Algoritmi e Strutture Dati

Lezione 16

29 ottobre 2025

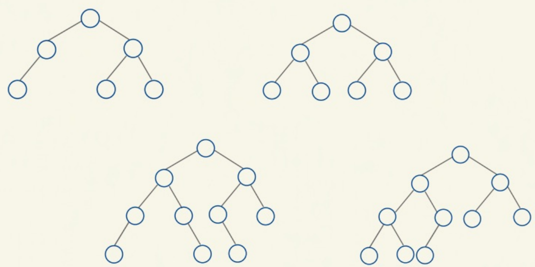
Alberi binari "quasi completi"

Definizione

Un *albero binario* è *quasi completo* quando è completo almeno fino al penultimo livello

in modo equivalente:

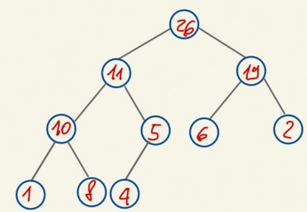
ogni nodo di profondità *minore* di $h-1$ possiede *entrambi* i figli, dove h è l'altezza dell'albero



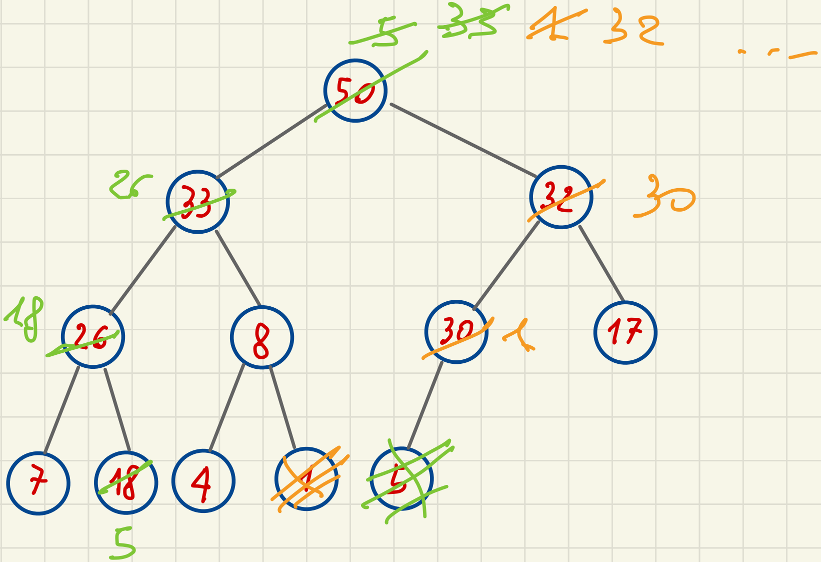
Heap (Mucchio)

