

Algoritmi e Strutture Dati

Lezione 15

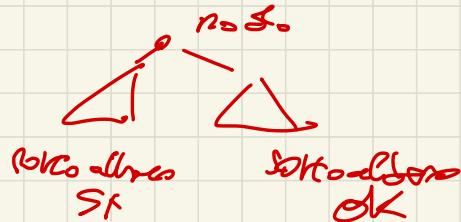
2 novembre 2022

Alberi generici



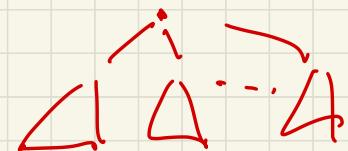
con radice

ALBERI (con radice)



→ Eredità-e-zis d' R.i. (e di sottoalbero)

→ Ogni nodo (eccetto la radice) ha
un unico padre



Esempio: indice di un Libro

1. Introduzione

- 1.1 Numeri di Fibonacci
- 1.2 Algoritmo numerico
- 1.3 Algoritmo ricorsivo

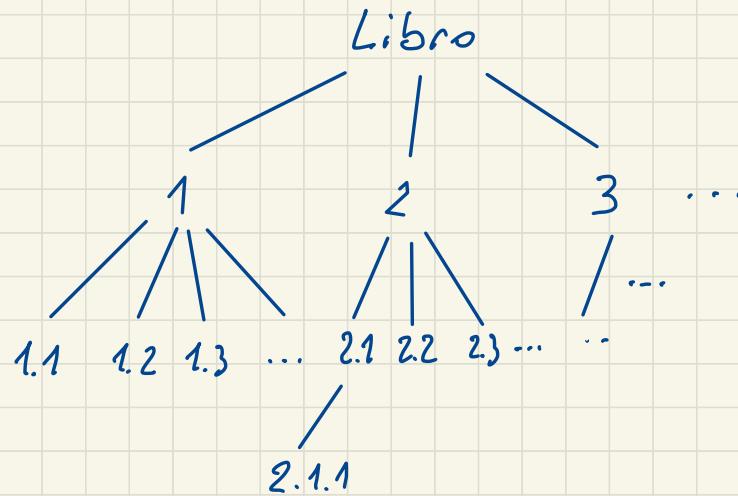
...

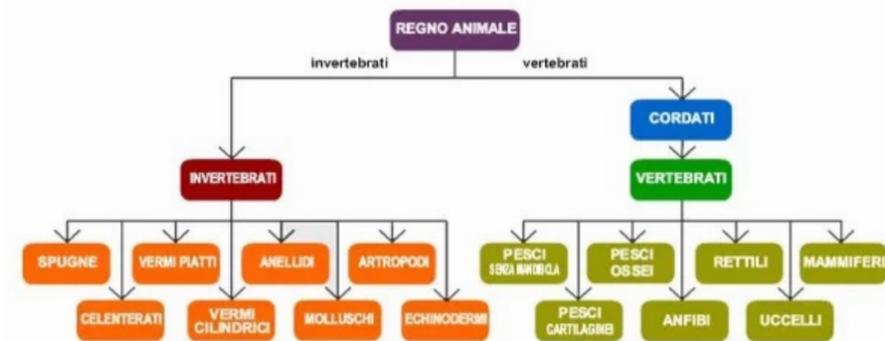
2. Modelli e metodologie

- 2.1 Modelli di calcolo
 - 2.1.1 Criteri di costo
- 2.2 Notazioni asintotiche
- 2.3 ...

...

