875. Koko Eating Bananas

Medium | Topics | Companies

Koko loves to eat bananas. There are `n` piles of bananas, the `ith` pile has `piles[i]` bananas. The guards have gone and will come back in `h` hours.

Koko can decide her bananas-per-hour eating speed of `k`. Each hour, she chooses some pile of bananas and eats `k` bananas from that pile. If the pile has less than `k` bananas, she eats all of them instead and will not eat any more bananas during this hour.

Koko likes to eat slowly but still wants to finish eating all the bananas before the guards return.

Return *the minimum integer* `k` *such that she can eat all the bananas within* `h` *hours*.

Example 1:

```
Input: piles = [3,6,7,11], h = 8
Output: 4
```

Example 2:

```
Input: piles = [30,11,23,4,20], h = 5
Output: 30
```

Example 3:

```
Input: piles = [30,11,23,4,20], h = 6
Output: 23
```

Constraints:

- $1 <= piles.length <= 10^4$
- $piles.length <= h <= 10^9$
- $1 <= piles[i] <= 10^9$

Example

piles = [3, 6, 7, 11]

Choosing

$K = 20$ (bananas / hour)

1hr 1hr 1hr 1hr

$\underbrace{\text{1hr 1hr 1hr 1hr}}_{h' = 4 \text{ hrs total}}$

(if $K \geq piles[i]$, Koko can eat that pile in 1hr)

We realize,

$K = 11$ (max element in piles) gives us the lowest h' (total time to eat).

→ Clearly, increase of K decreases h' (total time to eat) (and vice versa)

So a way to solve this would be:

1. Choose $K = max\_element(piles)$, in the example, $K = 11$
2. Calculate $total\_hours(piles, 11) = 4$ (let's call this h')
3. Since $4 < h$ $(4 < 8)$, we can choose a lower K how about $K = 10$

→ $h' = total\_hours(piles, 10) = 5$

→ Since $5 < h$ $(5 < 8)$, we can still choose a K. Maybe, $K = 9$

→ we repeat the process until we reach $K = 4$, $h' = total\_hours(piles, 4) = 8$. Since $8 = h$, we might have reached the min K, but trying $K = 3$

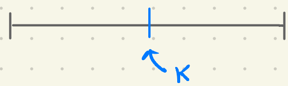
$h' = total\_hours(piles, 3) = 10$. $10 > h$ $(10 > 8)$ ✗

$K = 3$ takes 10 hours, so the min value is $K = 4$ ✓

Decreasing K one by one is really slow,
so we use Binary Search, from 0 to max_value(piles)

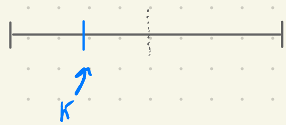
lo = 0, hi = max_value(piles)

→ We start it with $mid = \frac{max\_value(piles)}{2}$



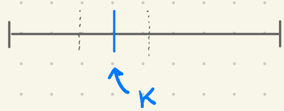
If $h' \le h$, let's try looking in the left range

(lo = lo
hi = mid − 1)



if $h' > h$, let's try looking in the right range

(lo = mid + 1
hi = hi)



if lo > h, we break this iteration.

and we keep track of min K that gives us
an $h' \le h$.