

# QuickHeapsort

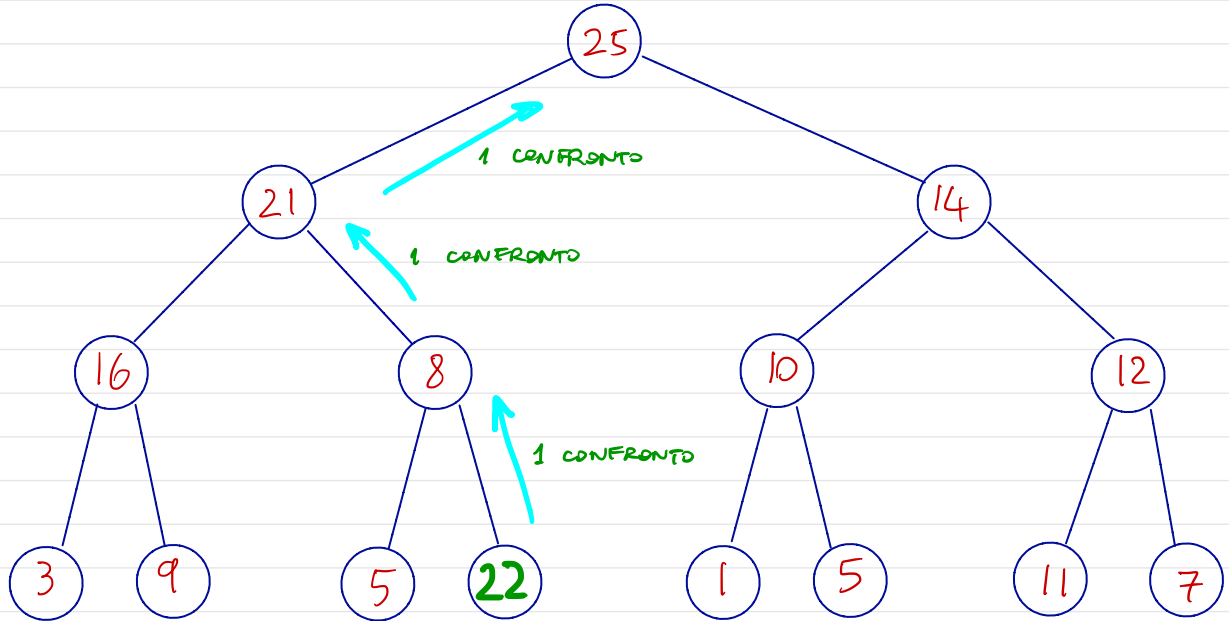
Cantone-Cincotti (2000, 2002)

Dickert-Weiß (2015)

## OSSERVAZIONE:

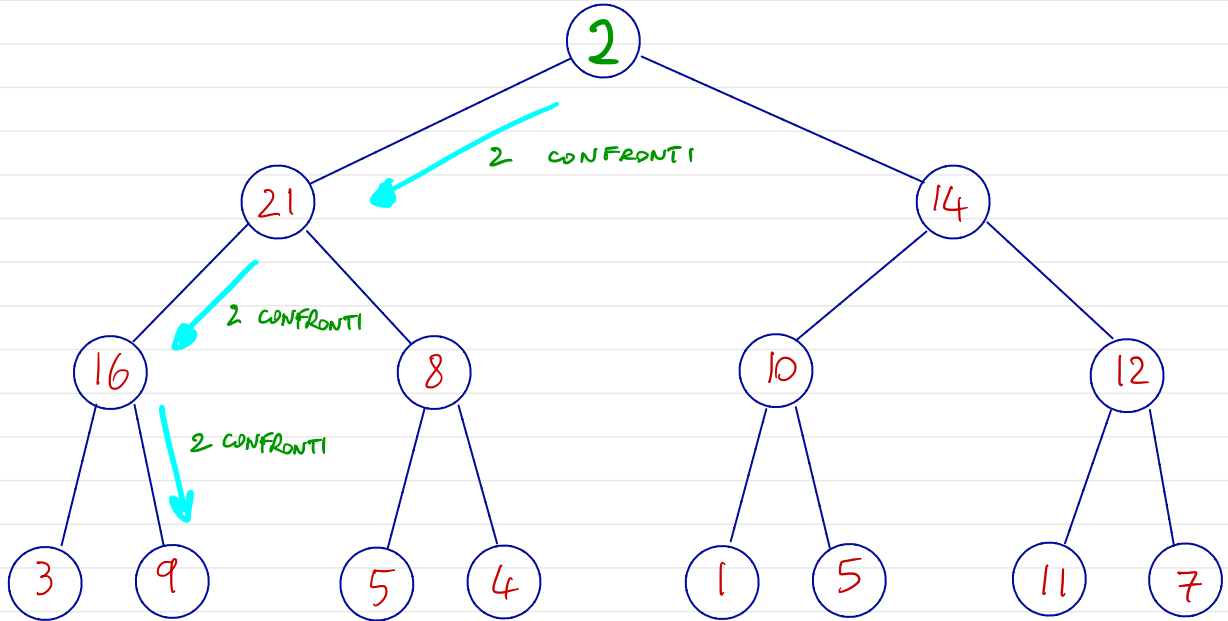
- IN UN HEAP (BINARIO) GLI ELEMENTI SI POSSONO MUOVERE
  - DAL BASSO VERSO L'ALTO (SIFT-UP)  
(ES. INSERT, INCREASE\_KEY (IN UN MAX-HEAP))
  - DALL'ALTO VERSO IL BASSO (SIFT-DOWN)  
(ES. HEAPIFY, HEAPSORT)

# SIFT-UP (RISALITA)



- DURANTE UN'OPERAZIONE DI SIFT-UP, PER CIASCUN LIVELLO VIENE EFFETTUATO UN SOLO CONFRONTO

# SIFT-DOWN (DISCESA)



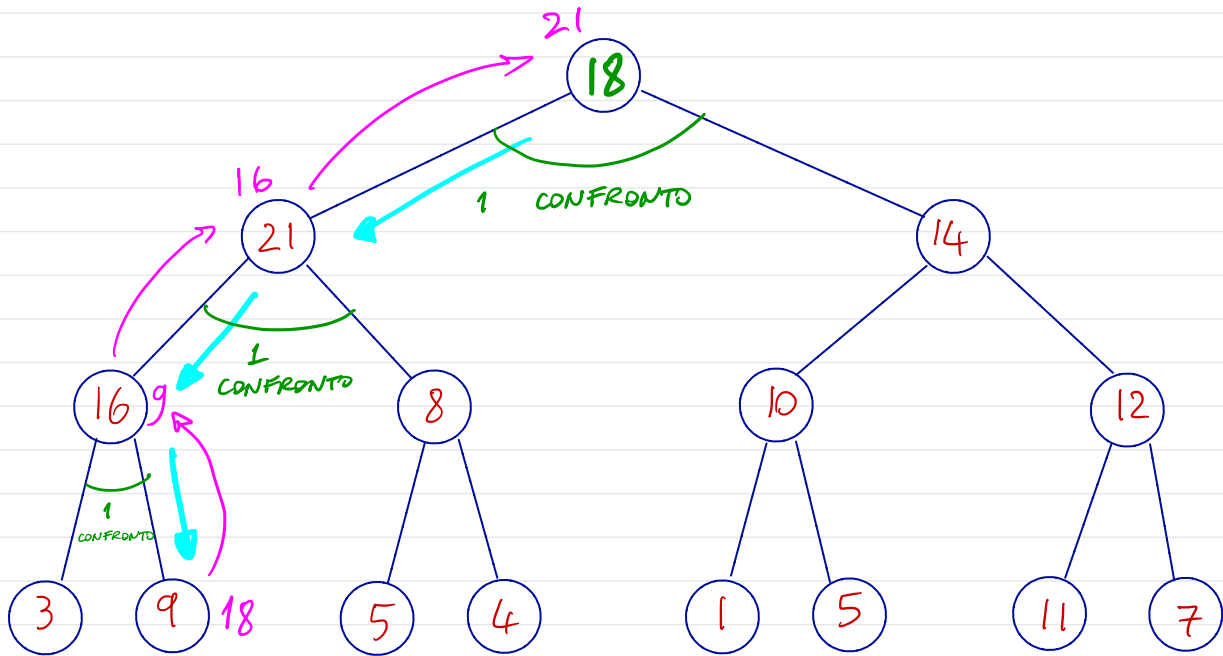
- DURANTE UN'OPERAZIONE DI SIFT-DOWN, PER CIASCUN LIVELLO VENGONO EFFETTUATI DUE CONFRONTI

