



```
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#define MAXPAROLA 30
#define MAXRIGA 80

int main(int argc, char *argv[])
{
    int freq[MAXPAROLA]; /* vettore di contatori
della frequenza delle lunghezze delle parole */
    char riga[MAXRIGA];
    int i, inizio, lunghezza;
    FILE *f;

    for(i=0; i<MAXPAROLA; i++)
        freq[i]=0;

    if(argc != 2)
    {
        fprintf(stderr, "ERRORE: serve un parametro con il nome del file\n");
        exit(1);
    }
    f = fopen(argv[1], "r");
    if(f==NULL)
    {
        fprintf(stderr, "ERRORE: impossibile aprire il file %s\n", argv[1]);
        exit(1);
    }

    while( fgets( riga, MAXRIGA, f ) != NULL )
```

Algorithms and Complexity

Introduction to complexity analysis

Stefano Quer

Dipartimento di Automatica e Informatica

Politecnico di Torino

Complexity Analysis

❖ Definition

- Forecast of resources (memory, time) needed by the algorithm for execution

❖ Features

- Machine-independent
- Assumption
 - Sequential single-processor model (traditional architecture)
- Independent of the input data of a particular instance of the problem

Complexity Analysis

- ❖ It depends on the size n of the problem
- ❖ A lower complexity may compensate hardware efficiency
- ❖ Examples
 - Integer multiplication: Number of bits of the operands
 - Sorting algorithm: Number of data to sort
 - ...
- ❖ Output
 - $T(n)$: execution time
 - $S(n)$: memory occupation

A Simple Counting Problem

- ❖ Write a program able to
 - Read an integer value N
 - Print-out the number M of ordered couples (i, j) such that i and j are integer values and $1 \leq i \leq j \leq N$
- ❖ Example
 - Input: $N=4$
 - Generated couples
 - $(1,1)(1,2)(1,3)(1,4) (2,2)(2,3)(2,4) (3,3)(3,4)(4,4)$
 - Output: $M = 10$

Algorithm 1

```
int count_ver1 (int N) {  
    int i, j, sum;  
  
    sum = 0;  
  
    for (i=0; i<=N; i++) {  
        for (j=i; j<=N; j++) {  
            sum++;  
        }  
    }  
  
    return sum;  
}
```

It generates all pairs:
 $1 \leq i \leq j \leq N$

It counts-them up

It returns the result

Algorithm 1

```
int count_ver1 (int N) {  
    int i, j, sum;  
  
    sum = 0;  
  
    for (i=0; i<=N; i++) {  
        for (j=i; j<=N; j++) {  
            sum++;  
        }  
    }  
  
    return sum;  
}
```

1

 $1 + (N + 1) + N$

$$\sum_{i=1}^N [1 + (N - i + 2) + (N - i + 1)]$$

$$\sum_{i=1}^N (N - i + 1)$$

1

Algorithm 1

$$T(N) = 4 + 2N + \sum_{i=1}^N (5 + 3N - 3i)$$

$$T(N) = 4 + 2N + \sum_{i=1}^N (5) + \sum_{i=1}^N (3N) - \sum_{i=1}^N (3i)$$

5N

 $3N^2$

$$T(N) = 4 + 7N + 3N^2 - 3 \sum_{i=1}^N i$$

$$T(N) = 4 + 7N + 3N^2 - 3 \frac{N(N+1)}{2}$$

$$T(N) = 1.5N^2 + 5.5N + 4$$

$$\begin{array}{c} 1 \\ 1 + (N+1) + N \\ \sum_{i=1}^N [1 + (N-i+2) + (N-i+1)] \\ \sum_{i=1}^N (N-i+1) \\ 1 \end{array}$$

Finite arithmetic progression
 $1 + 2 + 3 + \dots + n = \frac{n(n+1)}{2}$

Algorithm 2

```
int count_ver2 (int N) {  
    int i, sum;  
  
    sum = 0;  
  
    for (i=1; i<=N; i++) {  
        sum = sum + (N-i+1);  
    }  
  
    return sum;  
}
```

```
int count_ver1 (int N) {  
    int i, j, sum;  
    sum = 0;  
    for (i=0; i<=N; i++) {  
        for (j=i; j<=N; j++) {  
            sum++;  
        }  
    }  
    return sum;  
}
```

It generates all pairs:
 $1 \leq i \leq j \leq N$

Algorithm 2

```
int count_ver2 (int N) {  
    int i, sum;  
  
    sum = 0;  
  
    for (i=1; i<=N; i++) {  
        sum = sum + (N-i+1);  
    }  
  
    return sum;  
}
```

