MergeSort(a, 0, 3)

20	10	11	6
0	1	2	3

q = (0+3)/2   // q = 1

MergeSort(a, 0, 1)

20	10	11	6
0	1	2	3

q = (0+1)/2   // q = 0

MergeSort(a, 0, 0)

20	10	11	6
0	1	2	3

MergeSort(a, 1, 1)

20	10	11	6
0	1	2	3

Merge(a, 0, 0, 1)

10	20	11	6
0	1	2	3

MergeSort(a, 2, 3)

10	20	11	6
0	1	2	3

q = (2+3)/2   // q = 2

MergeSort(a, 2, 2)

10	20	11	6
0	1	2	3

MergeSort(a, 3, 3)

10	20	11	6
0	1	2	3

Merge(a, 2, 2, 3)

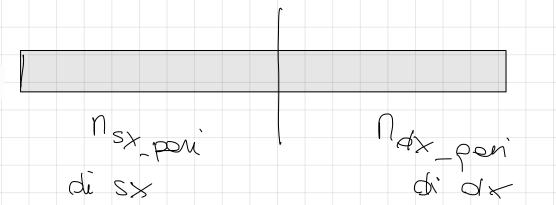
10	20	6	11
0	1	2	3

Merge(a, 0, 1, 3)

6	10	11	20
0	1	2	3

ESERCIZIO 2

Progettare un algoritmo di tipo *divide-et-impera* per contare il numero di elementi pari in un array di interi, e analizzarne la complessità.



return $n_{sx\_pari} + n_{dx\_pari}$

```
ContaPari(a, sx, dx)   //a: array di interi, sx e dx interi
    if (sx > dx) return 0;
    if (sx == dx) {
            if (a[sx] % 2 == 0) return 1;
            else return 0;
    }                                          O(1)
    // ci sono almeno 2 elementi
    cx = (sx + dx)/2          //divisione intera } O(1) Divisione
    return ContaPari(a, sx, cx) + ContaPari(a, cx+1, dx);
```

ricorsione $T(n/2)$ | combinazione $O(1)$ | ricorsione $T(n/2)$

Chiamata iniziale

ContaPari(a, 0, a.length - 1)

$$T(n) = \begin{cases} O(1) & n \leq 1 \\ 2T\left(\frac{n}{2}\right) + O(1) & n > 1 \end{cases}$$

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

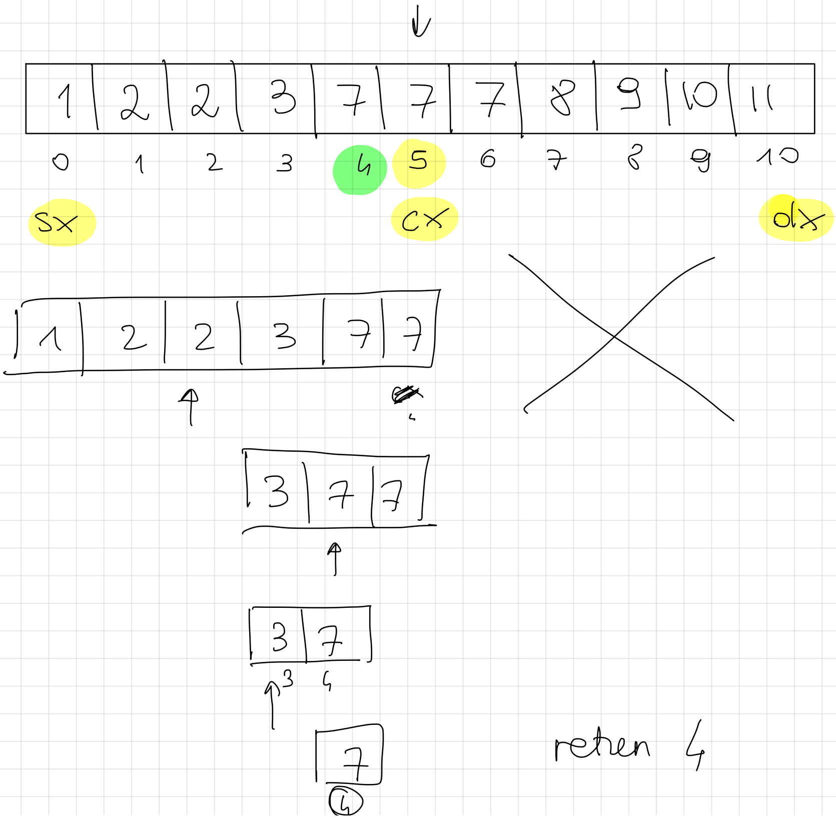
Limite inferiore

con dimensione dell'input: $L(n) = \Omega(n)$

L'algoritmo è ottimo.