- OLTRE AI CONFRONTI NECESSARI PER LA COSTRUZIONE DI UN HEAP, L'ESECUZIONE DI HEAPSORT PUO' DUNQUE RICHIEDERE FINO A  $2 \cdot \sum_{i=1}^n \lfloor \lg i \rfloor$  CONFRONTI

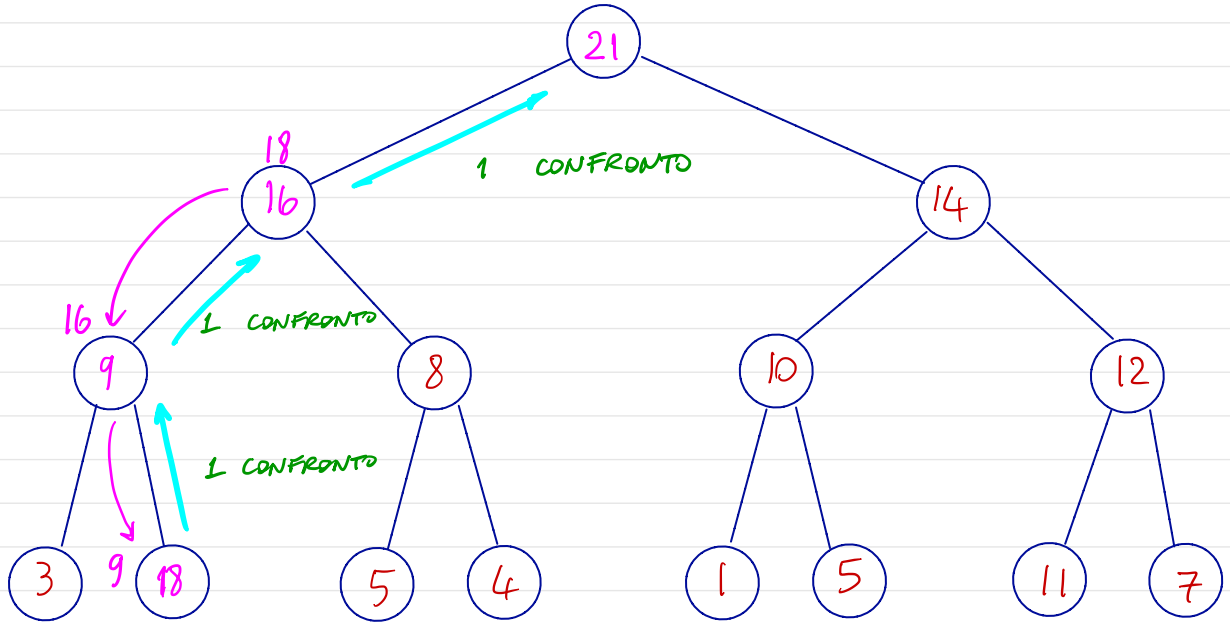
- E' POSSIBILE ABBASSARE IL VALORE DELLA COSTANTE 2 ?

- NELL'ALGORITMO BOTTOM-UP HEAPSORT C'E' UNA FASE INIZIALE DI DISCESA (SIFT-DOWN) CON UN SOLO CONFRONTO PER LIVELLO (COME SE LA CHIAVE FOSSE  $-\infty$ ), SEQUITA DA UNA FASE DI RISALITA (SIFT-UP)

# SIFT-DOWN (AD UN SOLO CONFRONTO)



# SIFT-UP



BENCHE' MEDIAMENTE LE FASI DI RISALITA SIANO MOLTO LIMITATE NEL CORSO DELL'ESECUZIONE DELL'ALGORITMO HEAPSORT, UNA RISPOSTA POSITIVA ALLA SEGUENTE DOMANDA CONSENTIREBBE UN'ULTERIORE RIDUZIONE DEL NUMERO DI CONFRONTI

DOMANDA: E' POSSIBILE **ELIMINARE** DEL TUTTO LA FASE DI RISALITA (SIFT-UP), PUR UTILIZZANDO LA PROCEDURA DI DISCESA (SIFT-DOWN) AD **UN SOLO** CONFRONTO?



UNA PRIMA SOLUZIONE (MA CON MEMORIA ESTERNA):

## EXTERNAL-HEAPSORT

Procedure External-Max-Heapsort ( $A, Ext$ )

INPUT: ARRAY  $A$  ED  $Ext$  DELLA STESSA LUNGHEZZA

OUTPUT: ARRAY  $Ext$  E' UNA PERMUTAZIONE ORDINATA DI  $A$

$n := \text{LENGTH}[A]$

BUILD-MAX-HEAP ( $A$ );

for  $j := n$  downto 1 do

$Ext[j] := A[1]$

$l := \text{Max-Special-Leaf}(A)$

$A[l] := ??$

end\_for;

end\_procedure;

## Function Max-SpecialLeaf (A)

$n := \text{LENGTH}[A]$

$i := 2$

/\* FIGLIO SINISTRO DELLA RADICE \*/

while  $i < n$  do

/\*  $i$  HA UN FRATELLO \*/

if  $A[i] < A[i+1]$  then  $i := i+1$ ; end if;

$A[\lfloor i/2 \rfloor] := A[i]$

/\* FA RISALIRE DI UN LIVELLO IL MAX  
TRA  $A[i]$  E IL FRATELLO DI  $A[i]$  \*/

$i := 2 \cdot i$ ;

/\* QUINDI SI SCENDE DI UN LIVELLO \*/

end\_while;

if  $i = n$  then

$A[\lfloor i/2 \rfloor] := A[n]$

$i := 2i$ ;

end\_if;

return  $i/2$ ;

end\_function;



• COME FARE A MENO DELL'ARRAY ESTERNO ?

• COME FARE A MENO DELL'ARRAY ESTERNO ?

OVVIAMENTE, UTILIZZANDO SOLTANTO L'ARRAY  
DI INPUT  $A[1..m]$

• E COME ?

• COME FARE A MENO DELL'ARRAY ESTERNO ?

OVVIAMENTE, UTILIZZANDO SOLTANTO L'ARRAY  
DI INPUT  $A[1..m]$

• E COME ?

