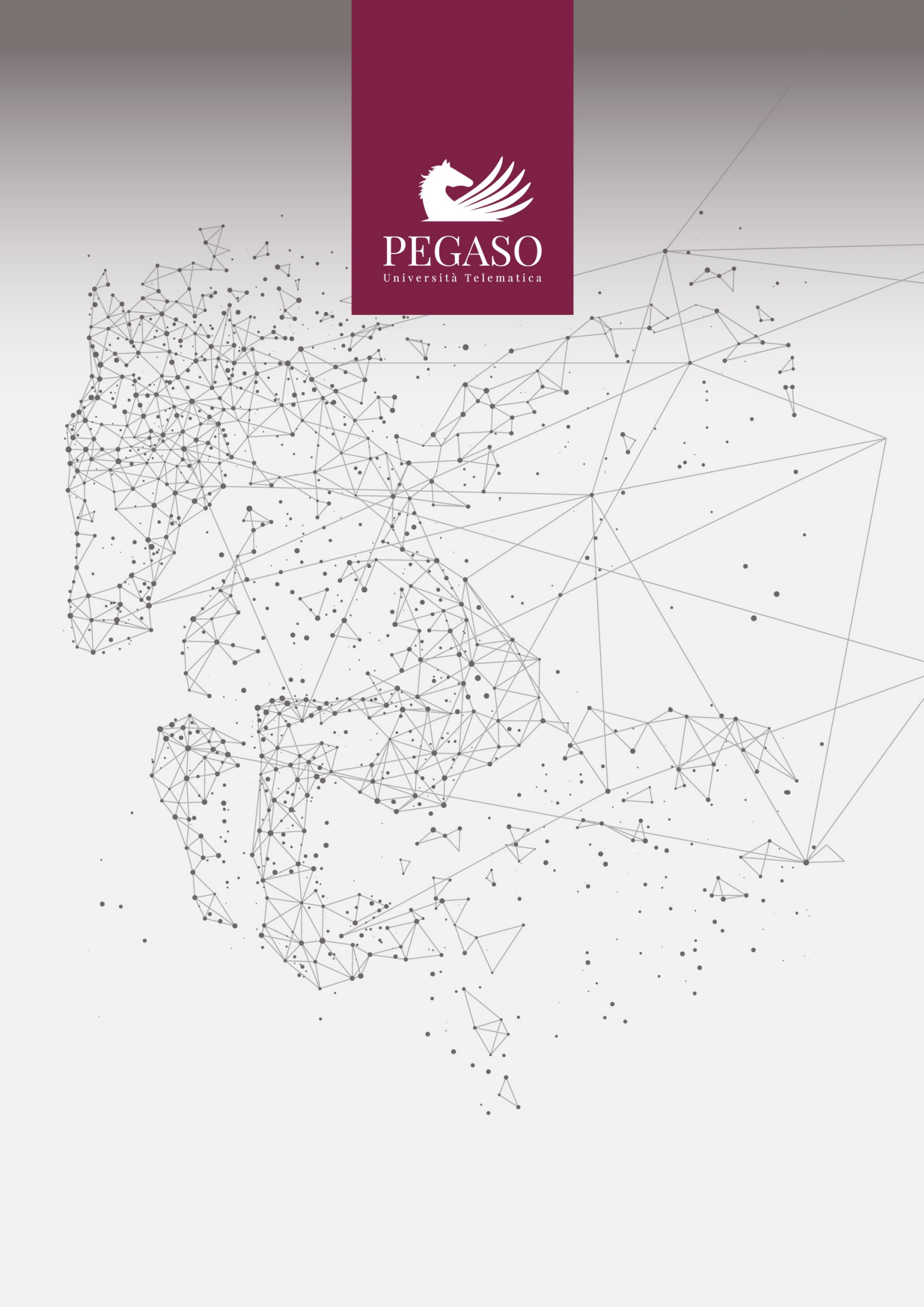
PEGASO
Università Telematica

Indice

1. Notazione asintotica

Gli algoritmi che risolvono uno stesso problema (decidibile) vengono confrontati sulla base della loro efficienza, cioè, in base al loro tempo d'esecuzione; tale tempo di esecuzione è e espresso come una funzione della dimensione dei dati di ingresso.

La dimensione n dei dati di ingresso dipende dallo specifico problema e può essere rappresentata dal numero stesso di dati di ingresso oppure dalla quantità di memoria necessaria per contenere tali dati.

