SQUARE ROOT OF NUMBER

Binary search way

BRUTE

pick element from 1 to N linearly.

```
for(i = 1 ⟶ n){
    if(i*i <= n){
        ans = i;
    }
    else break;
}
```

1 2 3 4 5 6 ~~7~~ 8 9 10

✓ ✓ ✓ ✓ ✓ ✓ x x x x

any no >6 not an answer

when (n=36) → 6×6

so whenever we get situation in which we can eliminate certain possibilities than we can think of binary search

n = 28

low = (1)

high (28)

mid = (14) → 14×14 > 28 ∴ eliminate right half

∴ new high = mid − 1

low = (1) high = (13) mid = (7) → 7×7 > 28 ∴ eliminate

low = (1) high = (6) mid = (3) → (3)×3 < 28 eliminate left half

low = (4) high = (6) mid = (5) → 5×5 < 28

start = (5) eliminate left

here

3 can be answer

if n is not perfect

square as 28 is

to look for mid if

any other no satisfy

this (2) condition

low = (6) high = (6) mid (6)

6×6 > 28 ∴ (5) is answer

floor value of √28

right eliminate

low = (6) high = (5) (end)

not possible

place

always

where answers

possible always.

∴ high will have

answer

always.

```
f(n){
  low = 1 ; high = n ;
  ans = 1 ;
  while ( low <= high ){
      mid = (low + high)/2 ;
      if (mid x mid <= n ){
            ans = mid ;
            low = mid + 1 ;
      }
      else high = mid - 1 ;
  }
return (high) or (answer) ;
```

ans

$\log_2(n)$

to identify pattern → range would be known * and if max or min * we have to found then * think binary search *

there max was $\sqrt{5 \times 5} < 28$

(max value of in integer) $\sqrt{28}$

FIND N^{th} ROOT OF M

$n=3 \quad M=27 \quad \sqrt[3]{27} = 3$

return (-1) is not an integer.

same as previous but not taking floor

```
f(n, m) {
    low = 1 , high = m
    while (low <= high) {
        mid = (low + high)/2;
        value = f(mid, n);
        if (value == m) return mid;
        else if (value < m) low = mid + 1;
        else high = mid - 1;
    }
return -1;
}
```

$O(\log_2(m))$

$O(\log_2 n)$ → if in built power function is used

$O(n)$ → if for loop is used.

time complexity $O(\log_2(m) \times \log_2(n))$

$O(\log_2(m+n))$

edge case $\boxed{n=10}$ $\boxed{m = 10^9}$

now → $mid \approx \left(\frac{10^9}{2}\right)$

$func\left(\frac{10^9}{2}, 10\right) \longrightarrow$ overflow

therefore
do not actually calculate $(\frac{10}{2}, 10)$ 10 times

just store result and whenever is crosses "m"
stop as it will be sufficient for the if condition to work

if m = 16
and ~~4×4=16 stop~~
5×5 = 25 → stop do not multiply further
∴ 25 > 16

f(num, pow,) { mid

long ans = 1;
for (i = 1 ⟶ pow) {
ans = ans × mid;
if (ans > num) return 2;
}
if (ans == num) return 1;
return 0;
}

∴ rewrite previous condition
value = f(mid, n);
ans compare with 1, 0 and 2 :

KOKO EATING BANANAS

piles[] = {3, 6, 7, 11} hours = 8

find min integer "k" such that Koko can eat all banana with hours "hours" given.

if koko decided to eat 2 banana/hour

then {3, 6, 7, 11}

$\frac{3}{2} = \lceil 1.5 \rceil = (2)$ hours to ~~compl~~ eat 3 banana (we always take ceil value)

$\frac{6}{2} = 3$ hour

$\frac{7}{2} = 3.5 \rightarrow 4$ hour

$\frac{11}{2} = 5.5 \rightarrow 6$ hour

$\rightarrow 2 + 3 + 4 + 6 = (15)$ which is > 8

∴ not a solution

if 5 banana per hour

$\frac{3}{6} = (1)$ hour

$\frac{6}{6} = 2$ hour

$\frac{7}{6} = 2$ hour

$\frac{11}{6} = 3$ hr

8 hour

∴ 5 banana/hour is ~~answer~~

2 but if 4 banana per hour

$\frac{3}{4} = 1$

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

$\frac{7}{4} = 2$

$\frac{11}{4} = 3$

(8)

ans 4 is (4) × 5

4 is answer

as 3 banana per hour will not satisfy condition

now if exact hours are not matching we need to (taking less than 8 hours also an answer but we need to find closest to 8)

OPTIMAL

Now [3, 6, 7, 11]

maximum rate can be (11) as after that same total hours

[3 6 7 11]

