

# Algoritmi e Strutture Dati

Foglio 2  
28/02/2025

**Esercizio 1.** Modificare la subroutine MERGE di MERGE-SORT in modo da non utilizzare le sentinelle.

**Esercizio 2.** Scrivere uno pseudocodice per l'algoritmo di ricerca binaria visto a lezione e dimostrarne la correttezza.

**Esercizio 3.** Scrivere uno pseudocodice per l'algoritmo di elevamento a potenza visto a lezione e dimostrarne la correttezza.

**Esercizio 4.** Sia  $T(n)$  il numero di righe stampate dal seguente algoritmo ricorsivo. Dare un limite asintotico stretto per  $T(n)$ .

```
function f(n)
if n > 1:
    print("still going")
    f(n/2)
    f(n/2)
```

**Esercizio 5.** Usando il metodo di sostituzione, dimostrare che la soluzione di  $T(n) = T(n - 1) + n$  è  $O(n^2)$  (si assuma che  $T(1) = c$ ).

**Esercizio 6.** Utilizzare un albero di ricorsione per determinare un buon limite asintotico superiore per  $T(n)$  e applicare il metodo di sostituzione per verificarlo (si assuma in tutti i casi che  $T(1) = c$ ):

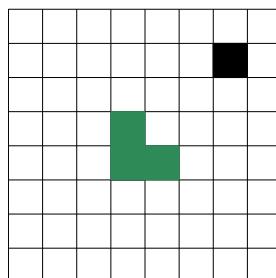
1.  $T(n) = 2T(n - 1) + 1;$
2.  $T(n) = 2T(n/2) + n \log n.$

**Esercizio 7.** Consideriamo il problema di calcolare la somma degli elementi di un array di  $n$  numeri. Usando il Divide et Impera, fornire un algoritmo che risolva il problema e stimarne la complessità. È migliore dell'algoritmo brute force che scorre l'array?

**Esercizio 8.** Ci è dato un vettore di  $n$  numeri e notiamo che alcuni di essi si ripetono. Come possiamo rimuovere tutti i doppioni in  $O(n \log n)$  passi?

**Esercizio 9.** Sia  $A$  un vettore di  $n$  interi distinti e ordinati. Vogliamo determinare se esiste un indice  $i$  tale che  $A[i] = i$ . Fornire un algoritmo con tempo di esecuzione  $O(\log n)$ .

**Esercizio 10.** Si vuole tassellare una scacchiera  $n \times n$  (con  $n$  potenza di 2), priva di una casella (per esempio, la casella nera in figura), con dei tasselli a forma di L occupanti tre caselle (tassello verde in figura e sue rotazioni). Fornire un algoritmo con tempo di esecuzione  $O(n^2)$ .



Esercizio 1. Modificare la subroutine MERGE di MERGE-SORT in modo da non utilizzare le sentinelle.

MERGE-SORT ( $A, p, r$ )

if  $p < r$

$$q \leftarrow \left\lfloor \frac{p+r}{2} \right\rfloor$$

MERGE-SORT ( $A, p, q$ )  
MERGE-SORT ( $A, q+1, r$ )  
MERGE ( $A, p, q, r$ )

MERGE ( $A, p, q, r$ )

$$n_1 \leftarrow q-p+1$$
$$n_2 \leftarrow r-q$$

// creo i due array di supporto  $L[1..n_1]$ ,  $R[1..n_2]$

// copio i vecchi valori a partire da posizione 'p' fino a 'r'

for  $i \leftarrow 1$  to  $n_1$   
 $L[i] \leftarrow A[p+i-1]$

for  $i \leftarrow 1$  to  $n_2$   
 $R[i] \leftarrow A[q+i]$

// ruolo sentinelle? Now uscire dai bound ...

$i \leftarrow 1$   
 $j \leftarrow 1$

for  $k \leftarrow p$  to  $r$

if  $i \leq n_1$   $\&$   $j \leq n_2$  // solito ...

if  $L[i] \leq R[j]$   
 $A[k] \leftarrow L[i]$   
 $i \leftarrow i+1$   
else  
 $A[k] \leftarrow R[j]$   
 $j \leftarrow j+1$

else

if  $i > n_1$   
 $A[k] \leftarrow R[j]$   
 $j \leftarrow j+1$   
else  
 $A[k] \leftarrow L[i]$   
 $i \leftarrow i+1$

!

