Chapter 2: Fundamentals of the Analysis of Algorithm Efficiency
2.3. Mathematical Analysis of Nonrecursive Algorithms

The core of the algorithm analysis: to find out how the number of the basic operations depends on the size of the input.

There are four rules to count the operations:

Rule 1: for loops - the size of the loop times the running time of the body

The running time of a for loop is at most the running time of the statements inside the loop times the number of iterations.

\[ \text{for} \ (i = 0; \ i < n; \ i++) \]
\[ \quad \text{sum} = \text{sum} + i; \]

a. Find the running time of statements when executed only once:

The statements in the loop heading have fixed number of operations, hence they have constant running time \( O(1) \) when executed only once.

The statement in the loop body has fixed number of operations, hence it has a constant running time when executed only once.

b. Find how many times each statement is executed.

\[ \text{for} \ (i = 0; \ i < n; \ i++) \] // \( i = 0; \) executed only once: \( O(1) \)
\[ \quad \text{//} \ i < n; \ n + 1 \text{ times} \quad O(n) \]
\[ \quad \text{//} \ i++; \ n \text{ times} \quad O(n) \]
\[ \quad \text{// total time of the loop heading:} \]
\[ \quad \text{//} \ O(1) + O(n) + O(n) = \quad O(n) \]
\[ \text{sum} = \text{sum} + i; \] // executed \( n \) times, \( O(n) \)

The loop heading plus the loop body will give: \( O(n) + O(n) = O(n) \).

Loop running time is: \( O(n) \)

Mathematical analysis of how many times the statements in the body are executed

\[
C(n) = \sum_{i=0}^{n-1} 1 = (n-1) - 0 + 1 = n
\]
If
a. the size of the loop is $n$
   (loop variable runs from 0, or some fixed constant, to $n$) and
b. the body has constant running time (no nested loops)

then the time is $O(n)$

Rule 2: Nested loops – the product of the size of the loops times the running time of the body

The total running time is the running time of the inside statements times the product of the sizes of all the loops

```plaintext
sum = 0;
for(i = 0; i < n; i++)
    for(j = 0; j < n; j++)
        sum++;
```

Applying Rule 1 for the nested loop (the ‘j’ loop) we get $O(n)$ for the body of the outer loop.

The outer loop runs $n$ times, therefore the total time for the nested loops will be $O(n) * O(n) = O(n^2)$

Mathematical analysis:

Inner loop:

$$S(i) = \sum_{j=0}^{n-1} 1 = (n-1) - 0 + 1 = n$$

Outer loop:

$$C(n) = \sum_{i=0}^{n-1} S(i) = \sum_{i=0}^{n-1} n = n \times \sum_{i=0}^{n-1} 1 = n(n-1) - 0 + 1 = n^2$$
What happens if the inner loop does not start from 0?

```java
sum = 0;
for( i = 0; i < n; i++)
  for( j = i; j < n; j++)
    sum++;
```

Here, the number of the times the inner loop is executed depends on the value of `i`

- `i = 0`, inner loop runs `n` times
- `i = 1`, inner loop runs `(n-1)` times
- `i = 2`, inner loop runs `(n-2)` times
  ...
- `i = n – 2`, inner loop runs `2` times
- `i = n – 1`, inner loop runs once.

Thus we get: \( 1 + 2 + \ldots + n \) = \( n(n+1)/2 = O(n^2) \)

**General rule for nested loops:**

Running time is the product of the size of the loops times the running time of the body.

**Example:**

```java
sum = 0;
for( i = 0; i < n; i++)
  for( j = 0; j < 2n; j++)
    sum++;
```

We have one operation inside the loops, and the product of the sizes is \( 2n^2 \)

Hence the running time is \( O(2n^2) = O(n^2) \)

**Note:** if the body contains a function call, its running time has to be taken into consideration.

```java
sum = 0;
for( i = 0; i < n; i++)
  for( j = 0; j < n; j++)
    sum = sum + function(sum);
```

Assume that the running time of function(sum) is known to be \( \log(n) \).

Then the total running time will be \( O(n^2*\log(n)) \)
Loops with logarithmic running time - loop variable changes by a factor

Example:

```plaintext
sum = 0;
for (k = 1; k < n; k = k*2)
    sum = sum + k;
```

This loop will run $O(\log(n))$ times because the loop variable will reach the upper limit of the loop in $O(\log(n))$ steps.

**Rule 3: Consecutive program fragments**

The total running time is the maximum of the running time of the individual fragments

```plaintext
sum = 0;
for( i = 0; i < n; i++)
    sum = sum + i;
sum = 0;
for( i = 0; i < n; i++)
    for( j = 0; j < 2*n; j++)
        sum++;
```

The first loop runs in $O(n)$ time, the second - $O(n^2)$ time, the maximum is $O(n^2)$

**Rule 4: If statement**

```plaintext
if C
    S1;
else
    S2;
```

The running time is the maximum of the running times of S1 and S2.

**Summary**

Steps in analysis of nonrecursive algorithms:

- Decide on parameter $n$ indicating *input size*
- Identify algorithm’s *basic operation*
- Check whether the number of time the basic operation is executed depends on some additional property of the input. If so, determine *worst*, *average*, and *best* case for input of size $n$
- Count the number of operations using the rules above.
Exercise

a. 
```java
sum = 0;
for (i = 0; i < n; i++)
    for (j = 0; j < n * n; j++)
        sum++;
```

b. 
```java
sum = 0;
for (i = 0; i < n; i++)
    for (j = 0; j < i; j++)
        sum++;
```

c. 
```java
sum = 0;
for (i = 0; i < n; i++)
    for (j = 0; j < i*i; j++)
        for (k = 0; k < j; k++)
            sum++;
```

d. 
```java
sum = 0;
for (i = 0; i < n; i++)
    sum++;

val = 1;
for (j = 0; j < n*n; j++)
    val = val * j;
```

e. 
```java
sum = 0;
for (i = 0; i < n; i++)
    sum++;
for (j = 0; j < n*n; j++)
    compute_val(sum,j);
```

The complexity of the function `compute_val(x,y)` is given to be $O(n \log n)$

Solutions:

(a) $O(n^3)$
(b) $O(n^2)$
(c) $O(n^5)$
(d) $O(n^2)$
(e) $O(n^3 \log(n))$