Guidelines for Asymptotic Analysis

1) **Loops:** The running time of the loop is, at most, the running time of the statements inside the loop (including tests) multiplied by the number of iterations.

```
for (int i = 1, m = 0; i <= n; i++) { // Execute n times
    m = m + 2;                        // Constant time, c
}
```

Total time: $c \times n = cn = \theta(n)$

2) **Nested loops:** Analyze from the inside out. The total running time is the product of the sizes of all the loops.

```
for (int i = 1, k = 0; i <= n; i++) { // Execute n times
    for (int j = 1; j <= n; j++) {    // Execute n times
        k = k + 2;                    // Constant time, c
    }
}
```

Total time: $c \times n \times n = cn^2 = \theta(n^2)$

3) **Consecutive statements:** Add the time complexities of each statement.

```
int m = 0;                            // Constant time, c0
for (int i = 1; i <= n; i++) {        // Execute n times
    m = m + 2;                        // Constant time, c1
}
int k = 0;                            // Constant time, c0
for (int i = 1; i <= n; i++) {        // Execute n times
    for (int j = 1; j <= n; j++) {    // Execute n times
        k = k + 1;                    // Constant time, c1
        cout << k;                    // Constant time, c2
    }
}
```

Total time: $2c_0 + c_1 n + (c_1 + c_2)n^2 = \theta(n^2)$

4) **If-then-else statements:** Choose the worst-case running time: the test, plus either the `then` or `else` part, whichever is larger.

```
if (n == 0) {                         // Constant time, c0
    return false;                     // Constant time, c1
} else {
    for (int i = 0; i < n; i++) {     // Execute n times
        if (list[i] != list2[i]) {    // Constant time, c2
            return false;             // Constant time, c1
        }
    }
    return true;                      // Constant time, c1
}
```

Total time: $c_0 + c_1 + c_2 n = O(n)$

Notice O instead of θ, since the loop might terminate prior to examining all n elements.

5) **Logarithmic complexity:** An algorithm is logarithmic it takes constant time to cut the problem size by a fraction (usually by ½).

```
for (int i = 1, m = 1; i <= n; i = i * 2) {
    m = m + 2;
}
```

Total time: $\theta(\lg n)$

```
for (int i = 1, m = 1; i <= n; i = i * 3) {
    m = m + 2;
}
```

Total time: $\theta(\log_3 n)$