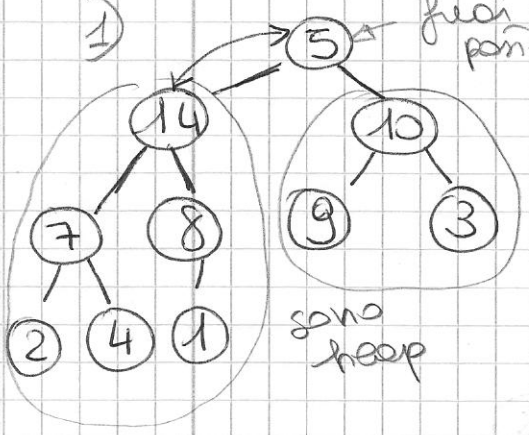
ESEMPI DI ESECUZIONE

① HEAPIFY

0	1	2	3	4	5	6	7	8	9
5	14	10	7	8	9	3	2	4	1

1)



Heapify(A, 0)

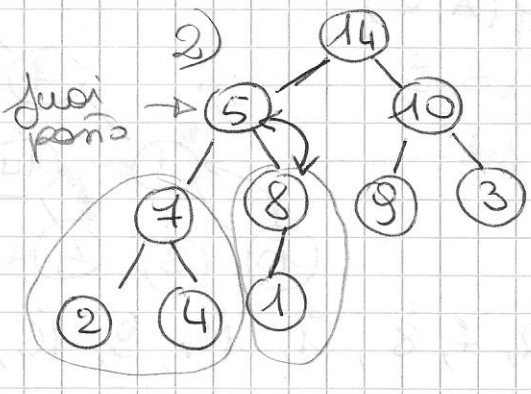
i = 0

l = 1 r = 2

indice_max = 1 scambio(A, 0, 1)

0	1	2	3	4	5	6	7	8	9
14	5	10	7	8	9	3	2	4	1

2)



Heapify(A, 1)

i = 1 l = 3 r = 4

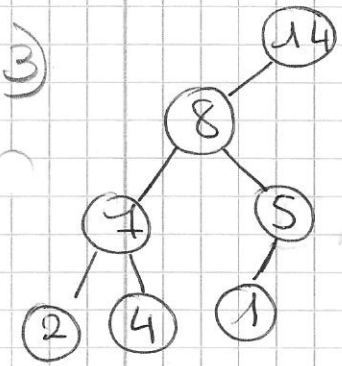
indice_max = 3

indice_max = 4

scambio(A, 1, 4)

0	1	2	3	4	5	6	7	8	9
14	8	10	7	5	9	3	2	4	1

3)



Heapify(A, 4)

i = 4

l = 9

indice_max = 4

r non esiste!!

indice_max = i quindi non effettuo scambi e non richiamo Heapify

Che succede se invece heapify su una foglia?
Non fa nulla perché non esistono i figli e quindi indice_max = i sempre

$i \geq n/2$

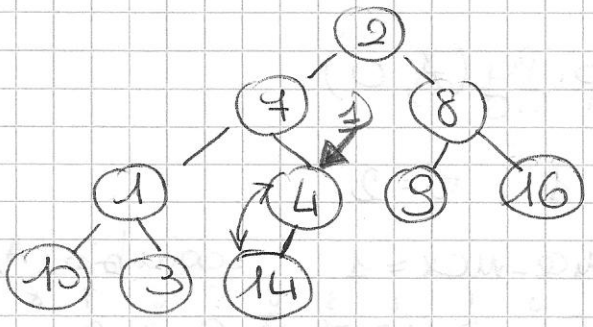
posso aggiungere un if in cima alla procedura if ($i < n/2$)

