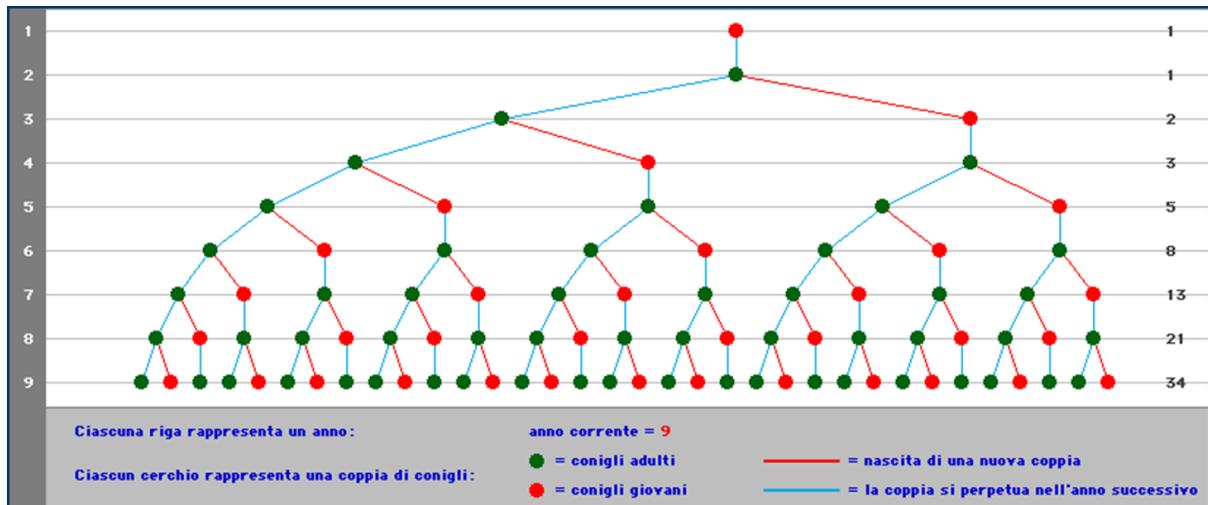


Algoritmo = sequenza di passi (eseguiti in maniera non ambigua, ordine prestabilito).
 Il programma aiuta a tradurre l'algoritmo in una lingua comprensibile al calcolatore.

Analisi teorica degli algoritmi

(Non richiesto all'esame)

ISOLA DEI CONIGLI



$$F_1=1$$

$$F_2=1$$

$$F_n=F_{n-1}+F_{n-2} \quad a^n = a^{n-1} - a^{n-2} \quad a^{n-2}(a^2-a-1)=0$$

$$F_n = \frac{1}{\sqrt{5}} (\phi_1^n - \phi_2^n)$$

Algoritmo Fibonacci 1 (intero n) → intero

$$\text{return } \frac{1}{\sqrt{5}} (\phi_1^n - \phi_2^n)$$

Costo unario

Algoritmo non troppo preciso perché approssima la radice di 5.

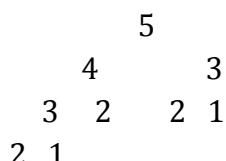
Algoritmo Fibonacci2 (intero n) → intero

if($n \leq 2$) return 1

else return Fibonacci2($n-1$)+Fibonacci2($n-2$)

Radice = primo nodo

Foglia = nodo non connesso ad un successivo

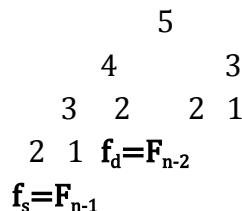


$$1) f = F_n$$

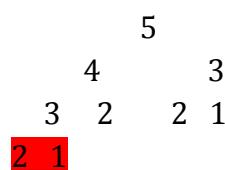
2) In qualunque albero che ha 2 figli per ogni nodo $i = f-1$

Principio di induzione

Caso base = 1 foglia



$$f = f_s + f_d = F_{n-1} + F_{n-2} = F_n$$



$$\hat{i} = \hat{f} - 1$$

$$\hat{f} = f - 1$$

$$\hat{i} = i - 1$$

$$i-1 = f-1-1$$

$$T(n) = 1*f + 2*i = 1*F_n + 2*(F_n - 1) = 3F_n - 2$$

Costo molto alto di tipo esponenziale

Inefficiente perché ricalcola sempre gli stessi sottoproblemi.

Algoritmo Fibonacci3 (intero n) → intero

Sia Fib[n] di itneri

$$\text{Fib}[1] = \text{Fib}[2] = 1$$

For $i=3$ to n //estremi inclusi

$$\text{Fib}[i] = \text{Fib}[i-1] + \text{Fib}[i-2]$$

Return Fib[n]

1	1	2	3	5	8	
1	2	3	4	5	6	n

$$T(n) = 1 + 1 * (n-3+1) + 1 = 1 + n - 2 + 1 = n$$

Costo n con programmazione dinamica

Algoritmo Fibonacci4 (intero n) → intero

$$a=b=1$$

For i=3 to n

c=a+b

a=b

b=c

return b

$$T(n) = 1 + 3(n-2) + 1 = 1 + 3n - 6 + 1 = 3n - 4$$

In tempo costa di più ma in spazio molto meno.

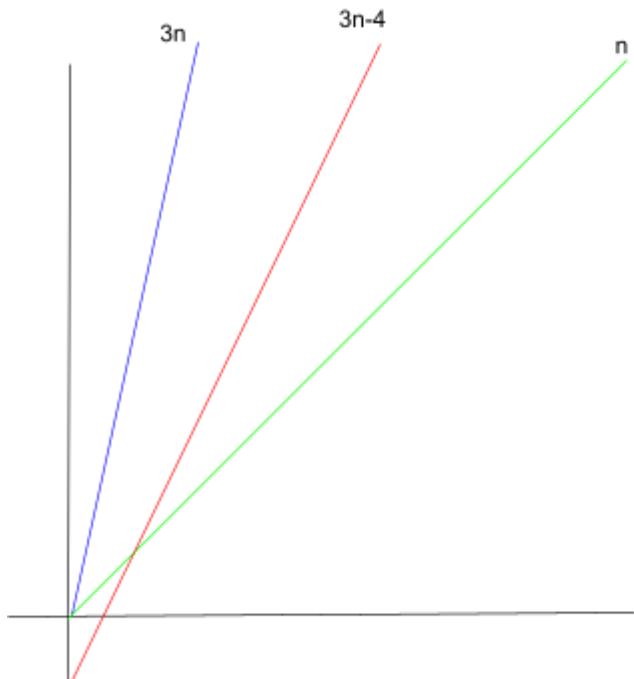
NOTAZIONI ASINTOTICHE

O GRANDE

$$f(n) = O(g(n))$$

$$\exists n_0 \geq 0, c > 0: \forall n \geq n_0 \quad f(n) \leq c * g(n)$$

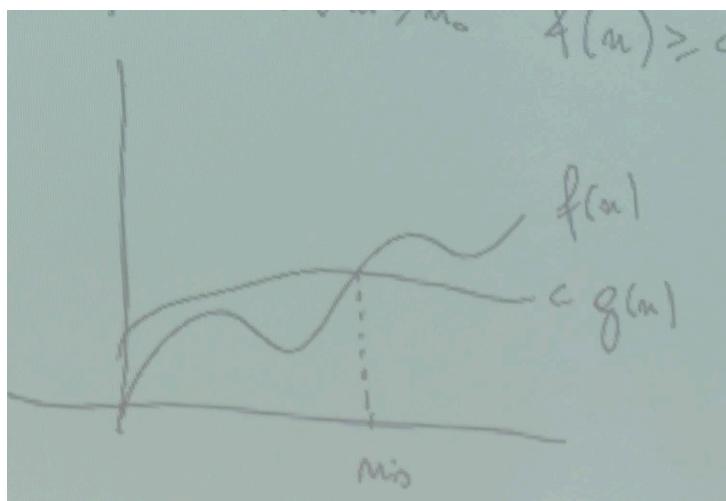
$$3n - 4 = O(n)$$



OMEGA GRANDE

$$f(n) = \Omega(g(n))$$

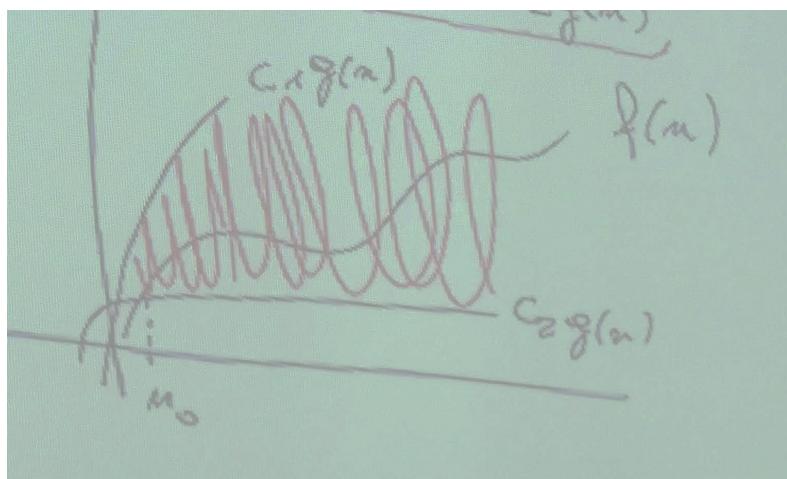
$$\exists n_0 \geq 0, c > 0: \forall n \geq n_0 \quad f(n) \geq c * g(n)$$



THETA GRANDE

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

$$\exists n_0 \geq 0, c_1 > 0, c_2 > 0: \forall n \geq n_0 \quad c_1 g(n) \leq f(n) \leq c_2 g(n)$$



Algoritmo ricercaSeq (lista L, elemento x) \rightarrow booleano

for each ($y \in L$)

if($y=x$) return T

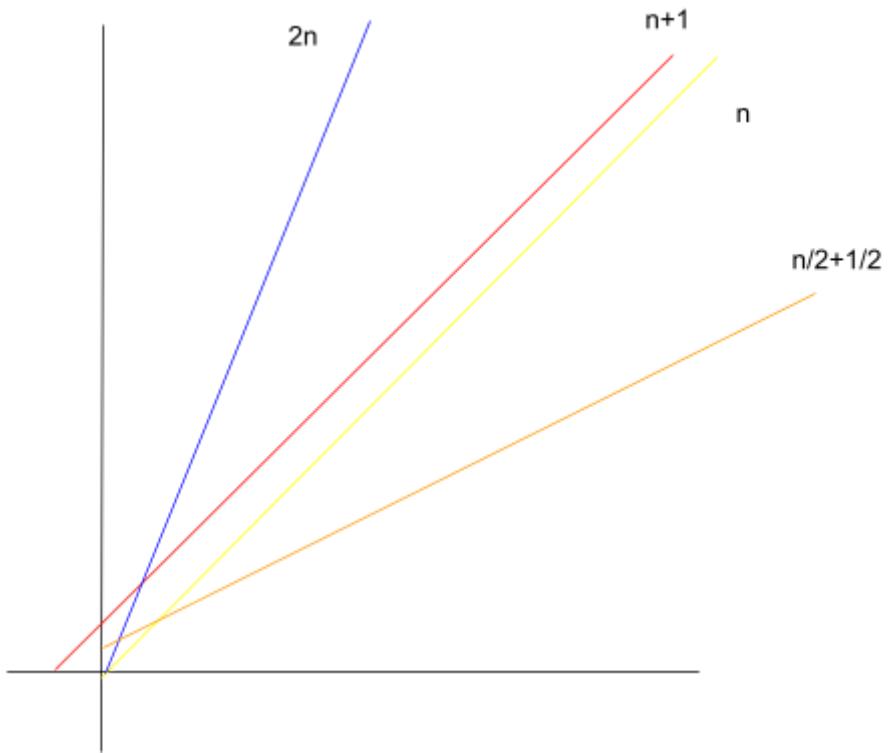
return F

$$T_{\text{best}}(n) = 1$$

$$T_{\text{worst } 1}(n) = n$$

$$T_{\text{worst } 2}(n) = n+1$$

$$T_{\text{avg}}(n) = \frac{1}{n} * 1 + \frac{1}{n} * 2 + \frac{1}{n} * 3 + \dots + \frac{1}{n} * i + \dots + \frac{1}{n} * n = \frac{1}{n} \sum_{i=1}^n i = \frac{n}{2} + \frac{1}{2}$$



$$\begin{array}{ll}
 n + 1 = \Omega(n) & n + 1 \geq n \\
 n + 1 = O(n) & n + 1 \leq 2n \\
 n + 1 = \Theta(n)
 \end{array}$$

$$\begin{array}{ll}
 \frac{n}{2} + \frac{1}{2} \leq n & \frac{n}{2} + \frac{1}{2} = O(n) \\
 \frac{n}{2} + \frac{1}{2} \geq c * n & \frac{n}{2} + \frac{1}{2} = \Omega(n) \\
 \frac{n}{2} + \frac{1}{2} = \Theta(n)
 \end{array}$$

Algoritmo ricercaBin (array L, elemento x) → booleano

a=1

b=lunghezza di L

while($L[\frac{a+b}{2}] \neq x$)

m= $\frac{a+b}{2}$

if($L[m] > x$) b=m-1

else a=m+1

if(a>b) return F

return T

$$\begin{aligned}
n &= 2^k \\
&\quad 2^{k-1} \\
&\quad 2^{k-2} \\
&\dots \\
&\quad 2^0 = 1
\end{aligned}$$

$$T(n) = 3 + 4(k+1) = 3 + 4k + 4 = 4k + 7 \quad \Theta(k)$$

$$\begin{aligned}
k &= \log_2 n \\
T(n) &= \Theta(\log_2 n)
\end{aligned}$$

Algoritmo ricercaBinRic (array L, elemento x) → booleano

```

n=lunghezza L
if(n=0) return F
i =  $\frac{n}{2}$ 
if(L[i]=x) return T
else if(L[i]>x) return ricercaBinRic(L[1:i-1], x)
    else return ricercaBinRic(L[i+1:n], x)

```

$$\begin{aligned}
T(0) &= 1 \\
T(1) &= 3 \\
T(n) &= 3 + 1 + T\left(\frac{n}{2}\right) \\
&\quad 3 + 1 + 1 + T\left(\frac{n}{2}\right) \\
&= k + T\left(\frac{n}{2}\right)
\end{aligned}$$

TECNICHE PER PASSARE DA UNA FUNZIONE DI RICORRENZA AL COSTO ASINTOTICO

- **Srotolamento** = srotolare fino ad arrivare ad un caso base

$$\begin{aligned}
T(n) &= K + T\left(\frac{n}{2}\right) \\
T\left(\frac{n}{2}\right) &= K + T\left(\frac{n}{4}\right) \text{ Sostituisco ad ogni } n \frac{n}{2} \\
T\left(\frac{n}{4}\right) &= K + T\left(\frac{n}{8}\right) \quad \frac{n}{4} * \frac{1}{2} \\
T(n) &= K + K + T\left(\frac{n}{4}\right) = K + K + K + T\left(\frac{n}{8}\right) = 3K + T\left(\frac{n}{2^3}\right) = iK + T\left(\frac{n}{2^i}\right)
\end{aligned}$$

Quanti sono gli i passi in modo tale che la i se ne vada? Chiedo che la dimensione dell'input sia uguale alla dimensione del caso base.

$$\frac{n}{2^i} = 1 \quad n = 2^i \quad i = \log_2 n$$

$$T(n) = K * \log_2 n + T(1) = K * \log_2 n + c$$

$\Theta(\log n)$

Ex

$$T(1) = 1$$

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

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

$$T\left(\frac{n}{4}\right) = \frac{n}{4} + T\left(\frac{n}{8}\right)$$

$$T\left(\frac{n}{8}\right) = \frac{n}{8} + T\left(\frac{n}{16}\right)$$

$$T(n) = n + \frac{n}{2} + T\left(\frac{n}{4}\right) = n + \frac{n}{2} + \frac{n}{4} + T\left(\frac{n}{8}\right) = n + \frac{n}{2} + \frac{n}{4} + \frac{n}{8} + T\left(\frac{n}{16}\right)$$

$$T(n) = \sum_{j=0}^{4-1} \frac{n}{2^j} + T\left(\frac{n}{2^4}\right) = \sum_{j=0}^{i-1} \frac{n}{2^j} + T\left(\frac{n}{2^i}\right)$$

$$\frac{n}{2^i} = 1 \quad n = 2^i \quad i = \log_2 n$$

$$T(n) = \sum_{j=0}^{\log_2 n - 1} \frac{n}{2^j} + T(1) = \sum_{j=0}^{\log_2 n - 1} \frac{n}{2^j} + 1 = 1 + n \sum_{j=0}^{\log_2 n - 1} \left(\frac{1}{2}\right)^j = 1 + n \left(\frac{\left(\frac{1}{2}\right)^{\log_2 n} - 1}{\frac{1}{2} - 1}\right) =$$

$$1 - 2n((2^{-1})^{\log_2 n} - 1) = 1 - 2n((2^{\log_2 n})^{-1} - 1) = 1 - 2n(n^{-1} - 1) = 1 - 2 + 2n = -1 + 2n$$

$$\left[\log_2 n = k \quad 2^k = n \quad 2^{\log_2 n} = 2^k = n \right] \quad T(n) = \Theta(n)$$

• **Sostituzione**=indovinare una maggiorazione della $T(n)$ [$T(n) \leq c * n$] $O(n)$
dimostrazione per induzione

$$T(1) = 1$$

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

$$T(n) \leq c * n$$

$$1) \quad T(1) \leq c * 1 \quad 1 \leq c * 1$$

$$2) \quad T\left(\frac{n}{2}\right) \leq c * \frac{n}{2} \text{ ipotesi induttiva} \quad T(n) = n + T\left(\frac{n}{2}\right) \leq n + c * \frac{n}{2} \leq c * n$$

$$1 + \frac{c}{2} \leq c \quad 1 \leq c * \frac{c}{2} \quad 1 \leq \frac{c}{2} \quad c \geq 2 \quad T(n) = O(n)$$

• **Teorema Master**

La chiamata ricorsiva DEVE lavorare su una frazione dell'input

$$T(1) = 1$$

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

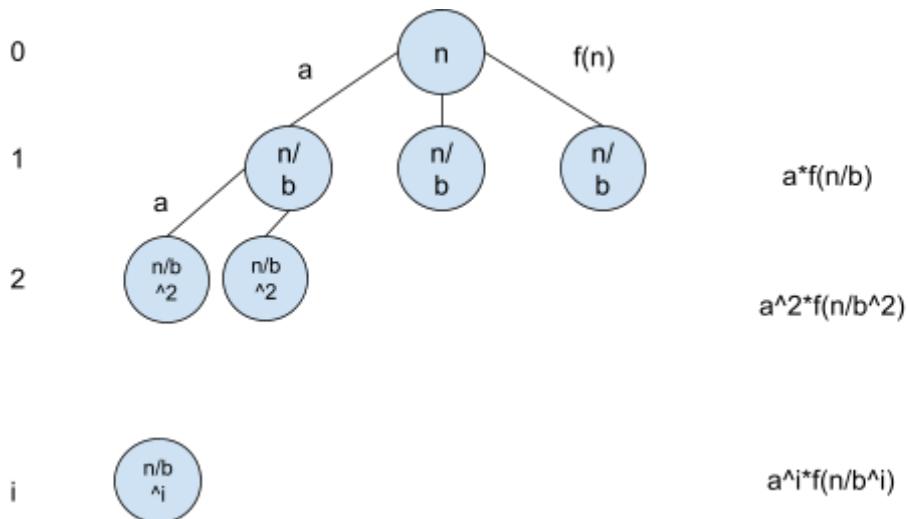
Confronto asintotico tra $f(n)$ e un termine di paragone che dipende dal numero e dal tipo di chiamate ricorsive $n^{\log_b a}$.