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

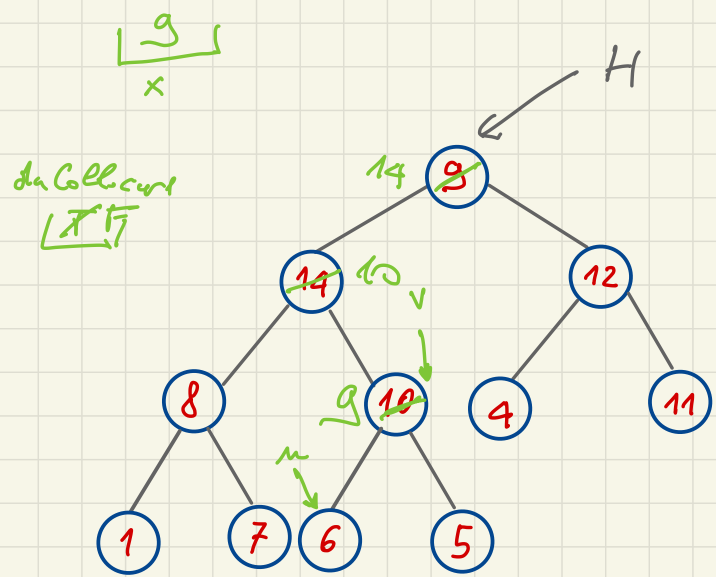


X

33 50

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

```
PROCEDURA risistema (Heap H)
  v ← H                      v: nodo in esame
  x ← v.chiave               x: chiave da sistemare
  y ← v.altri campi          y: campi associati a x
  daCollocare ← true
  DO
    IF v è una foglia THEN
      daCollocare ← false
    ELSE
      u ← figlio di v di valore max
      IF u.chiave > x THEN
        v.chiave ← u.chiave
        v.altricampi ← u.altricampi
        v ← u
      ELSE
        daCollocare ← false
  WHILE daCollocare
  v.chiave ← x
  v.altricampi ← y
```



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

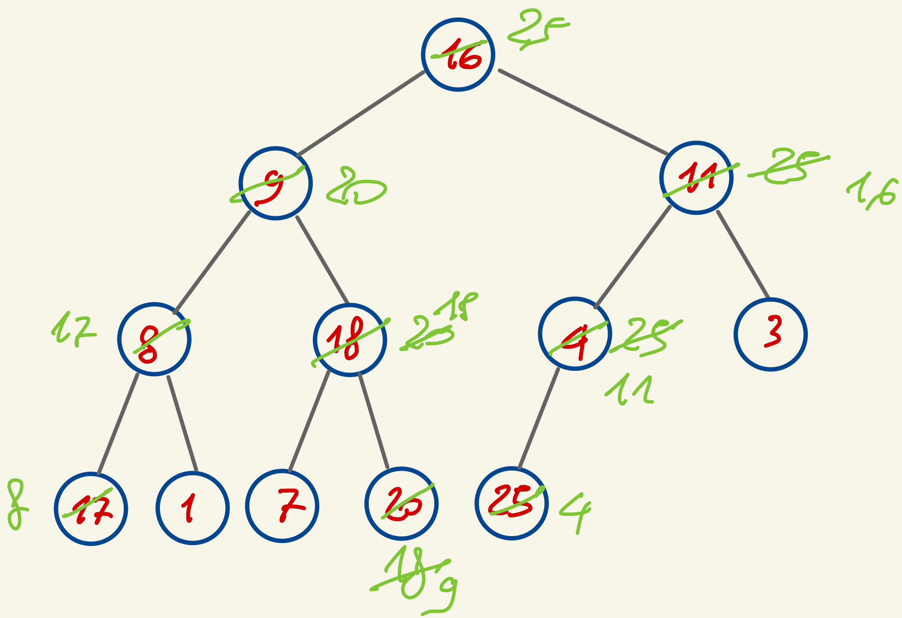
Costruzione di heap

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) ← uso dello stack

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

```
PROCEDURA creaHeap (Albero T)
    h ← altezza di T
    FOR p ← h DOWNTO 0 DO
        FOREACH nodo x di profondità p DO
            risistema (sottoalbero di radice x)
```



