HEAPSORT E STRUTTURA DATI "HEAP"

ALGORITMO	COMPLESSITÀ	SUL POSTO
INSERTION-SORT	$O(n^2)$	SI
MERGE-SORT	$\Theta(n \log n)$	NO
HEAPSORT	$\Theta(n \log n)$	SI

L'ALGORITMO HEAPSORT E' BASATO SULLA STRUTTURA DATI HEAP, PER LA GESTIONE EFFICIENTE DI CODE DI PRIORITA',

HEAP

UN HEAP (BINARIO) È UNA STRUTTURA DATI BASATA SU ALBERI BINARI QUASI COMPLETI (RAPPRESENTATI IN MANIERA EFFICIENTE MEDIANTE ARRAY) SODDISFACENTI UNA PROPRIETÀ DELL'HEAP:

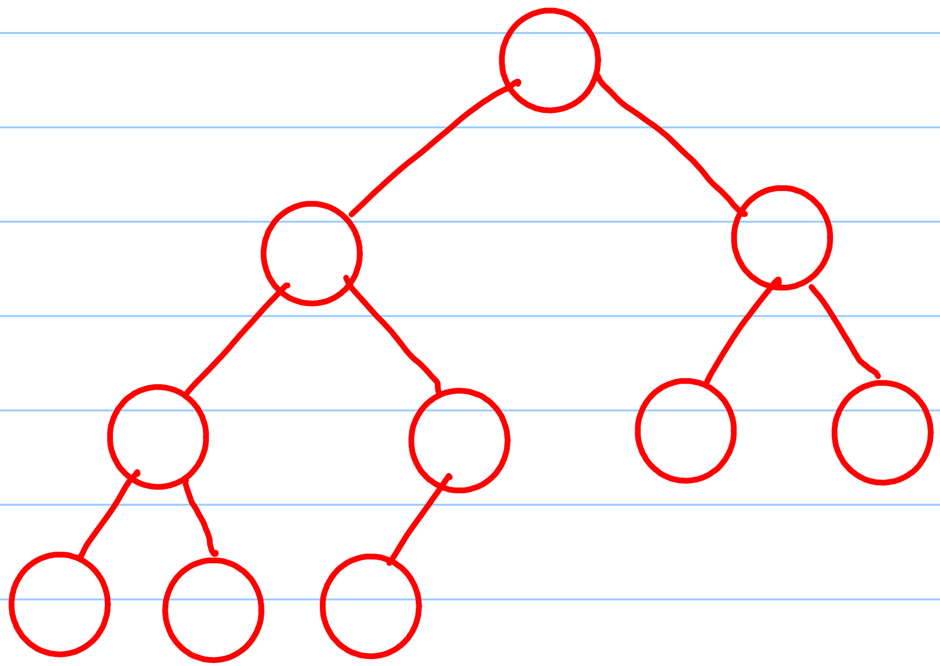
PROPRIETÀ DEL MAX-HEAP: IL VALORE DI UN NODO È MINORE O UGUALE AL VALORE DEL PADRE (SE ESISTENTE)

PROPRIETÀ DEL MIN-HEAP: IL VALORE DI UN NODO È MAGGIORE O UGUALE AL VALORE DEL PADRE (SE ESISTENTE)

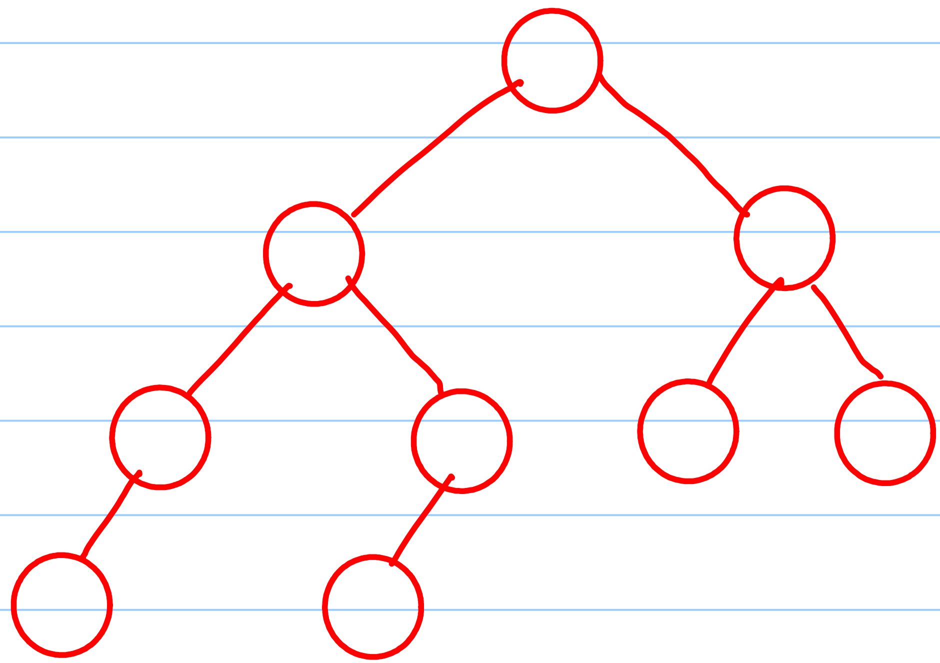
ALBERI BINARI QUASI COMPLETI

SONO ALBERI BINARI POSIZIONALI TALI CHE:

