Topic 6: Algorithm Analysis & Sorting[1]
(Version of November 14, 2011)

Pierre Flener

Computing Science Division
Department of Information Technology
Uppsala University
Sweden

Course 1DL201:
Program Construction and Data Structures

[1]Based on original slides by John Hamer and Yves Deville, with some figures from the CLRS textbook (which are © The MIT Press, 2009)

Asymptotic Algorithm Analysis

We can analyse an algorithm without needing to run it, and thus gain some understanding of its likely performance.

This analysis can be done at design time, before the program is written. Even if the analysis is approximate, performance problems may be detected.

The notation used in the analysis is helpful in documenting software libraries. It allows programs using such libraries to be analysed without requiring analysis of the library source code (which is often not available).

We will mostly analyse the runtime performance.
The same principles apply to memory consumption.
We speak of time complexity and space complexity.

The Θ Notation

The Θ notation is used to denote a set of functions that increase at the same rate (within some constant bound).

Formally, $\Theta(g(n))$ is the set of all functions $f(n)$ that are bounded below by $c_1 \cdot g(n) \geq 0$ and above by $c_2 \cdot g(n)$, for some constants $c_1 > 0$ and $c_2 > 0$, when n gets sufficiently large, that is, when n is at least some constant $n_0 > 0$.

The function $g(n)$ in $\Theta(g(n))$ is called a complexity function.

We write $f(n) = \Theta(g(n))$ when we mean $f(n) \in \Theta(g(n))$.

The Θ notation is used to give asymptotically tight bounds.

Terminology

Let $\lg x$ denote $\log_2 x$, let $k \geq 2$ be a constant, and let variable n denote the input size:

Function	Growth Rate	
1	constant	sub-linear
$\lg n$	logarithmic	
$\lg^2 n$	log-squared	
n	linear	polynomial
$n \cdot \lg n$		
n^2	quadratic	
n^3	cubic	
k^n	exponential	exponential
$n!$		super-exponential
n^n		

☞ From now on (and in the homeworks), we use $\Theta(1)$ instead of introducing constants such as t_0 and t_{add}.

UPPSALA UNIVERSITET

Example

Theorem: $n^2 + 5 \cdot n + 10 = \Theta(n^2)$.

Proof: We need to choose constants $c_1 > 0$, $c_2 > 0$, and $n_0 > 0$ such that

$$0 \leq c_1 \cdot n^2 \leq n^2 + 5 \cdot n + 10 \leq c_2 \cdot n^2$$

for all $n \geq n_0$. Dividing by n^2 (assuming $n > 0$) gives

$$0 \leq c_1 \leq 1 + \frac{5}{n} + \frac{10}{n^2} \leq c_2$$

The "sandwiched" term, $1 + \frac{5}{n} + \frac{10}{n^2}$, gets smaller as n grows. It peaks at 16 for $n = 1$, so we can pick $n_0 = 1$ and $c_2 = 16$. It drops to 6 for $n = 2$ and becomes close to 1 for $n = 1000$. It never gets less than 1, so we can pick $c_1 = 1$. $\square$

Exercise: Prove that $5 \cdot n^3 + 7 \cdot n^2 - 3 \cdot n + 4 \neq \Theta(n^2)$.

Keep Complexity Functions Simple

While it is formally (and trivially) correct to say that $n^2 + 5 \cdot n + 10 = \Theta(n^2 + 5 \cdot n + 10)$, the whole purpose of the Θ notation is to work with simple expressions.
Thus, we often do not expect any arbitrary factors or lower-order terms inside a complexity function.

We can simplify complexity functions by:

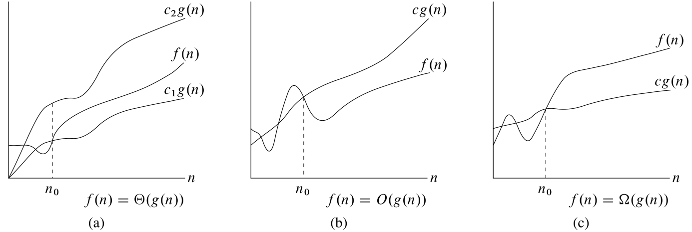
- Setting all constant factors to 1.
- Dropping all lower-order terms.

Since $\log_b x = \frac{1}{\log_c b} \cdot \log_c x$, where $\frac{1}{\log_c b}$ is a constant factor (when the bases b and c are constants), we shall use $\lg x$ in complexity functions.

