find sqrt of a number using BS:

if n

then like O(n) solution is

```
for (i = 0 -> n)
    if (i * i <= n) {
        ans = i * i
    }
    else:
        break;
}
```

either we have to return num which is $<= \sqrt{n}$

or we can

return floor of that num not >

we can do that in $\Theta(\log(n))$

if n = 28

l = 1 mid = 14.5 h = 28

if $14.5 \times 14.5 = 28 > n$

So

h = mid - 1

l=1 7 h=13

mid

$7 \times 7 = 49 \not\geq 28$

So

l=1 m=3 h=6

$3 * 3 = 9 < 28$

So

l = 4 m=5 h=6

$5 * 5 = 25 < 28$

we are updating ans with it

l = 6 ~~6~~ h=6

m

$36 > 28$

but loop breaks so

ans = 5

we are assigning m if $m * m < x$

Find $N^{th}$ root of M

N=3 M=27 $\sqrt[3]{27}$ = 3 = 3x3x3

N=4 M=69 $\sqrt[4]{69}$ = -1

→ 1x1x-1x-1=1

2x2x2x2=16

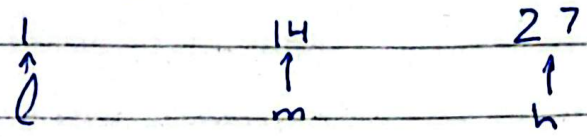
3x3x3x3=81 >

So in this case ans = -1

same O(n) solution like before

```
for(i=0 -> n)
    if (i*i*i*i==m)
        return i
    elif (i*i*i*i > m)
        -1
}
return -1
```

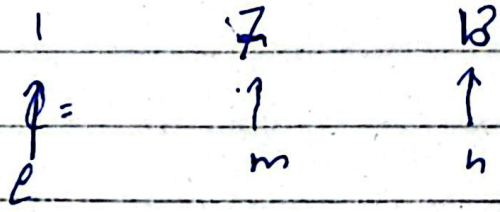
O(log(N) Solution.



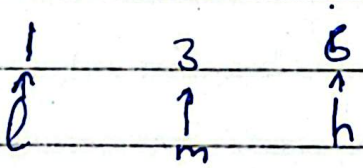

$14 \times 14 \times 14 > 27$

So

h = 13




$7 \times 7 \times 7 = ? > 27$




$3 \times 3 \times 3 = 27$

return ans else in another case h will become h<l

→ Also how much time the num is multiplied like n x n x n then i=3 So make a function for that.

# KoKo eating bananas.

piles = [3, 6, 7, 11]  h = 8

Return the min integer k such that KoKo can eat all bananas within h hours

h = 8 means she can only take 8 hrs max

k -> bananas / hr

For eg 2 bananas per hr

piles = [3, 6, 7, 11]

↑

2+1

2hrs to finish 3 bananas

we can have 1s but we will choose teil

piles = [3, 6, 7, 11]

↑ ↑ ↑ ↑

2hr 3hr 4hrs 6hr = 15 hrs > 8 so not possible.

So we will do 8 bananas per hr

piles = [3, 6, 7, 11]

1hr 2hr 3hrs 4hr = 10 still > 8

So 4 banana per hr

piles: [3, 6, 7, 11]

1 1 2 2 3 = 8

So minimum possible banana we can eat per hr = 4

if we see into the range like if we do 13 we will have 1+1+1+1 = 4hr

if we take 11 = 1+1+1+1 = 4hr but if we take 10 = 1+1+1+2 = 5 So the max pile is the max / hr we can have

So

1 ⟶ 11

ans will lie in this

function for total hrs.

```
func( arr , hours ):
    for (i=0 -> n)
        total_hrs += ceil (arr[i] // hourly)
        }
    return total_hrs;
}

for (i=H -> 1)
if req = ( function (arr, hrs)) <= hr
    return i
```

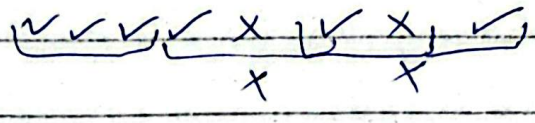
O(log(n)):

```
mid = ((l-1) + high(11))/2
ans = func (arr, mid)
if ans > hr :
    l = mid + 1
elif
    h = mid - 1
```

