



Gli heap



Fulvio CORNO - Matteo SONZA REORDA
Dip. Automatica e Informatica
Politecnico di Torino

1

a.a. 2001/200.

Sommario

- Gli heap
- L'algoritmo Heapsort
- Le code con priorità.

2

a.a. 2001/200.

Definizione di heap

Uno heap è un albero binario quasi completo nel quale, detto $key(i)$ il valore associato al generico nodo i , per ogni nodo i (ad esclusione della radice) è soddisfatta la proprietà

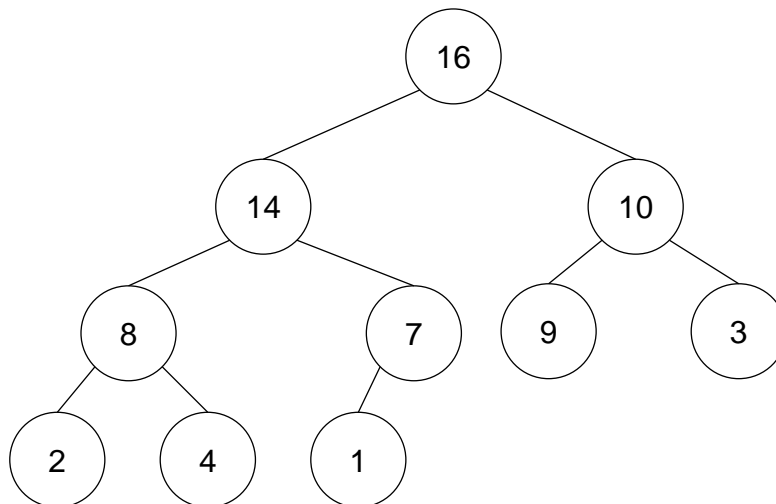
$$key(i) \leq key(padre(i))$$

Ne consegue che l'elemento con valore minore è memorizzato nella radice.

3

a.a. 2001/2002

Esempio



4

a.a. 2001/2002



Altezza

Ad ogni nodo di uno heap è associata una altezza, definita come il numero di archi sul più lungo cammino che va dal nodo ad una foglia.

L'altezza di uno heap è l'altezza della sua radice.

Poiché lo heap è un albero binario quasi completo, se il numero di nodi è pari ad n , la sua altezza sarà $\Theta(\lg n)$.

5

a.a. 2001/200.

Implementazione

Gli heap sono normalmente implementati come vettori.

In particolare

- La radice viene memorizzata nell'elemento di indice 1 (il primo del vettore)
- I due figli dell'elemento in posizione i vengono memorizzati negli elementi in posizione $2i$ (figlio sinistro) e $2i+1$ (figlio destro).

6

a.a. 2001/200.

Limit

Dal momento che lo heap è un albero binario quasi completo, l'indice i di tutti i suoi elementi nel vettore rispettano l'espressione

$$i \leq n$$

essendo n il numero di elementi nello heap.

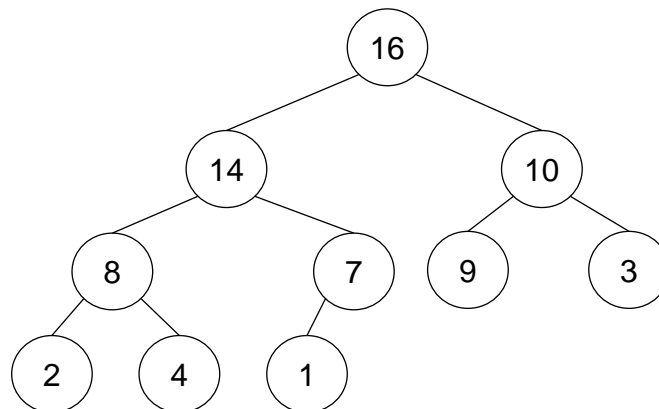
Di conseguenza, si può dire che:

- Se $i \leq \lfloor n/2 \rfloor$, il nodo i ha due figli
- Se $i = \lfloor n/2 \rfloor$, il nodo i ha uno (n pari) o due (n dispari) figli
- Se $i > \lfloor n/2 \rfloor$, il nodo i non ha alcun figlio

7

a.a. 2001/200.

Esempio di implementazione



1	2	3	4	5	6	7	8	9	10
16	14	10	18	7	9	3	2	4	1

8

a.a. 2001/200.

Costruzione di uno heap

Si utilizza come base la procedura $\text{Heapify}(A, i)$, i cui parametri sono il vettore A ed un indice i al suo interno.

La procedura presuppone che

- i due alberi corrispondenti ai figli dell'elemento memorizzato in i siano degli heap
- l'elemento in i possa avere valore minore di quelli dei figli.

Se necessario, la procedura provvede quindi a trasformare l'albero in uno heap.

9

a.a. 2001/200.

Heapify

$\text{HEAPIFY}(A, i)$

```
1   $l \leftarrow \text{LEFT}(i)$ 
2   $r \leftarrow \text{RIGHT}(i)$ 
3  if  $l \leq \text{heap-size}[A]$  e  $A[l] > A[i]$ 
4      then  $\text{largest} \leftarrow l$ 
5      else  $\text{largest} \leftarrow i$ 
6  if  $r \leq \text{heap-size}[A]$  e  $A[r] > A[\text{largest}]$ 
7      then  $\text{largest} \leftarrow r$ 
8  if  $\text{largest} \neq i$ 
9      then scambia  $A[i] \leftrightarrow A[\text{largest}]$ 
10      $\text{HEAPIFY}(A, \text{largest})$ 
```

10

a.a. 2001/200.

Heapify

HEAPIFY(A, i)

```
1  l ← LEFT(i)
2  r ← RIGHT(i)
3  if l ≤ heap-size[A] e A[l] > A[i]
4      then largest ← l
5      else largest ← i
6  if r ≤ heap-size[A] e A[r] > A[largest]
7      then largest ← r
8  if largest ≠ i
9      then scambia A[i] ↔ A[largest]
10     HEAPIFY(A, largest)
```

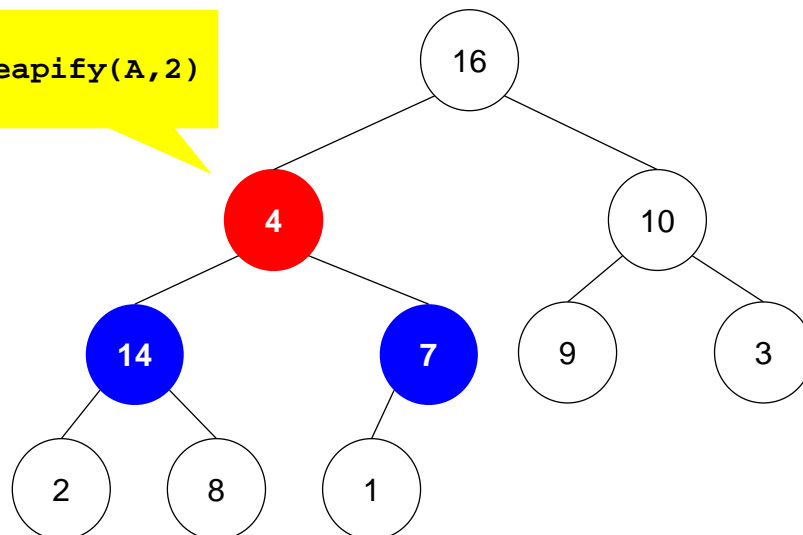
Le istruzioni 3-7 calcolano il massimo tra A[l], A[i], A[r], basandosi sull'ipotesi che A[l] e A[r] siano già degli heap.

11

a.a. 2001/200.

Esempio (1)

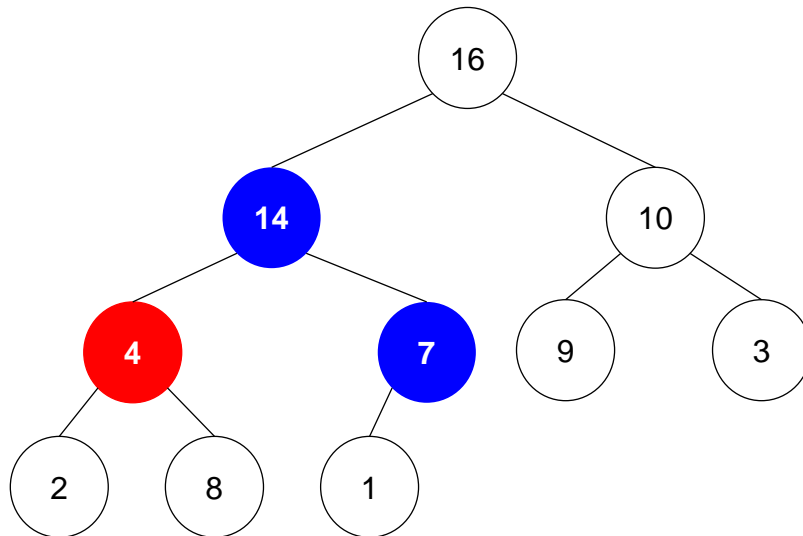
Heapify(A, 2)



12

a.a. 2001/200.