Esercizio 3

Progettare un algoritmo di ricerca binaria che, dato un array ordinato *a* di *n* interi, alcuni dei quali possono essere ripetuti e una chiave *k*, restituisca la posizione della prima occorrenza di *k* in *a*, se presente, e -1 altrimenti. Analizzare la complessità dell'algoritmo.



$k = 7$

posizione della prima occorrenza di 7: 4

return 4

RicercaBinariaSx (a, sx, dx, k)

Prima chiamata
RicercaBinariaSx(a, 0, n-1, k)
n = a.length

CASO BASE
chiusura ricorsione

```
if (sx > dx) return -1;
if (sx == dx) {
    if (a[sx] == k) return sx;
    else return -1;
}
// ci sono almeno due elementi
```

DIVISIONE

```
cx = (sx + dx)/2        // divisione intera
```

Ricorsione e combinazione

```
if (k <= a[cx]) return RicercaBinariaSx(a, sx, cx, k);
else return RicercaBinariaSx(a, cx+1, dx, k);
```

$$T(n) = \begin{cases} O(1) & n \leq 1 \\ T(n/2) + O(1) & n > 1 \end{cases}$$

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

Limite inferiore (AdD)

$$L(n) = \Omega(\log S(n))$$

$S(n)$ = numero di soluzioni del problema su istanze di dimensione n

$$S(n) = n + 1$$

k presente, prima occorrenza in $a[i]$ $\quad 0 \leq i \leq n-1$

k non presente

$$L(n) = \Omega(\log n)$$

l'algoritmo è ottimo!

Ricorrenze: Metodi di risoluzione

① Metodo di sostituzione

ipotesi di soluzione, da dimostrare per induzione

② Metodo dell'albero di ricorsione (iterativo)

nodi interni → costo delle chiamate ricorsive

foglie → costo della risoluzione diretta dei sottoproblemi elementari

→ si sommano i costi su tutti i livelli dell'albero per ottenere una stima della soluzione in forma chiusa della ricorrenza

③ Metodo principale

costo per dividere e combinare

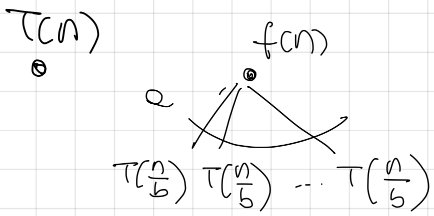
$$T(n) = \begin{cases} O(1) & n \le n_0 \\ \underbrace{aT\left(\frac{n}{b}\right)}_{\text{costo ricorsione}} + f(n) & n > n_0 \end{cases}$$

