Summary of Big-Oh

- The number of operations for an algorithm that processes input of size N can be represented as a
  function of N: \( f(N) \).
- The complexity of the algorithm depends on the type of \( f(N) \).
- In order to compare algorithms and to choose the best one among several algorithms we need to
  be able to compare functions.

Any two growing functions \( f(N) \) and \( g(N) \) can be in one of the three possible relations:

1. \( f(n) = \Theta(g(n)) \)
   - \( f(n) \) and \( g(n) \) have the same rate of growing,
   - \( \lim_{n \to \infty} \frac{f(n)}{g(n)} = c, \ 0 < c < \infty \)

2. \( f(n) = o(g(n)) \)
   - \( f(n) \) grows slower than \( g(n) \),
   - \( \lim_{n \to \infty} \frac{f(n)}{g(n)} = 0 \)

3. \( f(n) = \omega(g(n)) \)
   - \( f(n) \) grows faster than \( g(n) \),
   - \( \lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty \)

The first two relations are combined in the Big-Oh notation:

\( f(N) = O(g(N)) \): \( f(N) \) grows with the same rate or slower than \( g(N) \)

Rules to manipulate Big-Oh expressions:

Let \( T_1(N) \) and \( T_2(N) \) be run times, \( T_1(N) = O(f(N)), \ T_2(N) = O(g(N)) \)

A. \( T_1(N) + T_2(N) = \max(O(f(N)), O(g(N))) \)

B. If \( T(N) \) is a polynomial of degree \( k \), then \( T(N) = \Theta(N^k) = O(N^k) \).

C. \( \log^k N = O(N) \) for any constant \( k \).
Counting the number of operations

A. for loops

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

```plaintext
sum = 0;
for( i = 0; i < n; i++) sum = sum + i; // The running time is \(O(n)\)
```

B. Nested loops

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++;
// The running time is \(O(n^2)\)
```

Note: Loops that run in Logarithmic time

```plaintext
for( i = n; i > 1; i = i/2 ) {......}
for( i = 1; i < n; i = i*2 ) {......}
```

Running time: \(O( [\text{running time of the body}] \times \log(N) )\)

C. 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 < 2n; j++)
    sum++;
```

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

D: If statement

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

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

Things to watch for:

1. Is there a function call whose running time depends on N?
   If yes - its running time has to be taken into account

2. The number of iterations depends on how the loop variable changes - in a linear way or by a factor of 2 (3, 4, etc), or modified inside the loop.