Searching -2

→ B.S only sorted Arrays? NO!

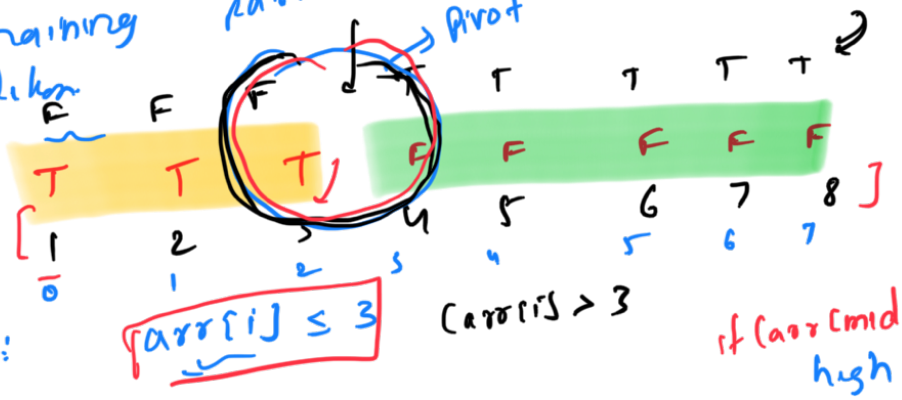
When can we apply Binary Search?

→ The main condition to apply B.S is the search space (Array) should be monotonous.

A = [3, 6, 7, 9, 13]

→ The place where we would want to search.

Monotonicity: If we are able to come up with a condition (f(x)) such that one part of the search space satisfies this condition, and the remaining part does not satisfy the condition.



Condition: arr[i] ≤ 3 arr[i] > 3

arr[0] ≤ 3 ? YES

if (arr[mid] ≤ 3)
high = mid-1
else
low = mid+1

these are called Monotonous search space.

if (arr[mid] ≤ C)
high = mid-1
else
low = mid+1.

Ex2: Search in sorted - Rotated Array

A = 

Step1 : Find No of times the array was rotated = K
↓
Index of 1st element of Right sorted Array

Condition

if(A[i] ≥ A[0])
Left Sorted Array

if(A[i] < A[0]
Right Sorted Array

→ We apply Binary search to find this pivot point.

Questions:

Given an array of size N (positive elements) and sum S, find the maximum K such that every subarray of size K has a sum ≤ S.

K : [ ]

S = 25

A = 1 20 3 7

K = 1 [1] [20] [3] [7] ✓

K = 2 [1,20] [20,3] [3,7] ✓

→ K = 3 [1,20,3] [20,3,7] ✗

K = 4

If $K = 3$ is not an answers, $K = 4, 5, 6, \ldots$ cannot be answer

If $K = 2$ is an answer, $K = 1, 0, \ldots$

$K = 10$ is an answer

$\Rightarrow K = 9, 8, 7, 6, 5, 4 \ldots$ will definitely be an answer

Binary Search on the Answer Space

Answer Space: Set of all possible answers

$A = [20, 30, 35, 19]$

$S = 10 \Rightarrow K = 0$

$Min(K) = 0$

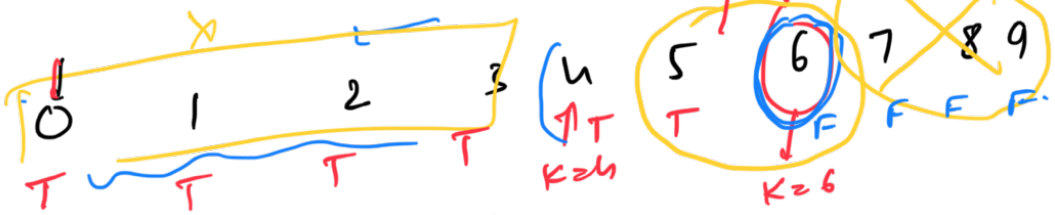
$A = [20, 30, 35, 19]$

$S = 200$

$Max(K) = N$ (size of Array)

→ Idea is the Answer space has some sort of Monotonicity / ordering

$N = 9$



Answer space

Answers: $K = 5$

Can K be the answer.

