Dimostrazione del teorema principale (o teorema fondamentale delle ricorrenze). II^a parte.

$$T(n) = \begin{cases} \Theta(1) & n = 1 \\ aT\left(\frac{n}{b}\right) + f(n) & n = b^i, \quad i \in \mathbb{N} \end{cases}$$

$$T(n) = \dots\dots = n^{\log_b a} \underbrace{T(1)}_{\Theta(1)} + \sum_{j=0}^{\log_b n - 1} a^j f\left(\frac{n}{b^j}\right) = \Theta\left(n^{\log_b a}\right) + \sum_{j=0}^{\log_b n - 1} a^j f\left(\frac{n}{b^j}\right)$$

Costo della soluzione dei sottoproblemi di dim 1.

Costo della divisione e combinazione (D & I)

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

① $f(n) = O\left(n^{\log_b a - \varepsilon}\right) \quad \varepsilon > 0$

$$\Rightarrow g(n) = O\left(n^{\log_b a}\right)$$

$$T(n) = \Theta\left(n^{\log_b a}\right) + O\left(n^{\log_b a}\right) = \Theta\left(n^{\log_b a}\right) \quad \checkmark$$

② $f(n) = \Theta\left(n^{\log_b a}\right)$ $\Rightarrow$ $g(n) = \Theta\left(n^{\log_b a} \cdot \log n\right)$

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

③ $f(n) = \Omega\left(n^{\log_b a + \varepsilon}\right)$ $\varepsilon > 0$

+ cond. di regolarità

$a f\left(\frac{n}{b}\right) \leq c \cdot f(n)$ $c < 1$ $\Rightarrow g(n) = \Theta(f(n))$

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

□

ESERCITAZIONE

④ Calcolare a^n con $O(\log n)$ moltiplicazioni.

$$a^n = \left(a^{n/2}\right)^2 \qquad n \text{ pari}$$

$$a^n = \left(a^{n/2}\right)^2 \cdot a \qquad n \text{ dispari}$$

```
Power(a, n)
    if (n == 0) return 1;                    // base
    if (n == 1) return a;                    // base
    p = Power(a, n/2);                       // ricors.
    if (n è pari) return p * p;              // combinazione
    else return p * p * a;                   // combinazione
```

Costi (# moltiplicazioni)

$\rightarrow M\left(\frac{n}{2}\right)$

≤ 2 moltiplicazioni

$M(n) \leq \begin{cases} 0 & n \leq 1 \\ M\left(\frac{n}{2}\right) + 2 & n > 1 \end{cases}$

$M(n) = \#$ di moltiplicazioni

$M(n) \leq M\left(\frac{n}{2}\right) + 2 \leq \left(M\left(\frac{n}{4}\right) + 2\right) + 2 = M\left(\frac{n}{4}\right) + 4 \leq$

$M\left(\frac{n}{8}\right) + 2 + 4 = M\left(\frac{n}{8}\right) + 6 = M\left(\frac{n}{2^3}\right) + 2 \cdot 3 \leq \dots$

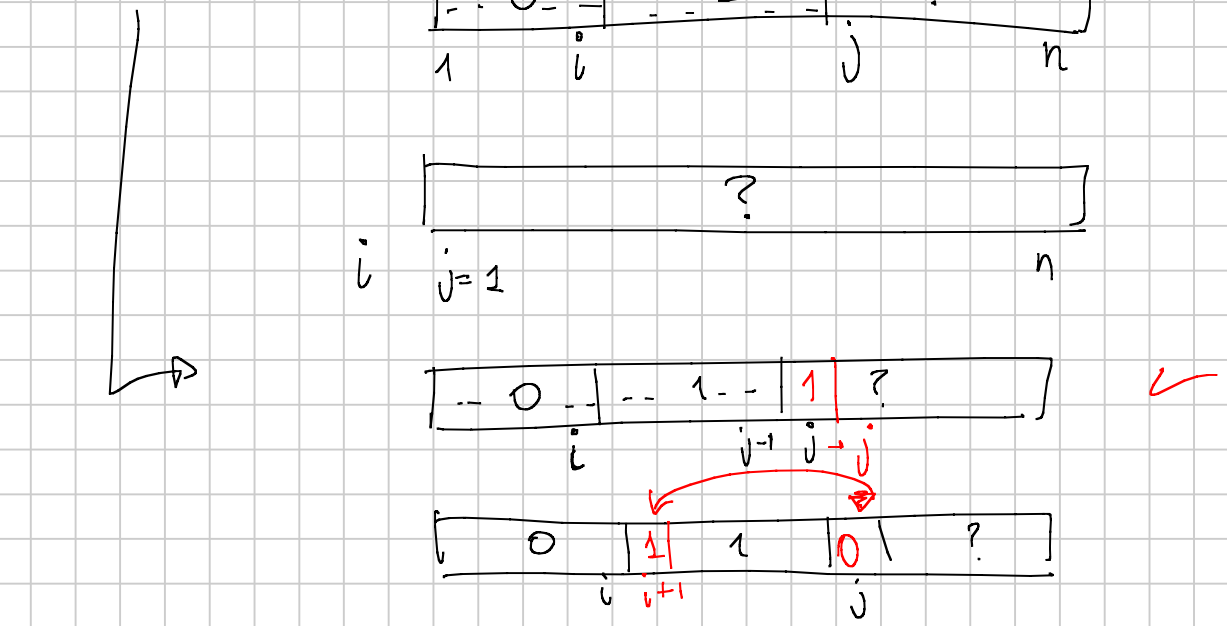
$\leq M\left(\frac{n}{2^i}\right) + 2 \cdot i \underset{i = \log_2 n}{=} M\left(\frac{n}{2^{\log_2 n}}\right) + 2 \cdot \log n = M(1) + 2 \log n =$

