Chapter 3 Exercise Solutions

Jörg Barkoczi

3

3.1

3.1-1

Let $f(n)$ and $g(n)$ be asymptotically nonnegative functions. Using the basic definition of Θ-notation, prove that $max(f(n), g(n)) = \Theta(f(n) + g(n))$.

Given constants c_1, c_2 and n_0, a function $(m(n) = max(f(n), g(n))) \in \Theta(f(n)+g(n))$ if and only if $0 \leq c_1(f(n)+g(n)) \leq m(n) \leq c_2(f(n)+g(n))$. Since $m(n) \leq f(n) + g(n)$ we already have an upper bound to work with and can thus default c_2 to just 1.

For the lower bound we can just take the factor $min(\frac{f(n)}{f(n)+g(n)}, \frac{g(n)}{f(n)+g(n)})$, which shrinks the expression $f(n) + g(n)$ to $min(f(n), g(n))$ and therefore is always smaller or equal to $max(f(n), g(n))$.

And thus our function $m(n)$ satisfies $0 \leq min(\frac{f(n)}{f(n)+g(n)}, \frac{g(n)}{f(n)+g(n)})(f(n)+g(n)) \leq m(n) \leq f(n) + g(n), \forall n > 0$ and is indeed in the set of functions described by $\Theta(f(n) + g(n))$.

3.1-2

Show that for any real constants a and b, where $b > 0$,

$$(n + a)^b = \Theta(n^b).$$

Let us start by showing that $(n + a)^b = O(n^b)$. This requires us to find constants c, n_0 such that

$$0 \leq (n + a)^b \leq cn^b, \forall n \geq n_0.$$

Let $c = 2^b$

$$\Leftrightarrow (n + a)^b \leq (2n)^b$$

and thus we have $(n + a)^b \leq (2n)^b, \forall n \geq n_0 = |a|$ which implies

$$(n + a)^b = O(n^b).$$

Next we need to show that $(n+a)^b = \Omega(n^b)$, which again requires us to find constants c, n_0 such that

$$0 \leq cn^b \leq (n+a)^b, \forall n \geq n_0.$$

Let $c = (\frac{1}{2})^b$.

$$\Rightarrow (n+a)^b \geq (\frac{1}{2}n)^b$$

and thus we have $(n+a)^b \geq (\frac{1}{2}n)^b, \forall n \geq n_0 = 2|a|$ which implies

$$(n+a)^b = \Omega(n^b).$$

And therefore, by Theorem 3.1, $(n+a)^b = \Theta(n^b)$.

3.1-3

Explain why the statement, "The running time of algorithm A is at least $O(n^2)$", is meaningless.

Saying that the running time of algorithm A is at least $O(n^2)$ gives no information about the worst-case running time, because "at least" implies the best-case input. It gives no information on the best-case running time either, since the O-notation bounds a function from the above, not from below as the Ω-notation does. Therefore the statement is to be considered meaningless.

3.1-4

Is $2^{n+1} = O(2^n)$?

Inequality to prove:

$$0 \leq 2^{n+1} \leq c \cdot 2^n, \forall n \geq n_0.$$

Let $c = 2$, then

$$2^{n+1} \leq 2 \cdot 2^n = 2^{n+1}, \forall n \geq 0.$$

Therefore $2^{n+1} = O(n^2)$.

Is $2^{2n} = O(2^n)$?

Inequality to prove:

$$0 \leq 2^{2n} \leq c \cdot 2^n, \forall n \geq n_0.$$
$$2^{2n} \leq c \cdot 2^n$$
$$2^n \leq c$$

There is obviously no constant c that satisfies

$$\lim_{n \to \infty} 2^n \leq c,$$

therefore $2^{2n} \neq O(2^n)$.

3.1-5

Prove Theorem 3.1.

"For any two functions $f(n)$ and $g(n)$, we have $f(n) = \Theta(g(n))$ if and only if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$" (Theorem 3.1).

Per definition $\Theta(g(n))$ requires the existence of constants c_1, c_2, n_0 such that

$$0 \leq c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n), \forall n \geq n_0.$$

Therefore we can split of the inequalities as

$$0 \leq c_1 \cdot g(n) \leq f(n), \forall n \geq n_0,$$

which implies $f(n) = \Omega(n)$, and

$$0 \leq f(n) \leq c_2 \cdot g(n), \forall n \geq n_0,$$

which implies $f(n) = O(n)$.

From the other side of the equivalence, if we have constants c_a, c_b, n_a, n_b such that

$$0 \leq c_a \cdot g(n) \leq f(n), \forall n \geq n_a,$$

and

$$0 \leq f(n) \leq c_b \cdot g(n), \forall n \geq n_b,$$

we can let $n_0 = max(n_a, n_b)$ and thus satisfy

$$0 \leq c_a \cdot g(n) \leq f(n) \leq c_b \cdot g(n), \forall n \geq n_0.$$

3.1-6

Prove that the running time of an algorithm is $\Theta(g(n))$ if and only if its worst-case running time is $O(g(n))$ and its best-case running time is $\Omega(g(n))$.

