Binary Search Class 4

Binary Search On Answer (V. IMP)

① Book Allocation
→ Split array largest sum
→ Capacity to ship packages
→ Painters Partition

② Koko Eating Bananas
→ Smallest divisor threshold.
③ Aggressive cows
④ Wood cutting EKO

Practice Problems:
- Minimise max distance
- Pratap Spoj
- Minimum Bouquets
- Minimized max distance to any store
- Magnetic force between balls
- Minimum speed to arrive on time

① Book Allocation Problem

Given N books, Every i^{th} book has A_i pages. $\forall i \in [1, N]$
Allocate contiguous books to M number of students such that maximum number of pages allocated to student is minimum of those in all permutations and print this minimum value

Constraints
1. Each student have at least 1 book
2. We should allocate each book to exactly 1 student
3. Contiguous Allocation
4. Book cannot be divided, unbreakable item

Eg: | 20 | 10 | 30 | 40 | → books = 4, Stud = 2

Possible configs:
- {20}, {10, 30, 40} → max val = 10+30+40 = 80
- {20, 10} {30, 40} → max val = 30+40 = 70
- {20, 10, 30} {40} → max val = 20+10+30 = 60

min = 60

If k = 3

① {20} {10} {30, 40} → max Load = 70 ⇐ max(20, 10, 30+40)

② {20, 10} {30} {40} → max Load = 40 ⇐ max(20+10, 30, 40)

③ {20} {10, 30} {40} → max Load = 40 ⇐ max(20, 10+30, 40)

min = 40

Apply Binary Search On Pages. {Not on Books or Array}

Iteration

① low = max of array of pages (40) high = Sum of array of pages (100)

low = 40 , high = 100

mid = low + (high - low)/2 = 70

Now check if mid is a possible answer

if yes, store it and update high = mid - 1 to look for better answer

else update low = mid + 1 to look for an answer

to check use fn is Possible with following approach

20	stud = 1 ,	load = 0+20	✓	load <= 70
10	stud = 1 ,	load = 20+10	✓	load <= 70
30	stud = 1 ,	load = 30+30	✓	load <= 70
40	stud = ,	load = 60+40	X	load (100) > 70
		So stud ++		
		new Load = 40	✓	load(max(60, 40)) <= 70

Here stud = 2 (<= 2) & maxload <= 70 , So one potential answer is found → 70. (true)

∴ for low = 40 , high = 100 , mid = 70, is Possible(70) = true

So new high = mid + 1 ⇒ high = 69

To find even better answer , store Ans = 70

Check in [40, 69] next

Ans = 70

(2) low = 40 high = 69

mid = 54

Again check for isPossible(mid)

20	Stud=1, load = 0+20 ✓	load<=54
10	Stud = 1, load = 20+10 ✓	load <=54
30	Stud= 2, newLoad = 0+30 ✓	load > 54, newLoad <= 54
40	Stud = 3, newLoad = 0+40 ✓	load > 54, newload <=54

Here, Stud = 3 (>2), So this is not a possible answer (false)

So now check for possible answer on right half

So newLow = mid+1 = 55, Check in [55+69]

Ans = 70

(3) low = 55 high = 69

mid = 55 + (69-55)/2 = 62

Now check isPossible(62)

mid = 62

	Stud	load	load<=62	newLoad <=62
20	1	20	✓	—
10	1	30	✓	—
30	1	60	✓	—
40	2	40	X	✓

∴ Stud = 2 (<=2), this is better possible ans (true)

So update ans = 62, & update high = mid-1 = 61

To check for better answer.

(Ans = 62)

④ low = 55 high = 61

mid = 58

∵ Stud = 3 (>2)

isPossible(58) gives false, so now look in right half by

low = mid + 1

and look for answer.

(Ans = 62)

⑤ low = 59 high = 61

mid = 60

∵ Stud = 2 (<=2)

isPossible(60) gives true, So update answer as 60 & then on updating high = mid − 1, we get high = 59

Checking in [59, 59]

(Ans = 60)

⑥ low = 59 high = 59

mid = 59

∵ Stud = 3 (>2)

here isPossible(59) gives false, So this is not a possible answer.

So update low = mid + 1 = 60

But Now (low > high) So loop breaks.

and ans. variable has 60 as best possible answer.