Condition:

→ Apply Binary Search on the Answer space

→ d Maximum / Minimum => there is definely a solution using Binary search

→ but we cannot guarantee it is the optimal solutions.

sum = 24 k = 3

A = [5, 17, 2, 4, 6, 8]

0 1 2 3 4 5

24 - 5 + 4 = 23

23 - 17 + 6 = 12

12 - 2 + 8 = 18

O(n)

N = 6

S = 24

Search space: [0, 6]

low high

O(n)

low	high	mid
0	6	3
4	6	5
4	4	4
4	3	

Action

ans = 3, low = mid + 1.

high = mid - 1

high = mid - 1. (Go Left)

ans = 3

k.

4 5 6

4 X X →

Right most

A → 5 17 2 4 6 8

Pre → 5 22 24 28 54 42

0 1 2 3 4 5

pre[2]

O(n)

Extra

pre[3] - pre[2]
pre[4] - pre[2]

Space

Array Size?

T.C:

Search Space => Answer Space => [0, N]

low → 0, high → N

Iterations = $\log_2(N+1)$

$\log N$

T.C: O(log M x T.C for feasibility check)

M => Size of Answer space

N

T.C: O(logN * N) = O(NlogN)

Answer space: [0, N] => (N+1)

Min, Max

Question: Aggressive Cows

-> N stalls placed on x-axis

-> K cows to be placed in these stalls

$(2 \leq K \leq N)$

Task: Place cows as far as possible from each other.