If the worst-case running time of an algorithm is $O(g(n))$, the implication is that the running time is bounded from above (by some funtion $f(n) = O(g(n))$) for any input of size n.
Likewise if the best-case running time of an algorithm is $\Omega(g(n))$, the implication is that the running time is bounded from below (by some function $f(n) = \Omega(g(n))$) for any input of size n.
Therefore by Theorem 3.1, that algorithm's running time is $\Theta(g(n))$.

Written as an equivalence:

$$\begin{aligned}
&f(n) \in \Theta(g(n)) \\
&\qquad \Leftrightarrow \exists c_1, c_2, n_0 : 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n), \forall n \geq n_0 \\
&\qquad \Leftrightarrow (0 \leq c_1 g(n) \leq f(n) \wedge 0 \leq f(n) \leq c_2 g(n)), \forall n \geq n_0 \\
&\qquad \Leftrightarrow f(n) \in \Omega(g(n)) \cap O(g(n))
\end{aligned}$$

3.1-7

Prove that $o(g(n)) \cap \omega(g(n))$ is the empty set.

For a function $f(n)$ to be considered a member of $o(g(n)) \cap \omega(g(n))$, the following equivalence must be true:

$$\begin{aligned}
&f(n) \in \omega(g(n)) \cap o(g(n)) \\
&\Leftrightarrow 0 \leq c_1 g(n) < f(n) \wedge 0 \leq f(n) < c_2 g(n) \\
&\Leftrightarrow 0 \leq c_1 g(n) < f(n) < c_2 g(n)
\end{aligned}$$

for all positive real constants c_1 and c_2, and for all n bigger than some positive constant n_0.
But since $c_1 g(n) < c_2 g(n)$ can't be true for all positive constants c_1 and c_2, $o(g(n)) \cap \omega(g(n))$ must be the empty set.

3.1-8

$$O(g(n,m)) = \{f(n,m) : \exists c, n_0, m_0 > 0 : 0 \leq f(n,m) \leq cg(n,m), \forall n \geq n_0 \vee \forall m \geq m_0\}$$

$$\Omega(g(n,m)) = \{f(n,m) : \exists c, n_0, m_0 > 0 : 0 \leq cg(n,m) \leq f(n,m), \forall n \geq n_0 \vee \forall m \geq m_0\}$$

$$\Theta(g(n,m)) = \{f(n,m) : f(n,m) \in \Omega(g(n,m)) \cap O(g(n,m))\}$$

3.2

3.2-1

Show that if $f(n)$ and $g(n)$ are monotonically increasing functions, then so are the functions $f(n) + g(n)$ and $f(g(n))$, and if $f(n)$ and $g(n)$ are in addition nonnegative, then $f(n) \cdot g(n)$ is monotonically increasing.

$f(n)$ and $g(n)$ being monotonically increasing implies that for $n \leq m$

$$\begin{aligned} & f(n) \leq f(m) \wedge g(n) \leq g(m) \\ & \Leftrightarrow f(n) + g(m) \geq f(n) + g(n) \\ & \Leftrightarrow f(m) + g(m) \geq f(n) + g(n), \end{aligned}$$

likewise it implies that $f(g(n)) \leq f(g(m))$.
If in addition both $f(n)$ and $g(n)$ are nonnegative, the following holds

$$\begin{aligned} & f(n)g(n) = f(n)g(n) \\ & \Leftrightarrow f(n)g(n) \leq f(n)g(m) \\ & \Leftrightarrow f(n)g(n) \leq f(m)g(m) \end{aligned}$$

3.2-2

Proof equation (3.16).

$$a^{\log_b c} = c^{\log_b a} \tag{3.16}$$

$$\begin{aligned} & a^{\log_b c} = c^{\log_b a} \\ & \Leftrightarrow a^{\frac{\log_b c}{\log_b a}} = c \\ & \Leftrightarrow a^{\log_a c} = c \\ & \Leftrightarrow c = c \end{aligned}$$

3.2-3

Prove equation 3.19 $\lg(n!) = \Theta(n \lg n)$. Also prove that $n! = \omega(2^n)$ and $n! = o(n^n)$.

Let us start with equation **3.19** and by showing its membership to $O(n \lg n)$.

Let c be some positive real constant, then

$$\begin{aligned}
&\lg(n!) \in O(n \lg n) \\
&\Leftrightarrow 0 \leq \lg(n!) \leq cn \lg n \\
&\Leftrightarrow \lg(n^n) \leq cn \lg n \\
&\Leftrightarrow n \lg n \leq cn \lg n
\end{aligned}$$

holds for all $n \geq n_0 = 1$ if $c >= 1$.

Now we will need to additionally show that $\lg(n!)$ is also a member of $\Omega(n \lg n)$.

Again let c be some positive real constant, then

$$\begin{aligned}
&\lg(n!) \in \Omega(n \lg n) \\
&\Leftrightarrow 0 \leq cn \lg n \leq \lg(n!)
\end{aligned}$$