3. Strutture dati elementari



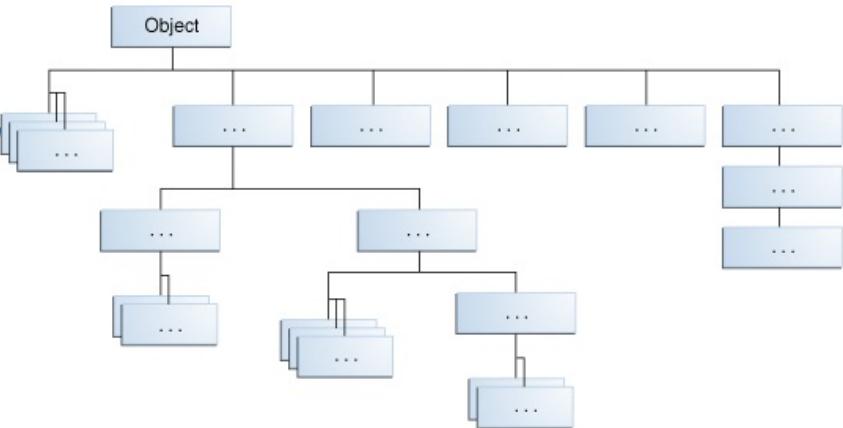


MergeSort ($0, 8$)

MergeSort ($0, 4$)

MergeSort ($4, 8$)

Gerarchia delle classi Java



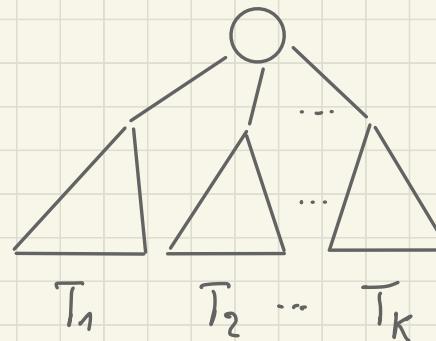
Albero con radice : definizione ricorsiva

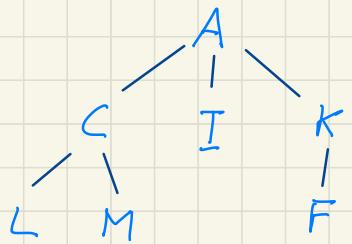
Un albero (con radice) è:

