Definition 1.4 [Big "oh"] The function $f(n) = O(g(n))$ (read as "f of n is big oh of g of n") iff (if and only if) there exist positive constants c and n_0 such that $f(n) \leq c * g(n)$ for all n, $n \geq n_0$. □

Example 1.14 The function $3n + 2 = O(n)$ as $3n + 2 \leq 4n$ for all $n \geq 2$. $3n + 3 = O(n)$ as $3n + 3 \leq 4n$ for all $n \geq 3$. $100n + 6 = O(n)$ as $100n+6 \leq 101n$ for all $n \geq 6$. $10n^2 + 4n + 2 = O(n^2)$ as $10n^2 + 4n + 2 \leq 11n^2$ for all $n \geq 5$. $1000n^2 + 100n - 6 = O(n^2)$ as $1000n^2 + 100n - 6 \leq 1001n^2$ for $n \geq 100$. $6 * 2^n + n^2 = O(2^n)$ as $6 * 2^n + n^2 \leq 7 * 2^n$ for $n \geq 4$. $3n + 3 = O(n^2)$ as $3n + 3 \leq 3n^2$ for $n \geq 2$. $10n^2 + 4n + 2 = O(n^4)$ as $10n^2 + 4n + 2 \leq 10n^4$ for $n \geq 2$. $3n + 2 \neq O(1)$ as $3n + 2$ is not less than or equal to c for any constant c and all $n \geq n_0$. $10n^2 + 4n + 2 \neq O(n)$. □

We write $O(1)$ to mean a computing time that is a constant. $O(n)$ is called *linear*, $O(n^2)$ is called *quadratic*, $O(n^3)$ is called *cubic*, and $O(2^n)$ is called *exponential*. If an algorithm takes time $O(\log n)$, it is faster, fo sufficiently large n, than if it had taken $O(n)$. Similarly, $O(n \log n)$ is bette than $O(n^2)$ but not as good as $O(n)$. These seven computing times–$O(1)$ $O(\log n)$, $O(n)$, $O(n \log n)$, $O(n^2)$, $O(n^3)$, and $O(2^n)$–are the ones we se most often in this book.

As illustrated by the previous example, the statement $f(n) = O(g(n$ states only that $g(n)$ is an upper bound on the value of $f(n)$ for all $n, n \geq n$ It does not say anything about how good this bound is. Notice that n $O(2^n)$, $n = O(n^{2.5})$, $n = O(n^3)$, $n = O(2^n)$, and so on. For the stateme $f(n) = O(g(n))$ to be informative, $g(n)$ should be as small a function of as one can come up with for which $f(n) = O(g(n))$. So, while we often s that $3n + 3 = O(n)$, we almost never say that $3n + 3 = O(n^2)$, even thou this latter statement is correct.

From the definition of O, it should be clear that $f(n) = O(g(n))$ not the same as $O(g(n)) = f(n)$. In fact, it is meaningless to say tl $O(g(n)) = f(n)$. The use of the symbol $=$ is unfortunate because t symbol commonly denotes the equals relation. Some of the confusion tl results from the use of this symbol (which is standard terminology) can avoided by reading the symbol $=$ as "is" and not as "equals."

Theorem 1.2 obtains a very useful result concerning the order of f (that is, the $g(n)$ in $f(n) = O(g(n))$) when $f(n)$ is a polynomial in n.

Theorem 1.2 If $f(n) = a_m n^m + \cdots + a_1 n + a_0$, then $f(n) = O(n^m)$.

Proof:

Definition 1.5 [Omega] The function $f(n) = \Omega(g(n))$ (read as "f of n is omega of g of n") iff there exist positive constants c and n_0 such that $f(n) \geq c * g(n)$ for all n, $n \geq n_0$. □

Example 1.15 The function $3n + 2 = \Omega(n)$ as $3n + 2 \geq 3n$ for $n \geq 1$ (the inequality holds for $n \geq 0$, but the definition of Ω requires an $n_0 > 0$). $3n+3 = \Omega(n)$ as $3n+3 \geq 3n$ for $n \geq 1$. $100n+6 = \Omega(n)$ as $100n+6 \geq 100n$ for $n \geq 1$. $10n^2 + 4n + 2 = \Omega(n^2)$ as $10n^2 + 4n + 2 \geq n^2$ for $n \geq 1$. $6 * 2^n + n^2 = \Omega(2^n)$ as $6 * 2^n + n^2 \geq 2^n$ for $n \geq 1$. Observe also that $3n + 3 = \Omega(1)$, $10n^2 + 4n + 2 = \Omega(n)$, $10n^2 + 4n + 2 = \Omega(1)$, $6 * 2^n + n^2 = \Omega(n^{100})$, $6 * 2^n + n^2 = \Omega(n^{50.2})$, $6 * 2^n + n^2 = \Omega(n^2)$, $6 * 2^n + n^2 = \Omega(n)$, and $6 * 2^n + n^2 = \Omega(1)$. □

As in the case of the big oh notation, there are several functions $g(n)$ for which $f(n) = \Omega(g(n))$. The function $g(n)$ is only a lower bound on $f(n)$. For the statement $f(n) = \Omega(g(n))$ to be informative, $g(n)$ should be as large a function of n as possible for which the statement $f(n) = \Omega(g(n))$ is true. So, while we say that $3n + 3 = \Omega(n)$ and $6 * 2^n + n^2 = \Omega(2^n)$, we almost never say that $3n + 3 = \Omega(1)$ or $6 * 2^n + n^2 = \Omega(1)$, even though both of these statements are correct.

