SCALER

Binary Search - 3

TABLE OF CONTENTS

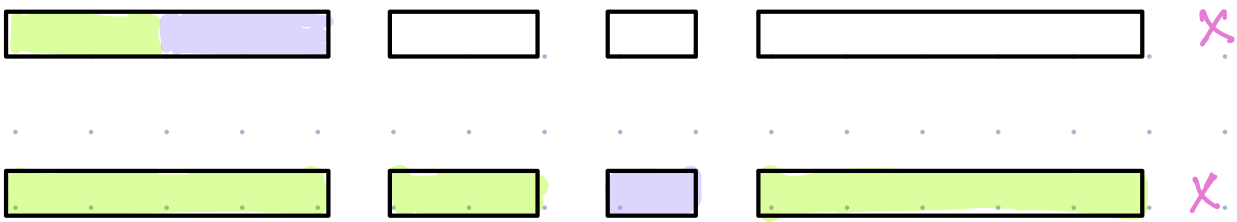
Painter's Partition Problem - 1

Given N boards with length of each board.

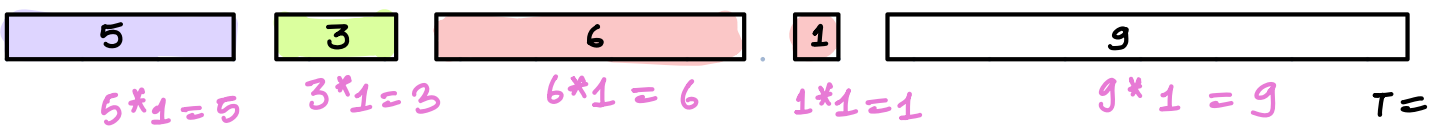
a) A painter takes T unit of time to paint 1 unit of length.

b) A board can only be painted by 1 painter.

c) A painter can only paint boards placed next to each other (i.e continuous segment).



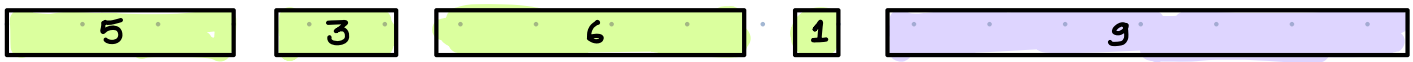
< *Question* > : Check if it is possible to paint all boards in “x” unit of time if P painters are available.



5*1=5 3*1=3 6*1=6 1*1=1 9*1=9

T=1
X=7
P=2

ans: False.



ans: true.

X = 15
T = 1
P = 2

ans : false

X = 10
T = 1
P = 2

Painters Reqd ≤ Available Painters ⟶ Possible.

< / > Code

```
bool PaintPossible (int arr[], int X, int T, int P ) {
    int painters = 1 , time_taken = 0 ;

    for (i = 0 ; i < arr.length ; i++) {
        if (arr[i] * T > X)
            return false ;

        time_taken += arr[i] * T ;
        if (time_taken > X) {      // Need to assign to new painter.
            painters ++ ;
            time_taken = arr[i] * T ;
            if (painters > P)
                return false ;
        }
    }
    return true ;
}
```

T.C : O(N)
S.C : O(1)

P = 2

24/2 = 12

Painter's Partition Problem - 2

Given N boards with length of each board

a) A painter takes T unit of time to paint 1 unit of length.

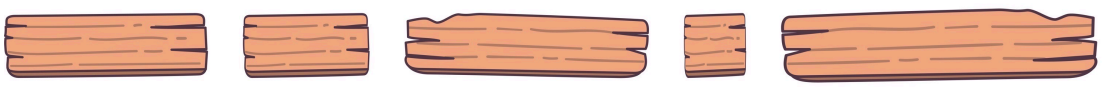
b) A board can only be painted by 1 painter.

c) A painter can only paint boards placed next to each other (i.e continuous segment).

< *Question* > : Find minimum time to paint all boards if P painters are available.

