Risoluzione delle Equazioni di Ricorrenza

Prof. Simone Faro
Dispensa didattica per il corso di Algoritmi e Laboratorio
Università di Catania

Anno Accademico 2025–2026

Indice

1 Introduzione alle equazioni di ricorrenza

Molti algoritmi, specialmente quelli ricorsivi, svolgono lo stesso tipo di lavoro su istanze del problema di dimensione più piccola e combinano poi i risultati ottenuti. Per analizzare il loro tempo di esecuzione (o, più in generale, il costo computazionale), si ricorre alla formulazione di una *equazione di ricorrenza* che esprime il costo totale $T(n)$ in funzione del costo dei sottoproblemi e del lavoro aggiuntivo svolto a ciascun livello di ricorsione.

Definizione (equazione di ricorrenza). Un'equazione di ricorrenza è una relazione del tipo

$$T(n) = a\,T\Big(\frac{n}{b}\Big) + f(n),$$

dove:

- $T(n)$ è il costo per risolvere un problema di dimensione n;
- a è il numero di sottoproblemi in cui il problema viene suddiviso;
- ciascun sottoproblema ha dimensione n/b;
- $f(n)$ rappresenta il lavoro non ricorsivo (cioè il costo per dividere, combinare o gestire i sottoproblemi).

Il termine **forma esplicita** o **soluzione** dell'equazione di ricorrenza indica un'espressione chiusa (non ricorsiva) per $T(n)$, utile per comprendere come cresce il tempo di esecuzione al crescere di n. Trovare questa forma esplicita è l'obiettivo dell'*analisi asintotica* di un algoritmo.

Esistono diversi metodi per risolvere o stimare le equazioni di ricorrenza più comuni. In questa dispensa ne analizzeremo tre, ciascuno con una diversa prospettiva:

1. **Metodo dell'albero di ricorsione** — fornisce un approccio intuitivo e visivo: si rappresenta la ricorsione come un albero, si calcola il costo per livello e si sommano i costi totali. È particolarmente utile per capire *dove* si concentra il lavoro (livelli iniziali o finali).
2. **Metodo della sostituzione** — si formula un'ipotesi sul comportamento asintotico di $T(n)$ (ad esempio $O(n \log n)$) e la si dimostra per induzione. È un metodo più formale, ma molto efficace per ricorrenze non standard.

3. **Metodo Master** — fornisce un risultato generale (detto *Teorema Master*) che consente di determinare l'ordine di grandezza di $T(n)$ confrontando $f(n)$ con $n^{\log_b a}$. È un metodo rapido e sistematico, ma applicabile solo a ricorrenze che rispettano alcune ipotesi regolari.

Ciascuno di questi metodi conduce allo stesso risultato finale, ma con livelli diversi di rigore, semplicità e generalità.

1.1 Identificare l'equazione di ricorrenza

Prima di affrontare la risoluzione di un'equazione di ricorrenza, è importante saperla formulare nel modo corretto. Questo significa comprendere come il comportamento ricorsivo di un algoritmo si traduce in una relazione che esprime il costo totale in funzione dei costi dei sottoproblemi generati e del lavoro aggiuntivo necessario per gestirli o combinarli.

Ogni algoritmo ricorsivo può essere idealmente scomposto in tre momenti fondamentali. Nella prima fase, il problema viene suddiviso in più sottoproblemi di dimensione minore. Segue poi la fase di risoluzione, in cui ciascun sottoproblema viene affrontato a sua volta in modo ricorsivo. Infine, i risultati parziali vengono combinati per ottenere la soluzione complessiva. Se indichiamo con $T(n)$ il costo dell'algoritmo su un problema di dimensione n, possiamo esprimere questo comportamento nel modo seguente:

$$T(n) = a\,T\left(\frac{n}{b}\right) + f(n),$$

dove a rappresenta il numero di sottoproblemi prodotti, b il fattore di riduzione della dimensione e $f(n)$ il costo complessivo delle operazioni non ricorsive, ossia di tutte quelle attività che non comportano nuove chiamate, come la divisione del problema, la fusione dei risultati o eventuali operazioni preliminari.

L'abilità nel derivare correttamente una relazione di questo tipo nasce dalla capacità di leggere nel codice — o nel ragionamento algoritmico — il numero di chiamate ricorsive e la quantità di lavoro che ciascuna richiede. Alcuni esempi concreti possono aiutare a chiarire questo processo.

Esempio 1 — Ricerca binaria

Nel caso della ricerca binaria, l'algoritmo esamina un intervallo ordinato, ne calcola il punto medio e confronta il valore cercato con l'elemento corrispondente. Se il valore coincide, la ricerca termina; altrimenti, viene ri-

chiamata ricorsivamente solo su una metà dell'intervallo, dimezzando così la dimensione del problema. Il codice seguente ne mostra la logica di base.

```
int binarySearch(int A[], int low, int high, int x) {
    if (low > high)
        return -1;
    int mid = (low + high) / 2;
    if (A[mid] == x)
        return mid;
    else if (A[mid] > x)
        return binarySearch(A, low, mid - 1, x);
    else
        return binarySearch(A, mid + 1, high, x);
}
```

In ogni chiamata si effettua al massimo una sola chiamata ricorsiva su metà dell'intervallo, mentre il lavoro non ricorsivo (calcolo dell'indice e confronto) è costante. La relazione che descrive il tempo di esecuzione è quindi

$$T(n) = T(n/2) + 1,$$

che conduce a una complessità asintotica $O(\log n)$, in accordo con il comportamento noto della ricerca binaria.

Esempio 2 — Merge Sort

L'algoritmo Merge Sort applica la strategia "divide et impera": divide l'array in due metà, ordina ciascuna ricorsivamente e poi fonde i risultati. Il codice seguente illustra una versione semplificata del procedimento.

```
void mergeSort(int A[], int left, int right) {
    if (left < right) {
        int mid = (left + right) / 2;
        mergeSort(A, left, mid);
        mergeSort(A, mid + 1, right);
        merge(A, left, mid, right);
    }
}
```

Ogni chiamata genera due sottoproblemi di dimensione $n/2$, e la fusione dei risultati richiede un tempo proporzionale a n. Di conseguenza, la ricorrenza che descrive il costo dell'algoritmo è

$$T(n) = 2T(n/2) + n.$$

Il termine $2T(n/2)$ rappresenta il lavoro dovuto alle due chiamate ricorsive, mentre n corrisponde al costo lineare della fase di fusione. La soluzione, che verrà ricavata più avanti mediante il metodo dell'albero di ricorsione, è $T(n) = O(n \log n)$.

Esempio 3 — Moltiplicazione di matrici con l'algoritmo di Strassen

La moltiplicazione classica di due matrici quadrate di dimensione $n \times n$ richiede circa n^3 operazioni elementari. Strassen mostrò che è possibile ridurre il numero di moltiplicazioni parziali a sette (invece di otto), aumentando però leggermente il numero di somme e sottrazioni di matrici. Il frammento seguente riassume in modo schematico la struttura ricorsiva dell'algoritmo.

```
Matrix StrassenMultiply(Matrix A, Matrix B) {
    if (A.size == 1)
        return A * B;
    else {
        divide A and B into 4 submatrices;
        compute 7 products recursively;
        combine results to form C;
        return C;
    }
}
```

Ogni chiamata genera sette sottoproblemi di dimensione $n/2$, e la combinazione dei risultati richiede operazioni di somma e differenza tra matrici di costo $O(n^2)$. La ricorrenza corrispondente è quindi

$$T(n) = 7T(n/2) + n^2.$$

La soluzione di questa equazione, che vedremo in seguito, è $T(n) = \Theta(n^{\log_2 7}) \approx \Theta(n^{2.81})$, valore inferiore al classico $O(n^3)$ della moltiplicazione tradizionale.

Questi esempi mostrano come il passaggio dal codice all'equazione di ricorrenza richieda soprattutto una buona interpretazione del comportamento dell'algoritmo. È necessario individuare quante volte la procedura si richiama su se stessa e stimare il lavoro residuo che rimane da svolgere una volta terminate le chiamate ricorsive. Un'analisi attenta di queste due componenti — la parte ricorsiva e quella non ricorsiva — consente di formulare correttamente la ricorrenza e di ottenere una stima affidabile della complessità complessiva.

1.2 Ricorrenze non uniformi

Non tutti gli algoritmi ricorsivi possono essere descritti mediante un'equazione del tipo

$$T(n) = a\,T\Big(\frac{n}{b}\Big) + f(n),$$

in cui il problema viene suddiviso in un numero costante a di sottoproblemi tutti della stessa dimensione n/b. In molti casi reali, la dimensione dei sottoproblemi può variare a seconda dei dati o del comportamento dell'algoritmo, e il numero di chiamate ricorsive può non essere fisso. In queste situazioni, l'equazione di ricorrenza deve tenere conto in modo esplicito delle diverse dimensioni e della possibile asimmetria tra i sottoproblemi.

Un esempio emblematico di questo tipo di comportamento è l'algoritmo QuickSort. Nel caso ideale, l'elemento pivot scelto per la partizione divide l'array esattamente in due parti uguali, ciascuna di dimensione $n/2$. In questa circostanza la ricorrenza assume la forma nota

$$T(n) = 2T(n/2) + n,$$

analoga a quella del Merge Sort. Tuttavia, nella realtà il pivot non è sempre centrale: può accadere che divida l'array in due porzioni di dimensione diversa, una molto piccola e l'altra molto grande. Il codice seguente mostra in modo semplificato la struttura dell'algoritmo.

```
void quickSort(int A[], int left, int right) {
    if (left < right) {
        int pivot = partition(A, left, right);
        quickSort(A, left, pivot - 1);
        quickSort(A, pivot + 1, right);
    }
}
```

Nel caso medio, la partizione tende a dividere i dati in modo abbastanza equilibrato, e si può scrivere una ricorrenza del tipo

$$T(n) = T(\alpha n) + T((1-\alpha)n) + n,$$

dove il parametro α rappresenta la frazione di elementi che ricadono nella prima metà dopo la partizione ($0 < \alpha < 1$). Quando $\alpha = 1/2$, la formula si riduce al caso bilanciato classico, ma se α si avvicina a 0 o a 1 il costo peggiora fino al caso limite

$$T(n) = T(n-1) + T(1) + n,$$

che corrisponde a un pivot sempre scelto male e produce complessità quadratica $O(n^2)$. L'analisi di QuickSort mostra quindi come la distribuzione delle dimensioni dei sottoproblemi influenzi fortemente la crescita del costo totale.