② BUILD HEAP

$A = [2, 7, 8, 1, 4, 9, 16, 10, 3, 14]$

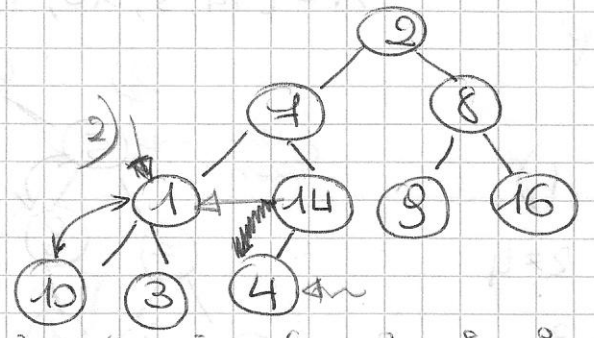
indici: 0 1 2 3 4 5 6 7 8 9

$(n/2-1)$ = 4 — $n/2$ = 5 — foglie: 5 6 7 8 9



1) i = 4 Heapify(A, 4)

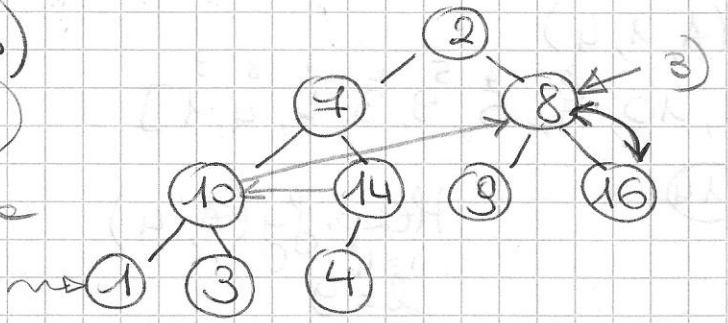
Heapify(A, 9) → foglia



$A = [2, 7, 8, 1, 14, 9, 16, 10, 3, 4]$

2) i = 3 Heapify(A, 3)

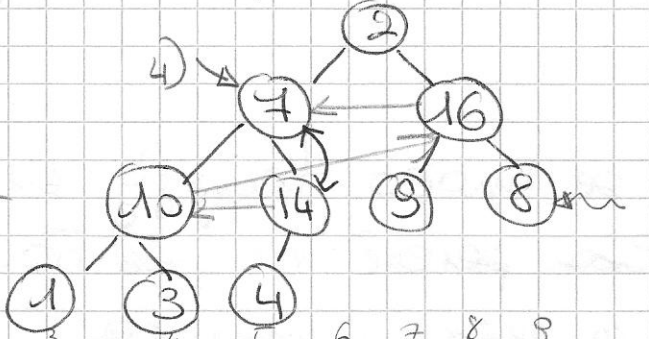
Heapify(A, 7) → foglia



$A = [2, 7, 8, 10, 14, 9, 16, 1, 3, 4]$

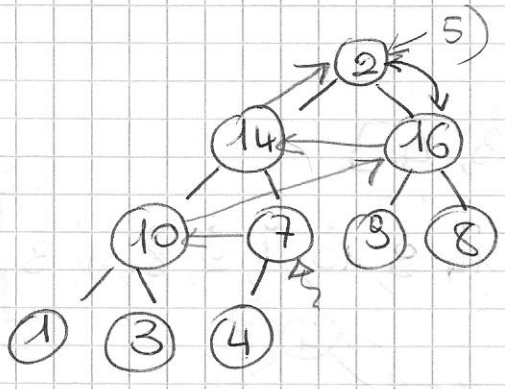
3) i = 2 Heapify(A, 2)

Heapify(A, 6) → foglia



$A = [2, 7, 16, 10, 14, 9, 8, 1, 3, 4]$

4) i = 1 Heapify(A, 1)

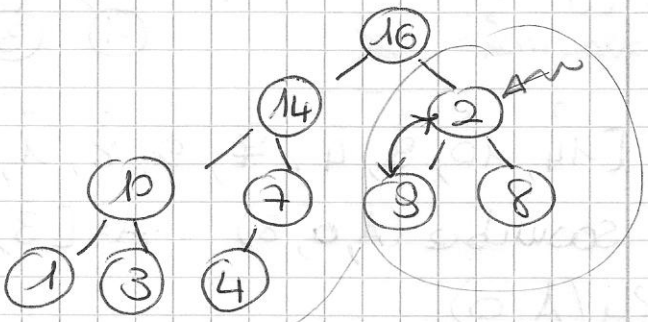


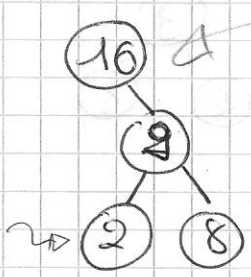
$A = [2, 14, 16, 10, 7, 9, 8, 1, 3, 4]$
$\quad\;\; 0 \;\; 1 \;\; 2 \;\; 3 \;\; 4 \;\; 5 \;\; 6 \;\; 7 \;\; 8 \;\; 9$

