5.10. An algorithm takes 0.5 ms for input size 100. How long will it take for input size 500 if the running time is the following (assume low-order terms are negligible):
a. linear
b. $O(N \log N)$
c. quadratic
d. cubic

5.11. An algorithm takes 0.5 ms for input size 100. How large a problem can be solved in 1 min if the running time is the following (assume low-order terms are negligible):
a. linear
b. $O(N \log N)$
c. quadratic
d. cubic

5.12. Complete the table in Figure 5.10 with estimates for the running times that were too long to simulate. Interpolate the running times for all four algorithms and estimate the time required to compute the maximum contiguous subsequence sum of one million numbers. What assumptions have you made?

5.13. Order the following functions by growth rate: N, $\sqrt{N}$, $N^{1.5}$, N^2, $N \log N$, $N \log \log N$, $N \log^2 N$, $N \log(N^2)$, $2/N$, 2^N, $2^{N/2}$, 37, N^3, and $N^2 \log N$. Indicate which functions grow at the same rate.

5.14. For each of the following six program fragments, do the following:
a. Give a Big-Oh analysis of the running time.
b. Implement the code and run for several values of N.
c. Compare your analysis with the actual running times.

```
// Fragment #1
for( int i = 0; i < n; i++ )
    sum++;

// Fragment #2
for( int i = 0; i < n; i++ )
    for( int j = 0; j < n; j++ )
        sum++;

// Fragment #3
for( int i = 0; i < n; i++ )
    sum++;
for( int j = 0; j < n; j++ )
    sum++;

// Fragment #4
for( int i = 0; i < n; i++ )
    for( int j = 0; j < n * n; j++ )
        sum++;
```