1

 $1 + (N + 1) + N$

$$\sum_{i=1}^N (4)$$

1

Algorithm 2

$$T(N) = 6N + 4$$

1

$$1 + (N + 1) + N$$

$$\sum_{i=1}^N (4) = 4N$$

1

Algorithm 3

```
int count_ver2 (int N) {  
    int i, sum;  
  
    sum = 0;  
  
    for (i=1; i<=N; i++) {  
        sum = sum + (N-i+1);  
    }  
  
    return sum;  
}
```

❖ The for cycle computes

$$\begin{aligned} & \sum_{i=1}^N (N - i + 1) = \\ &= N^2 + N - \sum_{i=1}^N i = \\ &= N(N + 1) - \frac{N(N+1)}{2} \\ &= \frac{N(N+1)}{2} \end{aligned}$$

➤ which can be used to substitute the entire cycle

Algorithm 3

❖ The for cycle computes

➤ $\sum_{i=1}^N (N - i + 1) = \frac{N(N+1)}{2}$

➤ which can be used to substitute the entire cycle

```
int count_ver3 (int N) {  
    return N * (N+1) / 2;  
}
```

```
int count_ver2 (int N) {  
    int i, sum;  
    sum = 0;  
    for (i=1; i<=N; i++) {  
        sum = sum + (N-i+1);  
    }  
    return sum;  
}
```

It generates all pairs:
 $1 \leq i \leq j \leq N$

Algorithm 3

❖ The for cycle computes

➤ $\sum_{i=1}^N (N - i + 1) = \frac{N(N+1)}{2}$

➤ which can be used to substitute the entire cycle

```
int count_ver3 (int N) {  
    return N * (N+1) / 2;  
}
```

4

$T(N) = 4$

Summary

Algorithm	$T(N)$	Order of $T(N)$
Version 1	$1.5N^2 + 5.5N + 4$	N^2
Version 2	$6N + 4$	N
Version 3	4	<i>constant</i>

Algorithm
Classification

Asymptotic behavior	Algorithm Class
1	Constant
$\log n$	Logarithmic
n	Linear
$n \log n$	Linearithmic
n^2	Quadratic
n^3	Cubic
2^n	Exponential

Summary

❖ Hypothesis

➤ 1 operation = 1 nsec = 10^{-9} sec

Wall-clock
(elapsed) time

Asymptotic behavior	10^3	10^4	10^5	10^6	10^7
N	1μs	10μs	100μs	1ms	10ms
20 n	20μs	200μs	2ms	20ms	200ms
n log n	9.96μs	132μs	1.66ms	19.9ms	232ms
20 n log n	199μs	2.7ms	32ms	398ms	4.6sec
n^2	1ms	100ms	10s	17min	1.2day
20 n^2	20ms	2s	3.3min	5.6h	23day
N^3	1s	17min	12day	32years	32 millenium

Some more examples

❖ Discrete Fourier Transform

- Decomposition of a N-sample waveform into periodic components
- Applications: DVD, JPEG, astrophysics,
- Trivial algorithm: Quadratic (N^2)
- FFT (Fast Fourier Transform): Linearitmic ($N \log N$)

❖ Simulation of N bodies

- Simulates gravity interaction among N bodies
- Trivial algorithm: Quadratic (N^2)
- Barnes-Hut algorithm: Linearitmic ($N \log N$)

Worst-case Asymptotic Analysis

❖ Goal

- Guess an upper-bound for $T(n)$ for an algorithm on n data in the worst possible case
- Asymptotic
 - For small n , complexity is irrelevant
 - Understand behaviour for $n \rightarrow \infty$

❖ Why worst-case analysis?

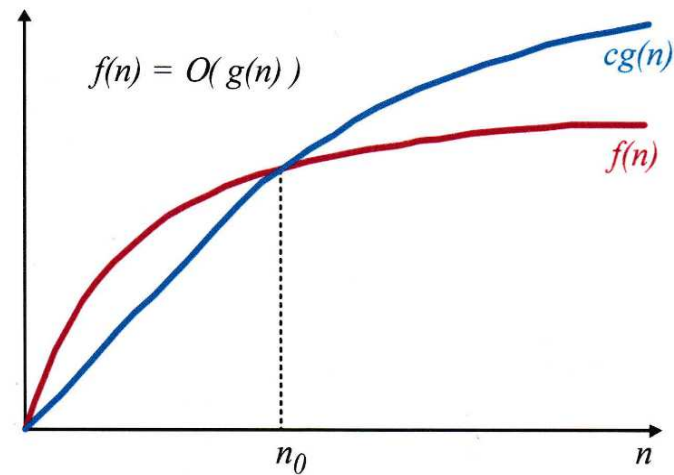
- Conservative guess
- Worst case is very frequent
- Average case
 - Either it coincides with the worst case
 - It is not definable, unless we resort to complex hypotheses on data

