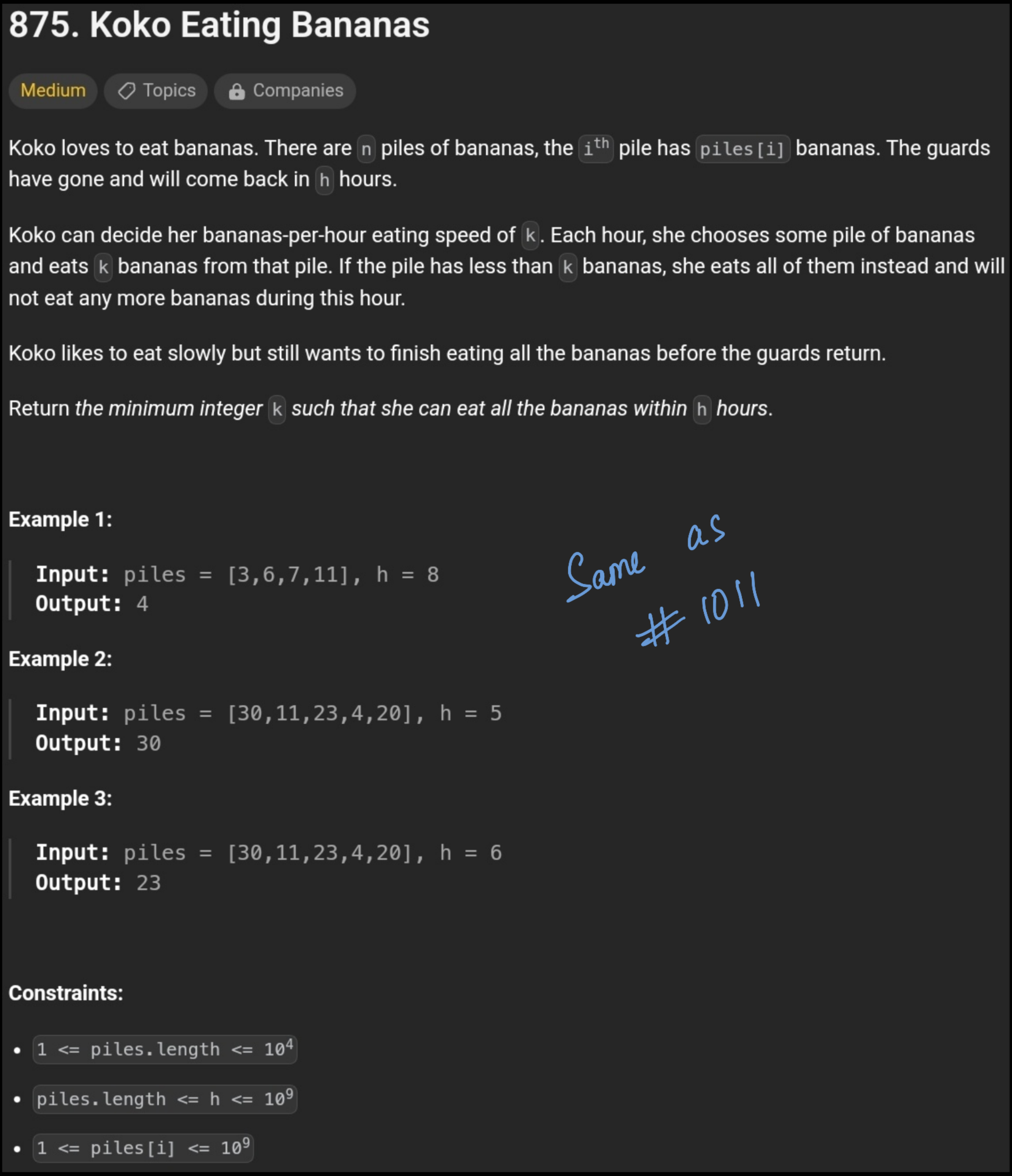


875. Koko Eating Bananas

Koko loves to eat bananas. There are `n` piles of bananas, the `i`th 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
```

Same as #1011

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}

Approach 1: Brute force

minimum value of K can be 1
maximum value of K can be max(piles[i])

So linearly check for each K from 1

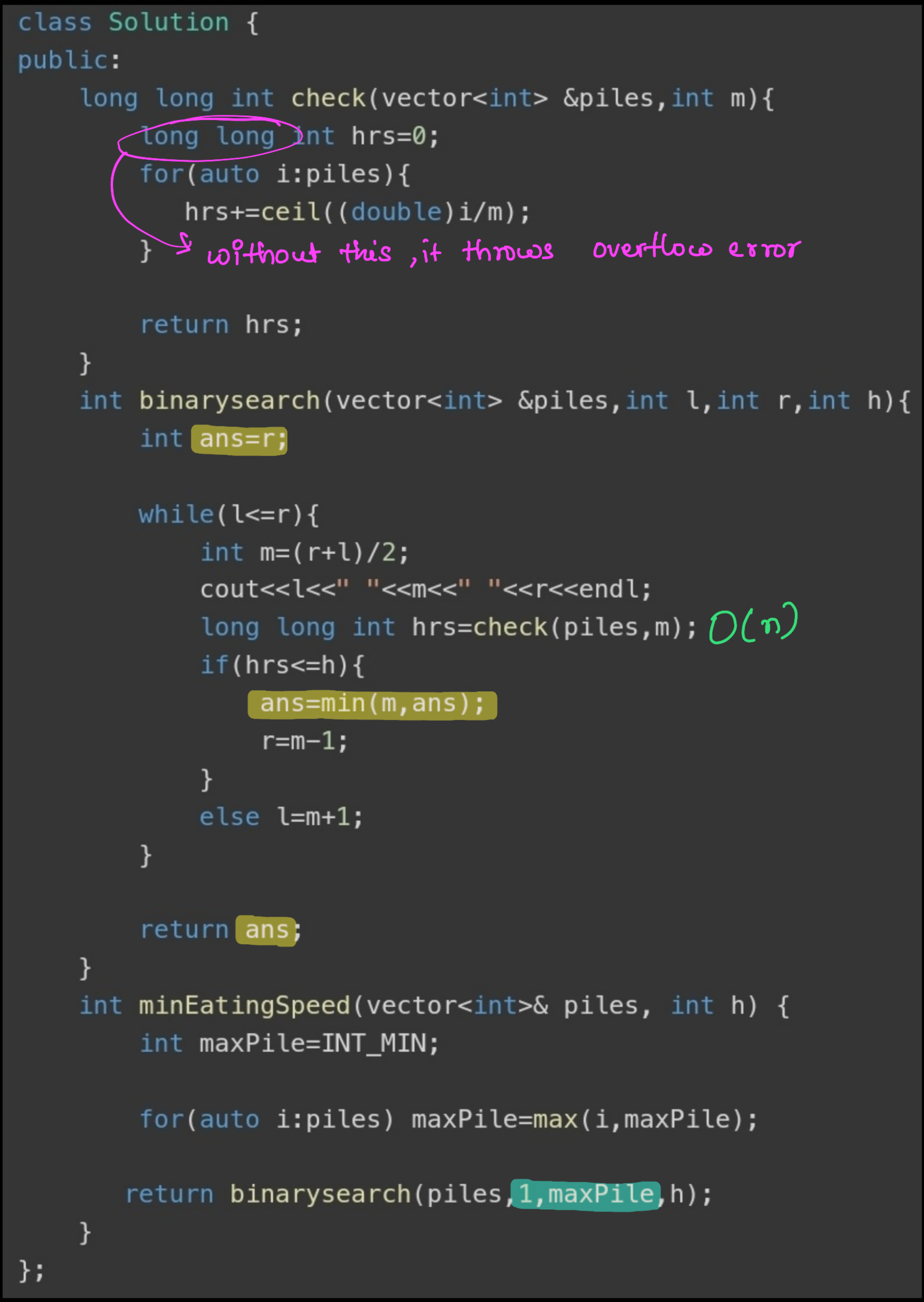
So linearly check for each k from 1 and the value of k which satisfies the condition that all piles can be completed in h hrs is the answer. So return it.

$T(n): O(max(piles[i]) * n)$

i.e. $\Omega(n^2)$

Approach 2:

instead of linear search do binary search



```
class Solution {
public:
    long long int check(vector<int> &piles,int m){
        long long int hrs=0;
        for(auto i:piles){
           hrs+=ceil((double)i/m);
        }  without this ,it throws overflow error

        return hrs;
    }
    int binarysearch(vector<int> &piles,int l,int r,int h){
        int ans=r;

        while(l<=r){
            int m=(r+l)/2;
            cout<<l<<" "<<m<<" "<<r<<endl;
            long long int hrs=check(piles,m); O(n)
            if(hrs<=h){
                ans=min(m,ans);
                r=m-1;
            }
            else l=m+1;
        }

        return ans;
    }
    int minEatingSpeed(vector<int>& piles, int h) {
        int maxPile=INT_MIN;

        for(auto i:piles) maxPile=max(i,maxPile);

       return binarysearch(piles,1,maxPile,h);
    }
};
```

$$T(n) : O(n \log \underset{\underset{max(piles[i])}{\downarrow}}{m})$$