

Università di Napoli Federico II – Scuola Politecnica e delle Scienze di Base
Corso di Laurea Magistrale in Ingegneria Informatica



Corso di Algoritmi e Strutture Dati

Progettazione di algoritmi - *Divide et Imera*



Progetto di algoritmi: #1: DIVIDE ET IMPERA



- **Idea**
 - Si suddivide il problema in sottoproblemi più semplici
 - Si risolvono ricorsivamente i sottoproblemi
 - Si combinano le soluzioni dei sottoproblemi per ottenere la soluzione del problema di partenza
- Solitamente, il tempo di esecuzione degli algoritmi basati sull'approccio del divide et impera si determina utilizzando semplici tecniche
- Esempio:
 - Insertion Sort adotta un approccio **incrementale**
 - Per progettare un algoritmo di ordinamento più efficiente si può utilizzare l'approccio *divide et impera*



S&T: Problema Torre di Hanoi

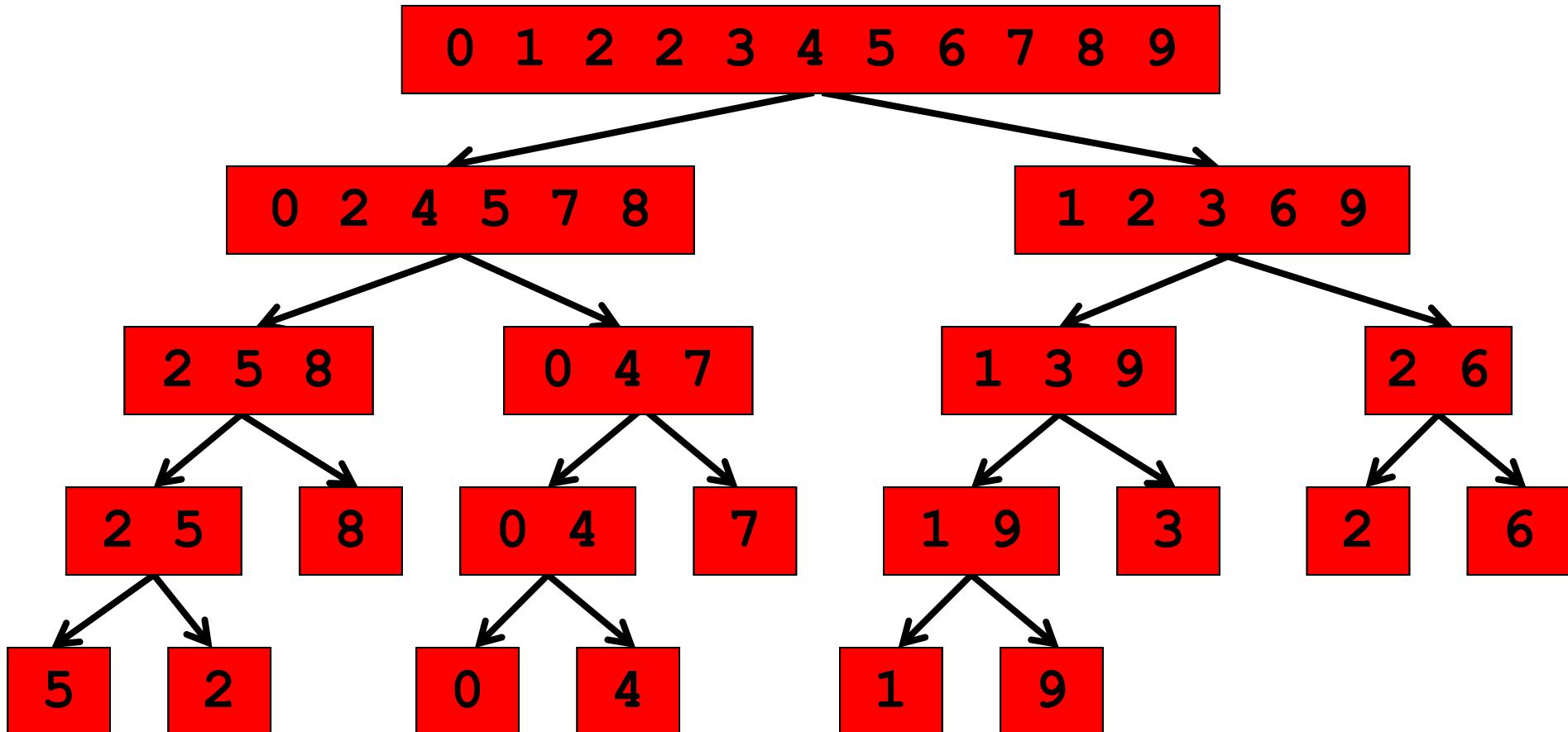
<https://www.khanacademy.org/computing/computer-science/algorithms/towers-of-hanoi/e/move-three-disks-in-towers-of-hanoi>