N = 5

T = 1



5 3 6 1 9

[P = 1] → 5*1 + 3*1 + 6*1 + 1*1 + 9*1 = 24 min

[P = 2] →

$P_1 = 1$, $P_2 = 4$ → 5 and 3 + 6 + 1 + 9 = 19 → Max (5, 19) = 19

$P_1 = 2$, $P_2 = 3$ → 5 + 3 = 8 and 6 + 1 + 9 = 16 → Max (8, 16) = 16

$P_1 = 3$ — 14
$P_2 = 2$ — 10
} Max(14, 10) = 14

$P_1 = 4$ — 15
$P_2 = 1$ — 9
} Max(15, 9) = 15

$P \propto 1/T$

Binary Search Approach

Target : Min time to paint boards

Search Space : [Max(arr) * T , sum(arr) * T]

(when each board is assigned to a different painter)

Max(arr) * T

(when all boards are painted by 1 painter)

sum(arr) * T

Represent Time

for time = mid
Is it possible to paint all boards by P painters ?

Yes

Store ans
&
Check for better ans on LHS
e = m - 1

No

painters > P
req

Dec Painters
→ Inc Time
Move to RHS
s = m + 1

</> Code

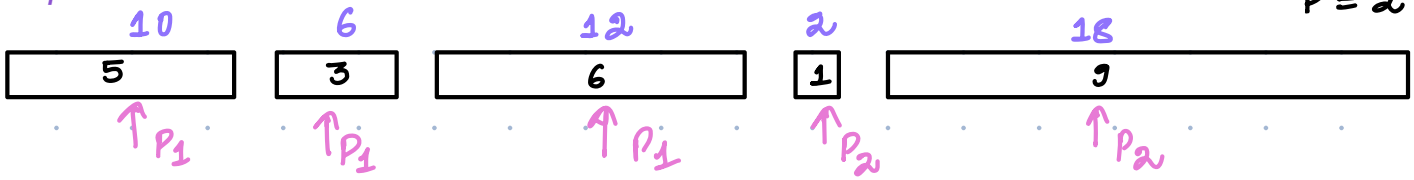
```
int MinTime(int[] arr, int P, int T) {

    int s = max(arr) * T;
    int e = sum(arr) * T;
    ans = 0;

    while (s <= e) {
        m = (s+e)/2;

        if(PaintPossible(arr, m, T, P) == true) {
            ans = m;
            e = m-1;
        }
        else {
            s = m+1;
        }
    }
    return ans;
}
```

T.C : O(log(searchSpace) * N)

S.C : O(1)

Example :

T = 2
P = 2



S	e	$m = \frac{S+e}{2}$	Min No of Painters at m	Is it possible to paint all boards by P=2 painters in time = m ?
9*2 = 18	24*2 = 48	33	2	Yes — ans = 33, e = m-1
18	32	25	3	No — S = m+1
26	32	29	2	Yes — ans = 29, e = m-1
26	28	27	3	No — S = m+1
28	28	28	2	Yes — ans = 28, e = m-1
28	27	→		Return ans (28)

Break till 10 : 35

Situation:

Imagine you are tasked with developing a system for evenly distributing the workload among a team of email response handlers in a customer service department. Each email is assigned a 'complexity score' which represents the estimated time and effort required to address it. The complexity scores are represented as an array, where each element corresponds to a single email.

Task

The goal is to divide the array into K contiguous blocks (where K is the number of email handlers), such that the maximum sum of the complexity scores in any block is minimized. This approach aims to ensure that no single email handler is overwhelmed with high-complexity emails while others have a lighter load.

Approach

This problem is same as painter's partition. This is just a real life example.