$= 2 \log n.$

$M(n) = O(\log n)$ □

(5) Ordinare (solo con confronti e scambi) un array a di dim. n che contiene 0 e 1.

j-esimo



```
OrdinaO1 (a)
  i = 0;
  for j = 1 to n {
        if (a[j] == 0) {
                i++;
                scambia (a[j], a[i])
        }
  }
```

$\Theta(1)$

$\Theta(1)$

$\Theta(n)$

$\Theta(n)$

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

ottimo

Invariante di ciclo

all'inizio della j-esima iterazione del for

$a(1..i)$ contiene elementi $= 0$

$a(i+1, \ldots, j-1)$ " " $= 1$

alla fine:

$j = n+1$

$a(1..i)$ 0

$a(i+1 \ldots n)$ 1.

(6) S: insieme di n interi

dato un intero k, determinare se esistono in S due elementi DISTINTI la cui somma è k.

S → memorizzato in un array a

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

```
CercaCoppia(a, k)
    MergeSort(a, 1, n);                          // Θ(n log n)
    for i = 1 to n {
        u = RicercaBinaria(a, k - a[i], 1, n);   // Θ(log n)
        if (u ≠ -1 && u ≠ i)  return <i, u>      // i ≠ u, a[i] + a[u] = k
    }
→   return <-1, -1>                              // ∄ una coppia di somma k
```

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

CercaCoppia2(a, k)

```
MergeSort(a, 1, n);
i = 1
j = n
while (i < j) {
    if (a[i]+a[j] == k)  return <i, j>;
    if (a[i]+a[j] < k)   i++;
    else j--;
}
return <-1, -1>
```