PUO' MAI ACCADERE CHE ENTRI NEL PRIMO ELSE CON  $i > n_1$   $\&$   $j > n_2$ ?

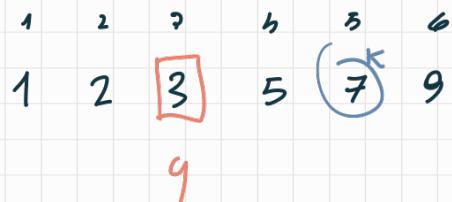
No,  $k \in [p..r]$  e ad ogni ciclo solo uno tra  $i, j$  viene aumentato di 1.  
Quante volte entra?  $r-p+1$

$$\Rightarrow i+j \leq r-p+1 =: n_1+n_2$$

Esercizio 2. Scrivere uno pseudocodice per l'algoritmo di ricerca binaria visto a lezione e dimostrarne la correttezza.

BINARY-SEARCH( $A, l, r, K$ ) : // A ordinato

```
if  $r < l$   
return -1  
 $q \leftarrow \left\lfloor \frac{r+l}{2} \right\rfloor$ 
```



```
if  $A[q] = K$   
return q  
else if  $A[q] < K$   
return BINARY-SEARCH( $A, q+1, r, K$ )  
else  
return BINARY-SEARCH( $A, l, q-1, K$ )
```

CORRETTO ? E' ricorsivo, sfrutto l'induzione per dimostrare la correttezza

- o Hp : BINARY-SEARCH( $A, l, r, K$ ) restituisce ' $-1$ ' se il sottoarray da controllare è vuoto ( $r < l$ ); altrimenti se presente, restituisce la posizione della chiave  $K$  nel sotto-array  $A[l..r]$ ,  $l \leq K \leq r$ . Inoltre, poiché  $A$  è ORDINATO, le posizioni  $[1..l-1] \cup [r+1..n]$  non contengono la chiave.

$$\boxed{P(n) \Rightarrow P(n+1)}$$

$$\Leftrightarrow P(n) \text{ vera } \forall n \in \mathbb{N}$$

- o CASO BASE 1: sottoarray  $A[l..r]$  vuoto

L'if iniziale permette di tornare  $-1$ .

- o CASO BASE 2: sottoarray  $A[l..r]$  di 1 elemento  $\Leftrightarrow l=r$

In questo caso  $q = \left\lfloor \frac{l+l}{2} \right\rfloor \equiv l$ .

Se  $A[l] = K$ , restituisce  $q$ ; altrimenti, in entrambi i casi rimanenti ( $A[l] < K \leq A[r] > K$ ) la chiamata ricorsiva restituisce ' $-1$ ' in quanto invocata su sottoarray vuoti.

- o PASSO : Suppongo vera Hp per dimensioni di  $A$  inferiori a  $n$ .

La procedura verifica se la chiave è in pos.  $q$ , altrimenti delega la ricerca a una delle due chiamate ricorsive; poiché entrambe gestiscono un input di dimensione inferiore a  $n$ , per ipotesi induttiva restituiscono correttamente il risultato.

$$\hookrightarrow \Delta x : r - (q+1) + 1 = r - \left\lfloor \frac{r+l}{2} \right\rfloor \leq r - \frac{r+l}{2} = \frac{r-l}{2} < r - l + 1 = n$$

$$\text{sx: } (q-1) - l + 1 = \left\lfloor \frac{r+l}{2} \right\rfloor - l \leq \frac{r+l}{2} - l = \frac{r-l}{2} < \frac{r+l}{2} < n$$

**Esercizio 3.** Scrivere uno pseudocodice per l'algoritmo di elevamento a potenza visto a lezione e dimostrarne la correttezza.

$$a^n = \begin{cases} a^{\frac{n}{2}} \cdot a^{\frac{n}{2}} & \text{se } n \text{ pari} \\ a^{\frac{(n-1)}{2}} \cdot a^{\frac{(n-1)}{2}} \cdot a & \text{se } n \text{ dispari} \end{cases}$$

POWERING( $a, n$ ):

```

if n = 0
    return 1
if PARI(n)
    K = POWERING(a, n/2)
    return K · K
else
    K = POWERING(a, (n-1)/2)
    return a · K · K

```

• CORRETTEZZA : induzione su  $n$

◦ BASE :  $n = 0 \rightarrow \text{POWERING}(a, 0) : 1 \quad \checkmark$

◦ PASSO : suppongo che  $\text{POWERING}(a, n)$  sia corretto, ossia calcoli  $a^n$ .