N = 5, K = 3

stalls[] : 1 2 4 8 9

C_1 1 C_2 4 C_3

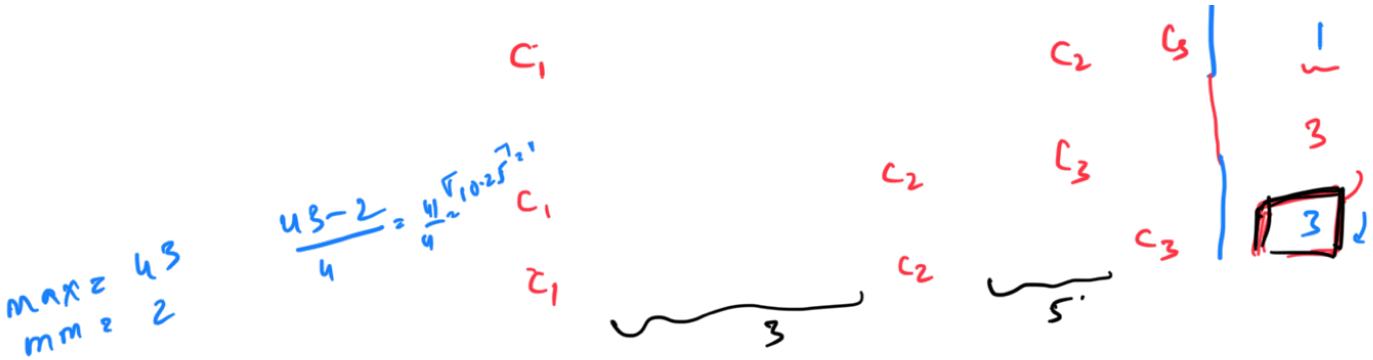
4, 1 => 1

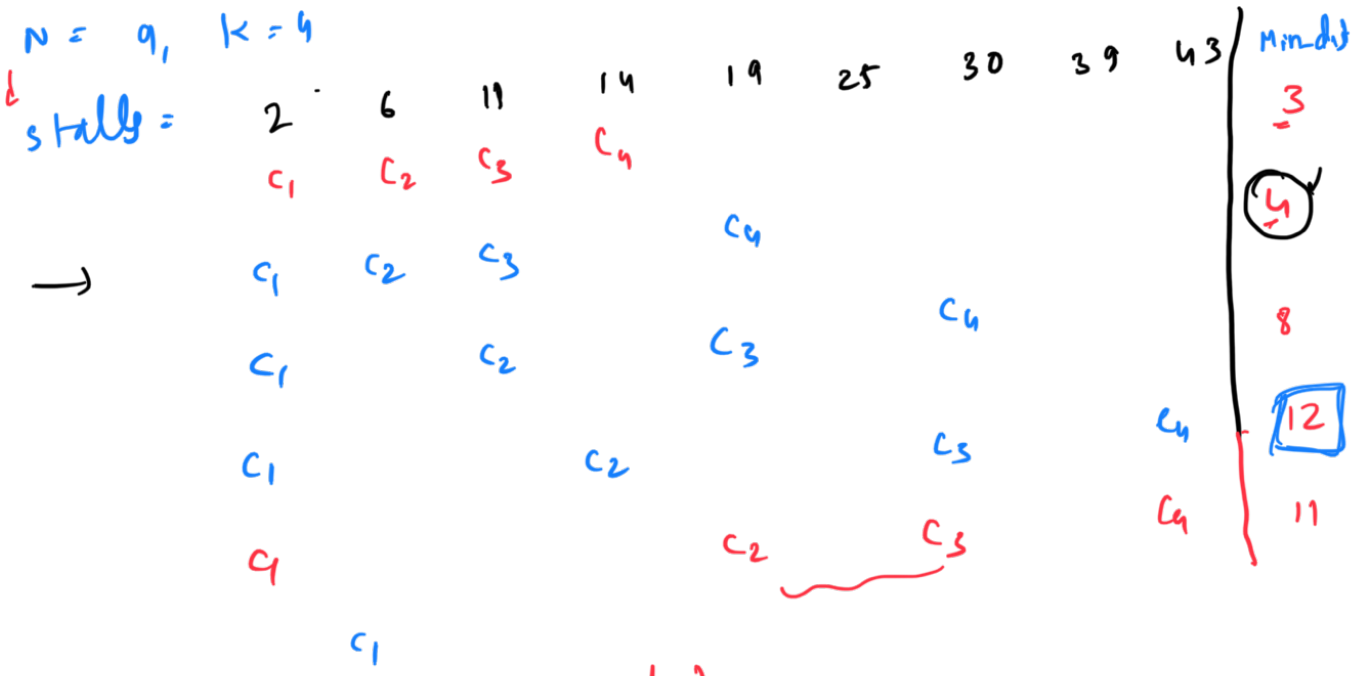
C_1 C_2 C_3

min dist

1

C_1 1 C_2 2 C_3





Ans = 12 (Max value)

→ We cannot place K cows such that they're atleast 13 units apart

Suggested Approach 1: Wrong

Place the cows at a distance (max - min)/k

Brute force:

consider all arrangements → (Back track)

$\binom{n}{k} \times O(n)$

$O\left(\binom{n}{k} \cdot O(n)\right)$ ⇒ Exponential

(k = n/2)

$$\frac{n!}{k!(n-k)!} \sim O(n!)$$

Better Approach:

Answer Space:

Answers ⇒ max of min distances

↘ Distance.

high: (max - min) : stalls[n-1] - stalls[0]

A = 1 3 4 10 13 14 k = 2

c_1 c_2.

14 - 1 = max - min = 13

low: Min dist between any 2 consecutive

k = 6

A = 1 3 4 10 13 14

c_1 c_2 c_3 c_4 c_5 c_6

Min distance between consecutive stalls

A = 1 5 9 13 16 18

low = 2

Search Space

low = Min dist b/w any 2 consecutive d

high = max - min

47 =

Simpler Question:

Check if you can place K cows where any 2 cows are atleast ' P ' distance apart

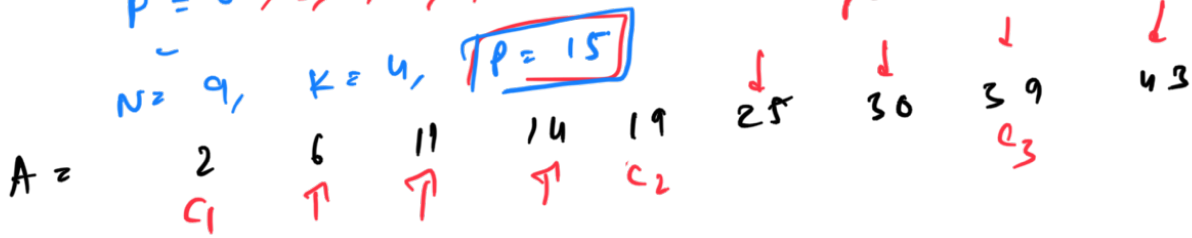
Return True

cows_placed = 5
position_cow = 39

P = 7, we are able to place 4 cow

P = 6, 5, 4, 3, 2, 1 => You can definitely place K cows

N = 9, K = 4, P = 15



cows_placed = 3
pos = 39

30 - 19 = 11

=> P = 15 cannot be an answer. 39 - 19 = 20

Can't place 4 cows such that any 2 cows are 16 units apart!