∴ Answer is 60 pages. for | 20 | 10 | 30 | 40 | & k = 2.

∴ Pseudocode: low = max of array, high = sum of array., ans = high

```
while( low <= high)
    mid = low + (high - low)/2;
    if ( isPossible (mid , arr , stud)) {
        ans = mid; high = mid - 1;
    } else
        low = mid + 1;
}
```

∴ CODE

```
int findPages(int[] pages, int books, int students){
    int low = maxOfArray(pages, books);
    int high = sumOfArray(pages, books);
    int ans = high;

    while(low <= high){
        int mid = low + (high - low)/2;
        if(isPossible(pages, books, mid, students)){
            ans = mid;
            high = mid - 1;
        } else {
            low = mid + 1;
        }
    }
    return ans;

int maxOfArray(int[] pages, int books){
    int max = 0;
    for(int i=0; i< books; i+1)
        max = Math.max(max, pages[i]);
    return max;
}
boolean isPossible(int[] pages, int books, int maxAllowed, int totalStud){
    int currStud = 1, currPages = 0;
    for(int i=0; i<books; i+1){
        if(currPages + pages[i] <= maxAllowed){
            currPages += pages[i];
        } else {
            currStud++;
```

```
            currPages = pages[i];
        }
    }
    if (currStud > totalStud) return false;
    return true;
}
int sumOfArray(int pages[], int books){
    int sum = 0;
    for(int i=0; i<books; i++)
        sum += pages[i];
    return sum;
}
```

The answer cannot stop at a value that is not sum of any k number of books, where $k \in [1, N]$

$O(\max + \text{sum} + \text{binSearch} \times \text{isPossible}) = O(n+n+n\log_2(S-m))$

Time Complexity : $O(N \times \log_2(\text{sum} - \text{maxVal}))$

② Koko Eating Bananas.

find min. per hour eating speed to finish all piles of bananas

Eg:

3	6	7	11

$h = 8$

if speed = 11 bananas per hour,

hour 1 → 3 ✓ pile 1 done

hour 2 → 6 ✓ pile 2 done

hour 3 → 7 ✓ pile 3 done

hour 4 → 11 ✓ pile 3 done

So for $k = 11$, $h = 4$ (< 8) possible answer = 11

for $k=6$

hour	pile	eaten	left
1	1	3 → 3	0
2	2	6 → 6	0
3	3	6 } 7	1
4	3	1	0
5	4	6	6
6	4	5	0

here, for $k=6$, $h=6$ (≤ 8)

So this is a better answer

for $k=3$

hour	pile #	eaten	left
1	1	3	0
2	2	3	3
3	2	3	0
4	3	3	4
5	3	3	1
6	3	1	0
7	4	3	8
8	4	3	5
9	4	3	2
10	4	2	0

for $k=3$, $h=10$ (>8) so this is not a possible answer

☆ min possible eating speed (low) = 1

☆ max possible eating speed (high) = max(piles)