Un altro esempio interessante è fornito dal problema delle Torri di Hanoi, che produce un albero di ricorsione fortemente sbilanciato. L'obiettivo è spostare n dischi da un piolo all'altro rispettando la regola secondo cui un disco non può mai essere posato su uno più piccolo. Il codice ricorsivo standard è il seguente:

```
void hanoi(int n, char from, char to, char aux) {
    if (n == 1)
        move(from, to);
    else {
        hanoi(n - 1, from, aux, to);
        move(from, to);
        hanoi(n - 1, aux, to, from);
    }
}
```

In questo caso, la procedura richiama se stessa due volte, ma non su problemi di dimensione $n/2$, bensì di dimensione $n-1$. Ogni passo comporta un singolo spostamento (di costo costante), seguito da due chiamate su un problema leggermente più piccolo. L'equazione di ricorrenza è quindi

$$T(n) = 2T(n-1) + 1.$$

A differenza delle ricorrenze che riducono la dimensione per un fattore costante, qui la dimensione decresce linearmente, di un'unità per volta. Il risultato è una crescita esponenziale: risolvendo la ricorrenza si ottiene

$$T(n) = 2^n - 1,$$

che corrisponde al numero minimo di mosse necessarie per risolvere il gioco.

Questi esempi mostrano che, quando la riduzione del problema non è uniforme, non è possibile applicare direttamente formule standard come il teorema Master, e occorre analizzare la ricorrenza caso per caso. Tuttavia, i principi generali rimangono gli stessi: è sempre possibile visualizzare la struttura delle chiamate come un albero, contare i nodi generati e sommare i costi dei livelli per ottenere il comportamento complessivo dell'algoritmo.

2 Il metodo dell'albero di ricorsione

Quando si affronta un'equazione di ricorrenza, il primo obiettivo è comprenderne il significato operativo. Ogni termine della ricorrenza rappresenta infatti un insieme di chiamate ricorsive e il lavoro associato ad esse. Il metodo dell'albero di ricorsione nasce proprio dall'idea di *visualizzare* questa struttura di chiamate come un albero, in cui ogni nodo rappresenta una singola invocazione dell'algoritmo e ogni arco una chiamata ricorsiva. In questo modo diventa possibile analizzare passo dopo passo come si distribuisce il lavoro complessivo nei diversi livelli della ricorsione.

Immaginiamo di avere un algoritmo descritto dalla ricorrenza

$$T(n) = a\,T\Big(\frac{n}{b}\Big) + f(n),$$

dove ciascun problema di dimensione n viene suddiviso in a sottoproblemi di dimensione n/b, e dove $f(n)$ rappresenta il costo del lavoro svolto localmente, cioè le operazioni di divisione e combinazione dei risultati. Il metodo dell'albero di ricorsione consiste nel rappresentare questa relazione sotto forma di un albero in cui:

- la *radice* dell'albero corrisponde al problema iniziale di dimensione n;
- i *nodi interni* rappresentano le chiamate ricorsive generate a ciascun livello;
- ogni *nodo* è etichettato con il costo del lavoro locale $f(n_i)$, dove n_i è la dimensione del sottoproblema corrispondente;
- i *figli* di un nodo rappresentano le chiamate generate da quel nodo, ognuna di dimensione ridotta di un fattore b.

La costruzione di questo albero permette di scomporre la ricorsione in livelli. Il primo livello contiene un solo nodo (il problema originale), il secondo livello contiene a nodi (uno per ciascun sottoproblema), il terzo livello ne contiene a^2, e così via. A ogni livello k dell'albero il numero di nodi è a^k, e la dimensione dei sottoproblemi è ridotta a n/b^k. Il costo totale associato al livello k può quindi essere scritto come

$$C_k = a^k\, f\Big(\frac{n}{b^k}\Big)\,,$$

cioè come il numero di nodi di quel livello moltiplicato per il costo del lavoro svolto in ciascun nodo. L'idea chiave del metodo è che il costo complessivo

dell'algoritmo non è altro che la somma dei costi dei singoli livelli:

$$T(n) = \sum_{k=0}^{L} C_k = \sum_{k=0}^{L} a^k \, f\left(\frac{n}{b^k}\right),$$

dove L rappresenta la profondità dell'albero, ossia il numero di livelli fino a quando la dimensione del problema non si riduce a una costante. Nel caso più comune, la ricorsione si arresta quando $n/b^L = 1$, da cui segue $L = \log_b n$.

Questo approccio fornisce non solo un modo intuitivo per stimare la crescita di $T(n)$, ma anche una rappresentazione visiva dell'andamento del lavoro. Osservando come varia il costo dei livelli successivi, è possibile capire se il lavoro complessivo è dominato dai livelli più alti (quelli vicini alla radice), dai livelli intermedi o da quelli più profondi dell'albero.

In molti casi, il comportamento della somma può essere descritto come una serie geometrica, in cui ogni livello contribuisce con un costo proporzionale al precedente. Quando la somma è dominata dai primi livelli, il costo totale è determinato dal lavoro iniziale $f(n)$; quando tutti i livelli hanno lo stesso ordine di grandezza, il costo cresce come il numero dei livelli; infine, quando i livelli inferiori diventano progressivamente più costosi, il termine dominante si sposta verso il fondo dell'albero. Queste tre possibilità corrispondono, in un certo senso, ai tre casi principali del Teorema Master, che verrà presentato più avanti.

Il grande vantaggio del metodo dell'albero di ricorsione è la sua immediatezza concettuale. Non richiede particolari ipotesi matematiche e può essere applicato anche a ricorrenze irregolari o ibride, per le quali altri metodi risultano meno efficaci. Permette inoltre di stimare il comportamento asintotico di un algoritmo senza necessariamente risolvere formalmente l'equazione, ma semplicemente riconoscendo quale parte dell'albero — la radice, i livelli intermedi o le foglie — contribuisce maggiormente al costo complessivo.

Nei paragrafi che seguono, applicheremo il metodo a diversi esempi di ricorrenze, mostrando passo per passo come costruire l'albero, calcolare i costi dei vari livelli e determinare la crescita complessiva di $T(n)$.

2.1 Applicazione del metodo dell'albero di ricorsione

Dopo aver compreso l'idea generale del metodo, possiamo applicarlo concretamente ad alcuni casi classici. Analizzeremo ora tre equazioni di ricorrenza che corrispondono a tre algoritmi ben noti: la ricerca binaria, l'algoritmo Merge Sort e l'algoritmo di Strassen per la moltiplicazione di matrici. In ciascun caso costruiremo idealmente l'albero della ricorsione, calcoleremo il costo di ogni livello e ne ricaveremo il comportamento complessivo.

Esempio 1 — Ricerca binaria

Consideriamo l'equazione

$$T(n) = T(n/2) + 1.$$

Ogni chiamata ricorsiva genera una sola chiamata su metà dell'input, e il lavoro aggiuntivo (calcolo dell'indice medio e confronto) è costante. Possiamo immaginare l'albero della ricorsione come una catena di chiamate successive, in cui ogni nodo produce un unico figlio di dimensione dimezzata. Il costo al livello 0 è pari a 1 volta il lavoro di quel nodo, cioè una quantità costante; al livello 1 il lavoro è ancora costante, e così via, fino a quando la dimensione del problema si riduce a 1. La profondità dell'albero è dunque pari a $\log_2 n$, poiché ad ogni passo la dimensione viene dimezzata.

Il costo totale si ottiene sommando il contributo di tutti i livelli:

$$T(n) = 1 + 1 + 1 + \cdots + 1,$$

dove il numero dei termini è pari a $\log_2 n + 1$. Da qui risulta immediatamente che

$$T(n) = O(\log n).$$

In questo caso, il costo per livello è costante e l'albero ha profondità logaritmica: il risultato è quindi una crescita proporzionale al numero dei livelli.

Esempio 2 — Merge Sort

Passiamo ora all'equazione

$$T(n) = 2T(n/2) + n,$$

che descrive il comportamento dell'algoritmo Merge Sort. In ciascun livello della ricorsione, l'array di dimensione n viene diviso in due metà, ognuna delle quali viene ordinata ricorsivamente. Il costo della fusione dei due sottoarray è proporzionale a n, poiché ogni elemento viene confrontato e copiato una sola volta.

Rappresentiamo la ricorsione come un albero. Il livello iniziale (radice) ha costo n. Dal nodo iniziale si generano due nodi figli, ognuno con un sottoproblema di dimensione $n/2$. Ciascuno di questi nodi ha un costo pari a $f(n/2) = n/2$, e quindi il costo complessivo del secondo livello è $2 \cdot (n/2) = n$. Al livello successivo, ci sono quattro nodi, ciascuno con costo $n/4$; il costo

totale del livello è ancora $4 \cdot (n/4) = n$. Continuando in questo modo, si osserva che ogni livello dell'albero ha un costo complessivo pari a n.

Poiché la dimensione del problema si riduce di un fattore 2 a ogni livello, la profondità dell'albero è pari a $\log_2 n$. Il costo totale dell'algoritmo è quindi dato dalla somma dei costi di tutti i livelli:

$$T(n) = n + n + n + \cdots + n = n(1 + \log_2 n).$$

Tralasciando le costanti, otteniamo

$$T(n) = O(n \log n).$$

Questo risultato ha anche un'interpretazione intuitiva: ad ogni livello dell'albero viene processato l'intero insieme dei dati, ma il numero di livelli è solo logaritmico. L'albero è dunque bilanciato e uniforme, e il lavoro complessivo è il prodotto del costo per livello (lineare) e della profondità (logaritmica).

Esempio 3 — Algoritmo di Strassen

Consideriamo infine la ricorrenza

$$T(n) = 7T(n/2) + n^2,$$

che rappresenta la complessità dell'algoritmo di Strassen per la moltiplicazione di matrici. Ogni moltiplicazione di matrici di dimensione $n \times n$ viene ridotta a sette moltiplicazioni di matrici di dimensione $n/2$, più alcune operazioni di somma e sottrazione di matrici, che complessivamente richiedono tempo proporzionale a n^2.

