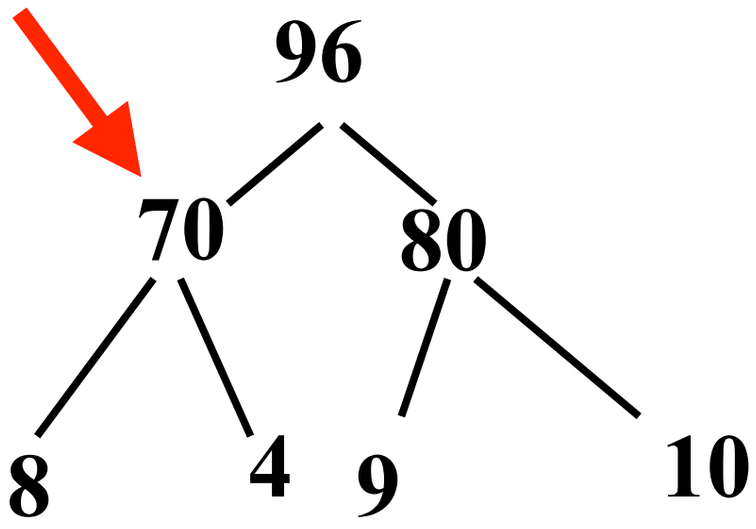
Esercizio 3° più grande

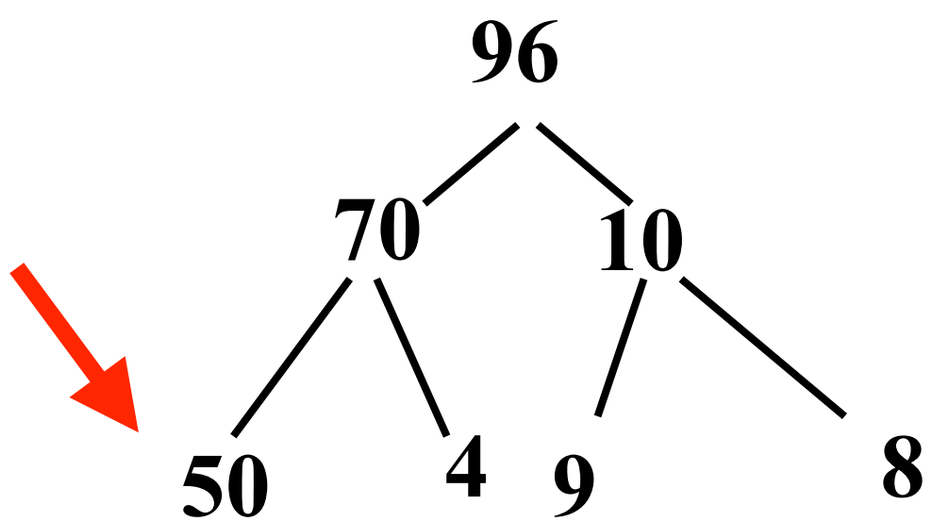
Sia dato un max-heap H di n interi. Dove può trovarsi il terzo intero più grande?
Si progetti un algoritmo per trovare il terzo intero più grande in tempo costante.
Discutere la correttezza dell'algoritmo proposto.

Esercizio 3° più grande

Sia dato un max-heap H di n interi. Dove può trovarsi il terzo intero più grande?
Si progetti un algoritmo per trovare il terzo intero più grande in tempo costante.
Discutere la correttezza dell'algoritmo proposto.

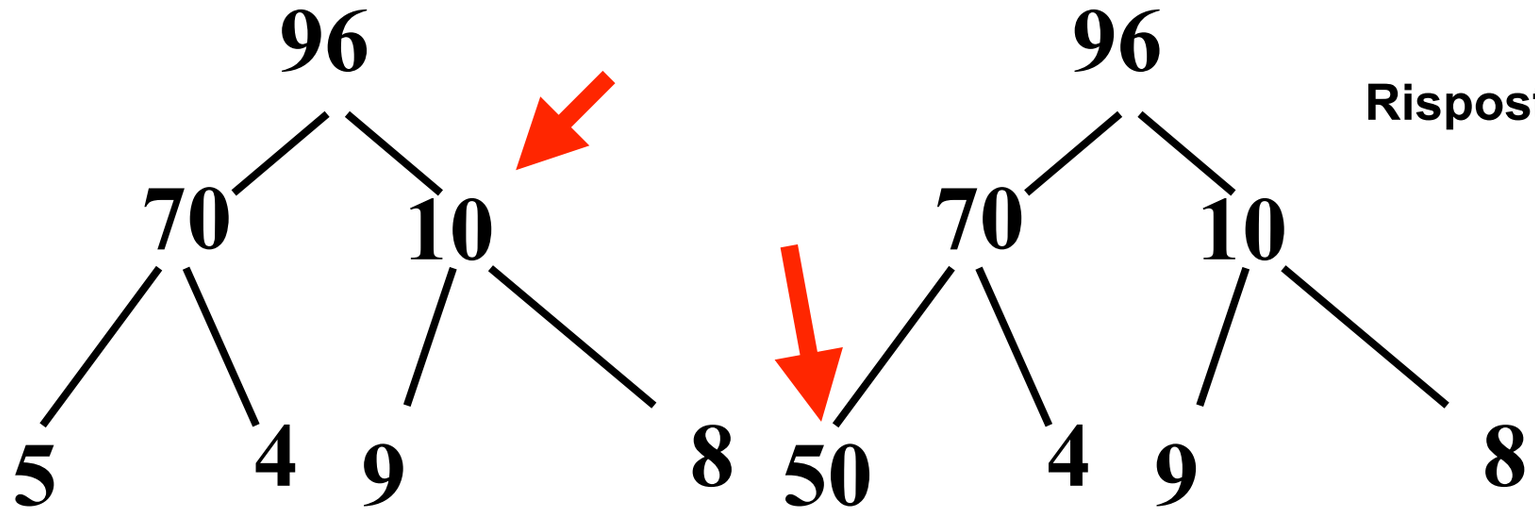
Risposta: il terzo intero più grande può trovarsi nel primo o nel secondo livello. Infatti un elemento del terzo livello ha almeno 2 elementi più grandi, quelli che determinano un cammino dall'elemento alla radice.