Heapify(A, 4) ⇒ indice_max = i mi fermo

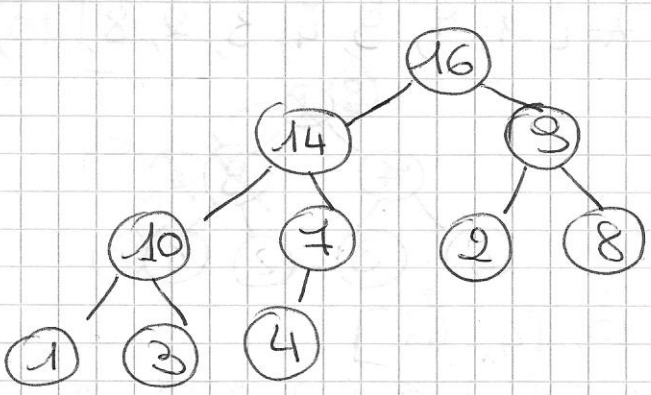
5) i = 0 Heapify(A, 0)

Heapify(A, 2)





Heapify(A, 5) ⇒ foglia ×



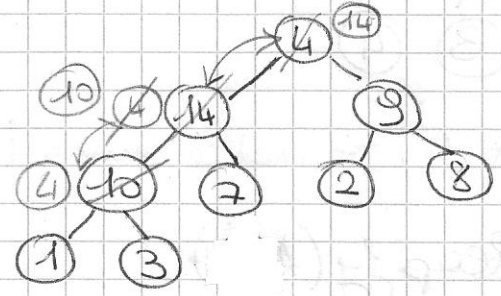
$A = [16, 14, 9, 10, 7, 2, 8, 1, 3, 4]$

③ HEAPSORT

$A = [16, 14, 9, 10, 7, 2, 8, 1, 3, 4]$ (índices 0 a 9)

1) $i = 9$ scambia$(A, 0, 9)$ $A = [4, 14, 9, 10, 7, 2, 8, 1, 3 \mid 16]$ ← heap size

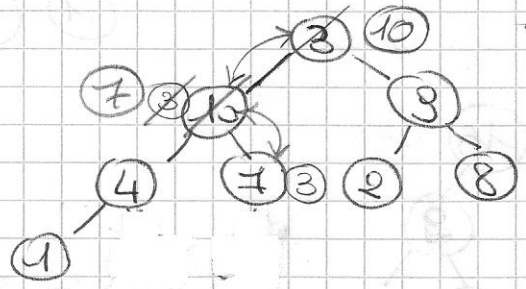
Heapify$(A, 0)$
Heapify$(A, 1)$
Heapify$(A, 3)$
indice_max = i ✗



$A = [14, 10, 9, 4, 7, 2, 8, 1, 3, 16]$

2) $i = 8$ scambia$(A, 0, 8)$ $A = [3, 10, 9, 4, 7, 2, 8, 1 \mid 14, 16]$

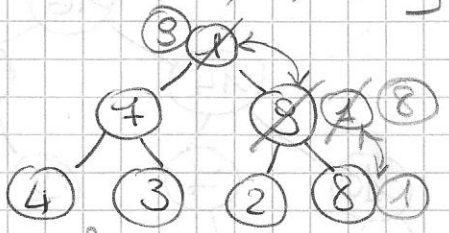
Heapify$(A, 0)$
Heapify$(A, 1)$
Heapify$(A, 4)$ foglia ✗



$A = [10, 7, 9, 4, 3, 2, 8, 1 \mid 14, 16]$

3) $i = 7$ scambia$(A, 0, 7)$ $A = [1, 7, 9, 4, 3, 2, 8 \mid 10, 14, 16]$

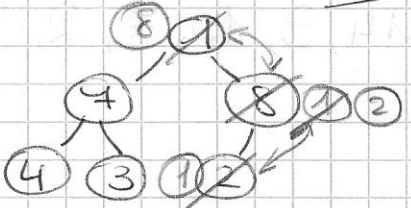
Heapify$(A, 0)$
Heapify$(A, 2)$
Heapify$(A, 6)$ foglia ✗