# Minimum No of days to make M bouquets.

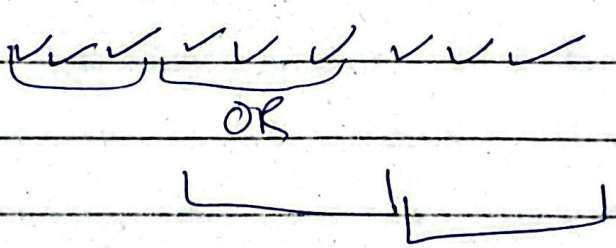
arr = [7, 7, 7, 7, 13, 11, 12, 7]  m = 2, k = 3

if we have 11

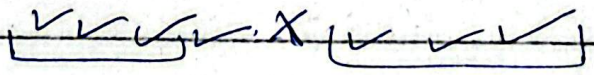


So we cannot

we can on 13



Also on 12



So we can do on 12, or 13 -

Ex 2:

bloomDay = [1, 10, 3, 10, 2]  m = 3, k = 2

Even on 10th day

we can only make 2 bouquets
So not possible

So -1

~~if (len(bd)) :~~

impossible case:

$M \times K > n$

$6 > 5$ so not possible.

return -1

possible case

arr = [7, 7, 7, 7, 13, 11, 12, 7]  m=2, k=3

what ever minimum we can do accordingly but we need minimum

like [7 . . . . . 13]

on 7

✓ ✓ ✓ ✓ x x x ✓

with 4/3 = 1 so set count = 1 again cuz 7 is not possible

On

counton 11 = ✓ ✓ ✓ ✓ x ✓ ✓ x ✓

counter 1 2 3 4 0 1 0 ✓

4%3 = 1  1%3 = 0

So

On 12 ✓ ✓ ✓ ✓ x ✓ ✓ ✓

counter = 1 2 3 4 1 2 3

4/3 = 1  3/3 = 1  1+2 = 2

at 12 we were able to perform 2 boquets so it is ans.

possible or not possible funct

```
func possible (arr, day, m, k)
    { counter = 0
      no_of_bouquet
      for (i = 0 ⟶ n-1)
          if (arr[i] <= day)
          { counter++ }
          else { counter = 0;
                 no of boquet += (counter // k)
                 counter = 0
          }
      noof boquet += (counter/k)
          if (no_of_boquet >= m)
              return T;
          else
          return F
    }
```

Now

binary search

l = 1 , max = max (arr)

Same problem approach

## LC : Smallest division given a threshold

Problem:

Capacity to ship packages within D days:

```
function to check (arr, value to check):
    summed = 0
    counter = 0
    for (i = 0 -> n)
        ~~if (summed~~
        ~~summed + arr[i]~~
    if (summed + arr[i] > value to check)
        count += 1
        summed = 0
    summed + arr[i]
return counter.
```

# find $K^{th}$ missing number.

arr = [5, 7, 10, 12]  K = 4

1, 2, 3, 4, 5, 6, 7

K=4 ↓

if K = 6

coz 5 is in arr

## Brute force solution:

arr = [2, 3, 4, 7, 11]  K = 5 6 7, 8, 9

↑ ↑ ↑ ↑ ↑  1 2 3 4 5

0 2 3 4 5

basically we are increasing value of K on finding every element.

```
for(i = n-1){
    if (arr[i] <= K){
        K++
    }
    else {
        break}
}
return K
```

O(log(N)) solution cuz
arr is sorted

what we can do is
we will find the range
in which answer is

arr = [2,3,4,7,11]  K=5

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11
1 ✓ ✓ ✓ 2 3 ✓ 4 5 6 ✓

for BS we will take entire range
and for that like in this
problem it is [7, 11]

Step1:
figure out 2 nearby indexes.
Step2:
~~if 11 is there~~
missing = arr[i] - (i + 1)
for every index

like

arr = [2, 3, 4, 7, 11]

Missing = [1, 5, 6, 8, 9, 10]

at i = 0 we expect 1

i = 1 we expect 2

But

Index = 0, 1, 2, 3, 4

2 3 4 7 11

at arr[2]

= 4 - (2+1)

= 4 - 3

= 1

at arr[4]

= 11 - (4+1)

= 11 - (5)

= 6

So

if missing < K

left = mid + 1

else

righ = m - 1

So if left stops at 8

like

arr[3] = 6 + 5 = 9

which is missing

---

## Aggressive Cows:

arr = [0, 3, 4, 7, 10, 9]  cows = 4

Aggressive cows => (mindist between cows) is max

Now prolems are

min (max)

or

max (min)

our task is to place cows in such a way that minimum of distance between any cows is maximum