Visualizziamo l'albero della ricorsione. Al livello 0 (radice) il costo è n^2. Al livello 1 ci sono 7 nodi, ciascuno con un sottoproblema di dimensione $n/2$, e quindi un costo per nodo pari a $(n/2)^2 = n^2/4$. Il costo complessivo del secondo livello è dunque $7 \cdot (n^2/4) = (7/4)n^2$. Al livello successivo ci saranno $7^2 = 49$ nodi, ciascuno con costo $(n/4)^2 = n^2/16$, per un totale di $49 \cdot (n^2/16) = (49/16)n^2$. In generale, il costo del livello k è

$$C_k = n^2 \left(\frac{7}{4}\right)^k.$$

Il numero di livelli dell'albero è pari a $\log_2 n$, poiché la dimensione viene dimezzata a ogni passo. Il costo totale si ottiene sommando i costi dei vari livelli:

$$T(n) = n^2 \sum_{k=0}^{\log_2 n} \left(\frac{7}{4}\right)^k.$$

A questo punto compare una somma che è una serie geometrica di ragione $r = \frac{7}{4}$, maggiore di 1. Ricordiamo che, per una serie geometrica con ragione $r > 1$,

$$\sum_{k=0}^{m} r^k = \frac{r^{m+1} - 1}{r - 1},$$

e che, quando m cresce, il termine dominante è r^m. Applicando questa formula otteniamo

$$T(n) = n^2 \cdot \frac{(\frac{7}{4})^{\log_2 n + 1} - 1}{\frac{7}{4} - 1} \approx \frac{4}{3}\, n^2 \left(\frac{7}{4}\right)^{\log_2 n}.$$

Poiché $(\frac{7}{4})^{\log_2 n} = n^{\log_2(7/4)} = n^{\log_2 7 - 2}$, si ottiene

$$T(n) = O\big(n^{\log_2 7}\big)\,.$$

Il termine dominante proviene dunque dai livelli più profondi dell'albero, dove il numero di nodi cresce più rapidamente del fattore di riduzione della dimensione dei sottoproblemi.

In questo caso, la crescita complessiva è più rapida che nel Merge Sort ma più lenta della moltiplicazione classica $O(n^3)$. L'albero di ricorsione consente di visualizzare immediatamente questa differenza: il numero di nodi aumenta più velocemente (sette per volta anziché due), mentre la riduzione della dimensione è la stessa, e ciò spiega l'esponente intermedio $n^{\log_2 7}$.

Questi tre esempi mostrano come il metodo dell'albero di ricorsione permetta di stimare con immediatezza la crescita di un algoritmo, osservando semplicemente l'equilibrio tra la quantità di lavoro svolta per livello e la profondità complessiva dell'albero. Si tratta di uno strumento intuitivo e molto utile per costruire rapidamente un modello mentale della complessità di un algoritmo prima di affrontare metodi più formali di risoluzione.

3 Il metodo della sostituzione

Un secondo approccio molto utile per risolvere le equazioni di ricorrenza è il *metodo della sostituzione*. Mentre il metodo dell'albero di ricorsione si basa su una rappresentazione visiva e intuitiva della distribuzione del lavoro, il metodo della sostituzione adotta una prospettiva più analitica: consiste nel formulare un'ipotesi sulla forma asintotica della soluzione e nel dimostrare che tale ipotesi è corretta attraverso un ragionamento induttivo.

L'idea di fondo è semplice. Si parte dall'equazione di ricorrenza e, osservando la struttura del problema, si tenta di "indovinare" la crescita di $T(n)$,

ad esempio $O(n)$, $O(n \log n)$, o $O(n^2)$. Una volta formulata questa ipotesi, la si sostituisce nell'equazione e si verifica se l'uguaglianza (o la disuguaglianza) risulta soddisfatta per valori sufficientemente grandi di n. Se l'ipotesi risulta coerente, viene così confermata; altrimenti, la si modifica finché non produce una forma valida.

Questo metodo è chiamato "della sostituzione" perché prevede di sostituire l'ipotesi nella ricorrenza, semplificarla e controllare che il risultato sia coerente. Dal punto di vista matematico, il procedimento può essere interpretato come una *dimostrazione per induzione* sull'ampiezza del problema.

In termini pratici, il procedimento segue tre fasi logiche:

1. Si formula un'ipotesi sul comportamento asintotico di $T(n)$. Spesso l'ipotesi deriva dall'intuizione fornita da metodi più intuitivi come l'albero di ricorsione.

2. Si sostituisce questa ipotesi all'interno dell'equazione di ricorrenza e si verifica se l'uguaglianza o disuguaglianza risulta soddisfatta.

3. Se necessario, si aggiusta l'ipotesi (ad esempio includendo una costante o un termine logaritmico) fino a ottenere un'espressione coerente.

3.1 Applicazione del metodo della sostituzione

Per comprendere in modo concreto come funziona il metodo della sostituzione, è utile applicarlo ad alcuni esempi significativi. Lo scopo non è soltanto verificare il risultato già noto, ma soprattutto capire il meccanismo logico che consente di confermare un'ipotesi asintotica mediante una dimostrazione formale.

In pratica, si procede come segue: si parte da una ricorrenza già nota o intuita (ad esempio, una di quelle incontrate nel metodo dell'albero di ricorsione), si formula un'ipotesi sul comportamento di $T(n)$ — per esempio $O(\log n)$, $O(n \log n)$ o $O(n^{\log_2 7})$ — e si verifica, sostituendola all'interno dell'equazione, se tale ipotesi è coerente con i termini della ricorrenza. Quando l'uguaglianza o disuguaglianza risulta valida per valori sufficientemente grandi di n, l'ipotesi è confermata; se invece non lo è, la si modifica leggermente, aggiungendo o correggendo fattori costanti o logaritmici.

Il metodo della sostituzione non si limita dunque a fornire una soluzione numerica: esso insegna a riconoscere e controllare la correttezza delle ipotesi che spesso emergono in modo intuitivo. Negli esempi che seguono applicheremo il procedimento alle stesse tre equazioni già analizzate con l'albero di ricorsione — quelle relative alla ricerca binaria, al Merge Sort e all'algoritmo

di Strassen — per mostrare come il metodo consenta di confermare in modo rigoroso i risultati ottenuti in precedenza.

Esempio 1 — Ricerca binaria

Consideriamo l'equazione

$$T(n) = T(n/2) + 1.$$

L'intuizione suggerisce una crescita logaritmica. Supponiamo quindi $T(n) \leq c \log_2 n$. Sostituendo nell'equazione, otteniamo:

$$T(n) \leq c \log_2 \frac{n}{2} + 1 = c(\log_2 n - 1) + 1 = c \log_2 n - c + 1.$$

Affinché la disuguaglianza $T(n) \leq c \log_2 n$ sia rispettata, è sufficiente che $-c + 1 \leq 0$, cioè $c \geq 1$. Anche in questo caso, la nostra ipotesi è coerente e $T(n) = O(\log n)$.

Esempio 1 — Ricorrenza del Merge Sort

Consideriamo l'equazione

$$T(n) = 2T(n/2) + n,$$

che descrive il tempo di esecuzione del Merge Sort. L'intuizione, come abbiamo visto analizzando l'albero di ricorsione, suggerisce che la complessità sia $O(n \log n)$. Cerchiamo di verificare formalmente questa ipotesi attraverso il metodo della sostituzione.

Supponiamo che

$$T(n) \leq c\, n \log_2 n,$$

per una certa costante positiva c e per tutti i valori di n maggiori di una soglia n_0. Sostituendo questa ipotesi nella ricorrenza otteniamo:

$$T(n) \leq 2 \cdot c \left(\frac{n}{2}\right) \log_2 \frac{n}{2} + n.$$

Semplificando:

$$T(n) \leq cn \log_2 \frac{n}{2} + n = cn(\log_2 n - 1) + n = cn \log_2 n - cn + n.$$

Affinché l'ipotesi sia verificata, dobbiamo avere:

$$T(n) \leq cn \log_2 n,$$

cioè

$$cn \log_2 n - cn + n \leq cn \log_2 n.$$

Questa disuguaglianza è vera ogni volta che $-cn + n \leq 0$, ovvero per $c \geq 1$. Pertanto, scegliendo una costante $c \geq 1$, l'ipotesi è verificata. Abbiamo così dimostrato che $T(n) = O(n \log n)$.

Esempio 3 — Algoritmo di Strassen

Consideriamo infine la ricorrenza

$$T(n) = 7T(n/2) + n^2.$$

Dalla precedente analisi con l'albero di ricorsione abbiamo visto che il termine dominante è $n^{\log_2 7}$. Proviamo a verificarlo con il metodo della sostituzione.

Poniamo

$$T(n) \leq c\, n^{\log_2 7}.$$

Sostituendo nella ricorrenza si ottiene:

$$T(n) \leq 7 \cdot c \left(\frac{n}{2}\right)^{\log_2 7} + n^2.$$

Poiché $(n/2)^{\log_2 7} = n^{\log_2 7}/7$, segue:

$$T(n) \leq 7 \cdot c \frac{n^{\log_2 7}}{7} + n^2 = cn^{\log_2 7} + n^2.$$

Si osserva che, purtroppo, l'induzione fallisce a causa del termine $+n^2$. Il rimedio standard è usare la seguente *ipotesi rinforzata* che "lasci spazio" per assorbire il termine n^2:

$$\boxed{T(n) \;\leq\; c\, n^{\alpha} \;-\; d\, n^2} \qquad (*)$$

per qualche $d > 0$ e per n sufficientemente grande. Alla fine, questo implica comunque $T(n) \leq c\, n^{\alpha}$.

Assumiamo che $(*)$ valga per $n/2$. Allora

$$T(n) = 7T(n/2) + n^2 \;\leq\; 7\Big(c\,(n/2)^{\alpha} - d\,(n/2)^2\Big) + n^2$$

$$= 7c\,\frac{n^{\alpha}}{2^{\alpha}} - \frac{7d}{4}\,n^2 + n^2.$$

Poiché $\alpha = \log_2 7$, si ha $2^{\alpha} = 7$, quindi il primo termine diventa $c\, n^{\alpha}$:

$$T(n) \;\leq\; c\, n^{\alpha} + \left(1 - \tfrac{7}{4}d\right) n^2.$$

Per ottenere l'obbiettivo $T(n) \leq c\,n^\alpha - d\,n^2$ è sufficiente imporre

$$1 - \tfrac{7}{4}d \;\leq\; -d \quad\Longleftrightarrow\quad 1 \;\leq\; \Big(-1 + \tfrac{7}{4}\Big)d \;=\; \tfrac{3}{4}\,d \quad\Longleftrightarrow\quad d \;\geq\; \tfrac{4}{3}.$$

Scegliamo, ad esempio, $d = 2$. Con questa scelta il passo induttivo è verificato:

$$T(n) \;\leq\; c\,n^\alpha - 2\,n^2$$

e quindi possiamo concludere che $T(n) = \Theta(n^{\log_2 7})$.

