

# INFORMATICA GENERALE II

Ingegneria delle Telecomunicazioni

Università di Trento

A.A. 2003/2004

II Bimestre

**Marco Roveri**

[roveri@irst.itc.it](mailto:roveri@irst.itc.it)

Analisi algoritmi

# Analisi Algoritmi

- Si progetti un programma che verifica se un numero  $v$  occorre all'interno di un insieme di numeri memorizzati in un array  $A$ . Se troviamo l'occorrenza ritorniamo l'indice dove occorre, altrimenti ritorniamo -1.
- Esempio:  $v=4$ 
  - $A[4] = \{1, 2, 5, 4\}$ ;
    - Output: 3
  - $A[4] = \{1, 3, 6, 2\}$ ;
    - Output: -1

# Analisi Algoritmi

- Una prima soluzione è:

```
int search(int A[], int v, int N) {  
    for(int i = 0; i < N; i++)  
        if ( v == A[i] ) return i;  
    return -1;  
}
```

- Quale è la bontà di questa soluzione?

# Come analizzare la bontà di un algoritmo

- Correttezza
  - Dimostrazione formale (matematica).
  - Ispezione informale.
- Utilizzo delle risorse
  - *Tempo di esecuzione.*
  - *Utilizzo della memoria.*
- Semplicità
  - Facile da capire e mantenere.

# Tempo di esecuzione

- Tempo necessario al programma per terminare.
- Dipende da molteplici fattori:
  - Hardware (velocità del processore, bus, disco..)
  - Compilatore (ottimizzazioni che il compilatore può effettuare, utilizzo di istruzioni speciali, ...)
  - Dimensione del problema di input (nel caso precedente N).

# Analisi Algoritmo search

- La ricerca sequenziale esamina N numeri per ricerca con esito negativo, e mediamente N/2 numeri per ricerca con esito positivo.
- Se tutti i numeri hanno uguale probabilità di essere quello cercato, e assumendo che il costo medio del confronto tra due numeri sia costante, allora:

$$(1+2 + \dots + N) / N = (N + 1)/2$$

è il costo medio dell'algoritmo.

# Analisi Algoritmo search

- Il costo dell'algoritmo di ricerca sequenziale è proporzionale a  $N$ , dimensione del problema.
  - Raddoppiando  $N$  mi aspetto un raddoppio del tempo di calcolo.
- Se dobbiamo effettuare  $M$  ricerche, il costo di tali ricerche sarà proporzionale a  $MN$ .
  - Se per confrontare due numeri occorrono  $c$  microsecondi,  $N = 10^6$  e  $M = 10^9$  il tempo medio richiesto sarà  $(c/2)10^9$  secondi, cioè circa  $16c$  anni 😞

# Algoritmo di ricerca binaria

- Assunzione: array  $A$  ordinato, ovvero per ogni  $1 \leq i \leq N - 1$ ,  $A[i - 1] \leq A[i]$
- Sotto questa ipotesi possiamo eliminare metà array, confrontando il numero da cercare con l'elemento al centro dell'array:
  - Se questi numeri sono uguali abbiamo finito.
  - Se il primo è minore del secondo, applichiamo stesso metodo alla metà di sinistra.
  - Altrimenti applichiamo stesso metodo alla metà di destra.

# Algoritmo di ricerca binaria

```
int binsearch( int A[], int v, int N) {  
    int l = 0, r = N-1;  
    while (r ≥ l) {  
        int m = (r + l) / 2;  
        if (v == A[m]) return m;  
        if (v < A[m])  
            r = m - 1;  
        else  
            l = m + 1;  
    }  
    return -1;  
}
```

Versione iterativa

# Algoritmo di ricerca binaria

```
int binsearch(int A[], int N, int v) {  
    return binsearch_recur(A, 0, N-1, v);  
}
```

```
int binsearch_recur(int A[], int l, int r, int v) {  
    int m = (l + r) / 2;  
  
    if (v == A[m]) return m;  
    if (v < A[m]) return binsearch_recur(A, l, m-1, v);  
    if (v > A[m]) return binsearch_recur(A, m+1, r, v);  
    return -1;  
}
```