- la struttura vuota

oppure

- un nodo cui sono associati $K \geq 0$ alberi





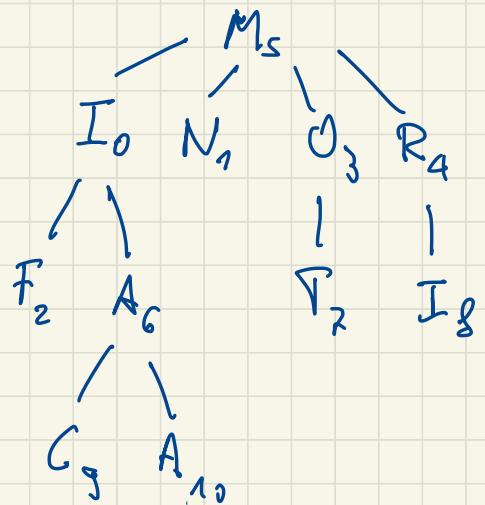
grado di un nodo = #figli

v. dell'albero = $\max_{\text{se. figli}} \text{grado}$

Rappresentazione: vettore dei padri / array con i valori contenuti nei nodi

array con le posizioni dei padri di classifici nodi

	0	1	2	3	4	5	6	7	8	9	10
info	I	N	F	O	R	M	A	T	I	C	A
padre	5	5	0	5	5	-1	0	3	4	6	6



Rappresentazione: vettore dei figli

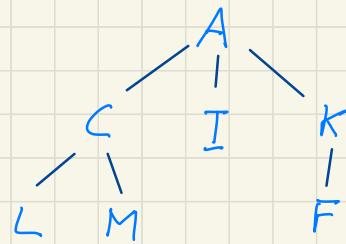
array con i valori contenuti nei nodi

o
array contenenti le posizioni dei figli di
ogni nodo
grado dell'albero

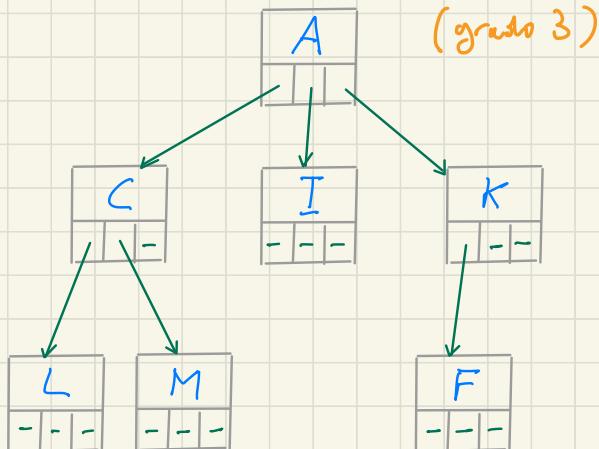
	0	1	2	3			
info	A	B	C		- .		
figli-1	2	3	-1		- .		
figli-2	1	m	-1		- .		
.							
figli-m	-1				- .		



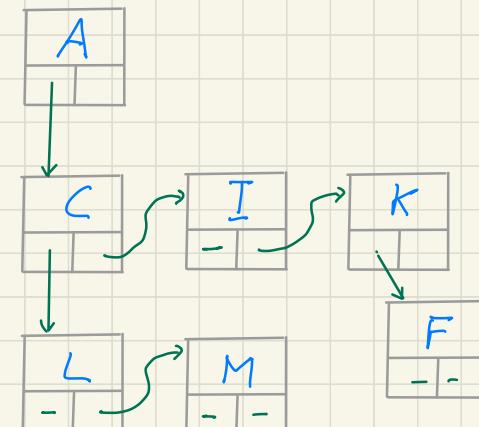
Rappresentazioni: collegate



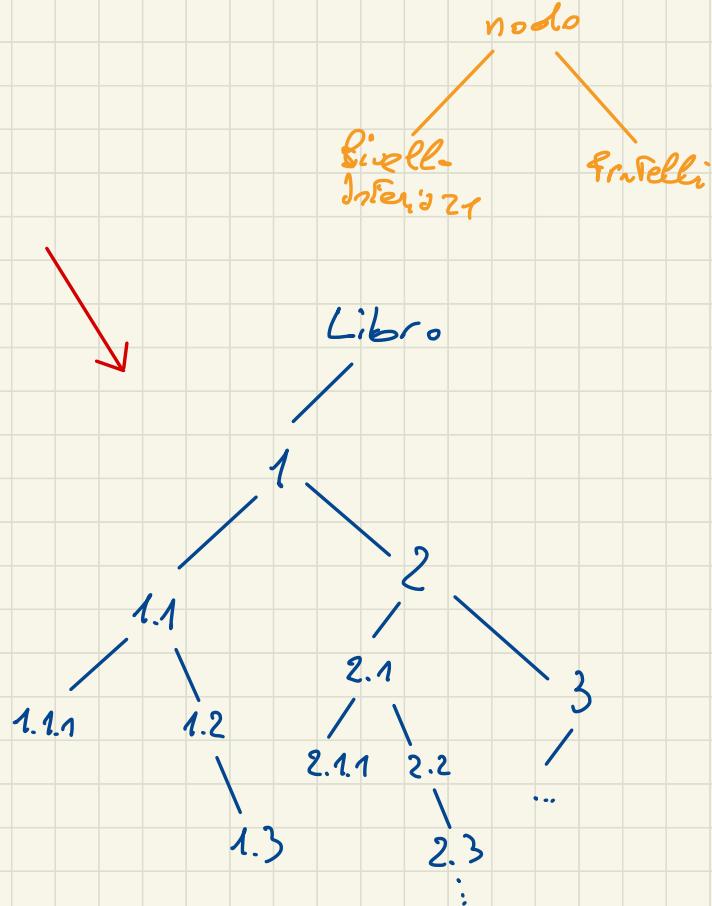
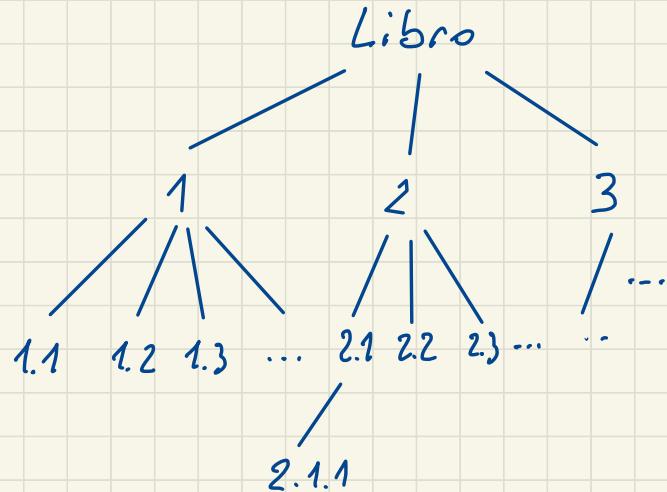
puntatori ai figli