Attraverso questi esempi si può comprendere il vantaggio del metodo della sostituzione. A differenza del metodo dell'albero, che fornisce una visione grafica della ricorsione, il metodo della sostituzione permette di ottenere una verifica rigorosa del risultato, trasformando l'intuizione in una dimostrazione matematica. Tuttavia, esso richiede di formulare un'ipotesi iniziale ragionevole, che spesso si basa proprio sull'intuizione costruita mediante il metodo dell'albero di ricorsione. I due approcci sono quindi complementari: il primo aiuta a capire "come" cresce la ricorrenza, il secondo permette di dimostrarlo con precisione.

4 Il metodo basato sul Teorema Master

Dopo aver visto il metodo dell'albero di ricorsione, che offre una rappresentazione intuitiva della struttura del costo, e il metodo della sostituzione, che consente una verifica formale dell'ipotesi asintotica, possiamo introdurre un terzo strumento, spesso più diretto ed efficace: il *Teorema Master*. Questo risultato fornisce una regola generale per determinare in modo sistematico l'ordine di grandezza di molte equazioni di ricorrenza della forma

$$T(n) = a\,T\Big(\frac{n}{b}\Big) + f(n),$$

dove:

- $a \geq 1$ è il numero di sottoproblemi in cui viene suddiviso il problema di dimensione n;
- $b > 1$ è il fattore di riduzione della dimensione di ciascun sottoproblema;
- $f(n)$ è il costo del lavoro non ricorsivo, ossia il tempo necessario per suddividere il problema e combinare i risultati.

L'idea del Teorema Master è confrontare la funzione $f(n)$ — che misura il lavoro esterno alla ricorsione — con la quantità $n^{\log_b a}$, che rappresenta il costo totale del lavoro ricorsivo. A seconda di quale dei due termini cresce più rapidamente, si individuano tre comportamenti distinti.

Enunciato del Teorema Master

Sia $T(n) = a\,T(n/b) + f(n)$ con $a \geq 1$, $b > 1$, e $f(n)$ una funzione positiva. Allora valgono i seguenti casi:

Caso 1: se esiste una costante $\varepsilon > 0$ tale che

$$f(n) = O\big(n^{\log_b a - \varepsilon}\big)\,,$$

cioè il lavoro non ricorsivo è asintoticamente più piccolo del lavoro interno alla ricorsione, allora

$$\boxed{T(n) = \Theta\big(n^{\log_b a}\big)}.$$

In questo caso domina il costo generato dalla parte ricorsiva dell'algoritmo (le chiamate interne). Esempio tipico: la moltiplicazione di Strassen, dove $a = 7$, $b = 2$, $f(n) = n^2$.

Caso 2 (generalizzato): se

$$f(n) = \Theta\big(n^{\log_b a} \log^k n\big)\,,$$

ossia il lavoro non ricorsivo ha lo stesso ordine di grandezza del lavoro ricorsivo (a meno di un fattore logaritmico), allora

$$\boxed{T(n) = \Theta\big(n^{\log_b a} \log^{k+1} n\big)}.$$

In questo caso, tutti i livelli dell'albero di ricorsione contribuiscono in modo equivalente al costo totale, e la moltiplicazione per un fattore $\log^k n$ nel termine $f(n)$ si traduce in un incremento di un ordine logaritmico nel costo complessivo.

Caso 3: se esiste una costante $\varepsilon > 0$ tale che

$$f(n) = \Omega\big(n^{\log_b a + \varepsilon}\big)\,,$$

cioè il lavoro non ricorsivo cresce più velocemente del lavoro interno, e se inoltre è verificata una condizione di *regolarità* (detta condizione di dominanza),

$$a\,f\left(\frac{n}{b}\right) \leq c\,f(n) \quad \text{per una costante } c < 1 \text{ e } n \text{ sufficientemente grande,}$$

allora

$$\boxed{T(n) = \Theta(f(n))}.$$

In questo caso, la parte ricorsiva diventa trascurabile rispetto al lavoro non ricorsivo. Esempio tipico: una ricorrenza come $T(n) = 2T(n/2) + n^2$, dove il termine n^2 domina.

4.1 Interpretazione intuitiva del Teorema Master

Il Teorema Master può essere interpretato come un modo sintetico per stabilire quale parte dell'albero di ricorsione domina il costo complessivo:

- nel *primo caso* domina il livello inferiore (cioè le foglie dell'albero), dove si accumula la maggior parte del lavoro ricorsivo;
- nel *secondo caso* ogni livello contribuisce in egual misura (a meno di un fattore logaritmico), e il numero di livelli aggiunge un ulteriore fattore logaritmico;
- nel *terzo caso* domina il livello superiore (la radice), dove il lavoro non ricorsivo è prevalente.

Il Teorema Master rappresenta dunque un ponte tra l'intuizione geometrica fornita dal metodo dell'albero e la precisione analitica della sostituzione: consente di riconoscere immediatamente l'andamento asintotico di molte ricorrenze comuni semplicemente confrontando $f(n)$ con $n^{\log_b a}$. Pur non essendo applicabile a tutti i casi (ad esempio, quando i sottoproblemi hanno dimensioni diverse o $f(n)$ presenta comportamenti irregolari), rimane uno degli strumenti più rapidi e potenti per l'analisi asintotica degli algoritmi ricorsivi.

4.2 Il confronto tra la funzione $f(n)$ e $n^{\log_b a}$

Uno degli aspetti più delicati nell'applicazione del Teorema Master consiste nel comprendere come confrontare la funzione non ricorsiva $f(n)$ con la quantità $n^{\log_b a}$, che rappresenta la crescita del lavoro complessivo generato dalle chiamate ricorsive. In sostanza, $n^{\log_b a}$ descrive quanto "grande" diventa l'albero della ricorsione, mentre $f(n)$ misura il costo aggiuntivo sostenuto a ciascun livello. Il comportamento finale di $T(n)$ dipende da quale di queste due componenti cresce più rapidamente.

Se $f(n)$ cresce molto meno di $n^{\log_b a}$, il termine ricorsivo domina; se cresce molto di più, prevale il termine non ricorsivo; se le due funzioni hanno crescita simile, i contributi si equilibrano e il costo totale si distribuisce tra tutti i livelli. Il parametro ε viene introdotto proprio per formalizzare questa differenza di crescita: esso rappresenta una "distanza esponenziale" tra le due funzioni.

Quando si scrive, ad esempio, $f(n) = O(n^{\log_b a - \varepsilon})$, si intende che $f(n)$ cresce in modo sensibilmente più lento rispetto a $n^{\log_b a}$, tanto da risultare inferiore di un intero fattore polinomiale. Allo stesso modo, $f(n) = \Omega(n^{\log_b a + \varepsilon})$

indica che $f(n)$ cresce più velocemente di $n^{\log_b a}$ per una certa costante positiva ε, cioè che è più "pesante" di qualsiasi moltiplicazione per $\log n$ o per altre funzioni subpolinomiali.

Un modo semplice per orientarsi è il seguente: se tra $f(n)$ e $n^{\log_b a}$ compare una differenza di potenze, anche minima, questa differenza è sufficiente a stabilire il caso corretto del teorema. Al contrario, quando le due funzioni sono dello stesso ordine, ma $f(n)$ contiene un termine moltiplicativo in $\log n$ o una funzione molto vicina alla crescita polinomiale, il problema rientra nel caso "intermedio".

Esempio 1. Consideriamo la ricorrenza $T(n) = 2T(n/2) + n$. Qui $a = 2$, $b = 2$ e quindi $n^{\log_b a} = n$. Poiché $f(n) = n$ ha la stessa crescita, non esiste un $\varepsilon > 0$ tale che $f(n)$ sia né più piccolo né più grande di un fattore polinomiale rispetto a $n^{\log_b a}$: ci troviamo dunque nel caso intermedio, e la soluzione è $T(n) = \Theta(n \log n)$.

Esempio 2. Consideriamo invece $T(n) = 2T(n/2) + n^2$. In questo caso $n^{\log_b a} = n$, ma $f(n) = n^2$ cresce più rapidamente di un intero fattore polinomiale, cioè $f(n) = \Omega(n^{1+\varepsilon})$ con $\varepsilon = 1$. Qui il termine non ricorsivo domina e la soluzione è $T(n) = \Theta(n^2)$.

In generale, il ruolo di ε non è quello di un valore da calcolare, ma di un indicatore concettuale: serve a distinguere tra crescite "molto più piccole" o "molto più grandi" rispetto a $n^{\log_b a}$. Una volta individuato se la funzione $f(n)$ è minore, uguale o maggiore di $n^{\log_b a}$ in senso polinomiale, l'applicazione del teorema diventa immediata.

4.3 Applicazione del Teorema Master

Dopo aver enunciato i tre casi principali del Teorema Master, possiamo ora applicarlo alle ricorrenze già incontrate nelle sezioni precedenti. L'obiettivo non è soltanto ottenere rapidamente la forma asintotica della soluzione, ma anche comprendere in quale dei tre casi rientra ciascun algoritmo e che cosa questo ci dice sul comportamento della sua ricorsione.

Esempio 1 — Ricerca binaria

La ricerca binaria è descritta dalla ricorrenza

$$T(n) = T(n/2) + 1.$$

In questo caso $a = 1$, $b = 2$ e $f(n) = 1$. Calcoliamo il termine di riferimento $n^{\log_b a}$: poiché $\log_2 1 = 0$, si ottiene $n^{\log_2 1} = n^0 = 1$. Confrontiamo ora $f(n)$

con questo valore:

$$f(n) = 1 = \Theta(1) = \Theta(n^{\log_2 1}).$$

Siamo dunque nel *secondo caso* del Teorema Master, quello in cui $f(n)$ ha lo stesso ordine di grandezza del termine ricorsivo.

Applicando la formula corrispondente, otteniamo:

$$T(n) = \Theta\big(n^{\log_2 1} \log n\big) = \Theta(\log n).$$

In ogni passo della ricerca binaria, il problema viene dimezzato, ma il lavoro svolto ad ogni livello (una sola comparazione) è costante. Poiché ci sono $\log_2 n$ livelli fino a ridurre il problema a un singolo elemento, il costo totale cresce in modo logaritmico. Il Teorema Master, in questo caso, conferma in modo immediato ciò che l'intuizione suggerisce: ogni livello contribuisce in modo uniforme, e il numero di livelli determina la crescita complessiva.