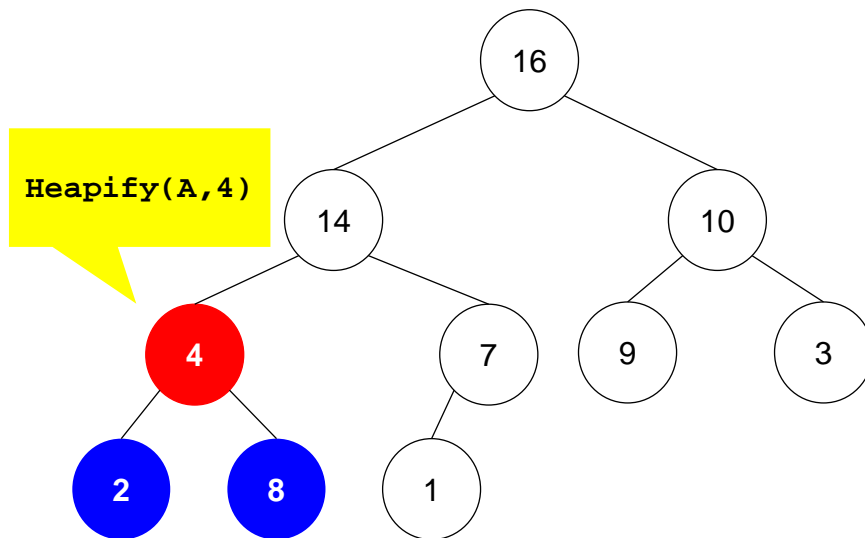
Esempio (2)



13

a.a. 2001/200.

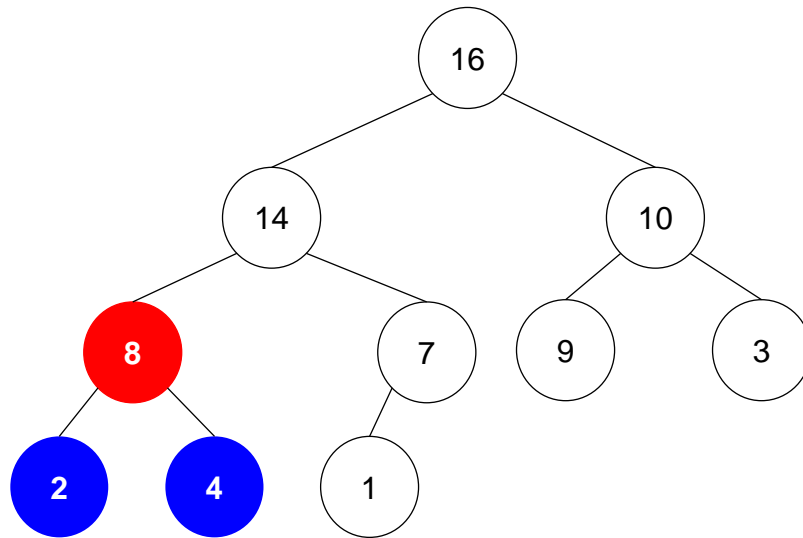
Esempio (3)



14

a.a. 2001/200.

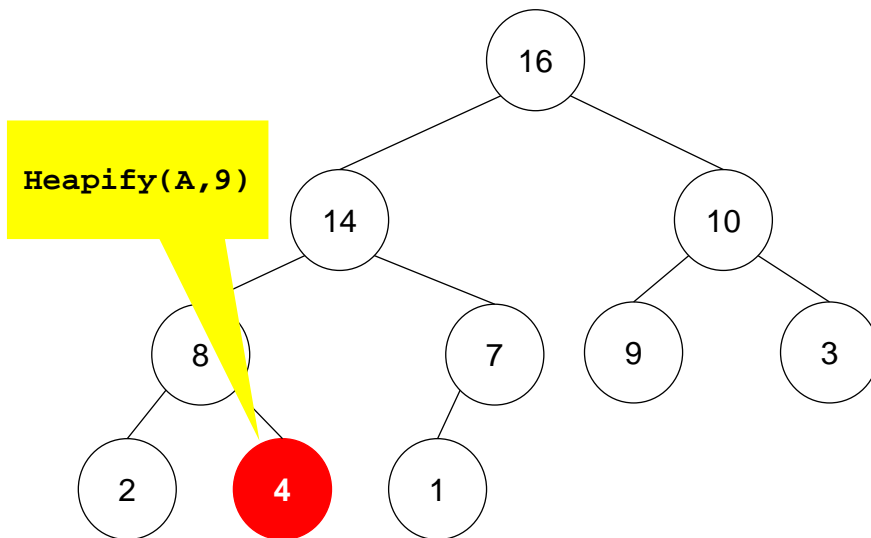
Esempio (4)



15

a.a. 2001/2002

Esempio (5)



16

a.a. 2001/2002

Complessità

La procedura `Heapify` ha una complessità data da

$$T(n) = \Theta(1) + T(2/3n)$$

dove

- $\Theta(1)$ è il costo per eseguire i confronti necessari per calcolare `largest`
- $T(2/3n)$ è il costo della chiamata recursiva su uno dei figli, pari alla dimensione del figlio nel caso peggiore.

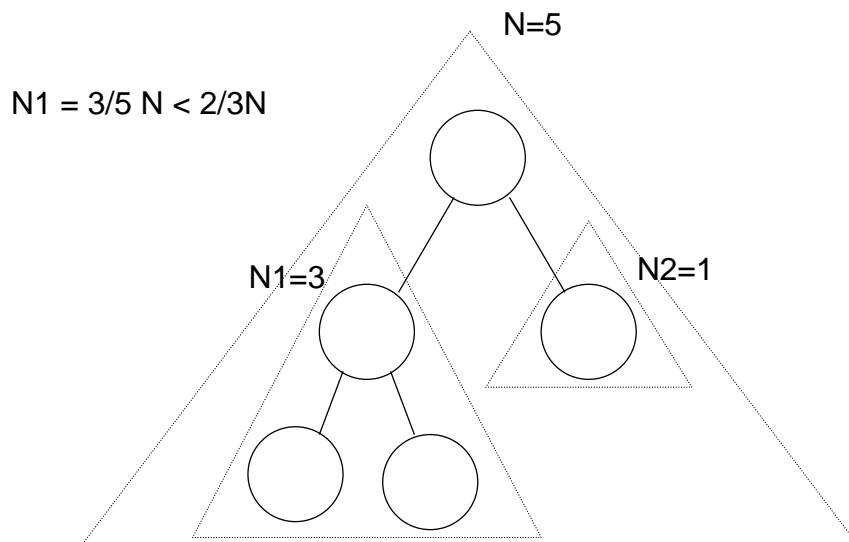
In base al teorema principale, tale complessità può essere espressa come

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

17

a.a. 2001/200.

Fattore 2/3: esempio



18

a.a. 2001/200.



Build-Heap

Utilizza la procedura `Heapify` per trasformare un albero binario (memorizzato in un vettore) in uno heap.

Consiste nell'applicare `Heapify` sugli elementi nella prima metà del vettore.

Build-heap

BUILD-HEAP(*A*)

```
1  heap-size[A] ← length[A]  
2  for i ← ⌊length[A]/2⌋ downto 1  
3      do HEAPIFY(A, i)
```

Esempio (0)

Si vuole trasformare in heap l'albero binario disegnato.

```
graph TD; 4((4)) --- 1((1)); 4 --- 3((3)); 1 --- 2((2)); 1 --- 16((16)); 2 --- 14((14)); 2 --- 8((8)); 16 --- 7((7)); 3 --- 9((9)); 3 --- 10((10));
```

21 a.a. 2001/200.