Merge Sort



- Approccio divide et impera
 - Suddivide una sequenza di n elementi in due sottosequenze di $n/2$ elementi
 - Ordina le sottosequenze ricorsivamente
- Fino ad ottenere una sottosequenza di lunghezza 1, che è ordinata
 - Combina (“merge”) due sottosequenze ordinate per produrre l’intera sequenza ordinata
- Due funzioni
 - Merge: combina due sottosequenze ordinate in una sequenza ordinata
 - Merge-Sort: ordina una sequenza suddividendola in due sottosequenze, ordinando ciascuna sottosequenza (mediante chiamata ricorsiva) e invocando la funzione Merge per ottenere la sequenza ordinata
- **Non** effettua un ordinamento sul posto

Merge-Sort



Merge

Fusione di sequenze ordinate

	p		q		r		
A	...	1	2	2	3	4	5
		6	7	...			
i				j			
L	2	4	5	7	∞	R	1
						2	3
						6	∞
n_1					n_2		

Merge (A, p, q, r)

$n_1 \leftarrow q - p + 1$

$n_2 \leftarrow r - q$

// create L[1.. n_1+1] and R[1.. n_2+1]

for i $\leftarrow 1$ to n_1

do L[i] $\leftarrow A[p+i-1]$

for j $\leftarrow 1$ to n_2

do R[j] $\leftarrow A[q+j]$

L[n_1+1] $\leftarrow R[n_2+1] \leftarrow \infty$

i $\leftarrow j \leftarrow 1$

for k $\leftarrow p$ to r

do if L[i] $\leq R[j]$

then A[k] $\leftarrow L[i]$

i $\leftarrow i+1$

else A[k] $\leftarrow R[j]$

j $\leftarrow j+1$

Merge: analisi di correttezza

- Invariante:
- *All'inizio di ciascuna iterazione dell'ultimo ciclo for, la sottosequenza A[p..k-1] contiene i k-p elementi più piccoli di L ed R ordinati. Inoltre, L[i] e R[j] sono i più piccoli elementi dei loro array a non essere stati ricoppiati in A*
- Ne deriva che:
- **L'invariante è vero prima della prima iterazione**
 - Per k=p, la sottosequenza A[p..k-1] è vuota ($k-p=0$). L[1] e R[1] sono i più piccoli elementi dei loro array a non essere stati ricoppiati in A
- **Una iterazione del ciclo conserva la verità dell'invariante**
 - A[p..k-1] contiene i k-p elementi più piccoli di L ed R ordinati
 - L[i] e R[j] sono i più piccoli elementi dei loro array a non essere stati ricoppiati in A
 - Se $L[i] \leq R[j]$, $L[i]$ viene ricoppiato in $A[k] \Rightarrow A[p..k]$ contiene i $k-p+1$ elementi più piccoli di L ed R ordinati. Incrementando k ed i, si ristabilisce l'invariante per la successiva iterazione
 - Analogo discorso se $R[j] < L[i]$

Merge: analisi di correttezza



- **L'invariante prova che l'algoritmo è corretto**
- Il ciclo termina quando $k=r+1$, quindi la sequenza $A[p..k-1]$ (i.e., $A[p..r]$) contiene i $k-p$ ($=r-p+1$) elementi più piccoli di L e R ordinati
 - Gli $r-p+1$ elementi più piccoli di L e R coincidono con tutti gli elementi originariamente in $A[p..r]$
- In L e R restano i due valori sentinella

Merge: analisi di complessità



Merge (A, p, q, r)