Esempio 2 — Merge Sort

La ricorrenza per il Merge Sort è

$$T(n) = 2T(n/2) + n.$$

Qui $a = 2$, $b = 2$ e $f(n) = n$. Il termine di riferimento è

$$n^{\log_b a} = n^{\log_2 2} = n.$$

In questo caso, $f(n)$ ha lo stesso ordine di grandezza di $n^{\log_b a}$:

$$f(n) = \Theta(n^{\log_2 2}) = \Theta(n).$$

Pertanto ci troviamo ancora una volta nel *secondo caso* del Teorema Master.

Applicando la regola del caso 2, otteniamo:

$$T(n) = \Theta\big(n^{\log_2 2} \log n\big) = \Theta(n \log n).$$

Il Merge Sort divide ricorsivamente il problema in due metà e combina i risultati in tempo lineare. Ogni livello della ricorsione lavora sull'intero insieme di dati, ma la profondità dell'albero è logaritmica. Il Teorema Master cattura questa dinamica in modo diretto: poiché il costo di ogni livello è simile, il numero dei livelli ($\log n$) moltiplica il costo per livello (n), generando la complessità $O(n \log n)$. L'analisi, che con il metodo dell'albero richiede un ragionamento per livelli, diventa qui immediata.

Esempio 3 — Algoritmo di Strassen

Consideriamo ora la ricorrenza

$$T(n) = 7T(n/2) + n^2.$$

In questo caso $a = 7$, $b = 2$ e $f(n) = n^2$. Il termine di riferimento è

$$n^{\log_b a} = n^{\log_2 7} \approx n^{2.81}.$$

Confrontiamo $f(n)$ con questo valore:

$$f(n) = n^2 = O\big(n^{\log_2 7 - \varepsilon}\big) \quad \text{con} \quad \varepsilon \approx 0.81.$$

Poiché $f(n)$ cresce più lentamente di $n^{\log_2 7}$, siamo nel *primo caso* del Teorema Master.

Di conseguenza,

$$T(n) = \Theta\big(n^{\log_2 7}\big) \approx \Theta(n^{2.81}).$$

L'algoritmo di Strassen riduce la complessità della moltiplicazione di matrici da $O(n^3)$ a circa $O(n^{2.81})$ grazie al fatto che produce sette sottoproblemi più piccoli invece di otto. In termini di albero di ricorsione, ogni livello genera un numero di nodi che cresce più velocemente del fattore di riduzione della dimensione: la maggior parte del lavoro si concentra quindi nelle foglie dell'albero. Il Teorema Master formalizza questo comportamento in modo immediato, mostrando che la crescita è dominata dalla parte ricorsiva e non dal lavoro di combinazione.

5 Risoluzione di alcune equazioni di ricorrenza

Dopo aver analizzato separatamente i tre principali strumenti per lo studio delle equazioni di ricorrenza — il metodo dell'albero di ricorsione, il metodo della sostituzione e il Teorema Master — possiamo ora applicarli congiuntamente ad alcuni esempi rappresentativi. L'obiettivo di questa sezione non è solo ottenere il risultato finale, ma soprattutto mettere a confronto i diversi approcci, mostrando come ciascuno di essi permetta di giungere alla stessa conclusione seguendo percorsi concettualmente distinti.

Ogni metodo, infatti, offre un diverso punto di vista sull'andamento della ricorrenza: l'albero di ricorsione evidenzia la distribuzione del lavoro nei vari livelli e consente una comprensione visiva della crescita; il metodo della sostituzione fornisce una conferma analitica rigorosa mediante un procedimento

induttivo; infine, il Teorema Master offre un criterio generale e immediato per riconoscere la forma asintotica della soluzione senza dover espandere o verificare manualmente la ricorrenza.

Negli esempi che seguono analizzeremo alcune ricorrenze classiche e altre di forma leggermente diversa, risolvendole con tutti e tre i metodi. In questo modo sarà possibile cogliere, di volta in volta, i punti di forza e le differenze di ciascun approccio, nonché la coerenza dei risultati che ne derivano.

5.1 Ricorrenza: $T(n) = T(n/2) + n$

Questa ricorrenza modella algoritmi che dimezzano la dimensione del problema a ogni passo, svolgendo però un lavoro lineare nella dimensione corrente. Mostriamo tre risoluzioni parallele.

1) Metodo dell'albero di ricorsione. Immaginiamo l'espansione ricorsiva come un albero con un solo figlio per nodo. Al livello k c'è una sola chiamata su istanza di dimensione $n/2^k$ e il lavoro locale è pari a $n/2^k$ (lineare nella dimensione del sottoproblema). La ricorsione termina quando $n/2^L = 1$, cioè a profondità $L = \log_2 n$.

Livello k	# nodi	Dimensione	Costo per nodo	Costo totale livello
0	1	n	n	n
1	1	$n/2$	$n/2$	$n/2$
2	1	$n/4$	$n/4$	$n/4$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$L = \log_2 n$	1	1	1	1

La somma dei costi per livello è

$$T(n) = n + \frac{n}{2} + \frac{n}{4} + \cdots + \frac{n}{2^L}.$$

Prima di usarla, richiamiamo la *serie geometrica finita*: per $0 < r < 1$,

$$\sum_{k=0}^{L} r^k = \frac{1 - r^{L+1}}{1 - r} \quad \Rightarrow \quad \sum_{k=0}^{L} \left(\tfrac{1}{2}\right)^k = \frac{1 - (1/2)^{L+1}}{1 - 1/2} = 2\Big(1 - (1/2)^{L+1}\Big) \le 2.$$

Applicandola con $r = \frac{1}{2}$ otteniamo

$$T(n) = n \sum_{k=0}^{L} \left(\tfrac{1}{2}\right)^k \le 2n \quad \Longrightarrow \quad T(n) = \Theta(n).$$

Intuitivamente, il primo livello pesa di più e i successivi dimezzano via via il contributo: la somma "si ferma" entro un fattore costante del primo termine.

2) Metodo della sostituzione. Proviamo a mostrare direttamente che $T(n) \le c\,n$ per n sufficientemente grande. Sostituendo l'ipotesi nella ricorrenza:

$$T(n) \le c\,(n/2) + n = \Big(\tfrac{c}{2} + 1\Big)n.$$

Affinché $T(n) \le c\,n$, basta avere $\frac{c}{2} + 1 \le c$, cioè $c \ge 2$. Fissando, ad esempio, $T(1) \le 1$ e scegliendo

$$c = \max\{2,\, T(1)\} = 2,$$

l'induzione (sul passaggio $n \mapsto n/2$) risulta valida e conclude che $T(n) \le 2n$ per tutte le potenze di due (e quindi $T(n) = O(n)$). Una stima inferiore lineare si ottiene notando che ogni livello ha costo positivo e il primo da solo vale n, per cui $T(n) \ge n$, dunque $T(n) = \Theta(n)$.

3) Teorema Master. Qui abbiamo $a = 1$, $b = 2$, $f(n) = n$ e $n^{\log_b a} = n^{\log_2 1} = n^0 = 1$. Poiché $f(n) = n = \Omega(n^{0+\varepsilon})$ con $\varepsilon = 1$ e vale la condizione di regolarità

$$a\,f(n/b) = 1 \cdot (n/2) \le c\,n \quad \text{con } c = \tfrac{1}{2} < 1,$$

ci troviamo nel *Caso 3* del Teorema Master e quindi

$$T(n) = \Theta(f(n)) = \Theta(n).$$

L'intuizione è che il lavoro non ricorsivo per livello ($\sim n/2^k$) non si propaga in modo "piatto" come in Merge Sort: il livello più alto è già grande e la coda della serie decresce abbastanza da non superarlo, lasciando una crescita complessiva lineare.

5.2 Ricorrenza: $T(n) = 3T(n/2) + n$

Questa ricorrenza descrive algoritmi che, da un problema di dimensione n, producono tre sottoproblemi di dimensione $n/2$ ciascuno, con un lavoro addizionale proporzionale a n. Vediamo come si comporta con i tre metodi.

1) Metodo dell'albero di ricorsione. Rappresentiamo la ricorsione come un albero: al livello k compaiono 3^k nodi, ognuno su un sottoproblema di dimensione $n/2^k$. Il costo locale (non ricorsivo) per nodo è lineare nella dimensione del sottoproblema, quindi vale $n/2^k$. La profondità è $L = \log_2 n$ (quando $n/2^L = 1$).

Livello k	# nodi	Dimensione	Costo per nodo	Costo totale livello C_k
0	1	n	n	n
1	3	$n/2$	$n/2$	$3\,\frac{n}{2}$
2	9	$n/4$	$n/4$	$9\,\frac{n}{4}$
3	27	$n/8$	$n/8$	$27\,\frac{n}{8}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
k	3^k	$n/2^k$	$n/2^k$	$n\left(\frac{3}{2}\right)^k$

La somma dei costi per livello è

$$T(n) \;=\; \sum_{k=0}^{L} C_k \;=\; n \sum_{k=0}^{L} \left(\frac{3}{2}\right)^k, \qquad L = \log_2 n.$$

Prima di usarla ricordiamo la *serie geometrica finita*: per $r \neq 1$,

$$\sum_{k=0}^{L} r^k \;=\; \frac{r^{L+1}-1}{r-1}.$$

Nel nostro caso $r = \frac{3}{2} > 1$, dunque

$$T(n) \;=\; n \cdot \frac{\left(\frac{3}{2}\right)^{L+1}-1}{\frac{3}{2}-1} \;=\; 2n\Big[\left(\tfrac{3}{2}\right)^{\log_2 n+1} - 1\Big].$$

Per n grande domina il termine con la potenza. Usiamo l'identità $a^{\log_b n} = n^{\log_b a}$:

$$\left(\tfrac{3}{2}\right)^{\log_2 n} = n^{\log_2(3/2)} = n^{\log_2 3 - 1}.$$

Otteniamo così

$$T(n) \;=\; \Theta\big(n \cdot n^{\log_2 3-1}\big) \;=\; \Theta\big(n^{\log_2 3}\big).$$

Intuitivamente, il costo per livello *cresce* (rapporto $3/2 > 1$), quindi i livelli più profondi (le foglie) pesano di più e determinano l'andamento complessivo.