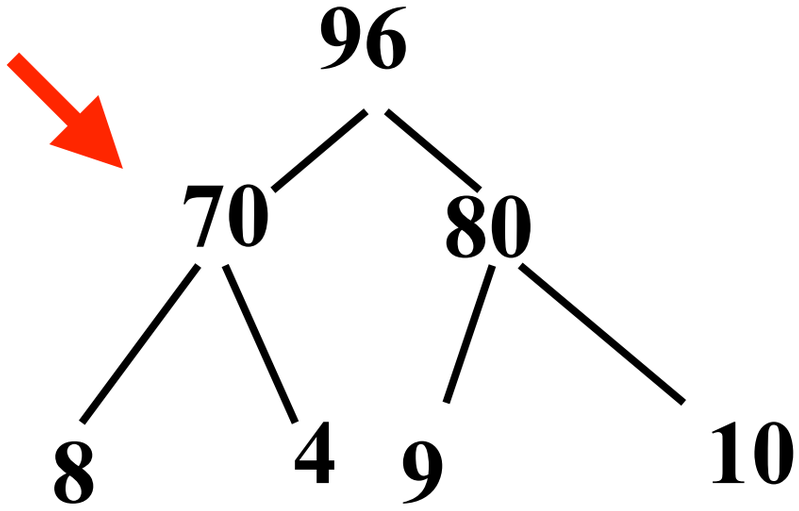
I casi con H.heapsize ≥ 7

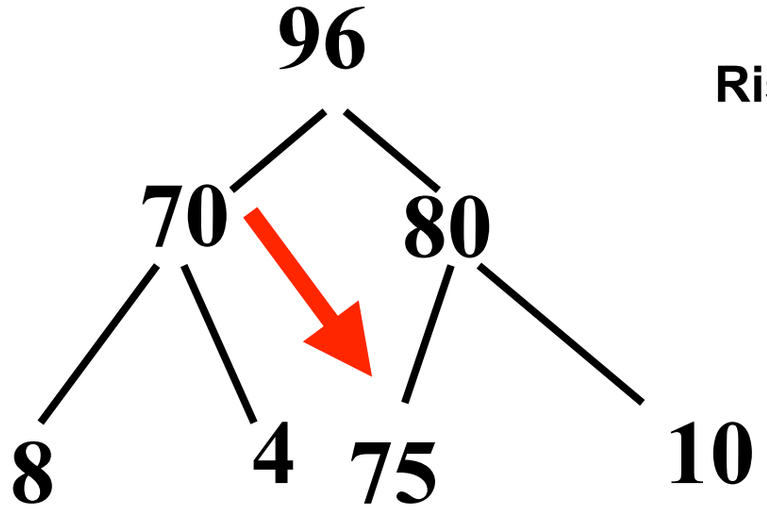
if H[2] > H[3]



Risposta: max(H[3], H[4],H[5])

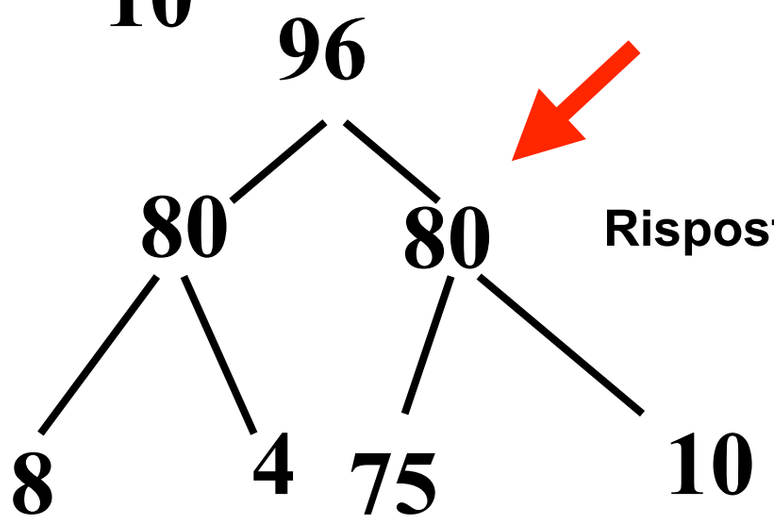
if H[2] < H[3]