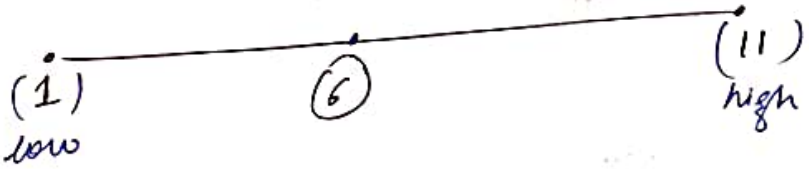
(11)

1+1+1+1 = 4 hours

1+1+11+ = 4 hrs (12)

and minimum consumption = (1) rate

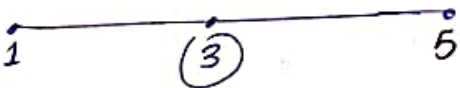
answer will be in this range



6 → 2 8 (1+1+2+2) = 6 < 8 possible but does much smaller rate exist

high = mid − 1;

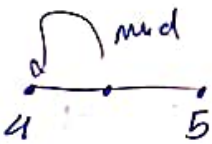
ans = 6



3 (1+2+3+4) = 10 > 8

∴ we need to increase rate

∴ low = mid + 1;



4 (1+2+2+3) = (8) ans = 4 ✗ 4

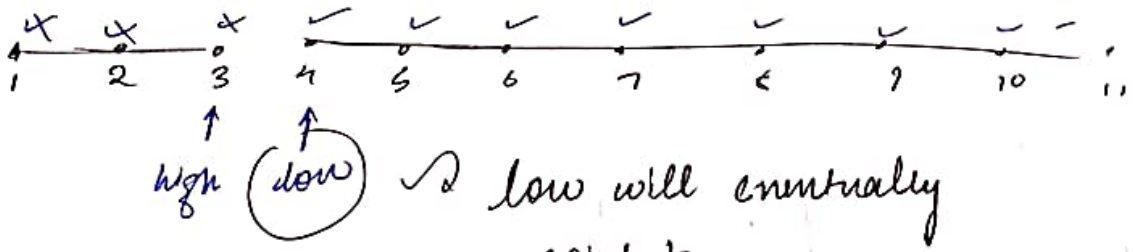
5 (1+2+2+3) = 8 but 5 > 4

4 ——— 4

low = high = mid 4 satisfy

3 4 ↪ low

↳ high break



low will eventually point to smallest (k)

∴ no extra element needed.

O(n)

```
fun(arr, houses) {
    low = 1; high = findMax(arr);
    while (low <= high) {
        int mid = (low + high)/2;
        int totalH = calculateTotalHours(arr, mid);
        if (totalH <= hour) {
            high = mid - 1;
        }
        else {
            low = mid + 1;
        }
    }
    return low;
}
```

$\log_2(max)$

'=' as better answer can exist as in case of 4 and 5

O(n)

time complexity → $O(n + n \times \log_2(max))$

typecast to double before dividing
// possible bug [arr[i] (double) / mid (double)]

AGGRESIVE COWS

you are given an array 'arr' of size 'n' which denotes the position of stalls. You are also given an integer 'k' which denotes the no of aggressive cows. You are given the task of assigning stalls to 'k' cows such that the minimum distance between any two of them is the maximum distance.

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

sort array

when dis = (1) 0 3 4 7 10 9

$c_1 \longleftrightarrow c_2 \longleftrightarrow c_3 \longleftrightarrow c_4$ ∴ min dis 1 possible

we have maintain min distance 1 between cows

now when dis = 2

[0 3 4 7 9 10]

$c_1 \longleftrightarrow c_2 \longleftrightarrow c_3 \longleftrightarrow c_4$

can't place here as dis = 1 ∴ on 7 yes as 7-3 ≥ 2

when dis = 3

[0 3 4 7 9 10] ✓

$c_1 \longleftrightarrow c_2 \longleftrightarrow c_3 \longleftrightarrow c_4$

when dis = 4 [0 3 4 7 9 10]

c_1 —— c_2 —— c_3 —— c_4 not possible

∴ minimum maximum distance is

(3)

BRUTE

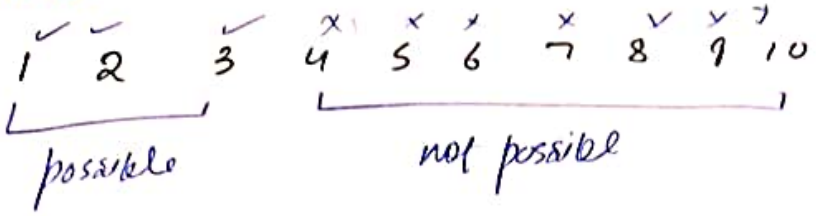
```
sort(arr.);
for( i=1 ; i <= (max-min) ; i++ ) {        // when two cows max possible min dist.
    if(canPlace(arr, i, cows) == true) continue;
    else return i-1;
}

canPlace( arr, i, cows ) {                 // dist
    countCows = 1 ; last = arr[0];         // placing first cow at first index
    for( i = 1 ----> =(n-1) {
        if( arr[i] - last ≥ dist ) {
            last = arr[i];
            countCows ++;
        }                                  // can be here
    }
    if (cows' ≥ cows) return true
    return false
}
```