2) Metodo della sostituzione. Poniamo $\alpha = \log_2 3$ e proviamo un'ipotesi rinforzata per assorbire il termine $+n$:

$$T(n) \;\le\; c\,n^{\alpha} \;-\; d\,n,$$

per $c, d > 0$ e n sufficientemente grande. Sostituendo nella ricorrenza,

$$\begin{aligned} T(n) &= 3T(n/2) + n \\ &\leq 3\Big(c\,(n/2)^{\alpha} - d\,(n/2)\Big) + n = c\,n^{\alpha} - \tfrac{3d}{2}\,n + n = c\,n^{\alpha} + (1 - \tfrac{3}{2}d)\,n, \end{aligned}$$

poiché $3 \cdot (n/2)^{\alpha} = 3n^{\alpha}/2^{\alpha} = n^{\alpha}$ (dato che $2^{\alpha} = 3$). Per ottenere $T(n) \leq c\,n^{\alpha} - d\,n$ è sufficiente imporre

$$1 - \tfrac{3}{2}d \;\leq\; -\,d \quad\Longleftrightarrow\quad 1 \;\leq\; \tfrac{1}{2}d \quad\Longleftrightarrow\quad d \;\geq\; 2.$$

Scegliamo $d = 2$. La base induttiva si verifica scegliendo c abbastanza grande. Ad esempio, se $T(1) \leq 1$, l'ipotesi $T(1) \leq c \cdot 1^{\alpha} - 2 \cdot 1 = c - 2$ impone $c \geq 3$. Con $c = 3$, $d = 2$ l'induzione regge e si conclude

$$T(n) \;\leq\; 3\,n^{\alpha} \quad\Rightarrow\quad T(n) = O\big(n^{\log_2 3}\big).$$

Una stima inferiore $\Omega(n^{\log_2 3})$ si ottiene osservando che il livello più profondo dell'albero ha costo dell'ordine di $n^{\log_2 3}$; quindi $T(n) = \Theta(n^{\log_2 3})$.

3) Teorema Master. Qui $a = 3$, $b = 2$, $f(n) = n$ e

$$n^{\log_b a} = n^{\log_2 3}.$$

Poiché $f(n) = n = O\big(n^{\log_2 3 - \varepsilon}\big)$ con $\varepsilon = \log_2 3 - 1 \approx 0.585 > 0$, siamo nel *Caso 1* del Teorema Master (il lavoro ricorsivo prevale). Ne segue

$$T(n) = \Theta\big(n^{\log_2 3}\big).$$

L'intuizione è coerente con l'albero: il costo per livello cresce in modo geometrico (rapporto > 1) e le foglie dominano la somma.

5.3 Ricorrenza: $T(n) = 2T(n/2) + n^2$

Questa ricorrenza descrive algoritmi che dividono il problema in due sottoproblemi di metà dimensione, ma sostengono un lavoro aggiuntivo quadratico nella dimensione corrente. Vediamo come si comporta con i tre metodi.

1) Metodo dell'albero di ricorsione. Rappresentiamo la ricorsione come un albero: al livello k compaiono 2^k nodi, ognuno su un sottoproblema di dimensione $n/2^k$. Il costo locale (non ricorsivo) per nodo è quadratico nella dimensione del sottoproblema:

$$\text{costo per nodo} \;=\; \Big(\tfrac{n}{2^k}\Big)^2 \;=\; \frac{n^2}{4^k}.$$

Il costo *totale* del livello k è quindi

$$C_k \;=\; 2^k \cdot \frac{n^2}{4^k} \;=\; \frac{n^2}{2^k}.$$

La profondità è $L = \log_2 n$ (quando $n/2^L = 1$). La tabella riassume i livelli:

Livello k	# nodi	Dimensione	Costo per nodo	Costo totale C_k
0	1	n	n^2	n^2
1	2	$n/2$	$(n/2)^2 = n^2/4$	$2 \cdot n^2/4 = n^2/2$
2	4	$n/4$	$(n/4)^2 = n^2/16$	$4 \cdot n^2/16 = n^2/4$
3	8	$n/8$	$(n/8)^2 = n^2/64$	$8 \cdot n^2/64 = n^2/8$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
k	2^k	$n/2^k$	$n^2/4^k$	$n^2/2^k$

La somma dei costi per livello è

$$T(n) \;=\; \sum_{k=0}^{L} C_k \;=\; n^2 \sum_{k=0}^{L} \left(\tfrac{1}{2}\right)^k, \qquad L = \log_2 n.$$

Prima di usarla, ricordiamo la *serie geometrica finita*: per $0 < r < 1$,

$$\sum_{k=0}^{L} r^k \;=\; \frac{1 - r^{L+1}}{1 - r} \;\le\; \frac{1}{1 - r}.$$

Applicando $r = \frac{1}{2}$ otteniamo

$$\sum_{k=0}^{L} \left(\tfrac{1}{2}\right)^k \;=\; 2\Big(1 - (\tfrac{1}{2})^{L+1}\Big) \;\le\; 2.$$

Quindi

$$T(n) \;=\; n^2 \sum_{k=0}^{L} \left(\tfrac{1}{2}\right)^k \;\le\; 2n^2.$$

Poiché già il primo livello vale n^2, si ha anche $T(n) \ge n^2$. Ne segue

$$\boxed{T(n) = \Theta(n^2)}.$$

L'intuizione è che il costo per livello *decresce* di un fattore $1/2$; il primo livello domina e la coda geometrica aggiunge solo un fattore costante.

2) Metodo della sostituzione. Mostriamo direttamente che $T(n) \le c\,n^2$ per n sufficientemente grande. Sostituendo l'ipotesi nella ricorrenza:

$$T(n) \;\le\; 2 \cdot c\,(n/2)^2 + n^2 \;=\; c\,\frac{n^2}{2} + n^2 \;=\; \Big(\tfrac{c}{2} + 1\Big) n^2.$$

Per avere $T(n) \le c\,n^2$ basta imporre $\frac{c}{2} + 1 \le c$, cioè $c \ge 2$. Fissando, ad esempio, $T(1) \le 1$ e scegliendo $c = \max\{2,\, T(1)\} = 2$, l'induzione è verificata e conclude che $T(n) \le 2n^2$ (per potenze di due, e quindi in generale fino a costanti). Per il vincolo inferiore, il primo livello fornisce $T(n) \ge n^2$, dunque

$$\boxed{T(n) = \Theta(n^2)}.$$

3) Teorema Master. Qui $a = 2$, $b = 2$, $f(n) = n^2$ e quindi

$$n^{\log_b a} \;=\; n^{\log_2 2} \;=\; n.$$

Poiché $f(n) = n^2 = \Omega(n^{1+\varepsilon})$ con $\varepsilon = 1$, e la condizione di regolarità vale

$$a\,f(n/b) \;=\; 2 \cdot (n/2)^2 \;=\; \tfrac{n^2}{2} \;\le\; c\,n^2 \quad (\text{con } c = \tfrac{1}{2} < 1),$$

siamo nel *Caso 3* del Teorema Master, e dunque

$$\boxed{T(n) = \Theta(f(n)) = \Theta(n^2)}.$$

L'interpretazione coincide con l'albero: il termine non ricorsivo è così grande che i livelli successivi contano via via meno; il costo è determinato dal livello più alto.

5.4 Ricorrenza: $T(n) = T(n/2) + \log_2 n$

Questa ricorrenza modella una decomposizione in un solo sottoproblema dimezzato, con un lavoro locale che cresce come il logaritmo della dimensione corrente. Vediamo i tre approcci.

1) Metodo dell'albero di ricorsione. L'albero ha un solo figlio per nodo: al livello k c'è una chiamata su istanza di dimensione $n/2^k$, e il costo locale vale $\log_2(n/2^k) = \log_2 n - k$. La profondità è $L = \log_2 n$ (quando $n/2^L = 1$). La struttura per livelli è:

Livello k	# nodi	Dimensione	Costo per nodo	Costo totale livello
0	1	n	$\log_2 n$	$\log_2 n$
1	1	$n/2$	$\log_2 n - 1$	$\log_2 n - 1$
2	1	$n/4$	$\log_2 n - 2$	$\log_2 n - 2$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$L = \log_2 n$	1	1	0	0

Il costo totale è quindi la somma

$$T(n) \;=\; \sum_{k=0}^{L} \big(\log_2 n - k\big), \qquad L = \log_2 n.$$

Prima di usarla, ricordiamo la *serie aritmetica*:

$$\sum_{k=0}^{L} k \;=\; \frac{L(L+1)}{2}.$$

Con $m = \log_2 n$ otteniamo

$$T(n) = \sum_{k=0}^{m} (m-k) = (m+1)m - \frac{m(m+1)}{2} = \frac{m(m+1)}{2} = \frac{(\log_2 n)(\log_2 n + 1)}{2}.$$

Ne segue immediatamente

$$\boxed{T(n) = \Theta\big((\log n)^2\big)}.$$

L'intuizione è che il costo per livello decresce *linearmente* con la profondità: sommando una "scaletta" da $\log n$ a 0 si ottiene, appunto, un valore quadratico nel logaritmo.

2) Metodo della sostituzione. Mostriamo un limite superiore esplicito con ipotesi rinforzata, così da evitare soglie implicite. Vogliamo provare

$$T(n) \;\le\; c\,(\log_2 n)^2 \;-\; d\,\log_2 n$$

per qualche scelta di $c, d > 0$ e per n abbastanza grande. Sostituendo nella ricorrenza:

$$\begin{aligned} T(n) &\le T(n/2) + \log_2 n \\ &\le c\,\big(\log_2(n/2)\big)^2 - d\,\log_2(n/2) + \log_2 n \\ &= c\,(\log_2 n - 1)^2 - d\,(\log_2 n - 1) + \log_2 n \\ &= c(\log_2 n)^2 + (-2c - d + 1)\,\log_2 n + (c + d). \end{aligned}$$

Per ottenere $T(n) \le c(\log_2 n)^2 - d \log_2 n$ è sufficiente che

$$(-2c - d + 1) \log_2 n + (c + d) \;\le\; -d \log_2 n,$$

ossia

$$(-2c + 1) \log_2 n + (c + d) \;\le\; 0.$$

Scegliendo, ad esempio, $c = 1$ e $d = 1$, la disuguaglianza diventa

$$(-1) \log_2 n + 2 \;\le\; 0 \quad \Longleftrightarrow \quad \log_2 n \ge 2,$$

quindi vale per $n \ge 4$. Con $T(1)$ costante, la base si verifica facilmente per una scelta opportuna delle costanti (qui $c = 1$, $d = 1$ e soglia $n_0 = 4$). Conclude che

$$T(n) \;\le\; (\log_2 n)^2 - \log_2 n \;\le\; (\log_2 n)^2,$$

cioè $T(n) = O\big((\log n)^2\big)$. L'espressione esatta trovata con l'albero fornisce anche il bound inferiore, per cui

$$\boxed{T(n) = \Theta\big((\log n)^2\big)}.$$

3) Teorema Master. Qui $a = 1$, $b = 2$, $f(n) = \log_2 n$ e quindi

$$n^{\log_b a} \;=\; n^{\log_2 1} \;=\; n^0 \;=\; 1.$$

Osserviamo che $f(n) = \Theta\big(n^0 \log^1 n\big) = \Theta\big(n^{\log_b a} \log^k n\big)$ con $k = 1$. Siamo dunque nel *Caso 2 (generalizzato)* del Teorema Master, e ne segue che

$$\boxed{T(n) = \Theta\big(n^{\log_b a} \log^{k+1} n\big) = \Theta(\log^2 n)}.$$

L'interpretazione coincide con l'albero di ricorsione: ogni livello contribuisce in modo comparabile (circa $\log n$), e su $\Theta(\log n)$ livelli la somma dei contributi produce un costo totale $\Theta(\log^2 n)$.