$$1) f(n) = O(n^{\log_b a - \varepsilon}) \quad \varepsilon > 0 \rightarrow T(n) = \Theta(n^{\log_b a})$$

$$2) f(n) = \Theta(n^{\log_b a}) \rightarrow T(n) = \Theta(n^{\log_b a} * \log n)$$

$$3) f(n) = \Omega(n^{\log_b a + \varepsilon}) \quad \varepsilon > 0 \\ | \rightarrow T(n) = \Theta(f(n))$$

$$a * f(\frac{n}{b}) \leq c * f(n) \quad c < 1$$



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

$$T(n) = \sum_{i=0}^{\log_b n} a^i f\left(\frac{n}{b^i}\right)$$

Srotolamento

$$T(1) = 1$$

$$T(n) = 9T\left(\frac{n}{3}\right) + n$$

$$T\left(\frac{n}{3}\right) = 9T\left(\frac{n}{9}\right) + \frac{n}{3}$$

$$T\left(\frac{n}{9}\right) = 9T\left(\frac{n}{27}\right) + \frac{n}{9}$$

$$T\left(\frac{n}{27}\right) = 9T\left(\frac{n}{81}\right) + \frac{n}{27}$$

$$\begin{aligned} T(n) &= 9 * (9T\left(\frac{n}{9}\right) + \frac{n}{3}) + n = 9 * (9 * (9T\left(\frac{n}{27}\right) + \frac{n}{9}) + \frac{n}{3}) + n = \\ &= 9 * (9 * (9 * (9T\left(\frac{n}{81}\right) + \frac{n}{27}) + \frac{n}{9}) + \frac{n}{3}) + n = 9^4 T\left(\frac{n}{3^4}\right) + 27n + 9n + 3n + n = \\ &= 9^4 T\left(\frac{n}{3^4}\right) + \sum_{j=0}^{4-1} 3^j n = 9^4 T\left(\frac{n}{3^4}\right) + \sum_{j=0}^{i-1} 3^j n \end{aligned}$$

$$\left[\frac{n}{3^i} = 1 \quad n = 3^i \quad i = \log_3 n \right]$$

$$\begin{aligned} T(n) &= 9^{\log_3 n} T(1) + \sum_{j=0}^{\log_3 n - 1} 3^j n = (3^2)^{\log_3 n} + \sum_{j=0}^{\log_3 n - 1} 3^j n = (3^{\log_3 n})^2 + \sum_{j=0}^{\log_3 n - 1} 3^j n = n^2 + n \sum_{j=0}^{\log_3 n - 1} 3^j \\ &= n^2 + n(\frac{3^{\log_3 n} - 1}{3 - 1}) = n^2 + n(\frac{n-1}{2}) = n^2 + \frac{n^2}{2} - \frac{n}{2} = \frac{3}{2}n^2 - \frac{n}{2} \\ T(n) &= \Theta(n^2) \end{aligned}$$

Teorema Master

$$T(1) = c = \Theta(1)$$

$$T(n) = k + T(\frac{n}{2}) \quad a = 1 \quad b = 2 \quad f(n) = k = \Theta(1)$$

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

Seconda caso del teorema

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

$$T(1) = 1$$

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

$$a = 1 \quad b = 2 \quad n^{\log_b a} = n^{\log_2 1} = 1 \quad f(n) = n$$

Terza caso del teorema

$$\begin{array}{lll} n = \Omega(n^{0+\varepsilon}) & \varepsilon = \frac{1}{2} & n = \Omega(n^{\frac{1}{2}}) \\ & \varepsilon = 1 & n = \Omega(n) \quad n = \Theta(n) \text{ al limite} \end{array}$$

$$0 < \varepsilon \leq 1$$

$$1 * \frac{n}{2} \leq c n$$

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

$$T(1) = 1$$

$$T(n) = 9T(\frac{n}{3}) + n \quad f(n) = n \quad a = 9 \quad b = 3$$

$$n^{\log_b a} = n^{\log_3 9} = n^2$$

Primo caso del teorema

$$\begin{array}{ll} n = O(n^{2-\varepsilon}) & \varepsilon = 0.5 \quad n = n^1 = O(n^{1.5}) \\ & \varepsilon = 1 \quad n = n^1 = O(n^1) \end{array}$$

$$T(n) = \Theta(n^{\log_b a}) = \Theta(n^2)$$

$$T(1) = 1$$

$$T(n) = 3T(\frac{n}{9}) + k \quad f(n) = k = \Theta(1) \quad a = 3 \quad b = 9$$

$$n^{\log_b a} = n^{\log_9 3} = n^{\frac{1}{2}} = \sqrt{n}$$

Secondo caso del teorema

$$f(n) = k = \Theta(n^0) = \Theta(1) = O(n^{\frac{1}{2}-\varepsilon}) \quad \varepsilon = \frac{1}{4}$$

$$T(n) = \Theta(n^{\log_b a}) = \Theta(n^{\frac{1}{2}}) = \Theta(\sqrt{n})$$

$$T(1) = 1$$

$$T(n) = 3T(\frac{n}{9}) + n \quad f(n) = n = \Theta(1) \quad a = 3 \quad b = 9$$

$$n^{\log_b a} = n^{\log_9 3} = n^{\frac{1}{2}}$$

Terzo caso del teorema

$$f(n) = \Omega(n^{\log_b a + \varepsilon}) = \Omega(n^{\frac{1}{2} + \varepsilon}) \quad \varepsilon = \frac{1}{2}$$

$$af(\frac{n}{b}) \leq cf(n) \quad c = \frac{1}{9}$$

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

$$T(1) = 1$$

$$T(n) = T(\frac{2}{3}n) + 1 \quad f(n) = 1 \quad a = 1 \quad b = \frac{3}{2}$$

$$n^{\log_b a} = n^{\log_{\frac{3}{2}} 1} = n^0 = 1$$

Secondo caso del teorema

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

$$T(1) = 1$$

$$T(n) = 3T(\frac{n}{4}) + n \log n \quad f(n) = n * \log n \quad a = 3 \quad b = 4$$

$$n^{\log_b a} = n^{\log_4 3} = n^{0.8}$$

Terzo caso del teorema

$$n * \log n = \Omega(n^{0.8+\varepsilon}) \quad \varepsilon = 0.1$$

$$3 * \frac{n}{4} * \log \frac{n}{4} \leq c * n * \log n \quad c = \frac{3}{4}$$

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

$$T(1) = 1$$

$$T(n) = 3T(\frac{n}{3}) + n \log n \quad a = 3 \quad b = 3 \quad f(n) = n * \log n$$

$$n^{\log_b a} = n^{\log_3 3} = n$$

Terzo caso del teorema

$$n * \log n = \Omega(n^{1+\varepsilon}) \quad \text{Non esiste } \varepsilon > 0$$

NON SI PUÒ APPLICARE IL TEOREMA MASTER

Applichiamo lo srotolamento

$$T(\frac{n}{3}) = 3T(\frac{n}{9}) + \frac{n}{3} \log \frac{n}{3}$$

$$T\left(\frac{n}{9}\right) = 3T\left(\frac{n}{27}\right) + \frac{n}{9} \log \frac{n}{9}$$

$$T(n) = 3(3T\left(\frac{n}{9}\right) + \frac{n}{3} \log \frac{n}{3}) + n \log n = 3(3(3T\left(\frac{n}{27}\right) + \frac{n}{9} \log \frac{n}{9}) + \frac{n}{3} \log \frac{n}{3}) + n \log n$$

$$T(n) = 3^3 T\left(\frac{n}{3^3}\right) + n \log \frac{n}{9} + n \log \frac{n}{3} + n \log n = 3^3 T\left(\frac{n}{3^3}\right) + \sum_{j=0}^{3-1} n \log \frac{n}{3^j}$$

$$T(n) = 3^i T\left(\frac{n}{3^i}\right) + \sum_{j=0}^{i-1} n \log \frac{n}{3^j}$$

$$\frac{n}{3^i} = 1 \quad 3^i = n \quad i = \log_3 n$$

$$T(n) = 3^{\log_3 n} T(1) + \sum_{j=0}^{\log_3 n - 1} n \log \frac{n}{3^j} = n + n \sum_{j=0}^{\log_3 n - 1} \log \frac{n}{3^j} = n + n \sum_{j=0}^{\log_3 n - 1} (\log n - \log 3^j) =$$

$$= n + n \sum_{j=0}^{\log_3 n - 1} \log n - n \sum_{j=0}^{\log_3 n - 1} \log 3^j = n + n(\log n)^2 - n \sum_{j=0}^{\log_3 n - 1} j =$$

$$= n + n(\log n)^2 - n \left(\frac{(\log n - 1)(\log n)}{2}\right) = n + n(\log n)^2 - \frac{n}{2}(\log n)^2 + \frac{n}{2} \log n =$$

$$= n + \frac{n}{2}(\log n)^2 + \frac{n}{2} \log n$$

$$T(n) = \Theta(n(\log n)^2)$$

Ex 1. Per ogni coppia di funzioni si dica se $f(n) = \Omega(g(n))$, $f(n) = \Omega(g(n))$ oppure $f(n) = \Theta(g(n))$

$$f(n) = 2^{n/2} \quad f(n) = \Omega(g(n)) \\ g(n) = n^{3/2} \quad n_0 = 16 \quad c = 1$$

$$f(16) = 2^{\frac{16}{2}} = 2^8$$

$$g(16) = 16^{\frac{3}{2}} = 16\sqrt{16} = 2^6$$

$$f(n) = n^2 + 8n \quad n_0 = 1 \quad c = 1 \\ g(n) = n^{3/2}$$

$$f(1) = 1 + 8 = 9$$

$$g(1) = 1$$

$$f(n) = n^2 - 8n$$

$$g(n) = n^{\frac{3}{2}}$$

$$f(1) = 1 - 8 = -7$$

$$g(1) = 1$$

$$f(16) = 16 * 16 - 8 * 16 = 16(16 - 8) = 16 * 8$$

$$g(1) = 16\sqrt{16} = 16 * 4$$

$$f(n) = 2^{n/2}$$

$$g(n) = n^{\log \log n} \quad n_0 = 16 \quad c = 1$$

$$f(16) = 2^{\frac{16}{2}} = 2^8$$

$$g(16) = 16^{\log \log 16} = 16^{\log 4} = 16^2 = (2^4)^2 - 2^8$$

$$f(256) = 2^{\frac{256}{2}} = 2^{128}$$

$$g(256) = 256^{\log \log 256} = 256^{\log 8} = 256^3 = (2^8)^3 - 2^{24}$$

$$f(n) = n \log \log n$$

$$g(n) = n^{1+\varepsilon}$$

$$f(n) = n^n$$

$$g(n) = n! \quad f(n) = \Omega(g(n))$$

Dalla definizione

$$f(n) = 2^n$$

$$g(n) = 2^{n/4}$$

$$f(n) = n^2 + 8n$$

$$g(n) = n^2 (\log n)^2 \quad f(n) = O(g(n))$$

$$n_0 = 4 \quad c = 1$$

$$f(4) = 16 + 8 - * 4 = 4(4 + 8) = 16 + 16 * 2 = 16 * 3$$

$$g(4) = 16 * 4$$

$$f(n) = 9^{\log_3 n}$$

$$g(n) = n^{3/2} \quad f(n) = (3^2)^{\log_3 n} = (3^{\log_3 n})^2 = n^2$$

$$f(n) = 1$$

$$g(n) = 2^9$$

$$f(n) = 4 \log_2 n$$

$$g(n) = n^{\log_2 n}$$

Costanti $f(n) = \Theta(g(n))$

$$n_0 = 4 \quad c = 1$$

$$f(4) = 4 * 2 = 8$$

$$g(4) = 4^2 = 16$$

$$\begin{aligned} f(n) &= 4^{\log_2 n} = n^2 \\ g(n) &= n^{\log_2 4} = n^2 \end{aligned}$$

Ex 2. Trovare la soluzione delle seguenti relazioni di ricorrenza, usando, se possibile, il teorema Master (altrimenti procedere disegnando l'albero della ricorsione, per iterazione, o per sostituzione)

$$T(1) = 1$$

$$T(n) = 4T\left(\frac{n}{2}\right) + n \quad a = 4 \quad b = 2 \quad n^{\log_b a} = n^{\log_2 4} = n^2$$

$$f(n) = n$$

Primo caso del teorema

$$n = O(n^{2-\varepsilon}) \quad \varepsilon = \frac{1}{2} \quad \varepsilon = 1$$

$$T(n) = \Theta(n^2)$$

$$T(1) = 1$$

$$T(n) = 4T\left(\frac{n}{2}\right) + n^2 \quad f(n) = n^2$$

Secondo caso del teorema

$$T(n) = \Theta(n^2 \log n)$$

$$T(1) = 1$$

$$T(n) = 4T\left(\frac{n}{2}\right) + n^3 \quad f(n) = n^3$$

Terzo caso del teorema

$$n^3 = \Omega(n^{2+\varepsilon}) \quad \varepsilon = 1$$

$$4\left(\frac{n}{2}\right)^3 \leq c * n^3 \quad c = \frac{1}{2}$$

$$T(n) = \Theta(n^3)$$

$$T(1) = 1$$

$$T(n) = 3T\left(\frac{n}{3}\right) + n \quad f(n) = n \quad a = 3 \quad b = 3 \quad n^{\log_b a} = n^{\log_3 2} = n$$

Secondo caso del teorema

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

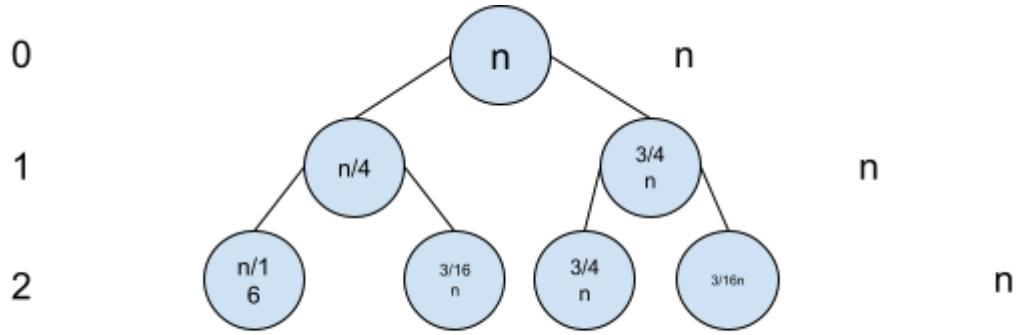
$$T(1) = 1$$

$$T(n) = T\left(\frac{n}{4}\right) + T\left(\frac{3n}{4}\right) + n \quad \text{Non si può applicare il teorema master}$$

$$T\left(\frac{n}{4}\right) = T\left(\frac{n}{16}\right) + T\left(\frac{3n}{16}\right) + \frac{n}{4}$$

$$T\left(\frac{3}{4}n\right) = T\left(\frac{3}{16}n\right) + T\left(\frac{3n}{16}\right) + \frac{3}{4}n$$

Troppo lungo



Mi concentro al problema più grande (destra) e sono sicuro che sono arrivato alla fine

$$\left(\frac{3}{4}\right)^i * n = 1$$

$$n = \left(\frac{4}{3}\right)^i \quad i = \log_{\frac{4}{3}} n$$

$$T(n) = (\log n + 1) * n = n \log n + n$$

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

$$T(1) = 1$$

$$T(n) = T(n - c) + T(c) + n^2 \quad c \geq 1$$

$$c = 1$$

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

$$c = n - 1$$

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

$$c = 2$$

$$T(n) = T(n - 2) + T(2) + n^2$$

$$c = n - 2$$

$$T(n) = T(2) + T(n - 2) + n^2$$

$$c = \frac{n}{2}$$

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

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

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

$$T(n - 2) = T(n - 3) + 1 + (n - 2)^2$$

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

$$= T(n - 3) + 1 + (n - 2)^2 + 1 + (n - 1)^2 + 1 + n^2 =$$

$$= T(n - i) + i + \sum_{j=0}^{i-1} (n - j)^2 = T(1) + n - 1 + \sum_{j=0}^{n-2} (n - j)^2 =$$

$$= n + \sum_{j=0}^{n-2} (n - j)^2 \text{ Calcolo una maggiorazione}$$

$$n + \sum_{j=0}^{n-2} (n - j)^2 \leq n + \sum_{j=0}^{n-2} n^2 \text{ La sommatoria a destra da sicuramente}$$

qualcosa di più grande rispetto a quella di sinistra

$$= n + n^2(n - 1) = n + n^3 - n^2$$

$$T(n) \leq \Theta(n^3)$$

$$T(n) = O(n^3)$$

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

$$f(n) = n^2$$

Terzo caso del teorema

$$f(n) = \Omega(n^{1+\varepsilon})$$

$$n^2 = \Omega(n^{1+\varepsilon})$$

$$2\left(\frac{n}{2}\right)^2 \leq c * n^2$$

$$c = \frac{1}{2}$$

$$T(n) = \Theta(n^2)$$

Calcoli il costo asintotico in funzione del numero n di elementi dell'array a[].

```
int fun(int a[], int n) {
    int i;
    if(n<2) return a[0];
    for(i=2;i<n;i++)
        a[i] = a[i]-a[i-1];
    return fun(a,1) + fun(a,n-1);
}
```

$$T(1) = 1$$

$T(n) = 1 + 1 * (n - 2) + 1 + T(1) + T(n - 1) = n + 1 + T(n - 1)$ Relazione di ricorrenza
Srotolamento

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

$$T(n - 2) = (n - 2) + 1 + T(n - 3)$$

$$\begin{aligned} T(n) &= n + 1 + (n - 1) + 1 + T(n - 2) = n + 1 + (n - 1) + 1 + (n - 2) + 1 + T(n - 3) \\ &= 3 + \sum_{j=0}^{n-1} (n - j) + T(n - 3) = i + \sum_{j=0}^{i-1} (n - j) + T(n - i) \end{aligned}$$

$$n - i = 1 \quad i = n - 1$$

$$\begin{aligned} T(n) &= n - 1 + \sum_{j=0}^{n-2} (n - j) + T(1) = n + \sum_{j=0}^{n-2} (n - j) = n + \sum_{j=0}^{n-2} n - \sum_{j=0}^{n-2} j = \\ &= n + n(n - 1) - \sum_{j=0}^{n-2} j = n + n(n - 1) - \frac{(n-2)(n-1)}{2} = \frac{n^2}{2} + \frac{3}{2}n - 1 \end{aligned}$$

$$T(n) = \Theta(n^2)$$

$T(n) = aT(n/2) + n \sqrt{n} + 3n \log n$ per $n > 1$; $T(1) = 1$; dove a è un intero positivo e \sqrt{n} è la radice quadrata di n. Si dica, giustificando le risposte, se per opportuni valori di a è possibile che sia:

a. $T(n) = \Theta(n^{1.5})$.

b. $T(n) = \Theta(n^2)$;

c. $T(n) = \Theta(n^{\log n})$;

d. $T(n) = \Theta(n^3)$.

(ESERCIZIO SUL TEOREMA MASTER)

a. $T(n) = \Theta(n^{1.5})$. **Sì a=1**

$$a = ? \quad b = 2 \quad f(n) = n\sqrt{n} + 3n \log n$$

$$n^{\log_b a} = n^{\log_2 a} \quad \Theta(n\sqrt{n}) = \Theta(n^{1.5})$$

Terzo caso del teorema

$$f(n) = \Omega(n^{\log_2 a + \varepsilon}) \quad a = 1 \quad \Omega(n^{\log_2 1 + \varepsilon}) = \Omega(n^{0+\varepsilon}) \quad \varepsilon = 1$$

$$a f(\frac{n}{b}) \leq c f(n)$$

$$f(\frac{n}{2}) = \frac{n}{2}\sqrt{\frac{n}{2}} + 3\frac{n}{2}\log\frac{n}{2} \leq c n\sqrt{n} + 3c n\log n$$

$$\frac{n}{2}\sqrt{\frac{n}{2}} + 3\frac{n}{2}(\log n - \log 2) \leq c n\sqrt{n} + 3c n\log n$$

$$\frac{n}{2}\sqrt{\frac{n}{2}} + 3\frac{n}{2}\log n - \frac{3}{2}n \leq c n\sqrt{n} + 3c n\log n \quad c = \frac{1}{2}$$

$$\frac{n}{2}\sqrt{\frac{n}{2}} - \frac{3}{2}n \leq \frac{n}{2}\sqrt{n} \quad 0 \leq -\frac{n}{2}\sqrt{\frac{n}{2}} + \frac{n}{2}\sqrt{n} + \frac{3}{2}n$$

b. $T(n) = \Theta(n^2)$; Sì scegliendo per forza $a=4$

Primo caso del teorema

$$f(n) = O(n^{\log_2 a - \varepsilon})$$

$$T(n) = \Theta(n^{\log_2 a}) = \Theta(n^2) \quad a = 4$$

$$\Theta(n^{1.5}) = O(n^{2-\varepsilon}) \quad \varepsilon = 1.5$$

c. $T(n) = \Theta(n^{\log n})$; Impossibile

Impossibile perché anche non facendo chiamate ricorsive spendo $n^{1.5}$

d. $T(n) = \Theta(n^3)$. Molto simile alla situazione 2 a=8

Se ne calcoli il costo asintotico in funzione del numero n di elementi dell'array a[].

