

Costo computazionale

- Tempo di esecuzione
- Necessità di memoria

Il tasso di crescita del tempo che impieghiamo nell'algoritmo si divide in tre notazioni

- Worst case $O()$
 - abbiamo $f(n) = 3n$ e il suo $O(f(n)) = O(n)$ perchè abbiamo una $c \geq 3$ tale che ci ritroviamo con un valore maggiore da n_0 in poi " $n \geq n_0$ "
 $n \leq O(n)$ o $n \leq c \cdot n$ per un $n \geq n_0$ e con una $c \geq 3$
- Best case $\Omega()$
 - abbiamo $f(n) = 3n$ e il suo $\Omega(f(n)) = \Omega(n)$ perchè abbiamo una $c \leq 3$ tale che ci ritroviamo con un valore inferiore a $f(n)$ da n_0 in poi " $n \geq n_0$ "
 $n \geq \Omega(n)$ o $n \geq c \cdot n$ per un $n \geq n_0$ e con una $c \leq 3$
- Average case $\Theta()$
 - il caso medio ci permette di dimostrare che senza cambiare quello che c'è all'interno di Θ (un "generico") possiamo aggiungere anche $\Omega()$ e $O()$ quindi sappiamo che esiste una c' tale che $f(n) \leq c' \cdot n$ ed esiste una c'' tale che $f(n) \geq c'' \cdot n$ e questo vale per un certo punto n_0 abbastanza grande.

Proprietà e dimostrazione dell'algebra della notazione asintotica

1. $\forall k > 0$ se $f(n) \rightarrow O(g(n))$ allora anche $k \cdot f(n) \rightarrow O(g(n))$.

Dim

- sappiamo che esistono delle c tali che $f(n) \leq c \cdot g(n) \quad \forall n > n_0$
allora $k \cdot f(n) \leq k \cdot c \cdot g(n) \implies k \cdot f(n) \leq c' \cdot g(n) \quad k \cdot f(n) \leq k \cdot c \cdot g(n)$ e $c' = c \cdot k$

2. $\forall f(n), d(n) > 0$ se $f(n) \rightarrow g(n)$ e $d(n) \rightarrow h(n)$ allora

$$f(n) + d(n) \rightarrow (O(g(n) + h(n)) = O(\max(g(n), h(n))))$$

Dim

- sappiamo che abbiamo quattro costanti c', c'', n_0' e n_0'' tali che
 $f(n) \leq c' \cdot g(n) \quad \forall n > n_0'$ e $d(n) \leq c'' \cdot h(n) \quad \forall n > n_0''$ ed allora sappiamo che
 $f(n) + d(n) \leq c' \cdot g(n) + c'' \cdot h(n) \leq \max(c', c'') \cdot (g(n) + h(n)) \quad \forall n > \max(n_0', n_0'')$
cioè $f(n) + d(n) \leq c \cdot (g(n) + h(n)) \quad \forall n > n_0 \implies f(n) + d(n) \rightarrow O(g(n) + h(n))$
e per le proprietà della notazione asintotica sappiamo che dobbiamo prendere solo il massimo tra la somma di funzioni quindi
 $f(n) + d(n) \rightarrow O(\max(g(n), h(n)))$

3. $\forall f(n), d(n) > 0$ se $f(n) \rightarrow g(n)$ e $d(n) \rightarrow h(n)$ allora

$$f(n) \cdot d(n) \rightarrow O(g(n) \cdot h(n))$$

Dim

- sappiamo che abbiamo quattro costanti c', c'', n_0' e n_0'' tali che
 $f(n) \leq c' \cdot g(n) \quad \forall n > n_0'$ e $d(n) \leq c'' \cdot h(n) \quad \forall n > n_0''$ ed allora sappiamo che
 $f(n) \cdot d(n) \leq (c' \cdot c'') \cdot (g(n) \cdot h(n)) \quad \forall n \geq \max(n_0', n_0'')$ allora
 $f(n) \cdot d(n) \rightarrow O(g(n) \cdot h(n))$ con la $c = c' \cdot c''$ e $n = \max(n_0', n_0'')$