Versione ricorsiva

# Analisi algoritmo ricerca binaria

- Se  $T_N$  è il numero di confronti richiesti nel caso peggiore da una ricerca binaria, il modo di ridurre un problema di dimensione  $N$  ad uno di dimensione  $N/2$  ci porta a:

$$T_N \leq T_{N/2} + 1 \quad \text{per } N \geq 2, T_1 = 1$$

Assumendo  $N = 2^n$  è facile mostrare che

$$T_N \leq n + 1 = \lg N + 1$$

# Analisi algoritmo di ricerca binaria

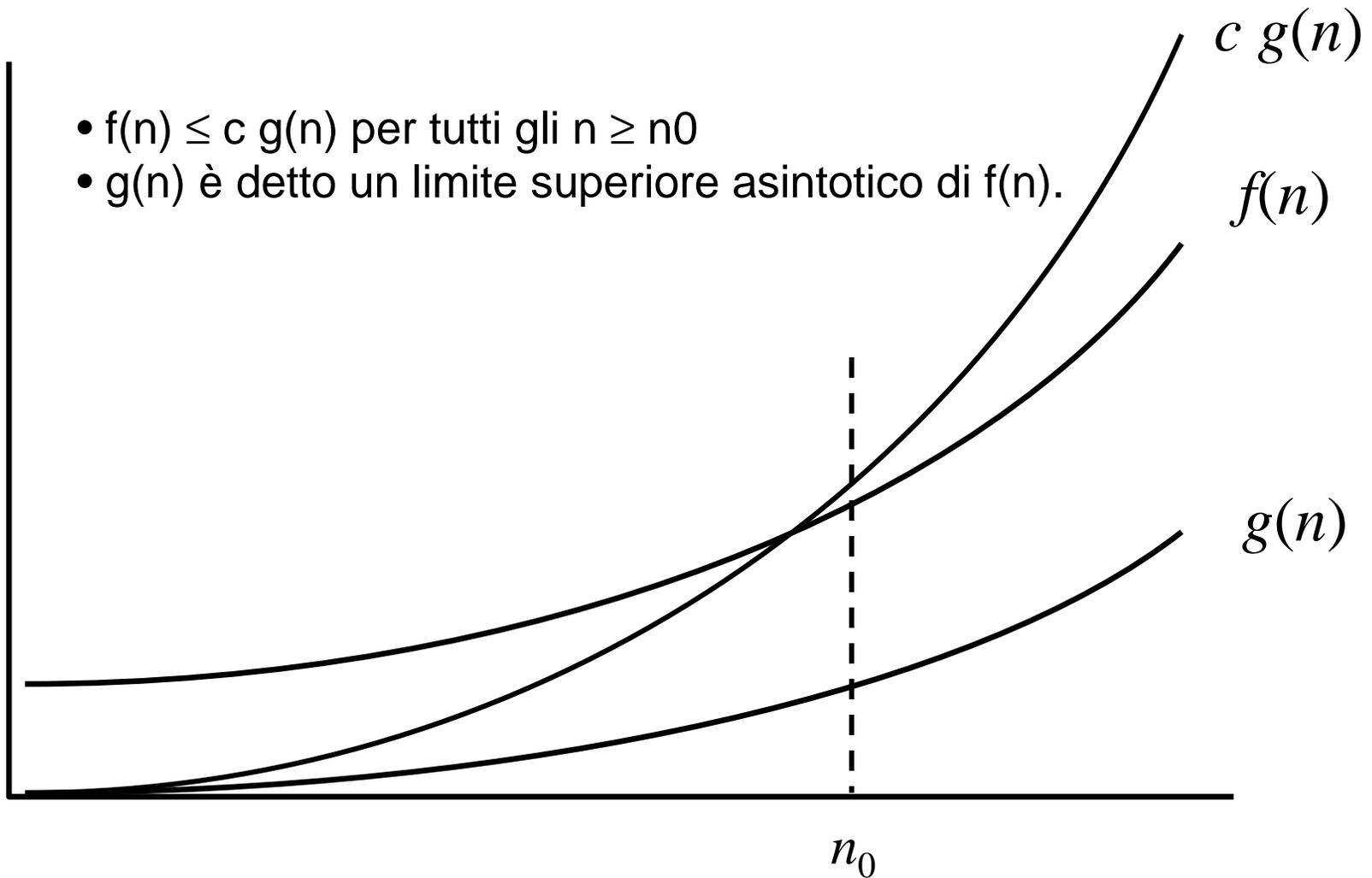
- Se dobbiamo effettuare  $M$  ricerche, il costo di tali ricerche sarà proporzionale a  $M \lg N$ .
- Se per confrontare due numeri occorrono  $c$  microsecondi,  $N = 10^6$  e  $M = 10^9$  il tempo medio richiesto sarà circa  $2c10^4$  secondi, cioè circa  $5.5c$  ore,  $\ll 16c$  anni 😊

