PROGRAMMAZIONE AVANZATA ed ELEMENTI DI INGEGNERIA DEL SOFTWARE

Complessità Computazionale

Principali qualità del software

- Correttezza
- Affidabilità
- Robustezza
- <u>**Prestazioni**</u>
- Usabilità
- Verificabilità
- **Manutenibilità**
- **Riusabilità**
- **Portabilità**
- Comprensibiltà
- Interoperabilità
- Produttività
- Tempestività
- Visibilità

Prestazioni

Per misurare l'efficienza di un algoritmo in maniera univoca, bisogna definire una metrica indipendente dalle tecnologie utilizzate, altrimenti uno stesso algoritmo potrebbe avere efficienza diversa a seconda della tecnologia sulla quale è eseguito.

Risorsa tempo: si dice che la macchina M opera in tempo f(*n*) se dato un input di lunghezza *n*, produce il risultato in f(*n*) passi.

Si misura il tempo in passi, indipendentemente dal tempo necessario per eseguire ciascun passo.

L'ipotesi è che ciascun passo 'elementare' abbia lo stesso costo costante

Risorsa spazio: si dice che la macchina M opera in spazio f(*n*) se dato un input di lunghezza *n*, utilizza f(*n*) celle temporanee (oltre ad input ed output) per effettuare la computazione.

Notazione O

Poiché calcolare f(n) è complicato, ci si riferisce in genere al limite asintotico superiore.

$f(n) = O(g(n))$ se

$\exists(n_0, c)$ tali che $\forall n > n_0$

$f(n) \leq cg(n)$

La funzione f(*n*) **da un certo *n* in poi** cresce al più come la funzione g(*n*).

Diremo che **l'algoritmo ha complessità $\boldsymbol{O(g(n))}$** (dell'ordine di g(n))