TC → O(max - min) x O(n) + $n \log n$ (sorting)

SC → O(1)

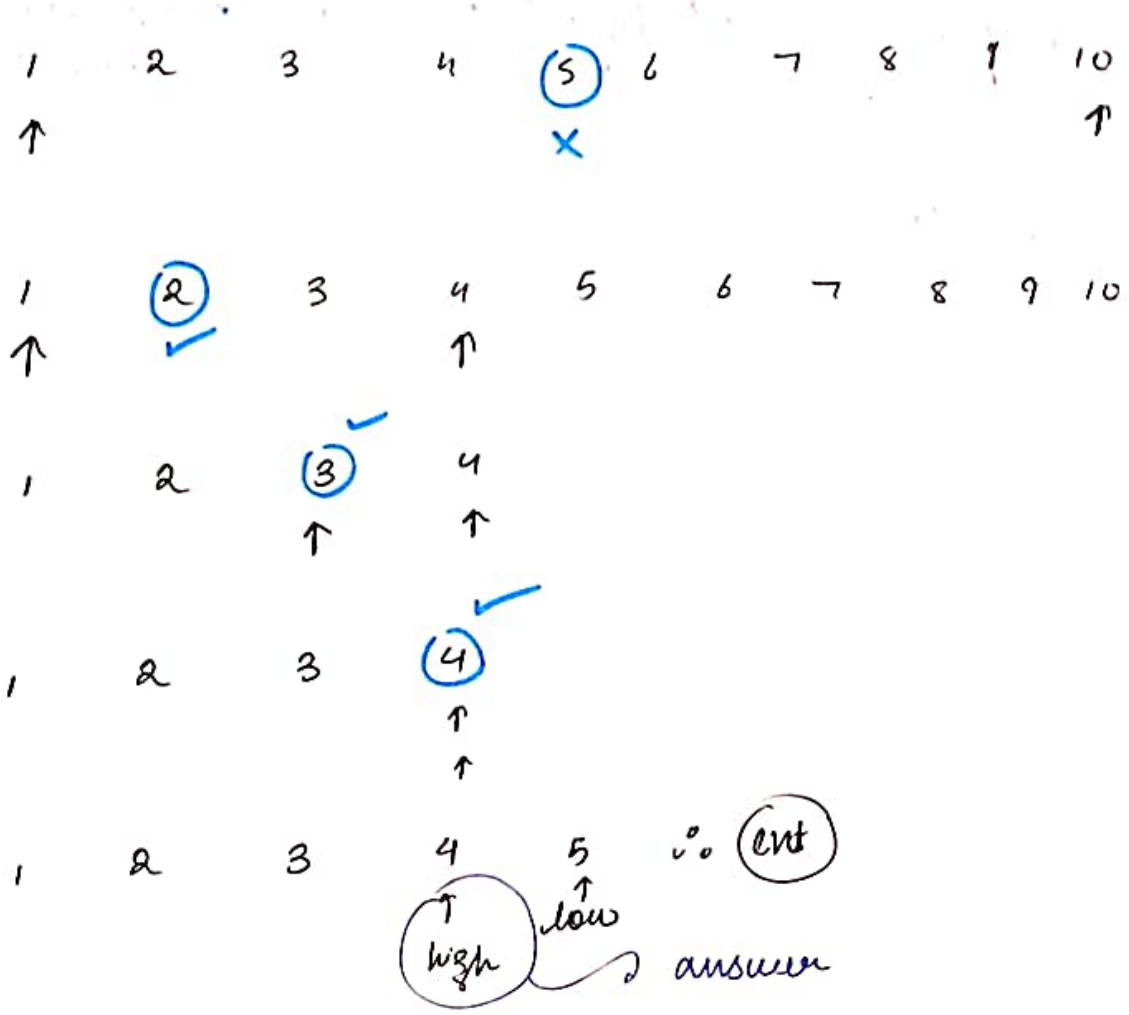
OPTIMAL



sorted

possible not possible

∴ Binary search




```
for ( arr, rows) {
    low = 0, wgh = arr[n-1] - a[0];
    sort (arr);  ~ O(n log n)
    while ( low < wgh ) {
        mid = (low + wgh) / 2;
```

```
if( canWeplace (arr, mid, cows ) == i ){
    low = mid + 1;
}
else high = mid - 1;
}
return high;
}
```

time complexity → $O(n \log n) + O[\log_2(max-min) \times O(n)]$

space → $O(1)$

ALLOCATE BOOKS

arr = [25, 46, 28, 49, 24]

Student = 4

Given an array arr of integer where 'a[i]' represents the number of pages in the i-th book. There are a 'm' number of student and the task is to allocate all the books to the student

