A mathematical expression that involves N's and N^2s and N's raised to other powers is called a polynomial, and that's what the "P" in "P = NP" stands for. P is the set of problems whose solution times are proportional to polynomials involving N's.

Obviously, an algorithm whose execution time is proportional to N^3 is slower than one whose execution time is proportional to N. But such differences dwindle to insignificance compared to another distinction, between polynomial expressions — where N is the number being raised to a power — and expressions where a number is raised to the Nth power, like, say, 2^N.

NB

If an algorithm whose execution time is proportional to N takes a second to perform a computation involving 100 elements, an algorithm whose execution time is proportional to N^3 takes almost three hours. But an algorithm whose execution time is proportional to 2^N takes 300 quintillion years. And that discrepancy gets much, much worse the larger N grows. 10^{18}

So the question "Does P equal NP?" means "If the solution to a problem can be verified in polynomial time, can it be found in polynomial time?" Part of the question's allure is that the vast majority of NP problems whose solutions seem to require exponential time are what's called NP-complete, meaning that a polynomial-time solution to one can be adapted to solve all the others. And in real life, NP-complete problems are fairly common, especially in large scheduling tasks. The most famous NP-complete problem, for instance, is the so-called traveling-salesman problem: given N cities and the distances between them, can you find a route that hits all of them but is shorter than ... whatever limit you choose to set?