Chapter 1

Introduction

Algorithm:

- The origin of the word **algorithm** is "al-kawarizmi", a ninth-century Persian mathematician:

  أبو جعفر محمد إبن موسى الخوارزمي

- A set of rules for solving a problem in a finite number of steps. (*Machine independent and language independent*).

Computer program:

- Implementation of an algorithm
- Series of well stated instructions to be performed by the computer
- These instructions are stated in a programming language. (*Machine independent*)

1.1 Designing an algorithm

Only a handful of techniques are used in the thousands of algorithms that are written to solve various problems. They describe the general manner in which the algorithm operates.

Technique: General method in which an algorithm operates.

Most common techniques:

- Divide-and-conquer
- Greedy technique
- Dynamic programming
- Backtracking (branch and bound)

1.2 Analyzing an algorithm

Analysis: Make use of mathematics to be able to predict the **time** and **space** the execution of its implemented program would require.

1.2.1 Measuring criteria

Consider the following measures to compare the work done by different algorithms:

- **Execution time (CPU time)**
  Not a good measure since machine dependent (example: different for different architecture and technologies).

- **Number of (executable) statements**
  It depends on the programming language being used and on the programmer's style and is therefore not a good criterion.

- **Number of repetitions in loop**
  It is not a good measure since the amount of work can vary from one pass to another. It is also language dependent.

- **Count basic operations**
  Choose a basic operation and count the number of times it is performed. This is a good measuring criterion.

  EXAMPLES:

Problem	**Basic operation**
Find key in list	Comparison (of key with entry in list)
Multiply two matrices	Multiplication (of two numbers)
Sorting	Comparison (of two entries in list)
Traverse (binary tree)	Traversing one link

1.2.2 Why do we need to analyze an algorithm ?

- Analysis is generally more reliable than experimentation. An experiment is input-data dependent, analysis can give guarantees about all inputs.

- Analysis helps in choosing among several algorithms that solve the given problem. [We don't need to implement and test all the different algorithms].

- Analysis allows us to predict the performance of a program before writing the code.

- Analysis allows to get a better understanding of the solution; to anticipate the "slow" and "fast" parts, and to try and improve the "slow" ones.

1.2.3 Θ, O and Ω Notations

Given a particular input (of size n, for example), the **running time** of an algorithm $T(n)$ is the number of basic operations (or steps) it performs. $T(n)$ is a function of the input size n. Its leading term is the **growth rate** of $T(n)$. We are interested in the **order of growth** (rate of growth) of $T(n)$.

Assume that $f(n)$ and $g(n)$ are running time functions.

1. **Θ-Notation**
   $f(n)$ is $\Theta(g(n))$

$$\text{iff } \exists\, c_1 \in \mathbf{R}^+, c_2 \in \mathbf{R}^+ \text{and } \exists\, n_0 \in \mathbf{Z}^+$$

$$\text{such that: } c_1 g(n) \leq f(n) \leq c_2 g(n)\ \forall\, n > n_0.$$

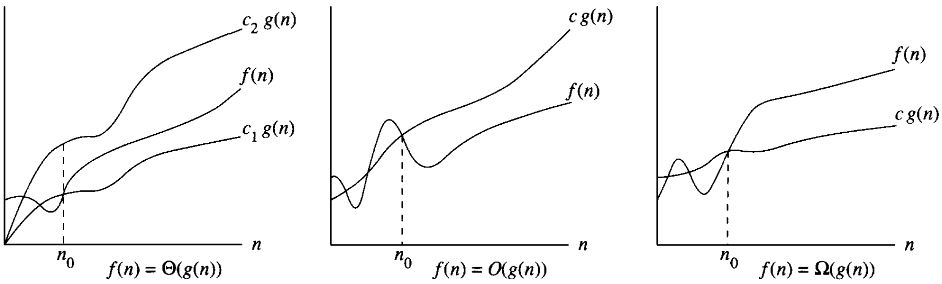

Figure 1.1: The Θ, O and Ω Notations [CLR90, p. 25]

Note:

(a) $f(n)$ is sandwiched between $c_1 g(n)$ and $c_2 g(n)$ for $n > n_0$.

(b) $f(n)$ is asymptotically bounded from above and below by $g(n)$.