(Nota: fun(a+n/3,n/3) significa la funzione fun applicata all'array a partire dall'elemento n/3 e con secondo parametro n/3).

```
int fun(int a[], int n) {
    int i,j,k;
    if(n<3) return 0;
    for(i=0;i<2;i++) {
        a[i]++;
        for(j=0;j<n;j++){
            a[j] += a[i];
            for(k=1;k<n-1;k++)
                a[i] += a[k];
        }
    }
    return fun(a,n/3) + fun(a+n/3,n/3) + fun(a+n/6,n/3);
}
```

$$T(1) = 1$$

$$\begin{aligned} T(n) &= 1 + 1 * 2 + (1 * n) * 2 + [(1 * (n - 2)) * n] * 2 + 1 + T\left(\frac{n}{3}\right) + T\left(\frac{n}{3}\right) + T\left(\frac{n}{3}\right) = \\ &= 1 + 2 + 2n + [n^2 - 2n] * 2 + 1 + 3T\left(\frac{n}{3}\right) = 2n^2 - 2n + 4 + 3T\left(\frac{n}{3}\right) \end{aligned}$$

$$f(n) = 2n^2 - 2n + 4 \quad \Theta(n^2)$$

$$n^{\log_b a} = n^{\log_3 3} = n$$

Terzo caso del teorema

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

$$\Theta(n^2) = \Omega(n^{1+\varepsilon}) \quad \varepsilon = 1$$

$$a * f\left(\frac{n}{b}\right) \leq c * f(n) \quad c < 1$$

$$3 * (2\left(\frac{n}{3}\right)^2 - 2\left(\frac{n}{3}\right) + 4) \leq c(2n^2 - 2n + 4)$$

$$\frac{2}{3}n^2 - 2n + 12 \leq 2cn^2 - 2cn + 4c \quad c = \frac{1}{3}$$

$$\frac{2}{3}n^2 - 2n + 12 \leq \frac{2}{3}n^2 - \frac{2}{3}n + \frac{4}{3} \quad 12 - \frac{4}{3} \leq 2n - \frac{2}{3}n \quad 32 \leq 4n \quad n \geq 8$$

Da un certo n_0 il teorema vale per $n \rightarrow \text{infinito}$

$$T(n) = \Theta(f(n)) = \Theta(n^2)$$

Se ne calcoli il costo asintotico in funzione del numero n di elementi degli array a[] e b[].

(Nota: fun(a+i,b+i, n/3) significa la funzione fun applicata agli array a partire dall'elemento i e con terzo parametro n/3).

```
int i,j,k;
int somma;
if(n<5) return (a[0]+b[0]+1);
somma=fun(a+n/3,b+n/3,n/3);
for(i=1;i<4;i+=2) {
    somma+=fun(a+i,b+i,n/3);
    for(j=0;j<n/2;j++)
        a[j] += a[j+1]+b[j+1];
}
return b[0]+somma;
}
```

$$T(1) = 1$$

$$\begin{aligned} T(n) &= 1 + 1 + T\left(\frac{n}{3}\right) + [(1 + T\left(\frac{n}{3}\right)) * 2] + (1 * \frac{n}{2}) * 2 + 1 = \\ &= 1 + 1 + T\left(\frac{n}{3}\right) + 2 + 2T\left(\frac{n}{3}\right) + n + 1 = 5 + n + 3T\left(\frac{n}{3}\right) \end{aligned}$$

$$f(n) = 5 + n \quad a = 3 \quad b = 3 \quad n^{\log_b a} = n^{\log_3 3} = n \quad \Theta(n)$$

Secondo caso del teorema

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

TIPO DATI DIZIONARIO

Oggetti: <elemento, chiave>

L'elemento è qualunque cosa (int, struct, ...)

OPERAZIONI:	VETTORE:	VETTORE ordinato:	LISTA:
insert	$\Theta(1)$	$\Theta(n)$	$\Theta(1)$
delete	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$
search	$\Theta(n)$	$\Theta(\log n)$	$\Theta(n)$

I dati vengono inseriti in fondo al vettore, nel primo spazio libero [Vettore]

Delete $\Theta(n)$ perché devo fare la search prima di cancellare $\Theta(1)$ [Vettore]

Per fare la **insert** devo fare lo shift verso destra degli elementi successivi all'elemento che voglio inserire, stessa cosa vale per la **delete** [Vettore ordinato]

Se il dizionario è già pieno è meglio il vettore ordinato, se devo inserire ma non li leggo molto spesso sarebbe meglio il vettore non ordinato. Però **nessuna delle due è veramente efficiente**.

Inserimento in testa ha costo costante, la **ricerca** è sequenziale, la **delete** pure è sequenziale perché prima dell'eliminazione devo trovarlo [Lista]

TIPO DI DATO PILA

[Implementazione pile e code.h](#)

[Implementazione.c](#)

Dati: oggetti

Sulla pila non voglio fare la ricerca.

La pila è **Lifo** (Last in first out).

OPERAZIONI:	PILA:
isEmpty	$\Theta(1)$
push	$\Theta(1)$
pop	$\Theta(1)$
top	$\Theta(1)$

TIPO DI DATO CODA

Dati: oggetti

La pila utilizza la logica **Fifo** (First in first out).

L'inserimento avviene da una parte ed esce dall'altra.

OPERAZIONI:	PILA:
isEmpty	$\Theta(1)$
enqueue	$\Theta(1)$
dequeue	$\Theta(1)$
first	$\Theta(1)$

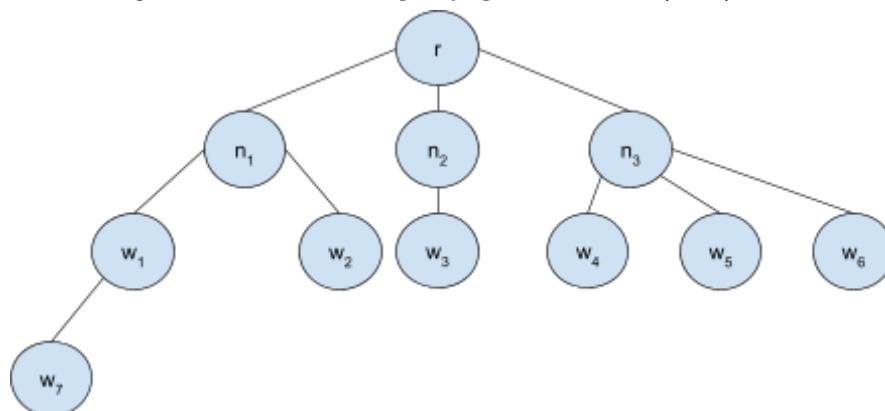
TIPO DI DATO ALBERO

[Implementazione alberi.h](#) [Implementazione.c](#)

Coppia $T(N, A)$ N =insieme dei nodi, A =insieme degli archi, $A \subseteq N \times N$

\forall nodo u tranne la radice ha 1 genitore $u \in N$: $(u, v) \in A$

\forall nodo u può avere \emptyset o più figli $w \in N$: $(u, v) \in A$



Radice: capostipite

Nodi con \emptyset figli: foglie

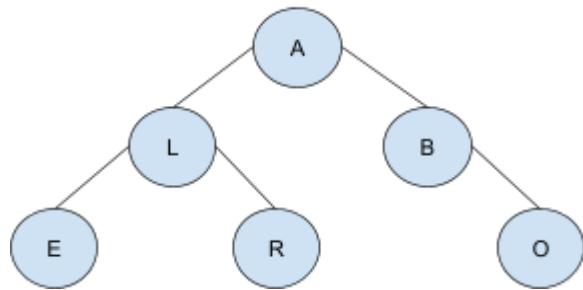
n^o figli di un nodo: grado

Livello di un nodo: distanza dalla radice

Altezza dell'albero: massimo livello

OPERAZIONI:	ALBERO:
Trova il padre	
Trova i figli	
visita	

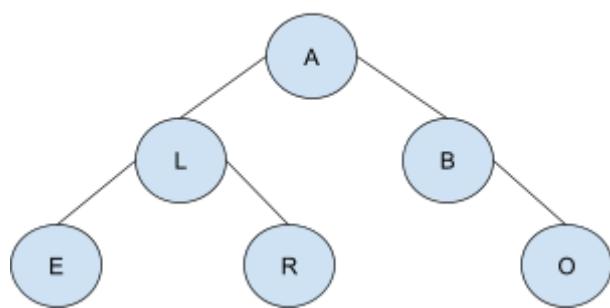
VETTORE PADRI

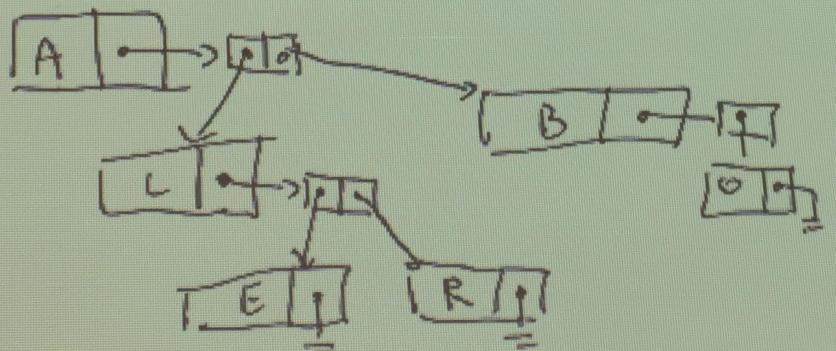
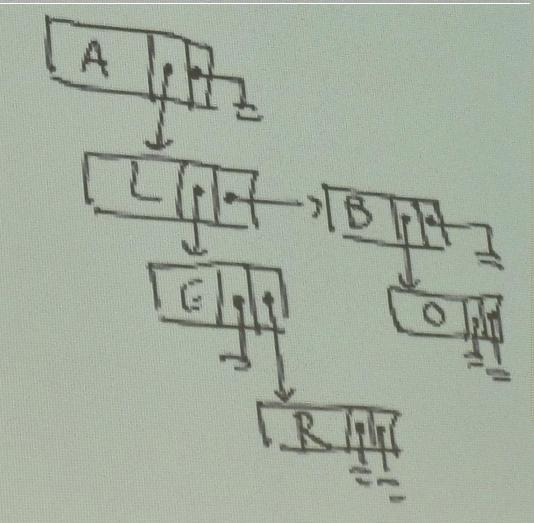
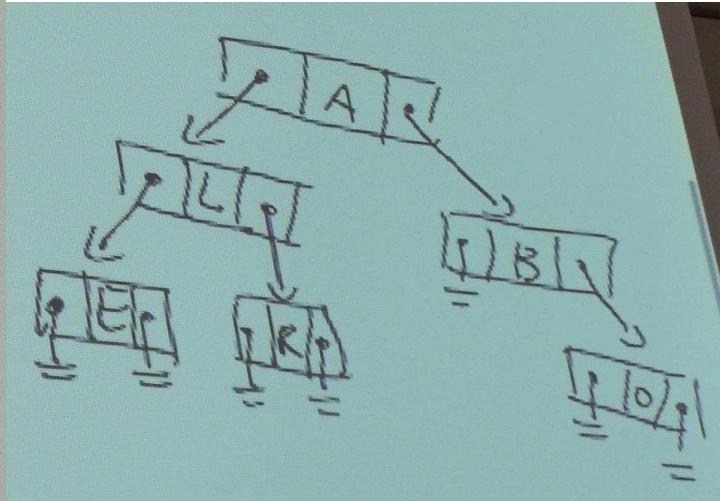
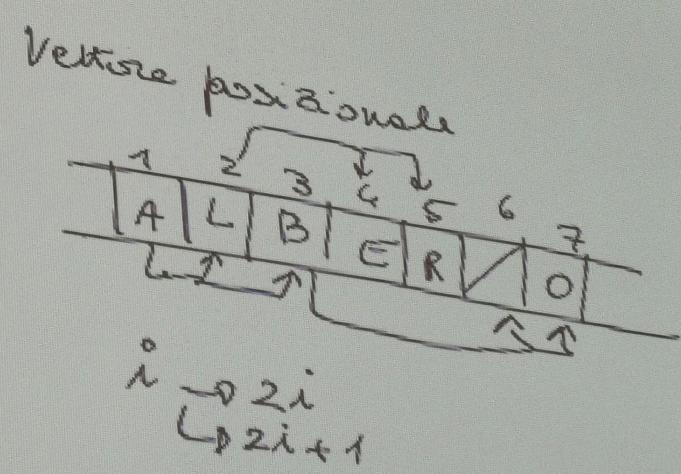


Albero binario

1	B	3
2	E	4
3	A	
4	L	3
5	R	4
6	O	1

VETTORE POSIZIONALE





PROCEDURA DI VISITA PILA

Algoritmo visita (nodo r)

$s = \{r\}$

while($s \neq \emptyset$)

 Estrai un nodo u da s

 Visita u

$s = s \cup \{\text{figli di } u\}$

Costo = $\Theta(n)$, per forza perché devo visitare tutti i nodi

Algoritmo visita DFS (nodo r)

DFS=depth first

Pila s

$s.push(r)$

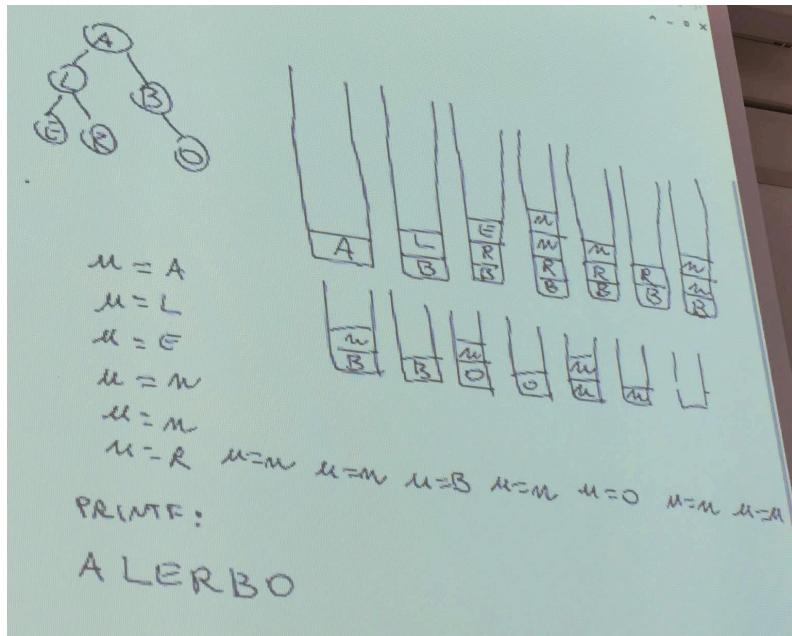
while(not(s.isEmpty))

```

u ← s.pop
if(u ≠ NULL)
    visita u
    s.push(figlio destro di u)
    s.push(figlio sinistro di u)

```

Costo = Θ(n)



PROCEDURA DI VISITA CODA

Algoritmo visita BFS (nodo r)

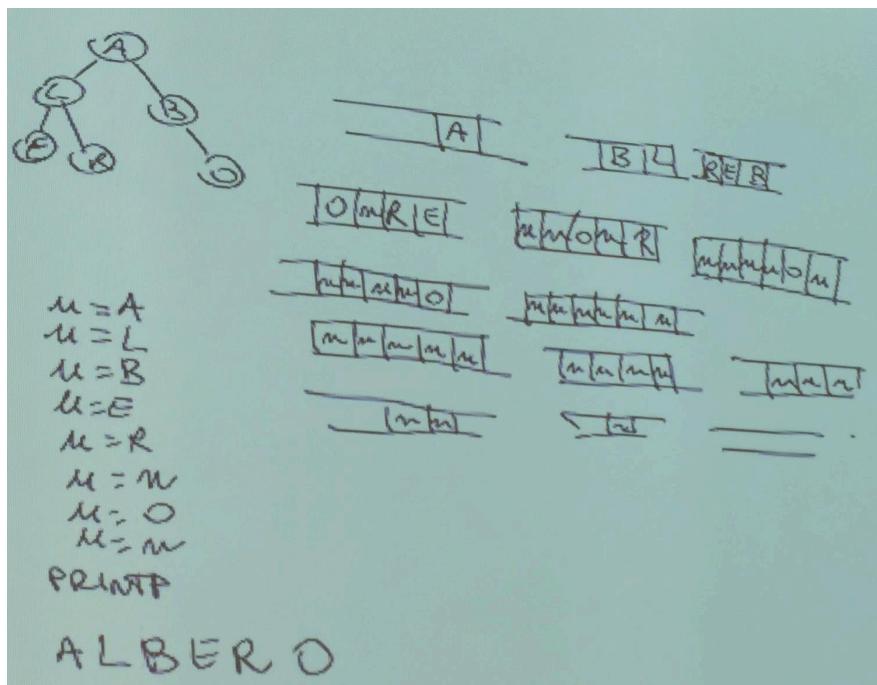
BFS=breadth first

```

Coda c
c.enqueue (r)
while(not(c.isEmpty()))
    u=c.dequeue
    if(u ≠ NULL)
        visita u
        c.enqueue (figlio sinistro di u)
        c.enqueue (figlio destro di u)

```

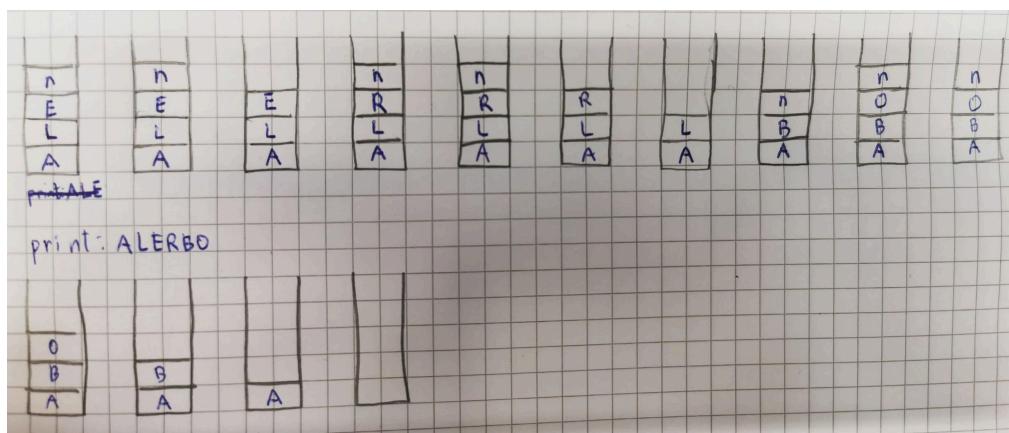
Costo = Θ(n)



Algoritmo visitaRicorsiva (nodo r)

```

if(r=NULL) return
visita r
Visita in pre ordine
visitaRicorsiva(figlio sinistro di r)
visitaRicorsiva(figlio destro di r)
    
```