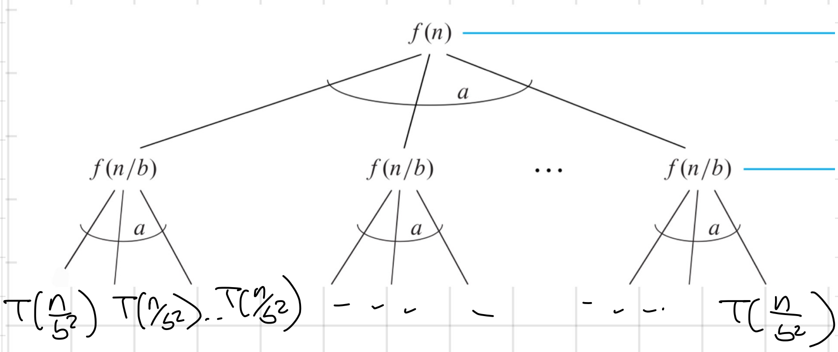
a intero

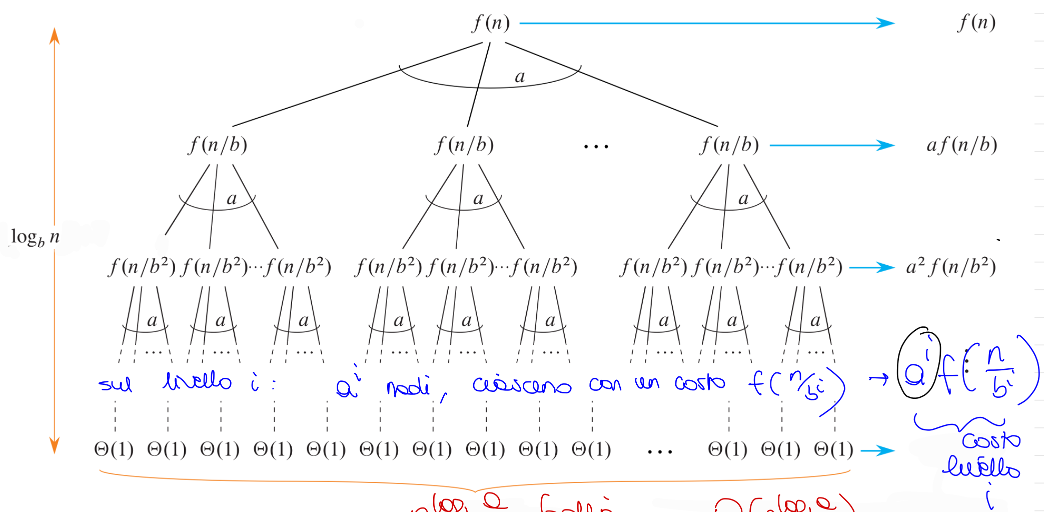
$a > 0$ $\quad$ $b > 1$ (deve valere $\frac{n}{b} < n$)

$$T(n) = \begin{cases} O(1) & n \le 1 \\ aT\left(\frac{n}{b}\right) + f(n) & n > 1 \end{cases}$$

n potenza esatta di b







$n^{\log_b a}$ foglie $\rightarrow \Theta(n^{\log_b a})$

$$\sum_{i=0}^{\log_b n-1} a^i f\left(\frac{n}{b^i}\right) + \Theta(n^{\log_b a})$$

$$\frac{n}{b^i} = 1 \qquad n = b^i \quad \Rightarrow \quad i = \log_b n$$

$$\# \text{ foglie} = a^{\log_b n}$$

$$a^{\log_b n} = n^{\log_b a}$$

$$a^{\log_b n} = \left(n^{\log_n a}\right)^{\log_b n} = \left(n^{\frac{\log_b a}{\cancel{\log_b n}}}\right)^{\cancel{\log_b n}} = \boxed{n^{\log_b a}}$$

$$T(n) = \underbrace{\sum_{i=0}^{\log_b n-1} a^i f\left(\frac{n}{b^i}\right)}_{\text{costo dei passi divide et impera}} + \underbrace{\Theta\left(n^{\log_b a}\right)}_{\text{costo della risoluzione diretta di tutti i sottoproblemi elementari}}$$

(x dividere e combinare le soluzioni)

Teorema principale

Siano $a > 0$ e $b > 1$ delle costanti, e $f(n)$ una funzione non negativa.

Sia $T(n)$ la ricorrenza definita per $n \in \mathbb{N}$ da

$$T(n) = aT(n/b) + f(n)$$

(l'espressione $aT(n/b)$ è da intendersi come

$$a' T(\lfloor n/b \rfloor) + a'' T(\lceil n/b \rceil),$$

per opportune costanti a' e a'' t.c. $a' + a'' = a$)

Allora il comportamento asintotico di $T(n)$ può essere caratterizzato nel modo seguente:

1. Se esiste una costante $\varepsilon > 0$ t.c.

$$f(n) = O\left(n^{\log_b a - \varepsilon}\right)$$

allora

$$T(n) = \Theta\left(n^{\log_b a}\right)$$

($\rightsquigarrow$ costo dominato dalle foglie)

$f(n)$ è polinomialmente più piccolo di $n^{\log_b a}$

($n^{\log_b a}$ cresce polinomialmente più velocemente di $f(n)$)

