1. Explain the meaning of the following expressions:

a. $f(n)$ is $O(1)$.

b. $f(n)$ is $\Theta(1)$.

c. $f(n)$ is $n^{O(1)}$.

2. Assuming that $f_1(n)$ is $O(g_1(n))$ and $f_2(n)$ is $O(g_2(n))$, prove the following statements:

a. $f_1(n) + f_2(n)$ is $O(\max(g_1(n), g_2(n)))$.

b. If a number k can be determined such that for all $n > k$, $g_1(n) \leq g_2(n)$, then $O(g_1(n)) + O(g_2(n))$ is $O(g_2(n))$.

c. $f_1(n) * f_2(n)$ is $O(g_1(n) * g_2(n))$ (rule of product).

d. $O(cg(n))$ is $O(g(n))$.

e. c is $O(1)$.

3. Prove the following statements:

a. $\sum_{i=1}^{n} i^2$ is $O(n^3)$ and more generally, $\sum_{i=1}^{n} i^k$ is $O(n^{k+1})$.

b. $an^k/\lg n$ is $O(n^k)$ but $an^k/\lg n$ is not $\Theta(n^k)$.

c. $n^{1.1} + n \lg n$ is $\Theta(n^{1.1})$.

d. 2^n is $O(n!)$ and $n!$ is not $O(2^n)$.

e. 2^{n+a} is $O(2^n)$.

f. 2^{2n+a} is not $O(2^n)$.

g. $2^{\sqrt{\lg n}}$ is $O(n^a)$.

4. Make the same assumptions as in Exercise 2 and, by finding counterexamples, refute the following statements:

a. $f_1(n) - f_2(n)$ is $O(g_1(n) - g_2(n))$.

b. $f_1(n)/f_2(n)$ is $O(g_1(n)/g_2(n))$.

5. Find functions f_1 and f_2 such that both $f_1(n)$ and $f_2(n)$ are $O(g(n))$, but $f_1(n)$ is not $O(f_2)$.

6. Is it true that

a. if $f(n)$ is $\Theta(g(n))$, then $2^{f(n)}$ is $\Theta(2^{g(n)})$?

b. $f(n) + g(n)$ is $\Theta(\min(f(n), g(n)))$?

c. 2^{na} is $O(2^n)$?

7. The algorithm presented in this chapter for finding the length of the longest subarray with the numbers in increasing order is inefficient, because there is no need to continue to search for another array if the length already found is greater than the length of the subarray to be analyzed. Thus, if the entire array is already in order, we can discontinue the search right away, converting the worst case into the best. The change needed is in the outer loop, which now has one more test:

```
for (i = 0, length = 1; i < n-1 && length < n==i; i++)
```

What is the worst case now? Is the efficiency of the worst case still $O(n^2)$?

8. Find the complexity of the function used to find the *k*th smallest integer in an unordered array of integers

```
int selectkth(int a[], int k, int n) {
    int i, j, mini, tmp;
    for (i = 0; i < k; i++) {
        mini = i;
        for (j = i+1; j < n; j++)
            if (a[j]<a[mini])
                mini = j;
        tmp = a[i];
        a[i] = a[mini];
        a[mini] = tmp;
    }
    return a[k-1];
}
```

9. Determine the complexity of the following implementations of the algorithms for adding, multiplying, and transposing $n \times n$ matrices:

```
for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
        a[i][j] = b[i][j] + c[i][j];

for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
        for (k = a[i][j] = 0; k < n; k++)
            a[i][j] += b[i][k] * c[k][j];

for (i = 0; i < n - 1; i++)
    for (j = i+1; j < n; j++) {
        tmp = a[i][j];
        a[i][j] = a[j][i];
        a[j][i] = tmp;
    }
```

10. Find the computational complexity for the following four loops:

a.
```
for (cnt1 = 0, i = 1; i <= n; i++)
    for (j = 1; j <= n; j++)
        cnt1++;
```

b.
```
for (cnt2 = 0, i = 1; i <= n; i++)
    for (j = 1; j <= i; j++)
        cnt2++;
```

c.
```
for (cnt3 = 0, i = 1; i <= n; i *= 2)
    for (j = 1; j <= n; j++)
        cnt3++;
```

d.
```
for (cnt4 = 0, i = 1; i <= n; i *= 2)
    for (j = 1; j <= i; j++)
        cnt4++;
```

11. Find the average case complexity of sequential search in an array if the probability of accessing the last cell equals $\frac{1}{2}$, the probability of the next to last cell equals $\frac{1}{4}$, and the probability of locating a number in any of the remaining cells is the same and equal to $\frac{1}{4(n-2)}$.

12. Consider a process of incrementing a binary n-bit counter. An increment causes some bits to be flipped: Some 0s are changed to 1s, and some 1s to 0s. In the best case, counting involves only one bit switch; for example, when 000 is changed to 001, sometimes all the bits are changed, as when incrementing 011 to 100.

Number	Flipped Bits
000	
001	1
010	2
011	1
100	3
101	1
110	2
111	1

Using worst case assessment, we may conclude that the cost of executing $m = 2^n - 1$ increments is $O(mn)$. Use amortized analysis to show that the cost of executing m increments is $O(m)$.