Esempio (1)

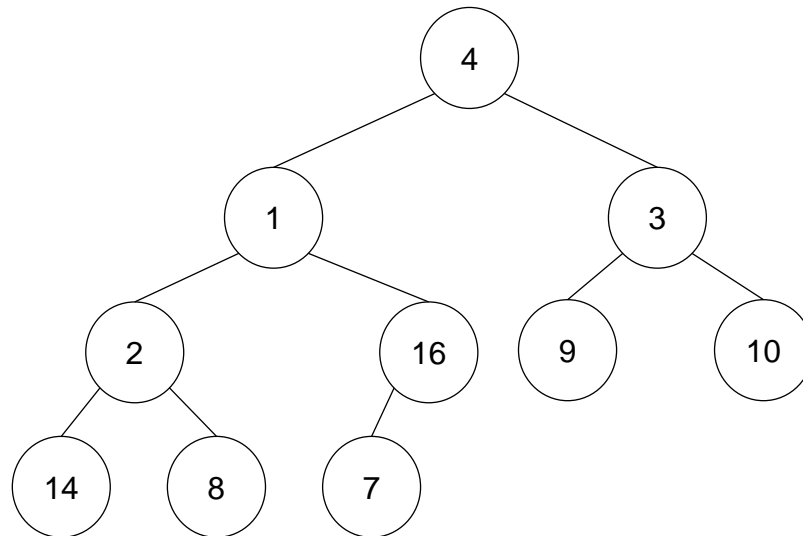
Heapify(A, 5)

```
graph TD; 4((4)) --- 1((1)); 4 --- 3((3)); 1 --- 2((2)); 1 --- 16((16)); 2 --- 14((14)); 2 --- 8((8)); 16 --- 7((7)); 3 --- 9((9)); 3 --- 10((10));
```

22 a.a. 2001/200.



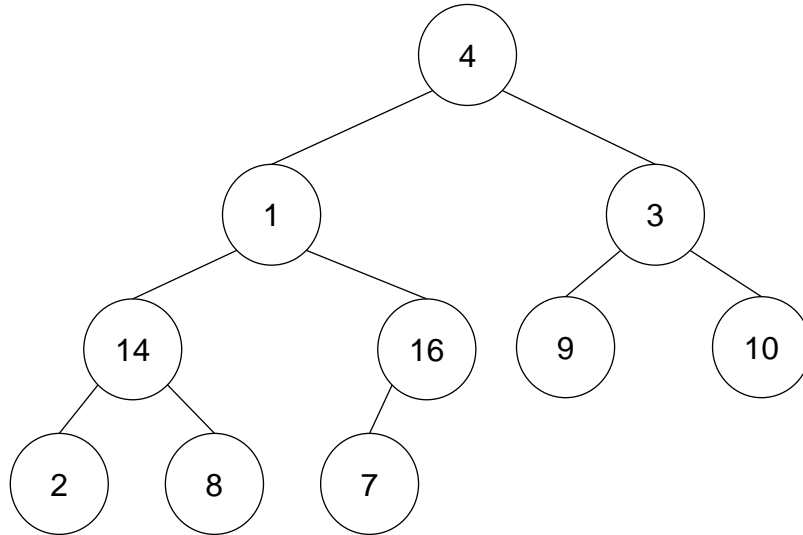
Esempio (2)



23

a.a. 2001/2002

Esempio (4)

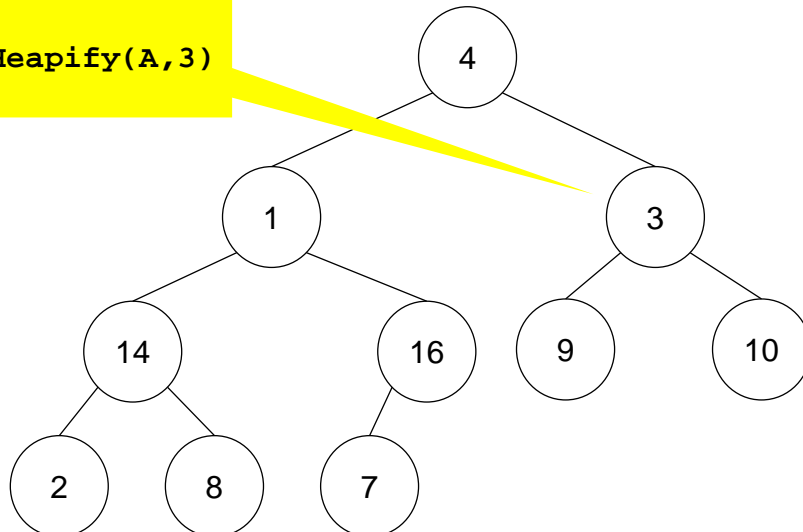


25

a.a. 2001/200.

Esempio (5)

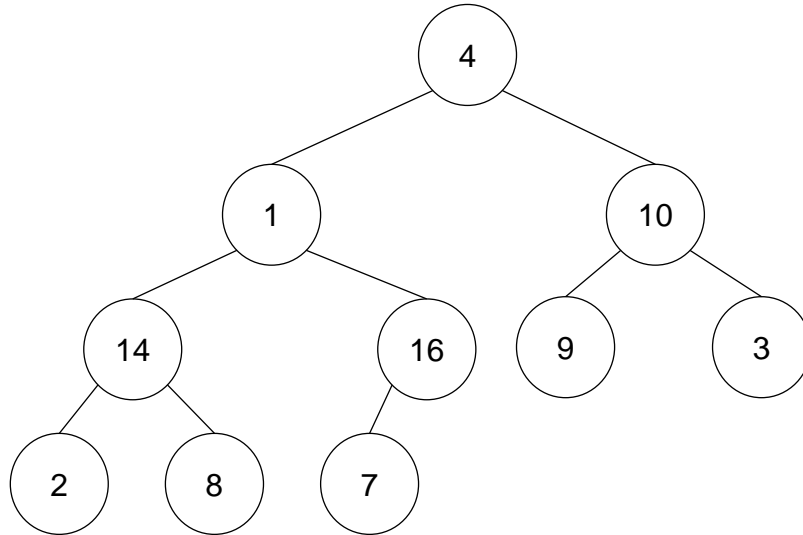
Heapify(A, 3)



26

a.a. 2001/200.

Esempio (6)

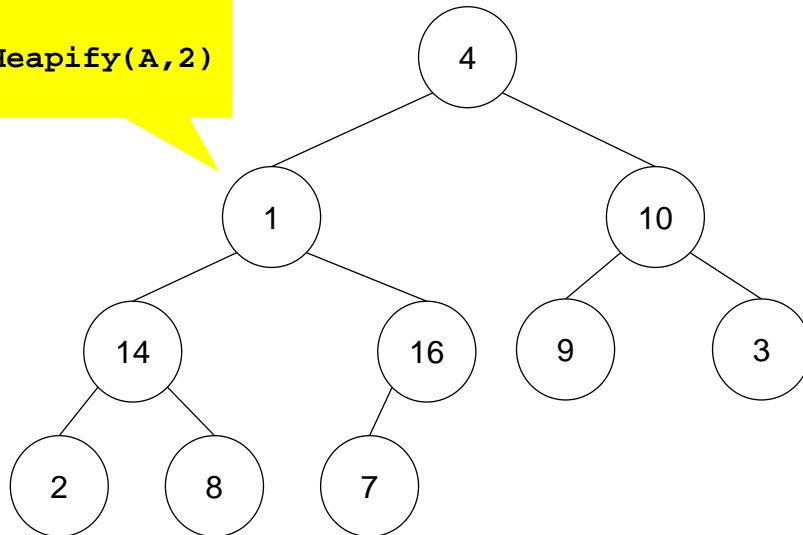


27

a.a. 2001/200.

Esempio (7)

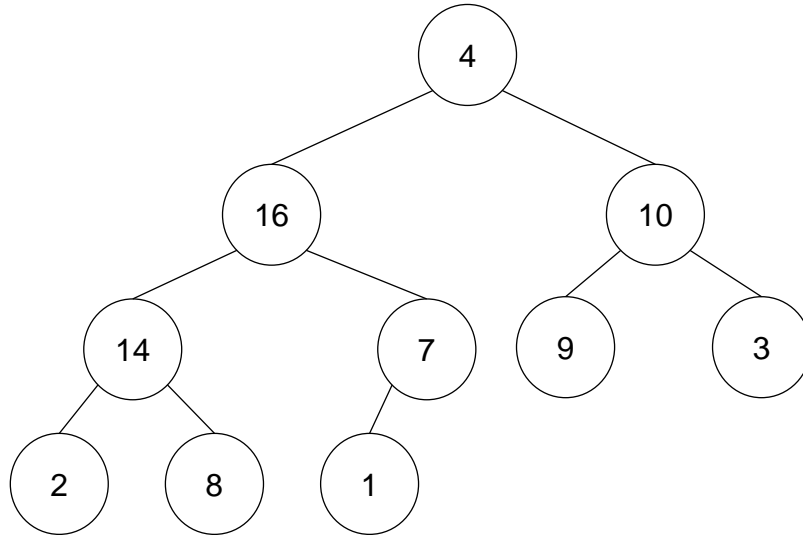
Heapify(A, 2)



28

a.a. 2001/200.