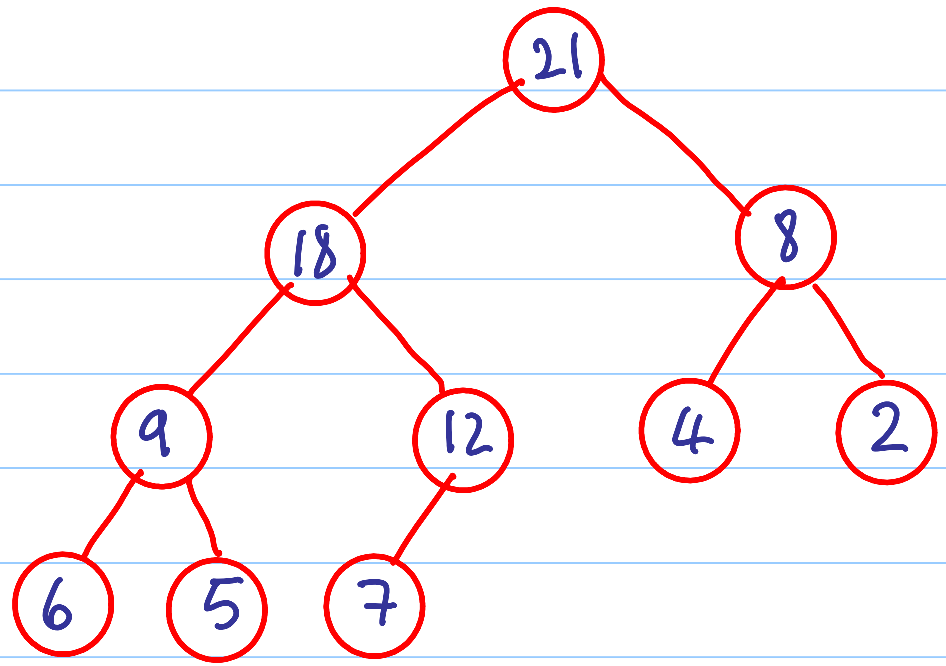
- TUTTI I LIVELLI, AD ECCEZIONE POSSIBILMENTE DELL'ULTIMO, SONO COMPLETI
- NELL'ULTIMO LIVELLO TUTTE LE FOGLIE SONO ADDOSSATE A SINISTRA



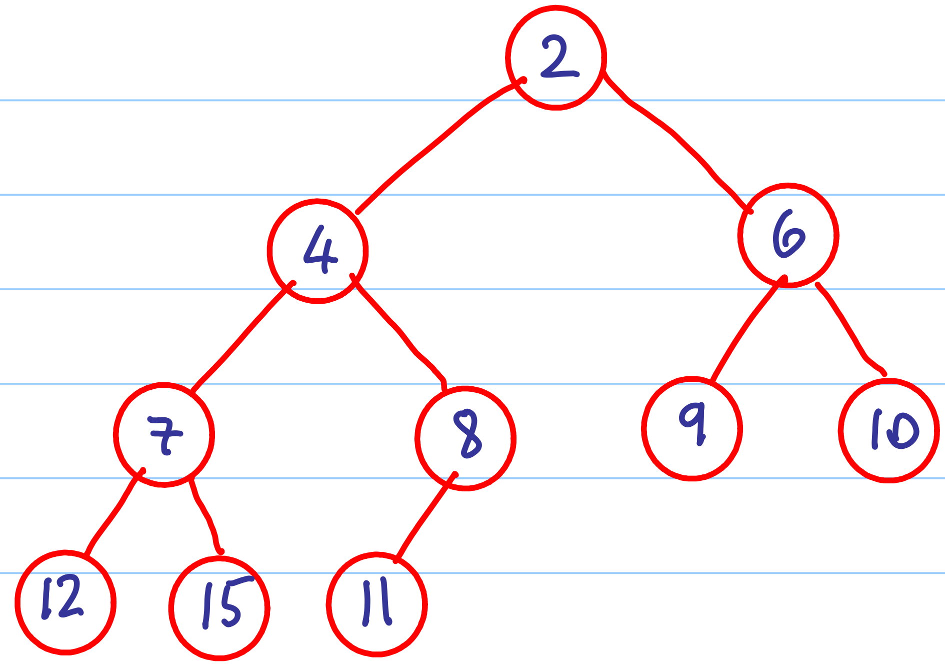
QUASI COMPLETO



NON È QUASI COMPLETO



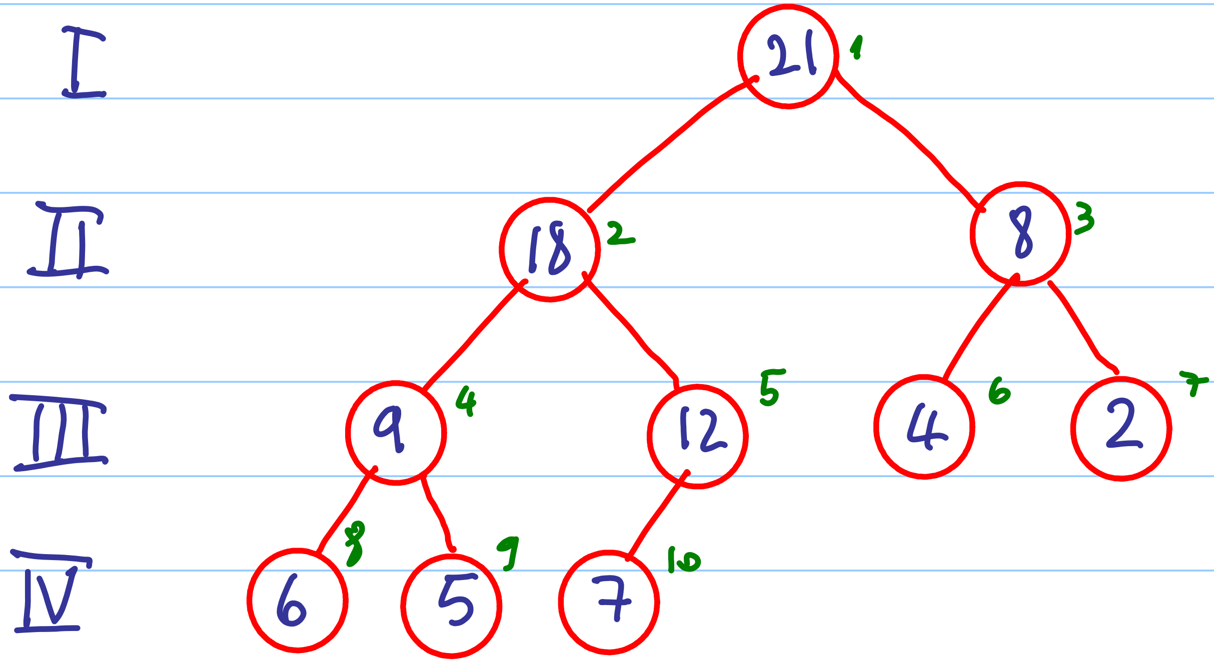
MAX-HEAP



MIN-HEAP

RAPPRESENTAZIONE MEDIANTE ARRAY

(MAX-HEAP)



A.heap-size

	1	2	3	4	5	6	7	8	9	10	11	12		n
A:	21	18	8	9	12	4	2	6	5	7			...	
	I	II	II	III	III	III	III	IV	IV	IV				

ARRAY A CON ATTRIBUTI

- $A.length$ (LUNGHEZZA)
- $A.heap\text{-}size$ (DIMENSIONE DELL' HEAP)