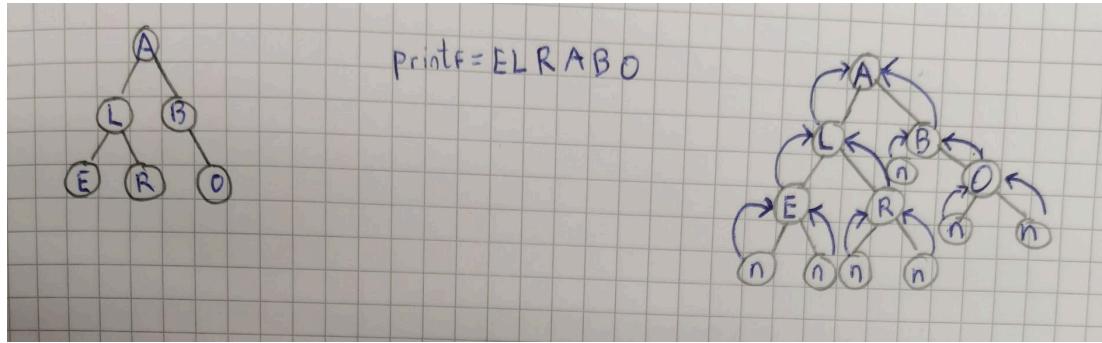


Stesso risultato della pila perché mi appoggio all'area di stack

Algoritmo visitaRicorsiva(nodo r)

```

if(r=null) return n
visitaRicorsiva(figlio sinistro di r)
Visita r
Visita simmetrica
visitaRicorsiva(figlio destro di r)
    
```



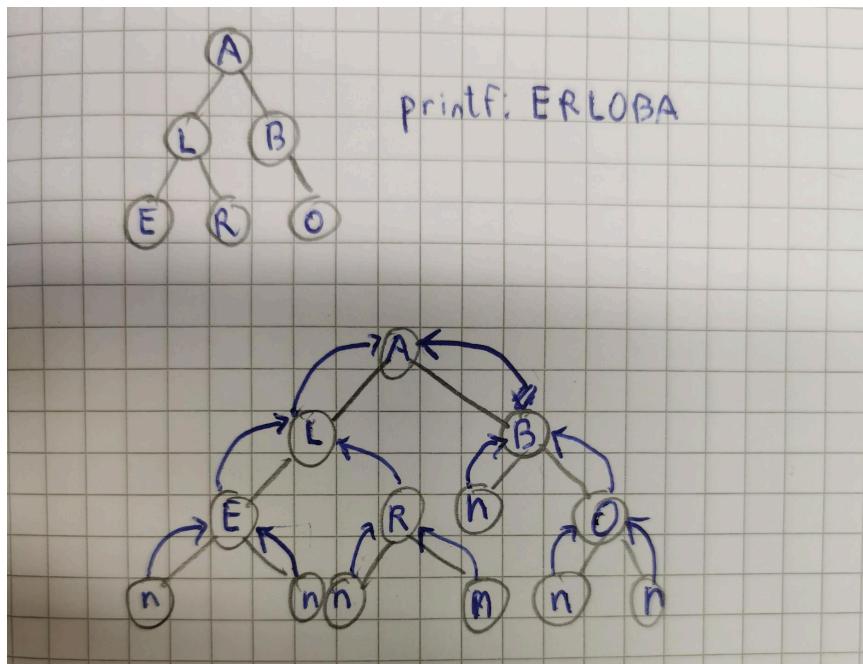
Algoritmo visitaRicorsiva(nodo r)

```

if(r=null) return n
visitaRicorsiva(figlio sinistro di r)
visitaRicorsiva(figlio destro di r)
Visita r

```

Post ordine

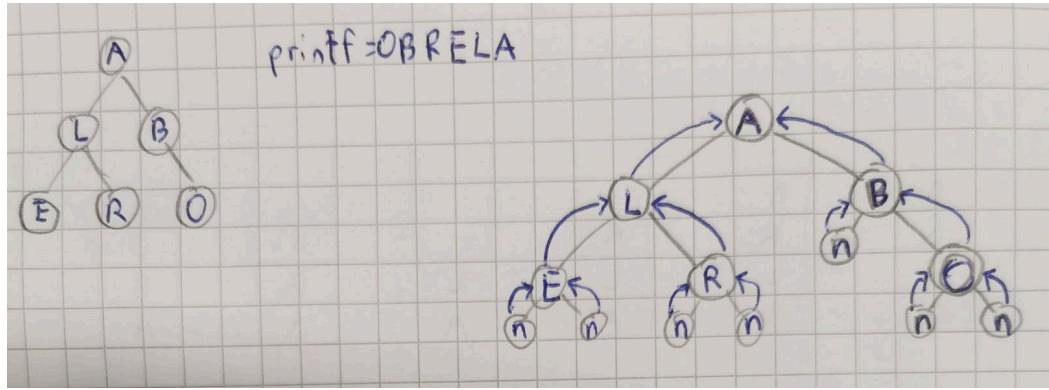


Algoritmo visitaRicorsiva(nodo r)

```

if(r=null) return n
visitaRicorsiva(figlio destro di r)
visitaRicorsiva(figlio sinistro di r)
Visita r

```



Albero binario con lo stesso numero di nodi a destra e sinistra = albero bilanciato

Albero binario con tutti i nodi da un lato = albero sbilanciato

Studiamo i due casi estremi per calcolare la complessità nei casi medi.

Visita post ordine (complessità)

1° caso

$$T(0) = 1$$

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

$$a = 2 \quad b = 2 \quad n^{\log_b a} = n^1 = n$$

$$f(n) = 2 = \Theta(1)$$

$$f(n) = O(n^{\log_b a - \epsilon})$$

$$\Theta(1) = O(n^{1-\epsilon})$$

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

2° caso

$$T(0) = 1$$

$$T(n) = 1 + 1 + T(n-1) + T(0) = 3 + T(n-1)$$

$$T(n-1) = 3 + T(n-2)$$

$$T(n-2) = 3 + T(n-3)$$

$$T(n) = 3 + 3 + T(n-2) = 3 + 3 + 3 + T(n-3) = 3 * 3 + T(n-3) = 3 * i + T(n-i)$$

$$n - i = 0 \quad i = n$$

$$T(n) = 3n + T(0) = 3n + 1$$

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

Ex 1.: calcolare l'altezza di un albero binario

Algoritmo calcoloAltezza (nodo r, int* h, int* hmax)

```
if(r=null){
```

```
    if(*h>*hmax) *hmax=*h
```

```
*h=-1;      -1 perché nel caso in cui l'albero sia vuoto non deve avere
```

altezza 0

```

        return n
    }
    calcoloAltezza(figlio destro di r, *h++, *hmax )
    calcoloAltezza(figlio sinistro di r, *h++, *hmax)

```

Algoritmo altezza(nodo r) → intero

```

if(r=null) return -1
sin=altezza(figlio sinistro di r)
des=altezza(figlio destro di r)
Return 1+max(sin, des)

```

Algoritmo foglie(nodo r) → intero

```

if(r=null) return -1
if(r=foglia) return 0
sin=foglie (figlio sinistro di r)
des=foglie (figlio destro di r)
Return 1+max(sin, des)

```

Ex 2. Calcolare il numero di foglie di un albero binario. Variante: calcolare il numero di foglie con chiave pari di un albero binario.

Algoritmo numFoglie(nodo r) → intero

```

if(r=null) return 0
if(r=foglia) return 1
sin = numFoglie(figlio sinistro di r)
des = numFoglie(figlio destro di r)
Return sin + des

```

Algoritmo numPariFoglie(nodo r) → intero

```

if(r=null) return 0
if(r=foglia){
    if(inf pari)return 1
    Else return 0
}
sin = numPariFoglie(figlio sinistro di r)
des = numPariFoglie(figlio destro di r)
Return sin + des

```

Ex 3. Calcolare il numero di nodi di un albero binario.

Algoritmo nodi(nodo r) → intero

```

if (r=null) return 0
sin = nodi(figlio sinistro di r)
des = nodi(figlio destro di r)
Return 1+(sin+des)      1+ per contare la radice

```

Ex 4. Cercare un elemento x tra le chiavi di un albero binario.

Algoritmo cerca(nodo r, elemento x) → booleano

```
if(r=null) return False  
if(r contiene x) return True  
if(cerca(figlio sinistro di r)) return True  
if(cerca(figlio destro di r)) return True  
Return False
```

Ex 5. Sommare le chiavi di un albero binario (ricorsivamente ed iterativamente).

Algoritmo somma(nodo r) → intero

```
if(r=null) return 0  
sin=somma(figlio sinistro di r)  
des=somma(figlio destro di r)  
return r→inf+sin+des
```

Albero.h

```
int somma(Albero r){  
    if(r=null) return 0;  
    Return ((r→inf)+somma(r→sinistro)+somma(r→destro));  
}  
  
int somma(Albero r){  
    Pila p=make.Pila();  
    Albero n=null;  
    int totale=0;  
    if(r!=NULL){  
        push(r, &p);  
        while(!emptyP(p)){  
            n=pop(&p);  
            if(n!=NULL){  
                totale=totale+(n→inf);  
                push(n→destro, &p);  
                push(n→sinistro, &p);  
            }  
        }  
    }  
    Return totale;  
}
```

PILA

```
int somma(Albero r){  
    Albero n=null;  
    int totale=0;  
    Coda c;
```

CODA

```

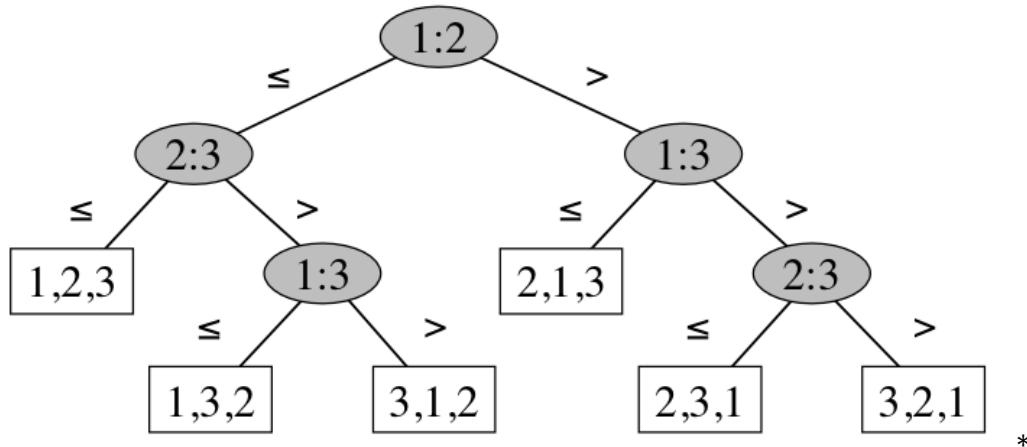
c=makeCoda();
if(r!=NULL){
    enqueue(r, &c);
    while(!(emptyC(c))){
        n=dequeue(&c);
        if(n!=NULL){
            totale=totale+(n->inf);
            enqueue(n->destro, &c);
            enqueue(n->sinistro, &c);
        }
    }
}
Return totale;
}

```

MODELLO BASATO SUI CONFRONTI cap4

Lower bound

Nel modello basato sui confronti non esiste nessun algoritmo che abbia una complessità inferiore di $\Theta(n \log n)$ per casi generici.



$h=3$

Foglie $k=3!=6$

Il numero di foglie = numero di confronti

Altezza albero = costo algoritmo

$$T(n) = h$$

$$\text{foglie } k = n!$$

[DOMANDA ESAME]

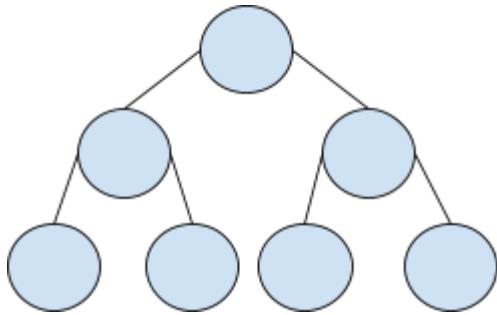
1) $h \geq \log_2 k$

2) $T(n) \geq \log_2 n! \geq n \log n$

Dimostrazione 1

Dimostrazione per induzione

Caso base: albero con un solo nodo, radice e foglia coincidono, non ha nodi interni, l'altezza=0



$$0 \geq \log_2 1$$

Il più grande dei sottoalberi ha un'altezza = all'altezza totale -1

$$h \geq \log_2$$

$$\begin{aligned}
 h &\geq \log_2 k \\
 h &\geq \log_2 \frac{k}{2} = \log_2 k - \log_2 2 \\
 h &\geq \log_2 k - 1 \\
 h &\geq \log_2 k
 \end{aligned}$$

Dimostrazione 2 (lower bound algoritmi di ordinamento)

$$T(n) \geq \log_2 n!$$

$$\begin{aligned}
 T(n) &\geq \log_2 n! = \log_2 (n * (n-1) * \dots * \frac{n}{2} * \dots * 2 * \dots * 1) = \log_2 n + \dots + \log_2 \frac{n}{2} + \dots + \log_2 1 \geq \\
 &\quad \log_2 n + \dots + \log_2 \frac{n}{2} \geq \frac{n}{2} \log_2 \frac{n}{2}
 \end{aligned}$$

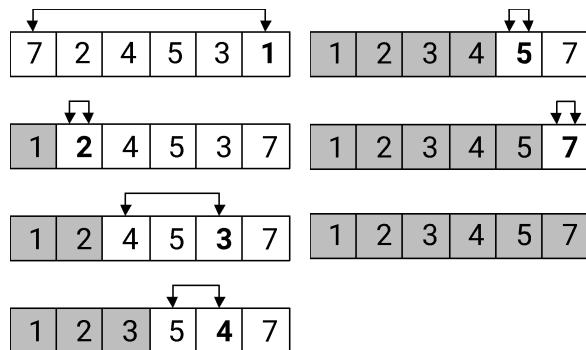
$$= \frac{n}{2} \log_2 n - \frac{n}{2} \log_2 2 = \frac{n}{2} \log_2 n - \frac{n}{2}$$

$T(n) = \Omega(n \log n)$ per definizione dimostro la tesi.

ALGORITMI QUADRATICI

Non richiedono un vettore d'appoggio ma ordinano sul posto, utilizzano gli scambi tra gli elementi. Sono ottimi per il costo in spazio ma non in tempo.

Selection sort



La porzione di testa contiene i mini e non dobbiamo toccarla.

Algoritmo selectionSort(array A)

For k=0 to n-2

 m=k+1

 For j=j+1 to n

 If (A[j]<A[m]) m=j

 Scambia A[m] con A[k+1]

7	2	4	3	1
---	---	---	---	---

n=5

k=0/3

k=0

m=1

j=%5

j=2 m=2

j=3

j=4

j=5 m=5

1	2	4	3	7
---	---	---	---	---

k=1

m=2

j=3/5

j=3

j=4

j=5

1	2	4	3	7
---	---	---	---	---

k=2

m=3

j=4/5

j=4 m=4

j=5

1	2	3	4	7
---	---	---	---	---

k=3

m=4

j=5/5

j=5

1	2	3	4	7
---	---	---	---	---

Complessità: conto quante volte viene fatta la if

$n - (k + 2) + 1$ ciclo ju j

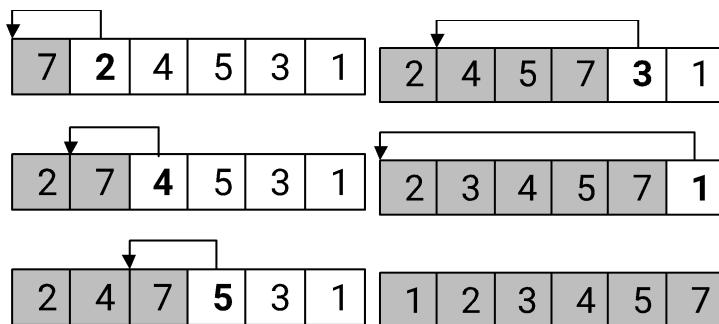
$$T(n) = \sum_{k=0}^{n-2} [n - (k + 2) + 1] \text{ numero di confronti}$$

$$\begin{aligned} &= \sum_{k=0}^{n-2} [n - k - 2 + 1] = \sum_{k=0}^{n-2} [(n - 1) - k] = \sum_{k=0}^{n-2} (n - 1) - \sum_{k=0}^{n-2} k = (n - 1)^2 - \frac{(n-2)(n-1)}{2} = \\ &= \frac{2(n^2+1-2n)-(n^2-3n+2)}{2} = \frac{n^2-n}{2} = \frac{n^2}{2} - \frac{n}{2} = \frac{n(n-1)}{2} \end{aligned}$$

$$T(n) = \Theta(n^2)$$

Insertion sort

La parte di destra è ordinata ma non so se contiene i minimi, voglio prendere la cella adiacente sulla destra, prendo il valore che contiene e lo inserisco nel punto giusto in testa.



Algoritmo insertionSort(array A)

For k=1 to n-1

 x=A[k+1]

 For j=k down to 0

 if(j>0 and A[j]<=x) break

 if(j<k)

 For t=k down to j+1 A[t+1]=A[t]

 A[j+1]=x

7	2	4	3	1
---	---	---	---	---

n=5

k=1/4

k=1
x=2
j=1/0
j=1
j=0
t=1/1 A[2]=A[1]

7	7	4	3	1
---	---	---	---	---

2	7	4	3	1
---	---	---	---	---

k=2
x=4
j=2/0
j=2
j=1
t=2/2 A[3]=A[2]

2	7	7	3	1
---	---	---	---	---

2	4	7	3	1
---	---	---	---	---

k=3
x=3
j=3/0
j=3
j=2
j=1
t=3/2 A[4]=A[3]

2	4	7	7	1
---	---	---	---	---

A[3]=A[2]

2	4	4	7	1
---	---	---	---	---

2	3	4	7	1
---	---	---	---	---

k=4
x=1
j=4/0
j=4
j=3
j=2
j=1
j=0

t=4/1 A[5]=A[4]

2	3	4	7	7
---	---	---	---	---

t=3 A[4]=A[3]

2	3	4	4	7
---	---	---	---	---

t=2 A[3]=A[2]

2	3	3	4	7
---	---	---	---	---

t=1 A[2]=A[1]

2	2	3	4	7
---	---	---	---	---

1	2	3	4	7
---	---	---	---	---

Complessità:

$$T(n) = \sum_{k=1}^{n-1} [k + 1] = \sum_{k=1}^{n-1} k + \sum_{k=1}^{n-1} 1 = \frac{(n-1)n}{2} + (n-1) = \frac{n^2}{2} - \frac{n}{2} + n - 1 = \frac{n^2}{2} - \frac{n}{2} - 1$$

$$T(n) = \Theta(n^2)$$

Bubble sort

Gli elementi grandi vanno verso il fondo del vettore

(1)

7	2	4	5	3	1
---	---	---	---	---	---

(3)

2	4	3	1	5	7
---	---	---	---	---	---

2 7
4 7
5 7
3 7
1 7

2 4
3 4
1 4

2 3
1 3

(2)

2	4	5	3	1	7
---	---	---	---	---	---

(4)

2	3	1	4	5	7
---	---	---	---	---	---

2 4
4 5
3 5
1 5

(5)

2	1	3	4	5	7
---	---	---	---	---	---

1 2

1 2

2	4	3	1	5	7
---	---	---	---	---	---

1	2	3	4	5	7
---	---	---	---	---	---

Algoritmo bubbleSort(array A)

For i=0 to n-1

 For j=2 to (n-i+1)

 if(A[j-1]>A[j]) scambia A[j-1] e A[j]