$A = [9, 7, 8, 4, 3, 2, 1, 10, 14, 16]$

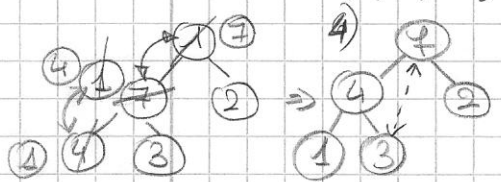
4) $i = 6$ scambia$(A, 0, 6)$ $A = [1, 7, 8, 4, 3, 2 \mid 9, 10, 14, 16]$

Heapify$(A, 0)$
Heapify$(A, 2)$
Heapify$(A, 5)$ foglia ✗

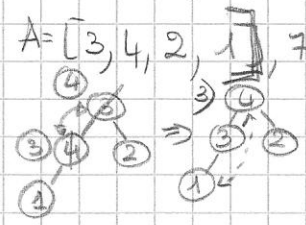


$A = [8, 7, 2, 4, 3, 1, 9, 10, 14, 16]$

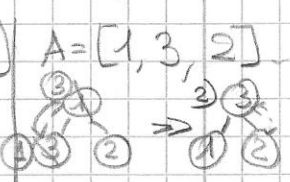
5) $i = 5$ scambia$(A, 0, 5)$ $A = [1, 7, 2, 4, 3, 8, 9, 10, 14, 16]$



$A = [3, 4, 2, 1 \mid 7, 8, 9, 10, 14, 16]$



$A = [1, 3, 2 \mid \ldots]$



$A = [2, 1 \mid \ldots]$

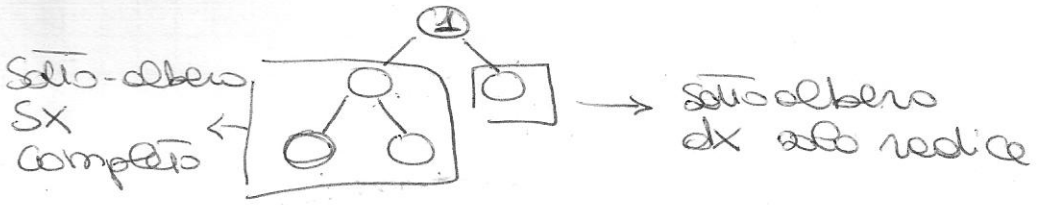
17 gennaio 2008 Es. 2 (8 punti)

Si dica quanti sono gli heap differenti che contengono le chiavi 1, 2, 3, 4, 5.

Supponiamo di lavorare con min-heap (heap binario basato sul minimo).

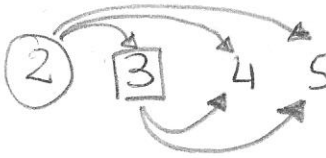
L'elemento minimo, cioè 1, deve essere la radice dell'albero binario.

Con 5 chiavi la rappresentazione ad albero binario di un heap binario ha la seguente forma:



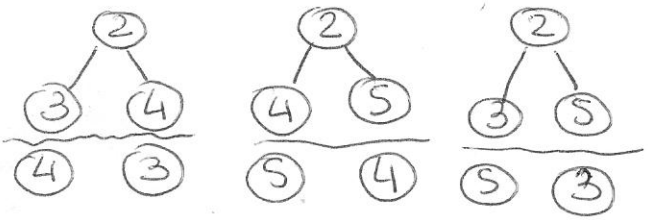
Bisogna trovare tutti i modi possibili (e ammissibili, cioè che rispettano la proprietà min-heap) per suddividere le rimanenti 4 chiavi nei sotto-alberi SX e dx.

Cosa altro sappiamo? Le 3 chiavi nel sottoalbero SX devono rispettare la proprietà min-heap, cioè la chiave della radice del sottoalbero SX deve essere < dei suoi figli.



come posso ottenere questo? Scegliendo come radice del sottoalbero SX 2 o 3.