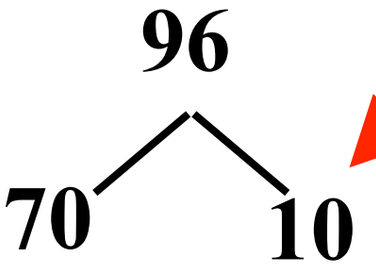
Risposta: max(H[2], H[6],H[7])

if H[2] = H[3]

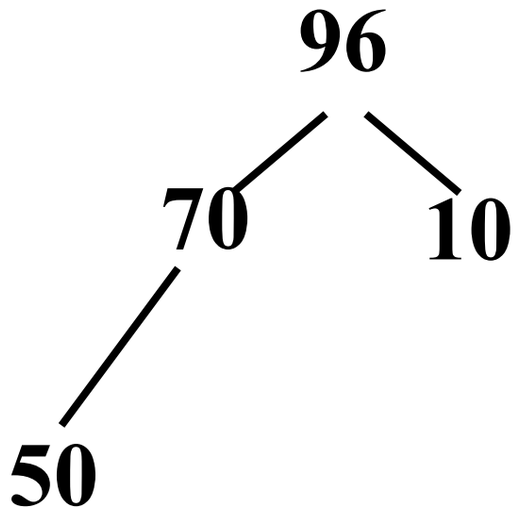


Risposta H[3]

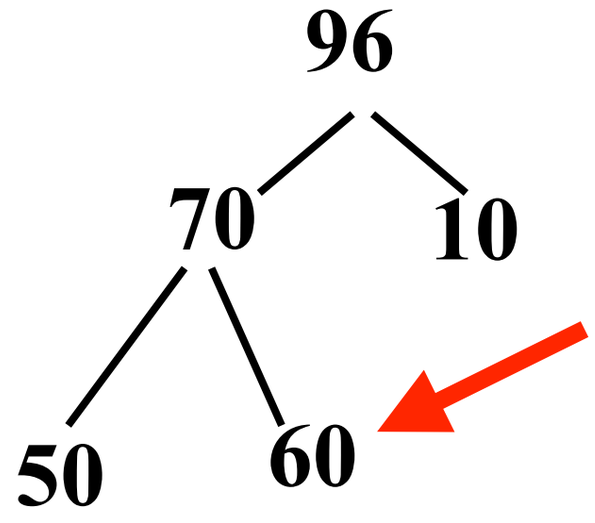
I casi con H.heapsize < 7



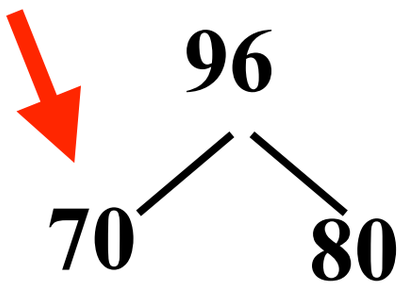
H[3]



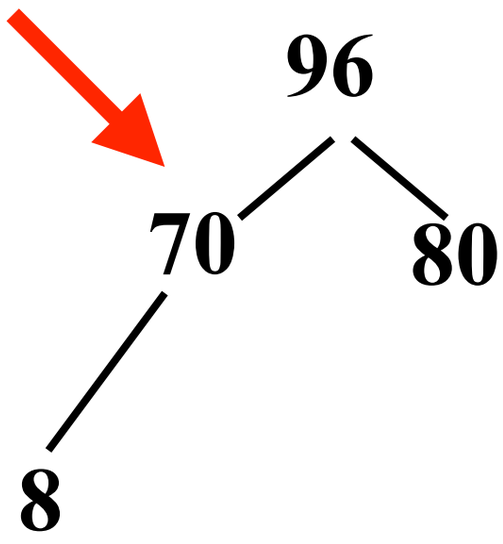
max(H[3], H[4])



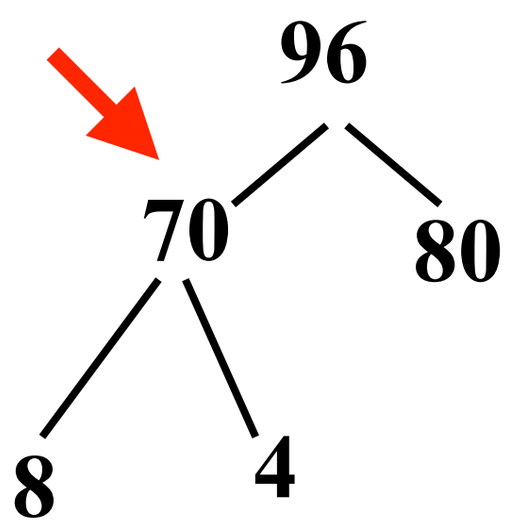
max(H[3], H[4],H[5])