# Notazione O

- Una indicazione del tempo medio di esecuzione di un algoritmo la si fornisce con la cosiddetta *notazione O*, o *limite superiore asintotico*.
  - $f(n) \leq c g(n)$  per tutti gli  $n \geq n_0$
  - $g(n)$  è detto un limite superiore asintotico di  $f(n)$ .
  - Scriviamo  $f(n) = O(g(n))$
  - Leggiamo  $f(n)$  è O-grande di  $g(n)$ .

# Notazione O

- $f(n) \leq c g(n)$  per tutti gli  $n \geq n_0$
- $g(n)$  è detto un limite superiore asintotico di  $f(n)$ .



# Notazione O

- In genere quando impieghiamo la notazione O, utilizziamo la formula più “semplice”. La notazione O() fornisce un limite superiore. Non rappresenta la funzione precisa del tempo di esecuzione.
- Esempio

$$3n^2+2n+5 = O(n^2)$$

- Funzioni classiche in ordine di grandezza crescente:
  - O(1) costante
  - O(lg N) logaritmico
  - O(N) lineare
  - O(N lg N) pseudolineare
  - O(N<sup>2</sup>) quadratico
  - O(N<sup>3</sup>) cubico
  - O(2<sup>N</sup>) esponenziale
  - O(N!) fattoriale
  - O(N<sup>N</sup>)

# Stima del tempo di esecuzione

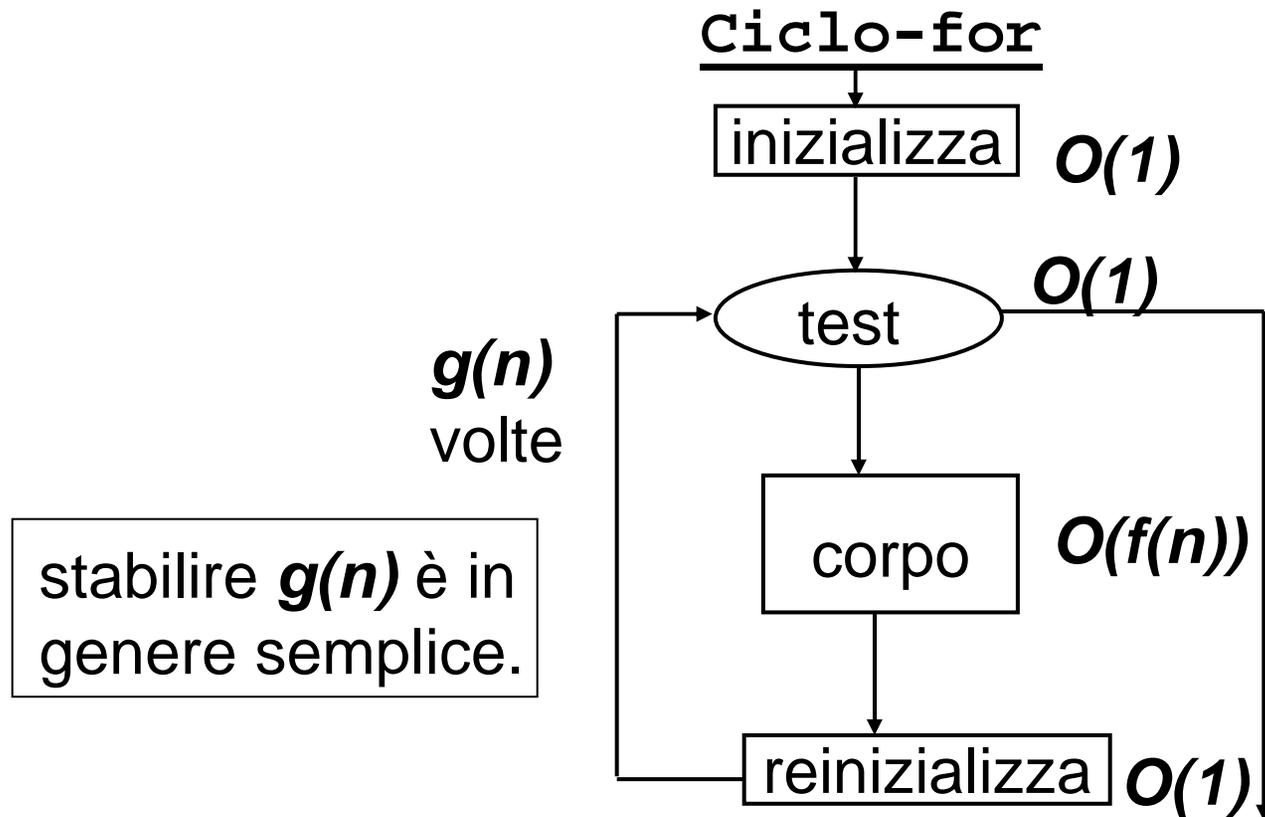
- Nel seguito descriveremo un modo semplice per *stimare* il limite asintottico superiore  $O()$  del tempo di esecuzione per diversi tipi di algoritmi:
  - *Iterativi*
  - *ricorsivi*