Theorem 1.3 is the analogue of Theorem 1.2 for the omega notation.

Theorem 1.3 If $f(n) = a_m n^m + \cdots + a_1 n + a_0$ and $a_m > 0$, then $f(n) = \Omega(n^m)$.

Proof: Left as an exercise.

Definition 1.6 [Theta] The function $f(n) = \Theta(g(n))$ (read as "f of n is theta of g of n") iff there exist positive constants c_1, c_2, and n_0 such t $c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all n, $n \geq n_0$.

Example 1.16 The function $3n + 2 = \Theta(n)$ as $3n + 2 \geq 3n$ for all n and $3n + 2 \leq 4n$ for all $n \geq 2$, so $c_1 = 3$, $c_2 = 4$, and $n_0 = 2$. $3n+3 = \Theta$ $10 * \log n + 4 = \Theta(l$ $10n^2 + 4n + 2 = \Theta(n^2)$, $6 * 2^n + n^2 = \Theta(2^n)$, and $10n^2 + 4n + 2 \neq$ $3n + 2 \neq \Theta(1)$, $3n + 3 \neq \Theta(n^2)$, $10n^2 + 4n + 2 \neq \Theta(n)$, $10n^2 + 4n + 2 \neq \Theta(1)$. $6 * 2^n + n^2 \neq \Theta(n^2)$, $6 * 2^n + n^2 \neq \Theta(n^{100})$, and $6 * 2^n + n^2 \neq \Theta(1)$.

The theta notation is more precise than both the the big oh and an notations. The function $f(n) = \Theta(g(n))$ iff $g(n)$ is both an upper an bound on $f(n)$.

Notice that the coefficients in all of the $g(n)$'s used in the pr three examples have been 1. This is in accordance with practice. W ...rselves saying that $3n + 3 = O(3n)$, that $10 = O(1$ ... $2^n + n^2 = O(6 * 2^n)$, or that $6 * 2$ ... is true.

```
Algorithm Transpose(a, n)
{
    for i := 1 to n − 1 do
        for j := i + 1 to n do
        {
            t := a[i, j]; a[i, j] := a[j, i]; a[j, i] := t;
        }
}
```

$O(n^2)$

Algorithm 1.24 Matrix transpose

```
Algorithm Mult(a, b, c, n)
{
    for i := 1 to n do
        for j := 1 to n do
        {
            c[i, j] := 0;
            for k := 1 to n do
                c[i, j] := c[i, j] + a[i, k] * b[k, j];
        }
}
```

$C_0 n$ $C_4 n$ $C_2 n^2$ n^2 $C_3 n$

Algorithm 1.25 Matrix multiplication

```
Algorithm Mult(a, b, c, m, n, p)
{
    for i := 1 to m do
        for j := 1 to p do
        {
            c[i, j] := 0;
            for k := 1 to n do
                c[i, j] := c[i, j] + a[i, k] * b[k, j];
        }
}
```

Algorithm 1.26 Matrix multiplication

10. The McWidget company has been bought out by a computer manufacturer who insists that all displays be in binary. Rework the McWidget example using a binary display.

11. Suppose that a sequence of tasks is performed. The actual complexity of the ith task is 1 when i is not a power of 2. When i is a power of 2, the complexity of the ith task is i. Use each of the methods (a) aggregate, (b) accounting, and (c) potential function to show that the amortized complexity of a task is $O(1)$.

12. Imagine that a data structure is represented as an array whose initial length is 1. The data structure operations are *insert* and *delete*. An *insert* takes 1 time unit except when the number of elements in the data structure prior to the insert equals the array length n; at this time, the insert takes n time units because we double the array length. A *delete* takes 1 time unit except when the number of elements left in the array is less than (array length)/4. When the number of elements left in the array is less than (array length)/4, the array length is halved and the *delete* takes (array length)/2 time units. Use each of the methods (a) aggregate, (b) accounting, and (c) potential function to show that the amortized complexity of each data structure operation is $O(1)$.

13. Show that the following equalities are correct:

(a) $5n^2 - 6n = \Theta(n^2)$

(b) $n! = O(n^n)$

(c) $2n^2 2^n + n \log n = \Theta(n^2 2^n)$

(d) $\sum_{i=0}^{n} i^2 = \Theta(n^3)$

(e) $\sum_{i=0}^{n} i^3 = \Theta(n^4)$.

(f) $n^{2^n} + 6 * 2^n = \Theta(n^{2^n})$

(g) $n^3 + 10^6 n^2 = \Theta(n^3)$

(h) $6n^3/(\log n + 1) = O(n^3)$

(i) $n^{1.001} + n \log n = \Theta(n^{1.001})$

(j) $n^{k+\epsilon} + n^k \log n = \Theta(n^{k+\epsilon})$ for all fixed k and ϵ, $k \geq 0$ and $\epsilon > 0$

(k) $10n^3 + 15n^4 + 100n^2 2^n = O(100n^2 2^n)$

(l) $33n^3 + 4n^2 = \Omega(n^2)$

(m) $33n^3 + 4n^2 = \Omega(n^3)$

14. Show that the following equalities are incorrect:

(a) $10n^2 + 9 = O(n)$