Notazione O

Esempio

$$f(n) = 3\,n^2 - 4n + 1000$$

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

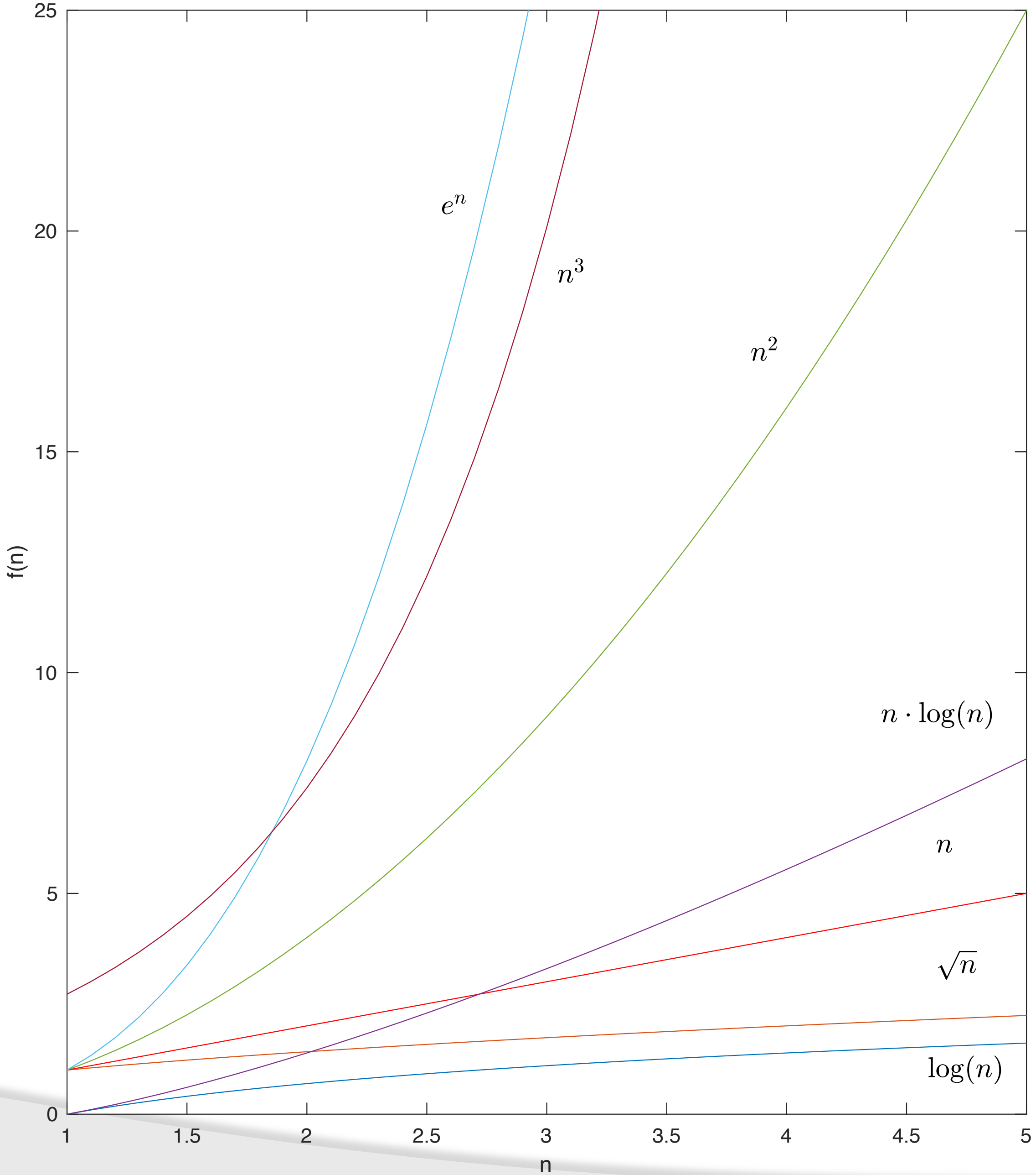
infatti

$$f(n) \leq c\,n^2$$

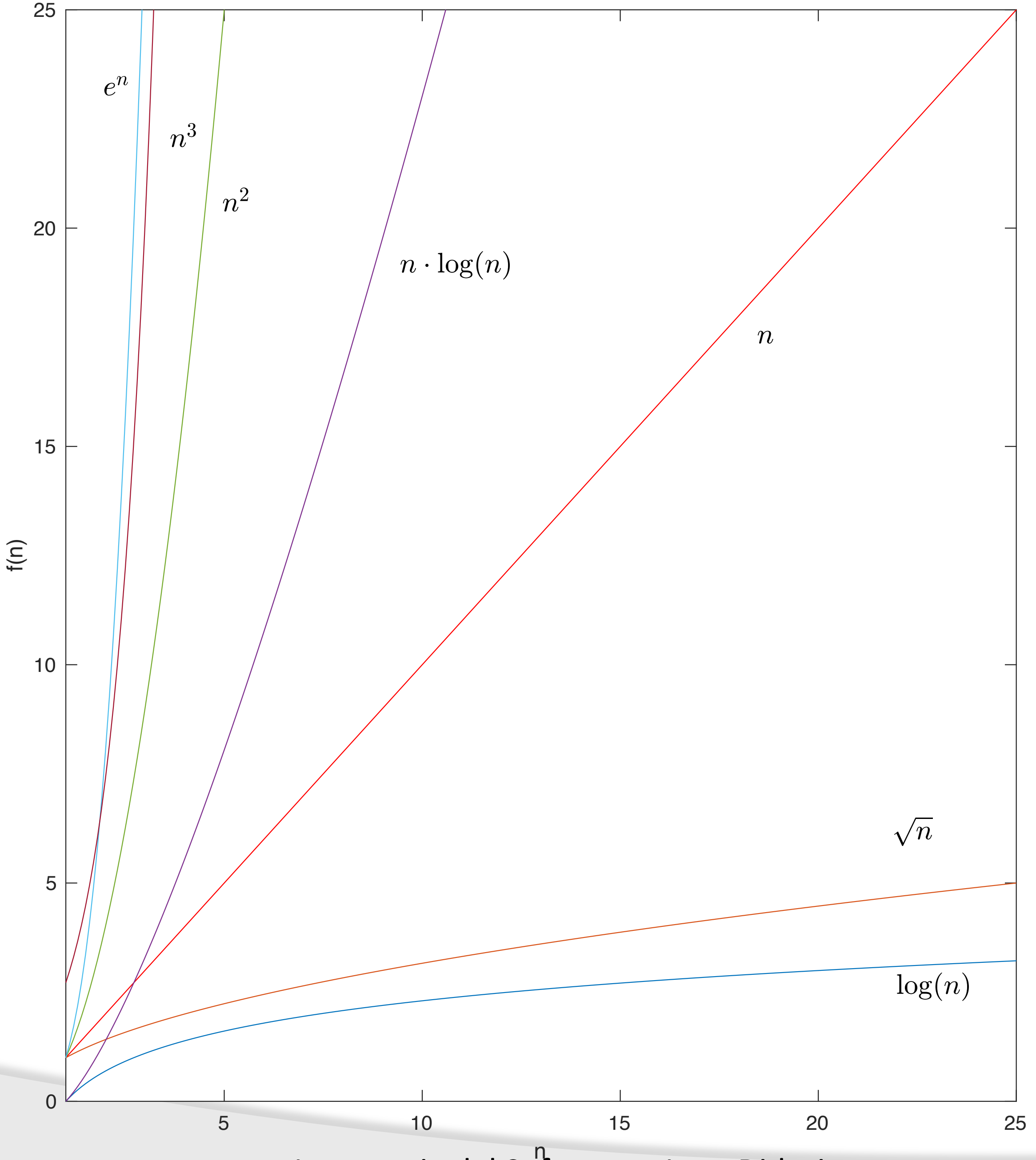
da un certo *n* in poi.