# Tempo di esecuzione: *operazioni semplici*

- operazioni aritmetiche (+, \*, ...)
- operazioni logiche(&&, ||, ...)
- confronti ( $\leq$ ,  $\geq$ , =, ...)
- assegnamenti (a = b) senza chiamate di funzione.
- operazioni di lettura (cin)
- operazioni di controllo (break, continue, return)

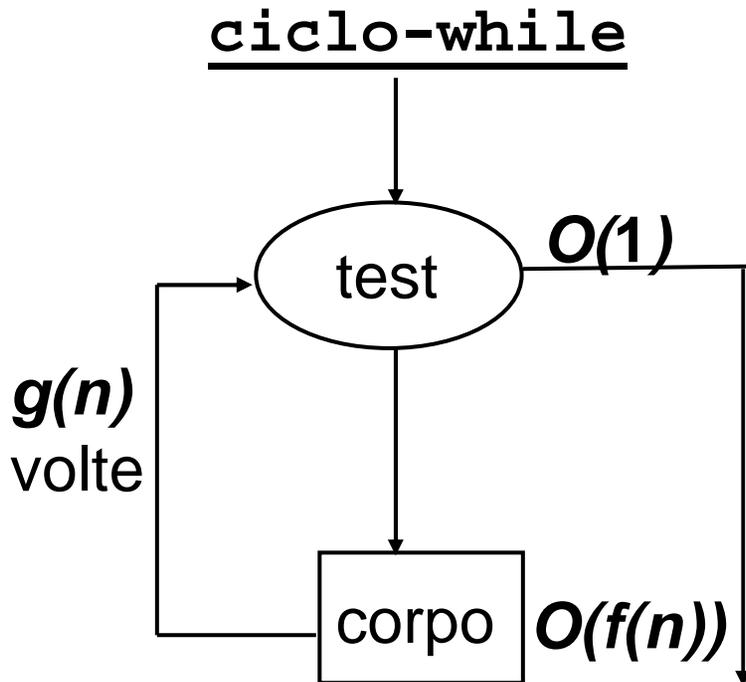
$$T(n) = O(1)$$

# Tempo di esecuzione: ciclo "for"



$$T(n) = O(g(n) \times f(n))$$

# Tempo di esecuzione: ciclo while

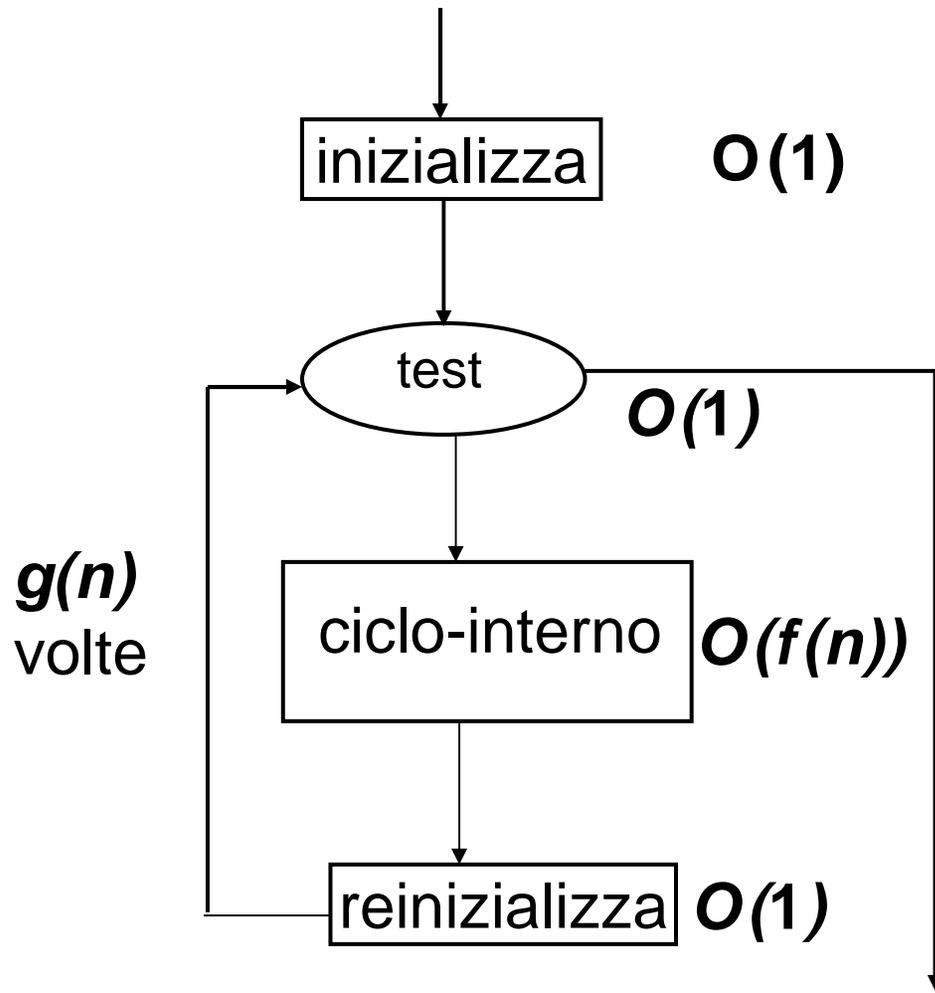


Bisogna stabilire un limite per il numero di iterazioni del ciclo,  $g(n)$ .

Può essere necessaria una prova induttiva per  $g(n)$ .

