WIKIPEDIA

Pseudo-polynomial time

In computational complexity theory, a numeric algorithm runs in **pseudo-polynomial time** if its running time is a polynomial in the *numeric value* of the input (the largest integer present in the input) — but not necessarily in the *length* of the input (the number of bits required to represent it), which is the case for polynomial time algorithms.

In general, the numeric value of the input is exponential in the input length, which is why a pseudo-polynomial time algorithm does not necessarily run in polynomial time with respect to the input length.

An NP-complete problem with known pseudo-polynomial time algorithms is called weakly NP-complete. An NP-complete problem is called strongly NP-complete if it is proven that it cannot be solved by a pseudo-polynomial time algorithm unless P=NP. The strong/weak kinds of NP-hardness are defined analogously.

Contents

Example

Consider the problem of testing whether a number n is prime, by naively checking whether no number in $\{2, 3, \ldots, \sqrt{n}\}$ divides n evenly. This approach can take up to $\sqrt{n} - 1$ divisions, which is sub-linear in the *value of n* but exponential in the *length of n* (which is about $\log(n)$). For example, a number n slightly less than 10,000,000,000 would require up to approximately 100,000 divisions, even though the length of n is only 10 digits. Moreover one can easily write down an input (say, a 300-digit number) for which this algorithm is impractical. Since computational complexity measures difficulty with respect to the *length* of the (encoded) input, this naive algorithm is actually exponential. It *is*, however, pseudo-polynomial time.

Contrast this algorithm with a true polynomial numeric algorithm — say, the straightforward algorithm for addition: Adding two 9-digit numbers takes around 9 simple steps, and in general the algorithm is truly linear in the length of the input. Compared with the actual numbers being added (in the billions), the algorithm could be called "pseudo-logarithmic time", though such a term is not standard. Thus, adding 300-digit numbers is not impractical. Similarly, long division is quadratic: an m-digit number can be divided by a n-digit number in $O(mn)$ steps (see Big O notation.)

In the case of primality, it turns out there is a different algorithm for testing whether n is prime (discovered in 2002), which runs in time $O((\log n)^6)$.

Another example of a problem that, generally, can only be solved in pseudo-polynomial time is the Knapsack problem — unless P=NP.

Generalizing to non-numeric problems

Although the notion of pseudo-polynomial time is used almost exclusively for numeric problems, the concept can be generalized: The function m is pseudo-polynomial if $m(n)$ is no greater than a polynomial function of the problem size n and an additional property of the input, $k(n)$. (Presumably, k is chosen to be something relevant to the problem.)

This makes numeric polynomial problems a special case by taking k to be the numeric value of the input.

The distinction between the value of a number and its length is one of encoding: if numeric inputs are always encoded in unary, then *pseudo-polynomial* would coincide with *polynomial*.

See also

- Strongly NP-complete
- Quasi-polynomial time

References

- M. R. Garey and D. S. Johnson. Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman and Company, 1979.