7	2	4	3	1
---	---	---	---	---

n=5

i=1/4

i=1

j=2/5

2	7	4	3	1
---	---	---	---	---

j=3

2	4	7	3	1
---	---	---	---	---

j=4

2	4	3	7	1
---	---	---	---	---

j=5

2	4	3	1	7
---	---	---	---	---

i=2

j=2/4

j=2

j=3

2	3	4	1	7
---	---	---	---	---

j=4

2	3	1	4	7
---	---	---	---	---

i=3

j=2/3

j=2

j=3

2	1	3	4	7
---	---	---	---	---

i=4

j=2/2

j=2

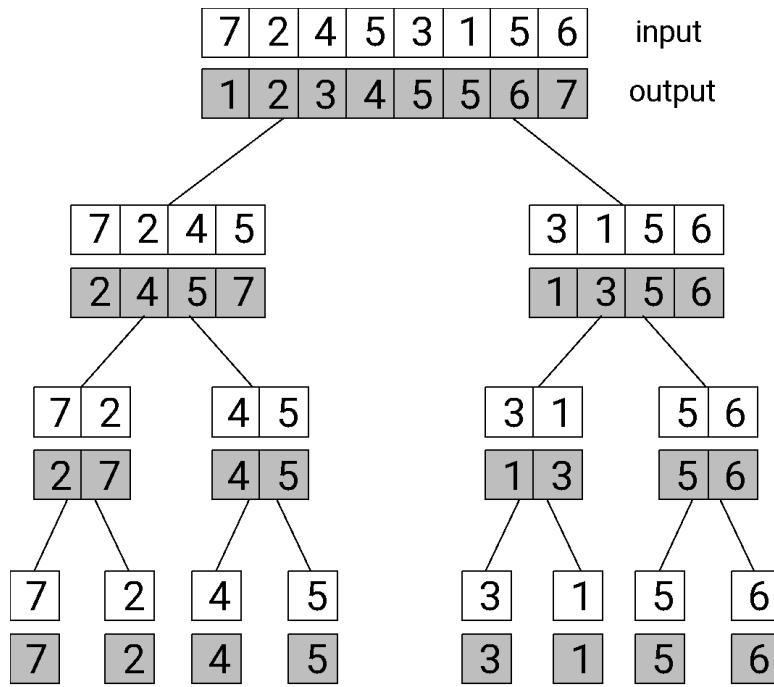
1	2	3	4	7
---	---	---	---	---

Complessità:

$$\begin{aligned} T(n) &= \sum_{i=1}^{n-1} [(n - i + 1) - 2 + 1] = \sum_{i=1}^{n-1} [n - i + 1 - 2 + 1] = \sum_{i=1}^{n-1} (n - i) = \sum_{i=1}^{n-1} n - \sum_{i=1}^{n-1} i = \\ &= n(n - 1) - \frac{(n-1)n}{2} = \frac{(n-1)n}{2} = \frac{n^2}{2} - \frac{n}{2} \\ T(n) &= \Theta(n^2) \end{aligned}$$

ORDINAMENTI (DIVIDE ET IMPERA)

Merge sort



algoritmo mergeSort(array A, indici i e f)

```

if(i>=f) return
m=(i+f)/2
mergeSort(A, i, m)
mergeSort(A, m+1, f)
merge(A, i, m, f)

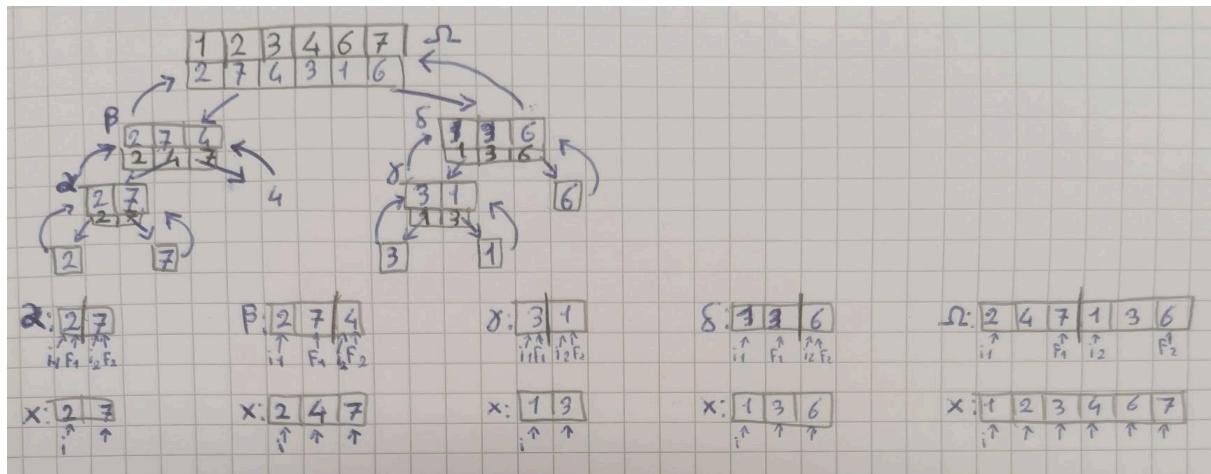
```

algoritmo merge(array A, interi i1, f1, f2)

```

sia x in array di lunghezza f2-i1+1
i=1 oldi1=i1 i2=f1+1
while (i1<=f1 and i2<=f2)
    if(A[i1]<=A[i2]) x[i]=A[i1]; i++; i1++
    else    x[i]=A[i2]; i++; i2++
if(i1<f1) copia A[i1:f1] in fondo a x
else copia A[i2:f2] in fondo a x
copia x in A [oldi1:f2]

```



La merge costa $n-1$

$$T(0) = T(1) = 0$$

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

$$f(n) = n - 1 \quad a = 2 \quad b = 2$$

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

Secondo caso del teorema

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

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

Quick sort

45	12	93	3	67	43	85	29	24	92	63	3	21
			↑								↑	

45	12	21	3	67	43	85	29	24	92	63	3	93
			↑								↑	

45	12	21	3	3	43	85	29	24	92	63	67	93
			↑			↑						

algoritmo quickSort(array A, indici i e f)

```

if(i>=f) return
m=partition(A, i, f)
quickSort(A, i, m-1)
quickSort(A, m-1, f)

```

procedura partition(array A, indici i e f)→indici

```

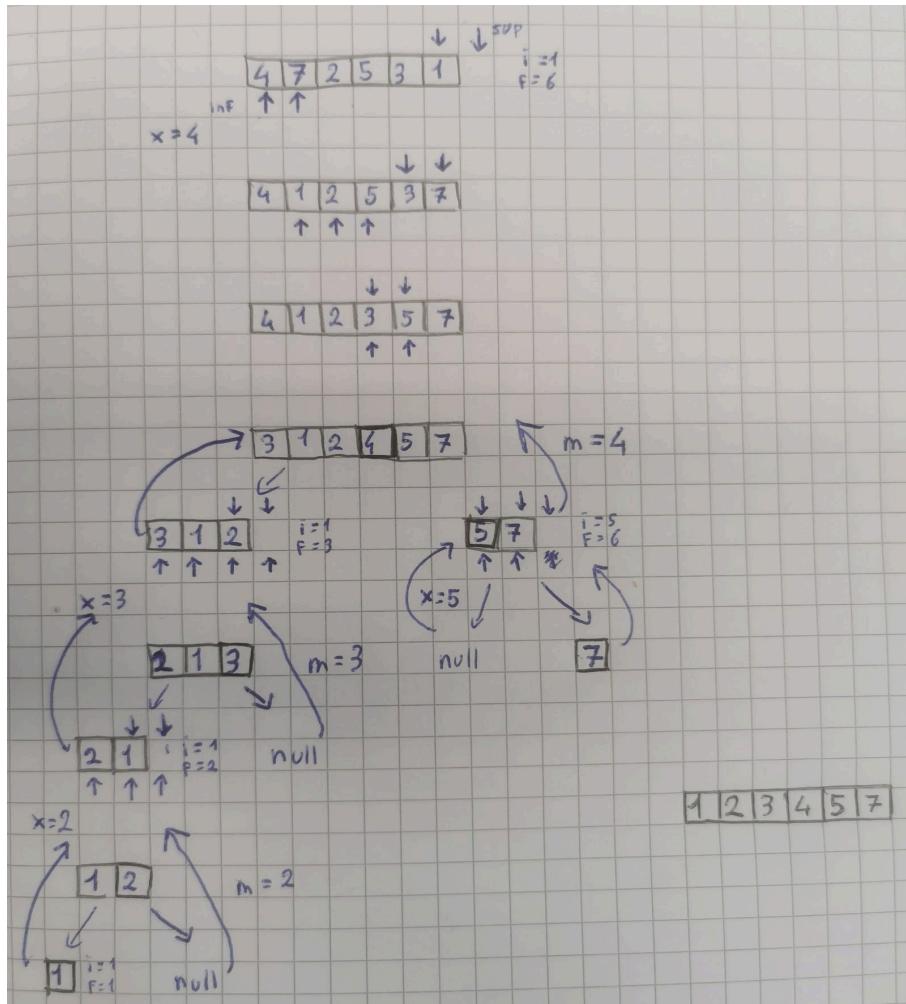
x=A[i]
inf=i

```

```

sup=f+1
while(true)
    do inf=inf+1 while(inf<=f and A[inf]<=x)
    do sup=sup-1 while(A[sup]>x)
    if(inf<sup) scambia A[inf] e A[sup]
    else break
scambia A[i] e A[sup]
return sup

```



Numero di confronti nella partition nel **caso migliore** (il perno sempre nel punto medio).

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

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

Numero di confronti nella partition nel **caso peggiore** (il perno è sempre ad un'estremità).

[DOMANDA D'ESAME]

$$T(n) = n - 1 + 0 + T(n - 1) = n - 1 + T(n - 1)$$

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

$$T(n - 1) = n - 3 + T(n - 3)$$

$$T(n) = (n - 1) + (n - 2) + (n - 3) + T(n - 3) =$$

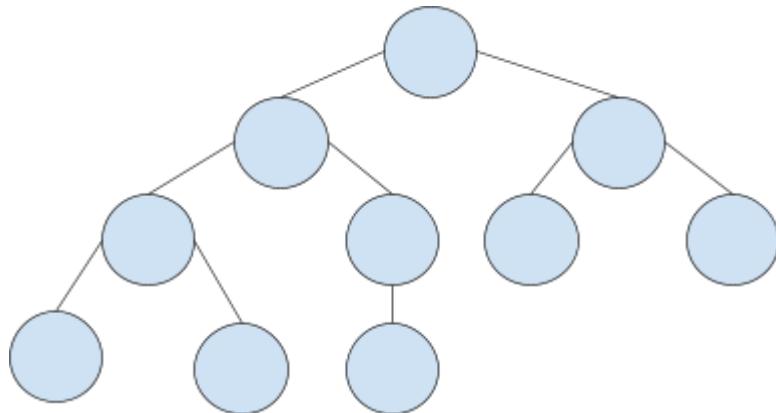
$$\begin{aligned}
 &= \sum_{j=1}^3 (n-j) + T(n-3) = \sum_{j=1}^i (n-j) + T(n-i) = \sum_{j=1}^{n-1} (n-j) + T(1) = \sum_{j=1}^{n-1} (n-j) = \\
 &= \sum_{j=1}^{n-1} n - \sum_{j=1}^{n-1} j = n(n-1) - \frac{(n-1)n}{2} = \frac{(n-1)n}{2} = \frac{n^2}{2} - \frac{n}{2} \\
 T(n) &= \Theta(n^2)
 \end{aligned}$$

Numero di confronti nella partition nel **caso medio**. [NON RICHIESTO].

Heap sort

L'heap è un albero binario:

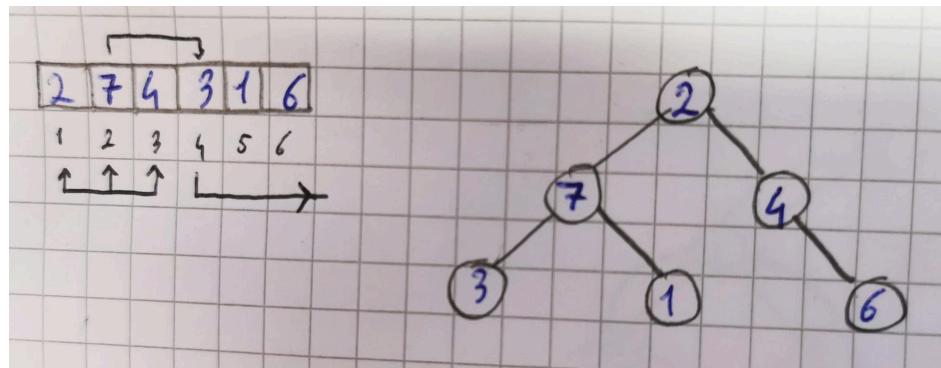
- completo fino al penultimo livello
- struttura rafforzata
- chiave (padre) \geq chiave (figli)



Schiacciato verso sinistra

$i \rightarrow 2i$

$| \rightarrow 2i+1$



Pre-processing:

heapify (heap H)

```

if(H è vuoto) return
heapify(sottoalbero sx di H)
heapify(sottoalbero dx di H)
fixheap(radice di H, H)
  
```

fixheap(nodo v, heap H)

```

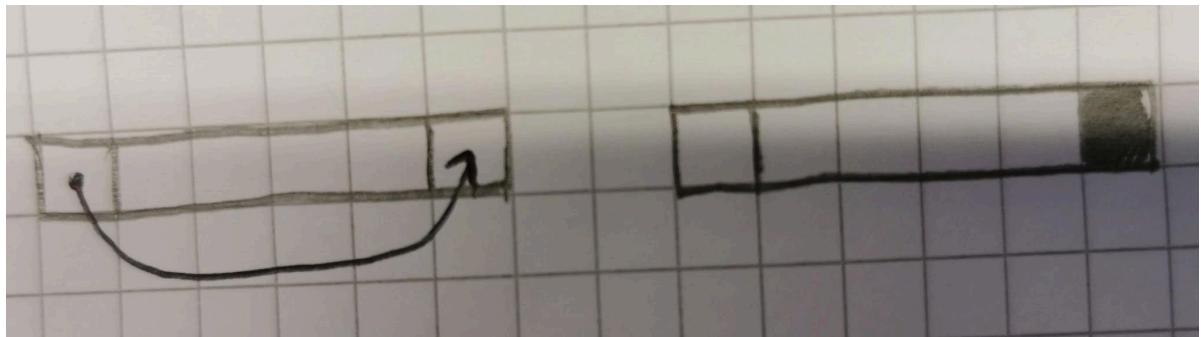
if(v è foglia) return
sia u il figlio di v di chiave max
if(chiave(v)<chiave(u))
    scambia chiave(v) e chiave(u)
    fixheap(u,H)

```

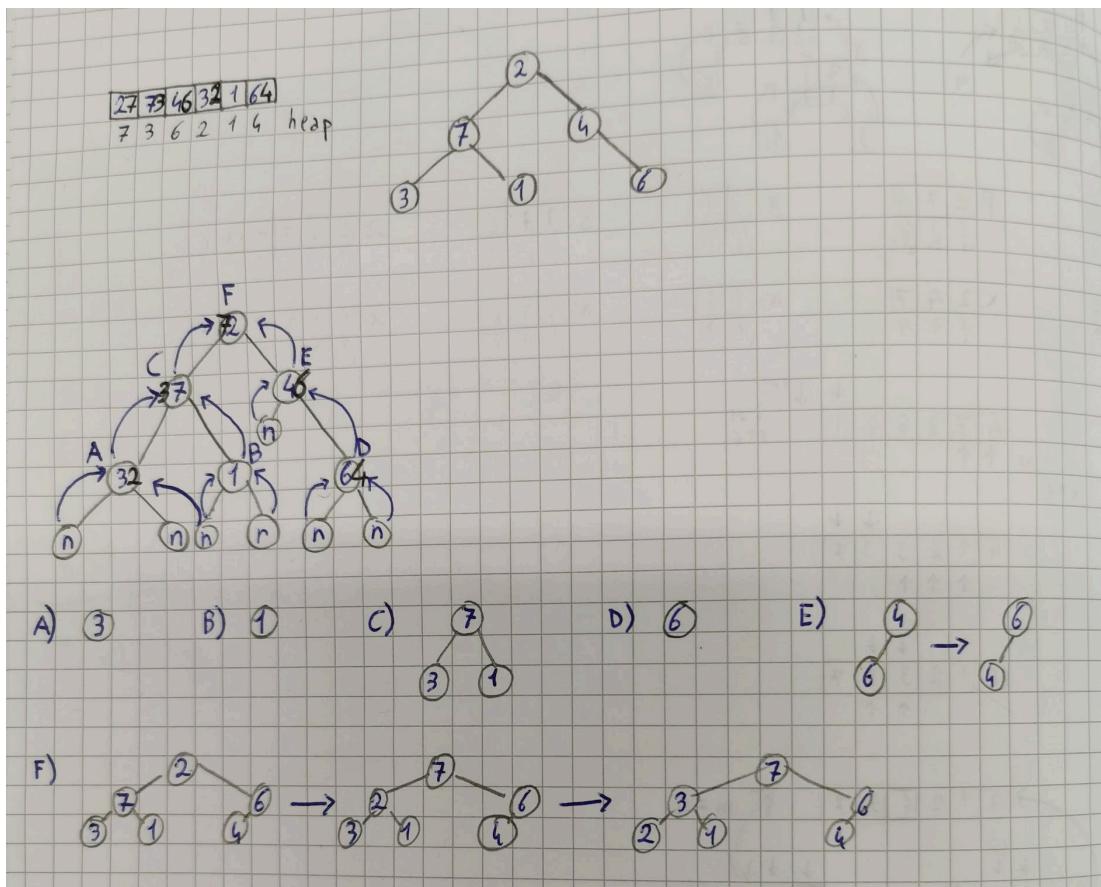
Sorting

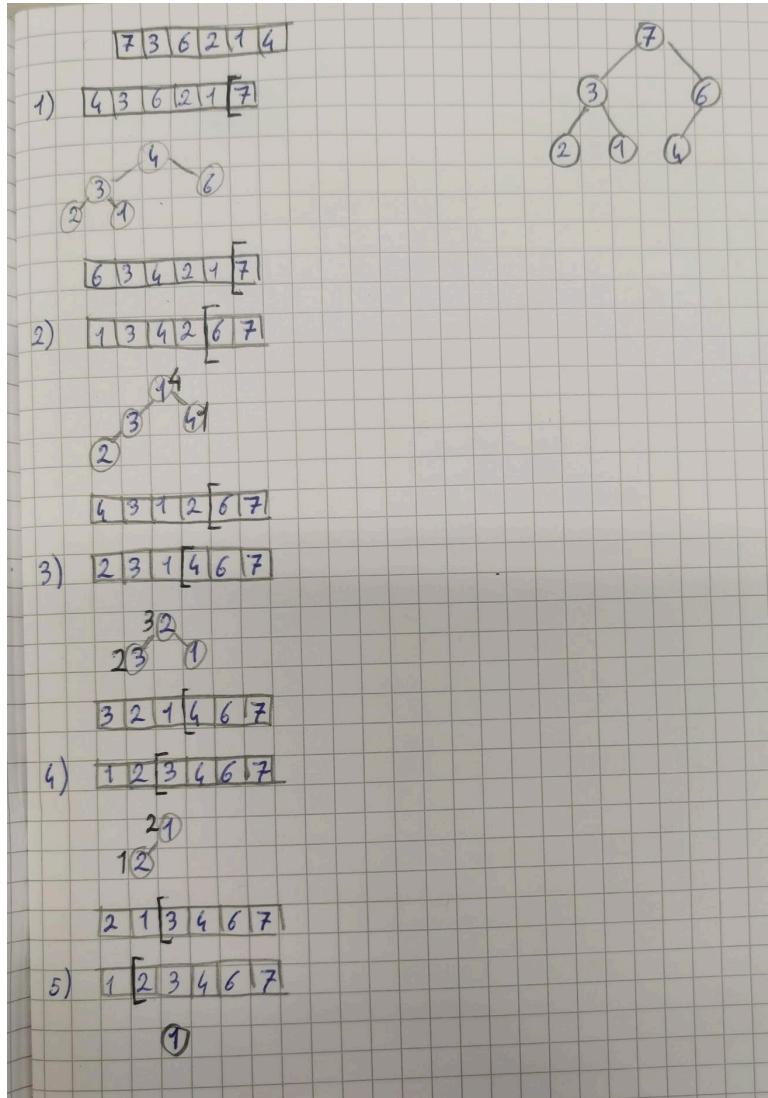
n-1 volte:

- estrazione del max
- fixheap sulla nuova radice



Nella coda avrò sempre il massimo. Man mano riduco la dimensione del vettore da controllare.





Caso peggiore fa $\Theta(h)$ confronti ($h = \text{altezza dell'albero}$).