$$T(n) = O(g(n) \times f(n))$$

# Tempo di esecuzione: cicli innestati



$$T(n) = O(g(n) \times f(n))$$

# Esempio

```
for(int i = 1; i <= n; i++)  
  for(int j = 1; j <= n; j++)  
    k = i + j;
```

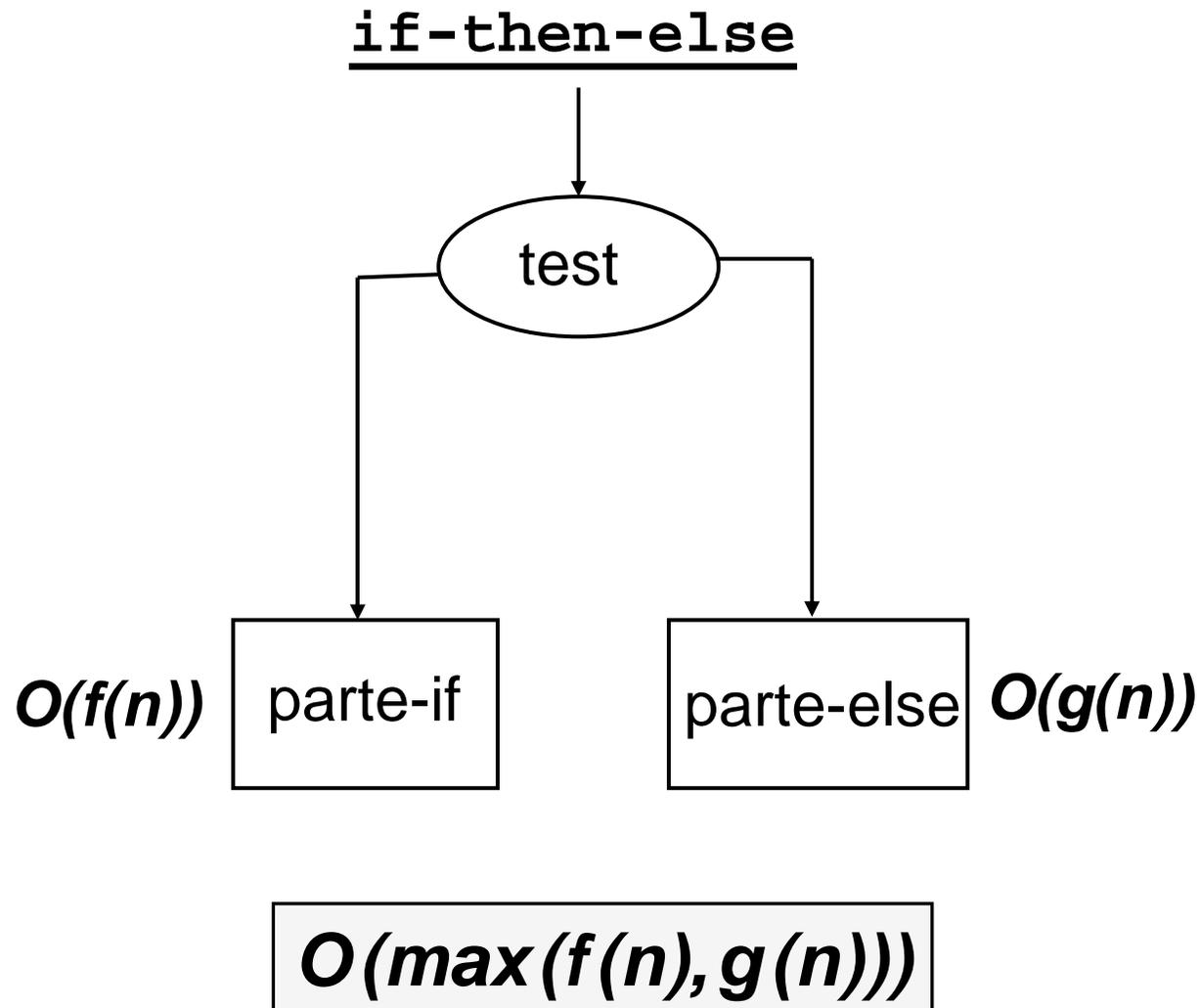
} =  $O(n)$  } =  $O(n^2)$

```
for(int i = 1; i <= n; i++)  
  for(int j = i; j <= n; j++)  
    k = i + j;
```