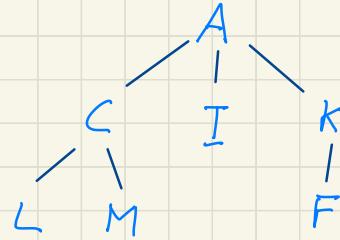
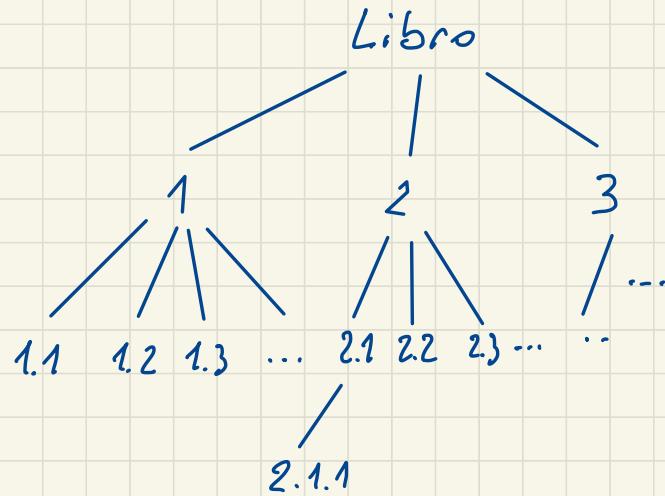


lista dei fratelli



ALBERI GENERICI → ALBERI BINARI





Visite

ampiezza

A, C, I, K, L, M, F

profondità

A, C, L, M, I, K, F

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

$$\cdot \sum_{i=0}^n i 2^i = ?$$

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

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

für Induktion

BASE
 $n=0$

$$\sum_{i=0}^n i \cdot 2^i = 0$$

$$-1 \cdot 2^1 + 2 = 0 \quad \underline{\text{OK}}$$

$$n-1 \rightarrow n$$

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

↑
ip. ind

$$= n \cdot 2^n + (n-1) \cdot 2^n + 2$$

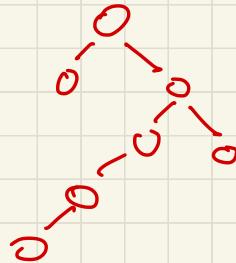
$$= n \cdot 2^n + n \cdot 2^n - 2^{n+1} + 2$$

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

Alberi binari: n° nodi vs altezza

n

h



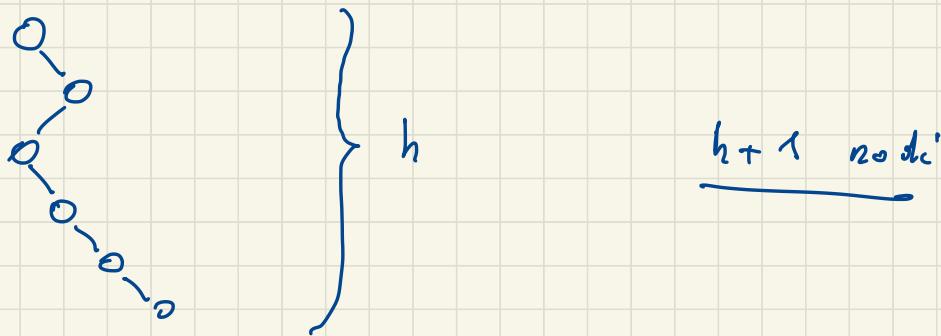
Che relazioni ci sono tra numero di nodi
e altezza in un albero binario?