Se  $n+1$  pari, allora  $\frac{n+1}{2}$  è pari ; in più  $\frac{n+1}{2} < n \quad \forall n \in \mathbb{N}$  quindi per Hp induttiva

$$\text{POWERING}(a, n+1) : a^{\frac{n+1}{2}} \cdot a^{\frac{n+1}{2}} = a^{\frac{2 \cdot \frac{n+1}{2}}{2}} = a^{n+1} \quad \checkmark$$

Se  $n+1$  dispari, allora  $\frac{n+1-1}{2} = \frac{n}{2}$  è pari ; inoltre  $\frac{n}{2} < n \quad \forall n \in \mathbb{N}$

$$\text{POWERING}(a, n+1) : a \cdot a^{\frac{n}{2}} \cdot a^{\frac{n}{2}} = a \cdot a^{\frac{2 \cdot \frac{n}{2}}{2}} = a^{n+1} \quad \checkmark$$

$\Rightarrow \text{POWERING}(a, n)$  corretto  $\forall n \in \mathbb{N}$ .

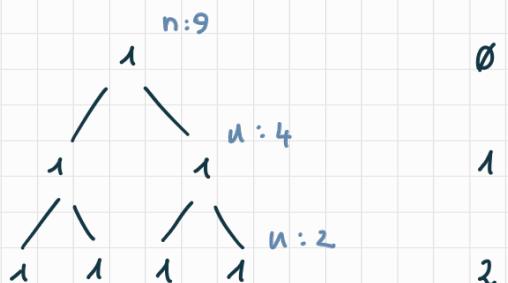
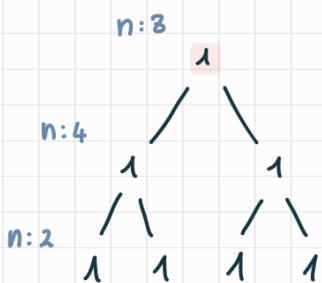
Esercizio 4. Sia  $T(n)$  il numero di righe stampate dal seguente algoritmo ricorsivo. Dare un limite asintotico stretto per  $T(n)$ .

```
function f(n)
if n > 1:
    print("still going")
    f(n/2)
    f(n/2)
```

- Scriviamo EQ. RICORRENZA

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

- Albero di ricorsione. N.B.:  $n$  potrebbe essere pari o dispari, questo non cambia il risultato finale  $\rightarrow$  divisione intera per 2



- Quanto è alto l'albero? Ad ogni chiamata divido l'input per 2. Indico con  $n_i$  la dimensione dell'input al livello  $i$ -esimo:

$$n_i = \frac{n}{2^i}$$

Le foglie hanno dimensione 2

$$[2 = n_h] \quad 2 = \frac{n}{2^i} \Leftrightarrow n = 2 \cdot 2^i \Leftrightarrow n = 2^{i+1} \stackrel{i \in \mathbb{N}}{\Leftrightarrow} i = \lfloor \log n \rfloor - 1$$

$$\text{Quindi } h = \lfloor \log n \rfloor$$

- Quanti nodi al livello  $i$ -esimo?  $\rightarrow N_i = 2^i$  (nodi al livello  $i$ )

$$\Rightarrow \# \text{ nodi totali } N = \sum_{i=0}^{\lfloor \log n \rfloor - 1} 2^i \stackrel{(1)}{=} 2^{\lfloor \log n \rfloor} - 1$$

- Costo totale:  $T(n) = 1 \cdot (2^{\lfloor \log n \rfloor} - 1) = 2^{\lfloor \log n \rfloor}$

$$(1) \sum_{k=0}^n 2^k = 2^{n+1} - 1 \quad \forall n \in \mathbb{N}$$

o Verifico per sostituzione che  $T(n) = O(n)$ .

