- Let A be an algorithm to a problem Π. For any instance I of Π, let $be(I)$ be the binary encoding of I. Let $|I|$ be the length of $be(I)$, that is $|I| = |be(I)|$ (ex. the length of a 0-1 knapsack instance is $O(n((\lg \max_i v_i) + (\lg \max_i w_i)) + \lg W))$. Also, let β be the length of unary representation of a maximum number in I (ex. W in a 0-1 knapsack instance), or 0 if no numbers occur in I. Significantly, the length of unary encoding of β is not necessarily bounded by $poly(|be(\beta)|)$.

  In the time complexity of A[1],

  - if the number of objects in I is in the exponent, then A is called an *exponential time* algorithm (ex. naive $O(2^n)$ time algo for vertex cover),
  - if $|I|$ is not in the exponent and β occurs in the base, noting that $\beta = 2^{\lg \beta}$, A is called a *pseudo-polynomial time* algorithm (ex. naive $O(n)$ time algorithm for determining whether the given positive integer n is a prime; or the dynamic programming based algorithm for 0-1 knapsack, which takes $O(Wn) = O(2^{\lg W} n)$ time),
  - if both $|I|$ and $\lg \beta$ are not in the exponent, but $\lg \beta$ does occur in the base, then A is a *weakly-polynomial time* algorithm (ex. computing $gcd(a, b)$ for $a > b \geq 1$ in $O(\lg b)$ time)
  - if β does not appear at all and the number n of objects in I appears in the base, then A is called a *strongly-polynomial time* algorithm, and (ex. $O(n \lg n)$ time algorithm to merge sort n numbers)
  - if the time complexity of A is $2^{(\lg n)^c}$ for $c > 1$, then A is called a *quasi-polynomial time* algorithm (such algorithms are considered better to exponential time algorithms but not as efficient as polynomial time algorithms).

- It is desirable to have a strongly-polynomial time algorithm. Both the strongly- and weakly-polynomial time algorithms are polynomial time algorithms. However, if the algorithm's time complexity is a weakly-polynomial, then that algorithm is said to be a weakly-polynomial time algorithm; otherwise, we simply say that it is a polynomial time algorithm. Further, both the pseudo-polynomial time and exponential time algorithms are exponential in time complexity since the input complexity is in the exponent of time complexity; significantly, a pseudo-polynomial time algorithm takes exponential time when the input instance has large numbers.

- Assuming $P \neq NP$, no NP-hard problem is solvable in polynomial time. However, the NP-hardness of a problem Π does not necessarily rule out the possibility of solving it with a pseudo-polynomial time algorithm. If Π is NP-hard and Π does not have numerical parameters, then Π cannot be solved by a pseudo-polynomial time algorithm unless $P = NP$. Thus, assuming $P \neq NP$, only NP-complete problems that are potential candidates for being solved by pseudo-polynomial time algorithms are those problems that have numerical parameters. Some of the examples for numerical problems include partition, 0-1 knapsack, integer partition, bin packing, and TSP. While other problems being numerical problems is obvious, the TSP problem is a numerical since the edge weights are numerical parameters. On the other hand, the CLIQUE problem may not be considered as a numerical problem since the clique size k in any reasonable instance is upper bounded by n.

  From the above definitions, an algorithm that solves a problem Π is called a pseudo-polynomial time algorithm for Π if its time complexity is $O(poly(|I|, \beta))$. For any decision problem Π, let Π_p denote

[1] Though we use $\lg n$ bits to label/index any object among n objects I has, in the word-RAM model of computation, usually, any of these labels or references to them fit in a word and hence occupy $O(1)$ space.

the subproblem of Π obtained by restricting Π to only those instances I that satisfy $\beta \leq poly(|I|)$. (To remind, $|I|$ is $|be(I)|$.) The problem Π is called *strongly NP-hard* if Π_p is NP-hard.

Lemma. *Unless $P = NP$, there can be no pseudo-polynomial time algorithm for any strongly NP-hard problem.*

Proof. For the sake of contradiction, assume a pseudo-polynomial time algorithm $\mathcal{A}$ exists for a strongly NP-hard problem Π. That is, Π_p is NP-hard. Given any input string w, first check whether w encodes an instance I of Π satisfying $\beta \leq poly(|I|)$. (That is, we check whether w is an instance of Π_p.) If so, apply $\mathcal{A}$ to I. For any $I \in \Pi_p$, the $\mathcal{A}$ takes $poly(|I|, \beta = O(poly(|I|))) = poly(|I|)$ time, contradicting the supposition $P \neq NP$. $\square$

In other words, a decision problem Π is strongly NP-hard if every problem in NP can be reduced to Π in polynomial time such that the length of every number in unary representation in the reduced instance $f(w)$ is at most a polynomial in the length of the binary encoding of w. That is, to prove Π is strongly NP-hard, one needs to give a polynomial time reduction from any strongly NP-hard problem to Π_p. One such strongly NP-hard numerical problem is binpacking, thanks to polynomial time reduction from 3-dimensional matching. The TSP is a strongly NP-hard problem since it remains NP-hard even if each number involved in it is upper bounded by the number of nodes in the graph. (And, even if one considers the CLIQUE problem as a numerical problem due to number k in any instance, this problem is strongly NP-hard since it remains NP-hard even if k in the instance is upper bounded by the number of nodes in the graph.) A decision problem Π is *strongly NP-complete* if Π belongs to NP and Π is strongly NP-hard.

- An NP-hard problem that is not strongly NP-hard is said to be *weakly NP-hard*. Any weakly NP-hard problem can have a pseudo-polynomial time algorithm without disobeying $P \neq NP$. The example weakly NP-hard numerical problems include 0-1 knapsack, integer knapsack, subsetsum, and partition. A decision problem Π is *weakly NP-complete* if Π belongs to NP and Π is weakly NP-hard.

References:
”Strong” NP-Completeness Results: Motivation, Examples, and Implications by M. R. Garey and D. S. Johnson, JACM ’78, Vol 25, No 3, pp 499-508.