UPPSALA UNIVERSITET

Variations on Θ: The *O* and Ω Notations

Variants of Θ include *O* ("big-Oh"), which drops the lower bound, and Ω ("big-Omega"), which drops the upper bound:



Examples: Any linear function $a \cdot n + b$ is in $O(n^2)$, $O(n^3)$, and so on, but not in $\Theta(n^2)$, $\Theta(n^3)$, and so on. Any quadratic function $a \cdot n^2 + b \cdot n + c$ is in $\Omega(n)$. We use O to give an upper bound on a function, and Ω to give a lower bound, but no claims are made about how tight these bounds are. We use Θ to give a tight bound, namely when the upper and lower bounds are the same.

Application of a Pre-Established Formula

Theorem 1 (proof omitted): If, for some constants a and b:

$$C(n) = \begin{cases} \Theta(1) & \text{if } n \leq b \\ a \cdot C(n-1) + \Theta(1) & \text{if } n > b \end{cases}$$

then the closed form of the recurrence is:

$$C(n) = \begin{cases} \Theta(n) & \text{if } a = 1 \\ \Theta(a^n) & \text{if } a > 1 \end{cases}$$

Another pre-established formula is in the Master Theorem:

page 44

UPPSALA UNIVERSITET

The Master Method and Master Theorem

From now on, we will ignore the base cases of a recurrence.

The closed form for a recurrence $T(n) = a \cdot T(n/b) + f(n)$ reflects the "battle" between the two terms in the sum. Think of $a \cdot T(n/b)$ as the process of "distributing the work out" to $f(n)$, where the actual work is done.

Theorem 2 (known as the Master Theorem, proof omitted):

1. If $f(n)$ is dominated by $n^{\log_b a}$ (see the next page), then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n)$ and $n^{\log_b a}$ are balanced (if $f(n) = \Theta(n^{log_b a})$), then $T(n) = \Theta(n^{log_b a} \cdot \lg n)$.
3. If $f(n)$ dominates $n^{\log_b a}$ and if the regularity condition (see the next page) holds, then $T(n) = \Theta(f(n))$.

Dominance and the Regularity Condition

The three cases of the Master Theorem depend on comparing $f(n)$ to $n^{\log_b a}$. However, it is not sufficient for $f(n)$ to be "just a bit" smaller or bigger than $n^{\log_b a}$. Cases 1 and 3 only apply when there is a polynomial difference between these functions, that is when the ratio between the dominator and the dominee is asymptotically larger than the polynomial n^ϵ for some constant $\epsilon > 0$.

Example: n^2 is polynomially larger than both $n^{1.5}$ and $\lg n$.

Counter-Example: $n \cdot \lg n$ is not polynomially larger than n.

In Case 3, a regularity condition requires $a \cdot f(n/b) \leq c \cdot f(n)$ for some constant $c < 1$ and all sufficiently large n.
(All the f functions in this course will satisfy this condition.)

Gaps in the Master Theorem

The Master Theorem does not cover all possible recurrences of the form $T(n) = a \cdot T(n/b) + f(n)$:

- Cases 1 and 3: The difference between $f(n)$ and $n^{log_b a}$ might not be polynomial.
  Counter-Example: The Master Theorem does not apply to the recurrence $T(n) = 2 \cdot T(n/2) + n \cdot \lg n$, despite it having the proper form. We have $a = 2 = b$, so we need to compare $f(n) = n \cdot \lg n$ to $n^{\log_b a} = n^1 = n$. Clearly, $f(n) = n \cdot \lg n > n$ for large enough n, but the ratio $f(n)/n$ is $\lg n$, which is asymptotically less than the polynomial n^ϵ for any constant $\epsilon > 0$, so we are not in Case 3.
- Case 3: The regularity condition might not hold.

UPPSALA UNIVERSITET

Common Cases of the Master Theorem

a	b	$n^{\log_b a}$	$f(n)$	Case	$T(n)$
1	2	n^0	$\Theta(1)$	2	$\Theta(\lg n)$
			$\Theta(\lg n)$	none	$\Theta(?)$
			$\Theta(n \cdot \lg n)$	3	$\Theta(n \cdot \lg n)$
			$\Theta(n^k)$, with $k > 0$	3	$\Theta(n^k)$
2	2	n^1	$\Theta(1)$	1	$\Theta(n)$
			$\Theta(\lg n)$	1	$\Theta(n)$
			$\Theta(n)$	2	$\Theta(n \cdot \lg n)$
			$\Theta(n \cdot \lg n)$	none	$\Theta(?)$
			$\Theta(n^k)$, with $k > 1$	3	$\Theta(n^k)$

(This table can only be used for looking up a closed form, but it cannot be referred to in the homeworks or exams.)