Optimal Approach (Using Binary Search):

Using Binary Search to optimize

If we ~~opse~~ observe carefully then our code we see ~~that~~ the ~~code~~ answer space [1, n] is sorted.

Thus we apply binary search in the answer space.

Edge case: How to eliminate the halves.

- place Low at 1, high at m
- based on ~~p~~ mid powered to n we check if we get our answer

→ if value is smaller we eliminate left half
→ if value is higher we eliminate right half

But if m and n are big

we can't store mid^n in a variable.
to resolve this we use a function

f(n, m, mid):
→ first declare variable ans to store mid^n
→ run loop n times and then multiply with 'ans'
→ if at a point ans > m, return 2
→ loop completed ans == m then return 1
→ if value is smaller return 0.

based on this above function, we check if mid is our possible answer or we eliminate the halves.
Thus we can avoid the integer overflow case.

Algorithm -

1. Place 2 pointer i.e. high and low
2. Calculate mid

   mid = (low + high) // 2
   ("//" refers to integer division)

3. Eliminate the halves accordingly:

   (1) If func(n, m, mid) == 1:

   we can conclude that mid is our answer.

   (2) If func(n, m, mid) == 0:

   we can conclude that mid is smaller than answer.

   eliminate left half
   consider right half } (i.e low = mid + 1).

   (3) If func(n, m, mid) == 2:

   conclude: mid is greater than answer

   eliminate right half
   consider left half } (high = mid - 1)