5.5 Ricorrenza: $T(n) = 4T(n/2) + n$

Questa ricorrenza descrive un algoritmo che, da un problema di dimensione n, genera quattro sottoproblemi di dimensione $n/2$ ciascuno, sostenendo un lavoro addizionale lineare in n. Vediamo i tre approcci.

1) Metodo dell'albero di ricorsione. Al livello k ci sono 4^k nodi, ognuno su un sottoproblema di dimensione $n/2^k$. Il lavoro non ricorsivo per nodo è proporzionale alla dimensione del sottoproblema, cioè $n/2^k$. Dunque il costo *totale* del livello k è

$$C_k \;=\; 4^k \cdot \frac{n}{2^k} \;=\; n\,(2^k).$$

La profondità dell'albero è $L = \log_2 n$ (quando $n/2^L = 1$). La struttura per livelli è:

Livello k	# nodi	Dimensione	Costo per nodo	Costo totale C_k
0	1	n	n	n
1	4	$n/2$	$n/2$	$4 \cdot (n/2) = 2n$
2	16	$n/4$	$n/4$	$16 \cdot (n/4) = 4n$
k	4^k	$n/2^k$	$n/2^k$	$n\,2^k$
$L = \log_2 n$	n^2	1	1	n^2

La somma dei costi per livello è

$$T(n) \;=\; \sum_{k=0}^{L} C_k \;=\; n \sum_{k=0}^{L} 2^k, \qquad L = \log_2 n.$$

Ricordiamo la *serie geometrica finita*: per $r \neq 1$,

$$\sum_{k=0}^{L} r^k \;=\; \frac{r^{L+1} - 1}{r - 1}.$$

Applicando $r = 2$ otteniamo

$$\sum_{k=0}^{L} 2^k \;=\; 2^{L+1} - 1 \quad \Rightarrow \quad T(n) \;=\; n\,(2^{L+1} - 1).$$

Poiché $2^L = 2^{\log_2 n} = n$, segue

$$T(n) \;=\; n\,(2n - 1) \;=\; 2n^2 - n \;=\; \Theta(n^2).$$

Intuitivamente il costo per livello *cresce* con rapporto 2; l'ultimo livello (le foglie) pesa circa n^2 e domina l'intera somma.

2) Metodo della sostituzione. Vogliamo mostrare un limite superiore quadratico con ipotesi rinforzata per assorbire il termine lineare. Supponiamo

$$T(n) \;\le\; c\,n^2 \;-\; d\,n,$$

per $c, d > 0$ e n abbastanza grande. Sostituendo nella ricorrenza:

$$\begin{aligned} T(n) &= 4T(n/2) + n \\ &\le 4\Big(c\,(n/2)^2 - d\,(n/2)\Big) + n \\ &= c\,n^2 - 2d\,n + n \\ &= c\,n^2 + (1 - 2d)\,n. \end{aligned}$$

Per avere $T(n) \le c\,n^2 - d\,n$ è sufficiente $1 - 2d \le -d$, cioè $d \ge 1$. Scegliamo $d = 1$. Per la base, se $T(1) \le 1$, la condizione $T(1) \le c \cdot 1^2 - 1$ impone $c \ge 2$. Con $c = 2$, $d = 1$ l'induzione è valida e otteniamo

$$T(n) \;\le\; 2n^2 - n \;\le\; 2n^2,$$

quindi $T(n) = O(n^2)$. Un limite inferiore $\Omega(n^2)$ si ha già dal costo dell'ultimo livello (che vale n^2), per cui

$$\boxed{T(n) = \Theta(n^2)}.$$

3) Teorema Master. Qui $a = 4$, $b = 2$, $f(n) = n$ e

$$n^{\log_b a} \;=\; n^{\log_2 4} \;=\; n^2.$$

Poiché $f(n) = n = O\big(n^{2-\varepsilon}\big)$ con $\varepsilon = 1 > 0$, ricadiamo nel *Caso 1* del Teorema Master (il termine ricorsivo domina). Ne segue

$$\boxed{T(n) = \Theta\big(n^{\log_2 4}\big) = \Theta(n^2)}.$$

L'interpretazione coincide con quella dell'albero: il numero di sottoproblemi cresce così rapidamente da far aumentare il costo per livello; le foglie, numerose ($\sim n^2$), determinano la crescita complessiva.

5.6 Ricorrenza: $T(n) = T(n/3) + n$

Questa ricorrenza modella algoritmi che riducono la dimensione del problema di un fattore 3 ad ogni passo, sostenendo però un lavoro lineare nella dimensione corrente. Vediamo i tre approcci.

1) Metodo dell'albero di ricorsione. L'albero ha un solo figlio per nodo. Al livello k compare una sola chiamata su istanza di dimensione $n/3^k$; il lavoro locale è $n/3^k$. La ricorsione termina quando $n/3^L = 1$, cioè alla profondità $L = \log_3 n$.

Livello k	# nodi	Dimensione	Costo per nodo	Costo totale livello
0	1	n	n	n
1	1	$n/3$	$n/3$	$n/3$
2	1	$n/9$	$n/9$	$n/9$
3	1	$n/27$	$n/27$	$n/27$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$L = \log_3 n$	1	1	1	1

La somma dei costi per livello è

$$T(n) \;=\; n + \frac{n}{3} + \frac{n}{9} + \cdots + \frac{n}{3^L} \;=\; n \sum_{k=0}^{L} \left(\tfrac{1}{3}\right)^k.$$

Prima di usarla, ricordiamo la *serie geometrica finita*: per $0 < r < 1$,

$$\sum_{k=0}^{L} r^k \;=\; \frac{1 - r^{L+1}}{1 - r} \;\le\; \frac{1}{1 - r}.$$

Applicando $r = \frac{1}{3}$ otteniamo

$$\sum_{k=0}^{L} \left(\tfrac{1}{3}\right)^k = \frac{1 - (1/3)^{L+1}}{1 - 1/3} = \frac{3}{2}\Big(1 - (\tfrac{1}{3})^{L+1}\Big) \;\le\; \frac{3}{2}.$$

Quindi

$$T(n) \;\le\; \frac{3}{2}\, n.$$

D'altra parte il primo livello vale n, dunque $T(n) \ge n$. Ne segue

$$\boxed{T(n) = \Theta(n)}.$$

Intuitivamente, il costo per livello decresce geometricamente (rapporto 1/3), quindi il primo livello domina e la coda aggiunge solo un fattore costante.

2) Metodo della sostituzione. Mostriamo un limite superiore lineare. Supponiamo

$$T(n) \;\le\; c\,n$$

per n sufficientemente grande. Sostituendo nella ricorrenza:

$$T(n) \;\le\; c\,(n/3) + n \;=\; \Big(\tfrac{c}{3} + 1\Big)n.$$

Affinché $T(n) \le c\,n$, è sufficiente

$$\tfrac{c}{3} + 1 \;\le\; c \quad\Longleftrightarrow\quad 1 \;\le\; \tfrac{2}{3}c \quad\Longleftrightarrow\quad c \;\ge\; \tfrac{3}{2}.$$

Con $T(1) \le 1$, scegliamo ad esempio $c = \frac{3}{2}$ (o più grande) e la base si verifica facilmente; segue $T(n) \le \frac{3}{2}n$ e dunque $T(n) = O(n)$. Il bound inferiore $T(n) \ge n$ (dal primo livello) conclude che

$$\boxed{T(n) = \Theta(n)}.$$

3) Teorema Master. Qui $a = 1$, $b = 3$, $f(n) = n$ e il termine di confronto è

$$n^{\log_b a} \;=\; n^{\log_3 1} \;=\; n^0 \;=\; 1.$$

Poiché $f(n) = n = \Omega\big(n^{0+\varepsilon}\big)$ con $\varepsilon = 1$, e vale la condizione di regolarità

$$a\,f(n/b) = 1 \cdot (n/3) \le c\,n \quad \text{con } c = \tfrac{1}{3} < 1,$$

siamo nel *Caso 3* del Teorema Master e dunque

$$\boxed{T(n) = \Theta(f(n)) = \Theta(n)}.$$

L'interpretazione coincide con l'albero: il lavoro non ricorsivo per livello diminuisce rapidamente; la somma è dominata dal livello iniziale.

5.7 Ricorrenza: $T(n) = 2T(n/4) + n$

Questa ricorrenza descrive un algoritmo che suddivide il problema in due sottoproblemi molto più piccoli (un quarto della dimensione originale), sostenendo però un lavoro addizionale lineare nella dimensione corrente. Vediamo i tre approcci.

1) Metodo dell'albero di ricorsione. Al livello k compaiono 2^k nodi, ciascuno su un sottoproblema di dimensione $n/4^k$. Il lavoro non ricorsivo per nodo è proporzionale alla dimensione del sottoproblema, quindi $n/4^k$. Il costo *totale* del livello k è

$$C_k \;=\; 2^k \cdot \frac{n}{4^k} \;=\; n\left(\frac{1}{2}\right)^k.$$

La profondità è $L = \log_4 n$ (quando $n/4^L = 1$). La tabella riassume i livelli:

Livello k	# nodi	Dimensione	Costo per nodo	Costo totale C_k
0	1	n	n	n
1	2	$n/4$	$n/4$	$2 \cdot (n/4) = n/2$
2	4	$n/16$	$n/16$	$4 \cdot (n/16) = n/4$
3	8	$n/64$	$n/64$	$8 \cdot (n/64) = n/8$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
k	2^k	$n/4^k$	$n/4^k$	$n\,(1/2)^k$

La somma dei costi per livello è

$$T(n) \;=\; \sum_{k=0}^{L} C_k \;=\; n \sum_{k=0}^{L} \left(\tfrac{1}{2}\right)^k, \qquad L = \log_4 n.$$

Prima di usarla, ricordiamo la *serie geometrica finita*: per $0 < r < 1$ vale

$$\sum_{k=0}^{L} r^k \;=\; \frac{1 - r^{L+1}}{1 - r} \;\le\; \frac{1}{1 - r}.$$

Applicando $r = \frac{1}{2}$ otteniamo

$$\sum_{k=0}^{L} \left(\tfrac{1}{2}\right)^k = 2\Big(1 - \left(\tfrac{1}{2}\right)^{L+1}\Big) \;\le\; 2,$$

da cui

$$T(n) \;\le\; 2n.$$

Poiché già il primo livello vale n, segue anche $T(n) \ge n$. Concludiamo che

$$\boxed{T(n) = \Theta(n)}.$$

L'intuizione è che il costo per livello decresce geometricamente (rapporto 1/2), quindi il primo livello domina e i livelli successivi aggiungono soltanto un fattore costante.