UTILIZZANDO PARTE DELL'ARRAY  $A[1..m]$  COME HEAP E  
PARTE COME ARRAY ESTERNO

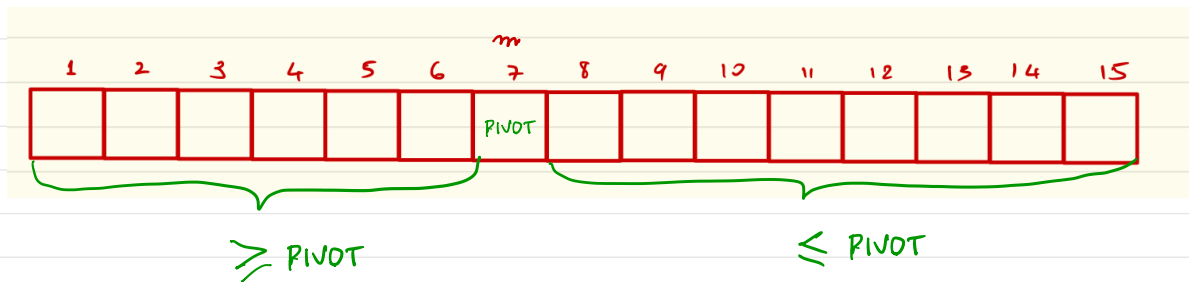
• INIZIALMENTE SI SELEZIONA UN PIVOT  $\text{PIVOT} = A[p]$  IN  $A[1..n]$

• QUINDI SI EFFETTUA UNA PARTIZIONE DI  $A[1..n]$  IN DUE SUB-ARRAY  $A[1..m-1]$ ,  $A[m+1..n]$  TALI CHE:

$\Delta A[m] = \text{PIVOT}$

$\Delta$  GLI ELEMENTI DI  $A[1..m-1]$  SIANO  $\geq \text{PIVOT}$

$\Delta$  GLI ELEMENTI DI  $A[m+1..n]$  SIANO  $\leq \text{PIVOT}$



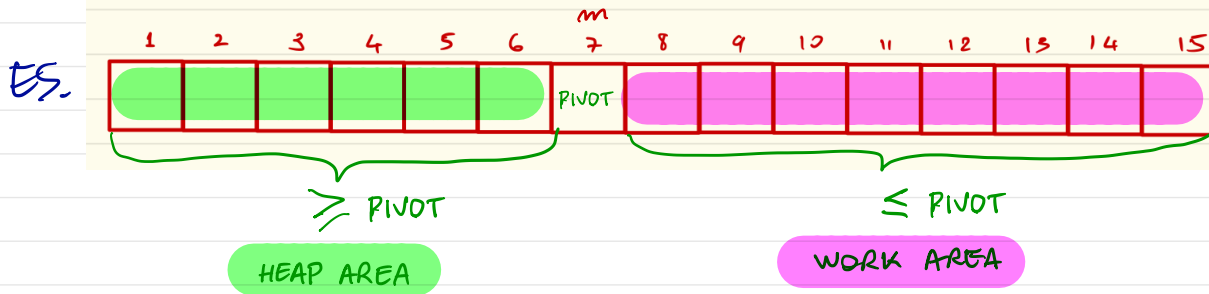
$\Delta$  LA PARTE PIÙ PICCOLA COSTITUIRÀ L'HEAP AREA, MENTRE QUELLA PIÙ GRANDE COSTITUIRÀ LA WORK AREA.

• SUPPONIAMO CHE  $m \leq \frac{n+1}{2}$

• IN TAL CASO

-  $A[1..m-1]$  COSTITUIRA' L' HEAP AREA

-  $A[m+1..n]$  COSTITUIRA' LA WORK AREA



• SI COSTRUISCA UN MAX-HEAP IN  $A[1..m-1]$

• SI ESEGUA L'ALGORITMO EXTERNAL-HEAPSORT SU  $A[1..m-1]$   
CONSIDERANDO COME ARRAY ESTERNO IL SUBARRAY  $A[n-m+2..n]$

• QUINDI SI SCAMBINO I VALORI DI  $A[m]$  E  $A[n-m+1]$



• SUPPONIAMO CHE  $m \leq \frac{n+1}{2}$

• IN TAL CASO

-  $A[1..m-1]$  COSTITUIRA' L' HEAP AREA

-  $A[m+1..n]$  COSTITUIRA' LA WORK AREA

• SI COSTRUISCA UN MAX-HEAP IN  $A[1..m-1]$

• SI ESEGUA L'ALGORITMO EXTERNAL-HEAPSORT SU  $A[1..m-1]$   
CONSIDERANDO COME ARRAY ESTERNO IL SUBARRAY  $A[n-m+2..n]$

• QUINDI SI SCAMBINO I VALORI DI  $A[m]$  E  $A[n-m+1]$

E GLI ELEMENTI CHE STAVANO IN  $A[n-m+2..n]$  DOVE  
VANNO A FINIRE ?

• SUPPONIAMO CHE  $m \leq \frac{n+1}{2}$

• IN TAL CASO

-  $A[1..m-1]$  COSTITUIRA' L' HEAP AREA

-  $A[m+1..n]$  COSTITUIRA' LA WORK AREA

• SI COSTRUISCA UN MAX-HEAP IN  $A[1..m-1]$

• SI ESEGUA L'ALGORITMO EXTERNAL-HEAPSORT SU  $A[1..m-1]$   
CONSIDERANDO COME ARRAY ESTERNO IL SUBARRAY  $A[n-m+2..n]$

• QUINDI SI SCAMBINO I VALORI DI  $A[m]$  E  $A[n-m+1]$

E GLI ELEMENTI CHE STAVANO IN  $A[n-m+2..n]$  DOVE  
VANNO A FINIRE ?

SONO UTILIZZATI AL POSTO DI  $-\infty$  !

## Procedure External-Max-Heapsort ( $A, Ext$ )

INPUT: ARRAY  $A$  ED  $Ext$  DELLA STESSA LUNGHEZZA

OUTPUT: ARRAY  $Ext$  E' UNA PERMUTAZIONE ORDINATA DI  $A$

$n := LENGTH[A]$

BUILD-MAX-HEAP ( $A$ );

for  $j := n$  downto 1 do

$temp := Ext[j]$

$Ext[j] := A[1]$

$l := Special-Leaf(A)$

$A[l] := temp$

end-for;

end-procedure;

• SUPPONIAMO CHE  $m \leq \frac{n+1}{2}$

• IN TAL CASO

-  $A[1..m-1]$  COSTITUIRA' L' HEAP AREA

-  $A[m+1..n]$  COSTITUIRA' LA WORK AREA

• SI COSTRUISCA UN MAX-HEAP IN  $A[1..m-1]$

• SI ESEGUA L'ALGORITMO EXTERNAL-HEAPSORT SU  $A[1..m-1]$  CONSIDERANDO COME ARRAY ESTERNO IL SUB-ARRAY  $A[n-m+2..n]$

• QUINDI SI SCAMBINO I VALORI DI  $A[m]$  E  $A[n-m+1]$

• IL SUB-ARRAY  $A[n-m+1..n]$  RISULTERA'

- ORDINATO

-  $A[i] \leq A[j]$ , PER  $1 \leq i \leq n-m < j \leq n$ .

DUNQUE BASTERA' ORDINARE RICORSIVAMENTE IL SUB-ARRAY  $A[1..n-m]$  PER COMPLETARE L'ORDINAMENTO.