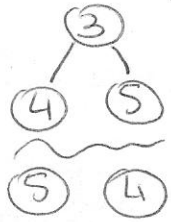
Se si sceglie (2) come radice, i possibili sotto-alberi SX sono:



in n° pari alle disposizioni semplici di 3 elementi di classe 2 (a 2 a 2)

$$\frac{n!}{(n-k)!} = \frac{3!}{(3-2)!} = 3! = 6$$

Se si sceglie (3):



sono solo 2

Il sottoalbero dx avrà come radice l'elemento non scelto per il sottoalbero SX, perciò il n° di possibili ~~mesti~~ min-heap diversi con chiavi 1 2 3 4 5 è 8

7 luglio 2004 EX. 3 (5 punti)

Si produca un esempio che dimostri che l'alg HEAPSORT non è stabile.

Un algoritmo STABILE preserva l'ordine di chiavi duplicate.

La stabilità di un alg di ordinamento è importante quando si vogliono ordinare gli stessi dati più di una volta in base a chiavi diverse.
Talvolta per es. le entry di un DB hanno chiavi multiple (pensiamo ad un DB di studenti con chiave primaria la matricola e una o più chiavi secondarie come la città di residenza, l'età, l'anno di immatricolazione, ...)
Noi potremmo voler ordinare tali dati in accordo a più di una chiave.
Il problema è che se noi ordiniamo gli stessi dati rispetto ad una chiave e poi rispetto ad una seconda chiave, il secondo ordinamento potrebbe alterare il primo.
Questo non succede se il secondo ordinamento è stabile.

Per es.

Cognome	Età
ALESSI	25
BINI	22
COLLI	25
DANTE	25
ENNA	22
FESTA	22

↑

Vogliamo ordinare gli studenti per età.

← [length[A]/2]

ordine crescente

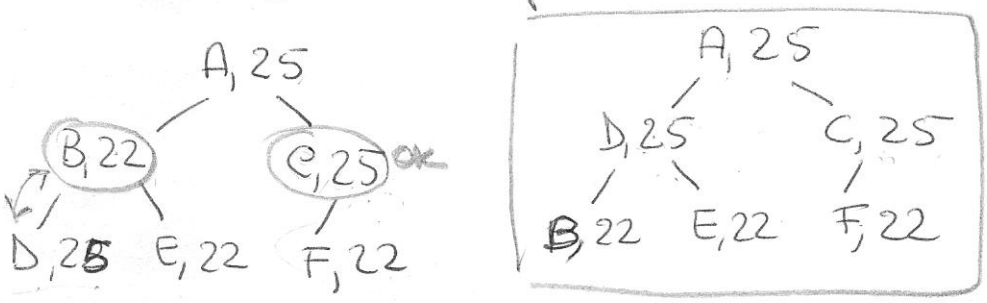
Cognome	Età
BINI	22
ENNA	22
FESTA	22
ALESSI	25
COLLI	25
DANTE	25