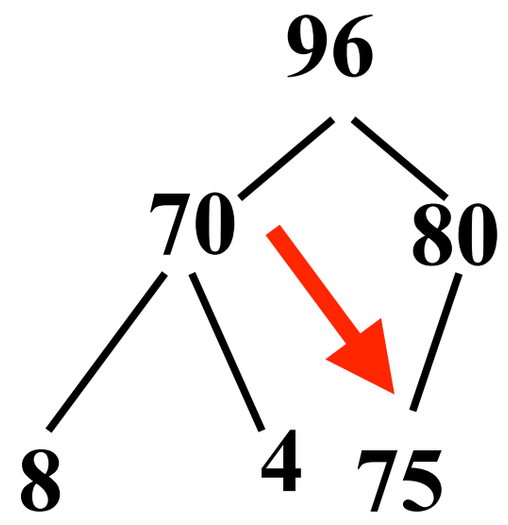
H[2]



H[2]



H[2]



max(H[2], H[6])

Pseudocodice per il 3° più grande

```
3PiùGrandeInMax-Heap (H)
input: un max-heap H
output: il terzo più grande in H
if H.heapsize < 3 then return “al più due elementi nel max-heap”
if H[2] > H[3] then  if H.heapsize ≥ 5 then return max(H[3], H[4],H[5])
                     if H.heapsize = 4 then return max(H[3], H[4])
                     if H.heapsize = 3 then return H[3]
if H[2] < H[3] then  if H.heapsize ≥ 7 then return max(H[2], H[6],H[7])
                     if H.heapsize = 6 then return max(H[2], H[6])
                     if H.heapsize ≤ 5 then  then return H[2]
if H[2] = H[3] then return H[3]
```

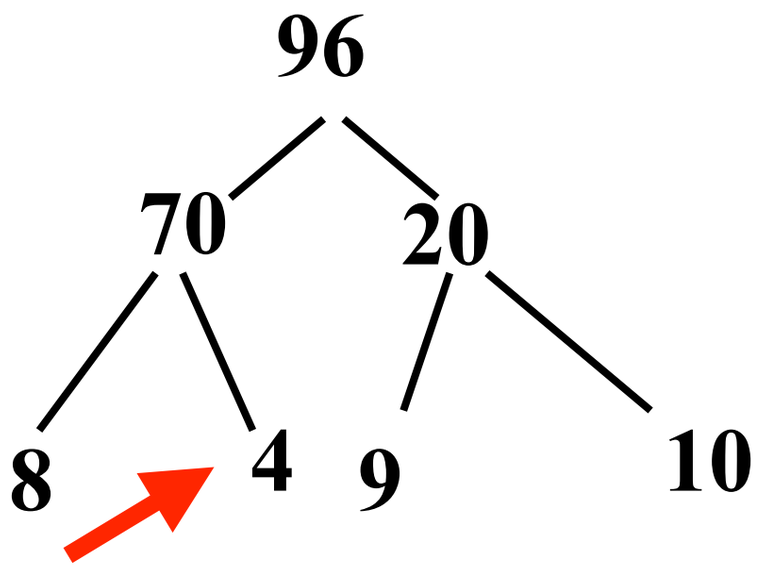
Esercizio il minimo

Dato un max-heap dove può trovarsi il minimo tra le chiavi memorizzate?
Si scriva un algoritmo iterativo per calcolare il minimo in un max-heap.

