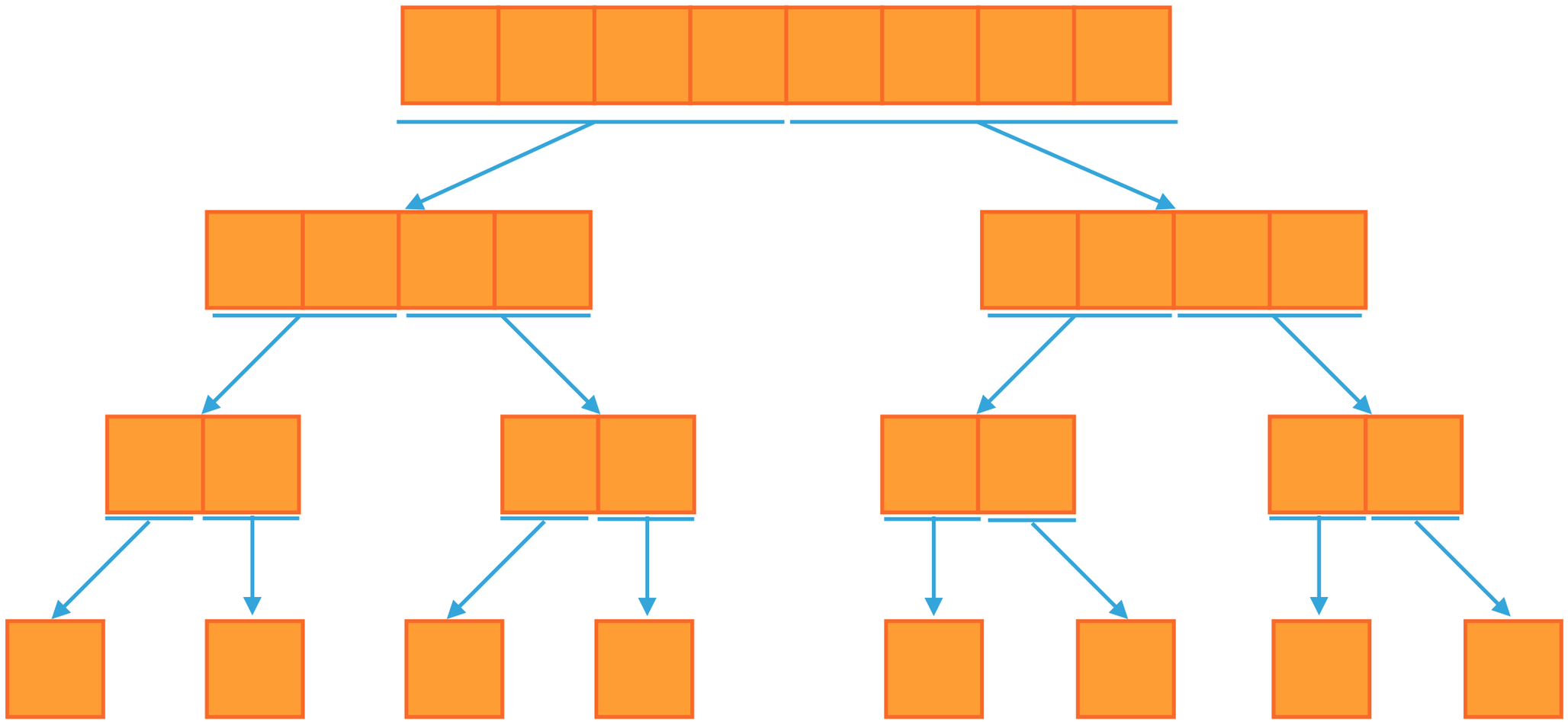
PROGRAMMAZIONE DINAMICA
PIÙ LUNGA SOTTOSEQUENZA COMUNE (LCS)

ALGORITMI E STRUTTURE DATI

DIVIDE ET IMPERA

- Metodo comune di risoluzione dei problemi
- Si basa sull'idea che possiamo esprimere una soluzione come combinazione di soluzioni di sotto-problemi più piccoli
- Un esempio standard: mergesort
- Proviamo a esplorare meglio la struttura dei sotto-problemi che andiamo a risolvere

MERGESORT

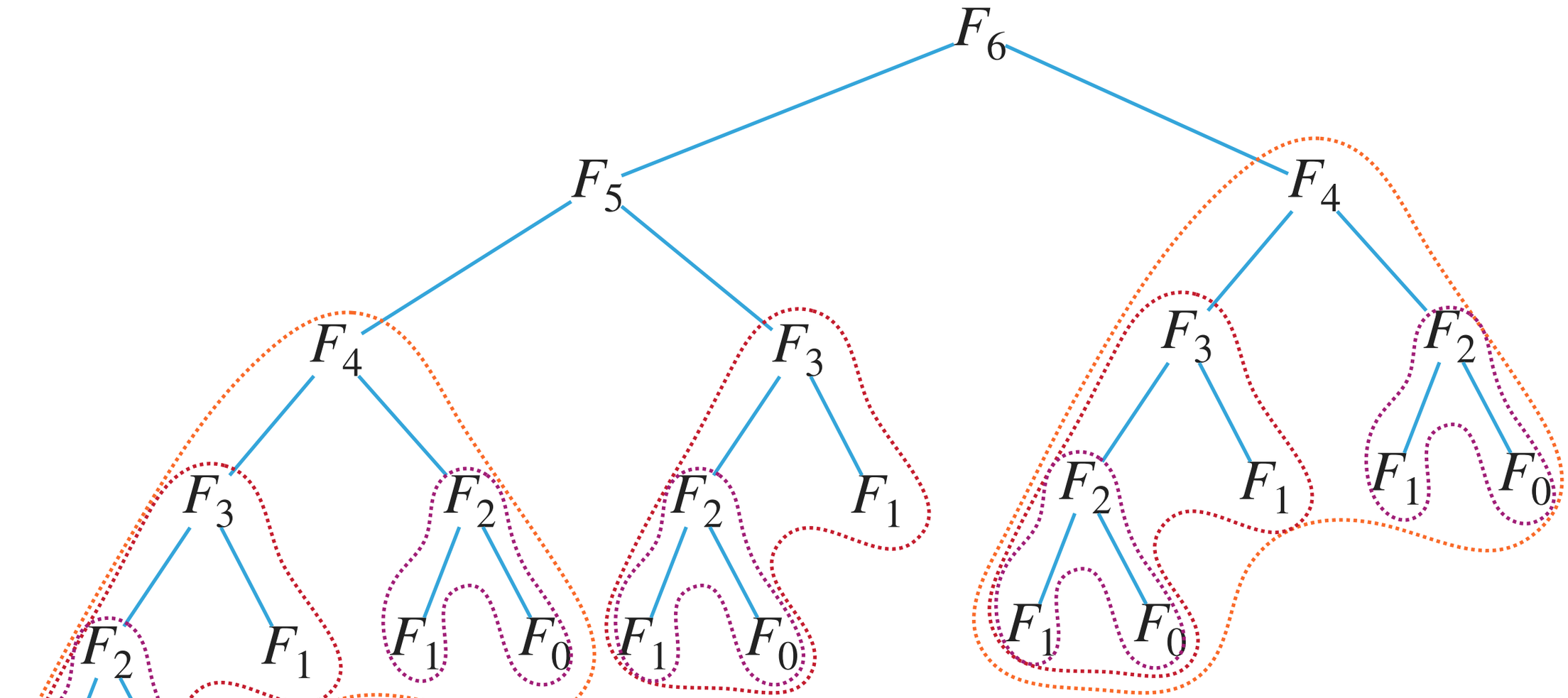


La dimensione dei sotto-array da ordinare si dimezza ma, ancora più importante, ciascuno dei sottoproblemi è indipendente!
E.g., ordinare la prima metà dell'array non è influenzato da ordinare la seconda metà.