13. How can you convert a satisfiability problem into a three-satisfiability problem for an instance when an alternative in a Boolean expression has two variables? One variable?

6. Reorder the following efficiencies from smallest to largest:

a. 2^n

b. $n!$

c. n^5

d. 10,000

e. $n\log_2(n)$

7. Reorder the following efficiencies from smallest to largest:

a. $n\log_2(n)$

b. $n + n^2 + n^3$

c. 2^4

d. $n^{0.5}$

8. Determine the big-O notation for the following:

a. $5n^{5/2} + n^{2/5}$

b. $6\log_2(n) + 9n$

c. $3n^4 + n\log_2(n)$

d. $5n^2 + n^{3/2}$

9. Calculate the run-time efficiency of the following program segment:

```
1   i = 1
2   loop (i <= n)
     1   print (i)
     2   i = i + 1
3   end loop
```

10. Calculate the run-time efficiency of the following program segment:

```
1  i = 1
2  loop (i <= n)
    1  j = 1
    2  loop (j <= n)
        1  k = 1
        2  loop (k <= n)
            1  print (i, j, k)
            2  k = k + 1
        3  end loop
        4  j = j + 1
    3  end loop
    4  i = i + 1
3  end loop
```

11. If the algorithm `doIt` has an efficiency factor of $5n$, calculate the run-time efficiency of the following program segment:

```
1   i = 1
2   loop i <= n
      1   doIt (...)
      2   i = i + 1
3   end loop
```

12. If the efficiency of the algorithm `doIt` can be expressed as $O(n) = n^2$, calculate the efficiency of the following program segment:

```
1  i = 1
2  loop (i <= n)
    1  j = 1
    2  loop (j < n)
        1  doIt (...)
        2  j = j + 1
    3  end loop
    4  i = i + 1
3  end loop
```

13. If the efficiency of the algorithm `doIt` can be expressed as $O(n) = n^2$, calculate the efficiency of the following program segment:

```
1   i = 1
2   loop (i < n)
     1   doIt (...)
     2   i = i * 2
3   end loop
```

14. Given that the efficiency of an algorithm is $5n^2$, if a step in this algorithm takes 1 nanosecond (10^{-9}), how long does it take the algorithm to process an input of size 1000?

15. Given that the efficiency of an algorithm is n^3, if a step in this algorithm takes 1 nanosecond (10^{-9}), how long does it take the algorithm to process an input of size 1000?

16. Given that the efficiency of an algorithm is $5n\log_2(n)$, if a step in this algorithm takes 1 nanosecond (10^{-9}), how long does it take the algorithm to process an input of size 1000?

17. An algorithm runs a given input of size n. If n is 4096, the run time is 512 milliseconds. If n is 16,384, the run time is 2048 milliseconds. What is the efficiency? What is the big-O notation?

18. An algorithm runs a given input of size n. If n is 4096, the run time is 512 milliseconds. If n is 16,384, the run time is 8192 milliseconds. What is the efficiency? What is the big-O notation?

19. An algorithm runs a given input of size n. If n is 4096, the run time is 512 milliseconds. If n is 16,384, the run time is 1024 milliseconds. What is the efficiency? What is the big-O notation?

20. Three students wrote algorithms for the same problem. They tested the three algorithms with two sets of data as shown below:

a. Case 1: $n = 10$
run time for student 1: 1
run time for student 2: 1/100
run time for student 3: 1/1000

b. Case 2: $n = 100$
run time for student 1: 10
run time for student 2: 1
run time for student 3: 1

What is the efficiency for each algorithm? Which is the best? Which is the worst? What is the minimum number of test cases (n) in which the best algorithm has the best run time?

8. Find the complexity of the function used to find the *k*th smallest integer in an unordered array of integers

```
int selectkth(int a[], int k, int n) {
    int i, j, mini, tmp;
    for (i = 0; i < k; i++) {
        mini = i;
        for (j = i+1; j < n; j++)
            if (a[j]<a[mini])
                mini = j;
        tmp = a[i];
        a[i] = a[mini];
        a[mini] = tmp;
    }
    return a[k-1];
}
```

9. Determine the complexity of the following implementations of the algorithms for adding, multiplying, and transposing $n \times n$ matrices:

```
for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
        a[i][j] = b[i][j] + c[i][j];

for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
        for (k = a[i][j] = 0; k < n; k++)
            a[i][j] += b[i][k] * c[k][j];

for (i = 0; i < n - 1; i++)
    for (j = i+1; j < n; j++) {
        tmp = a[i][j];
        a[i][j] = a[j][i];
        a[j][i] = tmp;
    }
```

10. Find the computational complexity for the following four loops:

a.
```
for (cnt1 = 0, i = 1; i <= n; i++)
    for (j = 1; j <= n; j++)
        cnt1++;
```

b.
```
for (cnt2 = 0, i = 1; i <= n; i++)
    for (j = 1; j <= i; j++)
        cnt2++;
```

c.
```
for (cnt3 = 0, i = 1; i <= n; i *= 2)
    for (j = 1; j <= n; j++)
        cnt3++;
```

d.
```
for (cnt4 = 0, i = 1; i <= n; i *= 2)
    for (j = 1; j <= i; j++)
        cnt4++;
```