Esercizio il minimo

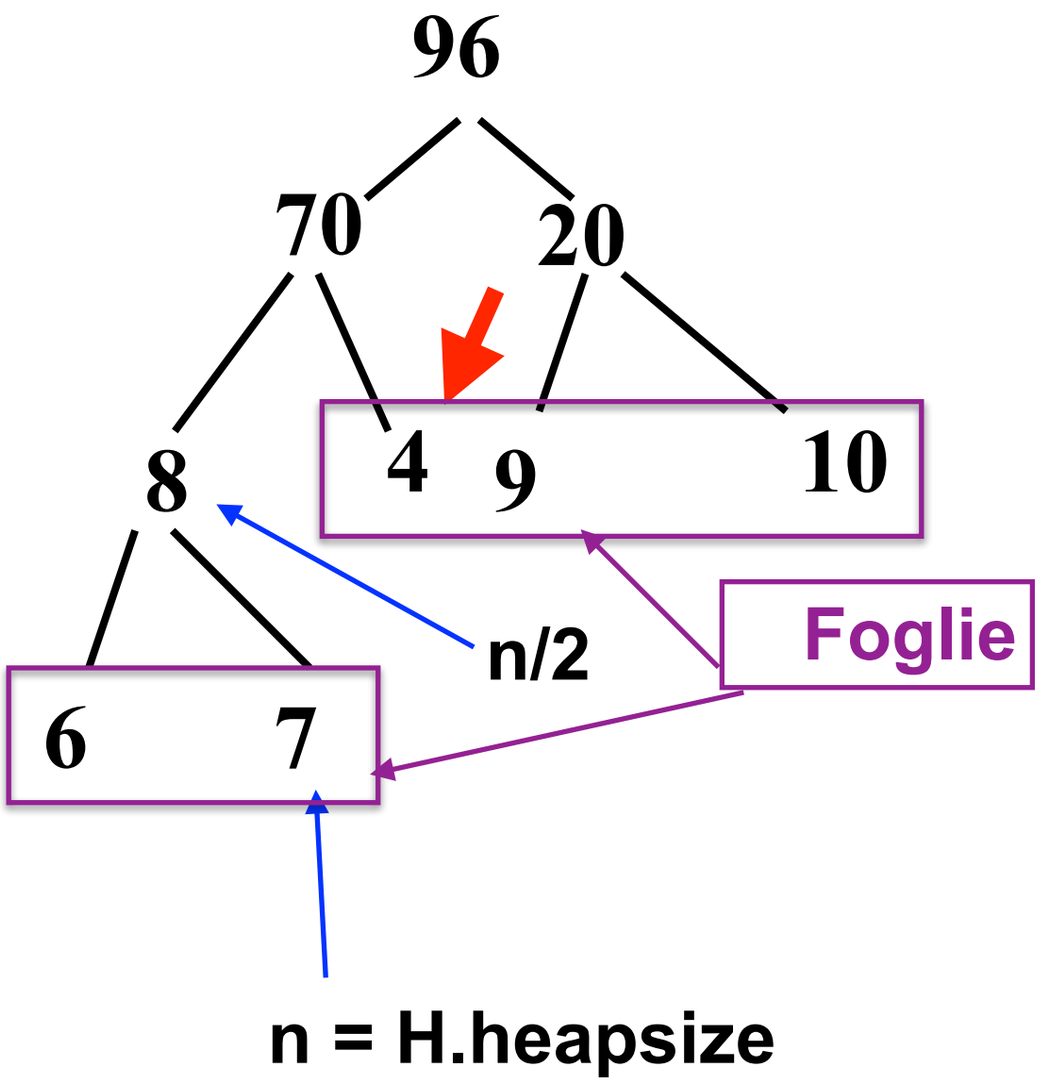
Dato un max-heap dove può trovarsi il minimo tra le chiavi memorizzate?
Si scriva un algoritmo iterativo per calcolare il minimo in un max-heap.

Soluzione: il minimo può trovarsi solo tra le foglie. Infatti i figli di ogni elemento sono più piccoli di lui.



Pseudocodice per il minimo

Soluzione: il minimo può trovarsi solo tra le foglie. Il primo nodo foglia sul penultimo livello è il successivo del padre dell'ultima foglia: H[n/2 + 1]




```
MinimoInMax-Heap (H)
input: un max-heap H
output: il minimo in H
n= H.heapsize
minInd = n/2+1
for i = n/2+2 to n do
if H[minInd] > H[i] then minInd = i
return minInd
```