ALGORITMO heapSort (AlberoBinarioQuasiCompleto H) $\rightarrow$ Lista

createHeap(H) $\Theta(n)$ confronti

X $\leftarrow$ lista vuota

WHILE $H \neq \emptyset$ DO

rimuovi da H l'elemento alla radice
e aggiungilo all'inizio di X

rimuovi la foglia più a destra e
colloca il contenuto nella radice

risistema H $O(\log n)$ cfr

$\underline{n}$ iterazioni

$O(n \log n)$ cfr

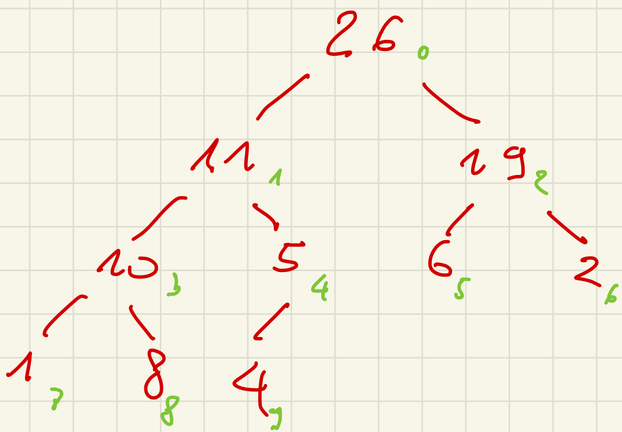
RETURN X

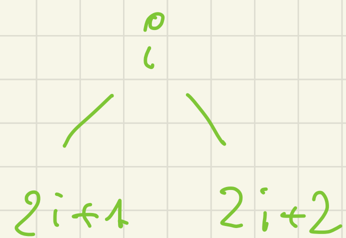
$$\Theta(n) + O(n \log n) = \Theta(n \log n)$$

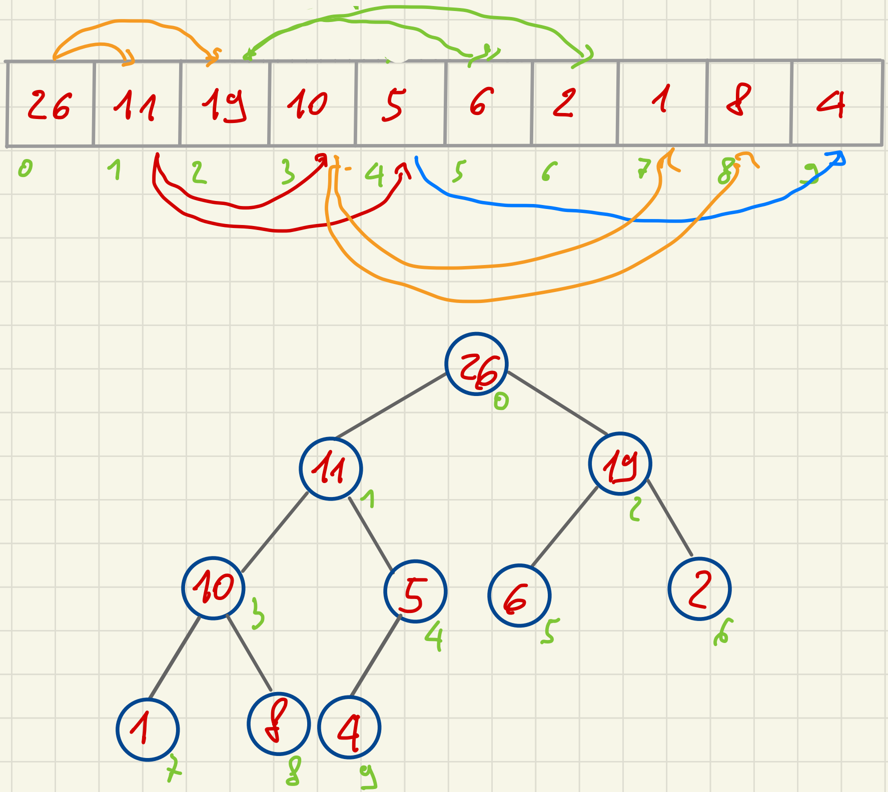
confronti

HeapSort: implementazione in loco

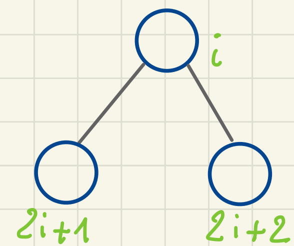
26	11	19	10	5	6	2	1	8	4
0	1	2	3	4	5	6	7	8	9