[ESAME]

Il numero di nodi dell'heap è compreso tra il numero di nodi di un albero di altezza $h-1$ e uno di altezza h

$$\sum_{i=0}^{h-1} 2^i \leq n \leq \sum_{i=0}^h 2^i$$

$$\frac{2^h - 1}{2-1} \leq n \leq \frac{2^{h+1} - 1}{2-1}$$

$$2^h - 1 \leq n \quad h \leq \log(n + 1) \quad h = O(\log n)$$

$$n + 1 \leq 2^{h+1} \quad \log(n + 1) \leq h + 1 \quad h \geq \log(n + 1) - 1 \quad h = \Omega(\log n)$$

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

$$\text{fixheap} = \Theta(\log n)$$

heapify:

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

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

$$f(n) = O(n^{1-\varepsilon}) \quad \varepsilon = \frac{1}{2}$$

Primo caso del teorema

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

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

Integer sort

Gli elementi devono avere un valore minore della lunghezza del vettore.

3	1	5	2	5
---	---	---	---	---

algoritmo integerSort(intero n, array X)

sia Y un array di dimensione n

for i=1 to n se Y[i]=0

for i=1 to n Y[X[i]]++

j=1

for i=1 to n

 while(Y[i]>0)

 X[j]>i

 j++

 Y[i]--

Utilizzo il vettore Y come vettore di contatori.

Y	1	2	3	4	5
	0	0	0	0	0
	1	1	1	0	2

X	3	1	5	2	5
	↑	↑	↑	↑	↑

2) Y	10	10	10	0	2	10
	↑	↑	↑	↑	↑	↑
X	1	2	3	5	5	
	↑	↑	↑	↑	↑	

Incrementi = $\Theta(n)$

Decrementi = $\Theta(n)$

IntegerSort = $\Theta(n)$

Caso in cui i valori sono maggiori della lunghezza del vettore.

algoritmo integerSort(intero n, array X)

sia Y un array di dimensione n

for i=1b to k Y[i]=0

for i=1 to n Y[X[i]]++

j=1

for i=1 to k

 while(Y[i]>0)

 X[j]>i

 j++

 Y[i]--

Incrementi= $\Theta(n)$

Decrementi= $\Theta(k + n)$

Se $k > n$ non conviene utilizzare l'integerSort perché la dimensione del vettore diventerebbe eccessivamente grande.

Radix sort

Variazione dell'integerSort.

713	318	721
-----	-----	-----

n=3 k=721

algoritmo radixSort(array A di n interi)

t=0

 while(esiste un numero che ha la t-esima cifra)

 bucketSort(A,10, t)

 t=t+1

procedura bucketSort(array A, interi b e t)

 sia Y un array di dimensione b

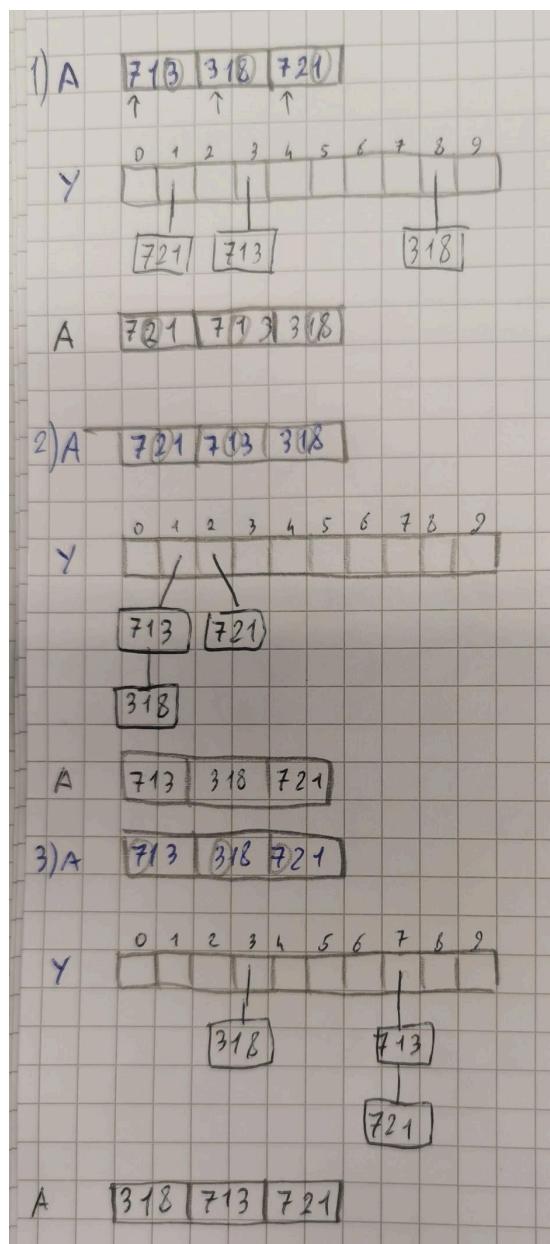
 for i=0 to b-1 Y[i]=lista vuota

 c=t-esima cifra di A[i]

 append A[i] alla lista Y[c]

 for i=0 to b-1

 copia ordinatamente in A gli elementi di Y[i]



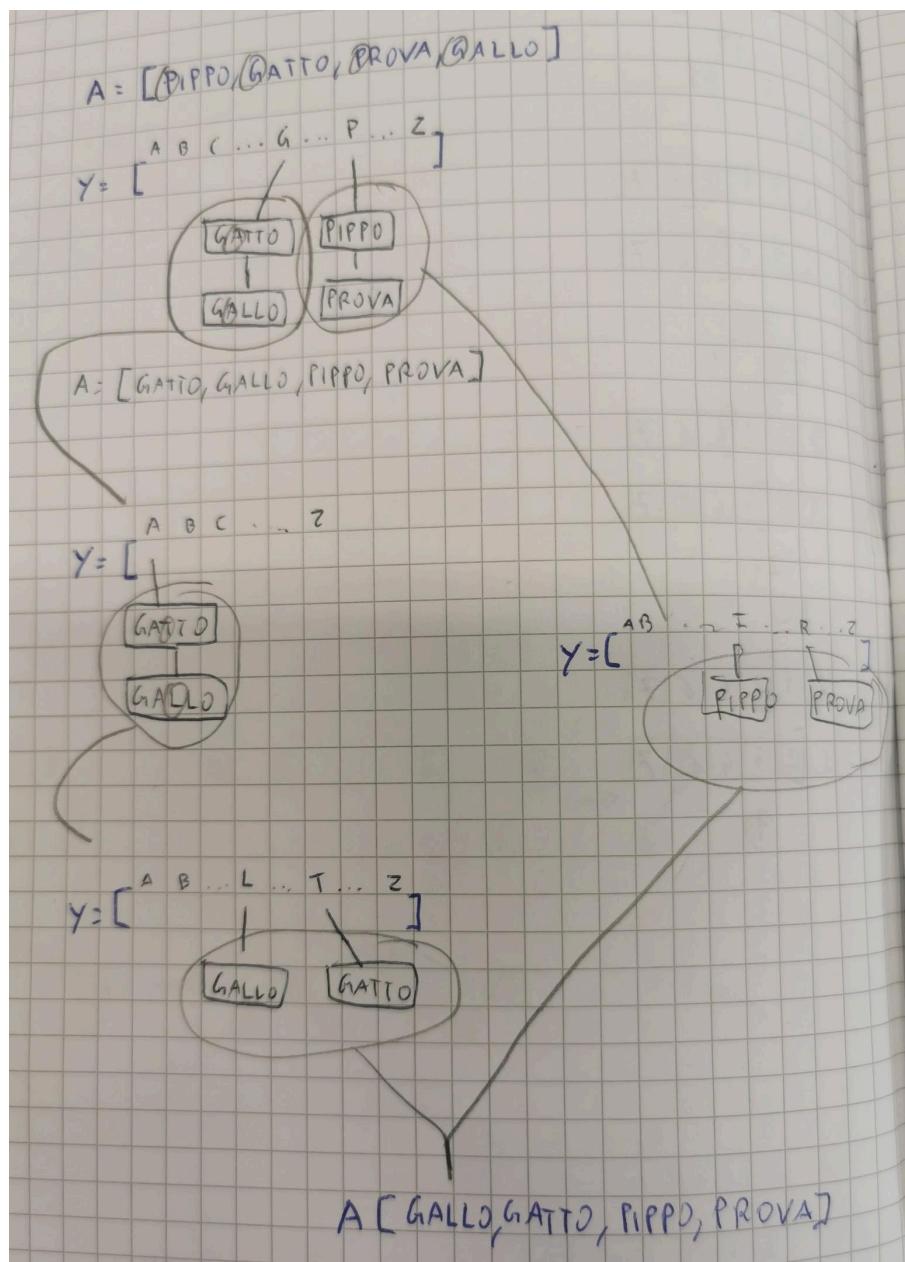
Algoritmo stabile

LEAST SIGNIFICANT DIGIT

$O((n + 10) \log_{10} k) \leftarrow$ costo radixSort

↑costo bucketSort

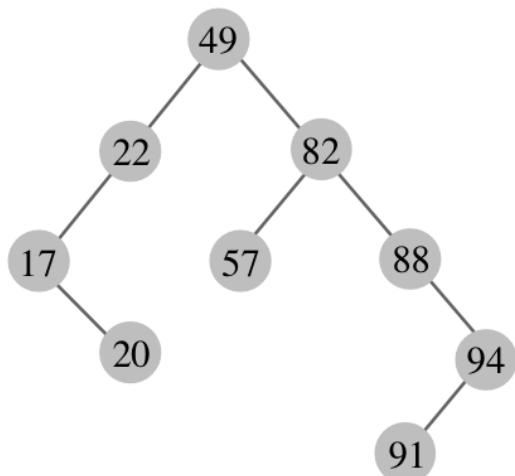
Il radixSort partendo dalla cifra più significativa può essere implementato anche nelle stringhe.



RICERCA cap6

Dizionario <elemento, chiave>

Albero binario di ricerca è un albero in cui ogni nodo ha al massimo due figli, ogni **sottoalbero sinistro ha valori minori** del valore corrente, invece nel **sottoalbero destro troverò solo valori maggiori** del valore corrente.



```

visita_simmetrica (nodo r)
    if(r=null) return n
    visitaRicorsiva(figlio sinistro di r)      17 20 22 49 57 82 88 91 94
    Visita r
    visitaRicorsiva(figlio destro di r)

```

Nodi stampati dal più piccolo al più grande.

Ricerca

```

algoritmo_search(chiave k)
    r=radice
    while(r!=null)
        if(k==chiave(r)) return elem(r)
        else if (k<chiave(r)) r=figlio sx di r
        else r=figlio dx di r
    return null

```

Costo= $O(h)$

L'albero potrebbe degenerare in un albero sbilanciato su un lato e il costo diventerebbe $O(n)$

Inserimento

Prima faccio la ricerca e una volta trovato il punto in cui inserire creo il nodo nuovo.
L'inserimento costa come la ricerca perché in più fa solo una malloc e sistema un puntatore.

Cancellazione

La cancellazione di una foglia è una search più una free.

Se ha un figlio solo collego direttamente il figlio con il nonno e poi faccio la free del nodo che devo eliminare.

Se voglio cancellare un nodo che ha due figli, devo trovare un valore da sovrascrivere al nodo da eliminare, un buon concorrente è il predecessore (57 predecessore di 82), devo trovare il massimo del sottoalbero sinistro quindi mi sposto a sinistra e proseguo finché posso verso destra. [Il predecessore al massimo ha una foglia]

```

algoritmo pred(nodo u) → nodo
    if(u ha figlio sx sin(u))
        return max(sin(u))
    while(parent(u) != null and u = figlio sx di suo padre)
        u=parent(u)
    return parent(u)

```

```

algoritmo_max(nodo u) → nodo
    while(figlio dx di u != null)
        u=figlio dx di u
    return u

```

Costo=O(h)

Albero degenerato=O(n)

ALBERI AVL

Introduciamo il fattore di bilanciamento di $v = |h \text{ sottoalbero sx} - h \text{ sottoalbero dx}|$

Un albero è AVL se per ogni nodo $v \leq 1$

[ESAME]

Albero AVL - Albero di Fibonacci (albero AVL più sbilanciato, $v=1$)

Altezza = h

Numero di nodi = $n_h = n_{h-1} + n_{h-2} + 1$

$$n_h = F_{h+3} - 1 \quad (\text{F successione di Fibonacci})$$

$$F_h = \Theta(\Phi^h)$$

Dimostrare questa proprietà dimostra che l'altezza dell'albero AVL è logaritmica.

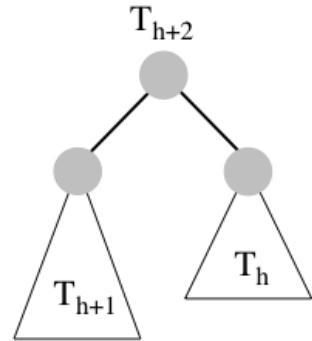
Dimostrazione (induzione)

$$0) 1 = F_{0+3} - 1 = F_3 - 1 = 2 - 1$$

$$n_{h-1} = F_{h-1+3} - 1 = F_{h+2} - 1 \quad \text{verificato per ipotesi induttiva}$$

$$n_{h-2} = F_{h-2+3} - 1 = F_{h+1} - 1 \quad \text{verificato per ipotesi induttiva}$$

$$n_h = F_{h+2} - 1 + F_{h+1} - 1 + 1 = F_{h+3} - 1 \quad \text{VERIFICATO}$$



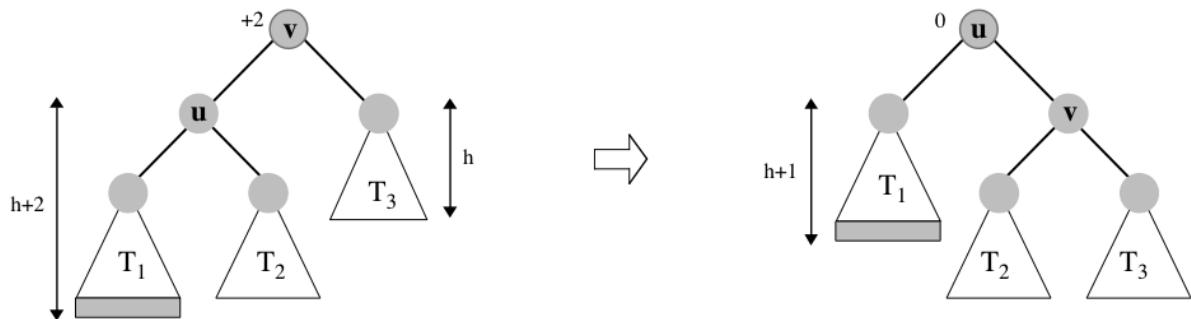
Rotazioni

Con la insert e la delete rischio di uscire dalla condizione di albero AVL perché potrei sbilanciare un nodo; in quel caso intervengo con delle operazioni, le rotazioni, per tornare in un albero AVL.

Le rotazioni hanno un **costo costante**, di conseguenza non peggiorano la complessità degli algoritmi.

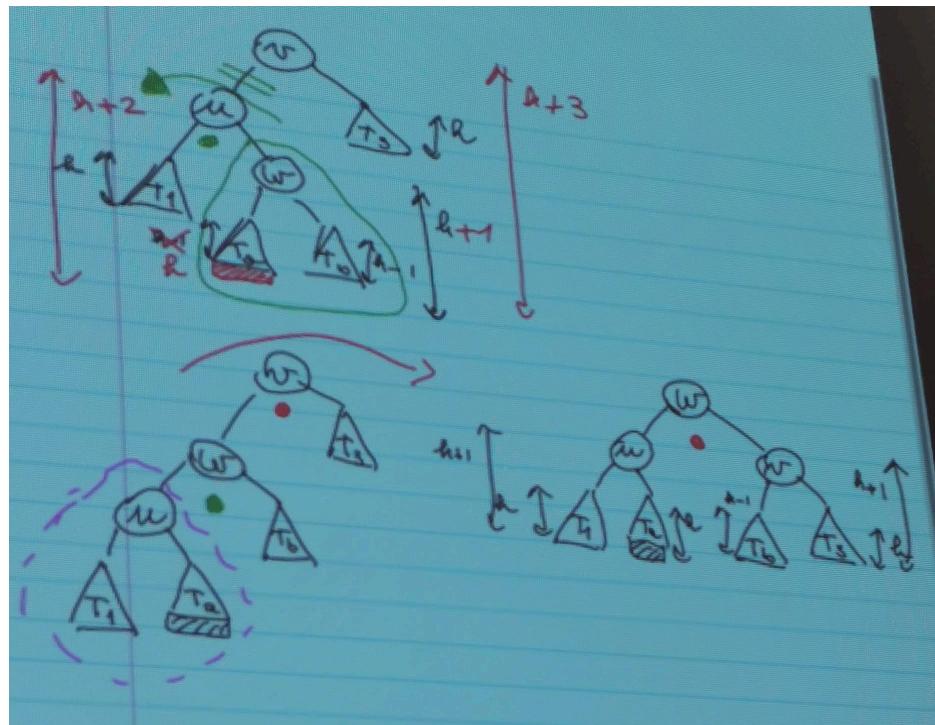
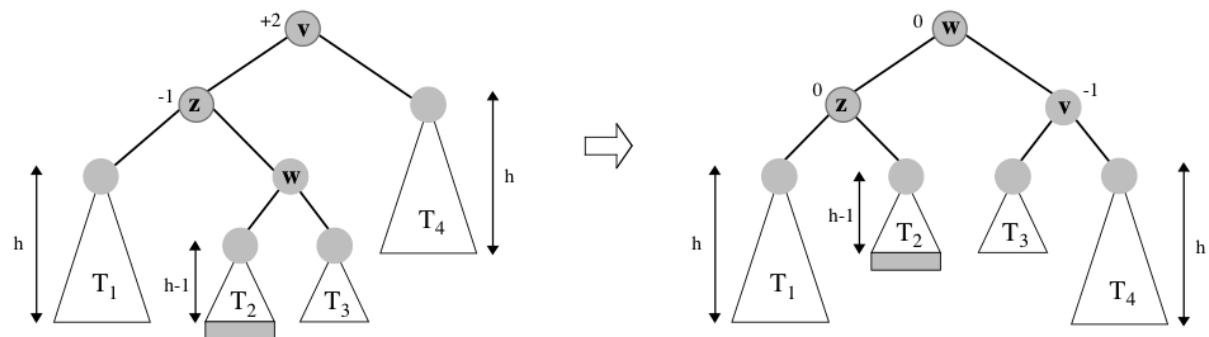
Solo nella cancellazione potrebbe verificarsi un problema in cascata perché potrebbe causare modifiche dell'albero fino alla radice.