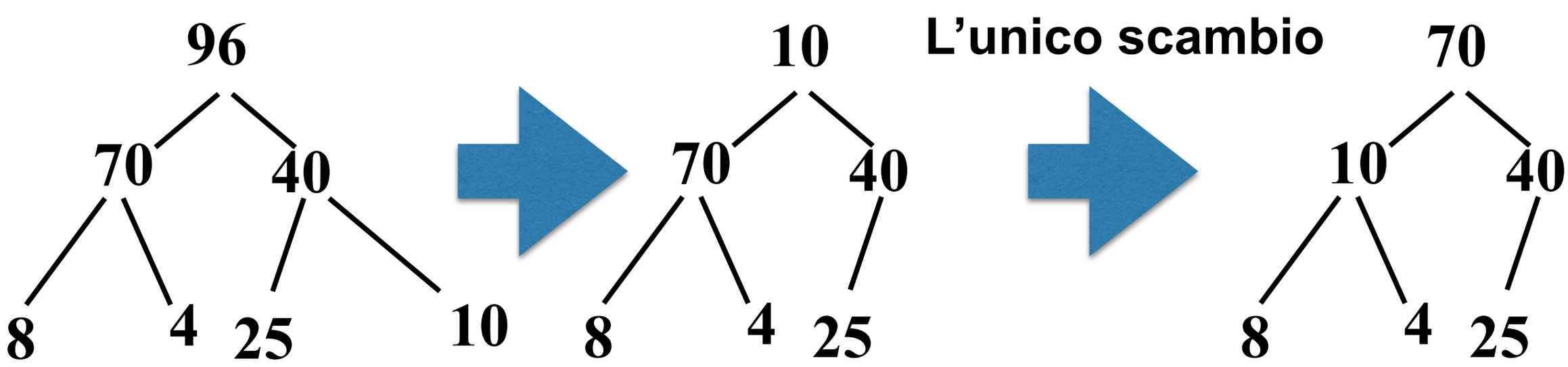
$n = 2^{h+1} - 1,\ n/2 = 2^h$

Esercizio estrazione massimo

Dato un MaxHeap di n > 2 elementi diversi, qual'è il minimo numero di scambi padre-figlio necessari nella rimozione del massimo, indipendentemente dal valore di n?
Si dia un esempio di MaxHeap con n = 7 elementi in cui si realizza tale minimo.

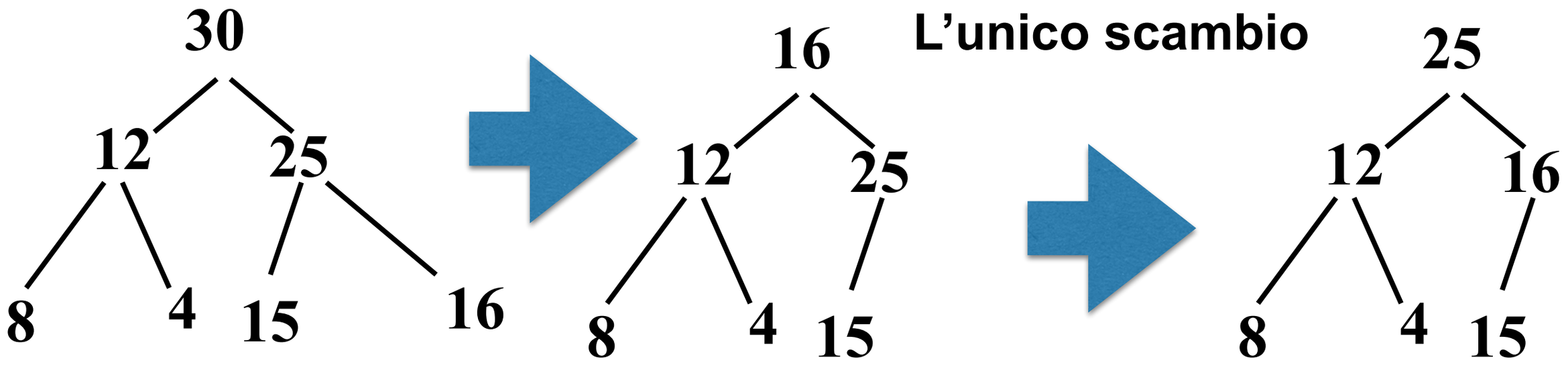
Esercizio Heap: tutti diversi

Soluzione: basterà un solo scambio padre figlio se la foglia più a destra è nel sotto albero destro, la radice del sotto albero sinistro ha un valore maggiore di tutti gli elementi nel sotto albero destro mentre quelli nel sotto albero sinistro diversi dalla sua radice sono minori dell'elemento nell'ultima foglia.
Esempio con n= 7, estrazione del massimo:



Esercizio Heap: tutti diversi

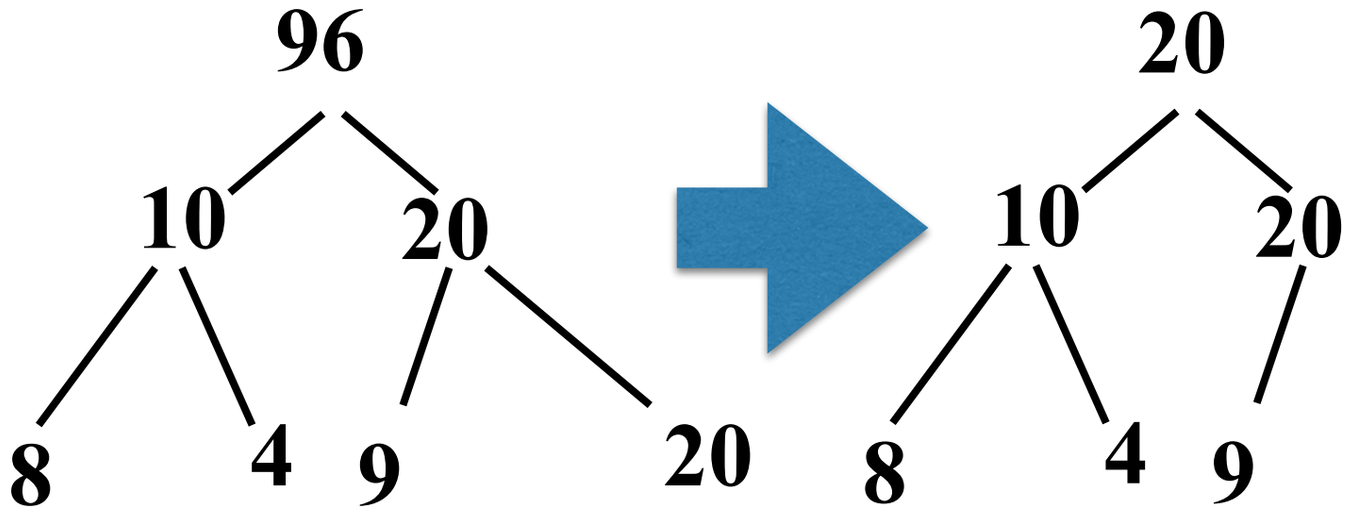
Nel caso di n = 7, si possono dare anche altri casi:



Ma questo caso non si generalizza a ogni dimensione del max-heap, perchè scendendo di un livello la foglia più a destra è più piccola ed essendo lo scambio sullo stesso cammino saranno necessari altri scambi.

Esercizio Heap: non tutti diversi

Risposta: non servirà alcuno scambio padre figlio se la foglia più a destra è nel sotto albero destro, il cammino da questa foglia alla radice è fatto di nodi tutti con lo stesso valore che è maggiore o uguale di quello del figlio sinistro della radice.
Esempio con n=7, estrazione del massimo:



Nessuno scambio

Gli m più grandi

Si consideri il problema di determinare gli m elementi più grandi in un array di n elementi.
Si confrontino dal punto di vista del tempo di esecuzione asintotico gli algoritmi qui sotto proposti.

1 Si ordinano gli elementi

2 Si costruisce un maxheap dall'array e poi si estrae per m volte il massimo.

l'analisi

1 Si ordinano gli elementi

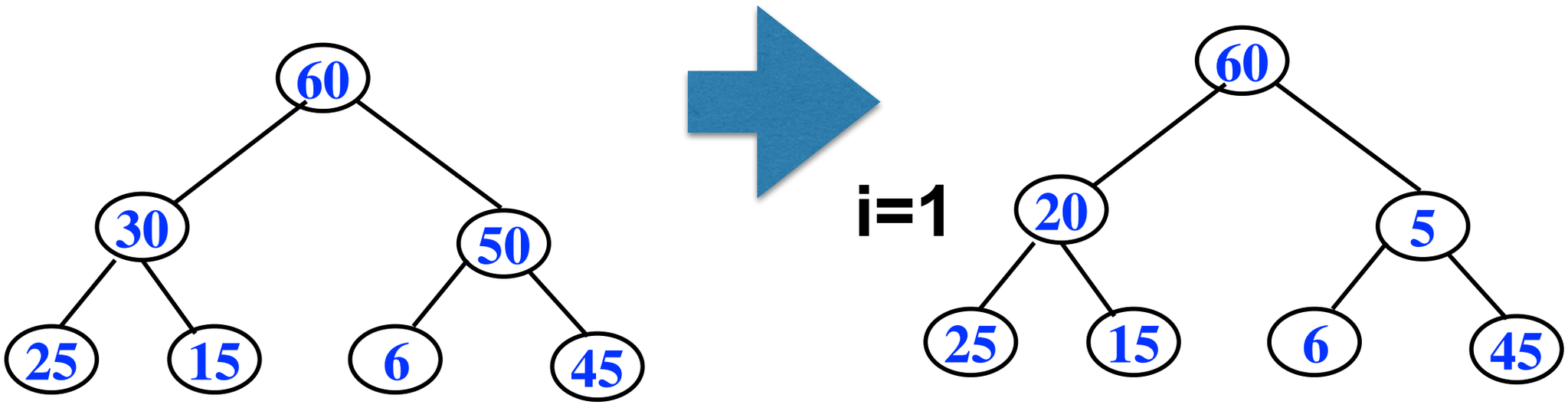
Il meglio che possiamo fare è in Θ(n lg n), poi si scorrono gli ultimi m, in tempo Θ(m), quindi in totale abbiamo un andamento in Θ(n lg n), visto che m ≤ n.

2 Si costruisce un maxheap dall'array e poi si estrae per m volte il massimo.

O(n) per la Build-Max-Heap
O(m lg n) per l'estrazione degli m elementi più grandi.
In totale abbiamo un andamento in O(n)+O(m lg n).
Poiché m ≤ n allora il tempo è in O(n lg n),
ma se m = O(lg n) allora il tempo è in O(n) perchè
$lg^2 n = O(n)$

MaxHeap modificato

Si supponga che un maxheap H di altezza h sia stato modificato in modo tale che gli elementi di un livello i, con 0 < i < h, siano stati sostituiti con elementi più piccoli di quelli originali del maxheap.
Si definisca un'algoritmo Ripristina(H,i) che ristabilisce la proprietà di essere Max-Heap, eventualmente violata dalla modifica sopra specificata.
Si valuti il tempo di esecuzione asintotico dell'algoritmo presentato. Esempio:



Soluzione

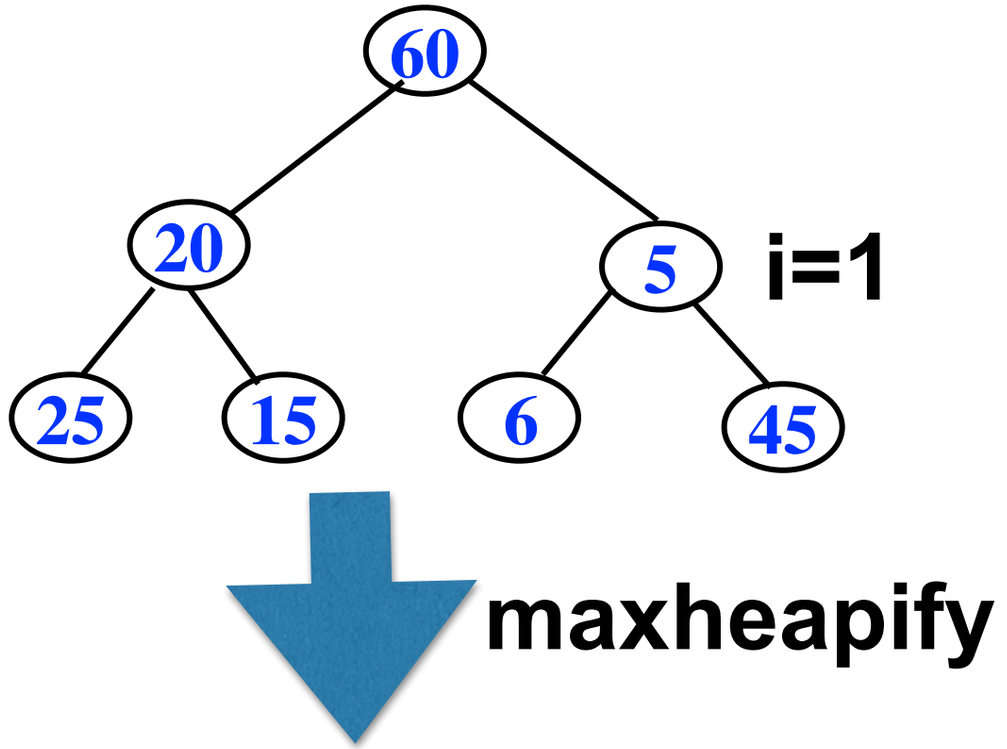
Poiché per j nel livello i deve valere che H[j] ≥ H[2j] e H[j] ≥ H[2j+1], oltre che H[j] ≤ H[j/2], si ha che se si è inserito un valore più piccolo di H[j] nella posizione j, la proprietà può essere violata rispetto ai figli e non rispetto al padre.

Poiché il resto dell'array non è modificato e H[2j] e H[2j+1] sono radici di un max-heap, si può utilizzare la Max-Heapify, su tutti i nodi del livello i.

Se i non è il livello massimo né il penultimo questi nodi sono quelli il cui indice è compreso tra 2^i e $2^{i+1}-1$.

Se i fosse il livello massimo la diminuzione dei valori non avrebbe conseguenze visto che questi nodi sono foglie.

Se il livello i fosse il penultimo, basterà chiamare la Maheapify sui nodi che hanno figli, cioè fino a n/2, se n è il numero degli elementi di H.



analisi

Poiché la Max-Heapify termina nel caso peggiore in tempo asintotico pari all'altezza dell'albero su cui è chiamata, dobbiamo considerare le 2^i chiamate di Max-heapify su sottoalberi di altezza h-i, supponendo il livello i pieno e dove h è l'altezza dell'albero. Sia n il numero dei nodi di H.

Quindi si ha $T(n,i) = O((\lg n - i)*2^i)$, dove $0 \leq i < \lg n$

Se i=0, si ha un tempo in O(lg n) perchè c'è un'unica chiamata alla radice, se i=h-1 si ha un tempo in O(n) perchè ci sono al più 2^{h-1} chiamate ciascuna su un albero di altezza 1.

Se i = h/2 si ha un tempo $O(\lg n\, n^{1/2})$.

Pseudocodice

Ripristina(H,i)

input: un array H e un indice i, H è un maxheap nel quale gli elementi del livello i sono stati diminuiti

prec: $0 \leq i < \lfloor \lg(H.heapsize) \rfloor$

output: la proprietà del maxheap è ripristinata per H

if $i < \lfloor \lg(H.heapsize) \rfloor - 1$ then $n = 2^{i+1} - 1$

if $i = \lfloor \lg(H.heapsize) \rfloor - 1$ then $n = H.heapsize/2$

for $j = 2^i$ to n do

Max-Heapify (H,j)

Max-Heapify (A,i)

Input: A è un array e i è un indice

Prec: A[left(i)] e A[right(i)] sono radici di max-heap mentre A[i] può essere più piccolo dei suoi figli

Postc: A[i] è radice di un max-Heap