} =  $O(n - i)$  } =  $O(n^2)$

$$T(n) = O(n \times n) = O(n^2)$$

# Tempo di esecuzione: if-then-else



# Esempio

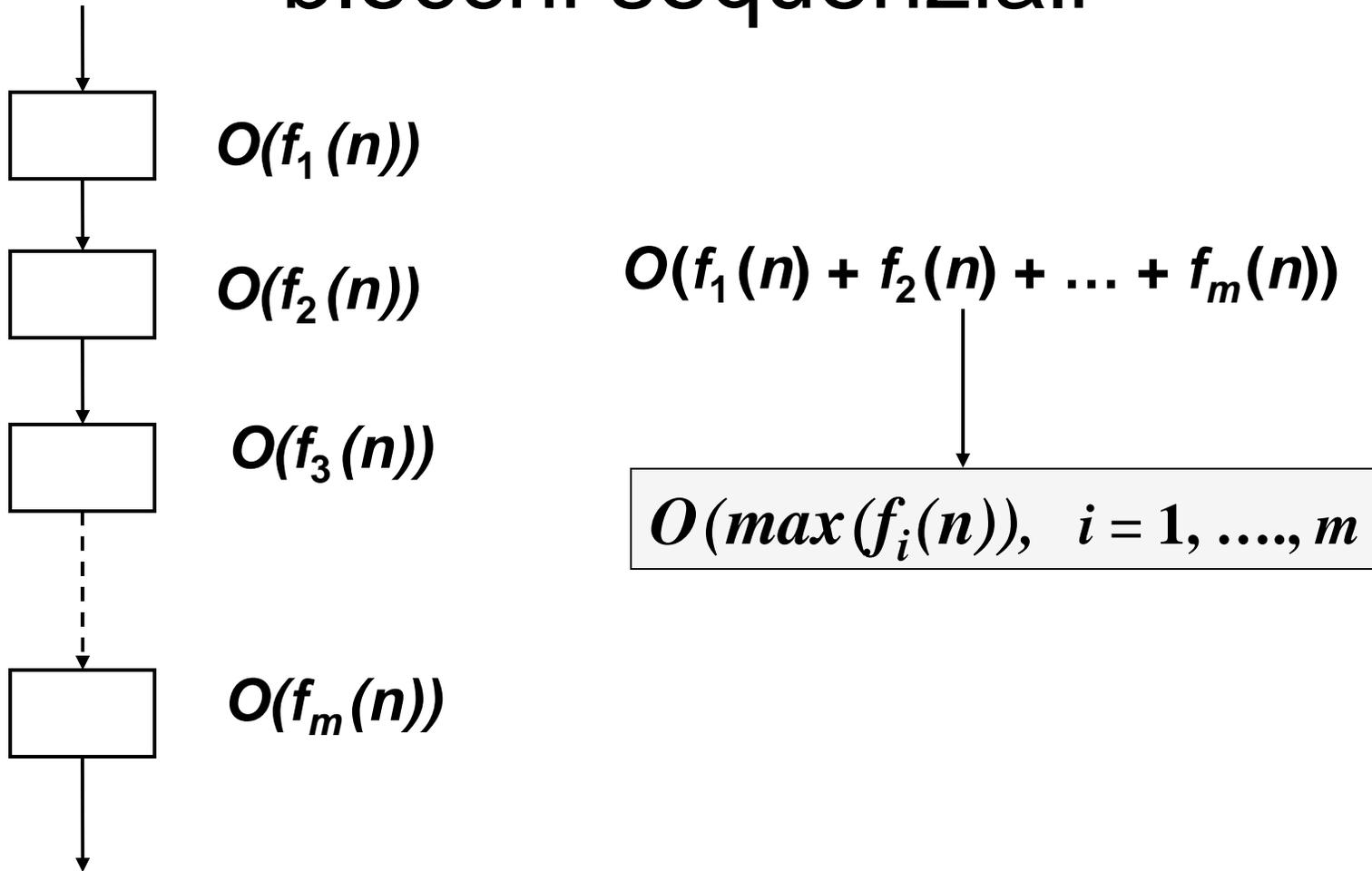
```
if (A[i][j] == 0)
  for(int i = 1; i <= n; i++)
    for(int j = i; j <= n; j++)
      A[i][j] = 0;
else
  for(int i = 1; i <= n; i++)
    A[i][0] = 1;
```

$\left. \begin{array}{l} \text{for(int i = 1; i <= n; i++)} \\ \text{for(int j = i; j <= n; j++)} \end{array} \right\} = O(n - i) \left. \vphantom{\begin{array}{l} \text{for(int i = 1; i <= n; i++)} \\ \text{for(int j = i; j <= n; j++)} \end{array}} \right\} = O(n^2)$

$\left. \begin{array}{l} \text{for(int i = 1; i <= n; i++)} \\ \text{A[i][0] = 1;} \end{array} \right\} = O(n)$

$$T(n) = \max(O(n^2), O(n)) = O(n^2)$$

# Tempo di esecuzione: blocchi sequenziali



# Tempo di esecuzione: algoritmi ricorsivi

- Il tempo di esecuzione è espresso mediante una *equazione di ricorrenza*.
  - Esempio:

$$\text{Binsearch : } T(n) = \begin{cases} O(1) & \text{se } n = 1 \\ T(n/2) + O(1) & \text{se } n > 1 \end{cases}$$

- Sono necessarie tecniche specifiche per risolvere le equazioni di ricorrenza.

# Soluzione di equazioni di ricorrenza

- Esistono molti metodi per risolvere le equazioni di ricorrenza:
  - Metodo iterativo
    - Si itera la regola induttiva di  $T(n)$  in termini di  $n$  e del caso base.
    - Si richiede poi manipolazione delle somme:  $\sum_1^n i = n(n+1)/2$
  - Il metodo di sostituzione
    - Si ipotizza una possibile soluzione.
    - Si sostituisce l'ipotetica soluzione nei casi base e induttivo.
    - Si dimostra la correttezza della ipotesi tramite induzione matematica.

# Metodo iterativo

$$\begin{aligned} T(n) &= T(n/2) + c = T(n/4) + c + c = \\ &= T(n/8) + c + c + c \end{aligned}$$

Alla  $i$ -esima iterazione abbiamo:

$$T(n) = T(n/2^i) + ic$$

L'iterazione raggiunge 1 quando,  $n = 2^i$ , ovvero quando  $i = \log(n)$ , quindi

$$T(n) = c + \log(n)c$$

E quindi possiamo dire che  $T(n) = O(\log(n))$