Rotazione semplice:

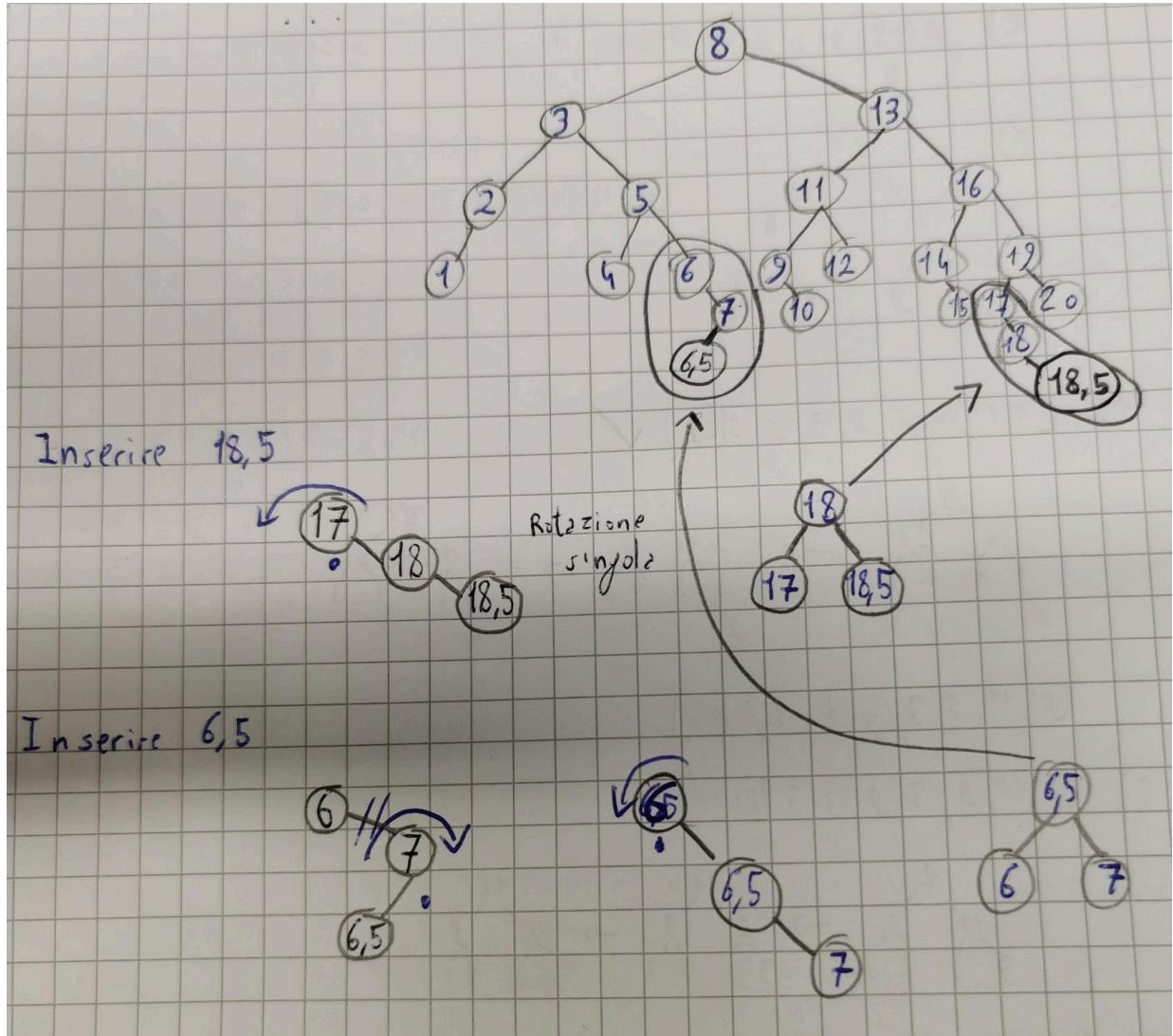


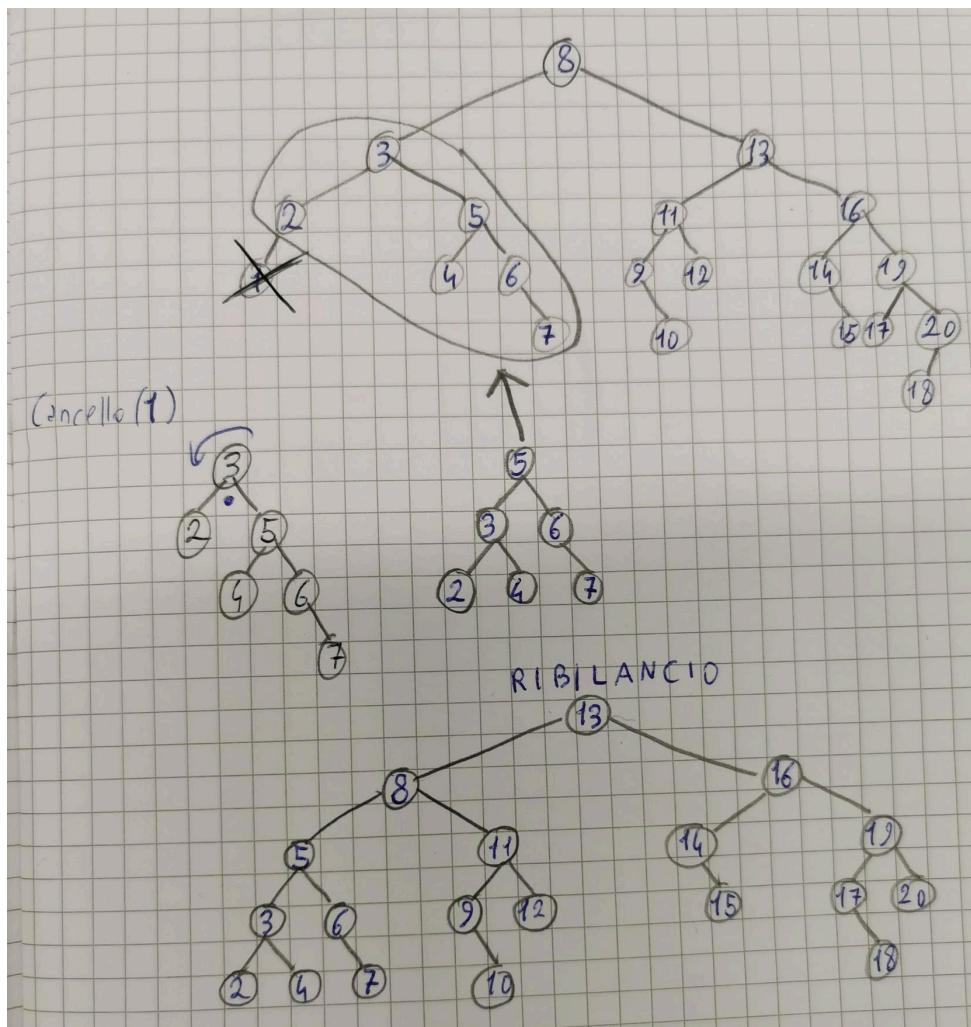
Rotazione doppia:

Stacchiamo "z" e "v" e lavoriamo nello stesso modo della rotazione semplice con "z" e "w", poi lavoro sull'albero completo.

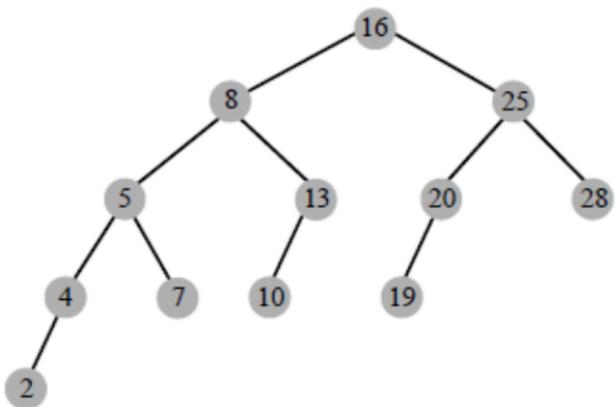


Prima simulare la ricerca per capire dove inserire il nuovo nodo.



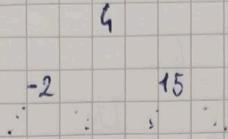


Ex 2. Dato il seguente albero AVL, mostrare le modifiche che subisce dopo gli inserimenti di 1, 3, 12, 15.



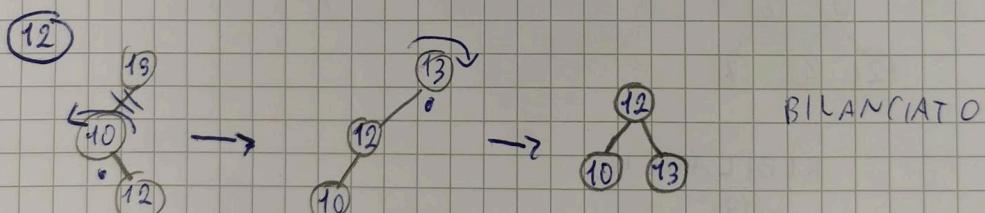
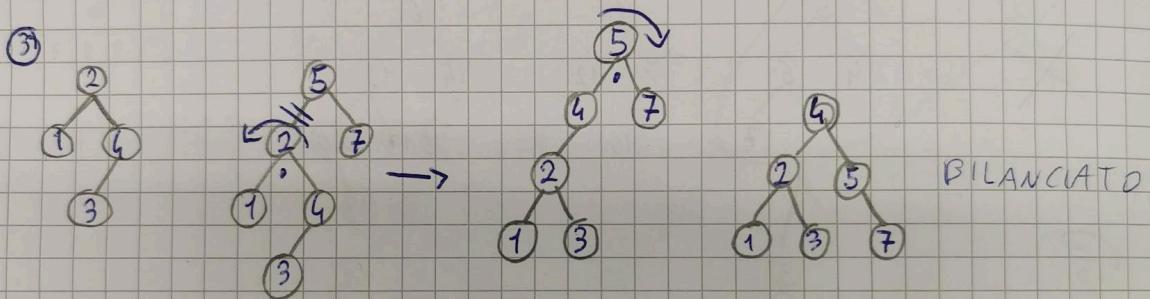
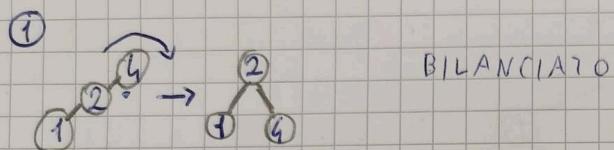
Ex 1
-8, -4, -2, 0, 1, 3, 4, 9, 12, 15, 16, 17, 21

Itero sul valore medio

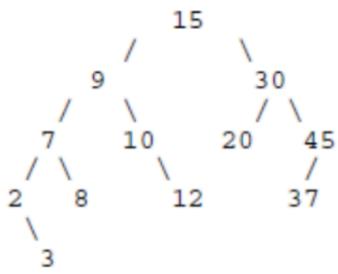


Ex 2

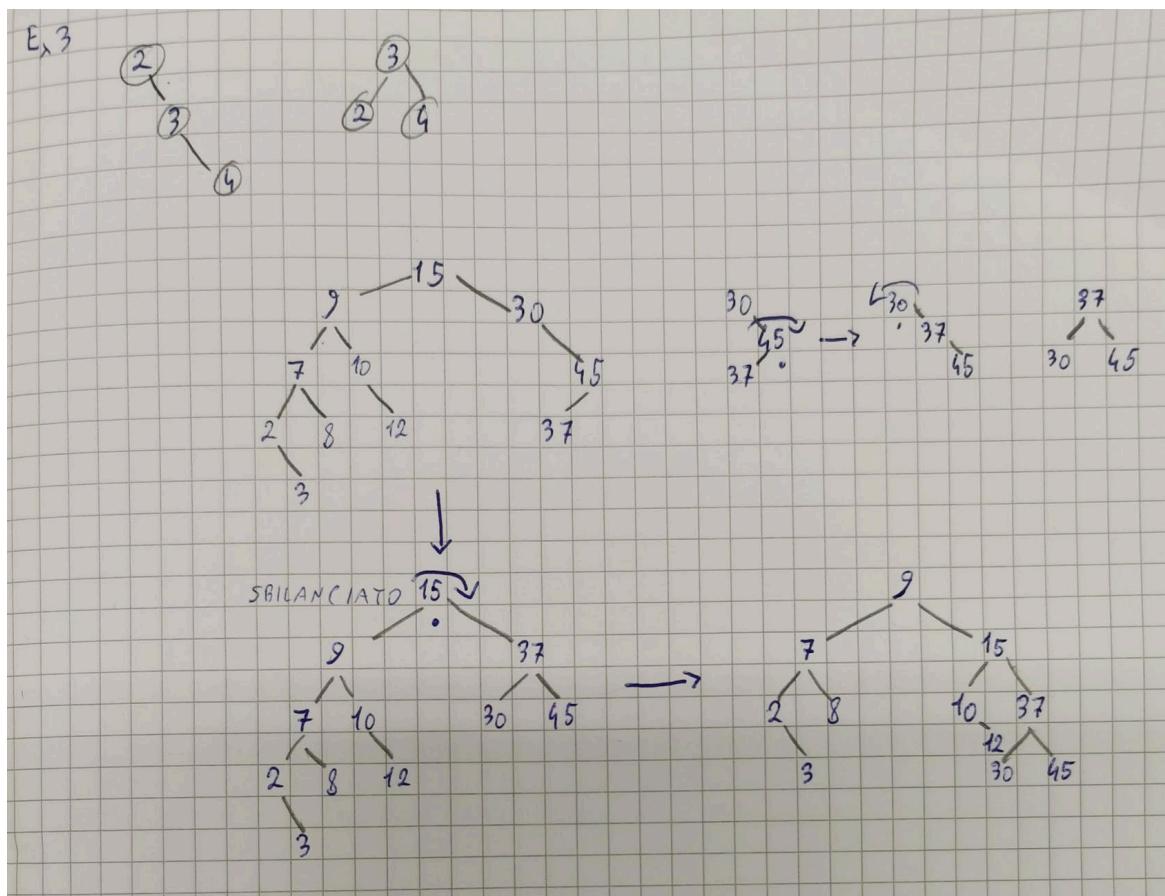
Insert: 1, 3, 12, 15



Ex 3. Dato il seguente albero AVL



si considerino l'inserimento della chiave 4 seguito dalla cancellazione della chiave 20 e si disegni l'albero risultante.



ALBERI CON BILANCIAMENTO PERFETTO (ALBERI 2-3)

Bilanciamento perfetto ma il numero dei figli può essere 2 o 3.

Alla base avremo le chiavi, se il padre ha 2 figli avrà come chiave il massimo del figlio destro, se ha tre figli avrà il massimo del figlio sinistro e il massimo del figlio centrale.

Algoritmo search (nodo r, chiave x) → elemento

```

if(r è una foglia)
    if(x=chiave(r)) return elemento(r)
    else return null
    v(i)=i-esimo figlio di r
  
```

```

if(x<=S[r]) search(v(1), x)
else if(r ha 2 figli o x<=M[r]) search(v(2), x)
else search(v(3), x)

T(h) = c + T(h - 1)
T(h - 1) = c + T(h - 2)
T(h - 2) = c + T(h - 3)
T(h)=c+c+T(h-2)=c+c+c+T(h-3)=3c+T(h-3)=i*c+T(h-i)
h - i = 0      i = h
T(h) = h * c
Stesse prestazioni degli alberi AVL.

```

Dimostrazione (altezza albero 2-3 logaritmica) [ESAME]:

$$1) 2^{h+1} - 1 \leq n \leq \frac{3^{h+1}-1}{2}$$

$$2) 2^h \leq f \leq 3^h$$

Per induzione, dimostrazione 1:

$$0) n = 1 \quad f = 1 \quad h = 0 \quad 2^0 \leq 1 \leq 3^0$$

Albero intero senza lo strato delle foglie: $2^{h-1} \leq F \leq 3^{h-1}$ per induzione.

$$2^h = 2 * 2^{h-1} \leq 2F \leq f \leq 3F \leq 3 * 3^{h-1} = 3^h \quad \text{TESI}$$

Dimostrazione 2:

$$0) n = 1 \quad f = 0 \quad h = 0 \quad 2^0 - 1 \leq 1 \leq \frac{3^0-1}{2}$$

$$2^{h-1} \leq N \leq \frac{3^h-1}{2} \quad \text{ipotesi induttiva}$$

$$2^h - 1 + 2^h \leq n = N + f \leq \frac{3^h-1}{2} + 3^h$$

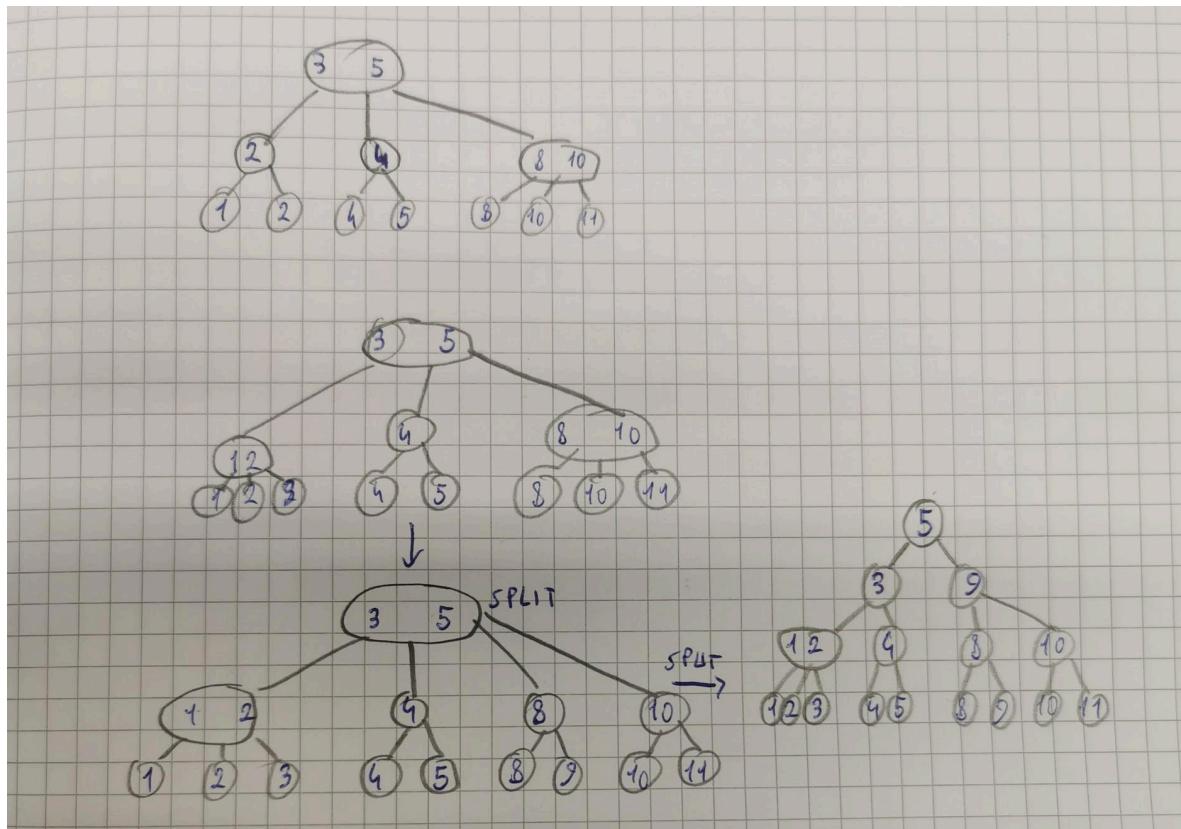
$$2^{h+1} - 1 \leq n \leq \frac{3^{h+1}-1}{2}$$

$$2^{h+1} \leq n + 1 \quad h + 1 \leq \log(n + 1) \quad h \leq \log(n + 1) - 1 \quad h = O(\log n)$$

$$n \leq \frac{3^{h+1}-1}{2} \quad 2n \leq n \leq 3^{h+1} - 1 \quad 2n - 1 \leq 3^{h+1} \quad \log(2n - 1) \leq h + 1$$