DIVIDE ET IMPERA: LIMITAZIONI

- Ma cosa succede quando i sotto-problemi che dobbiamo risolvere non sono indipendenti?
- Un esempio tipico è il calcolo diretto dei numeri di Fibonacci:
  - $F_n = F_{n-1} + F_{n-2}$, ma a sua volta la risoluzione di F_{n-1} richiede la risoluzione di F_{n-2}, dato che $F_{n-1} = F_{n-2} + F_{n-3}$
- Vediamo l'albero dei sotto-problemi

FIBONACCI: SOTTO-PROBLEMI RIPETUTI



Calcoliamo F_4 per due volte

Calcoliamo F_3 per tre volte

Calcoliamo F_2 per cinque volte

APPROCCIARE I SOTTO-PROBLEMI RIPETUTI

- Quando abbiamo dei sotto-problemi che si ripetono un semplice approccio ricorsivo immediato non è generalmente efficiente
- Nel caso dei numeri di Fibonacci più il sotto-problema è piccolo e più volte lo risolviamo.
- Questo numero cresce come il numero di Fibonacci, quindi esponenzialmente rispetto a n

APPROCCIARE I SOTTO-PROBLEMI RIPETUTI

- Se abbiamo un sotto-problema ripetuto è inutile risolverlo più volte, possiamo salvarci il risultato e riusarlo quando ci serve

- Questa è l'idea di base della **programmazione dinamica** e della **memoizzazione** (no, non è un errore di battitura)

- La principale differenza tra questi due approcci è data da come andiamo a costruire la soluzione finale: in un approccio bottom-up o top-down

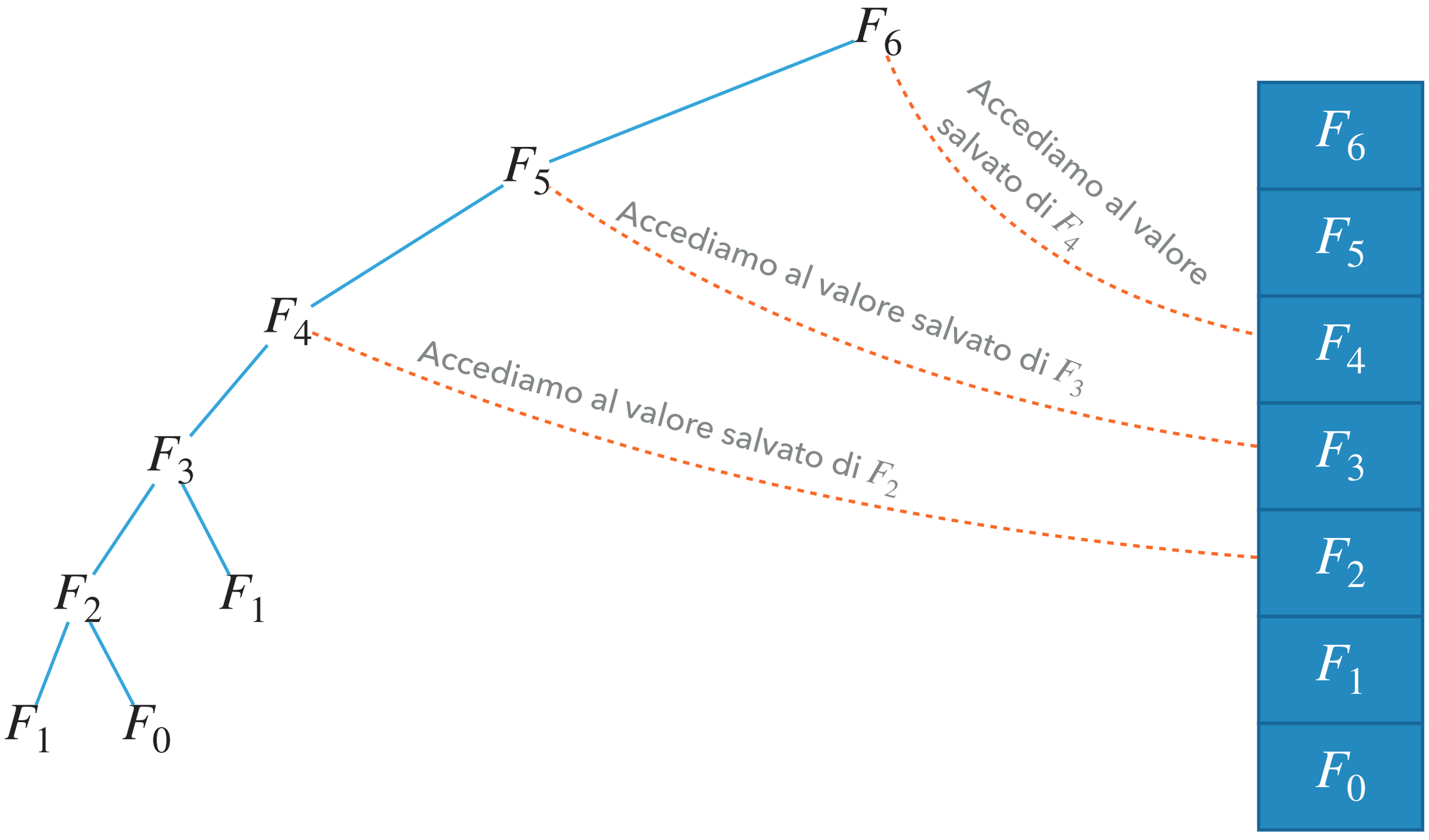
TOP-DOWN E BOTTOM-UP

- Un approccio **top-down** è quello di scomporre il problema in sotto-problemi più piccoli in modo ricorsivo
  - Per Fibonacci: il normale approccio di scrivere ricorsivamente $F_n = F_{n-1} + F_{n-2}$
- L'approccio **bottom-up** è quello di partire dai casi base e combinarli ripetutamente fino a quando non si ottiene il risultato atteso
  - Per Fibonacci: per calcolare F_n partiamo da F_0 e F_1 e calcoliamo, in ordine $F_2, F_3, \ldots, F_n$

MEMOIZZAZIONE

- L'idea è quella di tenere una tabella di sotto-problemi che abbiamo già risolto e, prima di fare una chiamata ricorsiva, verificare se abbiamo già la soluzione:
  - Se la soluzione è presente nella tabella usiamo quella senza fare chiamate ricorsive
  - Altrimenti facciamo la normale chiamata ricorsiva e, in aggiunta, salviamo il risultato ottenuto nella tabella
- In questo modo risolviamo ogni sotto-problema una volta sola

FIBONACCI: SOTTO-PROBLEMI RIPETUTI



IL TERMINE "PROGRAMMAZIONE DINAMICA"

- Ideata da Richard Bellman negli anni '50 (lo stesso dell'algoritmo di Bellman-Ford)
- Il nome "programmazione dinamica" non è molto informativo:
  - il termine programmazione è da intendersi nel senso di "pianificazione"
  - "Dinamica" è stato scelto, tra gli altri motivi, perché "it's impossible to use the word dynamic in a pejorative sense"

IDEA DI BASE

- Come idea di base della programmazione dinamica, pensiamo a definire una ricorrenza che lega soluzioni del problema a soluzioni di sotto-problemi
- Costruiamo una tabella di soluzioni
- Inseriamo nella tabella le soluzioni dei casi base
- Riempiamo iterativamente la tabella fino a quando non abbiamo ottenuto la soluzione al problema di partenza

UN PROBLEMA D'ESEMPIO

- "problema del taglio della barra" o "rod cutting problem"
- Abbiamo una barra di metallo di lunghezza n che possiamo tagliare in pezzi di dimensione intera: $1,2,\ldots,n$
- Un pezzo di lunghezza i viene venduto al prezzo p_i
- Vogliamo trovare un algoritmo che ci dica il modo migliore di tagliare la barra per massimizzare il prezzo di vendita totale

SOTTO-PROBLEMI RIPETUTI

Barra da tagliare, lunghezza $n = 6$

$4.5 \times 3 = 13.5$

16.7

$12.6 + 4.5 + 2.6 = 19.7$

Possiamo definire questo problema in modo ricorsivo?

Tabella dei prezzi

Lunghezza	Prezzo
1	2.6
2	4.5
3	6.1
4	12.6
5	10.4
6	16.7

UN PROBLEMA D'ESEMPIO

- Vogliamo massimizzare la somma dei prezzi di vendita dei singoli tagli
- Casi conosciuti: nessun taglio, barra di lunghezza $1,\ldots,n$
- Data una sbarra di lunghezza n, indichiamo con r_n il miglior prezzo di vendita totale
- Proviamo ad esprimere in modo ricorsivo r_n

UN PROBLEMA D'ESEMPIO

- Data una barra di lunghezza n abbiamo le seguenti possibilità:
  - La vendiamo intera, ottenendo p_n
  - Facciamo il primo taglio di lunghezza $k < n$, ottenendo r_k per il pezzo tagliato e r_{n-k} per la parte rimanente
- Quindi per trovare r_n calcoliamo $\max\{p_n, r_1 + r_{n-1}, r_2 + r_{n-2}, \dots\}$