o BASE  $n=1$ :  $T(1) = \emptyset \quad \checkmark$

o PASSO:  $T(n) \leq cn \quad \forall n \geq n_0$

$$T(n+1) := 1 + 2 \cdot T\left(\frac{n+1}{2}\right)$$

$$\leq 1 + 2cn$$

$\left(\frac{n+1}{2} < n \quad \forall n \in \mathbb{N}, \text{H.p. ind.}\right)$

Osserviamo che

$$1 + 2cn \leq cn$$

$$\Leftrightarrow 1 + cn \leq 0$$

! MAI

$$\boxed{P(n) \Rightarrow P(n+1)}$$

$\equiv P(n) \text{ vera } \forall n \in \mathbb{N}$

o Provo con una ipotesi induuttiva più forte:

$$T(n) \leq cn - \underbrace{c'}_{>0} \quad \forall n \geq n_0 \quad [c, c', n_0 \text{ fissati} > 0]$$

sottrago termine di ordine inferiore

o BASE:  $T(1) = \emptyset \leq c - c' \Leftrightarrow c' \leq c \quad (1)$

o PASSO: Sia  $n \geq 1$  tale per cui  $T(n) \leq cn - c' \quad \forall n \geq n_0$

$$\begin{aligned} T(n+1) &:= 2T\left(\frac{n+1}{2}\right) + 1 \\ &\leq 2(c - \frac{n+1}{2}c') + 1 \quad (\text{H.p. indukt.}) \\ &= c(n+1) - 2c' + 1 \\ &= c(n+1) - c' + (1 - c') \end{aligned}$$

$$T(n+1) \leq c(n+1) - c'$$

Oss. che  $\leq c(n+1) - c'$

$$(1 - c') \leq 0 \quad \forall n \in \mathbb{N} \Leftrightarrow c' \geq 1 \quad (2)$$

Quindi

$$\begin{aligned} T(n+1) &\leq c(n+1) - c' + (1 - c') \leq c(n+1) - c' \quad \forall n \in \mathbb{N} \\ &\Leftrightarrow c' \geq 1 \end{aligned}$$

→ In definitiva, basta soddisfare  $(1) \leq (2)$ , ad es.  $c' = c = 1$ ,

$$\rightarrow n_0 = c' = c = 1$$

o  $T(n) = \Omega(n)$  analogo ...

□

**Esercizio 5.** Usando il metodo di sostituzione, dimostrare che la soluzione di  $T(n) = T(n-1) + n$  è  $O(n^2)$  (si assuma che  $T(1) = c$ ).

- Albero di ricorsione



- Altezza

- ad ogni livello dimensione input decresce di 1:  $n_i = n-i$
- ultimo livello dimensione 1:  $n_h = 1 \Leftrightarrow n-h = 1 \Leftrightarrow h = n-1$   
 $\Rightarrow h = n-1$
- Costo per riga: dipende dalla dimensione dell'input
- Costo tot.

$$T(n) = \sum_{i=0}^{n-2} i + c = \frac{(n-2)(n-1)}{2} + c = \Theta(n^2).$$

o SOSTITUZIONE :  $T(n) \leq K n^2$ ,  $\forall n \in \mathbb{N}$  [fisso no, K nella dim.]

o BASE :  $T(1) := c \leq K \stackrel{K \cdot (1)^2}{\Leftrightarrow} K \geq c$  (1)

o PASSO :  $T(n) \leq K n^2$   $\forall n \in \mathbb{N}$

$$T(n+1) := T(n) + (n+1) \stackrel{\text{HP}}{\leq} K n^2 + n + 1 \stackrel{K(n+1)^2}{\leq} K(n+1)^2$$

Oss che

$$K n^2 + n + 1 \leq K(n+1)^2 = K n^2 + 2Kn + K$$

$$\Leftrightarrow n + 1 \leq 2Kn + K = K(2n+1)$$

$$\Leftrightarrow \frac{n+1}{2n+1} \leq K$$

$\leq 1 \quad \forall n \in \mathbb{N}$

$$\Leftrightarrow K \geq 1 \quad (2)$$

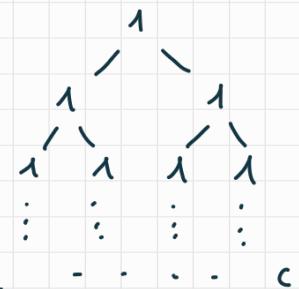
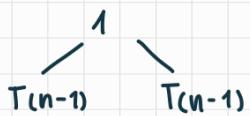
Quindi  $\forall n \in \mathbb{N} \leq K \geq 1 \quad T(n+1) \leq K(n+1)^2$ .

