Order Notation

O-notation:

$f(n) \in O(g(n))$ if **there exist** constants $c > 0$ and $n_0 > 0$ such that $0 \leq f(n) \leq cg(n)$ for all $n \geq n_0$.

Here the complexity of f is **not higher** than the complexity of g.

Ω-notation:

$f(n) \in \Omega(g(n))$ if **there exist** constants $c > 0$ and $n_0 > 0$ such that $0 \leq cg(n) \leq f(n)$ for all $n \geq n_0$.

Here the complexity of f is **not lower** than the complexity of g.

Θ-notation:

$f(n) \in \Theta(g(n))$ if **there exist** constants $c_1, c_2 > 0$ and $n_0 > 0$ such that $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all $n \geq n_0$.

Here f and g have the **same complexity**.

Order Notation (cont.)

o-notation:

$f(n) \in o(g(n))$ if **for all** constants $c > 0$, there exists a constant $n_0 > 0$ such that $0 \le f(n) \le cg(n)$ for all $n \ge n_0$.

Here f has **lower complexity** than g.

ω-notation:

$f(n) \in \omega(g(n))$ if **for all** constants $c > 0$, there exists a constant $n_0 > 0$ such that $0 \le cg(n) \le f(n)$ for all $n \ge n_0$.

Here f has **higher complexity** than g.

Exercises

1. Let $f(n) = n^2 - 7n - 30$. Prove from first principles that $f(n) \in O(n^2)$.
2. Let $f(n) = n^2 - 7n - 30$. Prove from first principles that $f(n) \in \Omega(n^2)$.
3. Suppose $f(n) = n^2 + n$. Prove from first principles that $f(n) \notin O(n)$.

Techniques for Order Notation

Suppose that $f(n) > 0$ and $g(n) > 0$ for all $n \geq n_0$. Suppose that

$$L = \lim_{n \to \infty} \frac{f(n)}{g(n)}.$$

Then

$$f(n) \in \begin{cases} o(g(n)) & \text{if } L = 0 \\ \Theta(g(n)) & \text{if } 0 < L < \infty \\ \omega(g(n)) & \text{if } L = \infty. \end{cases}$$

Exercises Using the Limit Method

1. Compare the growth rate of the functions $(\ln n)^2$ and $n^{1/2}$.

2. Use the limit method to compare the growth rate of the functions n^2 and $n^2 - 7n - 30$.

Additional Exercises

1. Compare the growth rate of the functions $(3 + (-1)^n)n$ and n.

2. Compare the growth rates of the functions $f(n) = n\,|\sin \pi n/2| + 1$ and $g(n) = \sqrt{n}$.

Relationships between Order Notations

$$f(n) \in \Theta(g(n)) \Leftrightarrow g(n) \in \Theta(f(n))$$

$$f(n) \in O(g(n)) \Leftrightarrow g(n) \in \Omega(f(n))$$

$$f(n) \in o(g(n)) \Leftrightarrow g(n) \in \omega(f(n))$$

$$f(n) \in \Theta(g(n)) \Leftrightarrow f(n) \in O(g(n)) \text{ and } f(n) \in \Omega(g(n))$$

$$f(n) \in o(g(n)) \Rightarrow f(n) \in O(g(n))$$

$$f(n) \in \omega(g(n)) \Rightarrow f(n) \in \Omega(g(n))$$

Algebra of Order Notations

"Maximum" rules: Suppose that $f(n) > 0$ and $g(n) > 0$ for all $n \geq n_0$. Then:

$$O(f(n) + g(n)) = O(\max\{f(n), g(n)\})$$

$$\Theta(f(n) + g(n)) = \Theta(\max\{f(n), g(n)\})$$

$$\Omega(f(n) + g(n)) = \Omega(\max\{f(n), g(n)\})$$

"Summation" rules: Supose I is a finite set. Then

$$O\left(\sum_{i \in I} f(i)\right) = \sum_{i \in I} O(f(i))$$

$$\Theta\left(\sum_{i \in I} f(i)\right) = \sum_{i \in I} \Theta(f(i))$$

$$\Omega\left(\sum_{i \in I} f(i)\right) = \sum_{i \in I} \Omega(f(i))$$

Some Common Growth Rates (in increasing order)

polynomial	$\Theta(1)$
	$\Theta(\log n)$
	$\Theta(\sqrt{n})$
	$\Theta(n)$
	$\Theta(n^2)$
	$\Theta(n^c)$
	$\Theta\left(n^{\sqrt{n}\log_2 n}\right)$ (graph isomorphism)
	$\Theta\left(e^{c(\log n)^{1/3}(\log\log n)^{2/3}}\right)$ (number field sieve)
exponential	$\Theta(1.1^n)$
	$\Theta(2^n)$
	$\Theta(e^n)$
	$\Theta(n!)$
	$\Theta(n^n)$