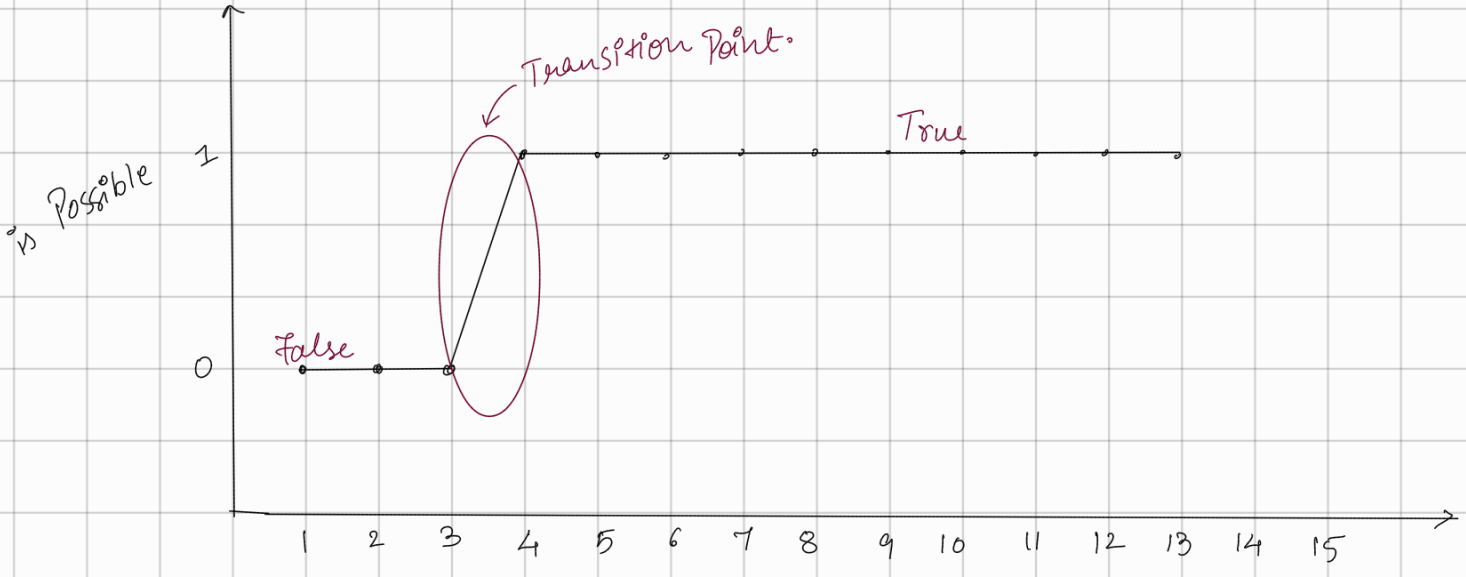
Here low = 1 & high = 11

Check linearly

eating speed	hours Needed	isPossible (hoursNeeded <= 8)
11	4	✓
10	5	✓
9	5	✓
8	5	✓
7	5	✓
6	6	✓
5	8	✓
4	8	✓
3	10	X
2	15	X
1	27	X

Observation: answers are monotonic function of eating speed

So, we can apply binary search on eating speed.



③ Aggressive cows.

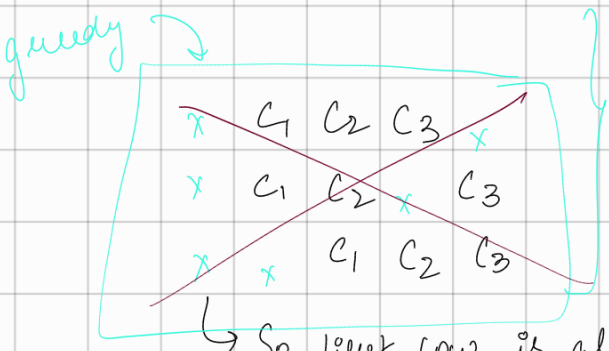
N — stalls. in long barn

Cow NO likey - likey, so maximize minimum distance b/w them

Eg: {1, 2, 4, 8, 9} ← stalls.

possible					min distance
C_1	C_2	C_3	x	x	1
C_1	C_2	x	C_3	x	1
C_1	C_2	x	x	C_3	1
C_1	x	C_2	C_3	x	3
C_1	x	C_2	x	C_3	3
C_1	x	x	C_2	C_3	1

→ max. min distance



These are removed from search space because using first spot will always give a better answer than leaving it.

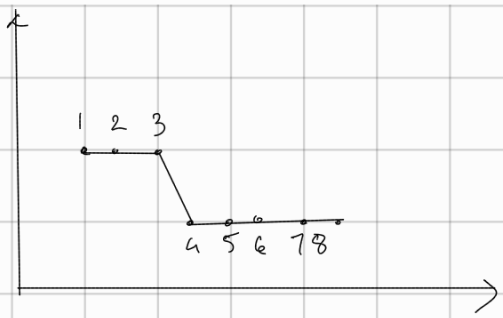
↳ So first cow is always in the first stall, other options will only give answers that cannot be minimum

So here we can apply binary search on Adjacent distance

low = min distance = 1

high = max distance = stall[n-1] − stall[0] // after sorting

Linearly distance	low 1	2	3	4	5	6	7	high 8
is Possible	yes	yes	yes	No	No	No	No	No
Placed At 1	C_1	C_1	C_1	C_1	C_1	C_1	C_1	C_1
2	C_2							
4	C_3	C_2	C_2					
8		C_3	C_3	C_2	C_2	C_2	C_2	
9				X	X	X	X	C_2


1 2 3
4 5 6 7 8