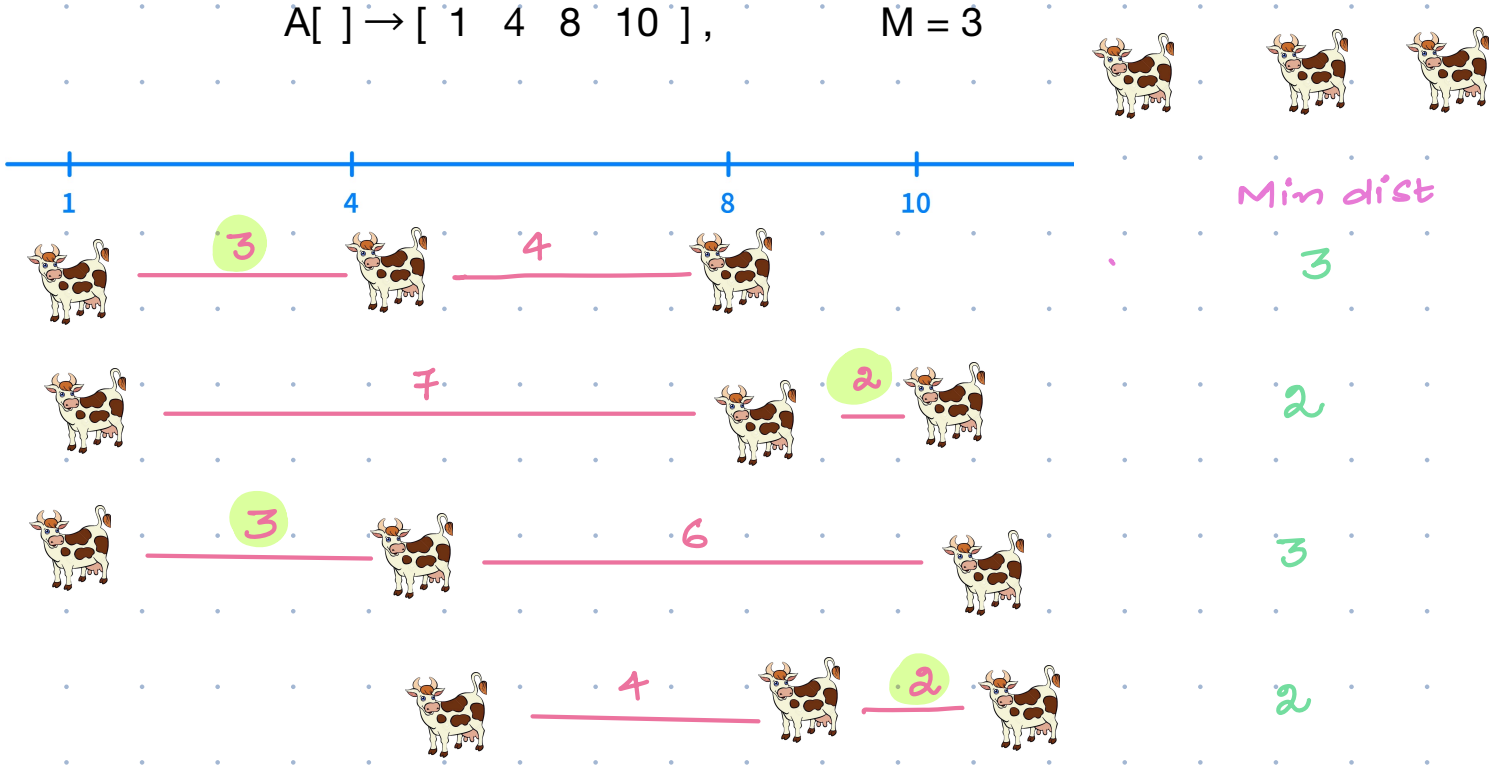
Aggressive Cows

< *Question* > : Farmer has built a bar with N stalls.Given a sorted array of integers A of size N where each element of the array represents the location of the stall and an integer M which represents the number of cows.

