first wrong

one

So for answers we can return high also

04/02/2026
Wednesday

Find Nth root of a number

$\sqrt[3]{27}$ m=27 N=3

ans=3

$\sqrt[3]{27} = 3$

3x3x3

$\sqrt[4]{69}$ m=69 N=4

ans=-1

if ans not found we return -1

Generally we can do in O(n) but we do binary search on answers

Given n=3 m=27

low=1 high=27

mid=14

Now I will check

14x14x14 > 27

So we move left side

high = mid-1;

high=13

low=1 mid=7 high=13

7x7x7 <= 27 false

we move left side

high = mid-1;

high = 6

low=1 mid=3 high=6

3x3x3=27

27==27 -true

So we return 3

take another one as example

$\sqrt[4]{69}$ m=69 n=4

low=1 mid=35 high= 69

35x35x35x35 <69 false

high=mid-1

=34

low=1 mid=17 high=34

17x17x17x17 < 34 false

high=mid-1

=16

low=1 mid=8 high=16

8x8x8x8 < 69 false

high = mid-1

=7

low=1 mid=4 high=7

4x4x4x4 <69 false

high = mid-1 high=3

low=1 ↓

mid=2

2x2x2x2 <69 -true

low=mid+1

low=3 mid=3 high=3

3×3×3×3 < 69 false

high = mid − 1
= 2

search space does not exist

so we does not found

return −1

```
public int FindNthroot (int n, int m) {
  int low=1
  int high=m
  while (low<= high)
  {
      int mid= low+ (high-low)/2;
      int midN= findpower (mid, m, n)
             ↓
          these will give values
          like 0, 1, 2
          If mid==m return 1
           ↓
          ans
          if ans> m return 2
          if ans<m return 0
      if (midN== 1)
            return mid;
      else if (midN== 2)
            high=mid-1;
      else
            low=mid+1;
  }
  return -1;
}
```

```
int findpower (int mid, int m)
                    int n
{   int ans=1;
    for(int i=1;i<=n;i++)
      {
        ans*=mid;
        if(ans>m)
            return 2;
      }
    if(ans==m)
        return 1
    else
        return 0;
}
```

we do this approach beuz when m is 10^9

low = 1 high = 10^9

mid = $\frac{10^9}{2}$

then when I calculate power

$\frac{10^9}{2} \times \frac{10^9}{2} \times \frac{10^9}{2} \times \frac{10^9}{2}$ it overflows

so that i did when it crosses 10^9 or given m return 2 like go left side

Koko eating Bananas

→ Return the min Integer k such that Koko can eat all bananas within h hours

piles[] = [3 6 7 11] h=8 k → bananas/hr

1 pile pile

for this we can do brute force O(n)

but we need to do in $\log_2 n$

Here take

k=1 1 banana/hr

3 6 7 11 27 hrs

k=2 2 bananas/hr

3 6 7 11

we take ceil of array element/k

1st ele 2 bananas 1 hr

remaining 1 banana 1 hr

total 2 hrs

[3 6 7 11]

2 3 4 6 totalhrs = 15

take k=3

3 bananas/hr

[3 6 7 11]

1 2 3 4 totalhrs = 10

take k=4

4 bananas/hr

[3 6 7 11]

1 2 2 3 totalhrs = 8

h=8

we want min right so answer is 4

Here for binary search on answers we take max element from array has high and low as 1

low=1 high=11

mid=6

6 bananas/hr

we send this to a function and check the time if it is less than or equal

1 1 2 2 6 hrs

6<=8

high=mid-1

low=1 mid=3 high=5

3 bananas/hr

1 2 3 4 10 hrs

So we go right side

bcuz hrs are increasing

low=mid+1

=4

low=4 mid=4 high=5

4 bananas/hr

1 2 2 3 8 hrs

I will store it in ans

ans=8 and go left side

high=mid-1

low=4 high=3

So search space does not exist

```
public int totaltime(int[] a, int m){
    int total=0;
    for(int i=0;i<a.length;i++)
    {
        total+= Math.ceil(a[i]/m);
    }
    return total;
}
```

*1.0

```
public int findminbananas(int[] a, int h)
{
    int low=1;
    int high= max(a);
    int ans=1;
    while(low<=high){
        int mid= low+(high-low)/2;
        int totaltime= totaltime(a,mid);
        if(t<=h){
            ans=mid;
            high=mid-1;
        }
        else{
            low=mid+1;
        }
    }
    return low;
}
```

t

Time complexity we are doing bs on max

$\log_2(max)$

and each time calculating time by taking mid

$O(n * \log_2(rmax))$