RADICE DELL' HEAP : $A[1]$

FIGLI DEL NODO i :

$$LEFT(i) = 2i$$

$$RIGHT(i) = 2i+1$$

PADRE DEL NODO i :

$$PARENT(i) = \lfloor i/2 \rfloor$$

PROPRIETÀ MAX-HEAP :

$$1 < i \leq A.heap\text{-}size \implies A[PARENT(i)] \geq A[i]$$

PROPRIETÀ

- IL PIÙ GRANDE ELEMENTO IN UN MAX-HEAP E' NELLA RADICE
- IL PIÙ PICCOLO ELEMENTO IN UN MIN-HEAP E' NELLA RADICE

ALTEZZA DI UN NODO: NUMERO DI ARCHI NEL CAMMINO SEMPLICE PIÙ LUNGO DAL NODO FINO AD UNA FOGLIA

ALTEZZA DI UN HEAP: ALTEZZA DELLA RADICE

PROPRIETA'

L'ALTEZZA DI UN HEAP CON n ELEMENTI E' $\lfloor \log n \rfloor$ (DUNQUE $\Theta(\log n)$)

DIM SIA h L'ALTEZZA DI UN HEAP CON n ELEMENTI.

LIVELLO	# NODI (n_i)
0	1
1	2
2	2^2
3	2^3
$\vdots$	$\vdots$
$h-1$	2^{h-1}
h	$1 \leq n_h \leq 2^h$

SI HA:

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

$$1 \leq n_h \leq 2^h$$

$$\rightarrow \quad 2^h \leq 2^h - 1 + n_h \leq 2^h - 1 + 2^h = 2^{h+1} - 1$$

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

$$\rightarrow \quad h \leq \log n < h+1$$

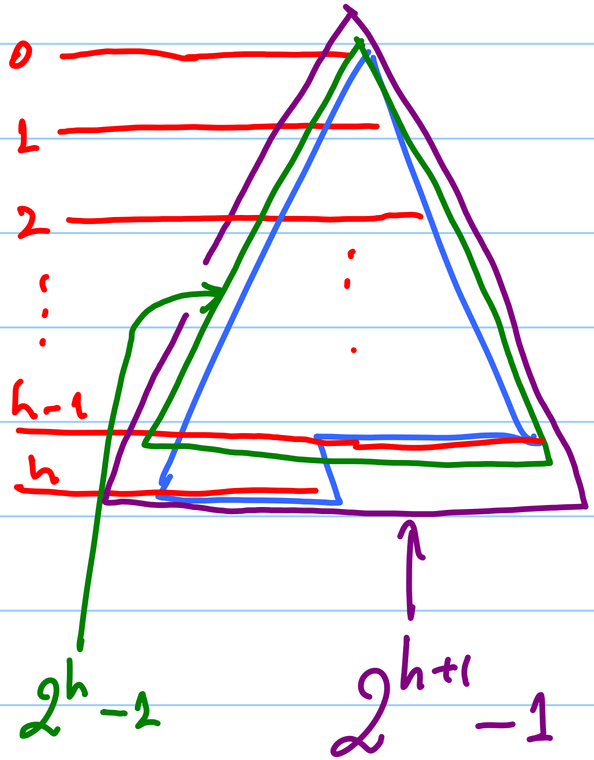
$$\rightarrow \quad h = \lfloor \log n \rfloor .$$

$\blacksquare$

PROPRIETA'

L'ALTEZZA DI UN HEAP CON n ELEMENTI E' $\lfloor \lg n \rfloor$
(DUNQUE $\Theta(\log n)$)

DIM SIA h L'ALTEZZA DI UN HEAP CON n ELEMENTI.



E' BEN NOTO CHE UN ALBERO BINARIO DI ALTEZZA i HA ESATTAMENTE $2^{i+1}-1$ NODI

DUNQUE SI HA:

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

DA CUI:

$$h \leq \lg n < h+1 \Rightarrow h = \lfloor \lg n \rfloor$$

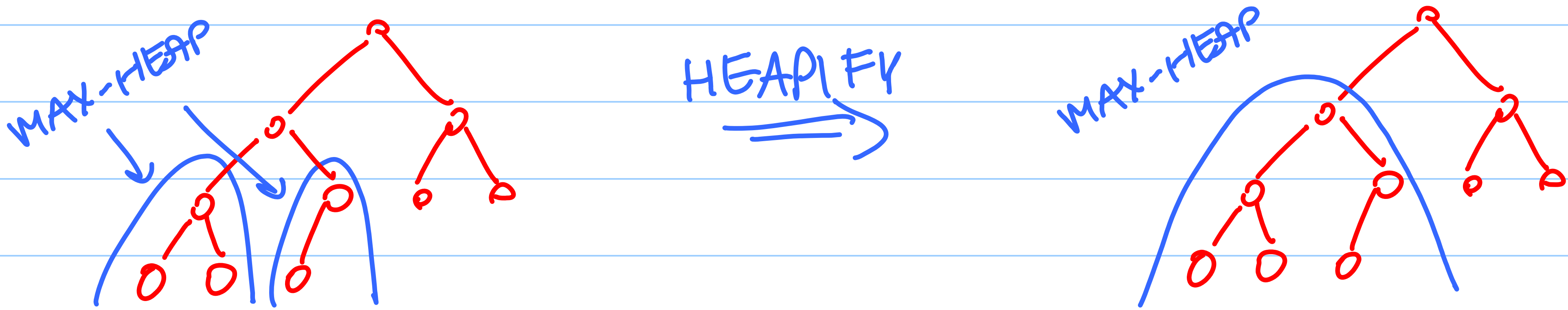
$\blacksquare$

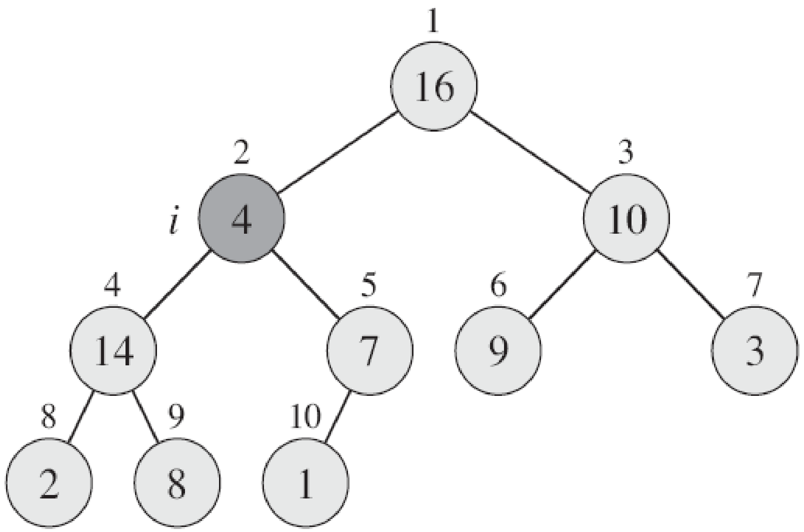
PROCEDURE

MAX-HEAPIFY

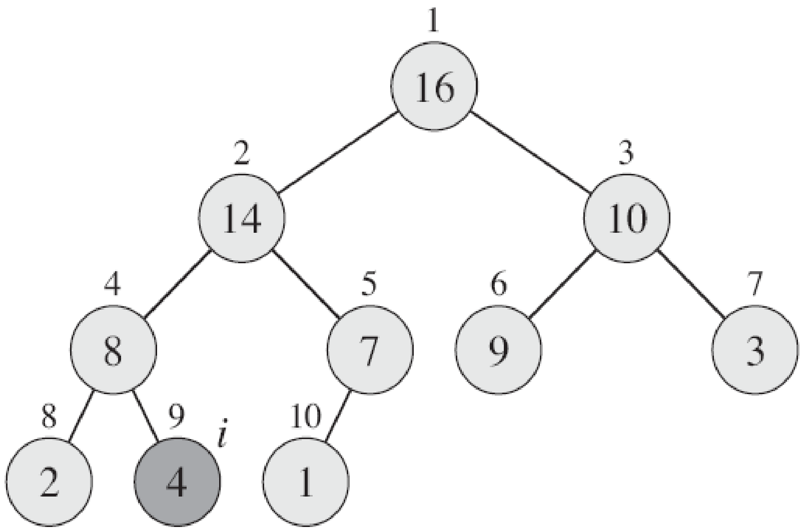
INPUT: UN ARRAY A E UN INDICE $1 \leq i \leq A.length$ TALI CHE GLI ALBERI CON RADICI LEFT(i) E RIGHT(i) SIANO MAX-HEAP

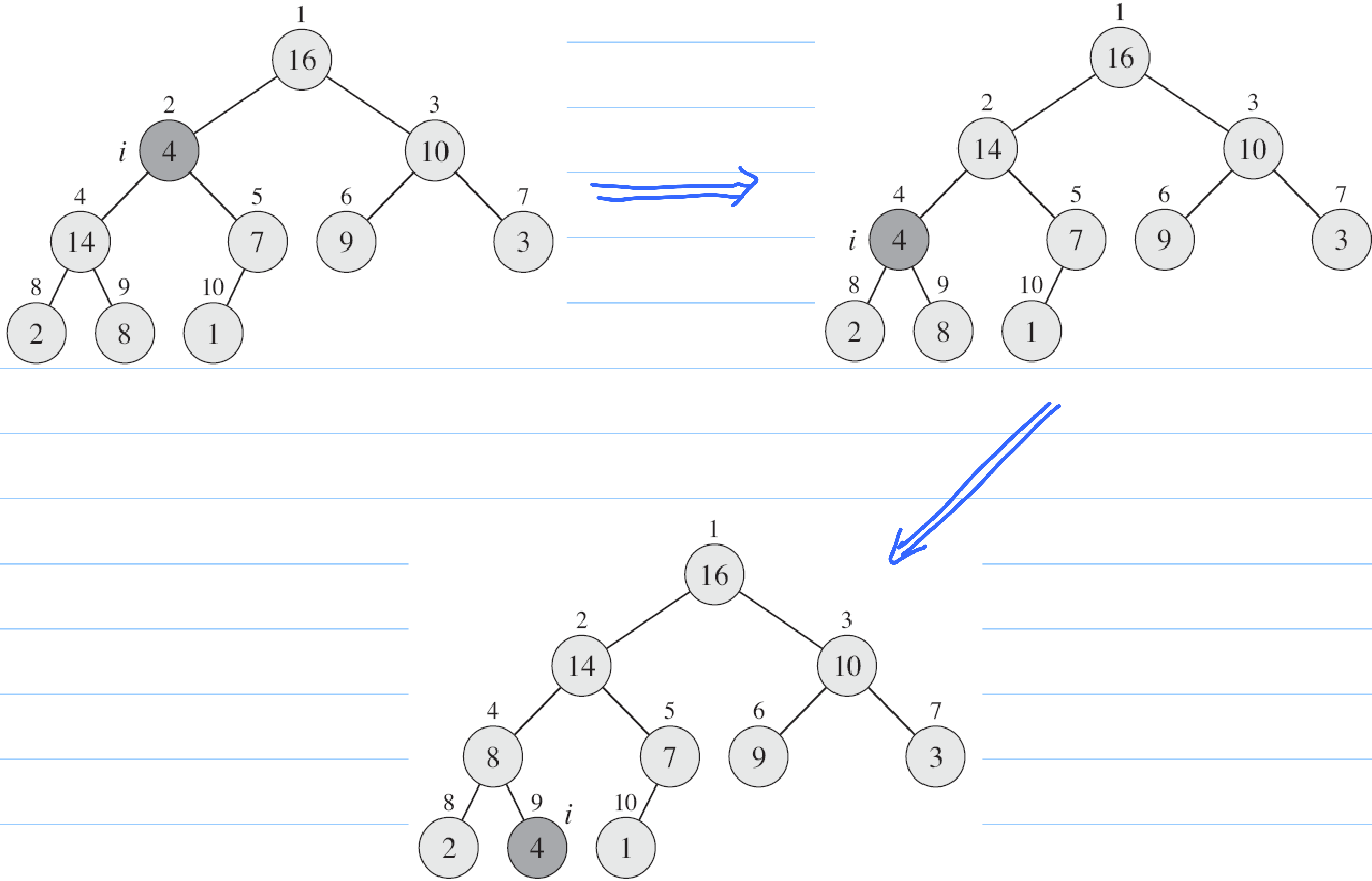
OUTPUT: UNA PERMUTAZIONE DELL'ARRAY A TALE CHE L'ALBERO CON RADICE i SIA UN MAX-HEAP





MAX-HEAPIFY (A, 2)