P = 16, 17, 18 ... cannot be answer

```
int check ( stalls [], N, K, P ) {
    cows_placed = 1
    prev_pos = stalls [0];
    for( i = 1; i < N; i++ ) {
        if stalls [i] - prev_pos >= P ) {
```

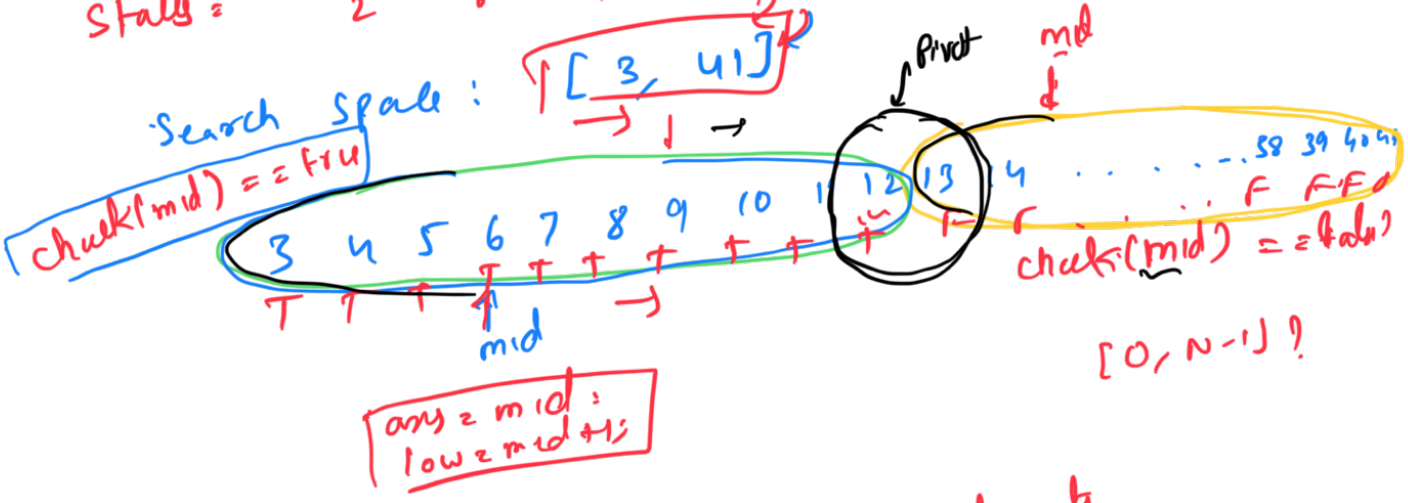
```
        cows_placed++;
        prev_pos = stalls[i];
    }
}
if(cows_placed >= k)
    return true;
else
    return false;
}
```

T.C: O(n)

K = 4

Stalls = 2 6 11 14 19 25 30 39 43

{ans = 12}




```
ans = mid;
low = mid+1;
```

low = min of consecutive elements

high = max - min.

```
while (low <= high) {
    mid = (high + low)/2;
    if(check(mid) == true) {
        ans = mid;
        low = mid+1;
    }
    else
        high = mid-1;
}
return ans;
```

((arr[N-1] - stalls[0]) - ---)) x O(n)

Size ∝ S.S: log(stalls[n-1] - stalls[0]);

Positions of stalls

T.C: O(n * log(stalls[n-1] - stalls[0]));

Question: Square Root of N

√10 = 3

T.C: O(log n)

For N=1, √10 = 3.15

low = 1
high = n

n = 10

mid for high

1 2 3 4 5 6 7 8 9 10

5 * 5 = 25 > 10

2 * 2 = 4

ans = 2
low = mid + 1

double low = 0, high = N

while (high - low ≤ 0.001) { } 3.13

mid = (high + low)/2;

Question: Painter's Partition

→ there are N boards each with a specified length.

→ there are K painters (K ≤ N)

→ One painter will only paint continuous boards

→ It takes 1(s) to paint 1 unit of board

→ Run time to get all the boards painted.

N = 4, K = 2

boards = 10 20 30 40

10+40, 20+30

→ P1 P2 P2 P2

→ P1 P1 P2 P2

→ P1 P1 P1 P2

P1: 10
P2: 90

Ans = 60

max Time

90
70
60

max

P1 : 60
P2 : 40

Can I paint all the boards in 5 hrs?