→ Soddisfo (1), (2) con  $n_0 = 1 \leq K = c + 1$ .

**Esercizio 6.** Utilizzare un albero di ricorsione per determinare un buon limite asintotico superiore per  $T(n)$  e applicare il metodo di sostituzione per verificarlo (si assuma in tutti i casi che  $T(1) = c$ ):

1.  $T(n) = 2T(n - 1) + 1;$
  2.  $T(n) = 2T(n/2) + n \log n.$

## o Alberi



livello

- Ø
- 1
- 2
- :
- n - 1

//  $n_i$ : dimensione input livello  $i$

$$n_i = n - i \quad \Rightarrow \quad n_h = 1 \quad \Leftrightarrow \quad h = n - 1$$

// Ni : numero nodi livello i

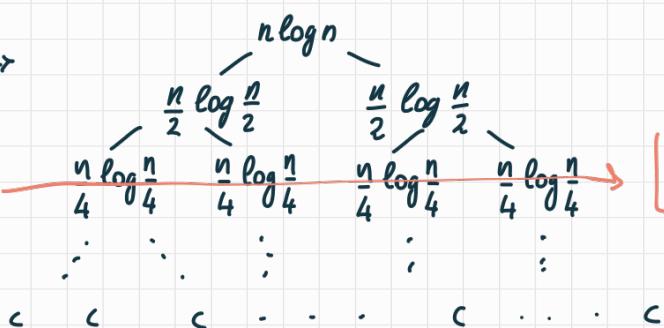
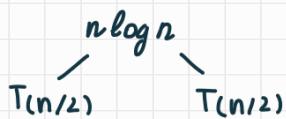
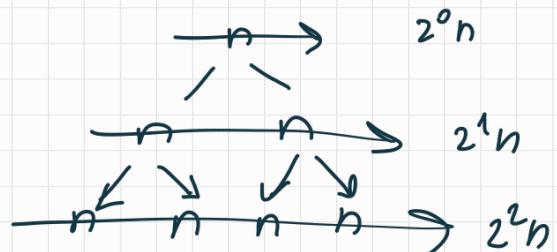
$$N_i = 2^i$$

// Ci : costo livello i

$$c_i = 2^i$$

$$\Rightarrow T(n) = \sum_{i=0}^{n-2} c_i + c \cdot 2^h = \sum_{i=0}^{n-2} 2^i + c \cdot 2^{n-1} = (2^{n-1} - 1) + c \cdot 2^{n-1}$$

$$= (1+c)2^{n-1} - 1 = \Theta(2^n)$$



$$n_i = \frac{n}{2^i} \Rightarrow n_h = 1 \Leftrightarrow \frac{n}{2^h} = 1 \Leftrightarrow h = \log n$$

$$\frac{h}{7} \rightarrow h = \Theta(\log_+ n)$$

$$N_k = 2^i$$

$$c_i = \sum_{i=1}^{N_i} n_i \log n_i = N_i \cdot n_i \log n_i = 2^i \cdot \frac{n}{2^i} \log \frac{n}{2^i} = n \log \frac{n}{2^i} = \boxed{n \log n - i}$$

$$\begin{aligned}
 \Rightarrow T(n) &= \sum_{i=0}^{\log n - 1} c_i = \sum_{i=0}^{\kappa} (u \log n - i) \\
 &= n \log n \cdot \sum_{i=0}^{\kappa} 1 - \sum_{i=0}^{\kappa} i \\
 &= n \log n \cdot \kappa - \frac{\kappa(\kappa+1)}{2} \\
 &= n \cdot \log n \cdot (\log n - 1) - \frac{\log n \cdot (\log n - 1)}{2} \\
 &= \Theta(n \cdot \log^2 n)
 \end{aligned}$$