Esempio (8)

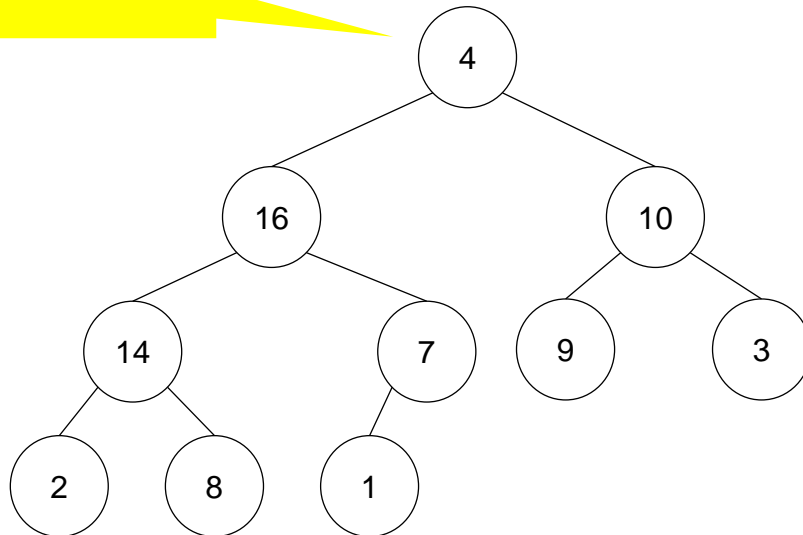


29

a.a. 2001/2002

Heapify(A,1)

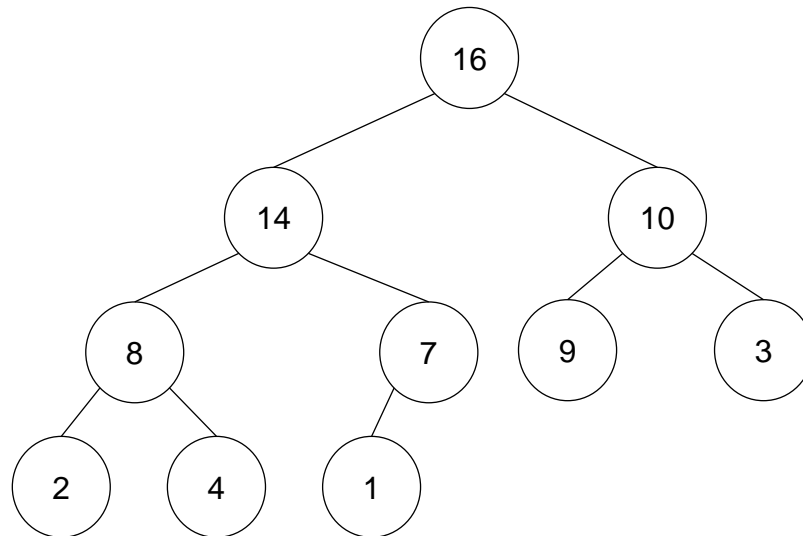
Esempio (9)



30

a.a. 2001/2002

Esempio (10)



31

a.a. 2001/200.

Complessità

È facile verificare che per trasformare in heap un albero binario di n elementi bisogna chiamare `Heapify` $n/2$ volte, e ciascuna di queste chiamate ha una complessità sicuramente inferiore a $O(\lg n)$.

Quindi la complessità di `Build-heap` è sicuramente $O(n \lg n)$.

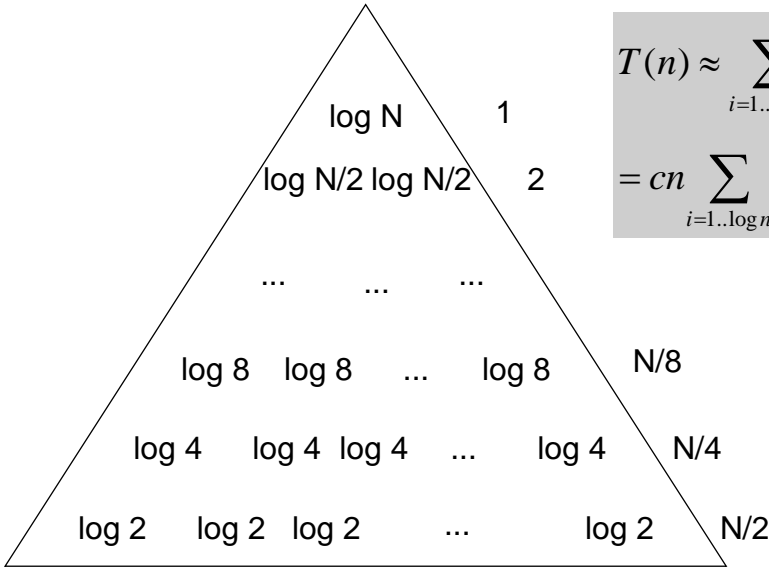
In realtà si può dimostrare che la complessità di `Build-heap` è $O(n)$.

32

a.a. 2001/200.



Giustificazione



$$T(n) \approx \sum_{i=1..log n} \frac{n}{2^i} \log 2^i =$$

$$= cn \sum_{i=1..log n} \frac{i}{2^i} \approx \dots \approx O(n)$$

33

a.a. 2001/200.

Sommario

- **Gli heap**
- **L'algorithmo Heapsort**
- **Le code con priorità.**

34

a.a. 2001/200.



Heapsort

Sfruttando le proprietà degli heap e le operazioni su di essi definite è possibile definire un algoritmo di ordinamento di vettori.

Heap sort si basa sull'iterazione dei seguenti passi:

1. Trasforma il vettore in uno heap
2. Scambia il primo elemento (che è sicuramente il più grande) con l'ultimo
3. Riduci la dimensione dello heap di 1
4. Ripeti da 1.

35

a.a. 2001/200.

Heapsort

HEAPSORT(A)

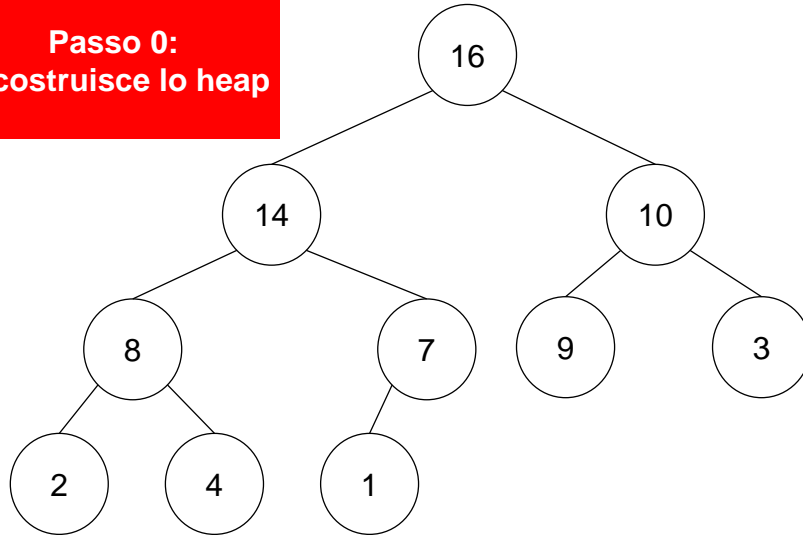
```
1   BUILD-HEAP( $A$ )
2   for  $i \leftarrow \text{length}[A]$  downto 2
3       do scambia  $A[1] \leftrightarrow A[i]$ 
4            $\text{heap-size}[A] \leftarrow \text{heap-size}[A] - 1$ 
5       HEAPIFY( $A, 1$ )
```

36

a.a. 2001/200.