Brute force:

Consider possible arrangements.

Exponential ⇒ Back Tracking

Efficient Approach:

Search Space: (Answer Space)

high = sum(A) K = 1

A = 10 20 30 40

P1 P1 P1 P1

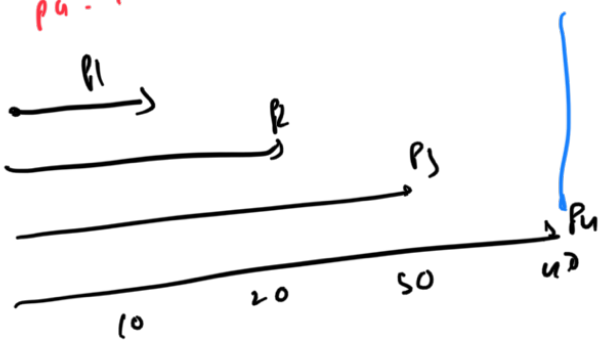
low = max(A) ⇒ (max(A)) K = 4 ⇒ N = 4, K = 4

A = 10 20 30 40

P1 P2 P3 P4

P1: 10 ✓
P2: 20 ✓
P3: 30 ✓
P4: 40 ✓

Search space: [max(A), sum(A)]:



Simpler Question:

Given n boards and k painters, check if painters can paint all the boards within time = mid

if you ... where each painter ... for P seconds

P = 80

P = 90

K = 2

A = 10 20 30 40

P1 = 10 + 20 + 30 [60] } P = 80 ✓

P2 = 40 [40]

→ Can we paint all boards with K painters such that every painter paints for atmost 90(s) ? ✓

K = 2

P = 50

A = 10 20 30 40

K = 2

P1 = 10 + 20 }

P2 = 30

P3 = 40

painters = 3

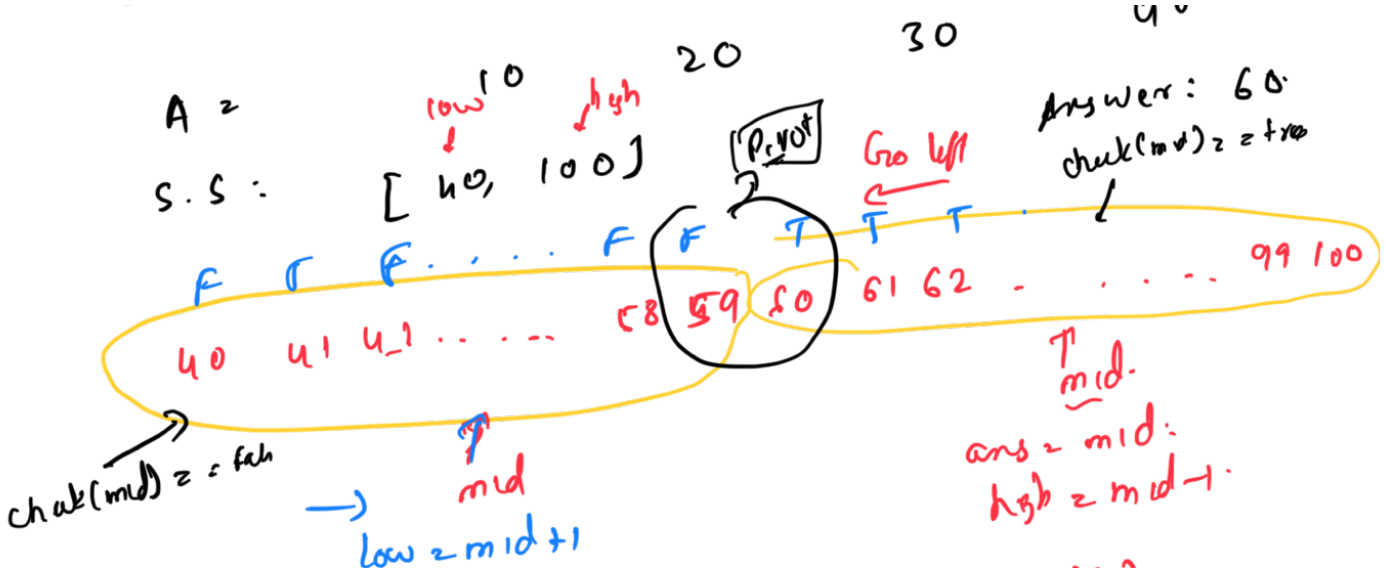
We cannot paint for P = 50

→ Can we paint all boards with 2 painters where each paint paint for atmost 110(s)?