O Asymptotic Notation

❖ Definition

$$T(n) = O(g(n)) \Leftrightarrow \exists c > 0, \exists n_0 > 0 \text{ such that } \forall n \geq n_0 \\ 0 \leq T(n) \leq cg(n)$$

$g(n)$ = loose upper bound



O Asymptotic Notation

❖ Examples

- $T(n) = 3n+2 = O(n)$
 - $c=4$ and $n_0=2$
- $T(n) = 10n^2+4n+2 = O(n^2)$
 - $c=11$ and $n_0=5$
- $T(n) = 3n+3 = O(n^2)$
 - $c=3$ and $n_0=2$

❖ Theorem

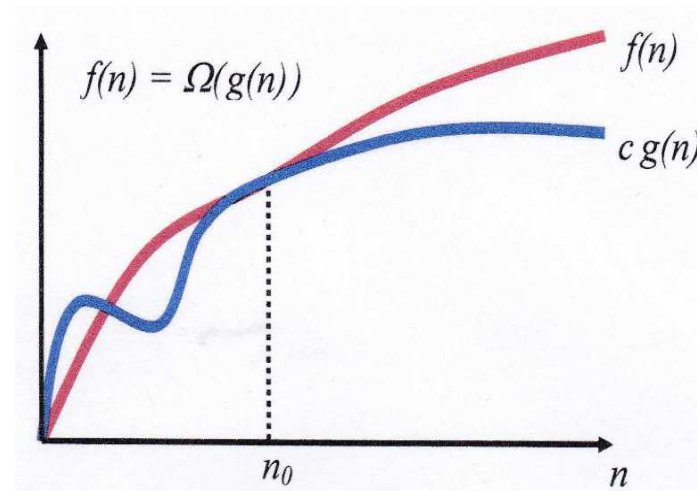
- If $T(n) = a_m n^m + \dots + a_1 n + a_0$
- Then $T(n) = O(n^m)$

Ω Asymptotic Notation

❖ Definition

$$T(n) = \Omega(g(n)) \Leftrightarrow \exists c > 0, \exists n_0 > 0 \text{ such that } \forall n \geq n_0 \\ 0 \leq c g(n) \leq T(n)$$

$g(n)$ = loose lower bound for $T(n)$



Ω Asymptotic Notation

❖ Examples

- $T(n) = 3n+3 = \Omega(n)$
 - $c=3$ and $n_0=1$
- $T(n) = 10n^2+4n+2 = \Omega(n^2)$
 - $c=1$ and $n_0=1$

❖ Theorem

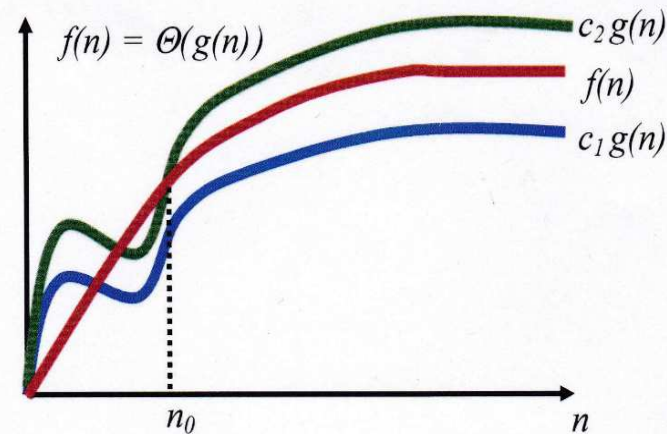
- If $T(n) = a_m n^m + \dots + a_1 n + a_0$
- Then $T(n) = \Omega(n^m)$

⊖ Asymptotic Notation

❖ Definition

$$T(n) = \Theta(g(n)) \Leftrightarrow \exists c_1, c_2 > 0, \exists n_0 > 0 \text{ such that } \forall n \geq n_0 \\ 0 \leq c_1 g(n) \leq T(n) \leq c_2 g(n)$$

$g(n)$ = tight asymptotic bound for $T(n)$



⊖ Asymptotic Notation

❖ Examples

- $T(n) = 3n+2 = \Theta(n)$
 - $c_1=3, c_2=4$ and $n_0=2$
- $T(n) = 3n+2 \neq \Theta(n^2)$
- $T(n) = 10n^2+4n+2 \neq \Theta(n)$

❖ Theorem

- If $T(n) = a_m n^m + \dots + a_1 n + a_0$ then $T(n) = \Theta(n^m)$
- Given two functions $g(n)$ and $T(n)$
- $T(n) = \Theta(g(n)) \Leftrightarrow$
 $T(n) = O(g(n))$ and $T(n) = \Omega(g(n))$