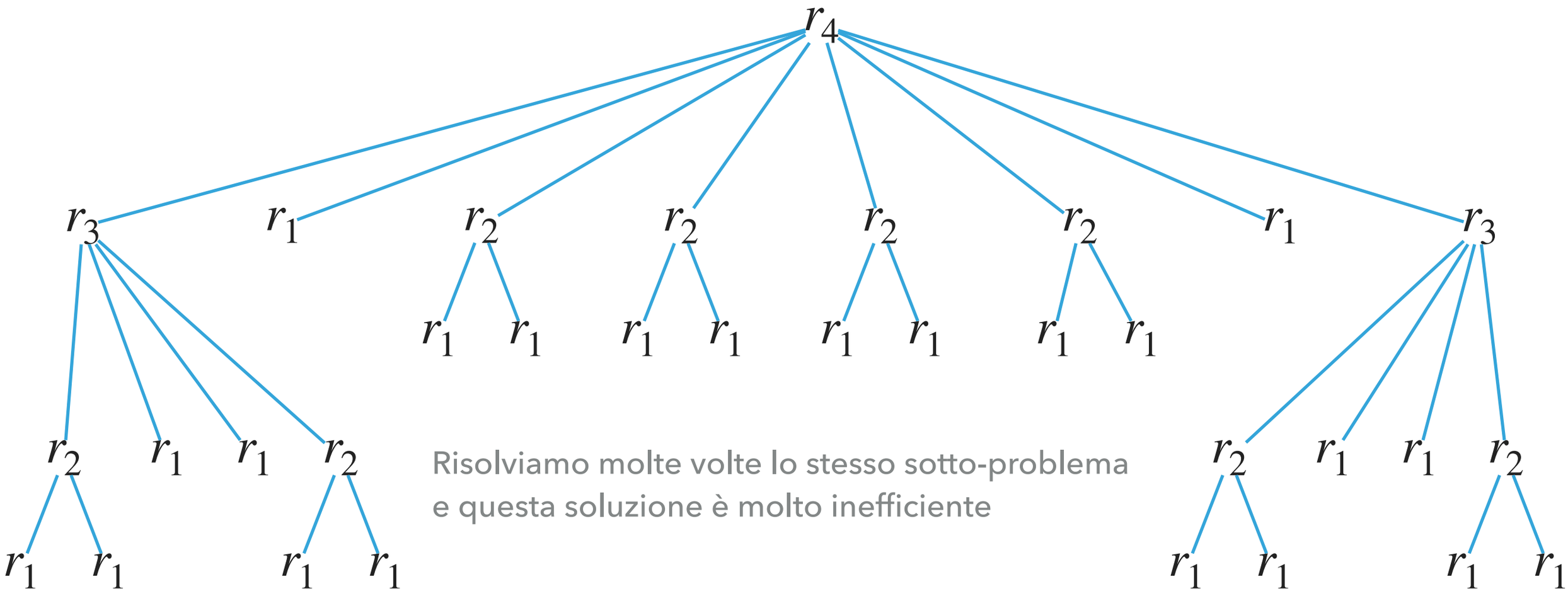
UN PROBLEMA D'ESEMPIO

- $r_n = \max\{p_n, r_1 + r_{n-1}, r_2 + r_{n-2}, \ldots\}$ ci mostra che il problema può essere ricondotto alla risoluzione di sotto-problemi ottimi più piccoli!
- Ora il problema è quello di calcolare r_n in modo efficiente
- Proviamo con una semplice implementazione ricorsiva

PSEUDOCODICE: TAGLIO DELLA BARRA

```
Parametri: lunghezza della barra n, tabella dei prezzi p
if n == 1: # non possiamo dividere ulteriormente la barra
    return p[0]
prezzo_vendita = p[n-1] # la nostra stima iniziale è senza tagli
for i in range(1, n): # per ogni possibile posizione di taglio
    tmp = taglio_barra(i, p) + taglio_barra(n - i, p) # chiamata ricorsiva
    if prezzo_vendita < tmp: # se abbiamo migliorato la stima la aggiorniamo
        prezzo_vendita = tmp
return prezzo_vendita
```

TAGLIO DELLA BARRA: SOTTO-PROBLEMI RIPETUTI



Risolviamo molte volte lo stesso sotto-problema e questa soluzione è molto inefficiente

Una stima un poco grezza del tempo di calcolo:

$$T(n) = 2\left(\sum_{i=1}^{n-1} T(i)\right) \geq 2T(n-1) = O(2^n)$$

UN PROBLEMA D'ESEMPIO: MIGLIORARE LA SOLUZIONE

- Dobbiamo trovare un modo più efficiente di risolvere il problema
- Applichiamo un approccio bottom-up per calcolare r_n
- Teniamo un array di lunghezza n che salva in posizione $i - 1$ il valore r_i ed iniziamo a riempire l'array da $r_1 = p_1$
- Per le posizioni successive usiamo la definizione:
  $r_n = \max\{p_n, r_1 + r_{n-1}, r_2 + r_{n-2}, \dots\}$

PSEUDOCODICE: TAGLIO DELLA BARRA

```
Parametri: lunghezza della barra n, tabella dei prezzi p
r = [0] * n
for i in range(0, n-1): # calcoliamo i valori di r_i a partire dal minore
    r[i] = p[i] # stima iniziale del valore di r[i]
    for j in range(0, i): # proviamo a vedere se potevamo fare un taglio
        if r[j] + r[i-j-1] > r[i]: # se il taglio migliora la situazione
            r[i] = r[j] + r[i-j-1] # aggiorniamo la nostra stima di r_{i-1}
return r[n-1] # il valore per r_n si troverà nell'ultima posizione dell'array
```

SOLUZIONE CON PROGRAMMAZIONE DINAMICA

Barra da tagliare, lunghezza $n = 6$

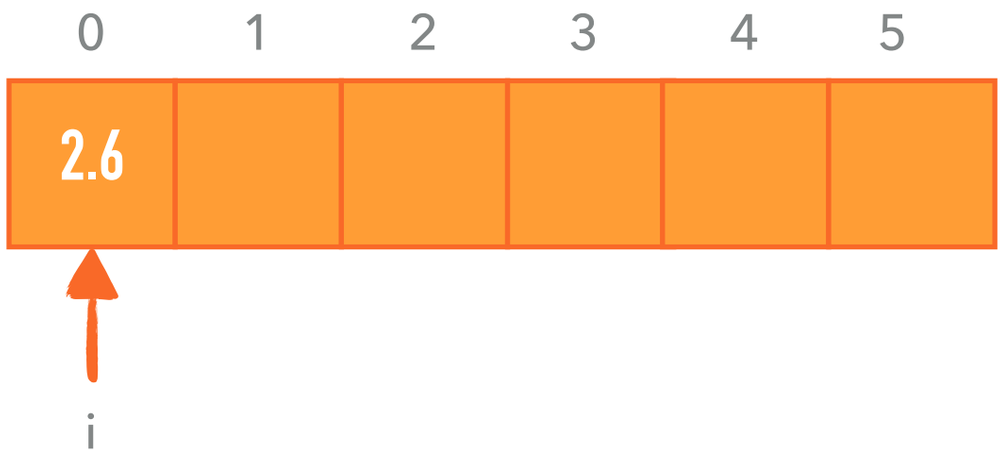
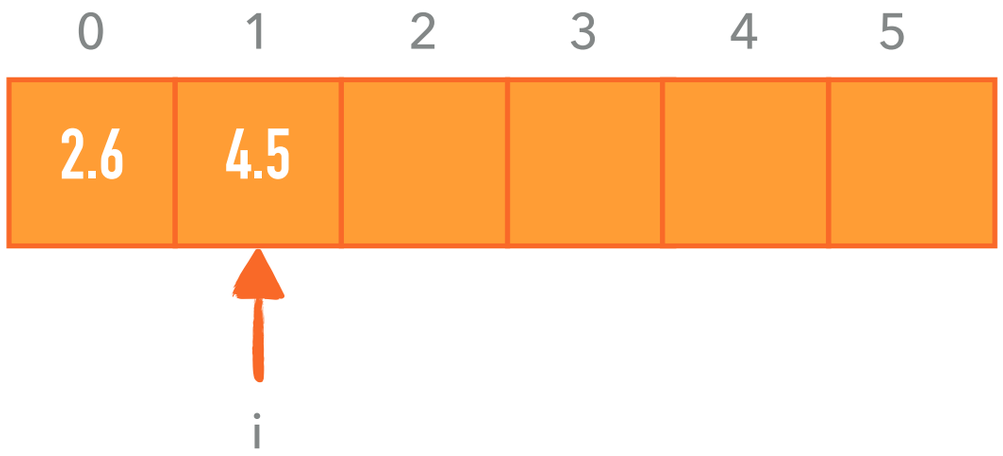


Tabella dei prezzi

Lunghezza	Prezzo
1	2.6
2	4.5
3	6.1
4	12.6
5	10.4
6	16.7

SOLUZIONE CON PROGRAMMAZIONE DINAMICA

Barra da tagliare, lunghezza $n = 6$



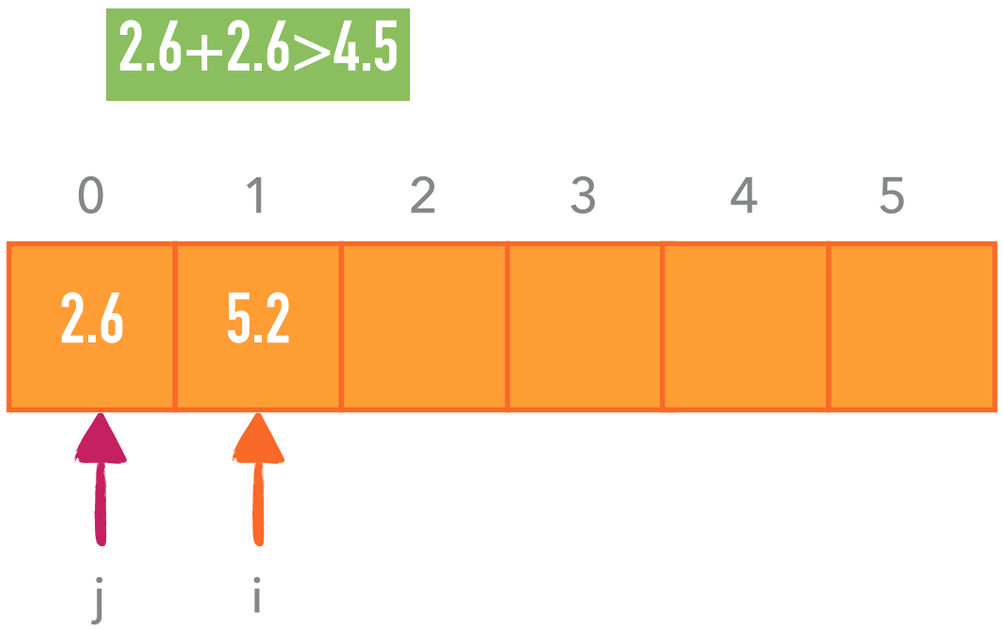
La nostra stima iniziale è semplicemente caso senza tagli