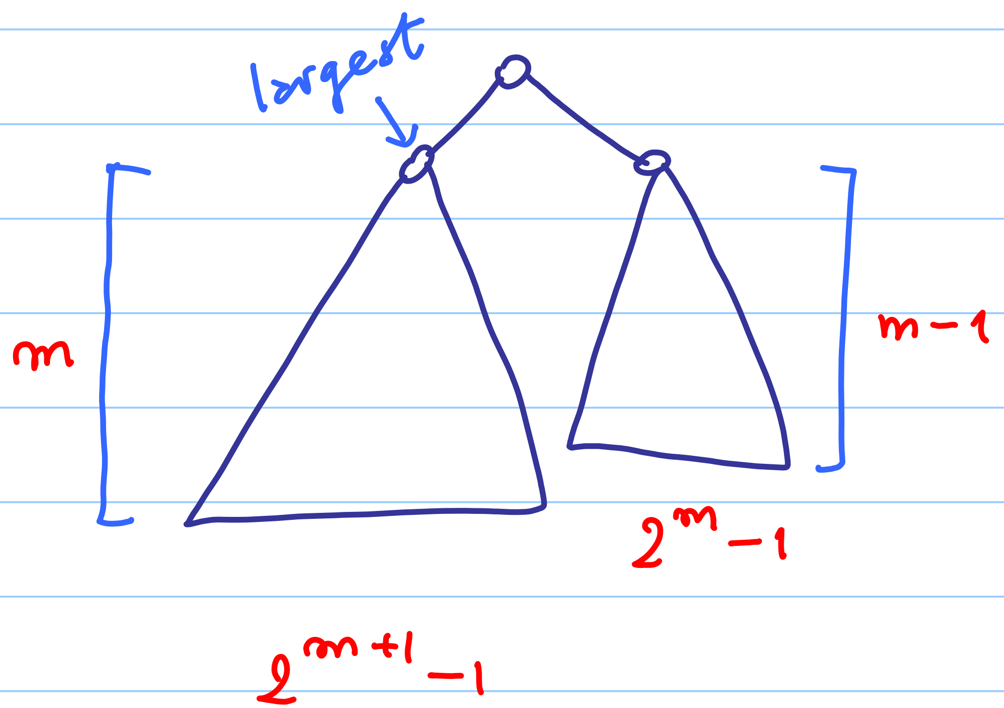
1
16
2
3
i
4
10
4
5
6
7
14
7
9
3
8
9
10
2
8
1
1
16
2
3
14
10
4
5
6
7
i
4
7
9
3
8
9
10
2
8
1
1
16
2
3
14
10
4
5
6
7
8
7
9
3
8
9
10
i
2
4
1

```
MAX-HEAPIFY(A, i)
l = LEFT(i)
r = RIGHT(i)
if l ≤ A.heap-size and A[l] > A[i]
    largest = l
else largest = i
if r ≤ A.heap-size and A[r] > A[largest]
    largest = r
if largest ≠ i
    exchange A[i] with A[largest]
    MAX-HEAPIFY(A, largest)
```

COMPLESSITÀ MAX-HEAPIFY

$$T(n) \le T(\text{size\_of}(A[\text{largest}])) + \Theta(1)$$

CASO PEGGIORE



$$n = 2^{m+1} - 1 + 2^m - 1 + 1$$
$$= 2^{m+1} + 2^m - 1$$
$$= 3 \cdot 2^m - 1$$

$$2^m = \frac{n+1}{3} \qquad 2^{m+1} = \frac{2}{3}(n+1)$$

$$\text{size\_of}(A[\text{largest}]) \le \frac{2}{3}(n+1) - 1$$

$$\text{size\_of}(A[\text{target}]) \leq \frac{2}{3}(n+1) - 1 = \frac{2n}{3} - \frac{1}{3} < \frac{2n}{3}$$

E DUNQUE

$$T(n) \leq T\left(\frac{2}{3}n\right) + \Theta(1)$$

$$a = 1, \quad b = \frac{3}{2} \qquad \log_b a = 0 \qquad n^{\log_b a} = \Theta(1)$$

TEOREMA MASTER $\Longrightarrow$

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

COSTRUZIONE DI UN HEAP

IDEA:

- LE FOGLIE GODONO DELLA PROPRIETÀ MAX-HEAP
- SE k È IL MASSIMO DEGLI INDICI DEI NODI INTERNI, LE CHIAMATE

  MAX-HEAPIFY (A, k)

  MAX-HEAPIFY $(A, k-1)$

  ...

  MAX-HEAPIFY $(A, 2)$

  MAX-HEAPIFY $(A, 1)$

  CONSENTONO DI PROPAGARE LA PROPRIETÀ MAX-HEAP ANCHE AI NODI $k, k-1, \dots, 2, 1$.