Esempio (0)

Passo 0:
si costruisce lo heap

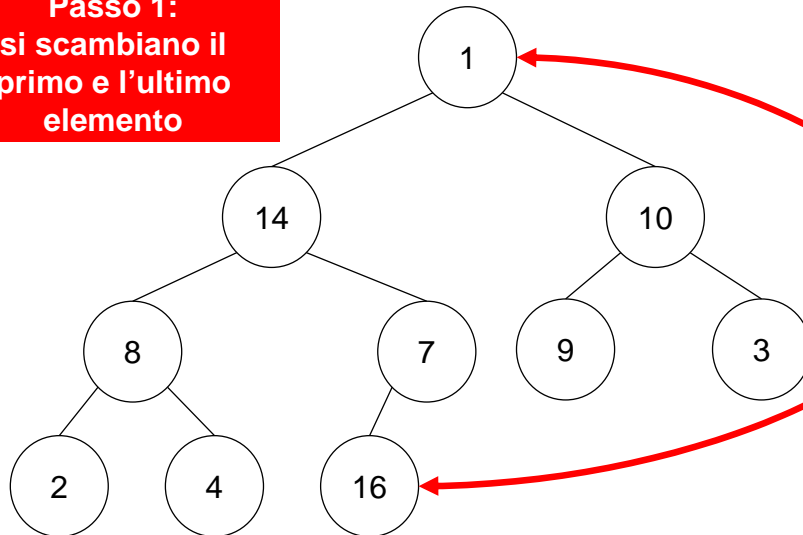


37

a.a. 2001/2002

Esempio (1)

Passo 1:
si scambiano il
primo e l'ultimo
elemento

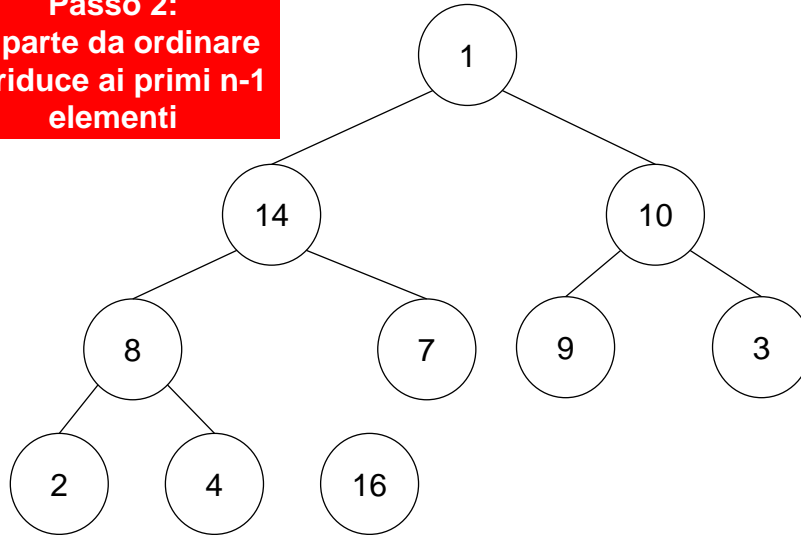


38

a.a. 2001/2002

Esempio (2)

Passo 2:
la parte da ordinare
si riduce ai primi n-1
elementi

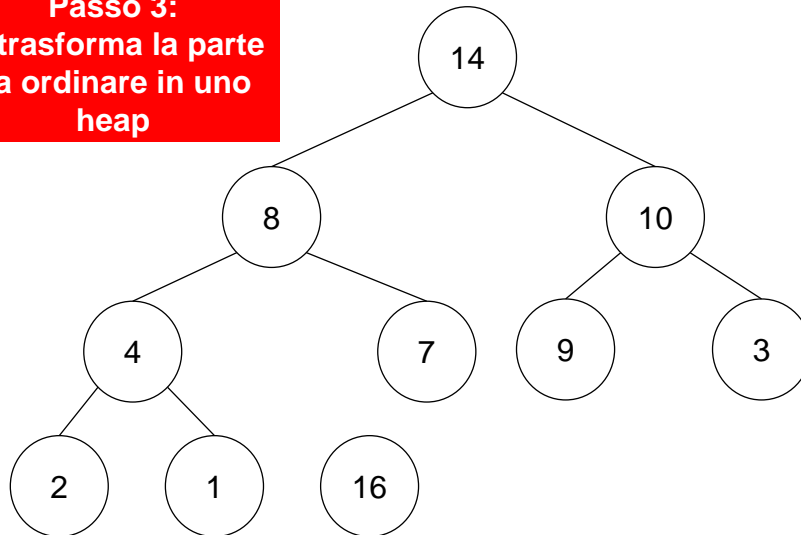


39

a.a. 2001/2002

Esempio (3)

Passo 3:
si trasforma la parte
da ordinare in uno
heap

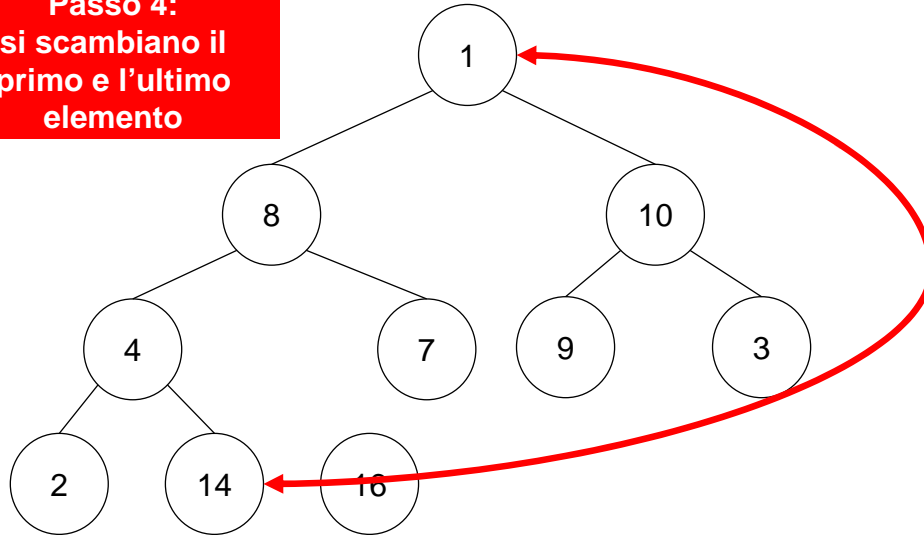


40

a.a. 2001/2002

Esempio (4)

Passo 4:
si scambiano il
primo e l'ultimo
elemento

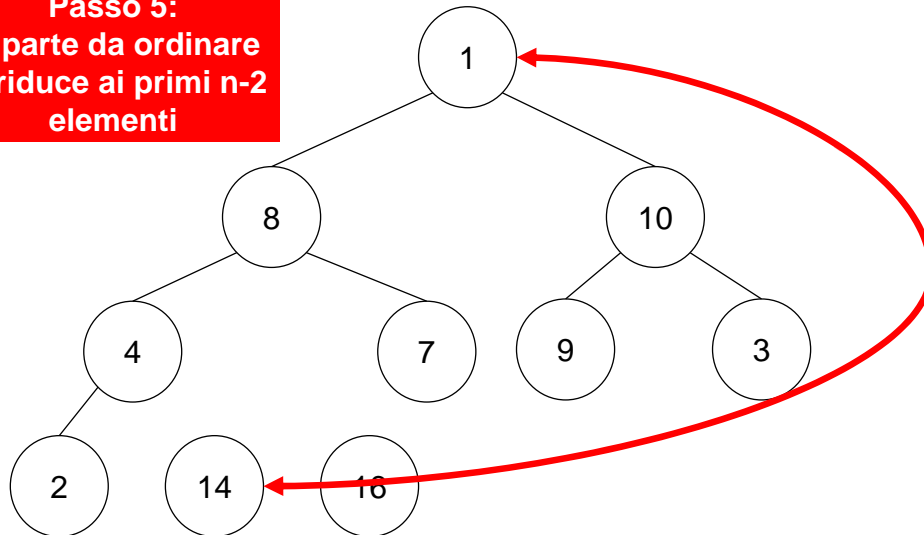


41

a.a. 2001/2002

Esempio (5)

Passo 5:
la parte da ordinare
si riduce ai primi $n-2$
elementi

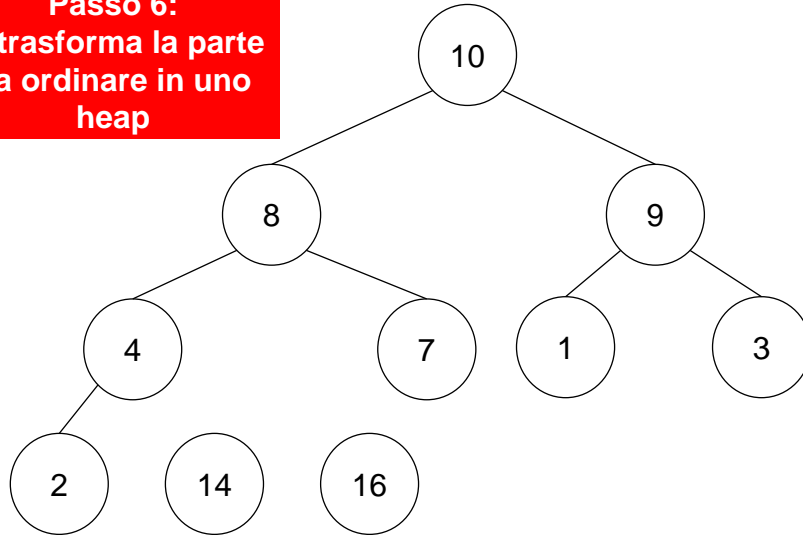


42

a.a. 2001/2002

Esempio (6)