Alberi binari: n^o nodi vs altezza

n

h

- Numero minimo di nodi per alberi di altezza h

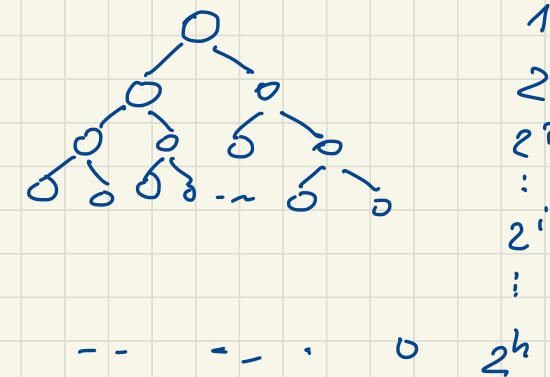


Alberi binari: n nodi vs altezza h

- Numero massimo di nodi per alberi di altezza h

albero completo
di altezza h

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



Alberi binari: n^o nodi vs altezza

n

h

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

$$h \leq n-1$$

$$2^{h+1} \geq n+1$$

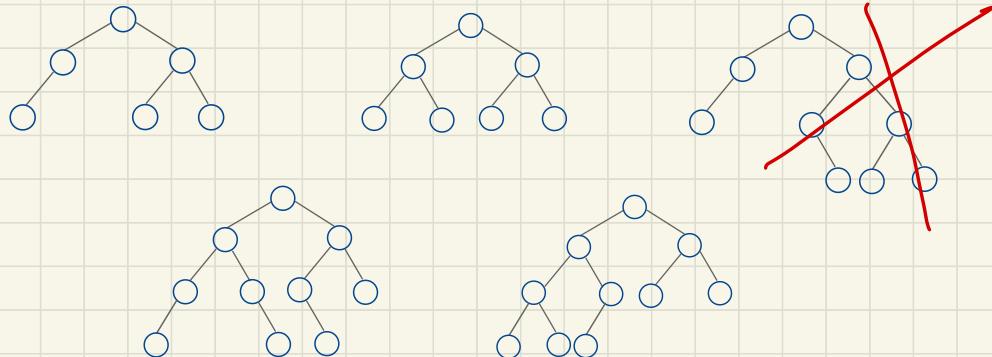
$$h+1 \geq \lg_2(n+1)$$

$$h \geq \lg_2(n+1) - 1$$

$$\lg_2(n+1) - 1 \leq h \leq n-1$$

Alberi binari "quasi completi"

Un albero binario è QUASI COMPLETO quando è completo almeno fino al penultimo livello



Alberi binari "quasi completi"

Un albero binario è QUASI COMPLETO quando è completo almeno fino al penultimo livello

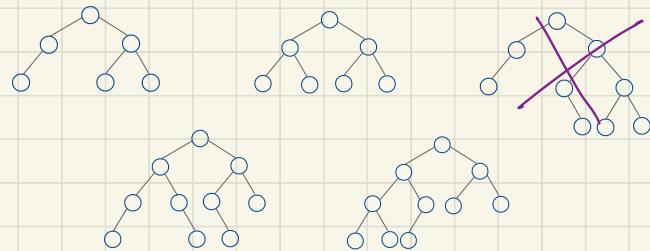
Proprietà:

Un albero binario di altezza h

è quasi completo se e solo se

ogni nodo di profondità $< h-1$

possiede ENTRAMBI i figli



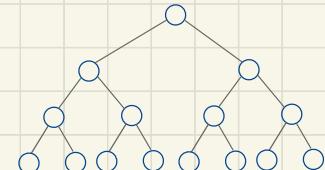
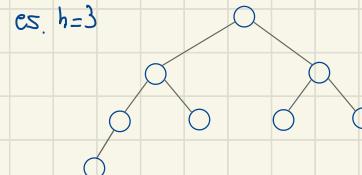
Alberi binari quasi completi: $\underbrace{n^o \text{ nodi}}_n$ vs $\underbrace{\text{altezza}}_h$

altezza h

- Guglielmo Ricci
al altezza $h-1$ | $2^h - 1$ nodi al profondità $\leq h-1$
- Nodi al profondità h : almeno 1, al max 2^h

$$2^h \leq n < 2^{h+1}$$

$$h \leq \log_2 n < h+1$$

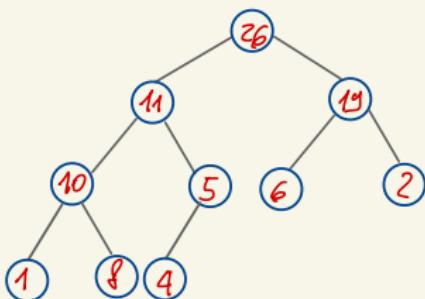


$$h = \lfloor \log_2 n \rfloor$$

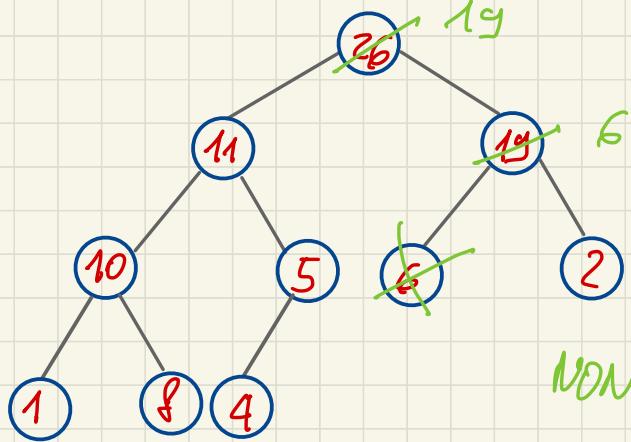
Heapsort

Definizione

Uno *heap* (o *max-heap*) è un albero binario quasi completo in cui la chiave contenuta in ciascun nodo è maggiore o uguale delle chiavi contenute nei figli