Tabella dei prezzi

Lunghezza	Prezzo
1	2.6
2	4.5
3	6.1
4	12.6
5	10.4
6	16.7

SOLUZIONE CON PROGRAMMAZIONE DINAMICA

Barra da tagliare, lunghezza $n = 6$



Facciamo variare j aggiornando la nostra stima

Tabella dei prezzi

Lunghezza	Prezzo
1	2.6
2	4.5
3	6.1
4	12.6
5	10.4
6	16.7

SOLUZIONE CON PROGRAMMAZIONE DINAMICA

Barra da tagliare, lunghezza $n = 6$

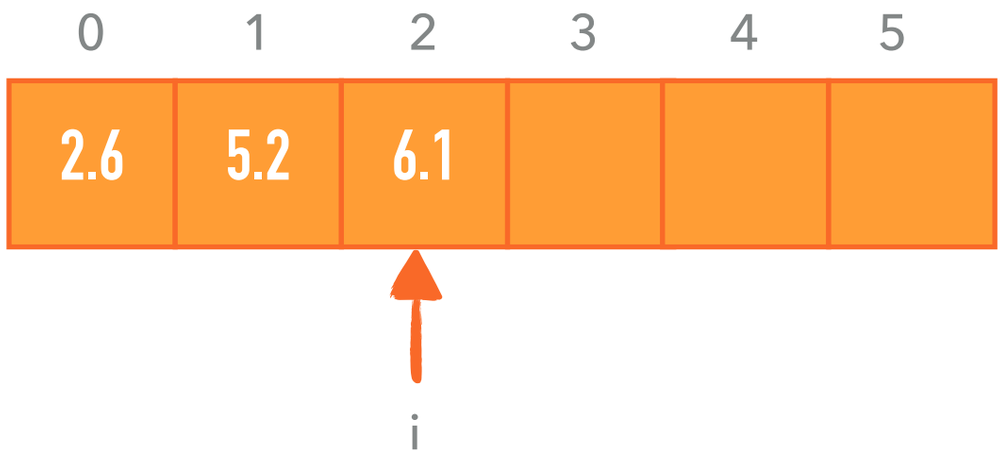


Tabella dei prezzi

Lunghezza	Prezzo
1	2.6
2	4.5
3	6.1
4	12.6
5	10.4
6	16.7

SOLUZIONE CON PROGRAMMAZIONE DINAMICA

Barra da tagliare, lunghezza $n = 6$

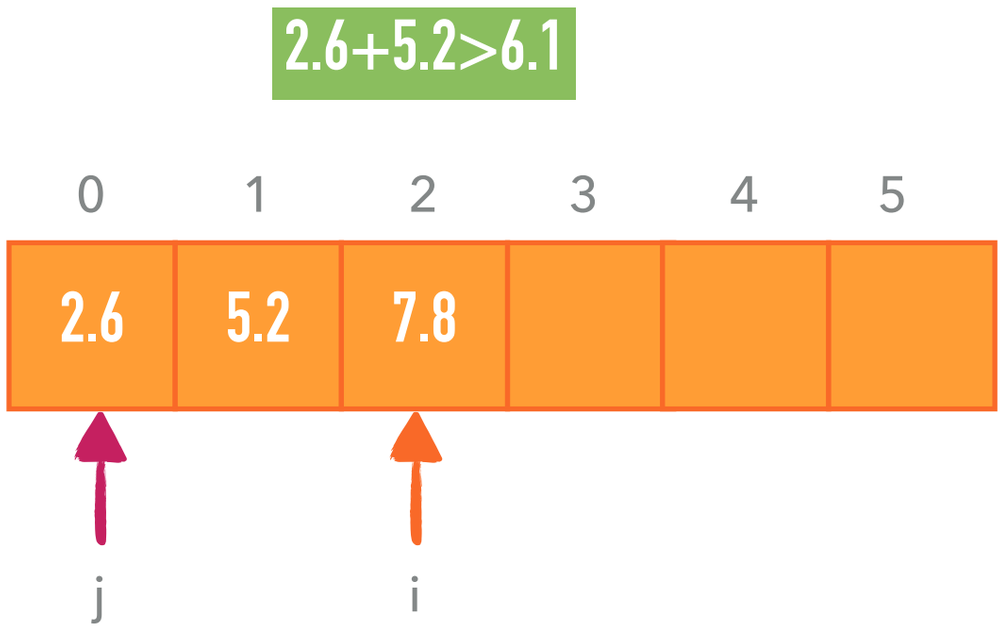


Tabella dei prezzi

Lunghezza	Prezzo
1	2.6
2	4.5
3	6.1
4	12.6
5	10.4
6	16.7

SOLUZIONE CON PROGRAMMAZIONE DINAMICA

Barra da tagliare, lunghezza $n = 6$

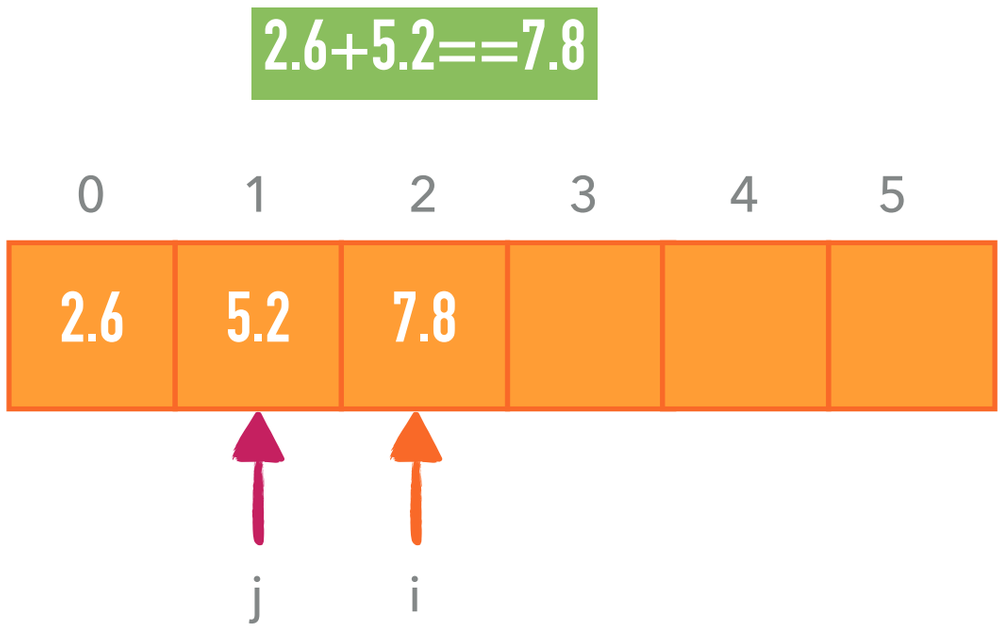


Tabella dei prezzi

Lunghezza	Prezzo
1	2.6
2	4.5
3	6.1
4	12.6
5	10.4
6	16.7

SOLUZIONE CON PROGRAMMAZIONE DINAMICA

Barra da tagliare, lunghezza $n = 6$

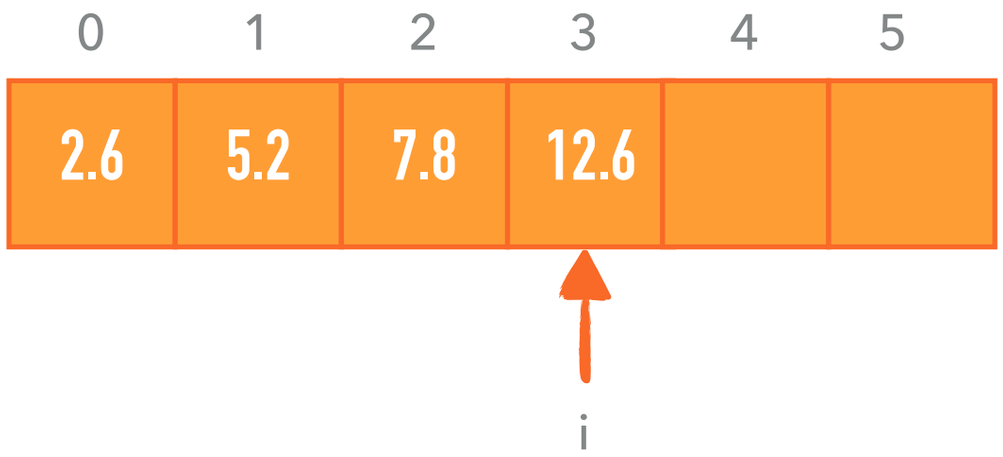


Tabella dei prezzi

Lunghezza	Prezzo
1	2.6
2	4.5
3	6.1
4	12.6
5	10.4
6	16.7

SOLUZIONE CON PROGRAMMAZIONE DINAMICA

Barra da tagliare, lunghezza $n = 6$

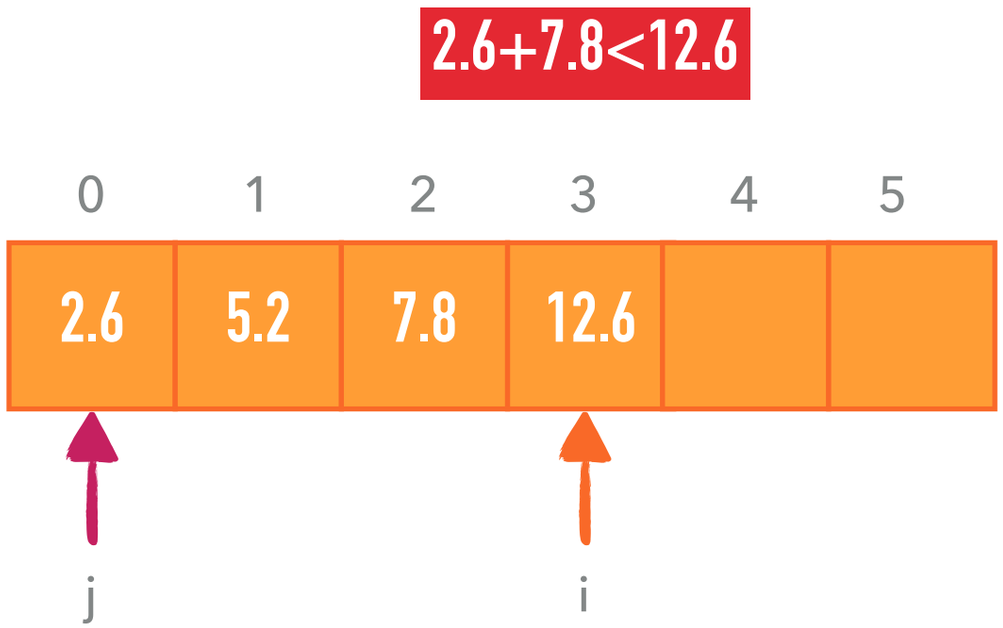


Tabella dei prezzi

Lunghezza	Prezzo
1	2.6
2	4.5
3	6.1
4	12.6
5	10.4
6	16.7

SOLUZIONE CON PROGRAMMAZIONE DINAMICA

Barra da tagliare, lunghezza $n = 6$

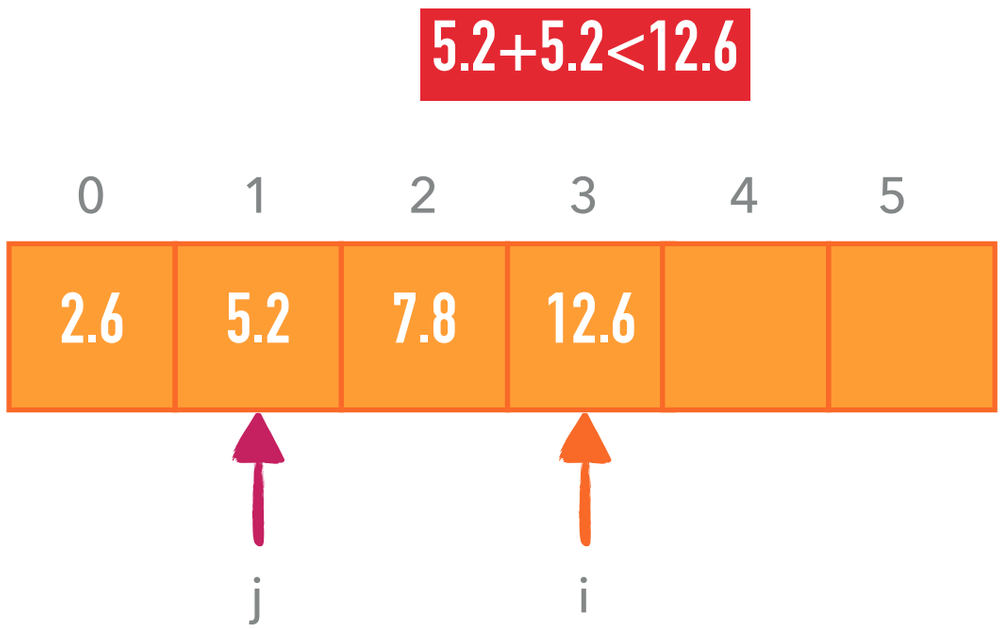


Tabella dei prezzi

Lunghezza	Prezzo
1	2.6
2	4.5
3	6.1
4	12.6
5	10.4
6	16.7

SOLUZIONE CON PROGRAMMAZIONE DINAMICA

Barra da tagliare, lunghezza $n = 6$

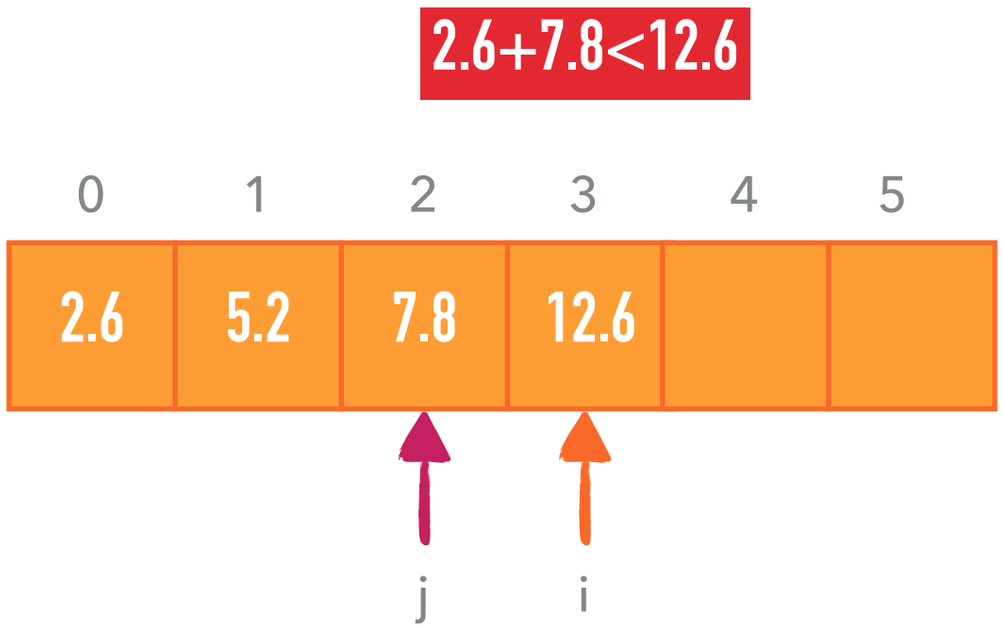


Tabella dei prezzi

Lunghezza	Prezzo
1	2.6
2	4.5
3	6.1
4	12.6
5	10.4
6	16.7

SOLUZIONE CON PROGRAMMAZIONE DINAMICA

Barra da tagliare, lunghezza $n = 6$

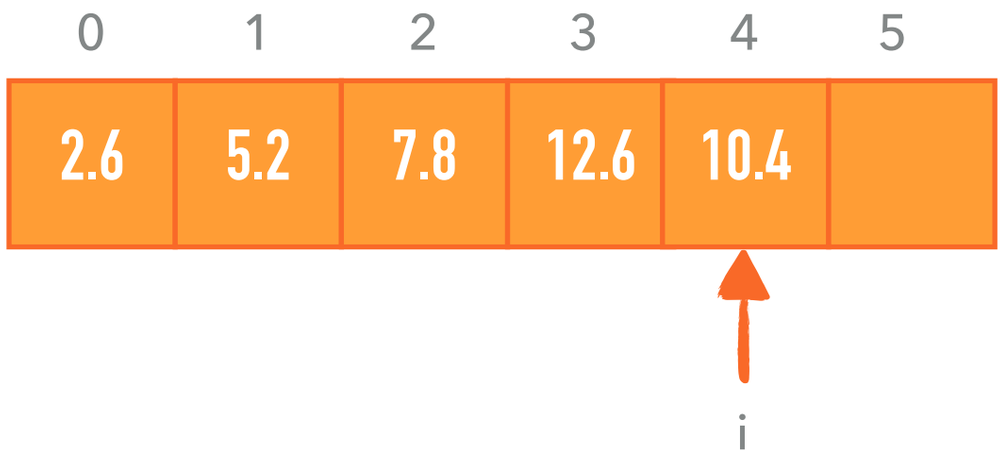