Tuttavia, la valutazione del tempo di esecuzione deve essere indipendente dalla tecnologia dell'esecutore e dunque non può essere misurato in una unità temprale come i "secondi" o il tempo di CPU. Per questo motivo si usa come unità di misura il numero di passi base compiuti durante l'esecuzione dell'algoritmo; un passo base può essere:

- l'esecuzione di un'istruzione di assegnamento (priva di chiamate di funzioni);
- la valutazione di un'espressione (priva di chiamate di funzioni) contenuta in un'istruzione di selezione;
- la valutazione di un'espressione (priva di chiamate di funzioni) contenuta in un'istruzione di ripetizione.

È utile a questo punto valutare il tempo d'esecuzione in tre diversi casi:

- **Caso pessimo**: determinato dall'istanza dei dati di ingresso che massimizza il tempo d'esecuzione e quindi fornisce un limite superiore alla quantità di risorse computazionali necessarie all'algoritmo.
- **Caso ottimo**: determinato dall'istanza dei dati di ingresso che minimizza il tempo d'esecuzione e quindi fornisce un limite inferiore alla quantità di risorse computazionali necessarie all'algoritmo.
- **Caso medio**: determinato dalla somma dei tempi d'esecuzione di tutte le istanze dei dati di ingresso, con ogni addendo moltiplicato per la probabilità di occorrenza della relativa istanza dei dati di ingresso.

Si introduce il concetto di "complessità asintotica" (e dunque di "notazione asintotica" in ordine di grandezza) dal momento che il confronto degli algoritmi che risolvono lo stesso problema decidibile si riduce al confronto di funzioni (che possono relazionarsi in modi diversi per istanze diverse dei dati di ingresso): tale notazione ci consente dunque di comparare il tasso di crescita (cioè, il comportamento asintotico) di una funzione nei confronti di un'altra.

Esistono tre notazioni asintotiche:

- **Notazione asintotica O** (notazione O grande): limite superiore asintotico.
- **Notazione asintotica Ω** (notazione Omega): limite inferiore asintotico.

- **Notazione asintotica θ** (notazione Theta): limite asintotico stretto.
- La **notazione asintotica** O è il limite superiore asintotico.

  Date due funzioni $f(n)$ e $g(n)$, si dice che $f(n) = O(g(n))$ se esiste un valore $c > 0$ tale che $0 \leq f(n) \leq c \cdot g(n)$ per ogni $n \geq n_0$.

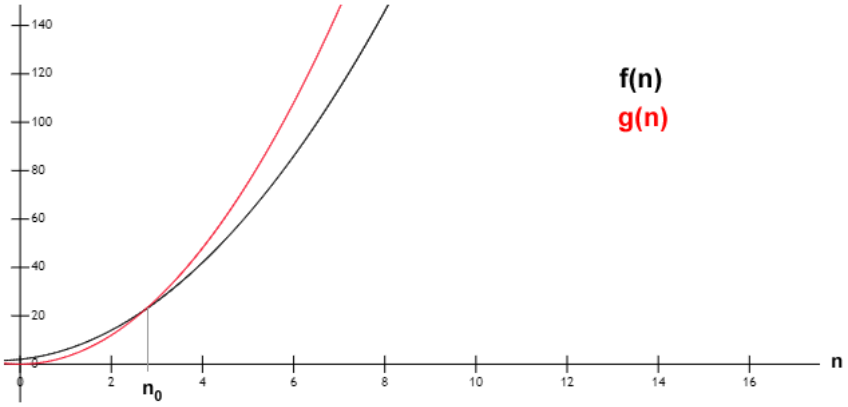

Figura 1 *- Notazione asintotica O*

La **notazione asintotica** Ω è il limite asintotico inferiore.

Date due funzioni $f(n)$ e $g(n)$, si dice che $f(n) = \Omega(g(n))$ se esiste un valore $c > 0$ tale che $0 \leq c \cdot g(n) \leq f(n)$ per ogni $n \geq n_0$.