P = 90, 91, 92 ... ...

✗ Not possible

Search space: [max(A) - sum(A)]



low = max(A), high = sum(A)

while (low <= high) {

mid = (high + low) / 2 ;

if (check(mid) == true) {

ans = mid ;

high = mid - 1 ;

} else {

low = mid + 1 ;

}

}

return ans ;

Size of Search space : sum(A) - max(A)

T.C : O(n x log(sum(A) - max(A)))

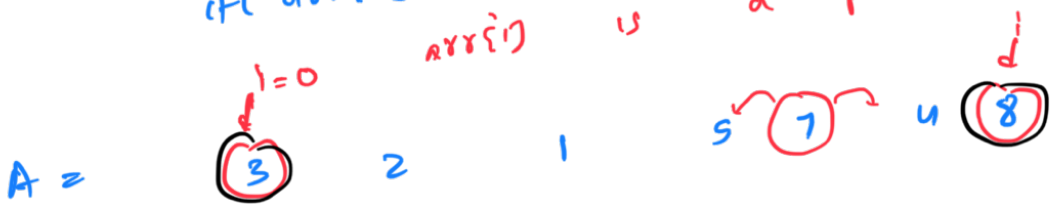
Question : Peak element in Array

-> Given array of distinct elements find a peak element. If there are multiple peaks, return any one of them

... than adjacent elements,

Peak: If it is greater ... then it a peak

if(arr[i] > arr[i-1] && arr[i] > arr[i+1] arr[i] is a peak.



Peaks: 7, 3, 8

A = [3] (0)

Base Cases: if(n == 1) return arr[0]

0) arr[i] ?

1) if (arr[0] > arr[1]) arr[0] is a peak

2) if(arr[n-1] > arr[n-2]) arr[n-1] is a peak.

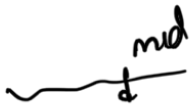
Brute Force:

Linear Traversal

T.C: O(n)

$T(n) = 2T(\frac{n}{2}) + O(1)$ → $O(n)$

Better Approach:



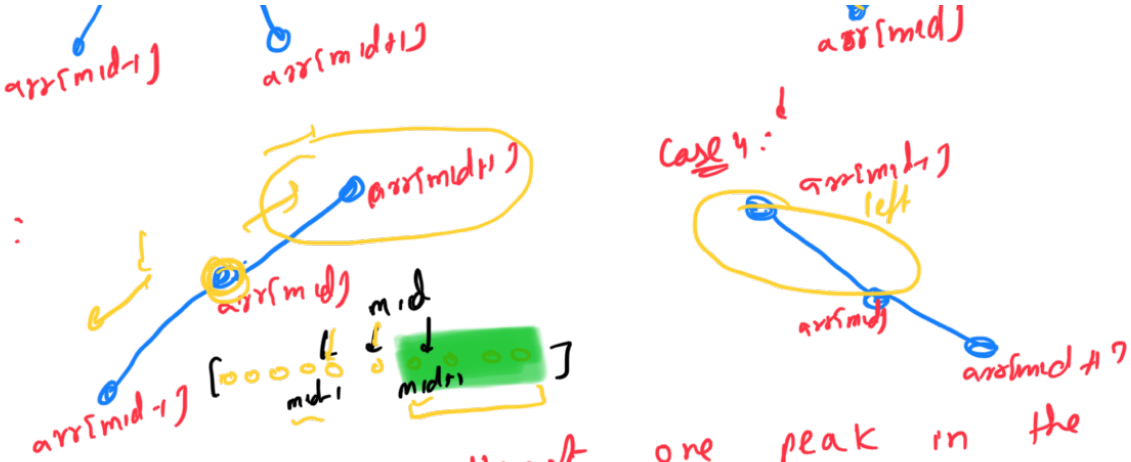
	0	1	2	3	4	5	6
A =	3	2	1	5	7	4	8

Search space: [0, 6] → Entire Array

arr[mid-1], arr[mid], arr[mid+1]

Case 1: arr[mid] is peak

Case 2:

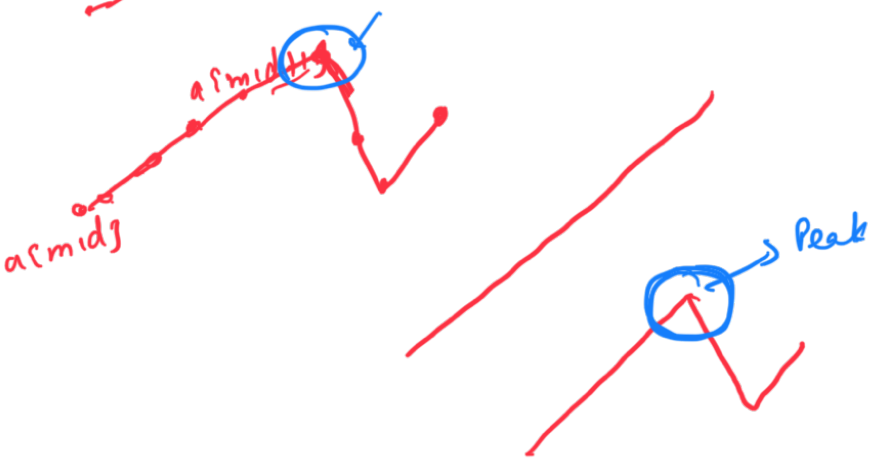


Claim: There exists atleast one peak in the side where it is increasing

Proof :

✓ Case 1:

Case 2:




```
int peakEle ( Arr, N ) {
    low = 0,  high = n-1
    // 3 Base Case
    while ( low ≤ high ) {
        mid = (high + low) / 2 ;
        if (arr[mid-1] < arr[mid] && arr[mid] > arr[mid+1] ;
            return arr[mid].
```

if arr[mid] < arr[mid+1]
low = mid+1;
}
else {
high = mid - 1;
}
}

T.C: O(logN)

mid = low + (high - low)/2

long long int low, high;

low = $10^{18} - 3$

high = $10^{18} - 1$

mid = (high + low)/2 = $2(\frac{10^{18}-2 + 10^{18}-1}{2})$

$= 2 \cdot \frac{10^{18}-1}{2}$

long long?

10^{18}

$mid = 10^{18} - 2$

low + (high - low)/2

Question: Max element in the array (Integers)

Efficient Approach: O(N)

Binary search:

low = -10^9

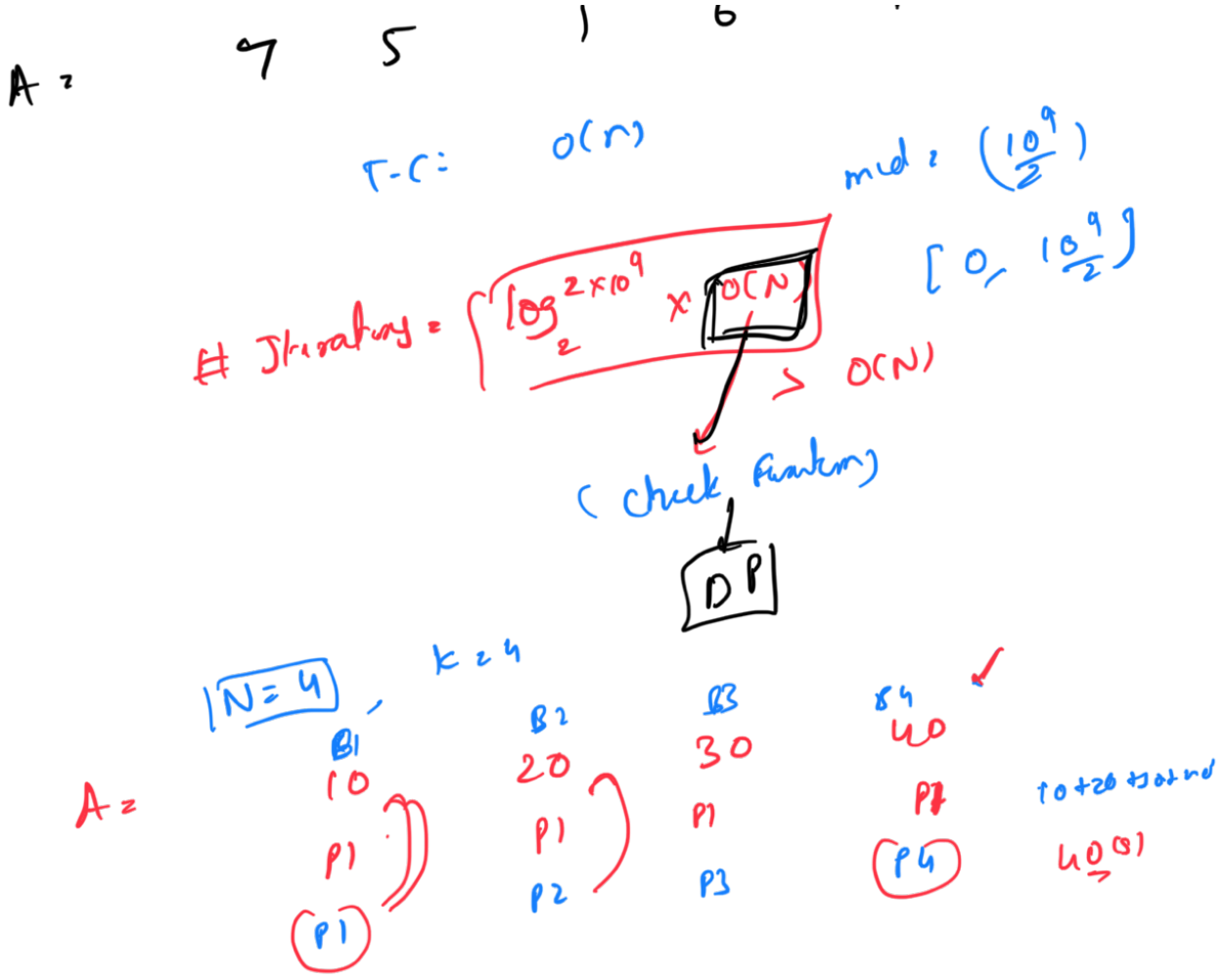
high = 10^9

Search space: $[-10^9, 10^9]$

low high

$[0, 10^9]$

mid = (high + low)/2: mid = 0



Question: Aggressive cows

```
bool check(int stalls[], int N, int k, int minDist){
      int last_placed = stalls[0];
      cows_placed = 1;

      for(int i=1;i<N;i++){
            if(stalls[i] - last_placed >= minDist){
                  last_placed = stalls[i];
                  cows_placed++;
            }
      }
      if(cows_placed >= k)
            return True;
      else
            return False;
}
```

```
int AggresiveCows(int stalls[], int n, int k){
      low = 1, high = stalls[n-1]-stalls[0];
      int ans = 0;
      while(low <= high){
            mid = (high + low) / 2;

            if(check(stalls, n, k, mid) == true){
                  ans = mid;
                  low = mid + 1;
            }
            else
                  high = mid - 1;
      }
return ans;
}
```

Question: Painter's Partition problem

```
bool check(int boards[], int n, int k, int maxTime){
      int painters = 1;
      int time = 0;

      for(int i=0;i<N;i++){
            time = time + boards[i];

            // Recently added board cross max time
            if(time > maxTime){
                  // Assign it to next painter
                  time = boards[i];
                  painters++;
            }
      }

      if(painters > k)
            return False;
      return True;
}
```

```
int paintersPartition(int boards[], int n, int k){
      lo = max(boards), high = sum(boards);
      int ans;

      while(low <= high){
            mid = (high + low)/2;
            if(check(boards, n, k, mid) == true){
                  ans = mid;
                  high = mid - 1;
            }
            else
                  low = mid + 1
      }
return ans;
}
```

Question : Max k such that every subarray of size k has sum > SUM

```
bool Feasible(k, SUM){
      curr_sum = 0;
      // Sum of first k elements
      for(int i = 0; i < k; i++)
            curr_sum += a[i];
      if(curr_sum > SUM)
                  return false;

      for(int i = k; i < n; i++){
            curr_sum = curr_sum - a[i-k];
            curr_sum = curr_sum + a[i];
            if(curr_sum > SUM)
                  return false;
      }
      return true;
}

int maxK(int arr[], int n, int SUM){
      low = 0, high = n, ans;
      while(low <= high){
            mid = (high + low)/2;
            if(Feasible(mid, SUM)){
                  ans = mid;
                  low = mid + 1;
            }
            else
                  high = mid - 1;
      }
      return ans;
}
```