2. O**-Notation**
$f(n)$ is $O(g(n))$

$$\text{iff } \exists\, c \in \mathbf{R}^+ \text{and } \exists\, n_0 \in \mathbf{Z}^+$$
$$\text{such that: } f(n) \leq cg(n)\ \forall\, n > n_0.$$

Note:

(a) $f(n)$ is of order $g(n)$.

(b) $f(n)$ is asymptotically upper bounded by $g(n)$.

3. Ω**-Notation**
$f(n)$ is $\Omega(g(n))$

$$\text{iff } \exists\, c \in \mathbf{R}^+ \text{and } \exists\, n_0 \in \mathbf{Z}^+$$
$$\text{such that: } cg(n) \leq f(n)\ \forall\, n > n_0.$$

Note:

(a) $f(n)$ is asymptotically lower bounded by $g(n)$.

Results:

- $f(n)$ is $O(g(n)) \Leftrightarrow g(n)$ is $\Omega(f(n))$
- $f(n)$ is $\Theta(g(n)) \Leftrightarrow \begin{cases} f(n) \text{ is } O(g(n)) \\ \quad\text{and} \\ f(n) \text{ is } \Omega(g(n)) \end{cases}$

1.3 Examples

A. Given a polynomial $p(n) = a_0 + a_1 n + \cdots + a_k n^k$, *where* $a_k > 0$.
Show that $p(n)$ *is* $\Theta(n^k)$.
Proof: We want to show $\exists\, c_1, c_2 \in \mathbf{R}^+$ and $n_0 \in \mathbf{Z}^+$ such that:

$$c_1 n^k \leq a_0 + a_1 n + \cdots + a_k n^k \leq c_2 n^k.$$

1. Show $a_0 + a_1 n + \cdots + a_k n^k \leq c_2 n^k$.
   Let $c_2 = |a_0| + |a_1| + \cdots + |a_k|$ and $n_0 = 1$.
   For all $n > n_0$ we have:

$$p(n) = a_0 + a_1 n + \cdots + a_k n^k$$

$$\leq |a_0| + |a_1| n + \cdots + |a_k| n^k$$
$$\leq |a_0| n^k + |a_1| n^k + \cdots + |a_k| n^k$$
$$= n^k(|a_0| + |a_1| + \cdots + |a_k|) = c_2 n^k$$

   which is what we wanted to prove.

2. Show $c_1 n^k \leq a_0 + a_1 n + \cdots + a_k n^k$.
   Let $c_1 = \frac{a_k}{2}$.
   We want to show that:

$$\frac{a_k}{2} n^k \leq a_0 + a_1 n + \cdots + a_k n^k$$

$$\text{i.e. } a_0 + a_1 n + \cdots + \frac{a_k}{2} n^k \geq 0$$

$$\text{or } \underbrace{n^k}_{>0} \left(\frac{a_k}{2} + \frac{a_{k-1}}{n} + \cdots + \frac{a_0}{n^k} \right) > 0$$

$$\text{If } \frac{a_k}{2} \geq \frac{|a_{k-1}|}{n} + \cdots + \frac{|a_0|}{n^k} \geq 0$$

$$\text{then } \left(\frac{a_k}{2} + \frac{a_{k-1}}{k} + \cdots + \frac{a_0}{n^k} \right) > 0 \text{ always}$$

$$\text{even if } a_0 < 0, a_1 < 0 \ldots a_{k-1} < 0.$$

   Let $M = \max\{|a_0|, |a_1|, \ldots, |a_{k-1}|\}$.
   Let $n_0 = \max\left\{1, \frac{2Mk}{a_k}\right\}$.
   For any $n > n_0$ and all $i, 0 \leq i \leq k-1$ and $a_i \neq 0$ we have:

$$\frac{|a_i|}{n^{k-i}} \leq \frac{|a_i|}{n} \text{ since } 0 \leq i \leq k-1 \text{ (i.e. } 1 \leq k-i)$$

$$\text{and } \frac{|a_i|}{n} \leq \frac{M}{n} \text{ since } M = \max\{|a_0|, |a_1|, \ldots, |a_{k-1}|\}.$$