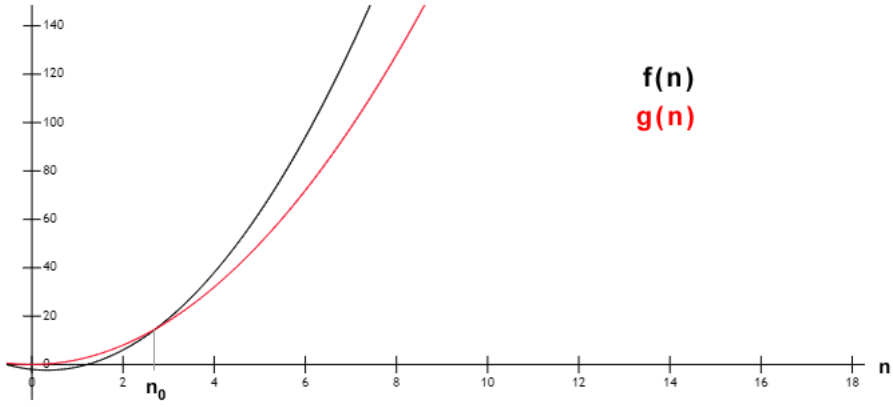

Figura 2 *- Notazione asintotica Omega*

La **notazione asintotica** $\boldsymbol{\theta}$ è il limite asintotico stretto.

Date due funzioni $f(n)$ e $g(n)$, si dice che $f(n) = \theta(g(n))$ se esistono due valori $c_1 > 0$ e $c_2 > 0$ tali che $0 \leq c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$ per ogni $n \geq n_0$.

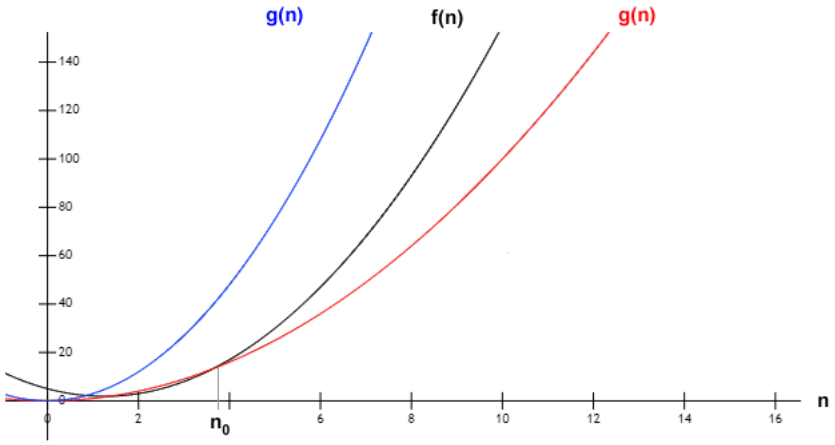

Figura 3 *- Notazione asintotica Theta*

Dal momento che le funzioni tempo d'esecuzione dei vari algoritmi che risolvono lo stesso problema decidibile vengono confrontate considerando il loro andamento al crescere della dimensione n dei dati di ingresso, all'interno delle funzioni, si può intuitivamente comprendere il perché si possano ignorare le costanti moltiplicative e i termini non dominanti al crescere di n.

2. Calcolo della complessità degli algoritmi non ricorsivi

Valutiamo il tempo di esecuzione per le seguenti istruzioni nei due casi di interesse:

- senza chiamate a funzioni;
- con chiamate a funzioni.

Assegnazione

- se non ci sono chiamate a funzioni: $T(n) = 1 \rightarrow O(1)$
- se ci sono chiamate a funzioni con $T'(n) = O(f'(n))$: $T(n) = 1 + T'(n) \rightarrow O(f'(n))$

Sequenza di istruzioni

Se abbiamo m istruzioni $S_1, S_2, \ldots, S_m$ con $m \geq 2$, ciascuna con tempo di esecuzione $T_i(n) = O(f_i(n))$ con $1 \leq i \leq m$:

$$T(n) = \sum_{i=1}^{m} T_i(n) = O(\max\{f_i(n) | 1 \leq i \leq m\}$$

Selezione

Il tempo di esecuzione di una istruzione di selezione del tipo:

IF $condizione$

THEN S_1

ELSE S_2

ENDIF

Essendo:

- $T_{condizione}(n) = O(f_{condizione}(n))$
- $T_{S_1}(n) = O(f_{S_1}(n))$
- $T_{S_n}(n) = O(f_{S_n}(n))$

Si calcola come segue:

$$T(n) = T_{condizione}\ (n) + eval\left(T_{S_1}(n), T_{S_2}(n)\right)$$
$$= O(\max\left\{f_{condizione}\ (n), eval\left(f_{S_1}(n), f_{S_2}(n)\right)\right\})$$

eval vale *min* o *max* a seconda se stiamo considerando il caso ottimo o pessimo rispettivamente

Iterazione

Il tempo di esecuzione di una istruzione di iterazione del tipo:

WHILE *condizione*

S

ENDWHILE

Essendo:

- numero di iterazioni: $i(n) = O(f(n))$
- $T_{condizione}(n) = O(f_{condizione}(n))$
- $T_s(n) = O(f_s(n))$

Si calcola come segue:

$$T(n) = i(n) \cdot \left(T_{condizione}(n) + T_s(n)\right) + T_{condizione}(n) = O(f(n) \cdot \max\{f_{condizione}(n), f_s(n)\})$$

Nel caso in cui i tempi $T_{condizione}(n)$ o $T_s(n)$ dipendano dalla specifica iterazione j con $1 \leq j \leq m$ allora abbiamo:

- $T_{condizione}(n, j)$
- $T_s(n, j)$

E ne consegue che:

$$T(n) = \sum_{j=1}^{i(n)} \left(T_{condizione}(n, j) + T_s(n, j)\right) + T_{condizione}(n, i(n) + 1)$$

3. Analisi della complessità - Esempi

Calcolo del fattoriale

```
#include <iostream>

using namespace std;

int fattoriale_non_ricorsivo(int n) {
  int fatt=1;
  for (int i = 2;i<=n;i++)
    fatt*=i;
  return(fatt);
}

int main() {
  int n=5;
  cout<<fattoriale_non_ricorsivo(n)<<endl;
}
```

Figura 4 *- Calcolo del fattoriale*

Analizziamo *T(n)*:

- L'inizializzazione della variabile $fatt = 1$ ha tempo costante
- Il ciclo for si deve eseguire *n-1* volte e ad ogni ciclo si eseguono 3 istruzioni:
  - Inizializzazione (*i=2* → solo la prima volta)
  - Confronto (*i<=n*)
  - Istruzione (*fatt*=n*)
  - Incremento (*i++* → tranne l'ultima volta)

- La restituzione finale *return(fatt)* ha tempo costante

$$T(n) = 1 + (n-1)\cdot(1+1+1) + 1 = 3\cdot n - 1 = O(n)$$

Calcolo di Fibonacci

```cpp
#include <iostream>

using namespace std;

int fibonacci_non_ricorsivo(int n) {
  if ((n==1) || (n==2))
    return 1;

  int ultimo=1,penultimo=1,fib;

  for (int i=3;i<=n;i++) {
    fib=ultimo+penultimo;
    penultimo=ultimo;
    ultimo=fib;
  }

  return(fib);
}

int main() {
  int n=7;
  cout<<fibonacci_non_ricorsivo(n)<<endl;
}
```

Figura 5 *- Calcolo di Fibonacci*

In questo caso di può verificare che nel caso $n \geq 3$ si ha:

$$T(n) = 1 + 1 + (n-2) \cdot (1 + 1 + 1 + 1 + 1) + 1 = 5 \cdot n - 7 = O(n)$$

mentre nel caso $n \leq 2$ ha complessitµa:

$$T(n) = 1 + 1 = 2 = O(1)$$

Calcolo del massimo

```cpp
#include <iostream>

using namespace std;

int massimo_non_ricorsivo(int a[],int n) {
  int max=a[0];
  for (int i = 1;i < n;i++)
    if (a[i] > max)
      max = a[i];
  return(max);
}

int main() {
  int a[]={1,3,4,5,7,8,3,2,1,67};
  int n=10;
  cout<<massimo_non_ricorsivo(a,n)<<endl;
}
```

Figura 6 *- Calcolo del massimo*

Nel caso peggiore (il massimo è alla fine) si ha:

$$T(n) = 1 + (n-1) \cdot (1+1+1+1) + 1 = 4 \cdot n - 2 = O(n)$$

Nel caso migliore (il massimo è all'inizio) si ha:

$$T(n) = 1 + (n-1) \cdot (1+1+1) + 1 = 3 \cdot n - 1 = O(n)$$

Come ci si può facilmente rendere conto, avere il massimo all'inizio o alla fine non comporta vantaggi in termini di complessità.

Bibliografia

- Alan Bertossi, Alberto Motresor: Algoritmi e strutture di dati, Città Studi Edizioni, terza edizione
- C. Demetrescu, I. Finocchi, G. F. Italiano: Algoritmi e strutture dati, McGraw-Hill, seconda edizione
- Crescenzi, Gambosi, Grossi: Strutture di Dati e Algoritmi, Pearson/Addison-Wesley
- Sedgewick: Algoritmi in C, Pearson, 2015