Sequences

Arithmetic sequence:

$$\sum_{i=0}^{n-1}(a+di) = na + \frac{dn(n-1)}{2} \in \Theta(n^2).$$

Geometric sequence:

$$\sum_{i=0}^{n-1} ar^i = \begin{cases} a\frac{r^n-1}{r-1} \in \Theta(r^n) & \text{if } r > 1 \\ na \in \Theta(n) & \text{if } r = 1 \\ a\frac{1-r^n}{1-r} \in \Theta(1) & \text{if } 0 < r < 1. \end{cases}$$

Sequences (cont.)

Arithmetic-geometric sequence:

$$\sum_{i=0}^{n-1}(a+di)r^i = \frac{a}{1-r} - \frac{(a+(n-1)d)r^n}{1-r} + \frac{dr(1-r^{n-1})}{(1-r)^2}$$

provided that $r \neq 1$.

Harmonic sequence:

$$H_n = \sum_{i=1}^{n} \frac{1}{i} \in \Theta(\log n)$$

More precisely, it is possible to prove that

$$\lim_{n\to\infty} (H_n - \ln n) = \gamma,$$

where $\gamma \approx 0.57721$ is **Euler's constant**.

Logarithm Formulae

1. $\log_b xy = \log_b x + \log_b y$
2. $\log_b x/y = \log_b x - \log_b y$
3. $\log_b 1/x = -\log_b x$
4. $\log_b x^y = y \log_b x$
5. $\log_b a = \frac{1}{\log_a b}$
6. $\log_b a = \frac{\log_c a}{\log_c b}$
7. $a^{\log_b c} = c^{\log_b a}$

Miscellaneous Formulae

$n! \in \Theta\left(n^{n+1/2}e^{-n}\right)$

$\log n! \in \Theta(n \log n)$

Another useful formula is

$$\sum_{i=1}^{\infty} \frac{1}{i^2} = \frac{\pi^2}{6},$$

which implies that

$$\sum_{i=1}^{n} \frac{1}{i^2} \in \Theta(1).$$

A sum of powers of integers when $c \geq 1$:

$$\sum_{i=1}^{n} i^c \in \Theta(n^{c+1}).$$

Two General Strategies for Loop Analysis

Sometimes a O-bound is sufficient. However, we often want a precise Θ-bound. Two general strategies are as follows:

- Use Θ-bounds **throughout the analysis** and thereby obtain a Θ-bound for the complexity of the algorithm.
- Prove a O-bound and a **matching** Ω-bound **separately** to get a Θ-bound. Sometimes this technique is easier because arguments for O-bounds may use simpler upper bounds (and arguments for Ω-bounds may use simpler lower bounds) than arguments for Θ-bounds do.

Techniques for Loop Analysis

Identify **elementary operations** that require constant time (denoted $\Theta(1)$ time).

The complexity of a loop is expressed as the **sum** of the complexities of each iteration of the loop.

Analyze independent loops **separately**, and then **add** the results: use "maximum rules" and simplify whenever possible.

If loops are nested, start with the **innermost loop** and proceed outwards. In general, this kind of analysis requires evaluation of **nested summations**.

Elementary Operations in the Unit Cost Model

For now, we will work in the **unit cost model**, where we assume that arithmetic operations such as $+$, $-$, $\times$ and integer division take time $\Theta(1)$.

This is a reasonable assumption for integers of bounded size (e.g., integers that fit into one work of memory).

If we want to consider the complexity of arithmetic operation on integers of arbitrary size, we need to consider **bit complexity**, where we express the complexity as a function of the length of the integers (as measured in bits).

We will see some examples later, such as **multiprecision multiplication**.