Tabella dei prezzi

Lunghezza	Prezzo
1	2.6
2	4.5
3	6.1
4	12.6
5	10.4
6	16.7

SOLUZIONE CON PROGRAMMAZIONE DINAMICA

Barra da tagliare, lunghezza $n = 6$

12.6+2.6>10.4

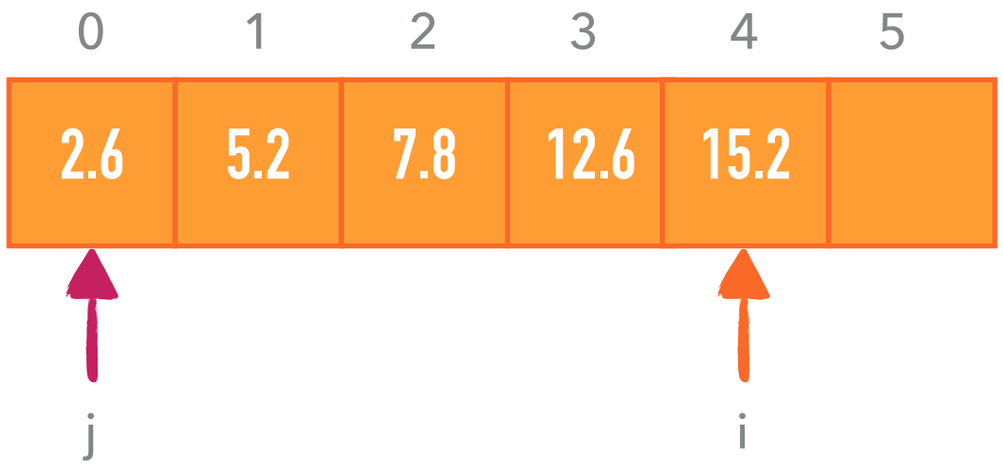


Tabella dei prezzi

Lunghezza	Prezzo
1	2.6
2	4.5
3	6.1
4	12.6
5	10.4
6	16.7

SOLUZIONE CON PROGRAMMAZIONE DINAMICA

Barra da tagliare, lunghezza $n = 6$

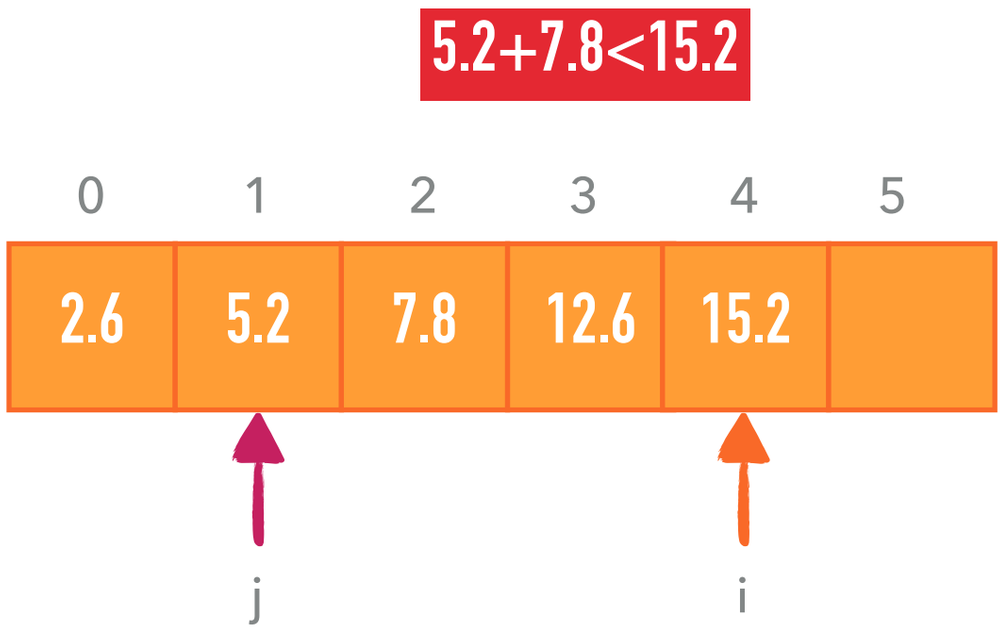


Tabella dei prezzi

Lunghezza	Prezzo
1	2.6
2	4.5
3	6.1
4	12.6
5	10.4
6	16.7

SOLUZIONE CON PROGRAMMAZIONE DINAMICA

Barra da tagliare, lunghezza $n = 6$

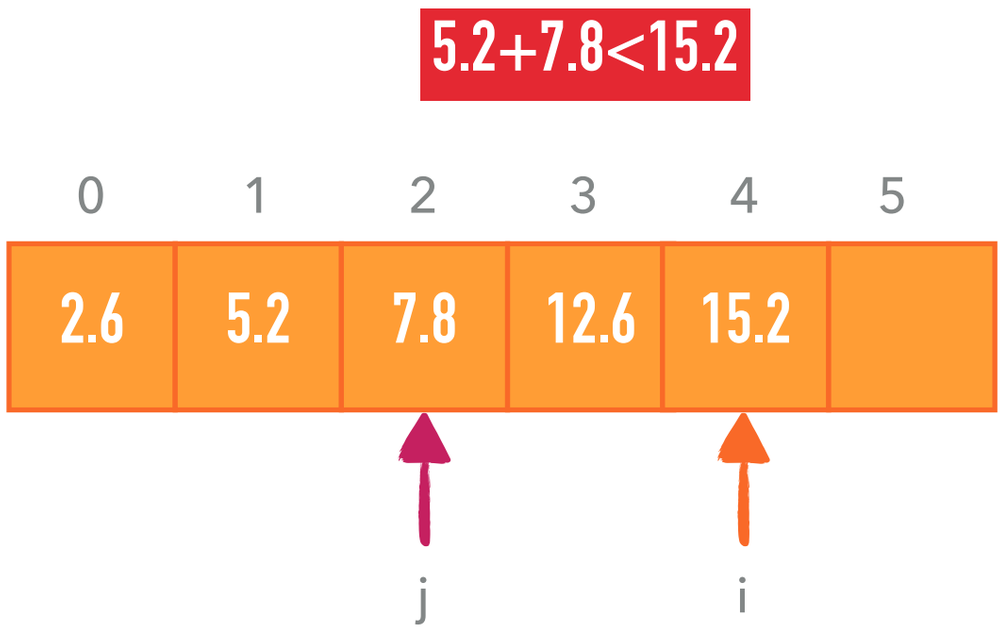


Tabella dei prezzi

Lunghezza	Prezzo
1	2.6
2	4.5
3	6.1
4	12.6
5	10.4
6	16.7

SOLUZIONE CON PROGRAMMAZIONE DINAMICA

Barra da tagliare, lunghezza $n = 6$

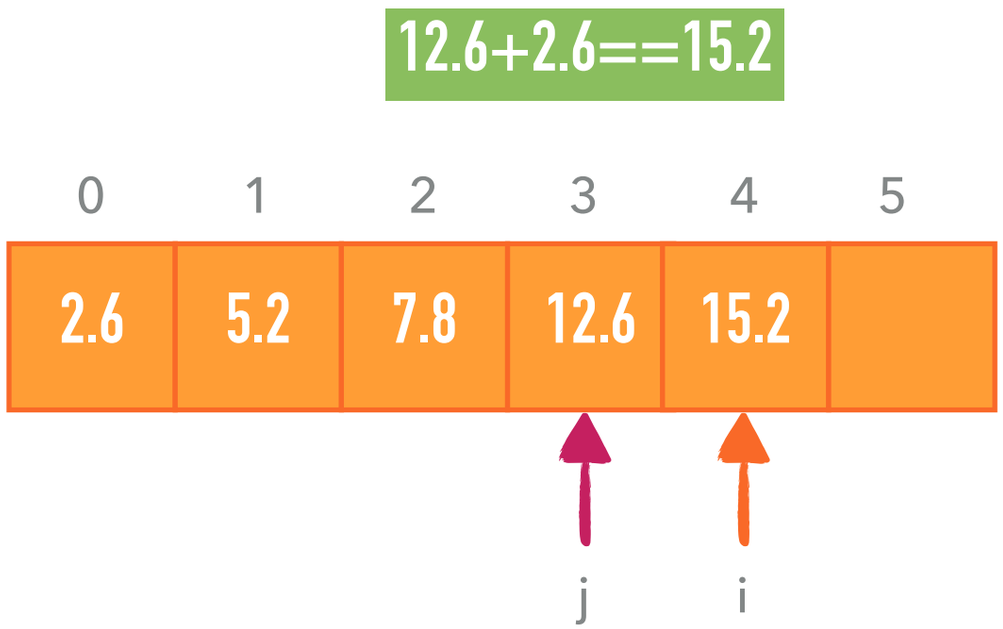


Tabella dei prezzi

Lunghezza	Prezzo
1	2.6
2	4.5
3	6.1
4	12.6
5	10.4
6	16.7

SOLUZIONE CON PROGRAMMAZIONE DINAMICA

Barra da tagliare, lunghezza $n = 6$

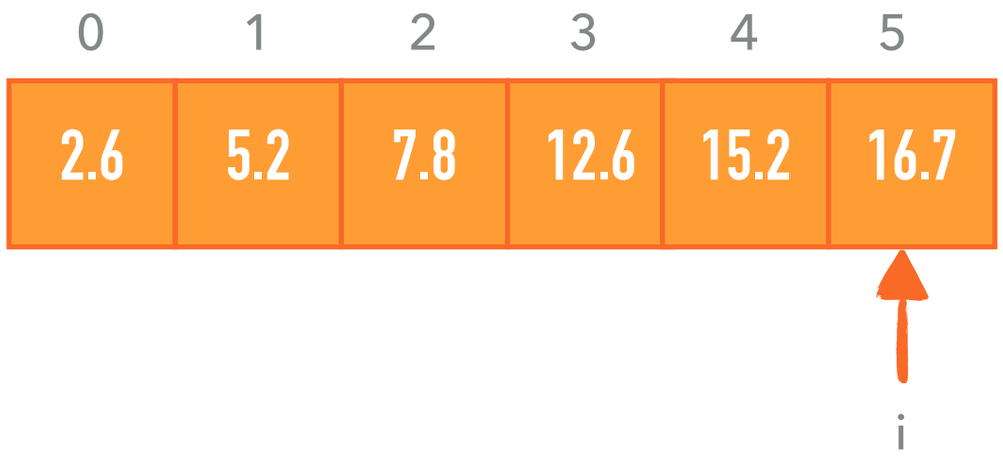


Tabella dei prezzi

Lunghezza	Prezzo
1	2.6
2	4.5
3	6.1
4	12.6
5	10.4
6	16.7

SOLUZIONE CON PROGRAMMAZIONE DINAMICA

Barra da tagliare, lunghezza $n = 6$

15.2+2.6>16.7

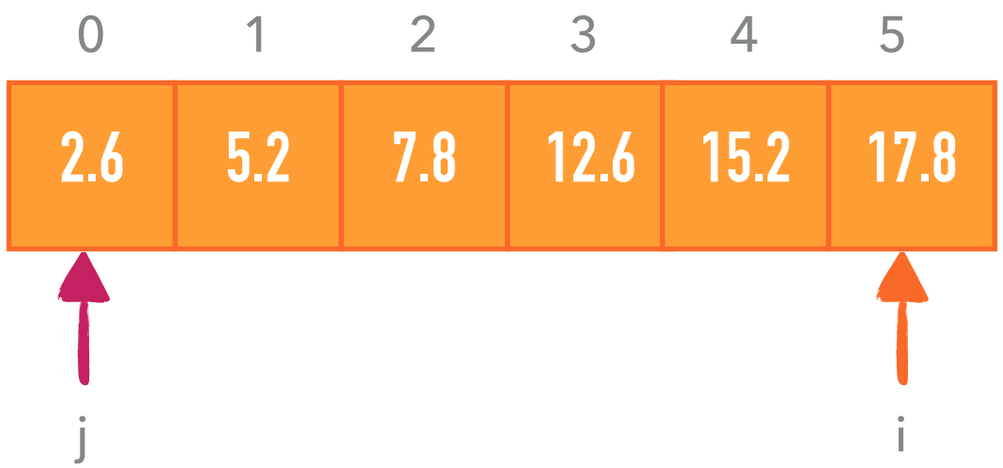


Tabella dei prezzi

Lunghezza	Prezzo
1	2.6
2	4.5
3	6.1
4	12.6
5	10.4
6	16.7

SOLUZIONE CON PROGRAMMAZIONE DINAMICA

Barra da tagliare, lunghezza $n = 6$

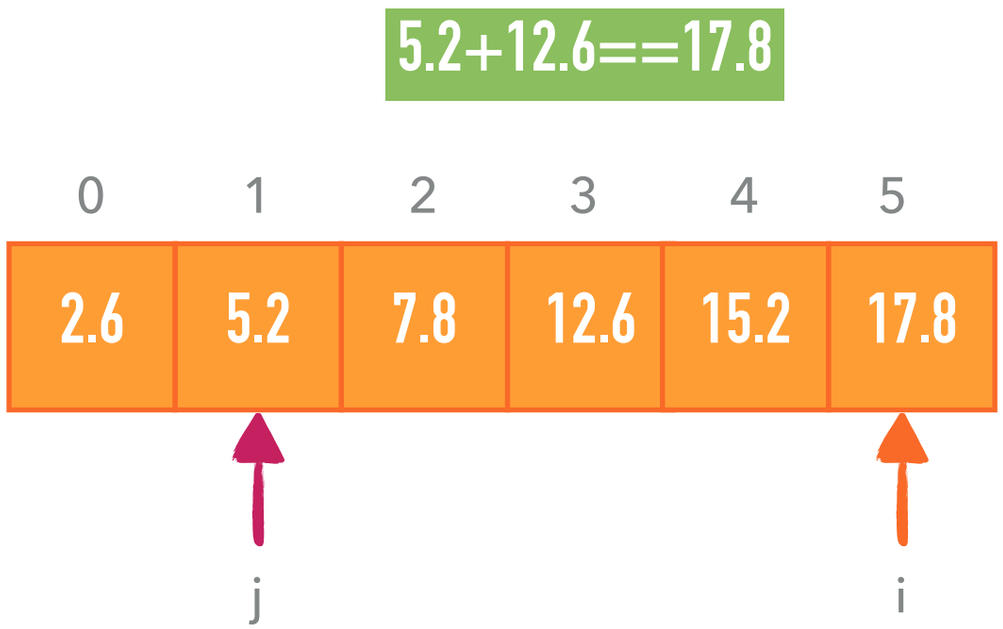


Tabella dei prezzi

Lunghezza	Prezzo
1	2.6
2	4.5
3	6.1
4	12.6
5	10.4
6	16.7

SOLUZIONE CON PROGRAMMAZIONE DINAMICA

Barra da tagliare, lunghezza $n = 6$

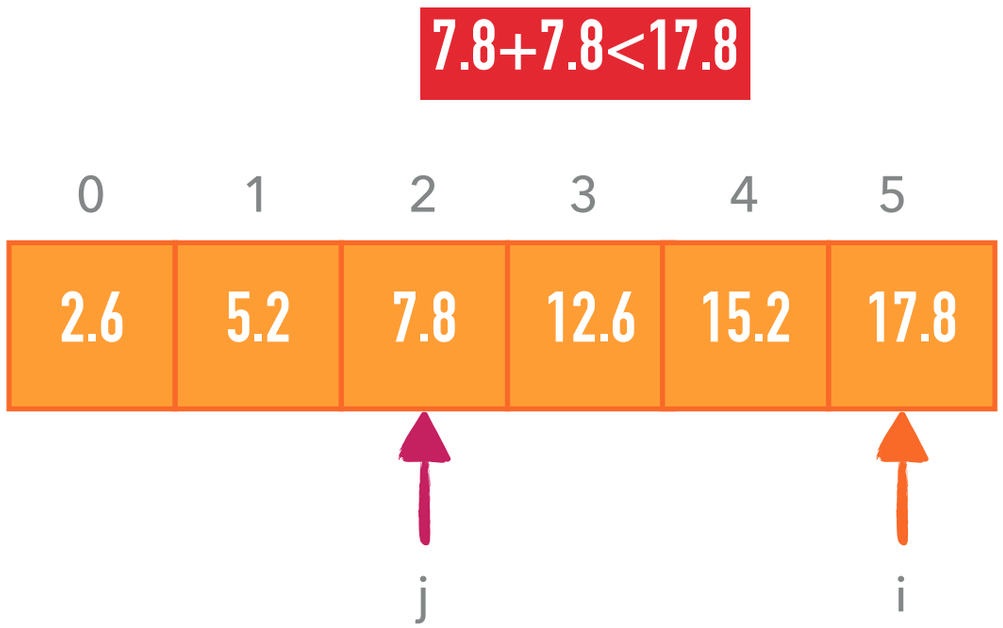


Tabella dei prezzi

Lunghezza	Prezzo
1	2.6
2	4.5
3	6.1
4	12.6
5	10.4
6	16.7

SOLUZIONE CON PROGRAMMAZIONE DINAMICA

Barra da tagliare, lunghezza $n = 6$