L'algoritmo ha complessità $\boldsymbol{O}(\boldsymbol{n^2})$

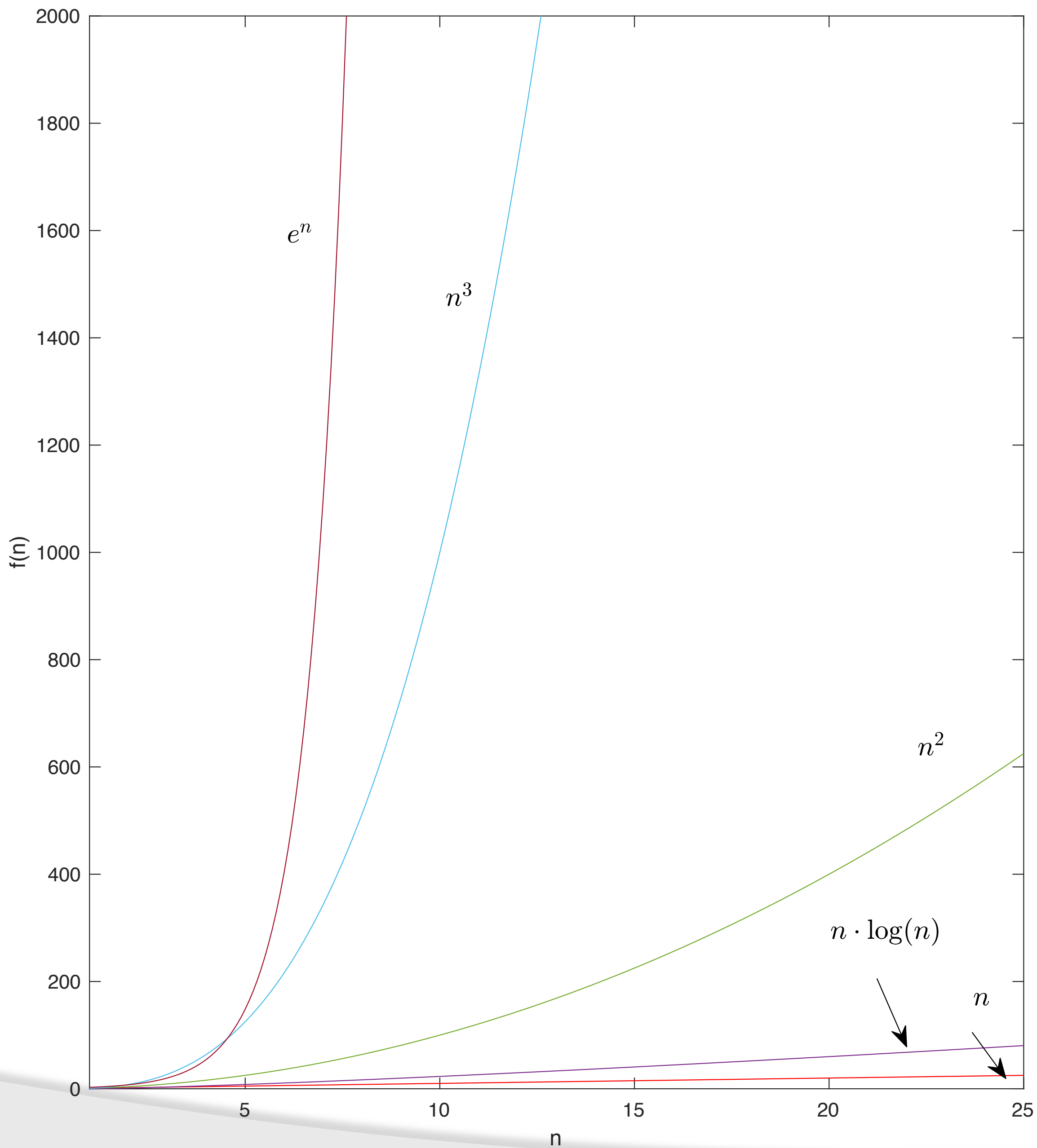
Complessità computazionale



Complessità computazionale



Complessità computazionale



È utile distinguere il caso ottimo, il caso peggiore e il caso medio.

- Caso ottimo: i dati sono i 'migliori' possibili per l'algoritmo, cioè quelli che richiedono meno elaborazioni
- Caso peggiore: i dati che richiedono il massimo numero di passi per l'algoritmo.
- Caso medio: è il caso più utile da analizzare perché fornisce un reale indicatore della complessità dell'algoritmo. Spesso è difficile determinare quali sono i dati medi. Si può utilizzare una stima empirica dei tempi.

Esempio

```
i=0;
while (i<n){
   i=i+1;
}
```

Esempio

```
i=0;
while (i<n){
   i=i+1;
}
```

Assegnamento esterno al ciclo: 1
Numero di test: n+1
Iterazioni totali: n
Computazioni interne al ciclo: 1 per ogni iterazione

$$f(n) = 1 + (n + 1) + n = 2n + 2$$

$O(n)$

Esempio

```
i=0;
while (i<n){
   … (k assegnamenti)
}
```

Esempio

```
i=0;
while (i<n){
   … (k assegnamenti)
}
```

Assegnamento esterno al ciclo: 1
Numero di test: n+1
Iterazioni totali: n
Computazioni interne al ciclo: k per ogni iterazione

$$f(n) = 1 + (n + 1) + kn = (k + 1)n + 2$$

O(n)

Esempio

```
i=0;
while (i<2*n){
   … (k assegnamenti)
}
```

Esempio

```
i=0;
while (i<2*n){
   … (k assegnamenti)
}
```

Assegnamento esterno al ciclo: 1
Numero di test: 2n+1
Iterazioni totali: 2n
Computazioni interne al ciclo: k per ogni iterazione

$$f(n) = 1 + (2n + 1) + k \cdot 2n = 2(k + 1)n + 2$$

O(n)

Esempio

```
i=0;
while (i<n){
    for(j=0; j<n; j++){
   … (k assegnamenti)
    }
}
```

Esempio

```
i=0;
while (i<n){
    for(j=0; j<n; j++){
   … (k assegnamenti)
    }
}
```

Assegnamento esterno al ciclo: 1

while numero di test: n+1; iterazioni: n

for numero di test: n+1; iterazioni: n

Computazioni interne al ciclo: k per ogni iterazione

$f(n) = 1 + (n+1) + n(n+1) + k \cdot n^2 =$

$= (k+1) \cdot n^2 + 2n + 2$

$O(n^2)$

Dipendenza dal valore dell'input

Il numero di passi può dipendere anche dal *valore* dell'input, non solo dalla *dimensione*

```
if (condizione){istruzioni}
```

```
if (condizione){
    corpo1
   }
else {
    corpo2
   }
```

Dipendenza dal valore dell'input

```
A={3,4,1,2,0}
```

Cerco l'elemento 1 con ricerca sequenziale

```
s=5;
i=0;
while(i<s && A[i] != 1)
  i = i+1;
```

Dipendenza dal valore dell'input

```
A={3,4,1,2,0}
```

Cerco l'elemento 1 con ricerca sequenziale

```
s=5;
i=0;
while(i<s && A[i] != 1)
  i = i+1;
```

Assegnazioni esterne al ciclo: 2

while: numero di test: 3; iterazioni: 2

Computazioni interne al ciclo: 1 per ogni iterazione

Passi base: 7

Dipendenza dal valore dell'input

```
A={3,4,1,2,0}
```

Cerco l'elemento 9 con ricerca sequenziale

```
s=5;
i=0;
while(i<s && A[i] != 9)
  i = i+1;
```