$$h < \log(2n - 1) - 1 \quad h = \Omega(\log n)$$

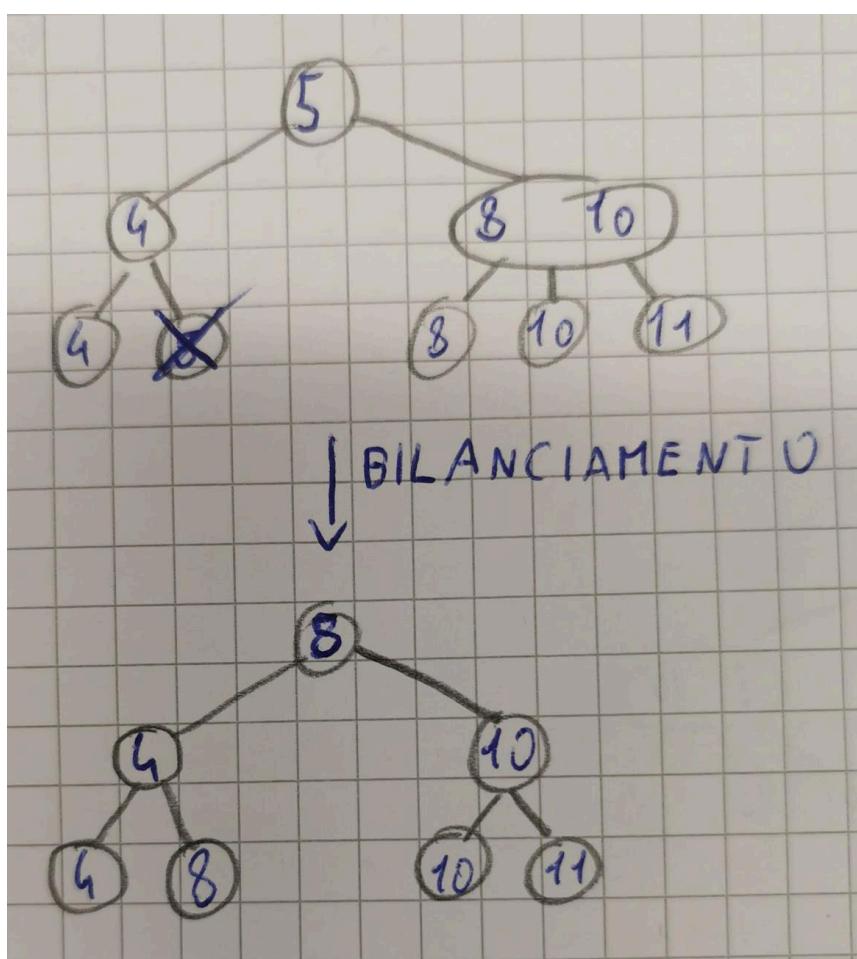
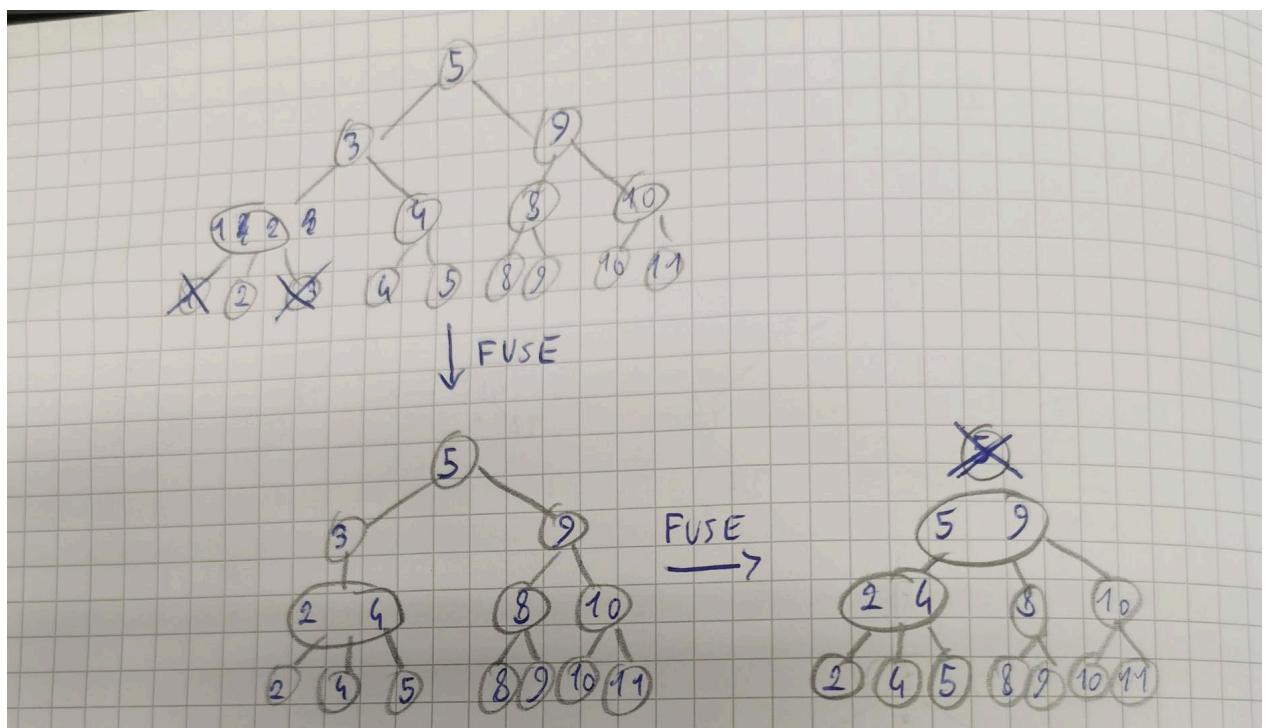
Costo search logaritmico



Algoritmo split(nodo v)

```

creo un nodo w
rendi v(1) e v(2) figli di w
if(parent[v]=null)
    crea un nodo r
    rendi w e v figli di r
else
    aggiungi w come figlio di parent[v]
    if(parent[v] ha 4 figli) split(parent[v])
  
```



B-ALBERO

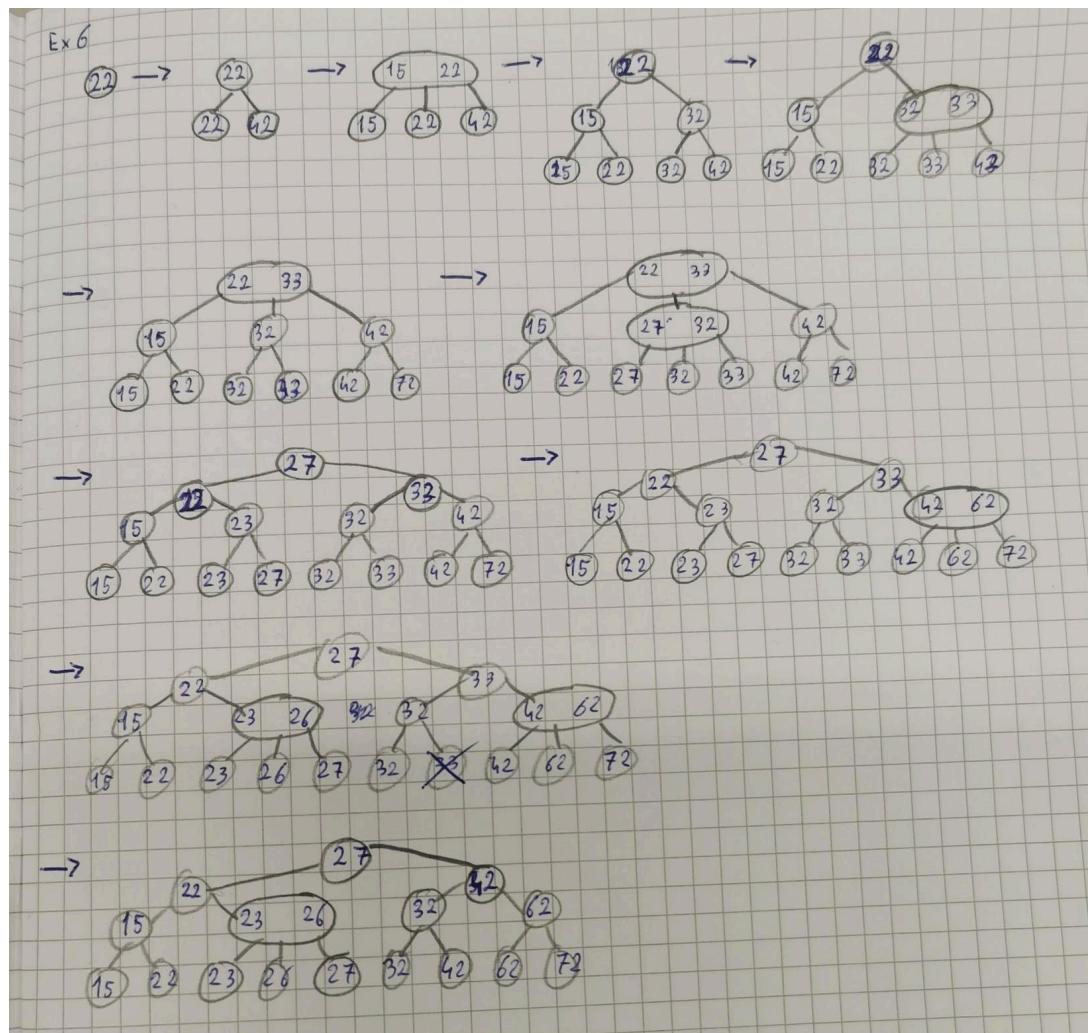
Struttura dati usata per indicizzare le tabelle dei data-base.

Ogni nodo contiene T o 2T figli (2T blocco del disco), devo usare almeno metà blocco.

Contiene il numero massimo di accessi al disco che devo compiere.

Ex. 6. Modificare progressivamente un albero 2-3 in base alle seguenti istruzioni (dove I significa “inserire” e C “cancellare”):

I22, I42, I15, I32, I33, I72, I27, I23, I62, I26, C33



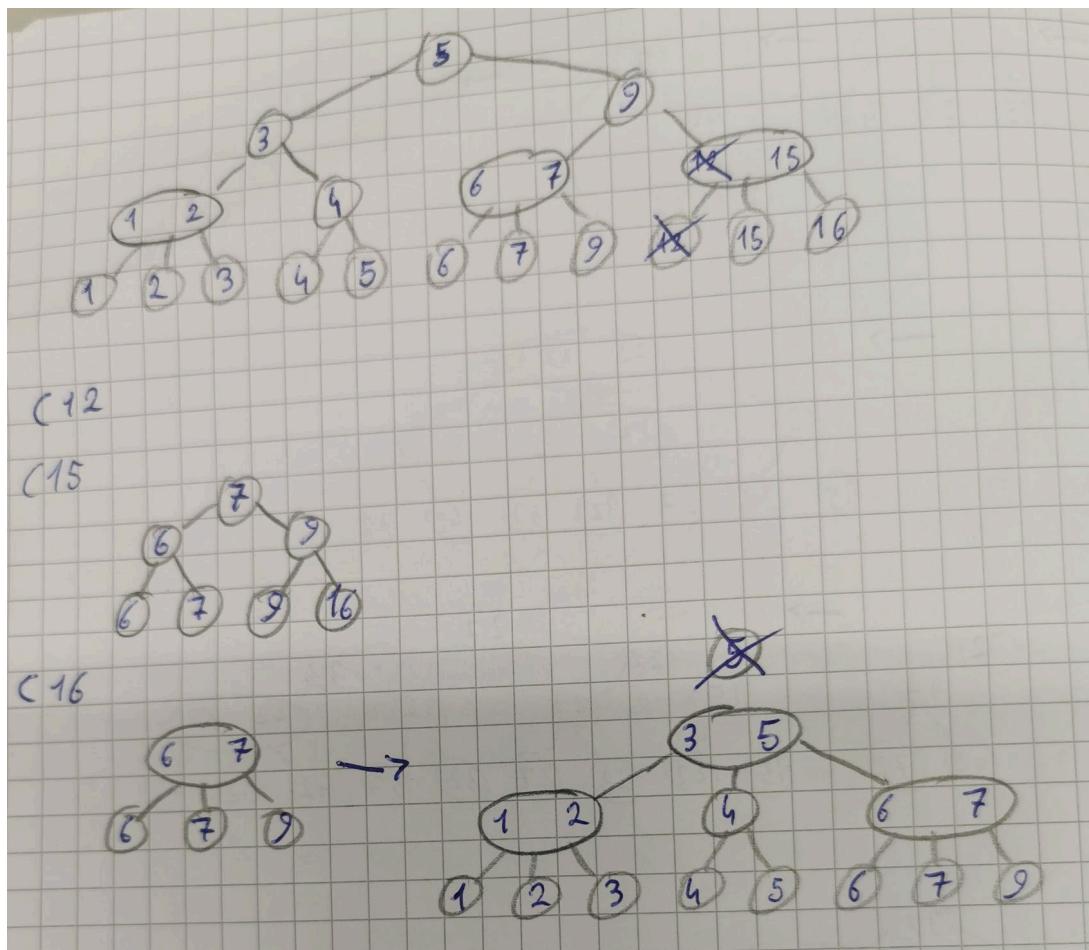


TABELLE HASH

Dizionario con tutte le operazioni $\Theta(1)$

Tabella ad accesso diretto: **array di interi** dove memorizziamo i dati all'indice corrispondente (ex. aggiungo il dato 3 in posizione 3).

Fattore di carico $\alpha = \frac{n}{m}$ [ex. n=numeri di studenti (100), m=numeri di matricola (10^6)]

Spreco molta memoria.

Esempio

234717-235717 (numeri di matricola) $n=100$

Invece che fare valori di chiave = indice applico una funzione per risparmiare spazio.

$$h(k)=k-234717$$

Funzione hash perfetta, ad ogni indice corrisponde una chiave, il costo è ancora $\Theta(1)$. Così facendo non ho più una tabella ad accesso diretto, perché ho fatto un calcolo sulla chiave, ma una **tabella hash**.

$\alpha = \frac{100}{1000}$ anche così ho una tabella sproporzionata.

Se permetto che ci possano essere dei valori diversi della chiave portano a celle uguali posso sfruttare molto meglio la memoria, nel momento di una collisione devo gestirla e sprecare del tempo. Lo spreco di tempo è comunque minore dello spreco di memoria.

Uniformità: - - -

TECNICHE PER GESTIRE LE COLLISIONI

- **Liste di collisione:**

- Tutti i valori che collidono in una cella gli appendo ad una lista che ha la testa nella cella.
- Cercare di dividere gli elementi in modo da avere liste di dimensioni simili.

BAMBOLA $m=11$

$$h(k) = \text{ASCII}(k) \% m$$

$A = 65$
$B = 66$
$L = 76$
$M = 77$
$O = 79$

BAMBOLA $m=11$

$h(k) = \text{ASCII}(k) \% m$

1	2	3	4	5	6	7	8	9	10	11
0	i	2	3	4	5	6	7	8	9	10
B	M		O						A	
									L	
									A	

$h(B) = 66 \% 11 = 0$

$h(A) = 65 \% 11 = 10$

$h(M) = 77 \% 11 = 0$

$h(O) = 79 \% 11 = 2$

$h(L) = 76 \% 11 = 10$

insert: $\Theta(1)$

search: $\Theta(1 + \frac{n}{m})$

spazio: $\Theta(m + n)$

- **Metodi di indirizzamento aperto (legge di scansione)**

- Ci dice dove mettere il dato se la sua cella è occupata.
- Sia in inserimento che in lettura.
- Scansione lineare o hashing doppio

$$c(k,i) = (h(k) + i) \% m$$

Scansione lineare
i=passo (all'inizio 0)

```
insert(elem e, chiave k)
    for(i=0 to m-1)
        if(v[c(k,i)].elem =(null or canc))
            v[c(k,i)]=(e, k)
        return
```

errore tabella piena

```

search(chiave k) → elem
    for i=0 to m-1
        if(v[c(k,i)].elem = null)
            return null
        if(v[c(k,i)].chiave = k and v[c(k,i)].elem != canc)
            return v[c(k,i)].elem
    return null

```

```

delete (chiave k)
    for i=0 to m-1
        if(v[c(k,i)].elem = null)
            chiave non in dizionario
        if(v[c(k,i)].chiave = k and v[c(k,i)].elem != canc)
            v[c(k,i)].elem = canc
            chiave non in dizionario

```

BAMBOOLA											m = 11
0	1	2	3	4	5	6	7	8	9	10	
B	M	B	C	L	A					A	

$c(B, 0) = (h(B) + 0) \% 11 = 0$
 $c(A, 0) = (h(A) + 0) \% 11 = 10$
 $c(M, 0) = (h(M) + 0) \% 11 = 0$
 $c(L, 1) = (h(L) + 1) \% 11 = 1$
 $c(C, 0) = 0$
 $c(B, 1) = (h(B) + 1) \% 11 = 1$
 $c(B, 2) = (h(B) + 2) \% 11 = 2$
 $c(C, 0) = (h(C) + 0) \% 11 = 2$
 $c(C, 1) = (h(C) + 1) \% 11 = 3$
 $c(L, 0) = (h(L) + 0) \% 11 = 10$
 $c(L, 1) = (h(L) + 1) \% 11 = 0$
 $c(L, 2) = 1$
 $c(L, 3) = 2$
 $c(L, 4) = 3$
 $c(L, 5) = 4$
 $c(A, 0) = 10$
 $c(A, 1) = 1$
 $c(A, 2) = 2$
 $c(A, 3) = 3$
 $c(A, 4) = 4$
 $c(A, 5) = 5$
 $c(A, 6) = 6$

AGGLOMERAZIONE

$$c(k,i) = (h(k) + i * h_2(k)) \% m \quad \text{Scansione doppia}$$

B A M B O L A	m = 11
$h(k) = ASC(11(k)) \% 11$	
$h_2(k) = ASC(11(k)) \% 10 + 1$	
0 1 2 3 4 5 6 7 8 9 10	
B C A L B M A	
$c(B, 0) = (h(B) + 0) \% 11 = 0$	
$c(A, 0) = (h(A) + 0) \% 11 = 10$	
$c(M, 0) = (h(M) + 0) \% 11 = 0$	
$c(M, 1) = (h(M) + 1 * h_2(M)) \% 11 = (0 + 1 * 8) \% 11 = 8$	
$c(B, 0) = 0$	
$c(B, 1) = (h(B) + 1 * h_2(B)) \% 11 = (0 + 1 * 7) \% 11 = 7$	
$c(O, 0) = (h(O) + 0) \% 11 = 2$	
$c(L, 0) = (h(L) + 0) \% 11 = 10$	
$c(L, 1) = (h(L) + 1 * h_2(L)) \% 11 = (10 + 1 * 7) \% 11 = 6$	
$c(A, 0) = 10$	
$c(A, 1) = (h(A) + 1 * h_2(A)) \% 11 = (10 + 1 * 6) \% 11 = 5$	

search scansione lineare $\frac{1}{2} + \frac{1}{2(1-\alpha)^2}$

search scansione doppia $\frac{1}{1-\alpha}$

Nella realtà viene utilizzato l'hashing dinamico, aggiungo una cella alla volta o raddoppio la tabella per aumentare lo spazio. Se ci sono troppe celle vuote posso ridurre la dimensione della tabella.

CODE CON PRIORITÀ

<elemento, chiave>

- Operazioni:
 - findMin; (Importante), $\Theta(1)$
 - insert; (Importante), $\Theta(\log_d n)$
 - delete; $\Theta(d * \log n)$
 - deleteMin; (Importante), $\Theta(d * \log n)$ stessa dimostrazione dell'heap sort
 - increasekey; $\Theta(d * \log n)$
 - decreasekey; $\Theta(\log n)$

Struttura dati: **d-heap**.

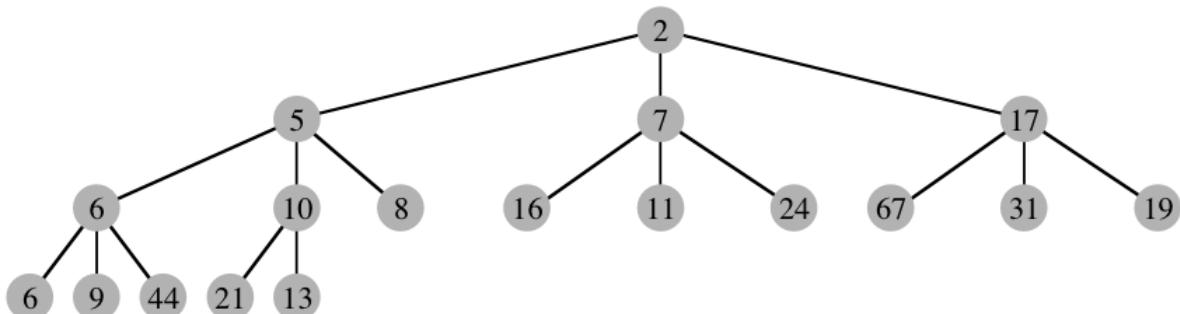
$d = \text{degree}$, ogni nodo interno ha al massimo d figli;

Albero completo fino al penultimo livello, ha una struttura rafforzata.

Il contenuto del nodo padre deve essere **minore o uguale** dei figli.

Il minimo di tutti è nella radice.

Non consente di fare la ricerca e l'ordinamento on place.



deleteMin

```

procedura muoviBasso (nodo v)
repeat
    sia u il figlio di v di chiave minima
    if(v non ha figli o chiave(v) <= chiave(u)) break
    scambia v e u
costo =  $\Theta(d * \log n)$ 
  
```

insert

```

procedura muoviAlto (nodo v)
while(v != radice and chiave(v) < chiave(padre(v)))
    scambia v e padre(v)
  
```

costo = $\Theta(\log_d n)$

delete

In base al nodo che cancello avrò bisogno della muoviAlto o muoviBasso in base alla situazione.

costo= $\Theta(d * \log n)$

increasekey

Molto probabile muoviBasso

costo= $\Theta(d * \log n)$

decreasekey

Molto probabile muoviAlto

costo= $\Theta(\log n)$