1. Show that

\[ f(n) \text{ is } \Theta(g(n)) \]

implies

\[ g(n) \text{ is } \Theta(f(n)). \]

**Solution.** \( f(n) \text{ is } \Theta(g(n)) \)

iff

there exist \( c > 0, d > 0, M \in \mathbb{N} \) such that \( c g(n) \leq f(n) \leq d g(n) \) for all \( n > M \)

iff

there exist \( c > 0, d > 0, M \in \mathbb{N} \) such that \( g(n) \leq \frac{1}{c} f(n) \) and \( \frac{1}{d} f(n) \leq g(n) \) for all \( n > M \)

implies

there exist \( c' > 0, d' > 0, M' \in \mathbb{N} \) (namely \( c' = \frac{1}{d}, d' = \frac{1}{c} \) and \( M' = M \)) such that \( c' f(n) \leq g(n) \leq d' f(n) \) for all \( n > M \)

iff

\( g(n) \text{ is } \Theta(f(n)). \)

2. Suppose we have a computational problem X and two algorithms for X, A and B. Suppose that the complexities of A and B are \( f(n) \) and \( g(n) \) respectively. Given each of the following pieces of information, which algorithm is more efficient (in the sense of its complexity having a lower rate of growth), and what is the strongest conclusion we can reach (in terms of \( \mathcal{O} \) and \( \Theta \)) about the problem’s complexity?

(a) \( f(n) \) is \( \mathcal{O}(g(n)) \) but \( f(n) \) is not \( \Theta(g(n)) \).

**Solution.** So the rate of growth of \( f(n) \) is at most that of \( g(n) \), but is not equal to the rate of growth of \( g(n) \). In other words, the rate of growth of \( f(n) \) is strictly less than that of \( g(n) \), which means that algorithm A is definitely more efficient than algorithm B.

The strongest conclusion we can draw about the complexity of the problem X is that it is \( \mathcal{O}(f(n)) \): the amount of time it takes to solve the problem is at most \( f(n) \), so at most a multiple of \( f(n) \). It may be possible to solve the problem with a lower rate of growth than \( \Theta(f(n)) \).

We can also say that X is \( \mathcal{O}(g(n)) \), but this would be a weaker conclusion.

(b) \( f(n) \) is \( \Theta(n^2) \) and \( g(n) \) is \( \mathcal{O}(f(n)) \).

**Solution.** So the rate of growth of \( g(n) \) is at most the rate of growth of \( f(n) \), and the rate of growth of \( f(n) \) is the same as the rate of growth of \( n^2 \). We can deduce that \( g(n) \) is \( \mathcal{O}(n^2) \), but this is not the point. Algorithm B must be at least as efficient as algorithm A, though they may be equally efficient.
The strongest conclusion we can draw about the complexity of the problem X is that it is \( O(g(n)) \). We can also say that X is \( O(f(n)) \) and that X is \( O(n^2) \), but these are weaker conclusions, unless we also know that \( g(n) = \Theta(f(n)) \).

(c) \( f(n) = O(g(n)) \) and the complexity of the problem is \( \Theta(g(n)) \).

**Solution.** So the rate of growth of \( f(n) \) is at most the rate of growth of \( g(n) \), which means that algorithm A must be at least as efficient as algorithm B. However, since the complexity of the problem is \( \Theta(g(n)) \), this means that the most efficient algorithm for X is \( \Theta(g(n)) \), i.e. has the same rate of growth as \( g(n) \). So A cannot have a strictly lower rate of growth than B; A and B are equally efficient.

We can also therefore deduce that \( f(n) = \Theta(g(n)) \) and that X is \( \Theta(f(n)) \), but this is no stronger than the given information that X is \( \Theta(g(n)) \).

3. Show that if \( b \in \mathbb{N} \) and \( t(n) \geq n \), \( t(n) \ast b^{t(n)} \) is \( 2^{O(t(n))} \).

(The above is used in the proof of a theorem in Lecture 22.)

**Note:** A function \( f(n) \) is \( 2^{O(t(n))} \) if there exist \( c > 0 \) and a positive integer \( M \) s.t. \( f(n) \leq 2^{ct(n)} \) for all \( n \geq M \).

**Solution.** We have to show that there exist \( c > 0 \) and \( M \in \mathbb{N} \) such that

\[
t(n) \ast b^{t(n)} \leq 2^{ct(n)} \text{ for all } n \geq M
\]

We have:

\[
t(n) \ast b^{t(n)} \leq 2^{t(n)} \ast b^{t(n)} = (2 \ast b)^{t(n)} \leq (2^b)^{t(n)} = 2^{bt(n)} \text{ for all } n \geq 1
\]

and therefore we can take \( c = b \) and \( M = 1 \) above.

This concludes the proof.

4. Let \textsc{Subset-Sum} be the problem of deciding if, given a collection of integers \( x_1, \ldots, x_n \) and a target number \( t \), there exists a subcollection \( \{y_1, \ldots, y_l\} \subseteq \{x_1, \ldots, x_n\} \) that adds up to \( t \):

\[
\sum_{i \in \{1, \ldots, l\}} y_i = t
\]

Show that \textsc{Subset-Sum} is in \textsc{NP}.

**Solution.** We can write a non-deterministic algorithm which generates in parallel all possible subcollections \( \{y_1, \ldots, y_l\} \subseteq \{x_1, \ldots, x_n\} \) (by choosing, for each \( i \in \{1, 2, \ldots, n\} \), whether \( x_i \) belongs to the subcollection or not). The machine then explores in parallel all the generated subcollections. It accepts on a particular branch (subcollection) if the sum of the elements in the subcollection is \( t \). The generation phase takes \( n \) steps, and the checking phase takes up to \( n \) steps.