Verifica della notazione asintotica

Attraverso i limiti possiamo capire se la nostra notazione è corretta o sbagliata e questa si può verificare per i diversi casi che può verificare un limite $\rightarrow +\infty$ di $f(n) \rightarrow O/\Omega/\Theta(g(n))$

1. $\lim_{n \rightarrow +\infty} f(n)/g(n) = n \in \mathbb{R}$ allora $f(n) \rightarrow \Theta(g(n))$ ovviamente anche O & Ω
2. $\lim_{n \rightarrow +\infty} f(n)/g(n) = 0$ allora $g(n) \rightarrow +\infty$ più velocemente di $f(n)$ $f(n) \rightarrow O(g(n))$
3. $\lim_{n \rightarrow +\infty} f(n)/g(n) = 0$ allora $f(n) \rightarrow +\infty$ più velocemente di $g(n)$ $f(n) \rightarrow \Omega(g(n))$

Sommatoria:

1. $i=0 \rightarrow n$ di $\sum i = n * ((n+1)/2) = \Theta(n^2)$
2. $i=x \rightarrow n$ di $\sum c^i = (c^{(n+1)} - c^x) / (c - 1) = \Theta(c^n)$

Calcolo costo Computazionale

1. I for rispettano il range o il numero di elementi in un array/qualsiasi altra cosa.
2. I while hanno diverse modalità e condizioni come while $n > 1$ (o qualsiasi numero) :
 - a. $n // x$ con x un qualsiasi numero $O(\log(x) n)$
 - b. $n - x$ con x un qualsiasi numero $O(n)$
 - c. $i = 1$ $i * i \geq n$ con $i += 1$ allora abbiamo $O(n^{1/2})$
3. I for/while con for/while annidati, dobbiamo calcolare le informazioni e poi moltiplicare il tutto
4. In caso di un caso migliore e peggiore, prendiamo innanzitutto quello maggiore e quello che si ripete al crescere di n senno poniamo c come il max.

Ricorsione

La ricorsione è ispirata dalle funzioni ricorsive, e serve per risolvere un problema in un modo naturale come ad esempio trovare il fattoriale di n cioè $n * (n-1)!$, quest'ultimo avrà come caso base $n == 0$ return 1.

Pro della ricorsione:

- la soluzione è più naturale la soluzione a differenza di una iterativa.

Contro della ricorsione:

- Utilizza molta memoria perché si deve ricordare i casi precedenti .
- Non è scalabile all'infinito a causa del grande utilizzo della memoria.

Calcolo costo computazionale per la ricorsione

1. Metodo Iterativo:

- Sostituiamo all'interno della $f(n)$ fino a quando non arriviamo a $T(1)$
- $T(n) = T(n-1) + \Theta(1)$ con $T(1) = \Theta(1)$
- $T(n) = [T(n-2) + \Theta(1)] + \Theta(1) \dots T(n) = [T(n-3) + \Theta(1)] + \Theta(1) + \Theta(1) \dots$
- da qui possiamo dire che dobbiamo trovare un k tale che $n-k = 1$ e con un paio di semplificazioni arriviamo che $k = n-1$ e questo è il limite della sommatoria ed infine ad ogni $T(n)$ abbiamo un $+\Theta(1)$ quindi la nostra sommatoria sarà $i=0 \rightarrow n-1 \sum +\Theta(1)$ che sarebbe $\Theta(n) - \Theta(1)$

2. Metodo di Sostituzione:

- È consigliata al fine di dimostrare la veridicità di un costo computazionale.
- $T(n) = T(n-1) + c$ $T(1) = d$
- Il nostro Principale obiettivo è quello di verificare il Ω e O , al fine di poter dire che $T(n) \leq k*n$ (O) e $T(n) \geq h*n$ (Ω) e in tutto questo avremmo un $T(1) = d$ tale che per (O) $d \leq k$ e per (Ω) $d \geq h$.
- Andiamo a vederlo per $O(n)$ della $T(n)$ sopra, sappiamo che la $k \geq d$, e che $T(n) = k*(n-1) + c$ e che $T(n) \leq k*n$, quindi $k*n - k + c \leq kn \implies k \geq c$. quindi sappiamo che $T(n) = O(n)$ se e solo se $[k \geq c]$ e $[k \geq d]$.
- Questo vale anche per $\Omega(n)$ ma con $k*n \leq T(n)$.