$n_1 \leftarrow q-p+1$

Tempo costante

$n_2 \leftarrow r-q$

// create L[1..n₁+1] and R[1..n₂+1]

for i $\leftarrow 1$ to n₁

do L[i] $\leftarrow A[p+i-1]$

Tempo proporzionale a n=n₁+n₂

for j $\leftarrow 1$ to n₂

do R[j] $\leftarrow A[q+j]$

L[n₁+1] $\leftarrow R[n_2+1] \leftarrow \infty$

Tempo costante

i $\leftarrow j \leftarrow 1$

for k $\leftarrow p$ to r

do if L[i] $\leq R[j]$

 then A[k] $\leftarrow L[i]$

 i $\leftarrow i+1$

 else A[k] $\leftarrow R[j]$

 j $\leftarrow j+1$

n iterazioni, ciascuna richiedente
un tempo costante \Rightarrow Tempo
proporzionale a n

Merge-Sort



Merge-Sort (A, p, r)

if $p < r$

$q \leftarrow \lfloor (p+r)/2 \rfloor$

Merge-Sort (A, p, q)

Merge-Sort ($A, q+1, r$)

Merge (A, p, q, r)

Invociamo Merge-Sort($A, 1, n$) per ordinare l'intera sequenza

MergeSort: analisi di complessità

- . **Per un approccio divide et impera**, detto
 - $\Theta(1)$ il tempo per risolvere il caso “banale”
 - $D(n)$ il tempo per suddividere un problema
 - $C(n)$ il tempo per ricombinare le soluzioni dei sottoproblemi
 - a il numero di sottoproblemi in cui si divide un problema
 - $1/b$ il fattore di riduzione della dimensione di un problema
- . Si ha che la complessità è esprimibile in forma di ricorrenza

$$T(n) = \begin{cases} \Theta(1) & \text{nel caso banale } n=1 \\ aT(n/b)+D(n)+C(n) & \text{altrimenti } (n>1) \end{cases}$$