* Each student gets at least one book
* Each book should be allocated to only one student
* Book allocation should be in contiguous manner
* if allocation not possible return -1

{25, 46, 28, 49, 24} m = 4

{ | 25 46 | 28 | 49 | 24

{ this is maximum here
and we cannot get another maximum in any arrangement that will be smaller than this maximum
∴ it is minimum maximum.

arr

OPTMAL

arr = [25, 46, 28, 49, 24]

student = 4

if 49 (max of array)

then ① → 25

② → 46

③ → 28

④ → 49

⑤ → 24

5 > 4

but we need to allocate to only 4 ∴

we ↑ 49 → 50 → 51

if 71

then ① — 25, ① → 25, 46

② — 46 ② → 28

③ → 28 ③ — 49, 24

④ ④ — 24

4 = 4 answer

(we can do binary search from min)

when student = ①

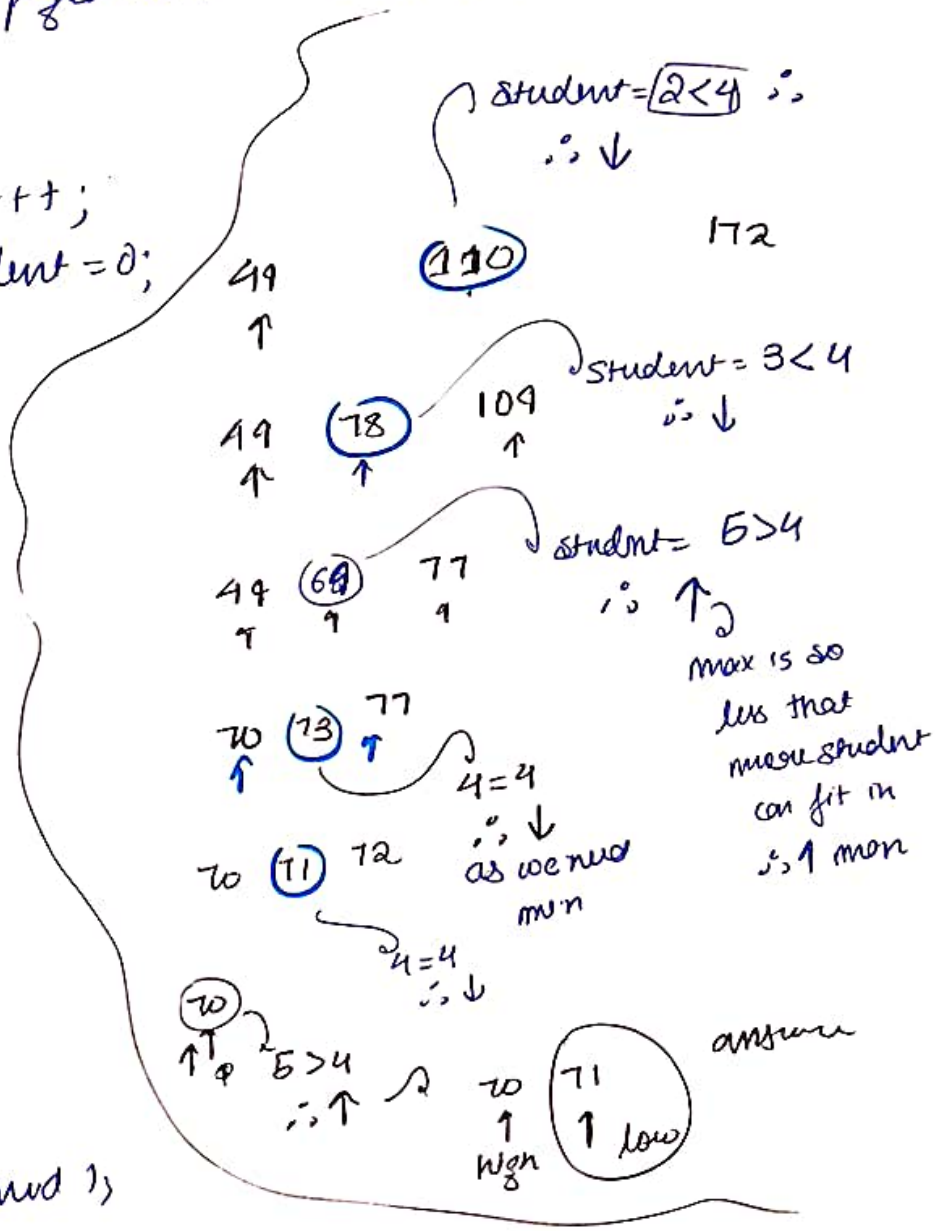
* then the max allocation would be sum of array