i FOGLIA $\iff n < 2i \leq 2n$

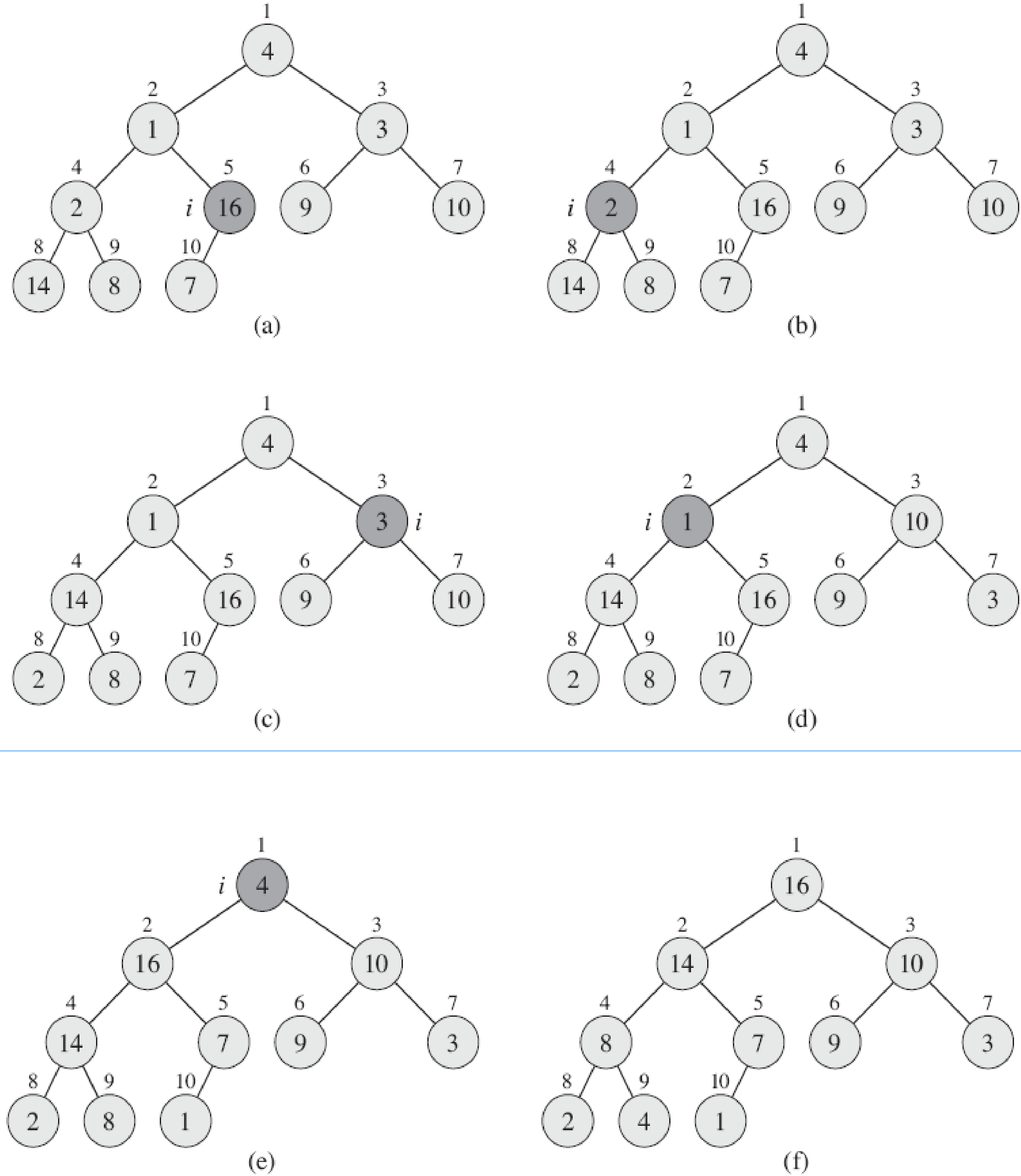
$\iff \frac{n}{2} < i \leq n \iff \lfloor \frac{n}{2} \rfloor + 1 \leq i \leq n$

QUINDI IL MASSIMO DEGLI INDICI DEI NODI INTERNI E' $\lfloor \frac{n}{2} \rfloor$.

```
BUILD-MAX-HEAP(A)
  A.heap-size = A.length
  for i = ⌊A.length/2⌋ downto 1
      MAX-HEAPIFY(A, i)
```

ESEMPIO

A									
4	1	3	2	16	9	10	14	8	7



COMPLESSITÀ DI BUILD-MAX-HEAP

LIMITE ASINTOTICO (NON STRETTO): $O(n \log n)$

PROPRIETÀ

IN UN HEAP CON n ELEMENTI CI SONO ESATTAMENTE $\left\lfloor \frac{n}{2} \right\rfloor$ NODI DI ALTEZZA ≥ 1.

PROCEDENDO PER INDUZIONE, LA PRECEDENTE PROPRIETÀ PUÒ ESSERE GENERALIZZATA.

PROPRIETÀ

IN UN HEAP CON n ELEMENTI CI SONO ESATTAMENTE $\left\lfloor \frac{n}{2^h} \right\rfloor$ NODI DI ALTEZZA $\geq h$.

PERTANTO:

PROPRIETÀ

IN UN HEAP CON n ELEMENTI CI SONO ESATTAMENTE $\left\lfloor \frac{n}{2^h} \right\rfloor - \left\lfloor \frac{n}{2^{h+1}} \right\rfloor$ NODI DI ALTEZZA $= h$.

$$T(n) \leq \sum_{h=1}^{\lfloor \lg n \rfloor} \left(\left\lfloor \frac{n}{2^h} \right\rfloor - \left\lfloor \frac{n}{2^{h+1}} \right\rfloor \right) \cdot O(h) = O\left(\sum_{h=1}^{\lfloor \lg n \rfloor} h \left(\left\lfloor \frac{n}{2^h} \right\rfloor - \left\lfloor \frac{n}{2^{h+1}} \right\rfloor \right) \right)$$

QUINDI

$$T(n) = O\left(\sum_{h=1}^{\lfloor \lg n \rfloor} h \left(\left\lfloor \frac{n}{2^h} \right\rfloor - \left\lfloor \frac{n}{2^{h+1}} \right\rfloor \right) \right)$$

$$T(n) = O\left(\sum_{h=1}^{\lfloor \lg n \rfloor} h \left(\left\lfloor \frac{n}{2^h} \right\rfloor - \left\lfloor \frac{n}{2^{h+1}} \right\rfloor \right)\right)$$

$$\sum_{h=1}^{\lfloor \lg n \rfloor} h \left(\left\lfloor \frac{n}{2^h} \right\rfloor - \left\lfloor \frac{n}{2^{h+1}} \right\rfloor \right) = \sum_{h=1}^{\lfloor \lg n \rfloor} h \cdot \left\lfloor \frac{n}{2^h} \right\rfloor - \sum_{h=1}^{\lfloor \lg n \rfloor} h \cdot \left\lfloor \frac{n}{2^{h+1}} \right\rfloor$$

$$= \left\lfloor \frac{n}{2} \right\rfloor + \sum_{h=2}^{\lfloor \lg n \rfloor} h \cdot \left\lfloor \frac{n}{2^h} \right\rfloor - \sum_{h=1}^{\lfloor \lg n \rfloor - 1} h \cdot \left\lfloor \frac{n}{2^{h+1}} \right\rfloor$$

$$= \left\lfloor \frac{n}{2} \right\rfloor + \sum_{h=1}^{\lfloor \lg n \rfloor - 1} (h+1) \cdot \left\lfloor \frac{n}{2^{h+1}} \right\rfloor - \sum_{h=1}^{\lfloor \lg n \rfloor - 1} h \cdot \left\lfloor \frac{n}{2^{h+1}} \right\rfloor$$

$$= \left\lfloor \frac{n}{2} \right\rfloor + \sum_{h=1}^{\lfloor \lg n \rfloor - 1} \left\lfloor \frac{n}{2^{h+1}} \right\rfloor$$

$$= \left\lfloor \frac{n}{2} \right\rfloor + \sum_{h=2}^{\lfloor \lg n \rfloor} \left\lfloor \frac{n}{2^h} \right\rfloor = \sum_{h=1}^{\lfloor \lg n \rfloor} \left\lfloor \frac{n}{2^h} \right\rfloor < \sum_{h=1}^{\infty} \frac{n}{2^h} = n$$

PERTANTO: $T(n) = O(n)$.