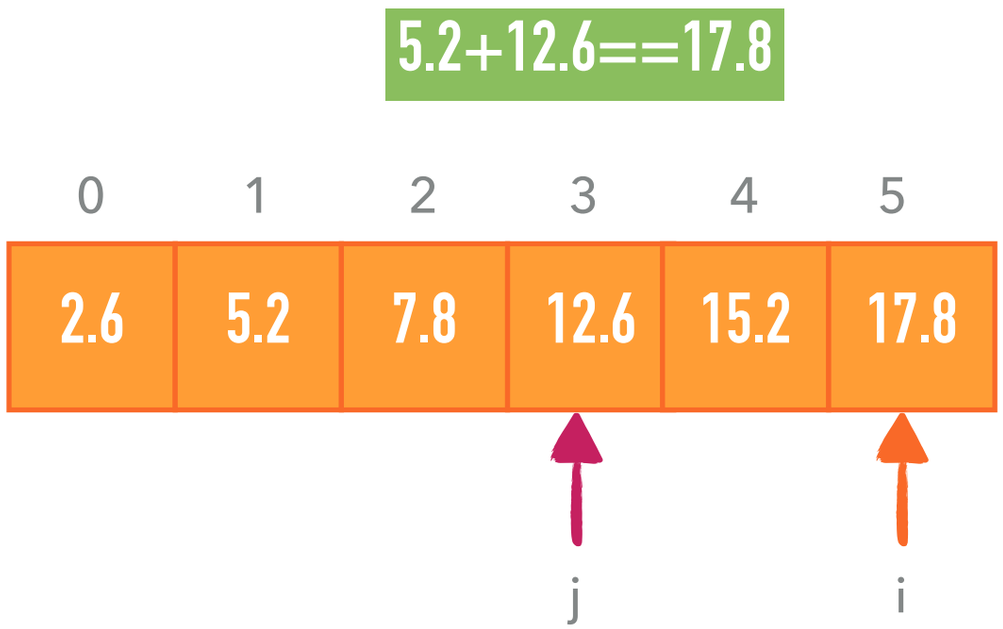


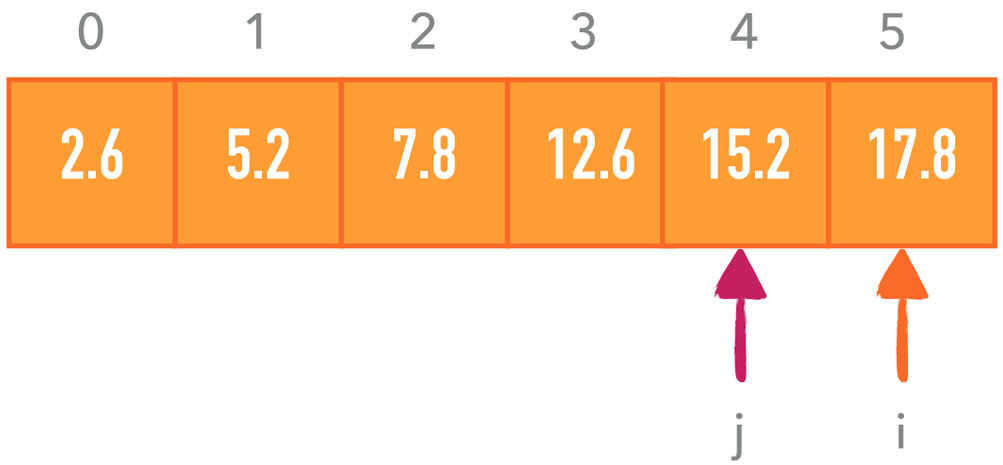
Tabella dei prezzi

Lunghezza	Prezzo
1	2.6
2	4.5
3	6.1
4	12.6
5	10.4
6	16.7

SOLUZIONE CON PROGRAMMAZIONE DINAMICA

Barra da tagliare, lunghezza $n = 6$

15.2+2.6==17.8



$12.6 + 2.6 + 2.6$

La soluzione ottima consiste quindi in questa sequenza di tagli

Tabella dei prezzi

Lunghezza	Prezzo
1	2.6
2	4.5
3	6.1
4	12.6
5	10.4
6	16.7

LA SOLUZIONE DI PROGRAMMAZIONE DINAMICA

- Riempiendo la tabella con le soluzioni dei sotto-problemi non dobbiamo mai ricalcolarli
- Complessità dell'algoritmo?
  - Due cicli for innestati, ognuno che esegue al più n iterazioni, quindi $O(n^2)$
- Siano quindi passati da un tempo esponenziale (in termini pratici intrattabile) ad un tempo quadratico

QUANDO APPLICARE LA PROGRAMMAZIONE DINAMICA

- **Sotto-struttura ottima**: la soluzione ottima ad un problema è composta da soluzioni ottime a sotto-problemi più piccoli
- **Sotto-problemi ripetuti**: trovare la soluzione ottima richiede di risolvere più volte lo stesso sotto-problema
- Sotto queste due condizioni possiamo pensare di applicare un algoritmo di programmazione dinamica

LONGEST COMMON SUBSEQUENCE

Quanto “simili” sono queste due sequenze di basi?

ACCGGTCGAGTGCGCGGAAGCCGGCCGAA

GTCGTTCGGAATGCCGTTGCTCTGTAAA

Un modo per valutare la similarità è trovare la già lunga sottosequenza di caratteri comune ad entrambe le sequenze

Cosa è una sottosequenza?

Cosa significa che è comune?

LONGEST COMMON SUBSEQUENCE

- Data una sequenza di simboli $X = \langle x_1, x_2, \ldots x_m \rangle$, una sequenza $Z = \langle z_1, z_2, \ldots, z_k \rangle$ è una sottosequenza di X se esiste una sequenza strettamente crescente di indici $\langle i_1, \ldots i_k \rangle$ tale per cui per ogni $j = 1, \ldots, k$ abbiamo che $x_{i_j} = z_j$