$$\text{And } \frac{M}{n} \leq \frac{a_k}{2k} \text{ since } \frac{2Mk}{a_k} \leq n_0 < n.$$

$$\text{So } \frac{|a_i|}{n^{k-i}} \leq \frac{a_k}{2k} \text{ for } n > n_0.$$

$$\text{And } \left(\frac{a_k}{2k}\right) k \geq \frac{|a_{k-1}|}{n} + \frac{|a_{k-2}|}{n^2} + \cdots + \frac{|a_0|}{n^k}.$$

$$\Rightarrow \frac{a_k}{2} \geq \frac{|a_{k-1}|}{n} + \frac{|a_{k-2}|}{n^2} + \cdots + \frac{|a_0|}{n^k}$$

   which is what we wanted to prove. □

B.i) Show that $\log_2 n$ *is* $O(n)$.
Proof: We want to show that $\exists\, c, \exists\, n_0$ so that $\log_2 n \leq cn \;\forall\, n > n_0$.
$2^n = (1+1)^n = \sum_{i=0}^{n} \binom{n}{i} = 1 + n + \frac{n(n-1)}{2} + \ldots + \frac{n(n-1)}{2} + n + 1$
$\Rightarrow 2^n \geq 1 + n + n^2 - n + n + 1\;(\forall\, n > 5) \; = n^2 + n + 2 > n$
So we have $2^n \geq n \;(\forall\, n > 5)$
$\Rightarrow \log_2 2^n \geq \log_2 n \;\Rightarrow n \log_2 2 \geq \log_2 n \;\Rightarrow \log_2 n \leq n$.
So for $c = 1$ and $n_0 = 4$ we finally have $\log_2 n \leq n \;\forall\, n > n_0$. $\square$

B.ii) Show that $\log_2 n$ *is NOT* $\Omega(n)$.
Proof: Show that $\forall\, c \in \mathbf{R}^+$ and $\forall\, n_0 \in \mathbf{Z}^+$, $\exists\, n \in \mathbf{Z}^+, n > n_0$ such that:

$$\log_2 n \leq cn \text{ or } n \leq 2^{cn}$$

We have: $2^{cn} = (1 + (2^c - 1))^n = \sum_{i=0}^{n} \binom{n}{i}(2^c-1)^i > \binom{n}{2}(2c-1)^2 \;\forall\, n \geq 2$.
We want to show $2^{cn} > n$, which is achieved by showing that $\binom{n}{2}(2^c - 1)^2 > n$
$\Leftrightarrow \frac{n(n-1)}{2}(2^c-1)^2 > n \;\Leftrightarrow \frac{(n-1)(2^c-1)^2}{2} > 1$
$\Leftrightarrow n - 1 > \frac{2}{(2^c-1)^2} \Leftrightarrow n > 1 + \frac{2}{(2^c-1)^2}$
So for any c and n_0 we define $n_1 = \max\left\{n_0, 1 + \frac{2}{(2c-1)^2}\right\}$
If $n > n_1$ then $2^{cn} > n \Rightarrow \log_2 n < cn$. So $\log_2 n$ is not $\Omega(n)$. $\square$

C. Show that the growth rate of $\sum_{i=1}^{n} \frac{1}{i}$ *is* $\Theta(\log_2 n)$.
Proof: Bounds for $\sum_{i=1}^{2^k} \frac{1}{i}$:

$$1 + \frac{1}{2} + \overbrace{\frac{1}{4} + \frac{1}{4}} + \overbrace{\frac{1}{8} + \frac{1}{8} + \frac{1}{8} + \frac{1}{8}} + \ldots + \frac{1}{2^k}$$

$$< 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{5} + \frac{1}{6} + \frac{1}{7} + \frac{1}{8} + \ldots + \frac{1}{2^k}$$

$$< 1 + \underbrace{\frac{1}{2} + \frac{1}{2}} + \underbrace{\frac{1}{4} + \frac{1}{4} + \frac{1}{4} + \frac{1}{4}} + \frac{1}{8} + \ldots + \frac{1}{2^k}$$

$$\Rightarrow 1 + \sum_{i=1}^{k} (2^{i-1} \frac{1}{2^i} < \sum_{i=1}^{n} \frac{1}{i} < \sum_{i=1}^{k-1} 2^i \frac{1}{2^i} + \frac{1}{2^k}$$

$$\Rightarrow 1 + \frac{1}{2} \sum_{i=1}^{k} 1 < \sum_{i=1}^{n} \frac{1}{i} < \sum_{i=1}^{k} 1 + \frac{1}{2^k}$$

$$\Rightarrow 1 + \frac{k}{2} < \sum_{i=1}^{2^k} \frac{1}{i} < k + \frac{1}{2^k}$$