Nel caso di Merge Sort: $D(n) = \Theta(1)$, $C(n) = \Theta(n)$, $a = b = 2$

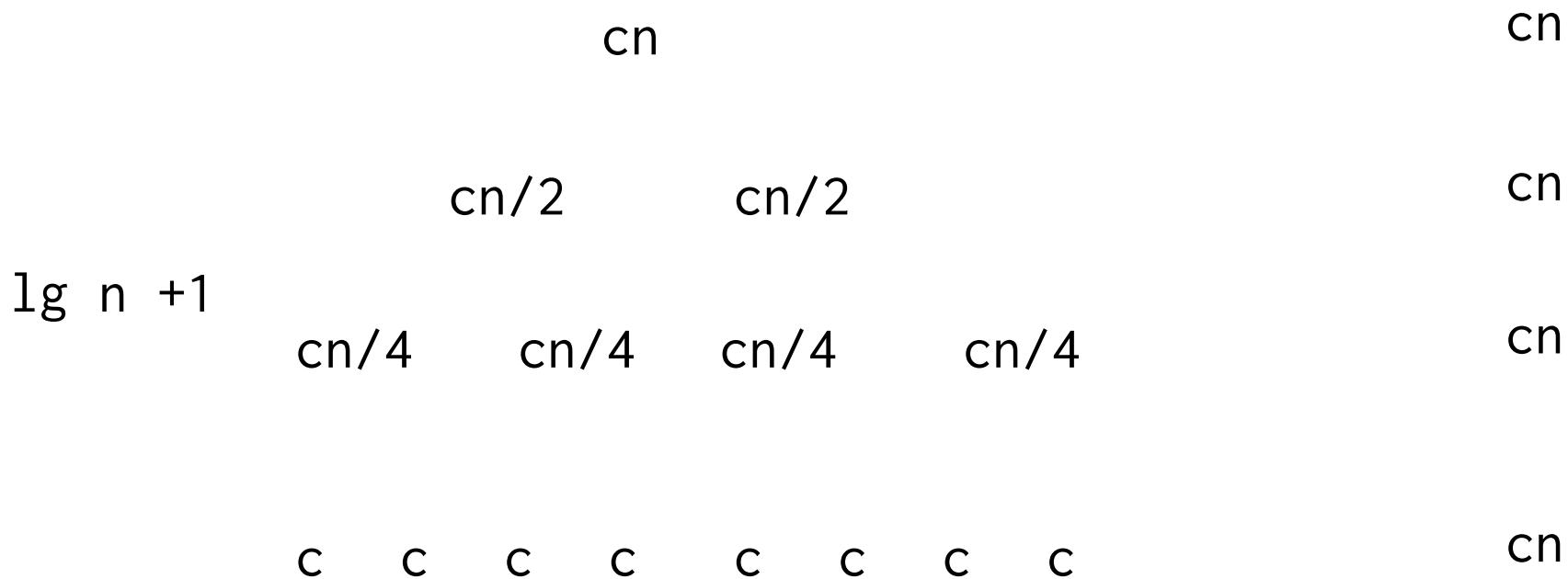
MergeSort: analisi di complessità



DIE
TI.
UNI
NA

Supponendo n sia una potenza di 2 e

$$T(n) = \begin{cases} c & \text{nel caso banale } n=1 \\ 2T(n/2)+cn & \text{altrimenti } (n>1) \end{cases}$$



Il costo totale è $cn(\lg n + 1) = cn \lg n + cn = \Theta(n \lg n)$



S&T: Problema Implementa MergeSort

<https://www.khanacademy.org/computing/computer-science/algorithms/merge-sort/pc/challenge-implement-merge-sort>



DIE
TI.
UNI



S&T: Problema

Problema del massimo sottoarray

Capitolo 4.1 Libro di testo

- Una ricorrenza è un'equazione o disequazione che descrive una funzione in termini del suo valore su ingressi di dimensione inferiore
- **Il tempo di esecuzione di algoritmi ricorsivi è solitamente espresso mediante una ricorrenza**
- es. MergeSort

$$T(n) = \begin{cases} \Theta(1) & \text{se } n=1 \\ 2T(n/2)+\Theta(n) & \text{altrimenti} \end{cases}$$

- Vedremo tre metodi per risolvere una ricorrenza
 - Metodo di sostituzione
 - Metodo dell'albero di ricorrenza
 - Metodo dell'esperto

Metodo di sostituzione



- Il metodo di sostituzione consiste di due passi
 - Ipotizzare la forma della soluzione
 - Verifica tramite principio di induzione
- Può essere usato per derivare sia limiti superiori che inferiori su una ricorrenza
- Esempio: abbiamo la ricorrenza $T(n) = 2T(\lfloor n/2 \rfloor) + n$
- Utilizziamo $T(1) = 1$ ai fini della verifica
- Supponiamo che la soluzione sia $T(n) = O(n \lg n)$
- Occorre provare che $T(n) \leq cn \lg n$ per una data costante c e per n sufficientemente grande

Metodo di sostituzione



- . Passo induttivo (assumiamo vero per $\lfloor n/2 \rfloor$, dimostriamo per n)
 - $T(n) = 2T(\lfloor n/2 \rfloor) + n$
 - $\leq 2 \cdot (c \lfloor n/2 \rfloor \lg(\lfloor n/2 \rfloor)) + n$
 - $\leq c n \cdot \lg(n/2) + n$
 - $= c n \cdot \lg n - cn \cdot \lg 2 + n$
 - $= c n \cdot \lg n - cn + n = cn \cdot \lg n - (c-1)n$
 - $\leq c n \cdot \lg n \quad \text{se } c \geq 1$
- . Caso base
 - Per $n=1$: $T(1) \leq c \lg 1 = 0$, il tempo di esecuzione non può essere ≤ 0
 - Si utilizzano $T(3)$ e $T(2)$ ($T(n) \leq cn \lg n$ deve valere per $n \geq n_0$)
 - Occorre mostrare che $T(2) = 4 \leq c2\lg 2$ e $T(3) = 5 \leq c3\lg 3$
 - È sufficiente che $c \geq 2$ per verificare i casi base

- Il metodo dell'albero di ricorrenza può essere usato per generare delle buone soluzioni di tentativo
 - Da verificare con il metodo di sostituzione se sono state fatte ipotesi semplificative per derivare le soluzioni
- I nodi dell'albero di ricorrenza riportano il costo per risolvere un dato (sotto)problema