- Esempio: CASALE ha CASA, CAAE, ALE, SALE come sottosequenze (le otteniamo considerando – in ordine – solo alcuni dei caratteri di CASALE), ma non LESA (abbiamo tutte le lettere ma non preserviamo l'ordine)

LONGEST COMMON SUBSEQUENCE

- Data tue sequenze $X = \langle x_1, x_2, \ldots, x_m \rangle$ e $Y = \langle y_1, y_2, \ldots, y_n \rangle$ vogliamo trovare la lunghezza della più lunga sottosequenza comune ad entrambe (possono esserci più sottosequenze di lunghezza massimale)
- Esempio: se X = CASSA e Y = ASSICURAZIONE, la già lunga sottosequenza comune è ASSA, di lunghezza 4: C**ASSA** e **ASS**ICUR**A**ZIONE

LONGEST COMMON SUBSEQUENCE

- Un modo per risolvere il problema è enumerare tutte le possibili sottosequenze e trovare quelle in comune
- Però il numero di sottosequenze possibili è estremamente elevato! (quante ce ne sono?)
- Proviamo a stabilire come deve essere la struttura di una soluzione ottima

LCS: CASI BASE

- Se $X = \langle x_1, \ldots, x_m \rangle$ e $Y = \langle\rangle$ (la sequenza vuota) allora la più lunga sottosequenza comune ha lunghezza 0 ed è $Z = \langle\rangle$
- Simmetricamente se è X a essere la sequenza vuota
- Quindi sappiamo la soluzione ottima nel caso in cui una (o entrambe) le sequenze siano vuote

LONGEST COMMON SUBSEQUENCE

- Sia $Z = \langle z_1, \ldots, z_k \rangle$ una più lunga sottosequenza comune tra $X = \langle x_1, \ldots, x_m \rangle$ e $Y = \langle y_1, \ldots, y_n \rangle$
- Consideriamo gli ultimi due elementi delle sequenze X e Y
- Se $x_m = y_n$ allora $z_k = x_m = y_n$, altrimenti sarebbe possibile allungare Z concatenandoci x_m.
  In questo caso $Z' = \langle z_1, \ldots, z_{k-1} \rangle$ è la più lunga sottosequenza comune di $X' = \langle x_1, \ldots, x_{m-1} \rangle$ e $Y' = \langle y_1, \ldots, y_{n-1} \rangle$