1

- o Dimostro  $T(n) = O(2^n) \Leftrightarrow \exists m, n_0 > 0 : 0 \leq \boxed{T(n) \leq m 2^n} \forall n \geq n_0$
- o BASE:  $T(1) = c \leq 2m \Leftrightarrow m \geq c/2 \quad (1)$
- o PASSO: Sia  $n, n_0$  t.c.  $T(n) \leq m 2^n \forall n \geq n_0$   
 $T(n+1) := 2T(n) + 1 \stackrel{H_p}{\leq} 2 \cdot (m 2^n) + 1 \longrightarrow m \cdot 2 \cdot 2^n + 1 = m 2^{n+1} \leq m 2^{n+1}$

Oss che

$$2m 2^n + 1 \leq m 2^{n+1} = 2m \cdot 2^n$$

$$\Leftrightarrow 1 \leq 0 \quad \text{ff}$$

- o Rafforz:  $T(n) \leq m 2^n - d \quad \forall n \geq n_0$
- o BASE:  $T(1) := c \leq 2m - d \Leftrightarrow m \geq \frac{c+d}{2} \quad (1)$
- o PASSO:  
 $T(n+1) := 2T(n) + 1 \stackrel{H_p}{\leq} 2 \cdot (m 2^n - d) + 1 = m 2^{n+1} - 2d + 1$

Ora

$$m 2^{n+1} - 2d + 1 \leq m 2^{n+1} \Leftrightarrow -2d + 1 \leq 0 \Leftrightarrow d \geq \frac{1}{2} \quad (2)$$

$$\rightsquigarrow \text{Scelgo } d = \frac{1}{2} \quad [(2)] \Rightarrow m = \frac{2c+1}{4}$$

$$\rightsquigarrow n_0 = 1, d = \frac{1}{2}, m = \frac{2c+1}{4}$$

$$(2) T(n) = 2T(n/2) + n \log n \quad , n > 1$$

o Dimostro  $T(n) = O(n \log^2(n)) \Leftrightarrow \exists m, n_0 > 0 : 0 \leq T(n) \leq m n \log^2(n) \quad \forall n \geq n_0$

o BASE:  $T(1) := c \leq m \Leftrightarrow m \geq c \quad (1)$

o PASSO: Sia  $n \geq n_0$  t.c.  $T(n/2) \leq m \frac{n}{2} \log^2\left(\frac{n}{2}\right) \quad \forall n \geq n_0$

$$\begin{aligned} T(n) &:= 2T(n/2) + n \log n \stackrel{H_p}{\leq} 2m \cdot \frac{n}{2} \log^2\frac{n}{2} + n \log n \\ &= mn \log^2\frac{n}{2} + n \log n \leq m \cdot n \cdot \log^2(n) \end{aligned}$$

Oss. che  $\rightarrow (\log n - 1)^2$

$$mn \log^2\frac{n}{2} + n \log n \leq mn \log^2 n$$

$$\Leftrightarrow mn(\log n - 1)^2 + n \log n = mn[\log^2 n - 2\log n + 1] + n \log n \leq mn \log^2 n$$

$$\Leftrightarrow -2mn \log n + mn + n \log n \leq 0$$

$$\Leftrightarrow mn(-2\log n + 1) + n \log n \leq 0$$

$$\Leftrightarrow m \cancel{\sqrt{}}(2\log n - 1) - \cancel{\sqrt{}} \log n \leq 0$$

$$\Leftrightarrow m \leq \frac{\log n}{2\log n - 1} \stackrel{\text{arco}}{\longrightarrow} \frac{1}{2 - \frac{1}{\log n}} \stackrel{\text{arco}}{\longrightarrow} \frac{1}{2} \quad (2)$$

$\rightarrow$  Scelgo  $m = \max\{c, 1\} \rightarrow (1) \text{ chiedeva } m \geq c$   
 $\quad \quad \quad (2) \text{ chiedeva } m \geq 1$

$$m = 1 + c \geq c$$

$$\geq 1$$

**Esercizio 7.** Consideriamo il problema di calcolare la somma degli elementi di un array di  $n$  numeri. Usando il Divide et Impera, fornire un algoritmo che risolva il problema e stimarne la complessità. È migliore dell'algoritmo brute force che scorre l'array?