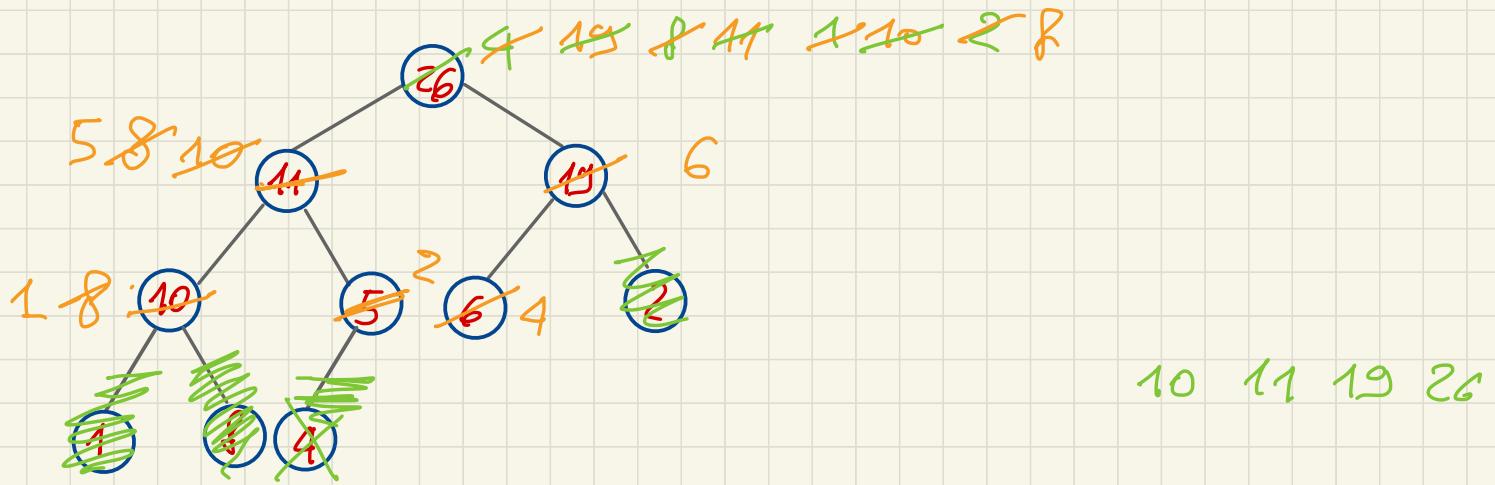


Per comodità, consideriamo heap in cui le foglie dell'ultimo livello si trovano più a sinistra possibile



NON È un albero

26



"risistema" (fixHeap) uno heap con radice "sbagliata"

PROCEDURA risistema (Heap H)

$v \leftarrow H$ v : nodo in esame

$x \leftarrow v.\text{chiave}$ x : chiave da risistemare

$y \leftarrow v.\text{altri campi}$ y : campi associati a x

$\text{daCollocare} \leftarrow \text{true}$

DO

 IF v è una Foglia THEN

$\text{daCollocare} \leftarrow \text{false}$

 ELSE

$m \leftarrow \text{figlio di } v \text{ di valore max}$

 IF $m.\text{chiave} > x$ THEN

$v.\text{chiave} \leftarrow m.\text{chiave}$

$v.\text{altri campi} \leftarrow m.\text{altri campi}$

$v \leftarrow m$

 ELSE

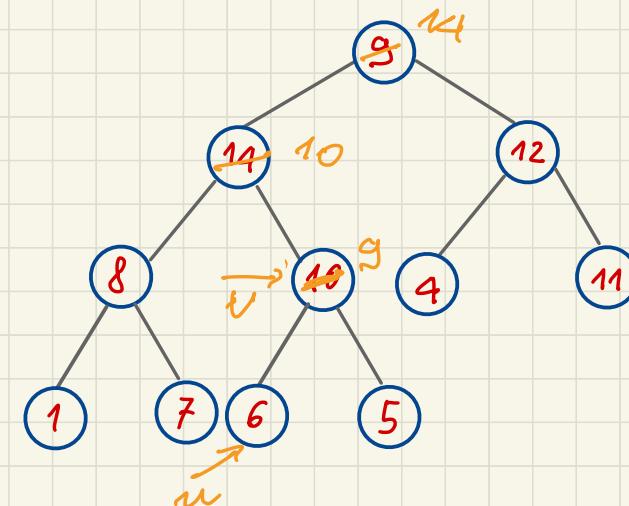
$\text{daCollocare} \leftarrow \text{false}$

 WHILE daCollocare

$v.\text{chiave} \leftarrow x$

$v.\text{altri campi} \leftarrow y$

$\times \underline{L} \underline{g} \underline{1}$



$\Theta(h)$ confronti ($h = \text{altezza}$)

Costruzione di heap

Dato un albero binario quasi completo contenente gli elementi da ordinare, come posso trasformarlo in uno heap?

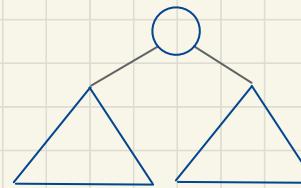
Soluzione 1: Tecnica divide-et-impera
(strategia top-down)

Soluzione 2: Dai sottoalberi più piccoli a quelli più grandi
(strategia bottom-up)

Soluzione 1: divide - et - impera

Albero vuoto \rightarrow è già un heap

Albero non vuoto



PROCEDURA creatheap (albero T)