Passo 6:
si trasforma la parte
da ordinare in uno
heap

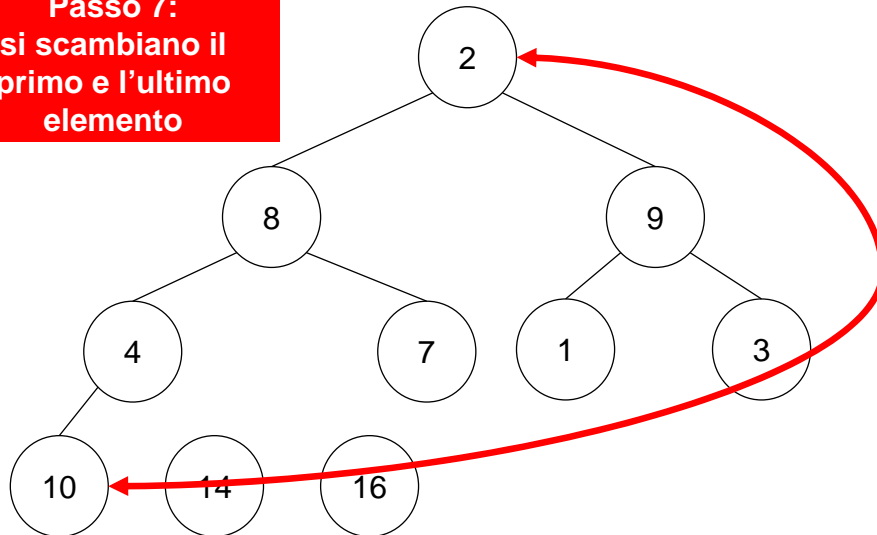


43

a.a. 2001/2002

Esempio (7)

Passo 7:
si scambiano il
primo e l'ultimo
elemento

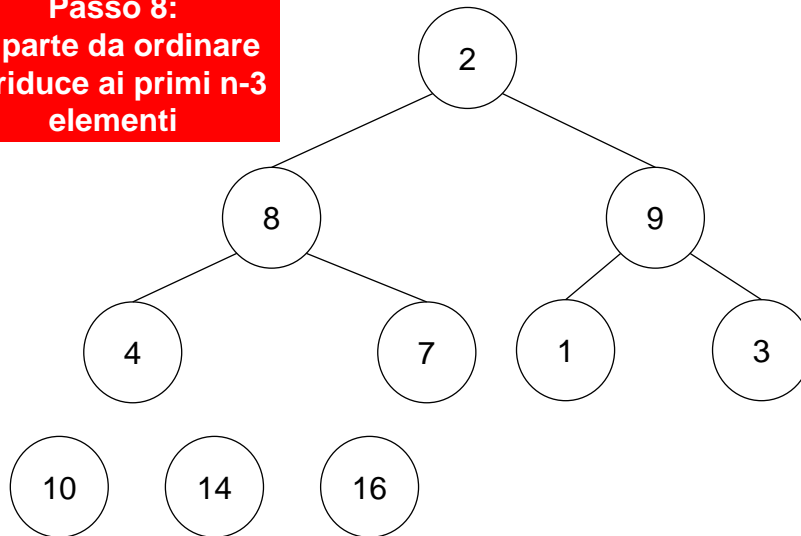


44

a.a. 2001/2002

Esempio (8)

Passo 8:
la parte da ordinare
si riduce ai primi $n-3$
elementi

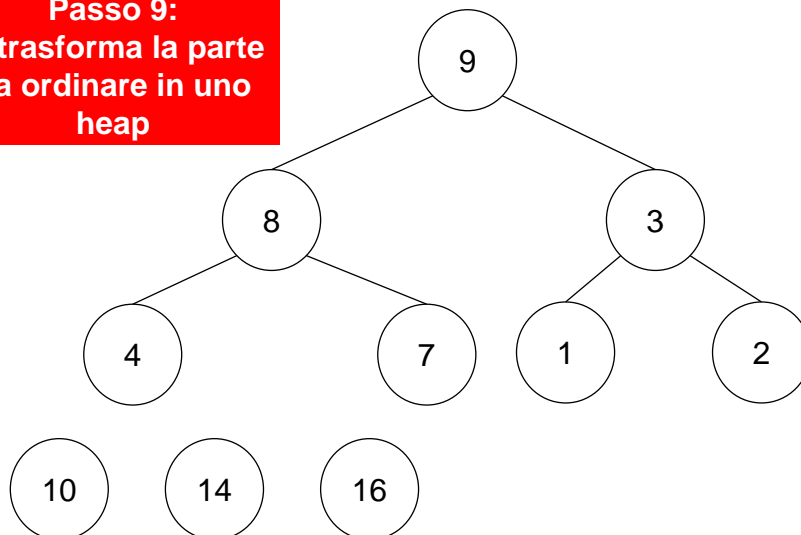


45

a.a. 2001/200.

Esempio (9)

Passo 9:
si trasforma la parte
da ordinare in uno
heap

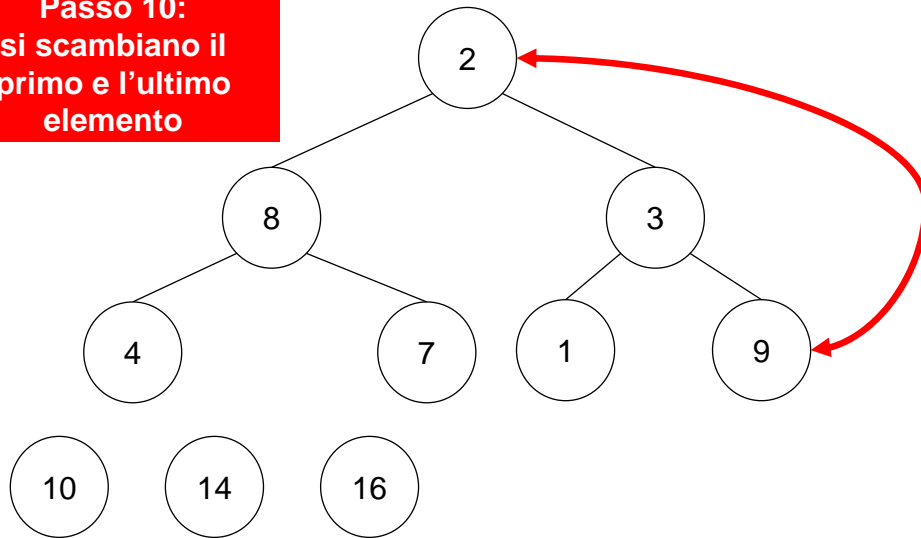


46

a.a. 2001/200.

Esempio (10)

Passo 10:
si scambiano il
primo e l'ultimo
elemento

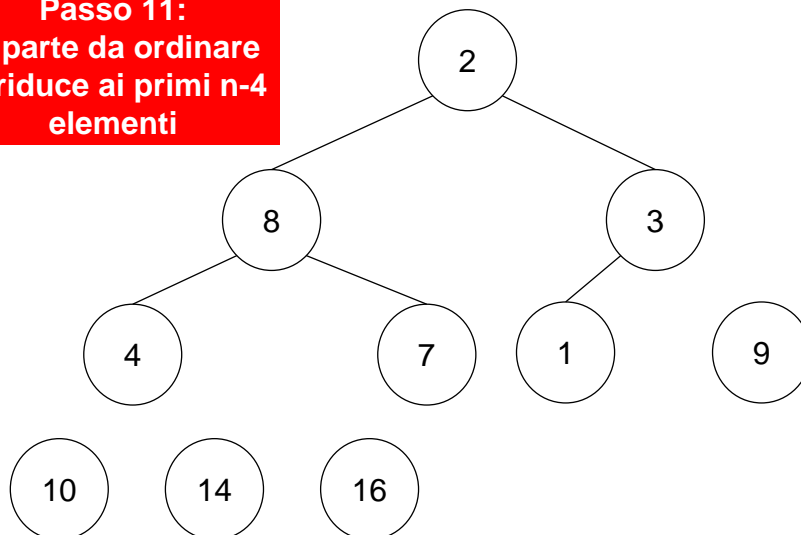


47

a.a. 2001/2002

Esempio (11)