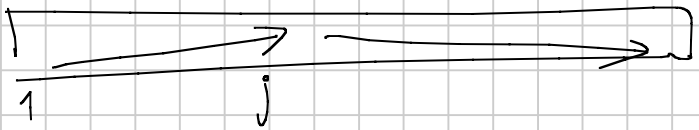
$\Theta(n \log n)$

$\Theta(1)$

$O(n)$

Esercizio 7

a: array di n interi distinti



$a[1..j] \rightarrow$

$a(j+1 \ldots n) \searrow$

$a(j) > a(j+1) \qquad (j < n)$

casi particolari

(2) Limite inferiore

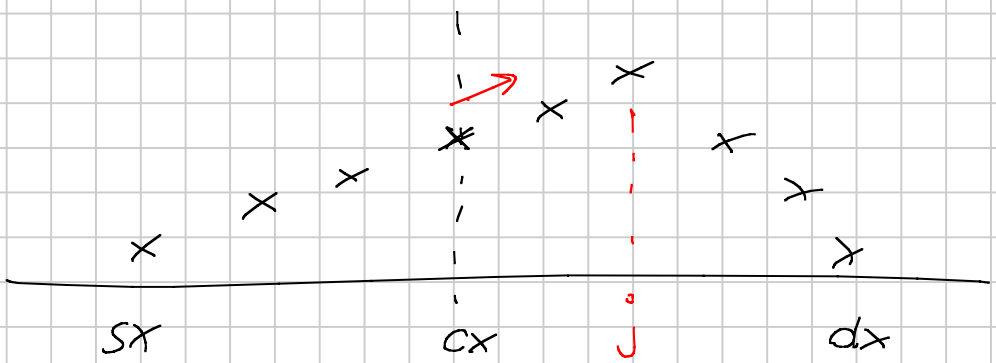
con la tecnica dell'Albero di Decisione

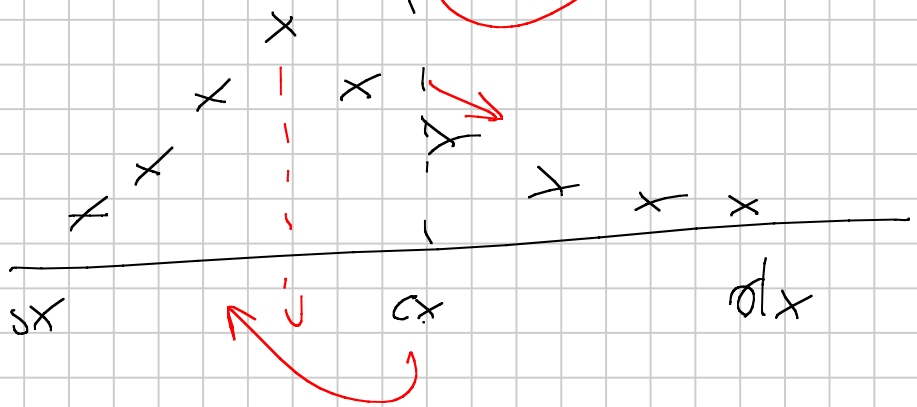
$$L_{\Pi}(n) = \Omega(\log s(n))$$

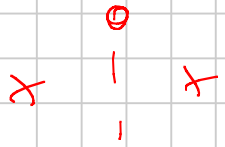
$$s(n) = n$$

$$\Rightarrow L_{\Pi}(n) = \Omega(\log n)$$

(3)


sx
cx
j
dx


sx
j
cx
dx



```
TrovaPos(a, sx, dx)
if (sx > dx) return -1;          Θ(1)
if (sx == dx) return sx;         Θ(1)
cx = (sx+dx)/2                   Θ(1)
if( a[cx] < a[cx+1]) return TrovaPos(a, cx+1, dx);
else return TrovaPos(a, sx, cx);
```

} $T(\frac{n}{2})$

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

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

⑧ Conta Occorrenze

Q: array ordinato, con elementi ripetuti
dato una chiave k, trovare il # di occorrenze di k in Q

$O(\log n + occ)$ $\quad occ = \#$ occorrenze di k in Q

~~$\Theta(\log n)$~~

occ può essere asintoticamente maggiore di $\log n$

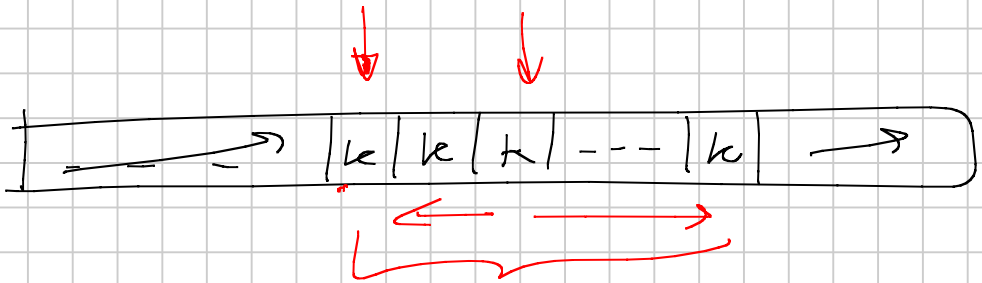
① L'algoritmo di costo $\Theta(\log n)$ è ottimo.

↝ Limite inferiore

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

~~#~~

$S(n) = n+1$
$0 \le occ \le n$



$O(\log n + occ)$ { in tempo $O(\log n)$ trovo k
in tempo $\Theta(occ)$ conto le occorrenze di k

RicercaBinaria DX/SX (a, k, sx, dx) // trova posizione della PRIMA/ULTIMA occorrenza di k in a

```
if (sx > dx) return -1;
if (sx == dx) {
        if (a[sx] == k) return sx;
        else return -1;
}
cx = (sx + dx) / 2;
if ( k <= a[cx] ) return RicercaBinariaSx (a, k, sx, cx);
else return RicercaBinariaSx (a, k, cx+1, dx);
```

ricercaBinariaDx (a, k, cx, dx)

↳ RicBinDx (a, k, sx, cx-1)

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

ContaOccorrenze (a, k)

```
primo = RicercaBinariaSx (a, k, 1, n);
if (primo == -1) return 0;   // k ∉ a
ultimo = RicercaBinariaDx (a, k, 1, n);
return ultimo - primo + 1
```

$\#$ occ.

$\Theta(\log n)$

$\Theta(1)$

$\Theta(\log n)$

$\Theta(1)$

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