2) Metodo della sostituzione. Dimostriamo un limite superiore lineare. Supponiamo

$$T(n) \;\le\; c\,n$$

per n sufficientemente grande. Sostituendo l'ipotesi nella ricorrenza:

$$T(n) \;\le\; 2 \cdot c\,(n/4) + n \;=\; \Big(\tfrac{c}{2} + 1\Big) n.$$

Affinché $T(n) \le c\,n$ sia verificata, è sufficiente

$$\tfrac{c}{2} + 1 \;\le\; c \quad\Longleftrightarrow\quad c \;\ge\; 2.$$

Scelta ad esempio la base $T(1) \le 1$, possiamo fissare $c = 2$ e l'induzione è valida (per potenze di quattro, e quindi in generale a meno di costanti). Otteniamo così $T(n) \le 2n$, cioè $T(n) = O(n)$. Il vincolo inferiore $T(n) \ge n$ (dal primo livello) completa la dimostrazione:

$$\boxed{T(n) = \Theta(n)}.$$

3) Teorema Master. Qui $a = 2$, $b = 4$, $f(n) = n$ e

$$n^{\log_b a} \;=\; n^{\log_4 2} \;=\; n^{1/2}.$$

Poiché $f(n) = n = \Omega\big(n^{1/2+\varepsilon}\big)$ con $\varepsilon = \frac{1}{2} > 0$ e vale la condizione di regolarità

$$a\,f(n/b) = 2 \cdot (n/4) = \tfrac{n}{2} \;\le\; c\,n \quad \text{con } c = \tfrac{1}{2} < 1,$$

ci troviamo nel *Caso 3* del Teorema Master. Ne segue

$$\boxed{T(n) = \Theta(f(n)) = \Theta(n)}.$$

L'interpretazione coincide con l'albero: la riduzione è molto forte (un quarto), quindi il numero di nodi non cresce abbastanza da compensare la perdita di dimensione; il costo è dominato dallo strato superiore.

5.8 Ricorrenza: $T(n) = 2T(n/2) + \sqrt{n}$

Questa ricorrenza modella un algoritmo che raddoppia il numero di sottoproblemi dimezzando la dimensione, ma sostiene un lavoro locale sublineare, pari a $\sqrt{n}$. Vediamo i tre approcci.

1) Metodo dell'albero di ricorsione. Al livello k compaiono 2^k nodi, ciascuno su un sottoproblema di dimensione $n/2^k$. Il lavoro non ricorsivo per nodo è

$$\sqrt{n/2^k} = \frac{\sqrt{n}}{2^{k/2}}.$$

Il costo *totale* del livello k è quindi

$$C_k \;=\; 2^k \cdot \frac{\sqrt{n}}{2^{k/2}} \;=\; \sqrt{n}\,2^{k/2} \;=\; \sqrt{n}\,(\sqrt{2})^k.$$

La profondità è $L = \log_2 n$ (quando $n/2^L = 1$). La tabella riassume i livelli principali:

Livello k	# nodi	Dimensione	Costo per nodo	Costo totale C_k
0	1	n	$\sqrt{n}$	$\sqrt{n}$
1	2	$n/2$	$\sqrt{n/2}$	$2\sqrt{n/2} = \sqrt{2}\,\sqrt{n}$
2	4	$n/4$	$\sqrt{n/4}$	$4\sqrt{n/4} = 2\,\sqrt{n}$
k	2^k	$n/2^k$	$\sqrt{n}/2^{k/2}$	$\sqrt{n}\,(\sqrt{2})^k$
$L = \log_2 n$	n	1	1	$\sqrt{n}\,(\sqrt{2})^{\log_2 n} = n$

Il costo totale è

$$T(n) \;=\; \sum_{k=0}^{L} C_k \;=\; \sqrt{n}\sum_{k=0}^{L}(\sqrt{2})^k, \qquad L = \log_2 n.$$

Richiamiamo la *serie geometrica finita*: per $r \neq 1$,

$$\sum_{k=0}^{L} r^k \;=\; \frac{r^{L+1}-1}{r-1}.$$

Con $r = \sqrt{2} > 1$,

$$T(n) \;=\; \sqrt{n}\,\frac{(\sqrt{2})^{L+1}-1}{\sqrt{2}-1} \;=\; \sqrt{n}\,\frac{\sqrt{2}\,(\sqrt{2})^{\log_2 n}-1}{\sqrt{2}-1}.$$

Usando $(\sqrt{2})^{\log_2 n} = n^{1/2}$,

$$T(n) \;=\; \frac{\sqrt{2}}{\sqrt{2}-1}\,n \;-\; \frac{\sqrt{n}}{\sqrt{2}-1} \;=\; \Theta(n).$$

L'intuizione è che il costo per livello *cresce* con rapporto $\sqrt{2}$, perciò l'ultimo livello (le foglie) vale circa n e domina la somma.

2) Metodo della sostituzione. Proviamo un'ipotesi rinforzata che assorba il termine $+\sqrt{n}$:

$$T(n) \;\le\; c\,n \;-\; d\,\sqrt{n},$$

per $c, d > 0$ e n grande. Sostituendo nella ricorrenza,

$$\begin{aligned} T(n) &= 2T(n/2) + \sqrt{n} \\ &\le 2\Big(c\,(n/2) - d\,\sqrt{n/2}\Big) + \sqrt{n} = c\,n - 2d\,\frac{\sqrt{n}}{\sqrt{2}} + \sqrt{n} \\ &= c\,n \;+\; \Big(1 - d\sqrt{2}\Big)\sqrt{n}. \end{aligned}$$

Per ottenere $T(n) \le c\,n - d\,\sqrt{n}$ è sufficiente

$$1 - d\sqrt{2} \;\le\; -d \iff 1 \;\le\; d(\sqrt{2}-1) \iff d \;\ge\; \frac{1}{\sqrt{2}-1} \;=\; \sqrt{2}+1.$$

Scegliamo dunque $d = \sqrt{2}+1$. Per la base, se $T(1) \le 1$, la condizione $T(1) \le c \cdot 1 - d \cdot 1$ impone $c \ge 1 + d$. Con $d = \sqrt{2}+1$ basta, ad esempio, $c = 4$. Con tali costanti l'induzione è valida e conclude che $T(n) \le c\,n$, quindi $T(n) = O(n)$. Per il limite inferiore, l'ultimo livello dell'albero contribuisce n, dunque $T(n) \ge n$ e complessivamente

$$\boxed{T(n) = \Theta(n)}.$$

3) Teorema Master. Qui $a = 2$, $b = 2$, per cui $n^{\log_b a} = n^{\log_2 2} = n$. Il termine non ricorsivo è $f(n) = \sqrt{n} = n^{1/2}$, che soddisfa

$$f(n) = O\big(n^{1-\varepsilon}\big) \quad \text{con } \varepsilon = \tfrac{1}{2} > 0.$$

Siamo quindi nel *Caso 1* del Teorema Master (il lavoro ricorsivo domina) e

$$\boxed{T(n) = \Theta\big(n^{\log_2 2}\big) = \Theta(n)}.$$

L'intuizione coincide con quella dell'albero: il numero di nodi cresce abbastanza da far "vincere" le foglie; il termine $\sqrt{n}$ non è sufficiente a uniformare i livelli come accade nel Merge Sort.

6 Esercizi sulle equazioni di ricorrenza

Per consolidare la comprensione dei metodi studiati, si propongono di seguito alcune equazioni di ricorrenza da risolvere utilizzando, per ciascuna, tutti e tre gli approcci trattati: il metodo dell'albero di ricorsione, il metodo della sostituzione e il Teorema Master. Lo scopo degli esercizi è quello di esercitarsi nell'individuare la struttura ricorsiva di un algoritmo, riconoscere la forma della corrispondente equazione di ricorrenza e determinare l'ordine di grandezza della sua soluzione asintotica. Le ricorrenze proposte sono volutamente varie, in modo da coprire differenti combinazioni dei parametri a, b e $f(n)$ e consentire una visione complessiva del comportamento dei tre metodi. Non è necessario disegnare l'albero in modo formale, ma è importante comprenderne la logica e il modo in cui si distribuisce il costo tra i livelli.

Si risolvano quindi le seguenti ricorrenze, determinandone il comportamento asintotico mediante i tre metodi studiati.

(1) $T(n) = 3T(n/3) + n$,

(2) $T(n) = 2T(n/3) + n \log_2 n$,

(3) $T(n) = 8T(n/2) + n^3$,

(4) $T(n) = T(n/2) + \sqrt{n}\, \log_2 n$,

(5) $T(n) = 2T(n/4) + n \log_2 n$,

(6) $T(n) = 3T(n/4) + n^2$,

(7) $T(n) = 5T(n/3) + n \log_2 n$,

(8) $T(n) = T(n/2) + n^2$,

(9) $T(n) = 5T(n/3) + n^2$,

(10) $T(n) = 7T(n/4) + n(\log n)^3$,

(11) $T(n) = 4T(n/4) + n \log_2 n$,

(12) $T(n) = 3T(n/2) + \sqrt{n}$,

(13) $T(n) = 6T(n/5) + n\sqrt{\log n}$,

(14) $T(n) = 9T(n/3) + n^2$,

(15) $T(n) = 2T(n/3) + n$,

(16) $T(n) = 2T(n/2) + n\sqrt{n}$,

(17) $T(n) = 7T(n/2) + n^3$,

(18) $T(n) = 2T(n/4) + n^2$,

(19) $T(n) = T(n/2) + \log_2^2 n$,

(20) $T(n) = 3T(n/3) + n^{3/2}$.