ESEMPIO:

PIVOT



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
15	7	13	2	8	3	1	4	12	9	10	5	11	6	14

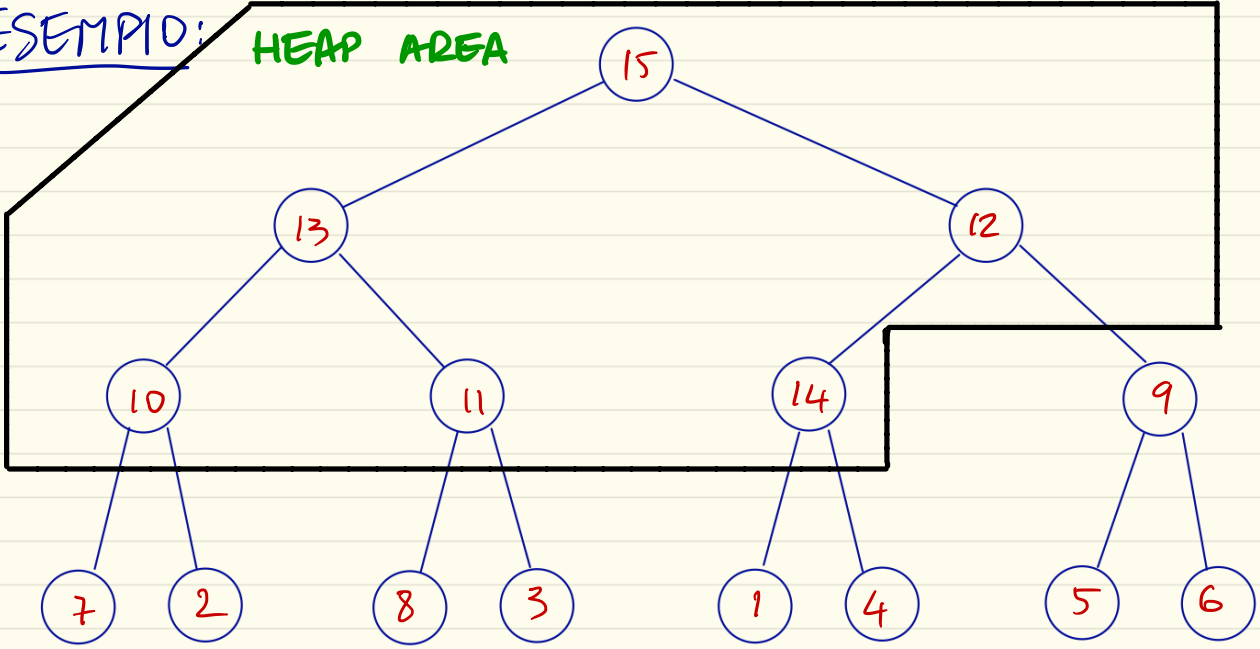
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
15	13	12	10	11	14	9	7	2	8	3	1	4	5	6

HEAP AREA

WORK AREA

ESEMPIO:

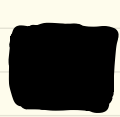
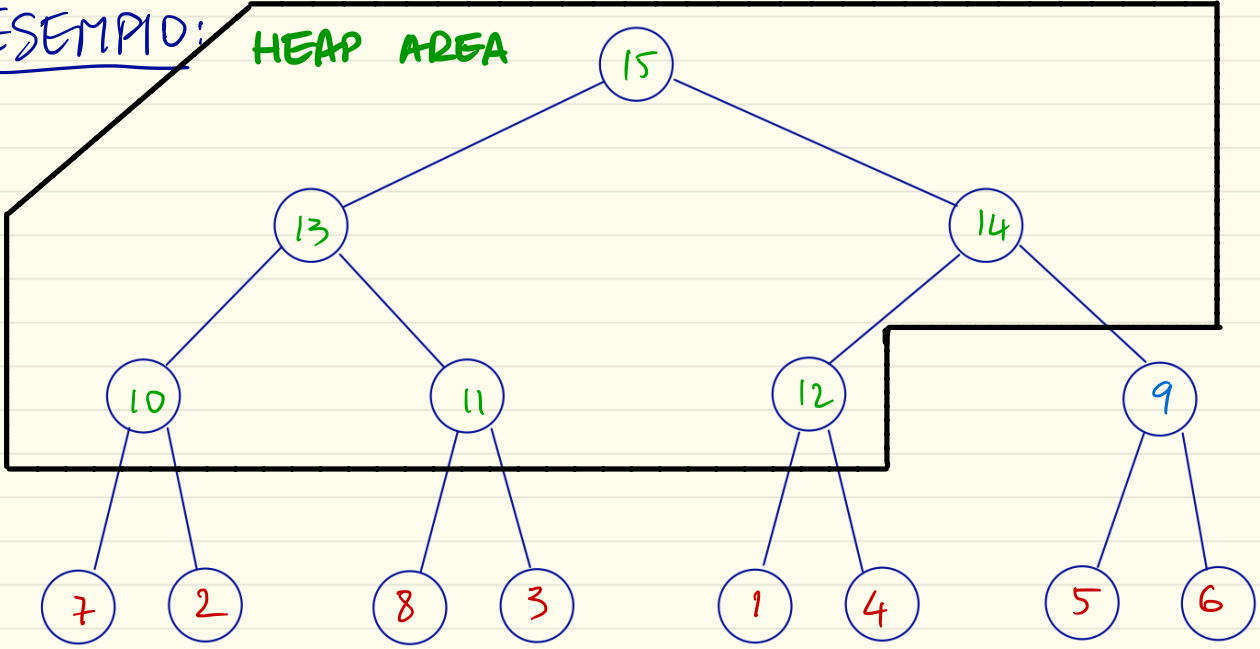
HEAP AREA



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
15	13	12	10	11	14	9	7	2	8	3	1	4	5	6

ESEMPIO:

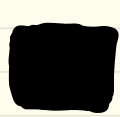
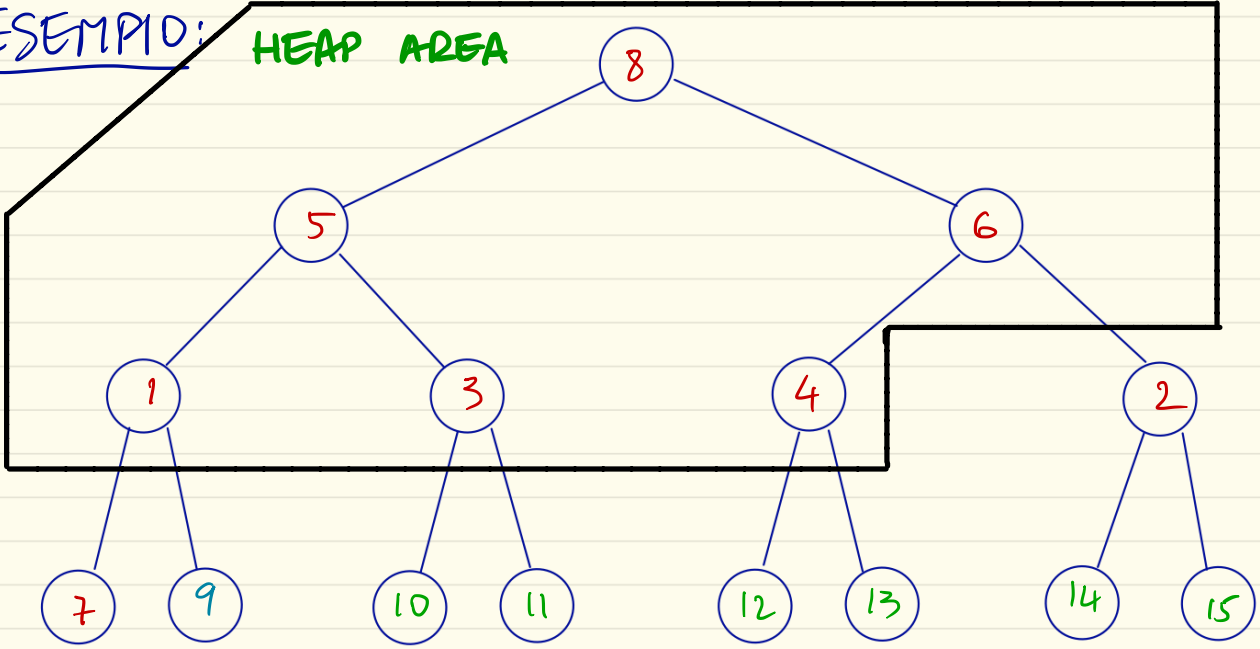
HEAP AREA