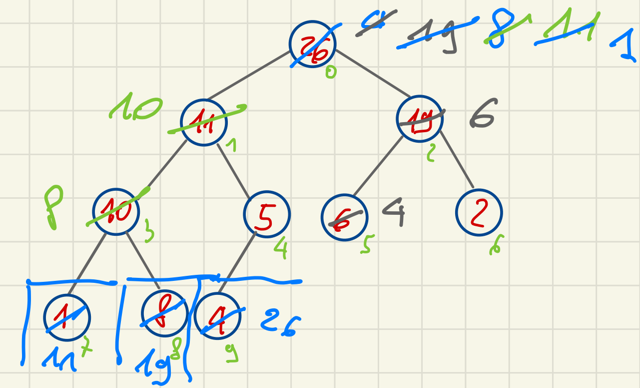


VETTORE POSIZIONALE



4									26
~~26~~	11	19	10	5	6	2	1	8	~~4~~
0	1	2	3	4	5	6	7	8	9

l



8								19	
~~19~~	11	6	10	5	4	2	1	~~8~~	26
0	1	2	3	4	5	6	7	8	9

l

$\underbrace{\qquad\qquad\qquad\qquad}_{\text{Heap}}$ $\underbrace{\qquad\qquad}_{\text{parte ordinata}}$

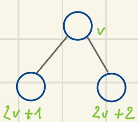
1							11		
~~11~~	10	6	8	5	4	2	~~1~~	19	26
0	1	2	3	4	5	6	7	8	9

l

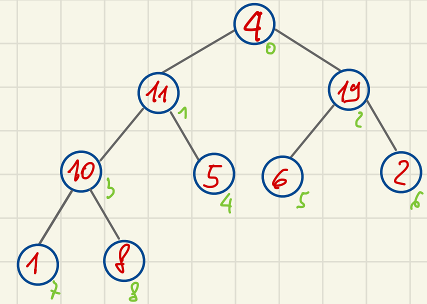
ripristina — —

PROCEDURA risistema (Array A[0..n-1], intero r, intero ℓ)

```
v ← r                  v: posizione in esame
x ← A[v]               x: chiave da sistemare
per semplicità viene indicata solo la chiave
daCollocare ← true
DO
  IF 2*v+1 ≥ ℓ THEN          v è l'indice di una foglia
  |  daCollocare ← false
  ELSE
  |  u ← indice del figlio di v di valore max
  |  IF A[u] > x THEN
  |  |  A[v] ← A[u]
  |  |  v ← u
  |  ELSE
  |  |_ daCollocare ← false
WHILE daCollocare
A[v] ← x
```




```
u ← 2*v+1
IF u+1 < ℓ AND A[u+1] > A[u] THEN
   u ← u+1
```



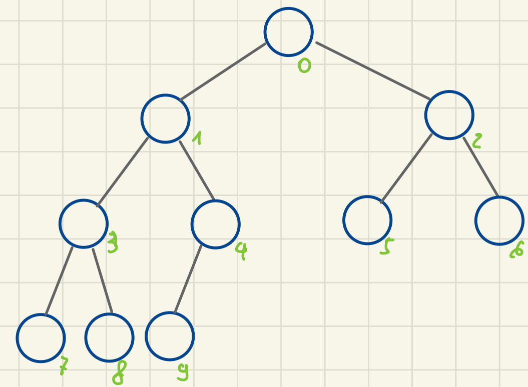
4	11	19	10	5	6	2	1	8	26
0	1	2	3	4	5	6	7	8	9
r									ℓ

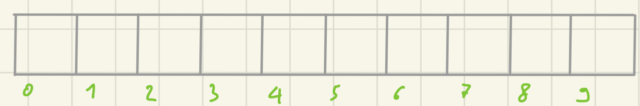
PROCEDURA creaHeap (Array A[0..n-1])

FOR ~~i ← n-1~~ i ← n/2 DOWNTO 0 DO

risistema (A, i, n)

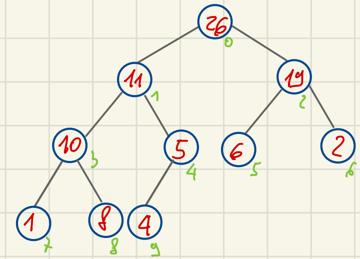
radice albero da risistemare





ALGORITMO heapSort(Array A[0..n-1])

```
creaHeap(A)
FOR ℓ ← n-1 DOWNTO 1 DO
    scambia A[0] con A[ℓ]
    risistema(A, 0, ℓ)
```