No of S

```
Students (arr, pages) {
    student = 1; pagesStudent = 0;
    for (i = 0 ⟶ n-1) {
        if (pagesStudent + arr[i] ≤ pages) {
            pagesStudent += arr[i];
        }
        else {
            student++;
            pagestudent = 0;
        }
    }
    return students
}

fun (
    low = max(arr);
    high = sum(arr);

while (low <= hg) {
    mid = (low + hgh) / 2;
no Students = Students (arr, mid);
    if (noStudents > m) low = mid + 1;
    else  high = mid - 1;
}
return low;
}
```



MINIMISE MAX DISTANCE BETWEEN GAS STATION

Problem: you are given a sorted array 'arr' of length 'n', which contains positive integer position of 'n' gas station you are also given an integer K. You have to place 'K' new gas station on the X-axis. You can place them anywhere on the non negative side of the X-axis even on non integer position. Let 'dist' be maximum value of the distance between adjacent gas stations after adding K new gas stations. find minimum value of 'dist'.

can be {13, 17, 23, 40} but sorted

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

gas station coordinate

place K new gas station so that you can minimize max, two gas station distance bet.

→ 1 2 3 4 5 6 7 8 9 — mom = 1

→ { 1 1.25 1.5 1.75 2 3 4 4.5 5 }

0.25 0.25 0.25 0.25 1 1 0.5 0.5

mom

this is the min mom.

→ 1 — 1.5 — 2 — 2.5 — 3 — 3.5 — 4 — 4.5 — 5

0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5

mom = 0.5

→ KTH MISSING NUMBER ←

arr = {2, 3, 4, 7, 11} K = 5

Note: Answers within 10^{-6} of the actual answer will be accepted.

they (setter) do not want so that your code do not run for a long time on the server while calculating exact answer beyond 6 decimal places as it will result in "time limit exceeded" therefore it is said that do not calculate beyond 6 decimal places as only that will be compared.

arr = {1, 7} K = 2

1 ——.——.——.7

$\frac{6}{3} = 2$

1 —— 3 —— 5 —— 7

placing gas station between them to minimize all * (approach)

arr = { 1 (12) 13 (4) 17 (6) 23 } K = 5

① → placing at max distance (12)

1 — 7 — 13 17 23

②nd 1 (6) 7 (6) 13 (4) 17 (6) 23

arr = {1, 2 3 4 5} K = 4

0 ——————→

ans can range from 0 ⟶ when all the gas station are placed on same coordinate (yes we can do this)

to (1) → which is current max distance between stations

Now linearly how this would work

0 0.1 0.2 0.3 - - - -

to maintain a min max ds of 0 we need infinite 'K' station which is ∞ > 4 ∴ not possible

now

0.1

to maintain 0.1

{1 1.1 1.2 1.3 1.4 1 2 3 — 4 5}

after using 4 station still we need more station as max min distance is still (1)

∴ ↑ distance

0.3

{1 1.3 1.6 1.9 2 2.3 .... 3 4 5}

here also need of gas station > K

~~Time of Zero~~

∞ when 0.5

⊙ { 1 1.5 2 2.5 3 3.5 4 4.5 5 }

for 0.5 K = needed station count

this is answer but to find minimum we have to go left to minimize

0.5 ←

no of gas > K low = mid

else high = mid

{ 1 2 3 4 5 }

0.4

$\frac{1}{0.4} = 2.5$M → 2 when in decimal

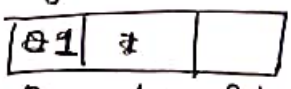
2 is no of station that we can place

but when $\frac{1}{0.5} = 2$ → ——o——o only 2 − 1 = 1 can be placed

∴ if exactly divisibly then decrement.

→ 1 (7) 13 17 23

mom = 6



{1 13 17 23}

{1 13 17 23}

keeping track of how many placed between them

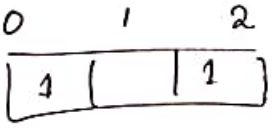
2nd place

already placed ∴ dis $\frac{12}{2} = 6$

placing in between them

4 (6)

∴



{1 13 17 23}

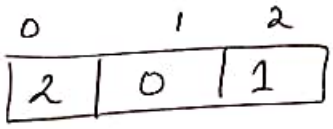
3rd

$\frac{12}{2} = 6$ 4 $\frac{6}{2} = 3$

Now mom = 6 ∴ to reduce we will place it there

all stations equally = $\frac{13-1}{3} = \frac{12}{3}$ = 4 mon new distance

→ 2 station to place ∴ 3 compartment



{1 13 17 23}

4th

$\frac{12}{3} = 4$ 4 $\frac{6}{2} = 3$

Now mom = 4 ∴ place between (13, 17)