IF $T \neq$ albero vuoto THEN

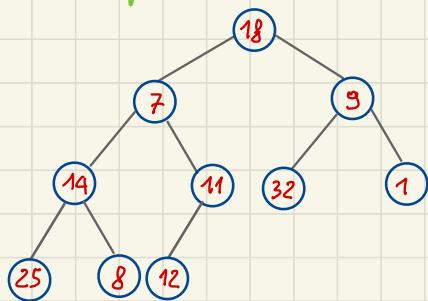
 creatheap ($T.cx$)

 creatheap ($T.dx$)

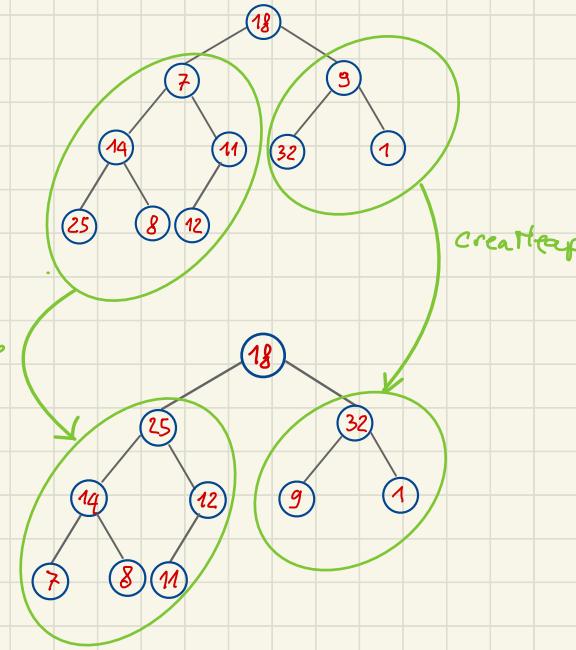
 risistema (T)

Esempio

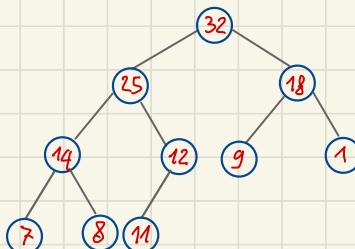
creatheap



creatheap

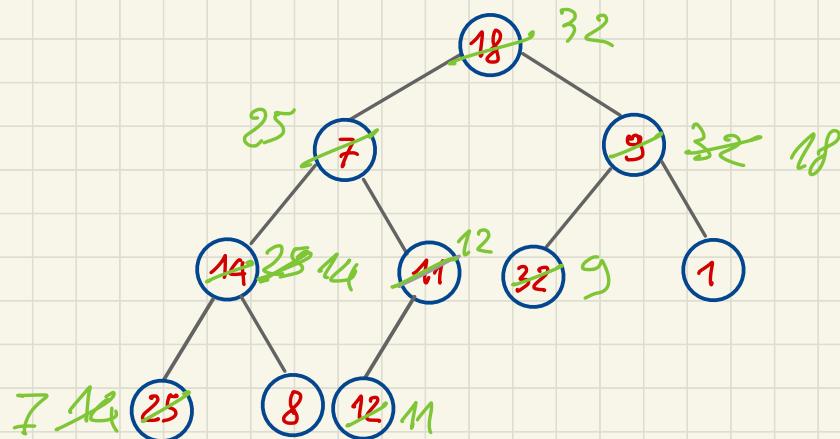


risistema



usa stack

Soluzione 2: partiamo dalle foglie

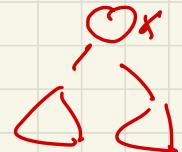


PROCEDURA creatHeap (Albero T)

$h \leftarrow$ albero di T

FOR $p \leftarrow h$ DOWNTO 0 DO

FOREACH nodo x di profondità p DO
risistem (cattalbero di radice x)



cfr

- 1 chiamata di risistem per ogni nodo
- risistem $\in \Theta(\underbrace{\text{profondità dell'albero}}_{\leq \log n})$
- n chiamate di risistem
 $\leq n \cdot \Theta(\log n) = \Theta(n \log n)$