LONGEST COMMON SUBSEQUENCE

- Se $x_m \neq y_n$ allora possiamo ignorare uno tra x_m e y_n (almeno uno dei due **non** farà parte della LCS di X e Y
- Potremmo avere che Z è la LCS di $X = \langle x_1, \ldots, x_m \rangle$ e $Y' = \langle y_1, \ldots, y_{n-1} \rangle$
- Oppure che Z è la LCS di $X' = \langle x_1, \ldots, x_{m-1} \rangle$ e $Y = \langle y_1, \ldots, y_n \rangle$

LONGEST COMMON SUBSEQUENCE

Proviamo ora a definire una soluzione.

Date $X = \langle x_1, \ldots, x_m \rangle$ e $Y = \langle y_1, \ldots, y_m \rangle$ indichiamo con $c_{i,j}$ la lunghezza dalla più lunga sottosequenza comune tra $\langle x_1, \ldots, x_i \rangle$ e $\langle y_1, \ldots y_j \rangle$

A noi interessa quindi il valore $c_{m,n}$

Ma $c_{i,j}$ è definito come:

$$c_{i,j} = \begin{cases} 0 & \text{se } i = 0 \text{ o } j = 0 \\ 1 + c_{i-1,j-1} & \text{se } x_i = y_j \\ \max\{c_{i-1,j}, c_{i,j-1}\} & \text{se } x_i \neq y_j \end{cases}$$

Possiamo quindi creare una matrice di $m + 1$ righe e $n + 1$ colonne in cui calcolare i valori di $c_{i,j}$

PSEUDOCODICE: LCS

```
Parametri: sequenza X di lunghezza m e Y di lunghezza n
c = matrice (m+1)×(n+1)
for i in range(0, m+1): # casi base: sequenza vuota
    c[i][0] = 0
for j in range(0, n+1):
    c[0][j] = 0
for i in range(1, m+1):
    for j in range(1, n+1):
        if X[i-1] == Y[j-1]: # Se il carattere in posizione i,j coincide
            c[i][j] = 1 + c[i-1][j-1]
        else: # altrimenti prendiamo il migliore dei due sottoproblemi
            c[i][j] = max(c[i-1][j], c[i][j-1])
return c[m][n]
```

LONGEST COMMON SUBSEQUENCE

			A	B	B	A
		0	1	2	3	4
	0	0	0	0	0	0
C	1	0	0	0	0	0
A	2	0	1	1	1	1
S	3	0	1	1	1	1
A	4	0	1	1	1	2