Metodo dell'albero di ricorrenza



DIE
TI.
UNI
NA

- Consideriamo $T(n) = 3T(\lfloor n/4 \rfloor) + \Theta(n^2)$
 - Costruiamo l'albero di ricorrenza per $T(n) = 3T(n/4) + cn^2$
 - Ignoriamo l'arrotondamento per difetto (approssimazione)
 - Supponiamo che n sia una potenza di 4 (approssimazione)
 - Esplicitiamo il coefficiente costante c

T(n)

$$cn^2$$

$$cn^2$$

$$T\left(\frac{n}{4}\right)$$

$$T\left(\frac{n}{4}\right)$$

$$T\left(\frac{n}{4}\right)$$

$$c \left(\frac{n}{4}\right)^2$$

$$c \left(\frac{n}{4}\right)^2$$

$$c \left(\frac{n}{4}\right)^2$$

Metodo dell'albero di ricorrenza

- Quanti livelli ha l'albero?
- Per un nodo a livello i (il livello della radice è zero), la dimensione dei (sotto)problemi è $n/4^i$
- la dimensione 1 si raggiunge per $n/4^i = 1$ ovvero $i = \log_4 n$

$$cn^2$$

$$c\left(\frac{n}{4}\right)^2 \quad c\left(\frac{n}{4}\right)^2 \quad c\left(\frac{n}{4}\right)^2$$

$$\log_4 n + 1$$

$$c\left(\frac{n}{16}\right)^2 c\left(\frac{n}{16}\right)^2 c\left(\frac{n}{16}\right)^2 c\left(\frac{n}{16}\right)^2 c\left(\frac{n}{16}\right)^2 c\left(\frac{n}{16}\right)^2 c\left(\frac{n}{16}\right)^2 c\left(\frac{n}{16}\right)^2$$

T(1)T(1)T(1)T(1)T(1)T(1)T(1)T(1)...T(1)T(1)T(1)T(1)

Metodo dell'albero di ricorrenza

- Qual è il costo a ciascun livello dell'albero?
- Ogni livello ha il triplo dei nodi del livello precedente, quindi il numero di nodi a livello i è 3^i
- al livello i , il costo di un singolo nodo è $c(n/4^i)^2$

$$cn^2$$

$$c\left(\frac{n}{4}\right)^2 \quad c\left(\frac{n}{4}\right)^2 \quad c\left(\frac{n}{4}\right)^2$$

$$\log_4 n + 1$$

$$c\left(\frac{n}{16}\right)^2 c\left(\frac{n}{16}\right)^2 c\left(\frac{n}{16}\right)^2 c\left(\frac{n}{16}\right)^2 c\left(\frac{n}{16}\right)^2 c\left(\frac{n}{16}\right)^2 c\left(\frac{n}{16}\right)^2 c\left(\frac{n}{16}\right)^2$$

T(1)T(1)T(1)T(1)T(1)T(1)T(1)T(1)...T(1)T(1)T(1)T(1)

Metodo dell'albero di ricorrenza

$$\begin{aligned} T(n) &= \sum_{i=0}^{\log_4 n - 1} 3^i c \left(\frac{n}{4^i} \right)^2 + 3^{\log_4 n} T(1) = \sum_{i=0}^{\log_4 n - 1} \left(\frac{3}{16} \right)^i cn^2 + \Theta(n^{\log_4 3}) < \sum_{i=0}^{\infty} \left(\frac{3}{16} \right)^i cn^2 + \Theta(n^{\log_4 3}) \\ &= \frac{1}{1 - (3/16)} cn^2 + \Theta(n^{\log_4 3}) = \frac{16}{13} cn^2 + \Theta(n^{\log_4 3}) \Rightarrow T(n) = O(n^2) \end{aligned}$$

cn^2

$$c \left(\frac{n}{4} \right)^2 \quad c \left(\frac{n}{4} \right)^2 \quad c \left(\frac{n}{4} \right)^2$$

