The programmer's primary responsibility is correctness of code. Efficiency of code must always be a secondary concern. Good programmers have a repertoire of algorithms that can be applied to many problems and have an understanding of efficiency.

KEY TERMS

asymptotic dominance (p. 544)
average case analysis (p. 561)
benchmark (p. 577)
best case analysis (p. 560)
big-O notation (p. 545)
binary search (p. 557)
bubble sort (p. 575)
chained hash table (p. 570)
clustering (p. 568)
code tuning (p. 578)
collision (p. 565)
collision resolution strategy (p. 567)
complexity theory (p. 542)
constant time (p. 548)
cost function (p. 543)
cubic time (p. 548)
division technique (p. 566)
dominance (p. 544)
exchange sorts (p. 575)
exponential time (p. 548)
hash function (p. 564)
hash table (p. 565)
hashing (p. 564)
in-place sorts (p. 593)
intellectual efficiency (p. 590)
linear probing (p. 567)
linear search (p. 556)
linear time (p. 548)
load factor (p. 571)
logarithmic time (p. 548)
midsquare technique (p. 566)
N log *N* sort (p. 587)
N^2 sorts (p. 577)
order of a function (p. 545)
perfect hash function (p. 565)
performance analysis (p. 543)
pivot (p. 581)
polynomial time (p. 548)
profiler (p. 579)
quadratic time (p. 548)
quicksort (p. 579)
radix sort (p. 591)
search key (p. 556)
space complexity (p. 543)
space efficiency (p. 542)
straight selection sort (p. 573)
synonyms (p. 565)
time complexity (p. 543)
time efficiency (p. 542)
worst case analysis (p. 560)

EXERCISES

12.1 For each pair of functions *T1* and *T2* below, tell whether *T1* dominates *T2*, *T2* dominates *T1*, or neither dominates the other. (The letter *T* is often used to suggest execution Time of an algorithm.)

a. $T1(X) = 2^*X^2$

$T2(X) = 2^*X$

b. $T1(X) = 5^*X^2$

$T2(X) = 2^*X^3$

c. $T1(Y) = 55^*Y^{15}+3^*Y^4+7500$

$T2(Y) = Y^{16}+2$

d. $T1(Z) = (Z + 3)^*(Z + 5)$

$T2(Z) = 250^*Z$

e. $T1(Z) = (Z + 7)^*(Z + 9)$

$T2(Z) = 2^*Z^3$

f. $T1(N) = 37^*N^2$

$T2(N) = N \log N$

g. $T1(N) = 37*N^2$

$T2(N) = N^2 \log N$

h. $T1(N) = \log_2 N$

$T2(N) = \log_3 N$

i. $T1(N) = 84$

$T2(N) = \log_9 N$

j. $T1(X) = \dfrac{3*X + 2}{X}$

$T2(X) = 766*X$

k. $T1(X) = \dfrac{X^4 + X^2 - 17}{X^3}$

$T2(X) = X^2$

l. $T1(W) = W^9$

$T2(W) = 9^W$

m. $T1(W) = W!$

$T2(W) = 67^W$

n. $T1(V) = V^V$

$T2(V) = 25^V$

o. $T1(X, Y) = 2*X^3*Y$

$T2(X, Y) = 3*X*Y$

p. $T1(X, Y) = X^2+75$

$T2(X, Y) = 7*Y$

q. $T1(V, W) = W^3 + W + 74$

$T2(V, W) = V + 18$

12.2 Using big-O notation, give the order of each function in Exercise 12.1.

12.3 Rank the following big-O measures from greatest to least. (Recall that $O(f) > O(g)$ if and only if f dominates g.)

a. $O(N)$

b. $O(N^3)$

c. $O(4^N)$

d. $O(\log_4 N)$

e. $O(\log_5 N)$

f. $O(N^2)$

g. $O(1)$

h. $O(N \log_3 N)$

i. $O(N^2 \log_3 N)$

12.4 Classify each big-O measure of Exercise 12.3 as one (or more) of the following: constant, linear, quadratic, cubic, logarithmic, polynomial, or exponential.

12.5 Determine the order of each function below, and classify each result as constant, linear, quadratic, cubic, logarithmic, or exponential.

a. $T(N) = 3*N^2 + N$

b. $T(N) = 55*N^3 + 77*N^2 + 99$

c. $T(N) = 2^N*N^2$

d. $T(N) = 7501$

e. $T(N) = \log N + 46*N$

f. $T(N) = 3^{(N+1)}$

g. $T(N) = \dfrac{N \log N}{2 + N}$

h. $T(N) = \dfrac{3*N^4 + 4*N^3}{5*N^2 + N}$

i. $T(N) = \dfrac{17^N}{N^2}$

12.6 Using big-O notation, estimate the running time of each of the following algorithms. You may assume that all variables are of type `int`.

a.
```
for (i = 1; i <= X; i++)
    for (j = 1; j <= X; j++)
        for (k = 1; k <= X; k++) {
            // Five assignment instructions
        }
```

b.
```
for (i = 10; i <= X; i++) {
    // Two assignment instructions
    for (j = 15; j <= X; j++) {
        for (k = 1; k <= X; k++) {
            // Five assignment instructions
        }
        // Seven assignment instructions
    }
}
```

c.
```
for (i = 1; i <= X; i++) {
    for (j = 1; j <= X; j++) {
        // Twenty assignment instructions
    }
    for (j = 1; j <= X; j++)
        if (j % 2 == 1)
            for (k = 1; k <= X; k++) {
                // Five assignment instructions
            }
}
```

d.
```
for (i = 1; i <= X; i++) {
    for (j = i; j <= X; j++) {
        // Six assignment instructions
    }
    if (i % 2 == 1) {
        // Four assignment instructions
    }
}
```

e.
```
for (i = 1; i <= X; i++)
    for (j = 1; j <= Y; j++)
        for (k = 1; k <= X; k++) {
            // Two assignment instructions
        }
```

f.
```
i = 1;
while (i <= X) {
    // Three assignment instructions
    j = 17;
    while (j <= 100) {
        // Two assignment instructions
        j++;
    }
    i++;
}
```

g.
```
i = X;
do {
    // Three assignment instructions
    j = 1;
    while (j <= X) {
        // Two assignment instructions
        j++;
    }
    i--;
} while (i >= 1);
```

h.
```
i = 1;
while (i <= X) {
    // Three assignment instructions
    j = 1;
    while (j <= X) {
        // Two assignment instructions
        j = j * 2;
    }
    i++;
}
```

i.
```
i = X;
while (i >= 1) {
    // Three assignment instructions
    j = 1;
    while (j <= X) {
        // Two assignment instructions
        j = j + 2;
    }
    i = i / 3;
}
```

12.7 Using big-O notation, estimate the running time of each of the following recursive functions.

a.
```
void RecA( int X )
{
    // Some task requiring constant time
    if (X > 0)
        RecA(X-1);
}
```

b.
```
void RecB( int X )
{
    int i;

    for (i = 1; i <= X; i++) {
        // Some task requiring constant time
    }
    if (X > 1)
        RecB(X-1);
}
```

c.
```
void RecC( int X )
{
    int i;

    for (i = 1; i <= X; i++) {
        // Some task requiring constant time
    }
    if (X > 1)
        RecC(X/2);
}
```

d.
```
void RecD( int X )
{
    int i;

    // Some task requiring constant time
    for (i = 1; i <= X; i++)
        if (X > 1)
            RecD(X-1);
}
```