Sorted arr

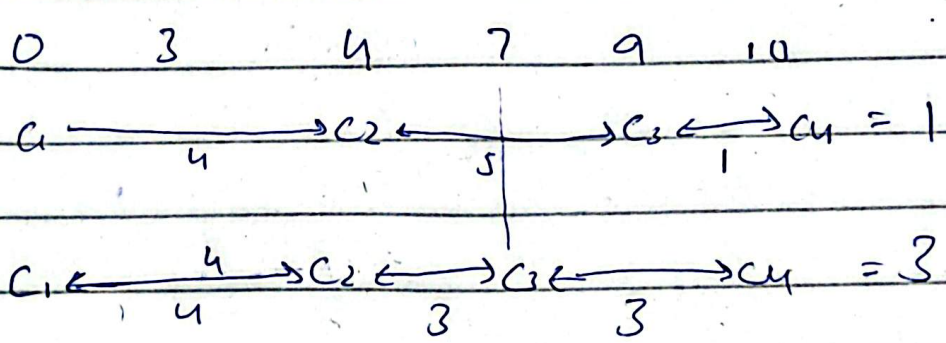
0 3 4 7 9 10

$c_1 \xleftarrow{3} c_2 \xleftarrow{1} c_3 \xleftarrow{3} c_4$ = 1

4 4

the min is 1

if we do




So we have to replace them in such a way that dist is max which is min

By linear Search:

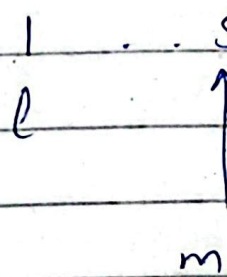
```
for (i — 1 , i <= max - min) {
    if (can we place (arr, i)):
        continue
    else:
        return i-1
}

max dist = (1, arr[0] - arr[0])
                (max-min)

function (can we place (arr, dist, cows)
    cut cows = 1, ~~coordinate~~ last = arr[0] → first cow.

    for (i-1— n)
    {
    if (arr[i] - last) >= dist)
    { cut cows += 1
        last = arr[i] }
if cut cows >= cows :
        return True.
else: return False
```

Now Changing it to Binary search:




~~if 5 possible:~~
~~l = m+1~~

if 5 possible:
h = m-1
else:
l = m+1

Same is painters portion problem.

Allocate books:

Same type just check we can allocate student according to given pages which is basically mid.

```
function check ( pages)
    curr-sum = 0
    students=1
    for (i=0 -> n):
        if curr_sum + ~~pages~~ arr[i] <=pages:
            curr = curr_sum + arr[i]
        else:
            student += 1
            curr_sum = arr[i]

if students > k return False
    else True
```

In this case

l = max (arr)

h = sum (arr).

Minimise Maximise distance to Gas station.

arr = [1, 2, 3, 4, 5]  K = 4

coordinates of gas station

our task is to place these no of new gas stations.

[1, 2, 3, 4, 5, 6, 7, 8, 9]

max dist = 1

if [1, 1.25, 1.5, 1.75, 2, 3, 4, 4.5, 5]

max = 1

if [1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5]

max = 0.5

we cannot do better than this.

# Median of 2 sorted arrays:

$arr_1 = [1, 3, 4, 7, 10, 12]$ $arr_2 = [2, 3, 6, 15]$

1, 2, 3, 3, 4, 6, 7, 10, 12, 15 $n = 10$

median is middle element but if we devide this array 5, 5 the

$$\frac{4+6}{2} = 5$$

for example like this

$arr_1 = [2, 3, 4]$ $arr_2 = [1, 3]$

1, 2, 3, 3, 4 $n = 5$

median

Brute force is making sorted array of these two. then doing median formula.

if some how we get 4 and 6 or 3 in case of odd

if we see in arr

arr = [1, 2, 3, 3, 4, 6, 7, 10, 12, 15]

n = 10 , idx1 = 4, idx2 = 5

if even length the

$\frac{n}{2}$, $\frac{n}{2} - 1$ => then $\frac{odd}{2}$

in case odd

arr1 = [1, 3, 4] arr = [2, 3]

arr3 = [1, 2, 3, 3, 4] n = 5

$\frac{5}{2} = 2.5 - 2$

in case of odd length we just discard $\frac{n}{2} - 1$ only $\frac{n}{2}$.