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
15	13	14	10	11	12	9	7	2	8	3	1	4	5	6

ESEMPIO:

HEAP AREA



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
8	5	6	1	3	4	2	7	9	10	11	12	13	14	15

NON ORDINATO

ORDINATO



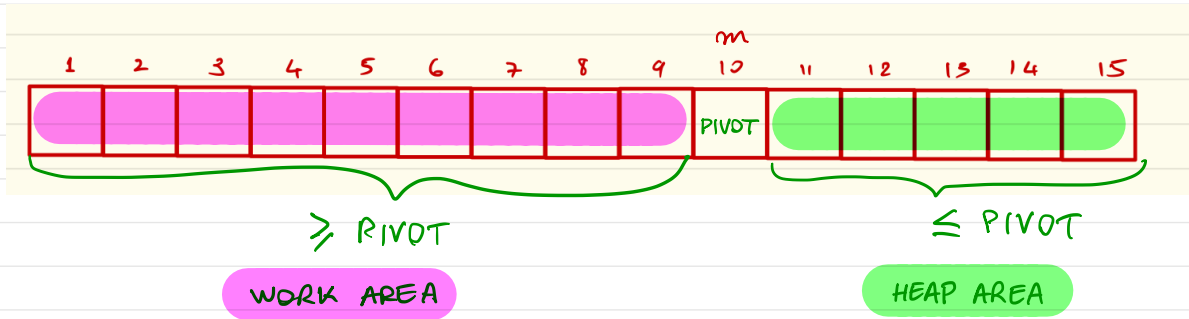
• SUPPONIAMO ORA CHE  $m > \frac{n+1}{2}$

• IN TAL CASO

-  $A[m+1..n]$  COSTITUIRA' L' HEAP AREA

-  $A[1..m-1]$  COSTITUIRA' LA WORK AREA

ES.



• SI COSTRUISCA UN MIN-HEAP IN  $A[m+1..n]$

• SI ESEGUA L'ALGORITMO EXTERNAL-MIN-HEAPSORT SU  $A[m+1..n]$   
CONSIDERANDO COME ARRAY ESTERNO IL SUBARRAY  $A[1..m-1]$

• QUINDI SI SCAMBINO I VALORI DI  $A[m]$  E  $A[n-m+1]$

• SUPPONIAMO ORA CHE  $m > \frac{n+1}{2}$

• IN TAL CASO

-  $A[m+1..n]$  COSTITUIRA' L' HEAP AREA

-  $A[1..m-1]$  COSTITUIRA' LA WORK AREA

• SI COSTRUISCA UN MIN-HEAP IN  $A[m+1..n]$

• SI ESEGUA L'ALGORITMO EXTERNAL-MIN-HEAPSORT SU  $A[m+1..n]$  CONSIDERANDO COME ARRAY ESTERNO IL SUBARRAY  $A[1..m-1]$

• QUINDI SI SCAMBINO I VALORI DI  $A[m]$  E  $A[n-m+1]$

• IL SUB-ARRAY  $A[n-m+1]$  RISULTERA'

- ORDINATO

-  $A[i] \leq A[j]$ , PER  $1 \leq i \leq n-m+1 < j \leq n$ .

DUNQUE BASTERA' ORDINARE RICORSIVAMENTE IL SUB-ARRAY  $A[n-m+2..n]$  PER COMPLETARE L'ORDINAMENTO.

ESEMPIO:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
15	7	13	2	8	3	1	4	12	9	10	5	11	6	14

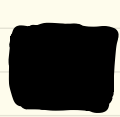
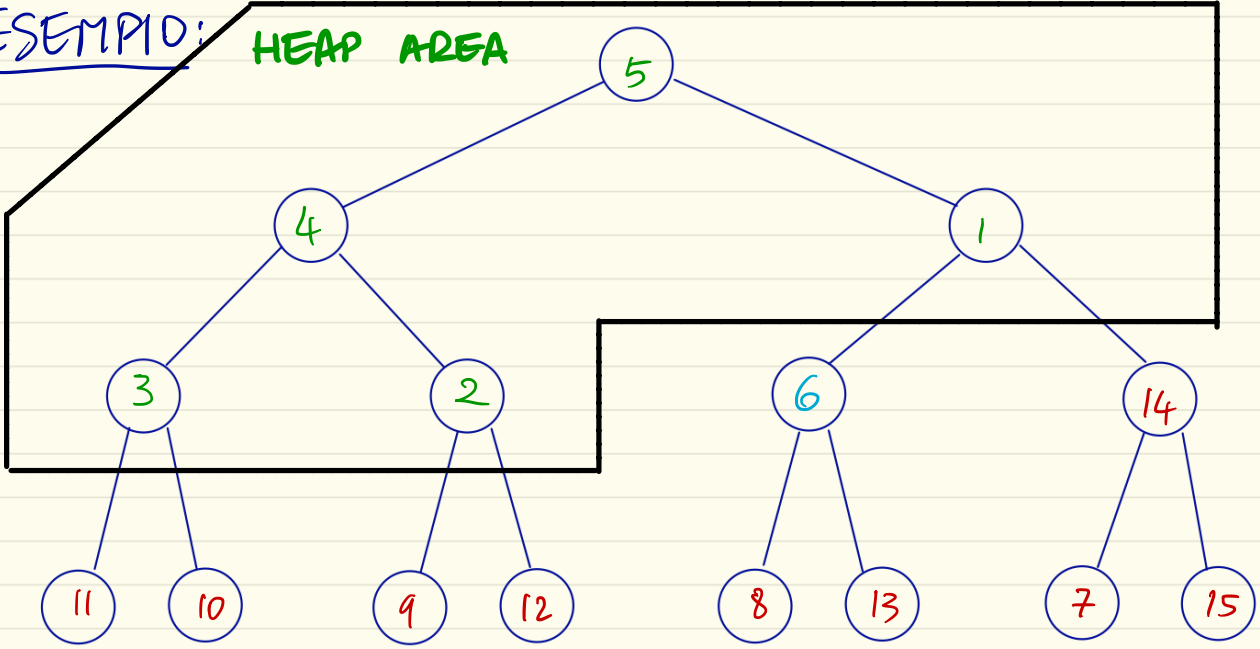
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
15	7	13	8	12	9	10	11	14	6	2	3	1	4	5

$\geq 6$   
WORK AREA

$\leq 6$   
HEAP AREA

ESEMPIO:

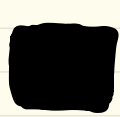
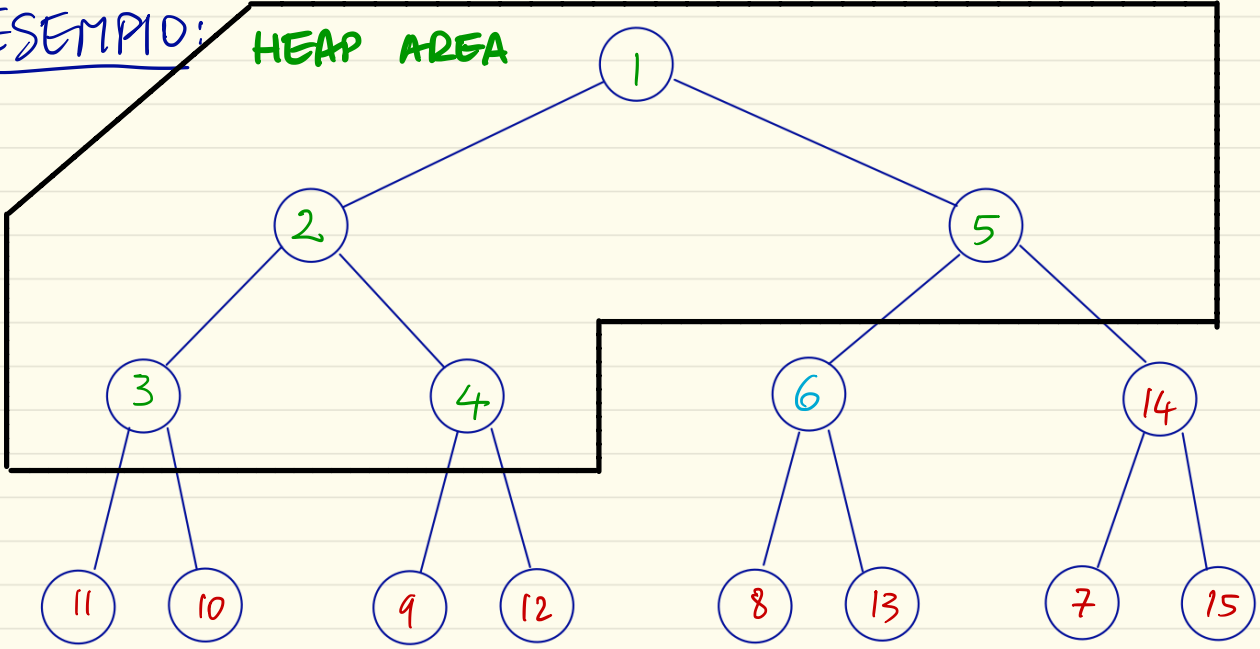
HEAP AREA



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
15	7	13	8	12	9	10	11	14	6	2	3	1	4	5

ESEMPIO:

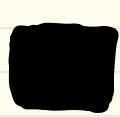
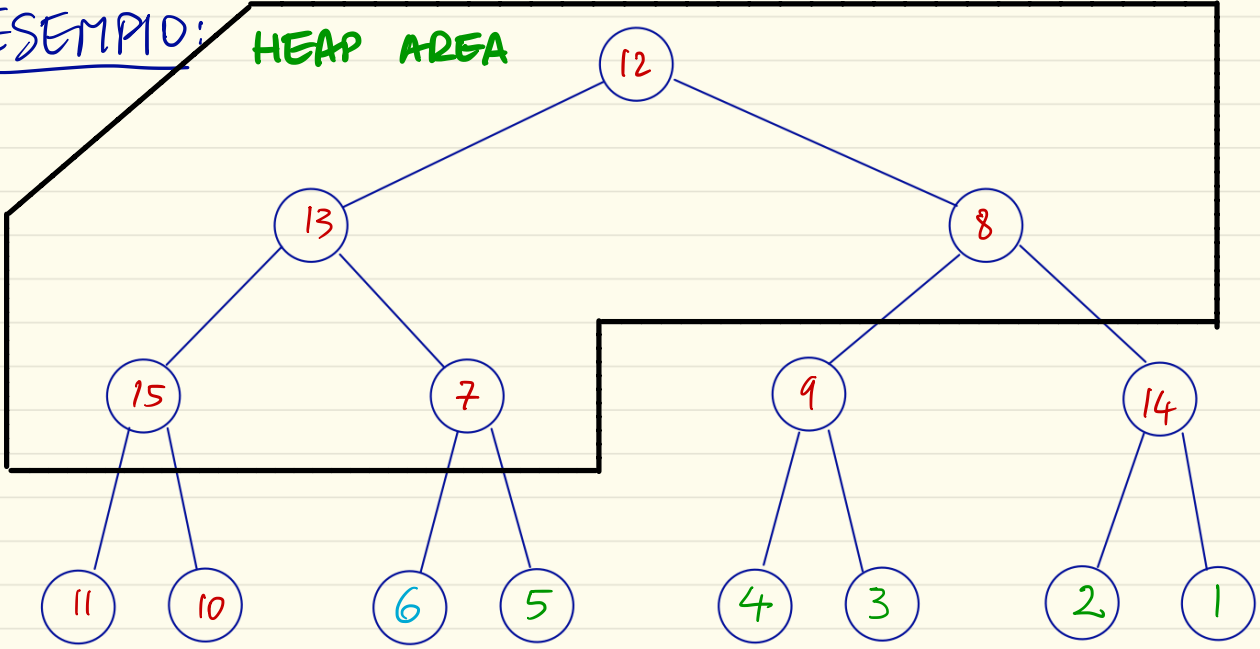
HEAP AREA



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
15	7	13	8	12	9	10	11	14	6	4	3	5	2	1

ESEMPIO:

HEAP AREA



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	2	3	4	5	6	10	11	14	9	7	15	8	13	12

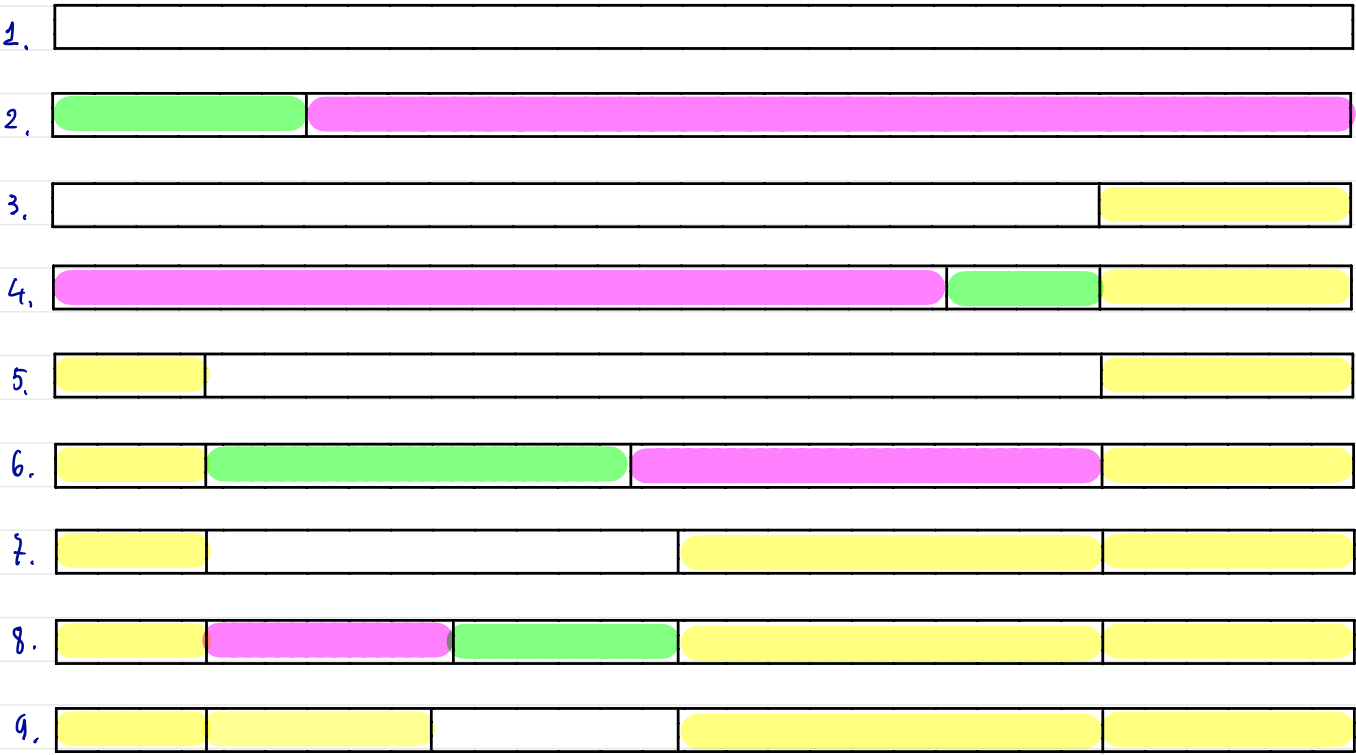
ORDINATO

NON ORDINATO

# ESEMPIO

WORK AREA  
HEAP AREA

SORTED AREA  
UNSORTED AREA



⋮ ecc, ⋮

procedure QuickHeapsort( $A$ );