2. Se esiste una costante $k \geq 0$ t.c.

$$f(n) = \Theta\left(n^{\log_b a} \cdot \log^k n\right)$$

allora

$$T(n) = \Theta\left(n^{\log_b a} \cdot \log^{k+1} n\right)$$

$f(n)$ cresce più velocemente di $n^{\log_b a}$ per un fattore polilogaritmico

(costo di ogni livello dell'albero è $n^{\log_b a} \log^k n$ e ci sono $\log n$ livelli)

CASO SEMPLICE $k = 0$

$$f(n) = \Theta\left(n^{\log_b a}\right) \Rightarrow T(n) = \Theta\left(n^{\log_b a} \cdot \log n\right)$$

3. Se esiste una costante $\varepsilon > 0$ t.c.

$$f(n) = \Omega\left(n^{\log_b a + \varepsilon}\right)$$

e soddisfa la CONDIZIONE di REGOLARITÀ:

$$a f\left(\frac{n}{b}\right) \leq c \cdot f(n) \qquad c < 1 \text{ costante}$$

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

$f(n)$ cresce polinomialmente più velocemente di $n^{\log_b a}$

costo dominato dalla radice

$\square$

Merge Sort

$$T(n) = \begin{cases} \Theta(1) & n \leq 1 \\ T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) + \Theta(n) & n > 1 \end{cases}$$

$$T(n) = 2T(n/2) + \Theta(n)$$

$a = 2 \qquad b = 2 \qquad f(n) = n$

$$n^{\log_2 2} = n^1 = n$$

$$f(n) = \Theta(n^{\log_b a})$$

2° caso con $k = 0$

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

Ricerca Binaria Sx

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

$a = 1 \qquad b = 2$

$$n^{\log_b a} = n^{\log_2 1} = n^0 = 1 \qquad f(n) = \Theta(1)$$

II° caso $\quad f(n) = \Theta(n^{\log_b a}) = \Theta(1)$

$$T(n) = \Theta(n^{\log_b a} \cdot \log n) = \Theta(n^0 \cdot \log n) = \Theta(\log n)$$

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

Ricerca Binaria

$$T(n) \leq T(n/2) + O(1)$$

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

Massimo Divide et Impera / Somma e Contatori Divide et Impera

$$T(n) = \begin{cases} \Theta(1) & n \leq 1 \\ 2T(n/2) + \Theta(1) & n > 1 \end{cases}$$

$a = 2$
$b = 2$

$$n^{\log_b a} = n^{\log_2 2} = n^1 = n$$

$$f(n) = \Theta(1)$$

$$f(n) = O\left(n^{\log_b a - \varepsilon}\right) = O\left(n^{1-\varepsilon}\right)$$

$\underline{0 < \varepsilon \leq 1}$ $\quad\varepsilon = 0,2$

$$\boxed{I^\circ \text{ caso} \quad T(n) = \Theta(n)}$$

Dettagli tecnici

1) $n^{\log_b a}$ deve crescere polinomialmente più velocemente di $f(n)$ (per un fattore n^ε, $\varepsilon > 0$)

2) $f(n)$ cresce più velocemente di $n^{\log_b a}$ per un fattore polilogaritmico ($\Theta(\log^k n)$, $k \geq 0$)

3) $f(n)$ deve crescere polinomialmente più velocemente di $n^{\log_b a}$ (per un fattore n^ε $\varepsilon > 0$)

+ condizione di regolarità

$$a f\left(\frac{n}{b}\right) \leq c \cdot f(n) \qquad c < 1 \text{ costante}$$

$\left(f(n) = n^k \quad \text{sempre soddisfatto}\right)$

Attenzione: casi in cui il teorema non si può applicare

1 ESEMPIO

$$T(n) = 2T\left(\frac{n}{2}\right) + \frac{n}{\log n}$$

$$n^{\log_b a} = n^{\log_2 2} = n \qquad f(n) = \frac{n}{\log n}$$

$n^{\log_b a} = n$ cresce più velocemente di $f(n)$, ma solo per un fattore logaritmico

NO caso 1.

$$f(n) = \frac{n}{\log n} = n \cdot \log n^{-1}$$

$k = -1$

NO caso 2

vuole $k \geq 0$

2 Partizioni non bilanciate

$$T(n) = T\left(\frac{2}{3}n\right) + T\left(\frac{n}{3}\right) + f(n)$$

NO teorema principale