n1 = len(A)

n2 = len(B)

total = n1 + n2

(total + 1) // 2

Suppose two arrays

2 4

x → $x_1$ $x_2$ | $x_3$ $x_4$ $x_5$ $x_6$

y → $y_1$ $y_2$ $y_3$ $y_4$ $y_5$ | $y_6$ $y_7$ $y_8$

5 3

7 7

as we know

$x_2 <= x_3$

$y_5 <= y_6$

So if

$x_2 \leq y_6$

$y_5 \leq x_3$

once we find these elements then

if even len(avg(max($x_2$, $y_5$), min($x_3$, $y_6$)))

if odd len(max($x_2$, $y_5$))

TC = O(log(min(n, y)))

Example:

X = [1, 3, 8, 9, 15]

Y = [7, 11, 18, 19, 21, 25]

= 1, 3, 7, 8, 9, 11, 15, 18, 19, 21, 25

st = 0

end = 4

partion x =

partion y =

Code

partion x + partion y
= (x + y + 1) / 2
↑ for odd even.

found:

maxLeft X ≤ minR Y

max left Y ≤ minR x

elseif

maxLeft > minR y

move towards left in x

else

move towards right in x

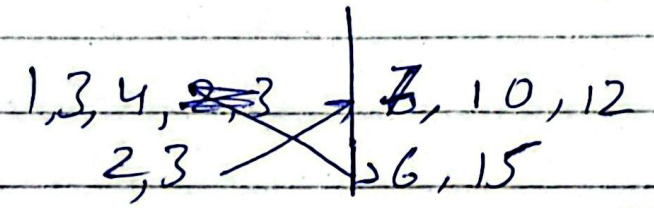
arr1 = 1, 3, 4, 7, 10, 12

arr2 = 2, 3, 6, 15

arr = 1, 2, 3, 3, 4 | 6, 7, 10, 12, 15

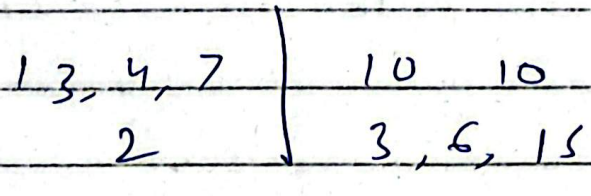
5 5

If we pick exactly 3 elements from arr1 and other from arr2




4 < 6 , 3 < 7

This is a valid symmetry So we can also do 4 from arr1




7 > 3 ✗

2 < 10 ✓

This is not

How determine if it is a valid symmetry.

~~greatest of left~~

$$\frac{\max(l_1, l_2) + \min(r_1, r_2)}{2}$$

| | |
|---|---|
| 1, 3, 4 $l_1$ | 7 $r_1$, 10, 12 |
| 2, 3 $l_2$ | 6 $r_2$, 15 |

we ~~can~~

0, 1, 2, 3, 4, 5 ...
x x x ✓ x x

h = len of (min no of arr)
l = 0

[0 — 4 — 6]

↓

how to determine
4 ~~or~~ beyond 4 is
not possible.

if $l1 > r1$
h = mid − 1

else if $l2 > r1$
l = mid + 1

find the row with maximum 1s'

0 0 1 1 1

0 0 0 0 0

0 1 1 1 1

0 0 0 0 0

0 1 1 1 1

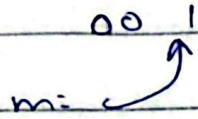
maxcount = -1 , idx = -1

```
for (i=0 - n-1)    count = 0
    for (j=0 - size[0]-1) {
        count += arr[i][j]
    }
    if (count > maxcount) {
        idx = i
        maxcount = count
    }
```

~~Ex~~ O(n m)

as entire row is sorted and interviewer ask for the optimized for version then use BS

just treat every row as individual arr.




So we can use first occurance
or upper bound of 0
or lower bound of 1

we can optimize the

```
for (i=0->n){
....
```

int idx of first 1 (arr[row], 1)

if idx of first 1 is < last:
return

just do cols-len - returns lower bound value

for max no.

# Search in a 2D Matrix (sorted)

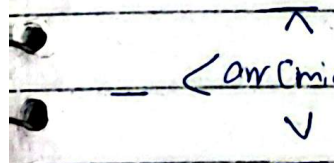
code in files:

---

find a peak element II 2D (matrix):

=> Simple solution could be find largest element in the matrix

=> O(log(N)) solution




for every mid_idx we first find the max_element idx of that row

max rowF0

for r in rows

if mat[r][mid_col] > mat[max_row]:

max_row = r

then if left_val > than that move mid.

right = mid - 1

move left col ←

else

left = mid_col + 1