Passo 11:
la parte da ordinare
si riduce ai primi $n-4$
elementi

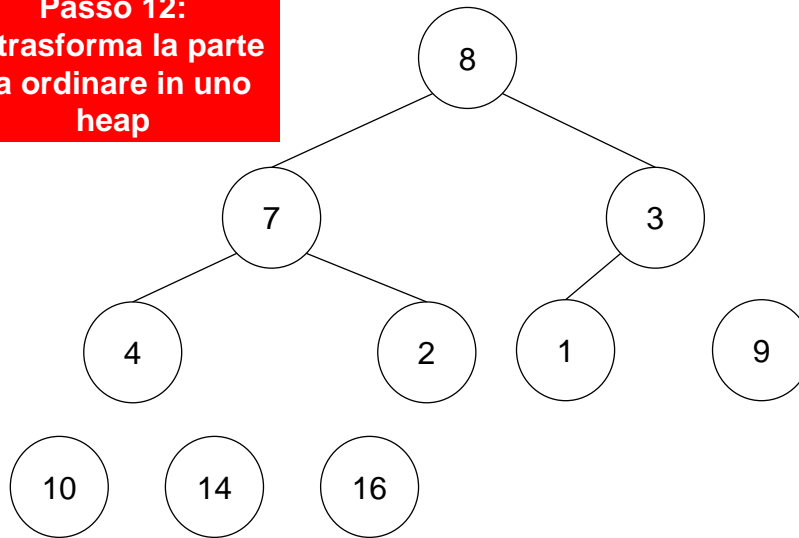


48

a.a. 2001/2002

Esempio (12)

Passo 12:
si trasforma la parte
da ordinare in uno
heap

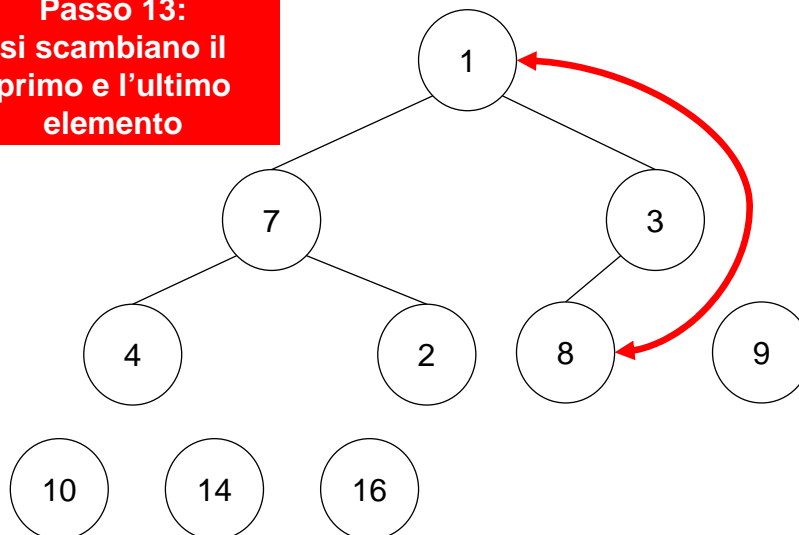


49

a.a. 2001/200.

Esempio (13)

Passo 13:
si scambiano il
primo e l'ultimo
elemento

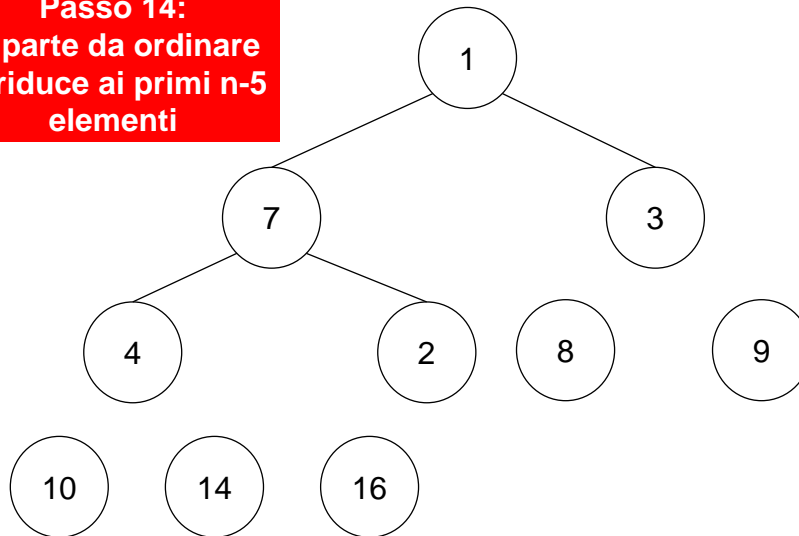


50

a.a. 2001/200.

Esempio (14)

Passo 14:
la parte da ordinare
si riduce ai primi n-5
elementi

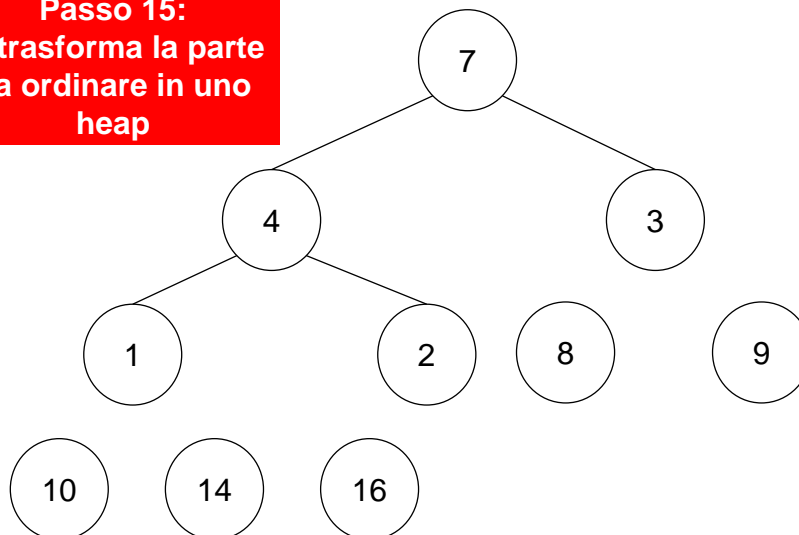


51

a.a. 2001/2002

Esempio (15)

Passo 15:
si trasforma la parte
da ordinare in uno
heap

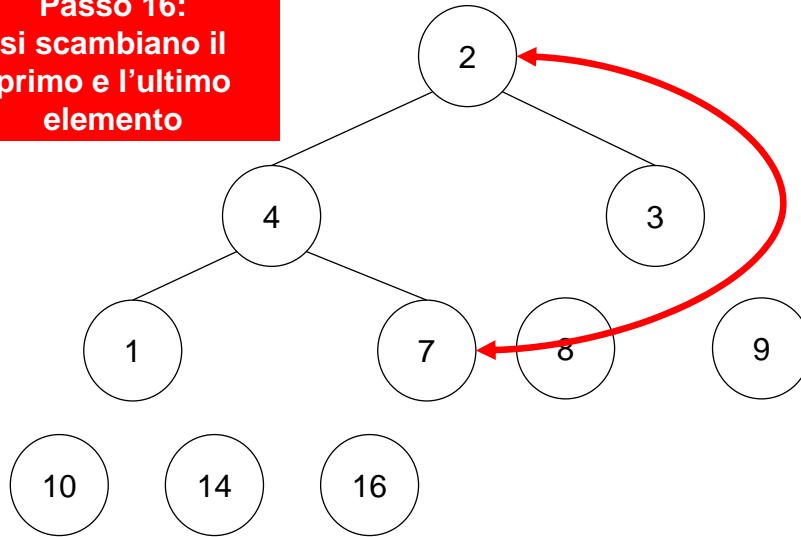


52

a.a. 2001/2002

Esempio (16)

Passo 16:
si scambiano il
primo e l'ultimo
elemento

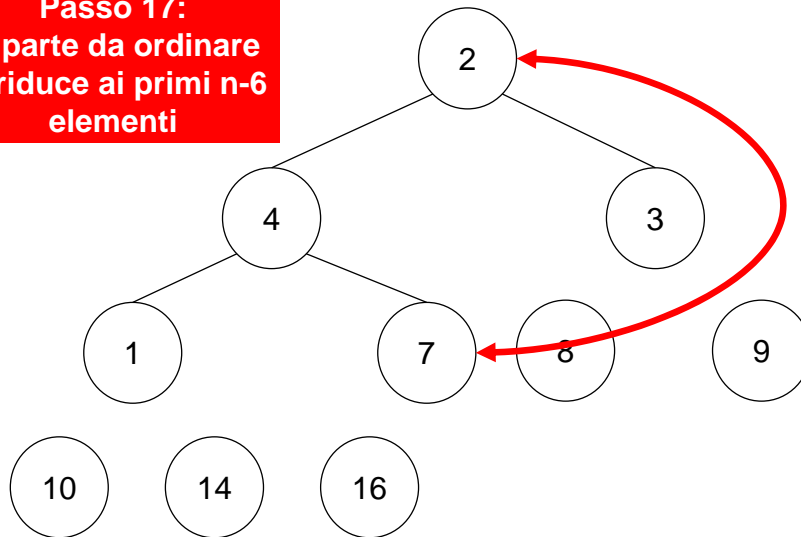


53

a.a. 2001/2002

Esempio (17)

Passo 17:
la parte da ordinare
si riduce ai primi $n-6$
elementi



54

a.a. 2001/2002



Complessità

L'algoritmo di heapsort richiama $n-1$ volte la procedura `Heapify`, che ha complessità $O(\lg n)$.