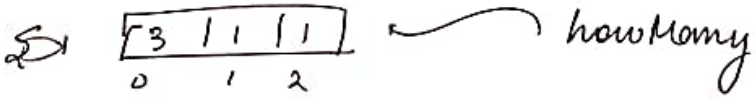
{1 13 17 23} b

now 5th $\frac{12}{3}=4$ $\frac{4}{2}=2$ $\frac{6}{2}=3$

∴ mom = 4

∴ place between them total $13-1=12$ $= \frac{12}{2+1+1} = \frac{12}{4} = 3$

already, new station, for compartment



{1 13 17 23}

$\frac{12}{4}=3$ $\frac{4}{2}=2$ $\frac{6}{2}=3$

∴ mom min = 3

```
for (gas = 1 --> =K ){
        momValue = -1 , manIdx = -1;
        for ( i=0 --> =(n-1){
           dif = arr[i+1] - arr[i];
        sectionlength= dif/(howMany[i]+1) ;
        if (mom < section length){
                        mom = sectionlength;
                        momIdx = i;
              }
        }
      howMany [momIdx ]++;
}
```

```
minimizeMaxDistance(arr, K){
    n = arr.length;
    double low = 0;
    double high = 0;
    for(i=0 ---> i<=n-1){          // defining range
        high = Math.max(high, (double)(arr[i+1] - arr[i]));
    }
    double diff = 1e-6;            // 10^-6
    while(high - low > diff){
        double mid = (low + high) / 2.0;
        int cnt = numberOfGasStationsRequired(mid, arr);
        if(cnt > K){
            low = mid;             // we cannot do mid+1 as there will be lot of possible answer that will get missed as answer is in double
        }
        else{
            high = mid;
        }
    }
return high;
}
```

```
numberofcrossturnonRq(dist, arr){
    n = arr.length;
    cnt = 0;
    for(j=1 ⟶ n){
    numberinbetween = (int)(arr[i]-arr[i-1]) / dist);

if((arr[i] -arr[i-1]) == (dist * numberinbetween)){
            numberinbetween--;
    cnt += numberinbetween;
}
return cnt;
}
```

// means that dist exactly divides the gap. i.e. -1 less station

KTH MISSING POSITIVE NUMBER

you are given strictly increasing array 'arr' and a + int K. find the K^{th} positive integer missing from 'arr'.

arr = {4, 7, 9, 10} = K = 4

missing no are 1, 2, 3, (5), 6, 8, 11, 12 .... K is 4 ∴ 4^{th} missing no output

BRUTE

arr = {2, 3, 4, 7, 11} K = 5

Now 5 could be our answer bc as {2, 3, 4} are present ∴ 5+3 = (8) can be our new answer

arr = {2, 3, 4, (7), 11} K = 5 6 7 8

but 7 also present ∴ shift answer

8+1 = 9

Now 11 > 9 ∴ no shifting required as no new number is present before 9.

∴ (9) solution

```
for( i=0 ---> i=n-1 ){
        if(arr[i] <= K) K++        TC → O(n)
        else return K;
}
```

OPTIMAL

arr = { 2, 3, 4, 7, 11 } **k = 5**

Now in no numbers were missing then the array would be

{1, 2, 3, (4), 5 }

Now — our original is (7) Now 7 − 4 = (3) ∴ there are 3 missing no before 7 that is why is there is no 4 in its place

Similarly

11 − 5 = (6) ∴ there are 6 missing number till this place

{ 2 3 4 7 11 }

1 1 1 3 6

Now 5th missing would lie between

Now BINARY SEARCH

{ 2, 3, (4) 7, 11 }

missing number = 4 − 3 = (1)

as 5 > 1 ∴ low = mid + 1;

{ 2 3 4 (7) 11 }

7 − 4 = (3)

as 5 > 3 ∴ low = mid + 1;

{2, 3, 4, 7, (11)}

11 − 5 = (6)

as 6 > 5 ∴ high = mid − 1;

{2, 3, 4, 7, 1, 3}

high, mid, low (break)

Now

arr[high] = 7 , missing = 3

but we need 5th missing ∴ 5 − 3 = (2) more missing

∴ arr[high] + more = (9) answer

```
fun( ){
  low = 0, high = n-1;
  while(low <= high){
    mid = (low + high)/2;
    missing = arr[mid] - (mid+1);
    if(missing < K) low = mid+1;
    else high = mid-1;
  }
```

∴ is no ≤

∴ first occurence

[Problem] high = -1

arr - {4, 7, 13} K = 3

low

TC → O(logn)

now ans → arr[high] + more

arr[high] + (K − missing)

arr(high] + K − (arr[high] − high − 1)

arr[high] + K − arr[high] + high + 1

= (K + high + 1) answer

or low + 0

as high + 1 = low

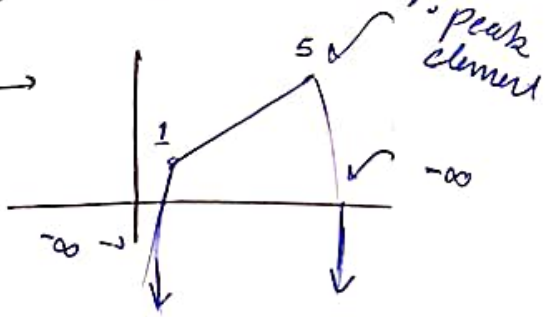
∴ return (high + 1 + K) or (low + K)