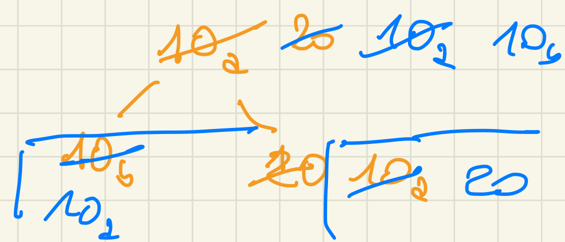


26	11	19	10	5	6	2	1	8	4
0	1	2	3	4	5	6	7	8	9
									↑ ℓ

HeapSort: riassumendo

- Algoritmo di ordinamento IN LOCO
- No memoria aggiuntiva (con createHeap bottom-up)
- #CFR $\Theta(n \log n)$
- Tempo $\Theta(n \log n)$ se i cfr costano $O(1)$
- NON STABILE

$10_2 \quad 10_6 \quad 20$

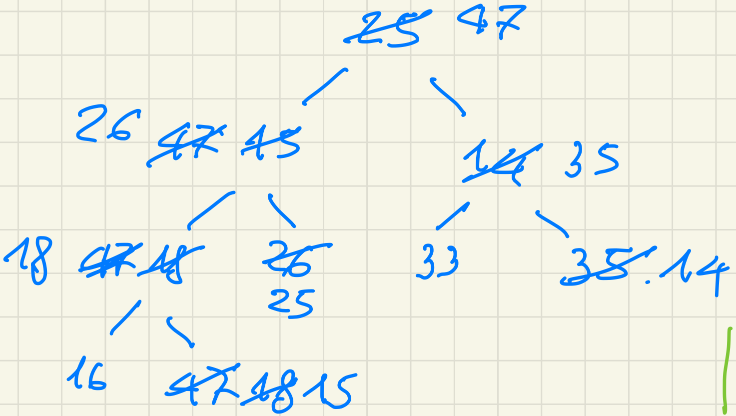


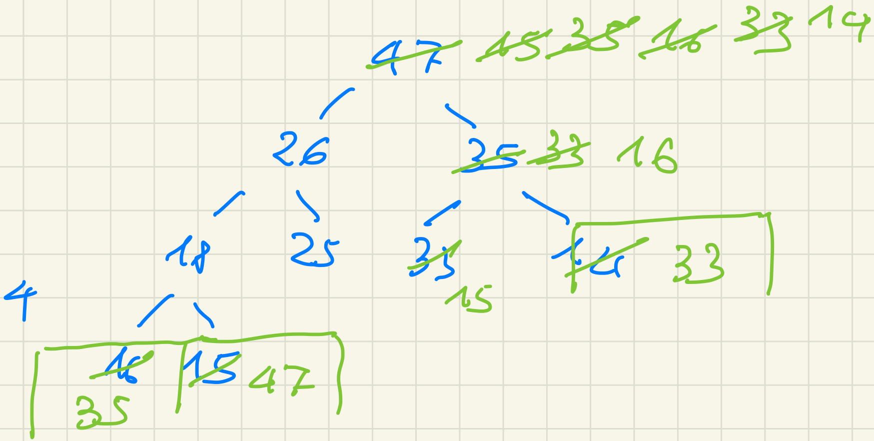
$10_6 \quad 10_2 \quad 20$

Esercizio. Considerate la seguente sequenza di numeri, memorizzata in un array: **25 15 14 18 26 33 35 16 47**

Supponete di ordinare la sequenza in modo crescente mediante l'algoritmo heapSort.