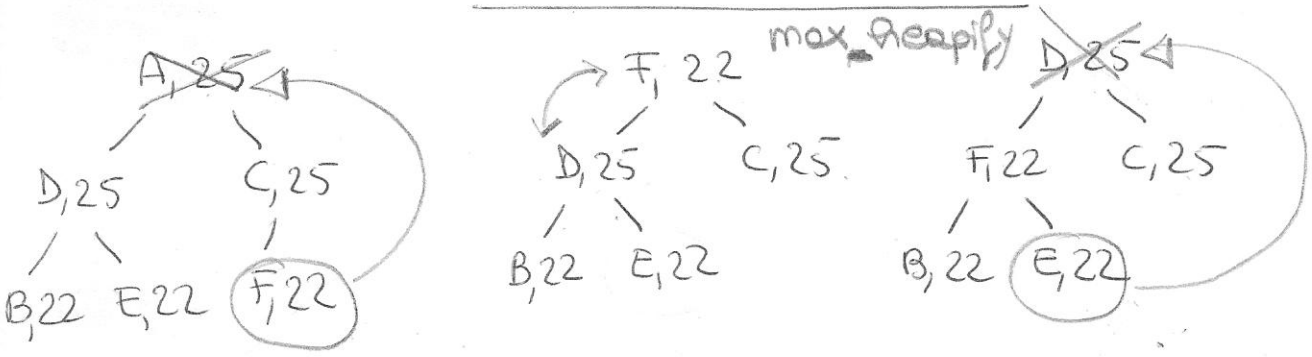
Algo di ordinamento stabile

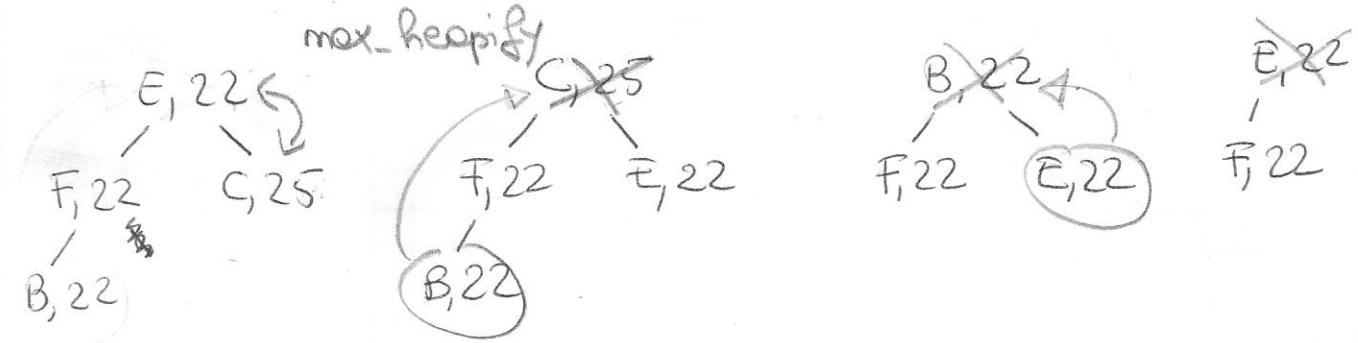
⇓

questo non accade usando HEAPSORT

Costruzione max-heap:







COGNOME	ETÀ
FESTA	22
ENNA	22
BINI	22
COLLI	25
DANTE	25
ALESSI	25

Iterative e ricorsiva

ESERCIZIO: Scrivere una funzione/procedura che dato in input un array di n interi verifichino efficientemente che l'array sia un heap.

Condizione necessaria e suff. affinché un array sia un min-heap:

$$\forall\ 0 \le i \le \left\lfloor \frac{A.length}{2} \right\rfloor \quad A[i] < A[2i+1] \text{ e } A[i] < A[2i+2]$$

se gli el $2i+1$ e $2i+2$ esistono

Iterative:

```
Verifica-heap (A)

    flag = TRUE
    i = 0
    while (flag AND i < ⌊A.length/2⌋)
        if 2i + 2 > n  // ∄ figlio dx
        // si verifica se il figlio SX ha un valore > del padre
            flag = A[i] < A[2i + 1]
        else
        // si verifica se i figli SX e dx hanno valore > del padre
            flag = (A[i] < A[2i+1]) AND
                   (A[i] < A[2i + 2])
        i++
    return flag
```

RICORSIVA:

verifica_heap(A, i)

if $i >= \lceil \frac{A.length}{2} \rceil$ // se i è l'indice di una foglia

return TRUE

else

if $2i+2 > n$ // il figlio dx non esiste

return $(A[i] < A[2i+1])$ and verifica_heap(A, 2i+1)

else

return $(A[i] < A[2i+1])$ and
$(A[i] < A[2i+2])$ and
verifica_heap(A, 2i+1)
verifica_heap(A, 2i+2)

Prima chiamata: verifica_heap(A, 0)

Condizione:

Caso base: foglia = è un heap di size 1.

Passo induttivo: • $\forall\, 0 \le i \le \lfloor \frac{A.length}{2} \rfloor$ $\quad A[i] < A[2i+1]$ $\quad A[i] < A[2i+2]$

• la condizione è

$A[2i+1 .. n/2-1]$ e/o $A[2i+2 .. n/2-1]$

sono un heap.