FIND PEAK ELEMENT

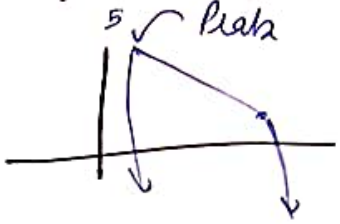
peak element → (arr[i-1] < arr[i] < arr[i+1])

return index of peak element. if there are multiple peak numbers, return the index of any peak num.

also → arr[] = { 1, 2, 3, 4, 5 } →



arr[] = { 5, 4, 3, 2, 1 }



for the first element, the previous element should be considered as -∞ as well as for the last element next element as -∞ also.

BRUTE

```
for(i=0; i<n; i++){
    if((i==0 || arr[i-1] < arr[i]) &&
        ((i== n-1) || arr[i] > arr[i+1])) return i;
}
```

if first element there is no previous i.e. 0

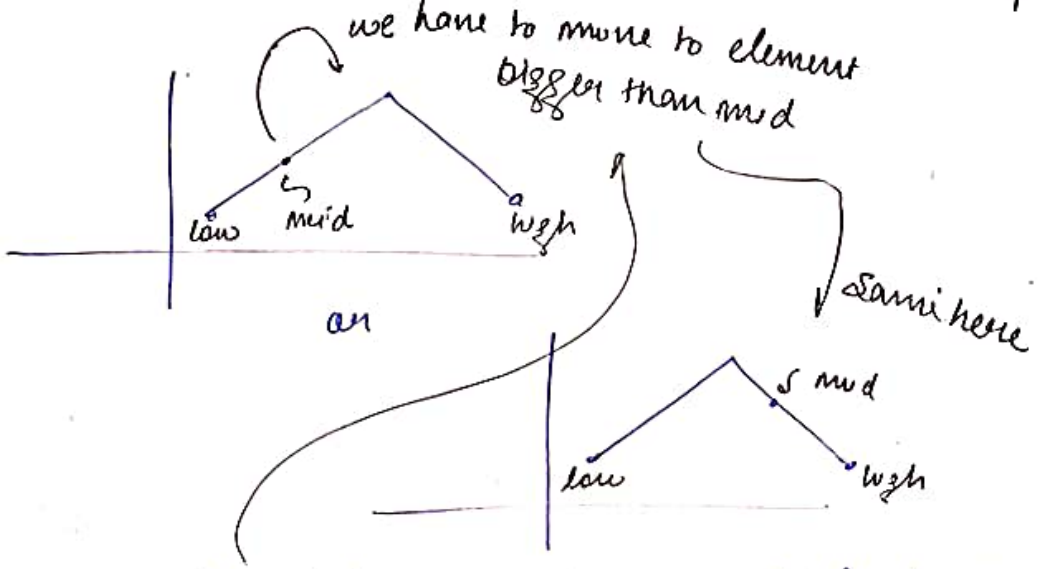
OPTIMAL

arr = {1, 2, 3, 4, 5, 6, 7, 8, 5, 1}

arr = {2} ← this is peak when one element

arr =

{1 2 3 4 5 6 7 8 5 1}



this is also followed when multiple peaks are there

{1 2 3 4 (5) 6 7 8 5 1}

mid

4 < 5

but 5 < 6 ∴ not peak (check from peak)

but 6 > 5 ∴ peak would be on right (low = mid + 1)

else (high = mid - 1)

```
fun(arr, n) {
    if (n == 1) return 0;
    if (arr[0] > arr[1]) return 1;
    if (arr[n-1] > arr[n-2]) return n-1;

    while( low = 1, high = n-2;

    while (low <= high) {
        mid = (low + high)/2;

        if (arr[mid-1] < arr[mid] && arr[mid] > arr[mid+1])
                return mid;

        else if (arr[mid] > arr[mid-1]) low = mid+1;
        else high = mid-1;

    }
    return -1;
}
```

index of peak

no including first and last

MEDIAN OF TWO SORTED ARRAYS OF DIFFERENT SIZES

- return median of two sorted arrays.

arr1[] = {2, 4, 6}, arr2[] = {1, 3, 5}

Result → 3.5

as in sorted array 1 2 3 4 5 6

$\frac{3+4}{2} = (3.5)$

arr1[] = {2, 4, 6}, arr2[] = {1, 3}

Result → √3

{1, 2, (3), 4, 6}

BRUTE

arr1[] = {1, 3, 4, 7, 10, 12} arr2[] = {2, 3, 6, 15}

use merge sort pointer approach

Space used extra → arr3[] = {1, 2, 3, 3, 4, 6, 7, 10, 12, 15}

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

```
i=0  j=0
while (i<n1 && j<n2){
    if (arr1[i] ≤ arr2[j]) arr3.add(arr1[i++])
    else arr3.add(arr2[j++]);

while (i<n1) arr3.add(arr1[i++]);
while (j<n2) arr3.add(arr3[j++]);
```

if (n%2 == 1) return arr3[n/2];

else $\left(\frac{arr3[\frac{n}{2}] + arr[\frac{n}{2}-1]}{2}\right)$;