For any n: $2^k \leq n \leq 2^{k+1}$ i.e. $k = \lfloor \log_2 n \rfloor$ where $\lfloor x \rfloor$ denotes the greatest integer not exceeding x.

$$\Rightarrow 1 + \frac{k}{2} < \sum_{i=1}^{2^k} \frac{1}{i} < \sum_{i=1}^{n} \frac{1}{i} < \sum_{i=1}^{2^{k+1}} \frac{1}{i} < k + 1 + \frac{1}{2^{k+1}}$$

$$k = \lfloor \log_2 n \rfloor \Rightarrow \log_2 n - 1 < k \leq \log_2 n.$$

$$\text{So we have: } 1 + \frac{k}{2} < \sum_{i=1}^{n} \frac{1}{i} < \sum_{i=1}^{2^{k+1}} \frac{1}{i} < k + 1 + \frac{1}{2^{k+1}}$$

$$1 + \frac{\log_2 n - 1}{2} < 1 + \frac{k}{2} < \sum_{i=1}^{n} \frac{1}{i} < \log_2 n + 1 + \frac{1}{2^{\log_2 n}}$$

$$\text{Since } k + 1 + \frac{1}{2^{k+1}} < \log_2 n + 1 + \frac{1}{2^{\log_2 n}}$$

$$\Rightarrow \frac{1}{2} + \frac{\log_2 n}{2} < \sum_{i=1}^{n} \frac{1}{i} < \log_2 n + 1 + \frac{1}{n} < \log_2 n + 2$$

$$\frac{\log_2 n}{2} < \sum_{i=1}^{n} \frac{1}{i} < 2\log_2 n \text{ since } \log_2 n > 2 \text{ for } n \geq 5.$$

For $c_1 = \frac{1}{2}, c_2 = 2$ and $n_0 = 4$ we have $\frac{\log_2 n}{2} < \sum_{i=1}^{n} \frac{1}{i} < 2\log_2 n \; \forall n > 4$.
So we have proved that $\sum_{i=1}^{n} \frac{1}{i}$ is $\Theta(\log_2 n)$. $\square$

1.4 Some common running times

1	$O(1)$ means that the number of executions of the basic operation is fixed.
$\log n$	logarithmic running time
n	linear running time
$n \log n$	arises when the problem is broken into independent subproblems. They are then solved and combined
n^2	quadratic [e.g. some sorting algorithms]
n^3	cubic running time
2^n	exponential; not practical

As n gets large, the difference between exponential and polynomial growth rates becomes **tremendous**.

In the next four chapters we study the most common algorithmic techniques mentioned in Section 1.1, namely, divide and conquer, greedy, dynamic programming and backtracking algorithms.

	Size n					
Growth Rates	10	20	30	40	50	60
n	.00001 second	.00002 second	.00003 second	.00004 second	.00005 second	.00006 second
n^2	.0001 second	.0004 second	.0009 second	.0016 second	.0025 second	.0036 second
n^3	.001 second	.008 second	.027 second	.064 second	.125 second	.216 second
n^5	.1 second	3.2 second	24.3 second	1.7 minutes	5.2 minutes	13.0 minutes
2^n	.001 second	1.0 second	17.9 minutes	12.7 days	35.7 years	366 centuries
5^n	.059 second	58 minutes	6.5 years	3855 centuries	2 x 10^8 centuries	1.3 x 10^{13} centuries

Table 1.1: Comparison of several polynomial and exponential time complexity functions [GJ79, p. 7]

Bibliography

[CLR90] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms.* MIT Press, Cambridge, 1990.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability. A Guide to the Theory of $\mathcal{NP}$-Completeness.* W.H. Freeman and Company, 1979.