Binary Search...

→ How do we calculate upper bound & lower bound in a binary search.

- Upper bound (>K)

Suppose we have an array →

					↓			
l	1	2	4	6	8	10	15	h

and we have value of K = 7.

Here, we are supposed to "find the smallest value greater than K"

So, from above array we know that the answer would be "8".

→ Here the 1st way to solve this is →

① O(n) + Sorted space (that is our array).
linear Search

So, we can divide our search space into two zones →

(since, it's upper bound)

(a) values > K → Favorable Zone &

(b) values < K → Unfavorable zone.

[Here, l = low, h = high]

l [1 | 2 | 4 | 6 | 8 | 10 | 15] h

[U U U U | F F F]

→ So, here our answer lies in the second zone, and the first element of our second zone is the answer we wanted.

[Since, we know that if our value lies in the favarable zone,

we add "+1" to our mid value to move to the next index

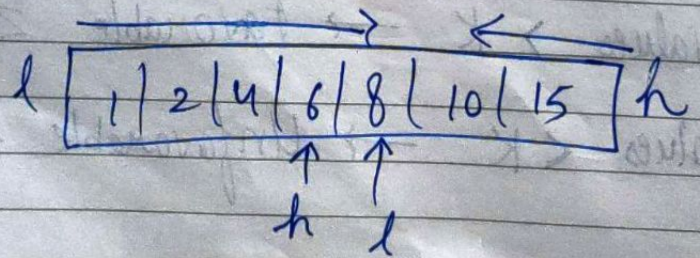
and similarly in case of unfavorable zone

we simply subtract "-1" from our mid value to move to the previous index.

So, after all the searching in the end our l will point to the first element of our ~~favorable zone~~ second zone

and h will point towards the last element of our first zone.]

Ex →



{(h+1, l)}

So, the algo for this or the sol^n for this would be →

```
while (l <= h)
{
    mid = l+(h-l)/2;

    if (a[mid] > K)
    {
        h = mid-1;
    }
    else
    {
        l = mid+1;
    }
}
return l;
```

→ Lower Bound >= K

if K = 6

ans ↓

1	4	5	6	8	10	12
U	U	U	F	F	F	F

So, as we did for the upperbound, in the similar way it will be calculated and the value which we will be returning would be 'l' or 'h+1'.

① Valid Perfect Square →

Given a Positive integer num, write a function which returns True, if num is a perfect square else False.

Ex ① num = 16
Output → True } since, 16 is a Perfect square of 4.

② num = 14
Output → False } since, 14 is not a Perfect square of any no.

(i×i) value <= K → Favorable

①
So,

1 2 3 4 5 6 ---- 16

[F | F | F | F] [U U U U U U]

h l

for 1×1 = 1 ✓ <= K , l = mid+1
2×2 = 4 —— 11 ——→
3×3 = 9
4×4 = 16 (Terminates here).

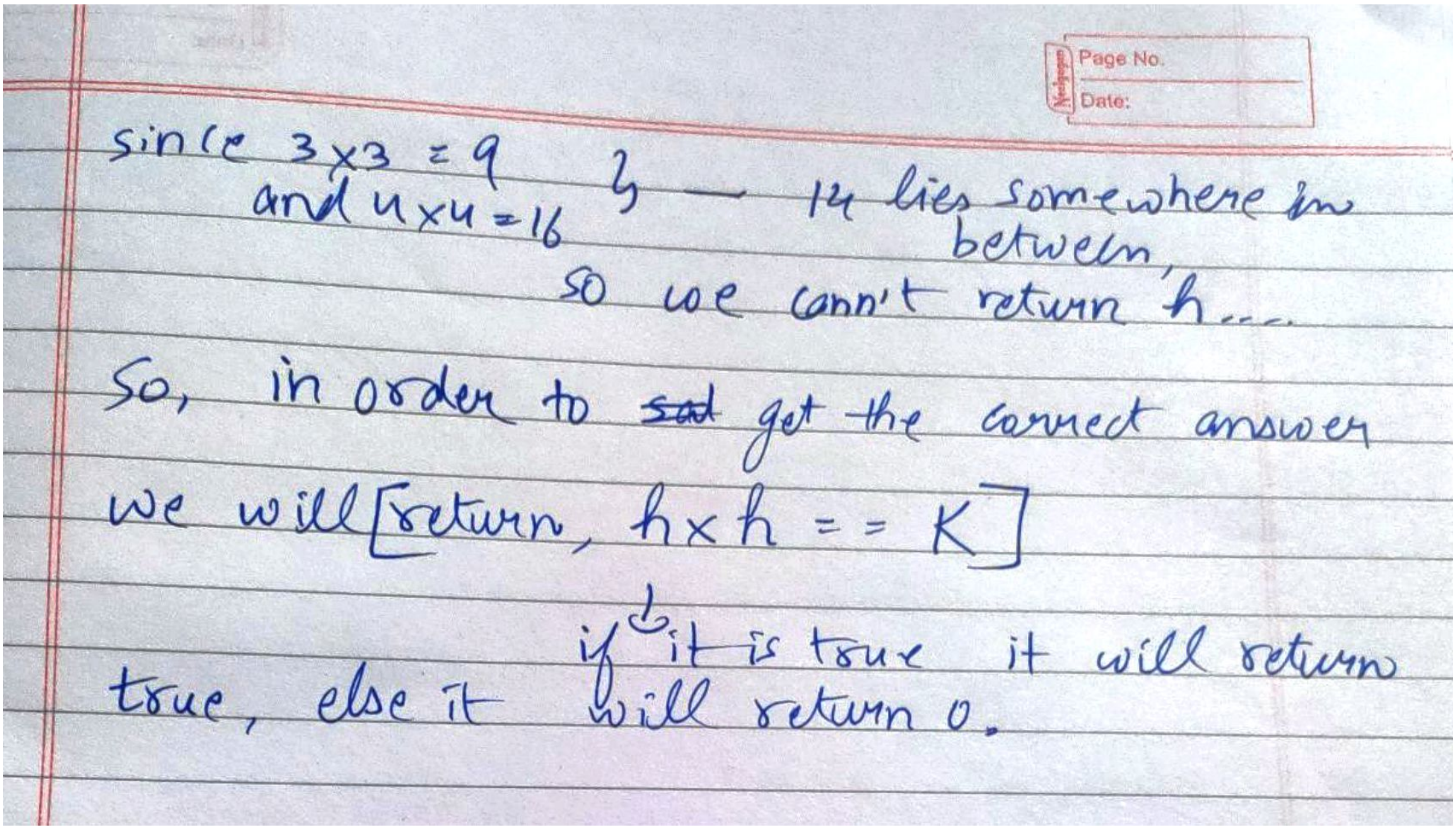
return h;

but in 2nd test case →

1 2 3 4 5 _ _ ___ 14

[F | F | F] [U U ______ U]

h l



since 3×3 = 9
and 4×4 = 16 } — 14 lies somewhere in between,
so we cann't return h...

So, in order to ~~sat~~ get the correct answer
we will [return, h×h == K]
↓
if it is true it will return true, else it will return 0.