Indicate il contenuto dell'array immediatamente dopo lo scambio che colloca **33** nella sua posizione definitiva.





$$\underbrace{14\quad 26\quad 16\quad 18\quad 25\quad 15}\quad \underbrace{33\quad 35\quad 47}$$

Algoritmi di ordinamento basati su confronti

Ordinamento

Problema dell'ordinamento:

Input: n elementi $x_1, x_2, \ldots, x_n$
provenienti da un dominio D su cui è definita una relazione $\leq$ di *ordine totale*

Output: Sequenza $x_{j_1}, x_{j_2}, \ldots, x_{j_n}$
con $(j_1, j_2, \ldots, j_n)$ permutazione di $(1, 2 \ldots, n)$
tale che $x_{j_1} \leq x_{j_2} \leq \cdots \leq x_{j_n}$

#permutazioni di n elementi: $n!$

Ordinamento: metodi basati su confronti

Algoritmo	*Numero confronti*	*Spazio*	*Note*	*Stabile*
selectionSort	$\Theta(n^2)$ sempre	$O(1)$	in loco	no
insertionSort	$\Theta(n^2)$ pegg. $n-1$ migliore	$\Theta(1)$	in loco	sì
bubbleSort	$\Theta(n^2)$ pegg. $n-1$ migliore	$\Theta(1)$	in loco	sì
mergeSort	$\Theta(n \log n)$	$\Theta(n)$	vettore ausil. $\Theta(n)$ stack ric. $\Theta(\log n)$	sì
quickSort	$\Theta(n^2)$ pegg. $\approx n \log_2 n$ migliore $\approx 1.39\, n \log_2 n$ medio	$\Theta(n)$ $\Theta(\log n)$	in loco + stack → stack versione base versione migliorata	no
heapSort	$\Theta(n \log n)$	$O(1)$	in loco	no

Ordinamento: metodi basati su confronti

Algoritmo	*Numero confronti*	*Spazio*	*Note*	*Stabile*
`selectionSort`	$\Theta(n^2)$ sempre	$\Theta(1)$	in loco	no
`insertionSort`	$\Theta(n^2)$ nel caso peggiore $n-1$ su array già ordinato	$\Theta(1)$	in loco	sì
`bubbleSort`	$\Theta(n^2)$ nel caso peggiore $n-1$ su array già ordinato	$\Theta(1)$	in loco	sì
`mergeSort`	$\Theta(n \log n)$	$\Theta(n)$	spazio $\Theta(n)$ per array ausiliario più $\Theta(\log n)$ per stack ricorsione	sì
`quickSort`	$\Theta(n^2)$ nel caso peggiore $\Theta(n \log n)$ caso migliore $\approx 1.39 n \log_2 n$ in media	$\Theta(n)$ $\Theta(\log n)$	in loco spazio $\Theta(1)$ più stack ricorsione: $\Theta(n)$ algoritmo base $\Theta(\log n)$ algoritmo migliorato	no
`heapSort`	$\Theta(n \log n)$	$\Theta(1)$	in loco	no

Ordinamento: metodi basati su confronti

È possibile ordinare array di n elementi utilizzando un numero di confronti tra chiavi che cresca meno di $n \log n$?

NO!

$x_i \leq x_j$?

