Ryan Pollack

1. Consider how we would compute the power of N^k:
As given in the problem, $N^k = N \cdot N \cdot N \cdot N \cdot N \dots$ where the number of Ns is based on the value of k (or "k times").
So, if $k=1$, the number of multiplications required to compute N^k is 1. If $k=2$, it would take 2 multiplications:

k	number of multiplications needed to compute N^k
1	1
2	2
3	3
4	4
5	5

As seen from the table, as k increases, the needed number of multiplications to compute N^k increases linearly with it. Therefore, we know computing N^k will take $O(k)$ multiplications.

2. Consider how we could compute the power of N^k, where $N^k = N^{2^k} = (N^2)^k = N^2 \cdot N^2 \cdot N^2 \cdot N^2 \cdot N^2 \dots$, where the number of N^2's appears k times. N^k can be computed by squaring N in k multiplications based on the given assumptions $k = 2^k$ and N^k taking $O(k)$ multiplications from problem 1. From this, we know $(N^2)^k$ is calculated in k multiplications via repeated squaring. Therefore, N^k can be computed in $O(k)$ multiplications via repeated squaring.

3. To do this, we want to compute N^k by writing k as binary representation. To represent k, we require $\lceil \log_2 k \rceil + 1$ bits/digits. So if:

$$N^{13} = 3^{1011_2} = 3^8 \cdot 3^4 \cdot 3^1$$

$$N^k = N^{[\log_2 k + 1]_2}$$

We require almost $\log_2 k$ multiplications.
Now, we need to compute $N^1, N^2, N^4, N^8, \ldots N^{[\log_2 k]}$.
So, we perform about $\log_2 k$ multiplications and each k is the square of the previous. So:

$$N^2 = (N^1)^2$$

$$N^4 = (N^2)^2$$

$$N^8 = (N^4)^2$$

$$N^{16} = (N^8)^2$$

The final complexity of algorithm is $O(\log_2 k)$, proving the argument.

4. Let's consider N^{2^k} multipying each other up to R terms to get N^k. So:

$$N^{2^1} \cdot N^{2^2} \cdot N^{2^3} \cdot N^{2^4} \ldots N^{2^R} = N^k$$

$$N^{(2^1 + 2^2 + 2^3 + 2^4 + \ldots + 2^R)} = N^k$$

We want to find R to 'see' how long it takes to get done. So:

$2^1 + 2^2 + 2^3 + 2^4 + \ldots + 2^R = k$, which is an example of geometric progression. So, we use the formula:

$$\frac{a(r^n - 1)}{r-1} \longrightarrow \frac{2^1(2^R - 1)}{2-1} = k$$

$$2^R - 1 = \frac{k}{2} \longrightarrow 2^R = \frac{k}{2} + 1$$

$$R \log_2 2 = \log_2 \left(\frac{k}{2} + 1\right)$$

$$O(\log_2 k) \approx R = \log_2 \left(\frac{k}{2} + 1\right)$$

So, it can be done at most $O(\log_2 k)$, proving the argument.

5. When multiplying N with itself K times requires k multiplications, as was shown in problem 1. Now, when computing N^k by repeated squaring it requires at most $2\log k$ multiplications, as shown in problem 4. As the numbers get bigger so that $k \to \infty$, we know that:

$$\lim_{k \to \infty} \frac{2\log_2 k}{k} = 0$$

Therefore, multiplication by repeated squaring is asymptotically far better than multiplying N by itself k times. The overall complexities are K (for multiplying N together k times) and $2\log k$ (for multiplication by repeated squaring).

Bonus. Considering each multiplication costs about $O((\log N)^2)$, computing N! requires N-1 multiplications as shown below:

$$\log^2 N + \log^2(N \cdot (N-1)) + \dots + \log^2(N!) =$$

$$\log^2(N^N \cdot (N-1)^{N-1} \cdot (N-2)^{N-2} \cdot \dots \cdot 1) =$$

$$O(\log^{\frac{N(N+1)}{2}} \cdot N^2)$$

$$O(N^2(\ln N)^2)$$