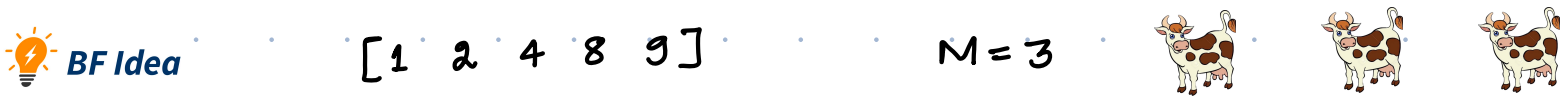
$2 \le M \le N$

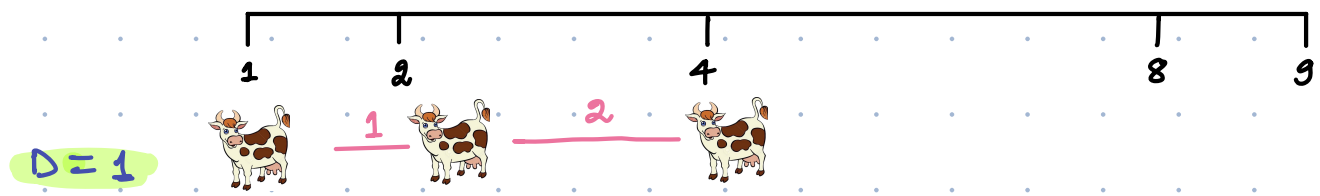
His cows don't like this barn layout and become aggressive towards each other once put into a stall. To prevent the cows from hurting each other, farmer wants to assign the cows to the stalls, such that the minimum distance between any two of them is as large as possible. What is the largest minimum distance?

A[ ] → [1 4 8 10], M = 3



Max Possible Min Dist = 3

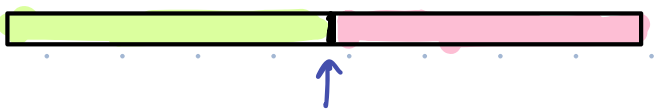




D = 5

✗ Not Possible

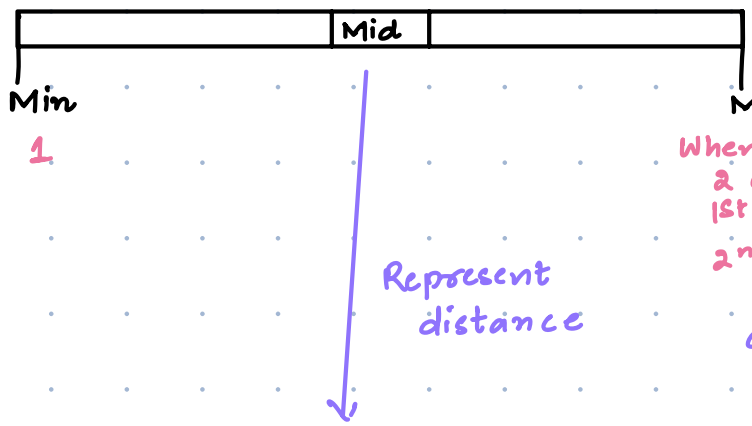
D = 6



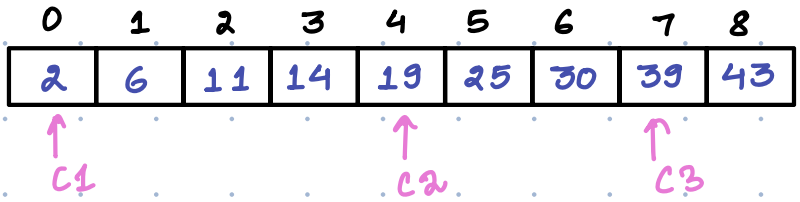
Idea -2

Target : Max Possible Min Distance

Search Space : [1 , arr[N-1] - arr[0]]



*Dry Run



M = 4

s	e	mid	Can we place M cows with distance ≥ mid?	
1	43-2 = 41	21	No	Move to LHS e = m-1
1	20	10	Yes	ans = 10 Move to RHS s = m+1
11	20	15	No	Move to LHS e = m-1
11	14	12	Yes	ans = 12 Move to RHS s = m+1
13	14	13	No	Move to LHS e = m-1
13	12			→ Return Ans

</> Code

```
int AggressiveCows(int[] arr, int M){        // M: no of cows.
    int n = arr.length(); ans = 0;
    int s = 1;
    int e = arr[n-1] - arr[0];

    while(s <= e){
        m = (s+e)/2;
        if(isPossible(arr, m, M) == true){
            ans = m;
            s = m + 1;
        }
        else{
            e = m - 1;
        }
    }
    return ans;
}

boolean isPossible(int[] arr, int dist, int M){        // M: No of cows.
    int cows = 1, last_pos = arr[0];

    for(i = 1; i < arr.length(); i++){
        if(arr[i] - last_pos >= dist){ // New cow can be placed here
            cows++;
            last_pos = arr[i];
            if(cows == M)
                return true;
        }
    }
    return false;
}
```

T.C : O(log(search space) * N)

S.C : O(1)