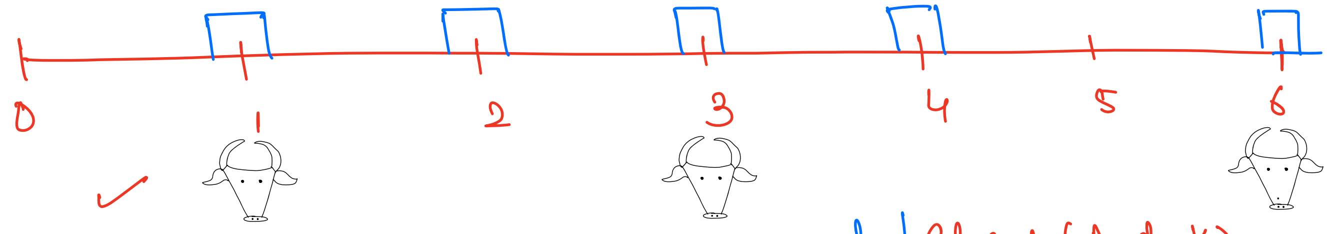
⇒ Iterate over the range of ans and check if we can place Ⓚ cows with this as distance b/w two closest cows.

```
for ( d= ansmax ; d >= ansmin ; d--) {
    // Check if we can place K cows
    // maintaining a minimum distance of d.
    if ( Check( A, d, K) ) {
        return d;
    }
}
```

Ex

	0	1	2	3	4
A :	1	2	3	4	6

K=3



ansmax = 5

ansmin = 1

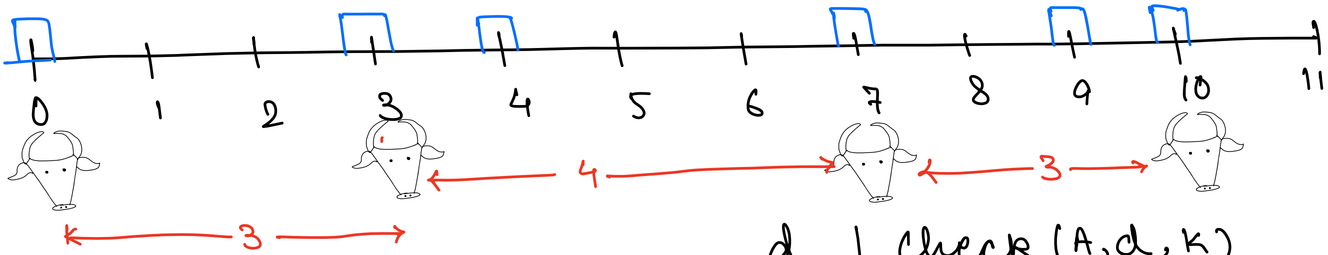
d	Check(A,d,K)
5	✗
4	✗
3	✗
2	✓

ans.

Quiz

A = 0, 3, 4, 7, 9, 10

K = 4.



d	check (A, d, K)
10	✗
9	✗
8	✗
7	✗
6	✗
5	✗
4	✗
3	✓

ans.

$ans_{max} = 10$

$ans_{min} = 1$

```
bool check ( A[], d, K) {
    // returns true if it is possible to
    // place K cows maintaining minimum
    // distance of atleast d.
    int prevPos = A[0];
    int cowsPlaced = 1;
    for( i= 1; i< N; i++ ){
        if ( A[i] - prevPos >= d ) {
            cowsPlaced ++;
            prevPos = A[i];
        }
        if ( cowsPlaced == K) {
            return true;
        }
    }
    return false;
}
```

TC of check( ) fun^n = $O(N)$

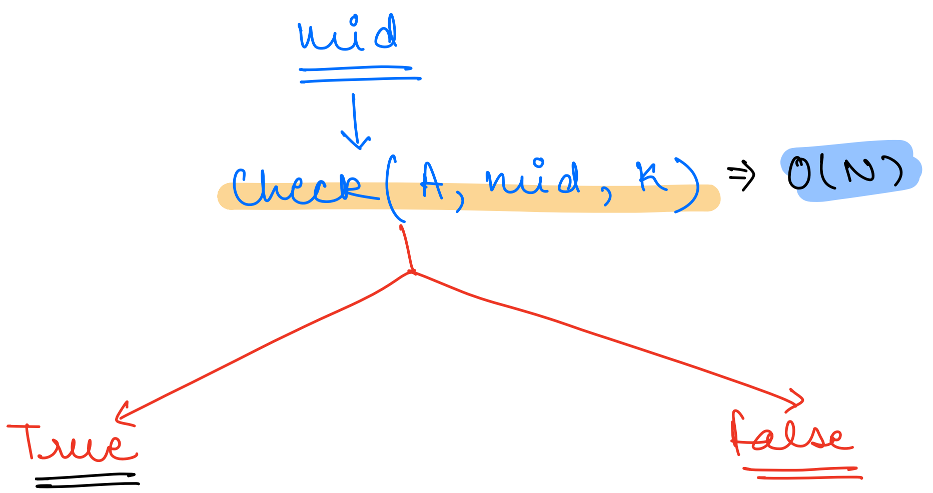
Total TC of this sol^n = $O(R * N)$

Range of the ans:

$[1, A[N-1] - A[0]]$

Target ⇒ distance b/w the closest cows.

Search space ⇒ $[1, A[N-1] - A[0]]$



True

- * Store (d) in the ans
- * Search for larger value of d.
- * Move to right.

ans = mid.

l = mid+1.

false

- * Move to left
- * r = mid-1;

$TC : O(N * \log_2 R)$

$SC : O(1)$