Example of Loop Analysis

Algorithm: *LoopAnalysis1* $(n : integer)$

(1) $sum \leftarrow 0$

(2) **for** $i \leftarrow 1$ **to** n

do $\begin{cases} \textbf{for } j \leftarrow 1 \textbf{ to } i \\ \textbf{do } \begin{cases} sum \leftarrow sum + (i-j)^2 \\ sum \leftarrow \lfloor sum/i \rfloor \end{cases} \end{cases}$

(3) **return** (sum)

Θ-bound analysis

(1)	$\Theta(1)$
(2)	Complexity of inner **for** loop: $\Theta(i)$
	Complexity of outer **for** loop: $\sum_{i=1}^{n} \Theta(i) = \Theta(n^2)$
(3)	$\Theta(1)$
total	$\Theta(1) + \Theta(n^2) + \Theta(1) = \Theta(n^2)$

Example of Loop Analysis (cont.)

Proving separate O- and Ω-bounds

We focus on the two nested **for** loops (i.e., (2)).

The total number of iterations is $\sum_{i=1}^{n} i$, with $\Theta(1)$ time per iteration.

Upper bound:

$$\sum_{i=1}^{n} O(i) \leq \sum_{i=1}^{n} O(n) = O(n^2).$$

Lower bound:

$$\sum_{i=1}^{n} \Omega(i) \geq \sum_{i=n/2}^{n} \Omega(i) \geq \sum_{i=n/2}^{n} \Omega(n/2) = \Omega(n^2/4) = \Omega(n^2).$$

Since the upper and lower bounds **match**, the complexity is $\Theta(n^2)$.

Another Example of Loop Analysis

Algorithm: *LoopAnalysis2*$(A : array; n : integer)$

$max \leftarrow 0$
for $i \leftarrow 1$ **to** n
  do {
    for $j \leftarrow i$ **to** n
      do {
        $sum \leftarrow 0$
        for $k \leftarrow i$ **to** j
          do {
            $sum \leftarrow sum + A[k]$
            if $sum > max$
              then $max \leftarrow sum$
return (max)

Another Example of Loop Analysis (cont.)

Θ-bound analysis The innermost loop (**for** k) has complexity $\Theta(j-i+1)$. The next loop (**for** j) has complexity

$$\begin{aligned}\sum_{j=i}^{n}\Theta(j-i+1) &= \Theta\left(\sum_{j=i}^{n}(j-i+1)\right)\\ &= \Theta\left(1+2+\cdots+(n-i+1)\right)\\ &= \Theta\left((n-i+1)(n-i+2)\right).\end{aligned}$$

The outer loop (**for** i) has complexity

$$\begin{aligned}\sum_{i=1}^{n}\Theta((n-i+1)(n-i+2)) &= \Theta\left(\sum_{i=1}^{n}(n-i+1)(n-i+2)\right)\\ &= \Theta\left(1\times 2+2\times 3+\cdots+n(n+1)\right)\\ &= \Theta\left(n^3/3+n^2+2n/3\right) \quad \text{from Maple}\\ &= \Theta(n^3).\end{aligned}$$

Another Example of Loop Analysis (cont.)

Proving an Ω-bound

Consider two loop structures:

L_1	L_2
$i = 1, \ldots, n/3$	$i = 1, \ldots, n$
$j = 1 + 2n/3, \ldots, n$	$j = i + 1 \ldots, n$
$k = 1 + n/3, \ldots, 1 + 2n/3$	$k = i \ldots, j$

It is easy to see that $L_1 \subset L_2$. L_2 is loop structure of the given algorithm.

There are $(n/3)^3 = n^3/27$ iterations in L_1 and n^3 iterations in L_3.

Therefore the number of iterations in L_2 is $\Omega(n^3)$.

Yet Another Example of Loop Analysis

Algorithm: *LoopAnalysis3*$(n : integer)$

$sum \leftarrow 0$

for $i \leftarrow 1$ **to** n

$$\textbf{do} \begin{cases} j \leftarrow i \\ \textbf{while } j \geq 1 \\ \quad \textbf{do} \begin{cases} sum \leftarrow sum + i/j \\ j \leftarrow \left\lfloor \frac{j}{2} \right\rfloor \end{cases} \end{cases}$$

return (sum)