Di conseguenza la complessità dell'algoritmo è $O(n \lg n)$.

55

a.a. 2001/2002

Sommario

- Gli heap
- L'algoritmo Heapsort
- Le code con priorità.

56

a.a. 2001/2002



Code con priorità

Una coda con priorità è una struttura dati su cui sono definite le seguenti operazioni:

- **INSERT(S, x)**: inserisce l'elemento x nell'insieme S
- **MAXIMUM(S)**: restituisce l'elemento di S con chiave massima
- **EXTRACT-MAX(S)**: restituisce e rimuove da S l'elemento di S con chiave massima.

Le code con priorità trovano numerose applicazioni pratiche.

57

a.a. 2001/200.

Code con priorità e heap

Gli heap sono le strutture più adatte ad implementare delle code con priorità.

L'operazione **MAXIMUM(S)** corrisponde semplicemente a restituire l'elemento radice.

L'operazione **INSERT(S,x)** corrisponde ad un normale inserimento in uno heap, e può essere eseguita tramite la seguente procedura **Heap-Insert**.

L'operazione **EXTRACT-MAX(S)** corrisponde all'eliminazione della radice da uno heap, e può essere eseguita tramite la seguente procedura **Heap-Extract-Max**.

58

a.a. 2001/200.

Heap-Insert

Complessità:
 $O(\lg n)$

HEAP-INSERT(A, key)

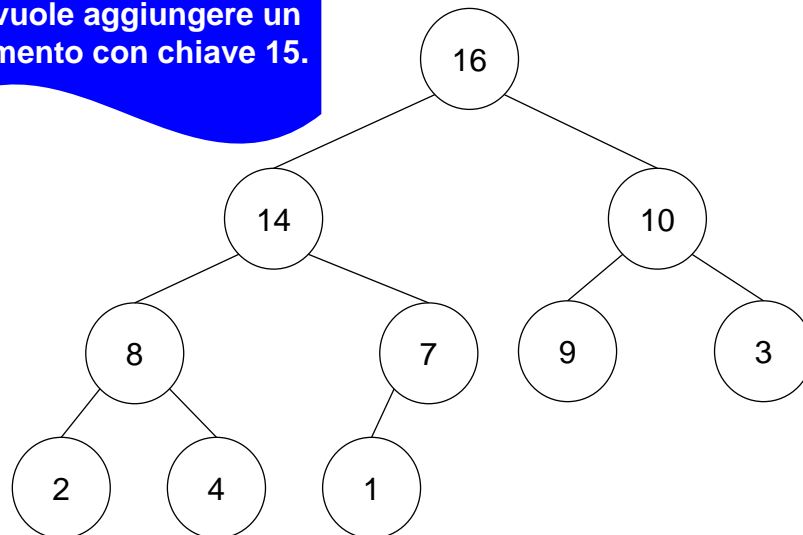
```
1   $heap-size[A] \leftarrow heap-size[A] + 1$ 
2   $i \leftarrow heap-size[A]$ 
3  while  $i > 1$  e  $A[PARENT(i)] < key$ 
4      do  $A[i] \leftarrow A[PARENT(i)]$ 
5           $i \leftarrow PARENT(i)$ 
6   $A[i] \leftarrow key$ 
```

59

a.a. 2001/2002

Esempio (0)

Si vuole aggiungere un
elemento con chiave 15.

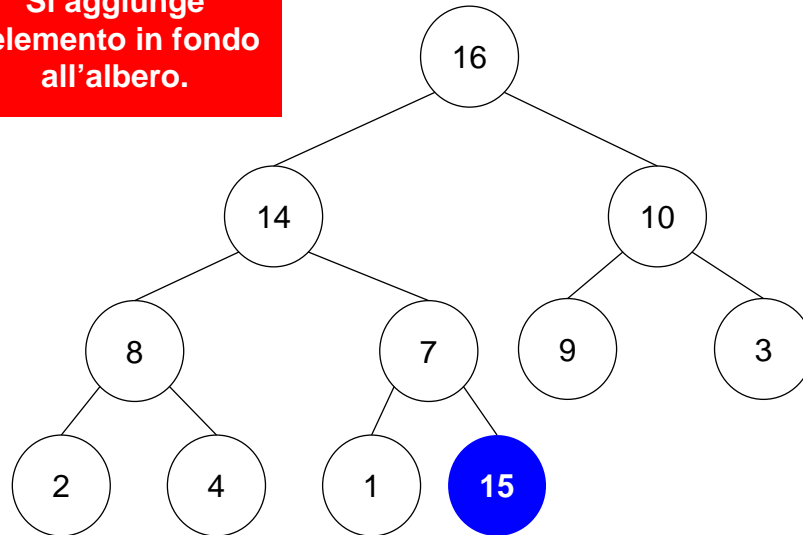


60

a.a. 2001/2002

Esempio (1)

Si aggiunge
l'elemento in fondo
all'albero.

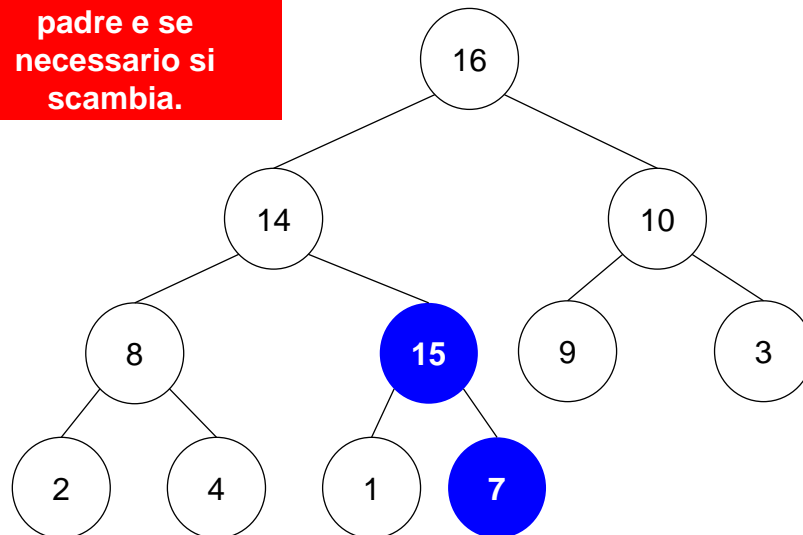


61

a.a. 2001/2002

Esempio (2)

Si confronta con il
padre e se
necessario si
scambia.

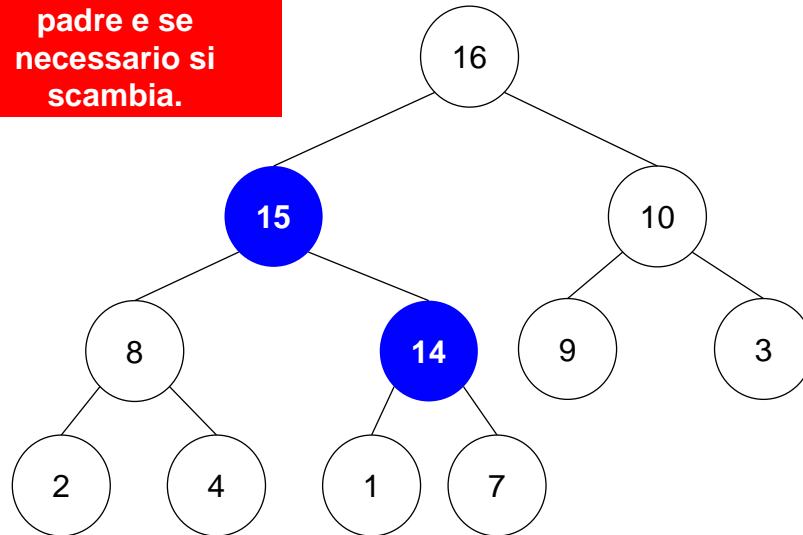


62

a.a. 2001/2002

Esempio (3)

Si confronta con il
padre e se
necessario si
scambia.



63

a.a. 2001/2002

Heap-Extract-Max

HEAP-EXTRACT-MAX(*A*)

```
1  if heap-size[A] < 1
2  then error "heap underflow"
3  max ← A[1]
4  A[1] ← A[heap-size[A]]
5  heap-size[A] ← heap-size[A] - 1
6  HEAPIFY(A, 1)
7  return max
```

Complessità:
 $O(\lg n)$

64

a.a. 2001/2002