

Sure, let's break it down with a simple explanation and example.

**Explanation**

The algorithm you're describing involves a variable st that starts at 2, and in each iteration of the loop, st is replaced with its square (st = st \* st). The loop continues until st is greater than or equal to N. The goal is to find out how many iterations (or steps) it takes for st to exceed N.

Let's analyze this with a step-by-step example and simplify the explanation:

1. **Initial State:**
   * st starts at 2.
   * steps starts at 0.
2. **Iteration 1:**
   * st becomes 2^2 (which is 4).
   * steps is incremented to 1.
3. **Iteration 2:**
   * st becomes 4^2 (which is 16).
   * steps is incremented to 2.
4. **Iteration 3:**
   * st becomes 16^2 (which is 256).
   * steps is incremented to 3.

**General Pattern**

Each time st is squared, the power of 2 doubles. Initially, st is 2^1, then it becomes 2^2, then 2^4, then 2^8, and so on.

Notice the pattern:

* In Iteration 1: st is 2^2 (which is 4).
* In Iteration 2: st is 2^4 (which is 16).
* In Iteration 3: st is 2^8 (which is 256).

This pattern shows that after each iteration, the exponent of 2 is doubling. If we denote the exponent in iteration i as 22i2^{2^i}22i, the loop continues until 22i2^{2^i}22i is greater than or equal to NNN.

**Time Complexity**

1. **Determine the Loop Count:**
   * The loop stops when 22i≥N2^{2^i} \geq N22i≥N.
   * Taking the logarithm (base 2) on both sides gives us: 2i≥log2N2^i \geq \log\_2 N2i≥log2​N.
   * Taking the logarithm (base 2) again gives us: i≥log2(log2N)i \geq \log\_2 (\log\_2 N)i≥log2​(log2​N).

Hence, the number of iterations iii is approximately log2(log2N)\log\_2 (\log\_2 N)log2​(log2​N).

**Example**

Suppose N=1000N = 1000N=1000:

1. **Iteration 1:** st = 22=42^2 = 422=4
2. **Iteration 2:** st = 42=164^2 = 1642=16
3. **Iteration 3:** st = 162=25616^2 = 256162=256
4. **Iteration 4:** st = 2562=65536256^2 = 655362562=65536 (which is greater than 1000).

So, it took 4 iterations for st to exceed 1000. This aligns with our formula of log2(log2N)\log\_2 (\log\_2 N)log2​(log2​N), which indicates that the time complexity is O(log(logN))O(\log (\log N))O(log(logN)).

**Conclusion.**

The algorithm's time complexity is O(log(logN))O(\log (\log N))O(log(logN)) because each iteration effectively doubles the power of 2, and the number of iterations needed to reach or exceed NNN grows very slowly compared to NNN.