MA1: L04

Tom Smedsaas

Asymptotic notation
Big O ($\mathcal{O}$), Omega (Ω) and Theta (Θ)

Asymptotic notation

Examples of statements about algorithms

- *The simple algorithm to compute* x^n *is* $\Theta(n)$
- *The squaring algorithm to compute* x^n *is* $\Theta(\log n)$
- *The binary search algorithm is* $\Theta(\log n)$

- *A successful linear search is* $\mathcal{O}(n)$
- *An unsuccessful linear search is* $\Omega\,(n)$

Asymptotic notation - definition

A function $t(n)$ is said to be $\mathcal{O}(f(n))$ if there exist two constants c and n_0 such that

$$|t(n)| < c|f(n)| \quad \text{for all} \quad n > n_0$$

Note that this is a *mathematical* definition!

Asymptotic notation

Example: $f(n) = n^2 + 10n + 100$ is $\mathcal{O}(n^2)$

Why?

If $n > 1$ then

$$|f(n)| = n^2 + 10n + 100 < n^2 + 10n \cdot n + 100 \cdot n \cdot n = 111n^2$$

Thus, the constants $c = 111$ and $n_0 = 1$ works.

Note that it is the *existence* that is important – not the values!

Asymptotic notation

Since $\mathcal{O}$ is an *upper* limit it doesn't have to mean so much.

The function on the previous slide is, for example, also $\mathcal{O}(n^3)$ and $\mathcal{O}(2^n)$.

To specify a *lower* limit we can use Ω with a similar definition but with > instead of <.

The example function is also $\Omega(n^2)$.

If a functions is both $\mathcal{O}(f(n))$ and $\Omega(f(n))$ it is said to be $\Theta(f(n))$.

In computer science it is quite common to say $\mathcal{O}$ when you really mean Θ.

UPPSALA UNIVERSITET

Usage in algorithm analysis

We use these concept to express the time a specific algorithm takes.

Instead of time we can look on the number of times a central operation is performed.

Example: The number of multiplications performed in the squaring algorithm to compute x^n is $\Theta(\log n)$.

Note that we do not have to specify the base of the logarithm since

$$\log_a x = \frac{1}{\log_b a} \cdot \log_b x$$

UPPSALA UNIVERSITET

Usage in algorithm analysis

Example:

$\mathcal{O}(1)$	indexing an array (a Python-list)
$\Theta(\log n)$	binary search
$\Theta(n)$	`insert(0, x)` in a Python-list of length *n*
$\Theta(n \log n)$	fast sorting methods
$\Theta(n^2)$	simple sorting methods
$\Theta(n^3)$	multiplication of $n \times n$ - matrices

Usage in algorithm analysis

When analyzing algorithms, one may need to distinguish between best, worst and average complexity.

Example:

To sort n elements with *insertion sort* requires

- $\Theta(n)$ operations in best case
- $\Theta(n^2)$ operations in worst case
- $\Theta(n^2)$ operations on the average

What is meant by ”*on the average*”?

Usage in algorithm analysis

When discussing searching, one often needs to distinguish between "successful" and ”unsuccessful" search.

For example, the function

```
def search(x, lst):
    for e in lst:
        if e == x:
            return True
    return False
```

requires for a *successful search*

- $\Theta(1)$ operations in best case
- $\Theta(n)$ operations in worst case
- $\Theta(n)$ operations on average

while an unsuccessful search always require $\Theta(n)$ operations.

UPPSALA UNIVERSITET

Time estimations

Suppose we know that the time of a certain algorithm is $\Theta\big(f(n)\big)$.

We can then estimate the time for large values of *n* with the function

$$t(n) = c \cdot f(n)$$

The constant c depends on the computer dator, details in the program etc.

For a particular implementation and computer, it can be estimated by making one or more time measurements.

Example

Suppose that the time for a program that implements a $\Theta(n \log n)$ - algorithm has been measured to 1 sec for $n = 10^3$.
How long will the program then take for $n = 10^6$?

$$t(n) = c\, n \log n$$

$$t(10^3) = c \cdot 10^3 \log 10^3 = 1$$

$$c = 1/3000$$

$$t(10^6) = \frac{1}{3000} \cdot 10^6 \log 10^6 = 2000 \ \mathrm{sec} \ \approx 33 \text{ min}$$

Example

Suppose we have measured the sam but for a $\Theta(n^2)$ - algorithm.
How long will the program then take for $n = 10^6$?

$$t(n) = c \cdot n^2$$

$$t(10^3) = c \cdot (10^3)^2 = 1$$

$$c = 10^{-6}$$

$$t(10^6) = 10^{-6} \cdot (10^6)^2 = 10^6 \text{ sec } \approx 12 \text{ days}$$

Conclusion : A $\Theta(n \log n)$ - algorithm is *much* faster than a $\Theta(n^2)$ - algorithm for large values n.

The end

.