BETTER in terms of space

{1, 2, 3, 4, 6, 7, 10, 12, 15}

idx1 idx2

here we will not store the element in new array we will sort and move the pointer until out idx1 value is reached and idx2 also and will use that directly without storing

arr1[7 ~~{1, 2, 3, 0~~

{1 3 4 7 10 12}

{2 3 6 15}

got the element

got the mid + 1

	1	2	3	3	(4)	(6)
cnt =	~~0~~	~~1~~	~~2~~	3	~~3~~ (4)	(5)

idx1

for the code check VS code

OPTIMAL

Binary search

(-) arr1 → {1 3 4 7 10 12}

arr2 → {2 3 6 15} → 1 2 3 3 4 | 6 7 10 12 15

as total 10 elements
∴ there will 5 elements each side of median

Now how many element from array 1 can be pick to make up the first half.

→ if we pick (1) element from arr1 then

from arr1 → 1 | 3 4 7 10 12
→ 2 3 6 15
remaining 4 from arr2

the remaining element

but

1 2 3 6 15 | 3 4 7 10 12

not correct configuration as 15<3

∴ we cannot pick only 1 element from arr1

if we pick (4)

then

1 3 4 (7) l_1 2 l_2	10_{s_1} 12 (3) 6 15 s_2

still not valid

here $l_1 > s_2$ ∴ we should pick less from arr1

if 3 pick

1 3 4 2 3	7 10 12 6 15

valid

if pick (2)

1 3 l_1 2 3 (6) l_2	s_1(4) 7 10 12 s_2 15

$l_2 > s_1$ ∴ we should pick more from arr1

Now we have got the condition to eliminate halfes right and left.

and also the range 0 — 6 ✓ all of them from arr1

we will pick the array with less elems

arr1 = {7, 12, 14, 15} ← mid1

arr2 = {1, 2, 3, 4, 9, 11} ← mid2

(0) low (2) mid1 (4) high

∴ mid2 = 5 − mid1 = 3 (element to pick from arr2)

mid1

7 12 | 14 15

1 2 3 | 4 9 11

mid2

here $l_1 > r_2$ ∴ high = mid − 1;

(0) low (mid1) mid2 = 5

(1) high

mid1

l_1 7 12 14 15

∞ 1 2 3 4 9 | 11

l_2 mid2

comparison does not matter

but 9 > 7

∴ $l_2 > r_1$ ∴ low = mid + 1

(2) low (1) high

low

mid1 = 1

mid2 = 4

7 | 12 14 15

1 2 3 4 | 9 11

here 7 < 9 & 4 < 12

∴ pick (1) only

if l_1 or l_2 does not exist take INT_MIN

l_1 = arr1[mid1 − 1] r_1 = arr1[mid1]

l_2 = arr2[mid2 − 1] r_2 = arr2[mid2]

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

but when odd element

arr1 = {2, 4}

arr2 = {1, 3, 4}

1 2 3 | 4 4

3 | 2

partition = $\frac{n1+n2+1}{2} = \frac{2+3+1}{2} = 3$ → on left

⇒ $\frac{\max(l_1, l_2)}{}$ ↪ median

if [2 | 3] then $\min(r_1, r_2)$

code

```
median(a, b) {
    n1 = a.size();
    n2 = b.size();
    if (n1 > n2) return median(b, a);   // making sure to work on smaller array & calling same function again with smaller array in front.

    int low = 0, high = n1;
    int left = (n1 + n2 + 1) / 2;   // work for even and odd both
    while (low <= high) {
        int mid1 = (low + high) / 2;
        int mid2 = left - mid1;

        int l1 = INT_MIN, l2 = INT_MIN;
        int r1 = INT_MAX, r2 = INT_MAX;

        if (mid1 < n1) r1 = a[mid1];   // when we take all element from first array, mid points to index after array size
        if (mid2 < n2) r2 = b[mid2];
```

```
if (mid1-1 ≥ 0) l1 = a[mid-1];   ← l1 should exist.
if (mid2-1 ≥ 0) l2 = b[mid-2];   ← l2 "  "
                                   ↑ solution
if (l1 <= r2 && l2 <= r1) {
        if(n%2==1) return max(l1, l2);
        return (double) (max(l1,l2) + min(r1,r2)) / 2
}
else if (l1 > r2) high = mid1 - 1;
else low = mid1+1;
}
return 0;
}
```

FAIL !!

SOLUTION

O(max(

O(log(min(n1, n2)))