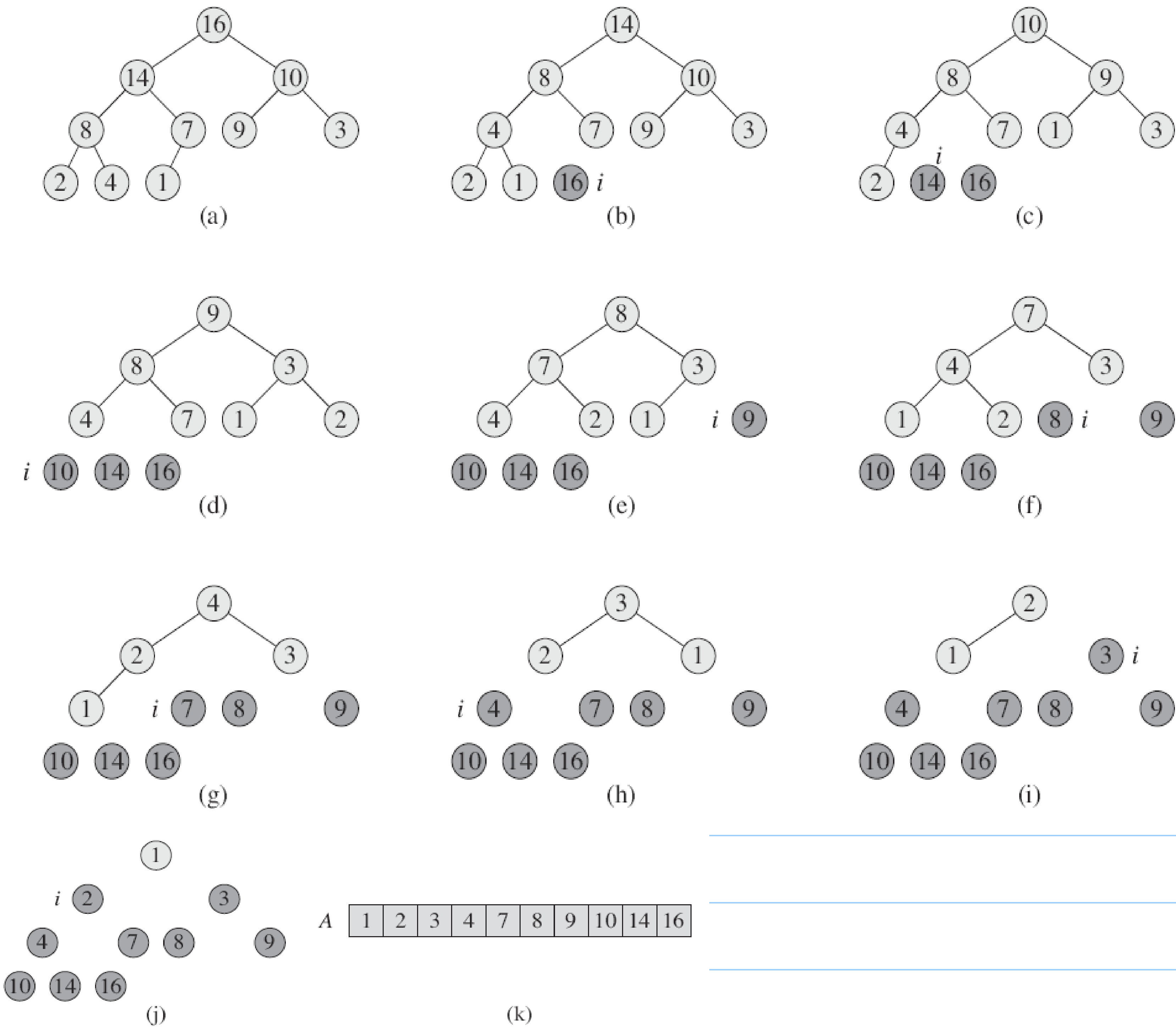
$\blacksquare$

L'ALGORITMO HEAPSORT

```
HEAPSORT(A)
BUILD-MAX-HEAP(A)
for i = A.length downto 2
    exchange A[1] with A[i]
    A.heap-size = A.heap-size − 1
    MAX-HEAPIFY(A, 1)
```

COMPLESSITÀ: $O(n \log n)$

ESEMPIO



ESERCIZI

6.1-1

What are the minimum and maximum numbers of elements in a heap of height h?

6.1-3

Show that in any subtree of a max-heap, the root of the subtree contains the largest value occurring anywhere in that subtree.

6.1-4

Where in a max-heap might the smallest element reside, assuming that all elements are distinct?

6.1-5

Is an array that is in sorted order a min-heap?

6.1-6

Is the array with values $\langle 23, 17, 14, 6, 13, 10, 1, 5, 7, 12 \rangle$ a max-heap?

6.2-1

Illustrate the operation of MAX-HEAPIFY$(A, 3)$ on the array $A = \langle 27, 17, 3, 16, 13, 10, 1, 5, 7, 12, 4, 8, 9, 0 \rangle$.

6.2-2

Starting with the procedure MAX-HEAPIFY, write pseudocode for the procedure MIN-HEAPIFY(A, i), which performs the corresponding manipulation on a min-heap. How does the running time of MIN-HEAPIFY compare to that of MAX-HEAPIFY?

6.2-3

What is the effect of calling MAX-HEAPIFY(A, i) when the element $A[i]$ is larger than its children?

6.2-4

What is the effect of calling MAX-HEAPIFY(A, i) for $i > A.\mathit{heap\text{-}size}/2$?

6.2-6

Show that the worst-case running time of MAX-HEAPIFY on a heap of size n is $\Omega(\lg n)$.

6.3-1

Using Figure 6.3 as a model, illustrate the operation of BUILD-MAX-HEAP on the array $A = \langle 5, 3, 17, 10, 84, 19, 6, 22, 9 \rangle$.

6.3-2

Why do we want the loop index i in line 2 of BUILD-MAX-HEAP to decrease from $\lfloor A.length/2 \rfloor$ to 1 rather than increase from 1 to $\lfloor A.length/2 \rfloor$?

6.4-1

Using Figure 6.4 as a model, illustrate the operation of HEAPSORT on the array $A = \langle 5, 13, 2, 25, 7, 17, 20, 8, 4 \rangle$.