3. Metodo dell'Albero

- È molto simile al Metodo Iterativo, soltanto che qui cerchiamo di dare anche una rappresentazione grafica al fine di capire come si sviluppa la ricorsione sotto forma d'albero.
- $T(n) = T(n-1) + \Theta(1)$
- disegniamo il costo per $T(n) = \Theta(1)$ e lo colleghiamo tramite un nodo a $T(n-1)$ e gli diamo $T(n-1) = \Theta(1)$ e così via alla fine il avremmo un albero con dei valori ai quali in alcuni casi gli potremmo assegnare delle variabili, e dobbiamo scoprire quanto ci mette $T(n) \rightarrow T(1)$ in questo caso $n-1$ Time quindi avremo $i=0 \rightarrow n-1 \sum +\Theta(1)$ che sarebbe $\Theta(n) - \Theta(1)$
- Ora vediamo $T(n) = 2T(n/2) + \Theta(n)^2$ con $T(1) = \Theta(1)$
- In cima abbiamo $\Theta(n)^2$, al 2° livello $= 2*(n/2)^2$, al 3° livello $= 4*(n/4)^2$, da qui vediamo una ripetizione di $2^i*(n/2^i)^2 \implies n^2 / 2^i$, e per vedere dove dobbiamo fermarci sappiamo che $n/2^k = 1$ quindi $n = 2^k$ 1 quindi $k = \log n$ e abbiamo $i=0 \rightarrow \log n \sum n^2/2^i$ che sarebbe $(siccome \text{ non dipende da } i) n^2 * \sum 1/2^i$ che sarebbe $\Theta(n^2) * \text{un num finito}$.

4. Metodo Principale:

- È molto semplice ma abbiamo delle condizioni da rispettare come:
 - $T(n) = a*T(n/b) + f(n)$ e $T(1) = \Theta(1)$.
- $S = n^{\log_b(a)}$.
- Se $f(n) = O(S)$ per qualche costante ϵ tale che $S = n^{\log_b(a-\epsilon)}$ allora $T(n) = n^{\log_b(a)}$
- Se $f(n) = \Theta(S)$ allora $T(n) = n^{\log_b(a)} * \log n$
- Se $f(n) = \Omega(S)$ per qualche costante ϵ tale che $S = n^{\log_b(a+\epsilon)}$

e se $f(n/b) \leq c * f(n)$ per qualche c costante < 1 allora $T(n) = \Theta(f(n))$

Ricordo che $n^{1+\varepsilon} > n * \log n$ per $n \rightarrow +\infty$

Ricerca di un valore (V) in una lista casuale

1. Tempo minimo $\Theta(n)$.
2. Ci calcoliamo la lunghezza della lista.
3. Scorriamo la lista per indici e se troviamo V ci segniamo l'indice e breakiamo il for.
4. ritorniamo quello che ci chiede esercizio (indice).

Ricerca di un valore (V) in una lista ordinata “ Binary search ”

1. $n = \text{len(Lista)}$
1. Considerando che è una lista ordinata possiamo dividere la lista $\log n$ volte
2. Quindi andremo a vedere l'elemento in $\text{lista}[n//2]$ e se uguale a V ritorniamo $n//2$, altrimenti se V è maggiore allora controlleremo $[n//2 + n//4]$ e ripeteremo le stesse operazioni, invece se minore controlleremo $[n//2 - n//4]$ e ripeteremo le stesse operazioni.
3. Il tempo sarà uguale a $\Theta(n)$ se ci calcoliamo la len(Lista) altrimenti $\Theta(\log n)$.