$n := \text{LENGTH}(A)$ ;

if  $n > 1$  then

$\text{PIVOT} := \text{CHOOSE-PIVOT}(A)$ ;

$m := \text{PARTITION-REVERSE}(A, \text{PIVOT})$ ;

if  $m \leq \frac{n+1}{2}$  then

        External-Max-Heapsort( $A[1..m-1]$ ,  $A[n-m+2..n]$ );

        Exchange( $A[m]$ ,  $A[n-m+1]$ )

        QuickHeapsort( $A[1..n-m]$ )

else

        External-Min-Heapsort( $A[m+1..n]$ ,  $A[1..n-m]$ );

        Exchange( $A[m]$ ,  $A[n-m+1]$ );

        QuickHeapsort( $A[n-m+2..n]$ );

end-if;

end-if;

end-procedure;



## Procedure External-Min-Heapsort ( $A, Ext$ )

INPUT: ARRAY  $A$  ED  $Ext$  DELLA STESSA LUNGHEZZA

OUTPUT: ARRAY  $Ext$  E' UNA PERMUTAZIONE ORDINATA DI  $A$

$n := \text{LENGTH}[A]$

BUILD-MIN-HEAP ( $A$ );

for  $j := 1$  downto  $n$  do

temp :=  $Ext[j]$

$Ext[j] := A[1]$

$l := \text{Min-Special-Leaf}(A)$

$A[l] := \text{temp}$

end\_for;

end\_procedure;

## Function Min-Special Leaf (A)

$n := \text{LENGTH}[A]$

$i := 2$

while  $i < n$  do

$\nexists A[i] > A[i+1]$  then  $i := i+1$ ; end  $\nexists$ ;

$A[\lfloor i/2 \rfloor] := A[i]$

$i := 2 \cdot i$ ;

end\_while;

if  $i = n$  then

$A[\lfloor i/2 \rfloor] := A[n]$

$i := 2i$ ;

end\_if;

return  $i/2$ ;

end\_function;

/\* FIGLIO SINISTRO DELLA RADICE \*/

/\* i HA UN FRATELLO \*/

/\* FA RISALIRE DI UN LIVELLO IL MIN  
TRA  $A[i]$  E IL FRATELLO DI  $A[i]$  \*/

/\* QUINDI SI SCENDE DI UN LIVELLO \*/

# COMPLESSITA'

**Theorem 1** The expected number  $\mathbb{E}[T(n)]$  of comparisons by basic (resp. improved) QuickHeapsort with pivot as median of  $k$  randomly selected elements on an input array of size  $n$  satisfies  $\mathbb{E}[T(n)] \leq n \lg n + c_k n + o(n)$  with  $c_k$  as follows:

$k$	<i>Cantone - Cincotti</i> $c_k$ basic QHS variant	<i>Diekert - Weiss</i> $c_k$ improved QHS variant
1	+2.72	+1.97
3	+1.92	+1.17
$f(n)$	+0.72	-0.03

2.996

old result (2002)  
basic QHS

The last row is valid for all  $f \in \omega(1) \cap o(n)$  with  $1 \leq f(n) \leq n$ , e.g.,  $f(n) = \sqrt{n}$ .

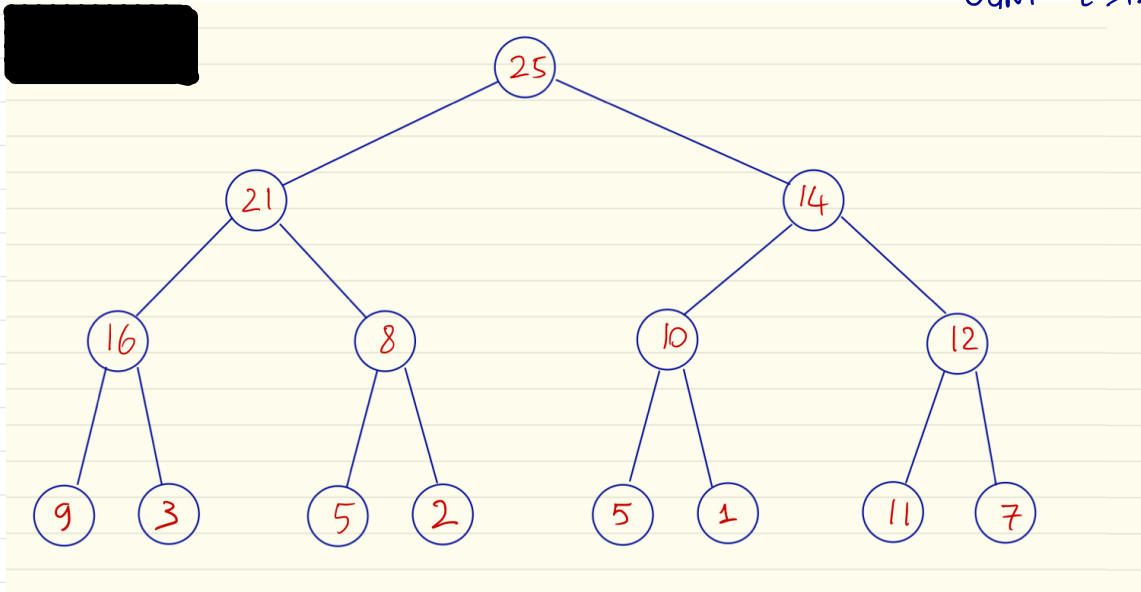
Clever Quicksort needs  $1.18n \log_2 n$  comparisons on average

L'ANALISI DI DIEKERT-WEISS SI BASA SU UN ALGORITMO OTTIMIZZATO PER LA COSTRUZIONE DI UN HEAP APPARSO NEL 2012:

Chen, J., Edelkamp, S., Elmasry, A., Katajainen, J.: In-place Heap Construction with Optimized Comparisons, Moves, and Cache Misses. In: B. Rován, V. Sassone, P. Widmayer (eds.) MFCS, vol. 7464 of LNCS, pp. 259–270. Springer (2012)