Dipendenza dal valore dell'input

```
A={3,4,1,2,0}
```

Cerco l'elemento 9 con ricerca sequenziale

```
s=5;
i=0;
while(i<s && A[i] != 9)
  i = i+1;
```

Assegnazioni esterne al ciclo: 2

while: numero di test: 6; iterazioni: 5

Computazioni interne al ciclo: 1 per ogni iterazione

Passi base: 13

Ricorrenze Fondamentali

Algoritmo che cicla su input **eliminando** un elemento per volta

$$\boldsymbol{C_N = C_{N-1} + N}; \qquad\qquad N \geq 2; C_1 = 1$$

$$\begin{aligned} C_N &= \\ &= C_{N-1} + N = \\ &= C_{N-2} + (N-1) + N = \\ &= C_{N-3} + (N-2) + (N-1) + N = \\ &\ldots \\ &= 1 + 2 + \cdots + (N-2) + (N-1) + N \quad = \\ &= \frac{N(N+1)}{2} \cong \frac{N^2}{2} \end{aligned}$$

Ricorrenze Fondamentali

Algoritmo che **dimezza** input ad ogni passo

$$\boldsymbol{C_N = C_{N/2} + 1;} \qquad N \geq 2; C_1 = 1 \qquad \boxed{N = 2^n}$$

$$C_N = C_{2^n} =$$
$$= C_{2^{(n-1)}} + 1 =$$
$$= C_{2^{(n-2)}} + (1 + 1) =$$
$$= C_{2^{(n-3)}} + (1 + 1 + 1) =$$

$$\dots$$

$$= C_{2^0} + n =$$
$$= n + 1 = \log_2 N + 1$$

Ricorrenze Fondamentali

Algoritmo che **dimezza** input ad ogni passo, ed **esamina** ogni el. di input

$$\boldsymbol{C_N = C_{N/2} + N}; \qquad N \geq 2; C_1 = 1$$

$$\boxed{N = 2^n}$$

$$C_N = C_{2^n} =$$
$$= C_{2^{(n-1)}} + N =$$
$$= C_{2^{(n-2)}} + (N + N/2) =$$
$$= C_{2^{(n-3)}} + (N + N/2 + N/4) =$$

$$\ldots$$

$$= N + \frac{N}{2} + \frac{N}{4} + \cdots \cong 2N$$

Usiamo le proprietà della serie geometrica

$$\sum_{k=0}^{n} \frac{1}{2^k} = 2\left(1 - \left(\frac{1}{2}\right)^{n+1}\right)$$

Ricorrenze Fondamentali

Formula generale divide et impera

$$\boldsymbol{C_N = 2\, C_{N/2} + N}; \qquad N \geq 2; C_1 = 0$$

$$\boxed{N = 2^n}$$

$$\frac{C_{2^n}}{2^n} = \frac{C_{2^{(n-1)}}}{2^{(n-1)}} + 1 = \frac{C_{2^{(n-2)}}}{2^{(n-2)}} + 2 =$$

$$= \frac{C_{2^{(n-n)}}}{2^{(n-n)}} + n = \frac{C_1}{2^0} + n = \log_2 N$$

$$C_N = N \ \log_2 N$$

Ricorrenze Fondamentali

Formula generale divide et impera

$$\boldsymbol{C_N = 2\ C_{N/2} + 1}; \qquad N \geq 2; C_1 = 0$$

$$\boxed{N = 2^n}$$

$$\frac{C_{2^n}}{2^n} = \frac{C_{2^{(n-1)}}}{2^{(n-1)}} + \frac{1}{2^n} = \frac{C_{2^{(n-2)}}}{2^{(n-2)}} + \frac{1}{2^n} + \frac{1}{2^{n-1}} =$$

$$= \frac{C_{2^{(n-n)}}}{2^{(n-n)}} + \sum_{i=0}^{n} \frac{1}{2^i} = \sum_{i=0}^{n} \frac{1}{2^i}$$

Usiamo le proprietà della serie geometrica

$$\sum_{k=0}^{n} \frac{1}{2^k} = 2\left(1 - \left(\frac{1}{2}\right)^{n+1}\right)$$

$$C_N \cong 2N$$

FONTI

Fonti:

- A. Bellini, A. Giudi, Linguaggio C
- Sedgewick, Algoritmi in C.
- Cormen, Leiserson, Rivest, Stein - Introduzione agli algoritmi e strutture dati
- Altre fonti rielaborate dal docente