Improved Fast Powering Algorithm

```python

@timer
def mod_pow_2(g, A, N):
    """
    Computes g**A (mod N), where g is the base, A is the exponent
    and N is the modulus.

    parameters
    ----------
    g: int
        The base
    A: int
        The exponent
    N: int
        The modulus

    Returns
    -------
    int: g**A (mod N)
    """

    a, b = g, 1
    while A > 0:
        if (A-1)%2 == 0:
            b = (b*a)%N
        a, A = (a*a)%N, A//2
    return b
```

Listing 1: Fast Powering Algorithm

Proof.

1. **Initialization** The algorithm begins by initializing a and b. That is, $a = g$ and $b = 1$. Here, a is initially the base $g \mod N$, and b is $b \equiv 1 \mod N$.

2. **Loop Invariant** For each iteration, the algorithm mantains that $ba^A \equiv g^A \mod N$.

3. **While loop execution** The algorithm enters a while loop and runs until $A \not> 0$.

   For each iteration, we check the parity of A using $A \equiv 1(\mod 2)$. If A is odd, then the least significan bit in the binary representation of A is 1. In this case, we update b as $b = ba \mod N$. This ensure that b accumulates the power of g whenever the current bit in the binary representation of A is 1.

4. Next, we update a to $a^2 \mod N$ regardless of whether A is odd or even. This prepares for the next iteration and squares the base for the next bit of A.

5. Also update A as $A = floor(A/2) \mod N$. This shifts the bits in the binary representation of A to the right.

6. **Termination**

   The loop continues until A brcomes 0. In the end, $b \equiv g^A \mod N$.

$\square$