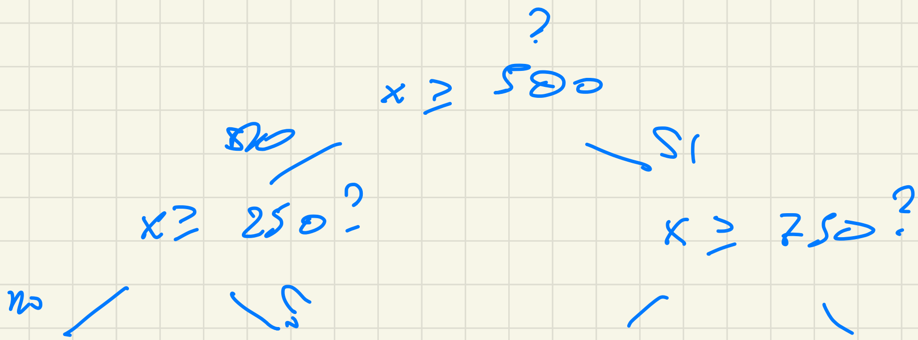
sì

no

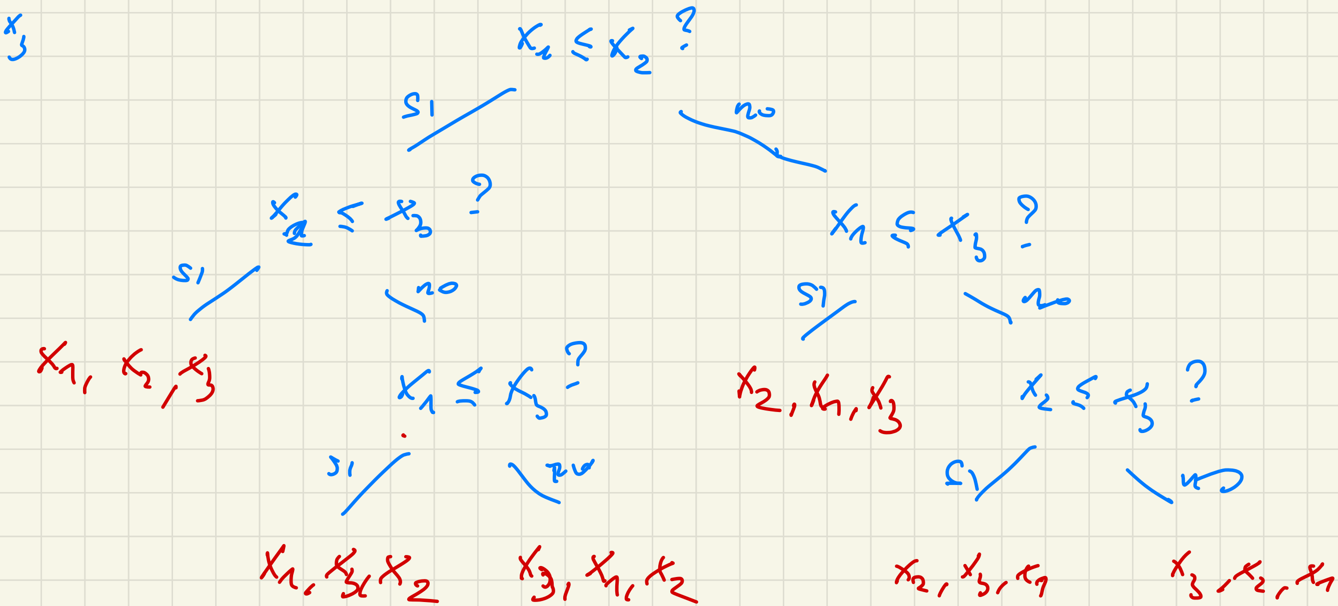
ALBERO DI DECISIONE

nodi interni: domande sì/no

foglie: possibili risultati



x_1, x_2, x_3



$$\text{Altezza} \geq \log_2(\#\,\text{nodi}) \geq \log_2(\underbrace{\#\,\text{foglie}}_{\geq\, \#\,\text{permutazioni} \,=\, n!}) \geq \log_2(n!)$$

$$\#\,\text{cfr} \geq \log_2(n!)$$

minimo di cfr $\geq \lg_2(n!)$

$$n! \approx \sqrt{2\pi n} \cdot \left(\frac{n}{e}\right)^n$$ appross. di Stirling

$$\lg_2(n!) \approx \lg_2\left(\sqrt{2\pi n}\left(\frac{n}{e}\right)^n\right) =$$

$$= \lg_2 \sqrt{2\pi} + \lg_2 \sqrt{n} + \lg_2\left(\frac{n}{e}\right)^n \rightarrow n \lg_2 \frac{n}{e}$$

$$= \lg_2 \sqrt{2\pi} + \frac{1}{2}\lg_2 n + n\lg_2 n - n\lg_2 e$$

$$= \Theta(n \lg n)$$