1	2	3	4	5	6	7
1	4	7	2	1	4	9

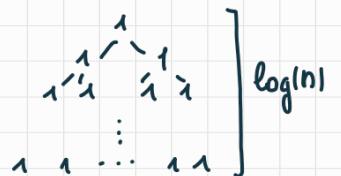
- con algoritmo lineare :  $T(n) = \Theta(n)$
- soluzione 1 : divido in maniera bilanciata

1	2	3	4	5	6	7
1	4	7	2	1	4	9

$$S(1, 3) + 2 + S(5, 7)$$

$$\begin{aligned} \rightsquigarrow T(n) &= 2T(n/2) + 1 \\ &= 2n - 1 = \Theta(n) \end{aligned}$$

$$\sum_{i=0}^{\log n} 2^i = \frac{2^{\log n + 1} - 1}{2 - 1}$$



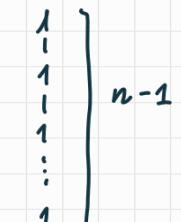
- soluzione 2 : divido in maniera sbilanciata e 1 branch

1	2	3	4	5	6	7
1	4	7	2	1	4	9

$$1 + S(2, n)$$

$$\begin{aligned} \rightsquigarrow T(n) &= T(n-1) + 1 \\ &= n - 1 + 1 = n = \Theta(n) \end{aligned}$$

$$\sum_{i=0}^{n-1} 1 = n - 1$$



$$\begin{aligned} \text{SUM}(A, i, r) &\rightarrow \text{SUM}(A, 1, 7) \rightarrow 1 + \text{SUM}(A, 2, 7) \\ &\rightarrow \dots \rightarrow \text{SUM}(A, 7, 7) = 9 \end{aligned}$$

**Esercizio 8.** Ci è dato un vettore di  $n$  numeri e notiamo che alcuni di essi si ripetono. Come possiamo rimuovere tutti i doppioni in  $O(n \log n)$  passi?

1 4 5 1 2 5 1 7 9

- abbiamo un algo che ordina in  $\Theta(n \log n)$  ...

1 1 1 2 4 5 5 7 9

- scorriamo l'array e copiamo un elemento nell'array risultato ogni volta che è diverso da quello nella posizione precedente

→ ✓ ✓ ✓ ✓ ✓ ✓  
1 1 1 2 4 5 5 7 9

UNIQUE-ELEM(A) : // A[1..n]

// creo B[1..n]

MERGE-SORT(A) //  $\Theta(n \log n)$

$j \leftarrow 1$   
 $B[j] \leftarrow A[j]$  // copio primo elemento  
 $j \leftarrow j+1$

for  $i \leftarrow 2$  to  $A.length$

if  $A[i-1] \neq A[i]$

$B[j] \leftarrow A[i]$   
 $j \leftarrow j+1$

$B.length = j-1$   
return B

$$f(x) = x$$

**Esercizio 9.** Sia  $A$  un vettore di  $n$  interi distinti e ordinati. Vogliamo determinare se esiste un indice  $i$  tale che  $A[i] = i$ . Fornire un algoritmo con tempo di esecuzione  $O(\log n)$ .

- Input :  $A$  ordinato, elem. distinti

1	2	3	4	5	6	7
1	2	4	5	6	8	10

- Idea : Divide et Impera

$$A[i] = i \Leftrightarrow A[i] - i = 0$$

1	2	3	4	5	6	7
1	2	4	5	6	8	10

$\text{exg}(A, l, r)$

~~∅ ∅ 1 1 1 2 3~~

if  $l > r$  return -1

$$q \leftarrow \left\lfloor \frac{l+r}{2} \right\rfloor$$

if  $A[q] = q$   
return  $q$

else if  $A[q] > q$   
return  $\text{exg}(A, l, q-1)$   
else  
return  $\text{exg}(A, q+1, r)$

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

$$= \Theta(\log n)$$

$$\begin{bmatrix} 1 \\ 1 \\ 1 \\ \vdots \\ 1 \\ 1 \end{bmatrix} \quad h = \Theta(\log n)$$

$$T(n) = \sum_{i=0}^{\lfloor \log n \rfloor} 1$$

$$= \lfloor \log n \rfloor .$$

**Esercizio 10.** Si vuole tassellare una scacchiera  $n \times n$  (con  $n$  potenza di 2), priva di una casella (per esempio, la casella nera in figura), con dei tasselli a forma di L occupanti tre caselle (tassello verde in figura e sue rotazioni). Fornire un algoritmo con tempo di esecuzione  $O(n^2)$ .