$\log_4 n + 1$

$$c \left(\frac{n}{16} \right)^2 c \left(\frac{n}{16} \right)^2$$

$T(1)T(1)T(1)T(1)T(1)T(1)T(1)\dots T(1)T(1)T(1)T(1)$

Metodo dell'albero di ricorrenza

- . Verifichiamo con il metodo di sostituzione che $T(n)=O(n^2)$
 - Ovvero $\exists d>0 : T(n) \leq dn^2$
- . Passo induttivo (assumiamo vero per $\lfloor n/4 \rfloor$, dimostriamo per n)
 - $T(n) = 3T(\lfloor n/4 \rfloor) + \Theta(n^2)$
 - $\leq 3T(\lfloor n/4 \rfloor) + cn^2$
 - $\leq 3d \lfloor n/4 \rfloor^2 + cn^2$
 - $\leq 3d(n/4)^2 + cn^2$
 - $= (3/16)dn^2 + cn^2$
 - $\leq dn^2 \quad \text{se } d \geq (16/13)c$
- . Per $n=1$ (caso base), $T(1) \leq d$
 - Verificato quindi per $d \geq T(1)$

Metodo dell'albero di ricorrenza

- Con il metodo di sostituzione abbiamo verificato che $T(n)=O(n^2)$
- In realtà è anche $T(n)=\Omega(n^2)$
- La prima chiamata ricorsiva contribuisce con un costo $\Theta(n^2)$ e quindi $\Omega(n^2)$ è un limite inferiore per la ricorrenza
- Quindi $T(n)=\Theta(n^2)$

Metodo dell'esperto



- Si basa sul teorema dell'esperto
- Siano $a \geq 1$ e $b > 1$ costanti, $f(n)$ una funzione e $T(n) = aT(n/b) + f(n)$
 - n/b può essere interpretato sia come $\lfloor n/b \rfloor$ che $\lceil n/b \rceil$
- Allora
 - Se $f(n) = O(n^{\log_b a - \epsilon})$ per un dato $\epsilon > 0$, allora $T(n) = \Theta(n^{\log_b a})$
 - Se $f(n) = \Theta(n^{\log_b a})$, allora $T(n) = \Theta(n^{\log_b a} \lg n)$
 - Se $f(n) = \Omega(n^{\log_b a + \epsilon})$ per un dato $\epsilon > 0$ e $af(n/b) \leq cf(n)$ per una data $c < 1$ ed n sufficientemente grande, allora $T(n) = \Theta(f(n))$
- Osservazioni
 - Intuitivamente, la soluzione è data dalla più grande tra $f(n)$ e $n^{\log_b a}$
 - $f(n)$ deve essere **polynomialmente** più piccola (grande) di $n^{\log_b a}$ (c' è ϵ)
 - => C'è un gap tra il caso 1) e 2) e tra il caso 2) e 3)

Metodo dell'esperto

- $T(n) = 9T(n/3) + n$
 - $a=9, b=3, f(n)=n, n^{\log_b a} = n^{\log_3 9} = n^2$
 - $n = f(n) = O(n^{\log_3 9 - \epsilon}) = O(n^{2-\epsilon})$ con $\epsilon=1$
 - Si applica il caso 1) $\Rightarrow T(n) = \Theta(n^2)$
- $T(n) = T(2n/3) + 1$
 - $a=1, b=3/2, f(n)=1, n^{\log_b a} = n^{\log_{3/2} 1} = n^0 = 1$
 - $1 = f(n) = \Theta(n^{\log_b a}) = \Theta(1)$
 - Si applica il caso 2) $\Rightarrow T(n) = \Theta(\lg n)$
- $T(n) = 3T(n/4) + n \lg n$
 - $a=3, b=4, f(n)=n \lg n, n^{\log_b a} = n^{\log_4 3}$
 - $n \lg n = f(n) = \Omega(n^{\log_4 3 + \epsilon})$ con $\epsilon \approx 0,2$ ($\log_4 3 \approx 0,793$)
 - $af(n/b) = 3(n/4) \lg(n/4) \leq (3/4)n \lg n = cf(n)$ con $c=3/4$
 - Si applica il caso 3) $\Rightarrow T(n) = \Theta(n \lg n)$

Metodo dell'esperto



- $T(n)=2T(n/2)+n\lg n$
 - $a=2, b=2, f(n)=n\lg n, n^{\log_b a} = n^{\log_2 2} = n$
 - $f(n)=n\lg n$ non è polinomialmente più grande di n
- $f(n)/n^{\log_b a} = n\lg n / n = \lg n$ è asintoticamente più piccolo di $n^\epsilon, \epsilon>0$
 - Questa ricorrenza ricade nel gap tra i casi 2) e 3)