CODE DI PRIORITÀ

CODE DI MAX-PRIORITÀ E CODE DI MIN-PRIORITÀ

UNA CODA DI MAX-PRIORITÀ SUPPORTA LE SEGUENTI OPERAZIONI

- INSERT(S, x) : INSERISCE x IN S
- MAXIMUM(S) : RESTITUISCE L'ELEMENTO DI S CON LA CHIAVE MAX
- EXTRACT-MAX(S) : ESTRAE DA S L'ELEMENTO CON LA CHIAVE PIÙ GRANDE E LO RESTITUISCE
- INCREASE-KEY(S, x, k) : AUMENTA IL VALORE DELLA CHIAVE DI x AL NUOVO VALORE k (CON k NON INFERIORE AL VALORE CORRENTE DELLA CHIAVE DI x)

APPLICAZIONI

CODE DI MAX-PRIORITÀ

- GESTIONE PRIORITÀ SU RISORSE CONDIVISE

CODE DI MIN-PRIORITÀ

- SIMULATORE CONTROLLATO DA EVENTI

```
HEAP-MAXIMUM(A)
    return A[1]
```

COMPLESSITÀ: $O(1)$

```
HEAP-EXTRACT-MAX(A)
    if A.heap-size < 1
        error "heap underflow"
    max = A[1]
    A[1] = A[A.heap-size]
    A.heap-size = A.heap-size - 1
    MAX-HEAPIFY(A, 1)
    return max
```

COMPLESSITÀ: $O(\lg n)$

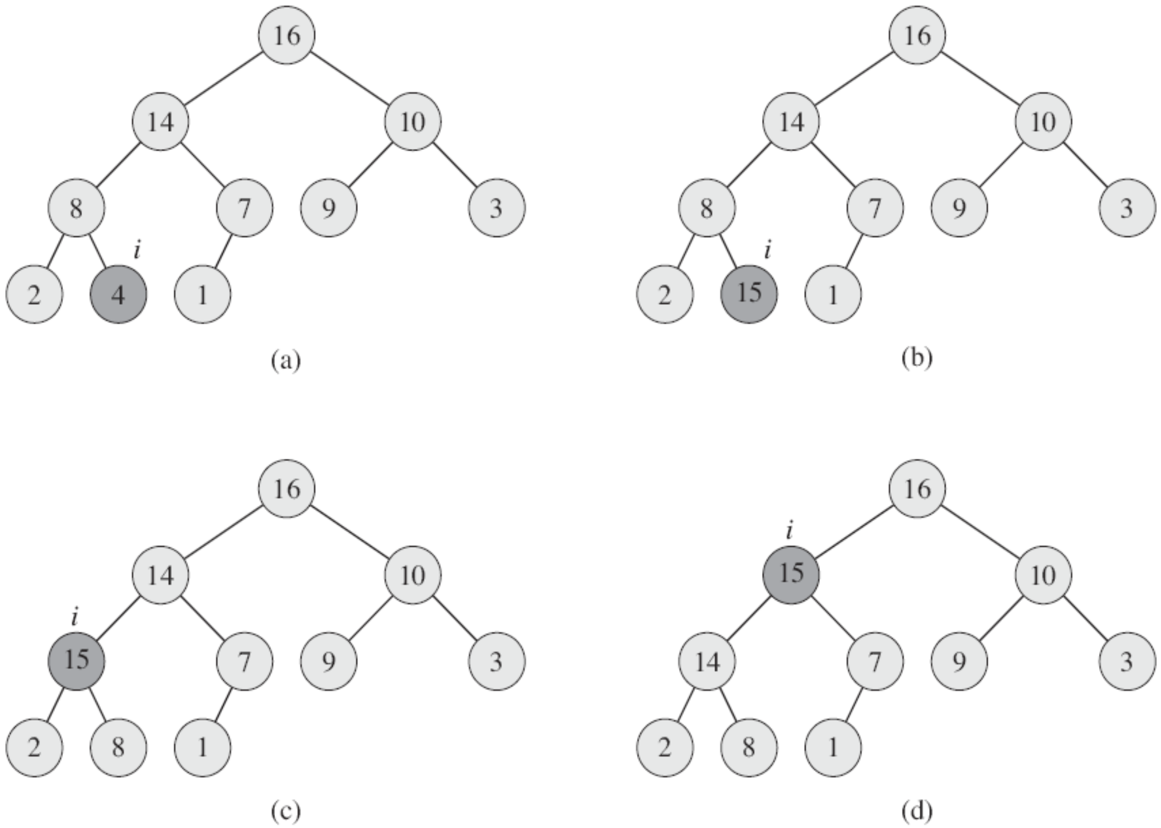
HEAP-INCREASE-KEY(A, i, key)

```
if key < A[i]
    error "new key is smaller than current key"
A[i] = key
while i > 1 and A[PARENT(i)] < A[i]
    exchange A[i] with A[PARENT(i)]
    i = PARENT(i)
```

COMPLESSITÀ: $O(\lg n)$

ESEMPIO

HEAP-INCREASE-KEY$(A, 9, 15)$



MAX-HEAP-INSERT(A, key)

1 $A.heap\text{-}size = A.heap\text{-}size + 1$
2 $A[A.heap\text{-}size] = -\infty$
3 HEAP-INCREASE-KEY$(A, A.heap\text{-}size, key)$

COMPLESSITÀ: $O(\lg n)$

ESERCIZI

6.5-1

Illustrate the operation of HEAP-EXTRACT-MAX on the heap $A = \langle 15, 13, 9, 5, 12, 8, 7, 4, 0, 6, 2, 1 \rangle$.

6.5-2

Illustrate the operation of MAX-HEAP-INSERT$(A, 10)$ on the heap $A = \langle 15, 13, 9, 5, 12, 8, 7, 4, 0, 6, 2, 1 \rangle$.

6.5-4

Why do we bother setting the key of the inserted node to $-\infty$ in line 2 of MAX-HEAP-INSERT when the next thing we do is increase its key to the desired value?

6.5-6

Each exchange operation on line 5 of HEAP-INCREASE-KEY typically requires three assignments. Show how to use the idea of the inner loop of INSERTION-SORT to reduce the three assignments down to just one assignment.

6.5-7

Show how to implement a first-in, first-out queue with a priority queue. Show how to implement a stack with a priority queue.

6.5-8

The operation HEAP-DELETE(A, i) deletes the item in node i from heap A. Give an implementation of HEAP-DELETE that runs in $O(\lg n)$ time for an n-element max-heap.

6.5-9

Give an $O(n \lg k)$-time algorithm to merge k sorted lists into one sorted list, where n is the total number of elements in all the input lists. (*Hint:* Use a min-heap for k-way merging.)