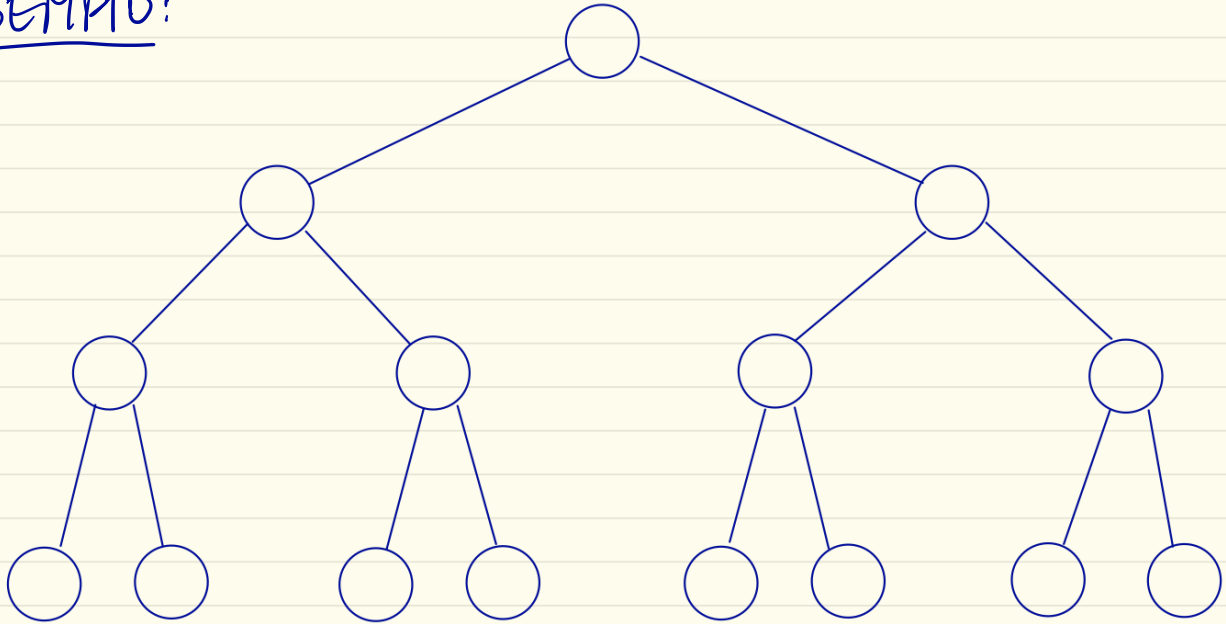
UN MIGLIORAMENTO PER RISPARMIARE CIRCA  $\frac{3}{4}n$  CONFRONTI  
(DIEKERT-WEISS, 2015)

- E' SUFFICIENTE ORDINARE LE FOGLIE CON UNO STESSO PADRE IN SENSO NON-DECRESCENTE, NEL CASO DI MAX-HEAP, E IN SENSO NON-CRESCENTE, NEL CASO DI MIN-HEAP, MANTENENDO POI TALE ORDINAMENTO DOPO OGNI ESTRAZIONE

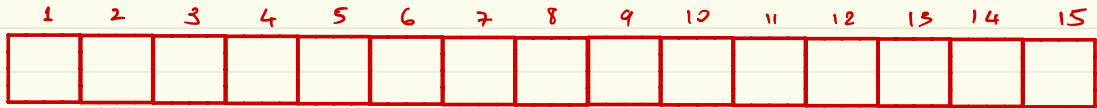




ESEMPIO:



Ext:



LEMMA LET  $\alpha, \beta \in \mathbb{R}$ ,  $i \in \mathbb{N}$ . THEN

$$\alpha < i < \beta \iff \lfloor \alpha \rfloor + 1 \leq i \leq \lceil \beta \rceil - 1. \quad \square$$

---

Fathers of two leaves can be characterized as follows:

In a heap with  $n$  elements,

$$\text{node } i \text{ is father of two leaves} \iff 2i+1 \leq n \text{ \& } 4i > n$$

$$\iff n < 4i \leq 2n - 2$$

$$\iff \frac{n}{4} < i \leq \frac{2n-2}{4}$$

$$\iff \left\lfloor \frac{n}{4} \right\rfloor + 1 \leq i \leq \left\lfloor \frac{2n-2}{4} \right\rfloor \quad \square$$

Hence, in a heap with  $n$  nodes, the number of nodes that are fathers of two leaves is:

$$\left\lfloor \frac{2n-2}{4} \right\rfloor - \left( \left\lfloor \frac{n}{4} \right\rfloor + 1 \right) + 1 = \left\lfloor \frac{2n-2}{4} \right\rfloor - \left\lfloor \frac{n}{4} \right\rfloor$$

LEMMA  $\left\lfloor \frac{n}{4} \right\rfloor - 1 \leq \left\lfloor \frac{2n-2}{4} \right\rfloor - \left\lfloor \frac{n}{4} \right\rfloor \leq \left\lfloor \frac{n}{4} \right\rfloor + 1$ . □

$h_i$  := dimension of  $i$ -th heap

$t$  := total number of heaps = total number of pivots

We have:  $\sum_{i=1}^t h_i = n - t$

The expected value of  $t$  is  $O(\lg n)$ .



The total number of comparisons due to the sift-down phase is:

$$\text{basic QHS} \rightarrow \sum_{i=1}^t \lfloor \lg h_i \rfloor \cdot (h_i - 1)$$

$$\text{improved QHS} \rightarrow \sum_{i=1}^t \left[ \lfloor \lg h_i \rfloor \cdot (h_i - 1) - (h_i - 1) \right]$$

$$\text{gross saving} \rightarrow \sum_{i=1}^t h_i - t = n - 2t$$

$$\text{extra cost of improved QHS} \rightarrow \left\lfloor \frac{n}{4} \right\rfloor + 1$$

$$\text{net saving} \rightarrow n - 2t - \left( \left\lfloor \frac{n}{4} \right\rfloor + 1 \right) \geq n - 2t - \frac{n}{4} - 1 = \frac{3}{4}n - 2t - 1$$

$$\text{Expected net saving} \rightarrow \frac{3}{4}n - O(\lg n) \simeq 0.75n$$

and put the right one at the place of the former left one. Summing up over all heaps during an execution of standard QuickHeapsort, we invest  $\frac{n+2t}{4}$  comparisons in order to save  $n$  comparisons, where  $t$  is the number of recursive calls. The expected number of  $t$  is in  $O(\log n)$ . Hence, we can expect to save  $\frac{3n}{4} + O(\log n)$  comparisons. We call this version the improved variant of QuickHeapsort.

LEMMA  $\lfloor \frac{n}{4} \rfloor - 1 \leq \lfloor \frac{2n-2}{4} \rfloor - \lfloor \frac{n}{4} \rfloor \leq \lfloor \frac{n}{4} \rfloor + 1$

Proof The claim is equivalent to

$$2\lfloor \frac{n}{4} \rfloor - 2 \leq \lfloor \frac{2n-2}{4} \rfloor \leq 2\lfloor \frac{n}{4} \rfloor + 2 \quad (*)$$

By way of contradiction, assume

$$\lfloor \frac{2n-2}{4} \rfloor > 2\lfloor \frac{n}{4} \rfloor + 2, \quad \lfloor \frac{n}{4} \rfloor = \frac{n}{4} - \frac{n \bmod 4}{4}$$

Then

$$\frac{2n-2}{4} \geq \lfloor \frac{2n-2}{4} \rfloor > 2\lfloor \frac{n}{4} \rfloor + 2 \stackrel{\text{since}}{=} 2\frac{n}{4} - 2\frac{n \bmod 4}{4} + 2$$

$$-\frac{1}{2} > -\frac{n \bmod 4}{2} + 2$$

$$\frac{3}{2} \geq \frac{n \bmod 4}{2} > 2 + \frac{1}{2} = \frac{5}{2}, \quad \text{contradiction,}$$

Thus,  $\lfloor \frac{2n-2}{4} \rfloor \leq 2\lfloor \frac{n}{4} \rfloor + 2$

- Next, by way of contradiction assume

$$2\left\lfloor \frac{n}{4} \right\rfloor - 2 > \left\lfloor \frac{2n-2}{4} \right\rfloor$$

Then

$$2\left\lfloor \frac{n}{4} \right\rfloor - 2 \geq 2\left\lfloor \frac{n}{4} \right\rfloor - 2 > \left\lfloor \frac{2n-2}{4} \right\rfloor = \frac{2n-2}{4} - \frac{(2n-2) \bmod 4}{4}$$

$$-2 > -\frac{1}{2} - \frac{(2n-2) \bmod 4}{4}$$

$$\frac{3}{4} \geq \frac{(2n-2) \bmod 4}{4} > \frac{3}{2}, \text{ contradiction.}$$

Thus,  $2\left\lfloor \frac{n}{4} \right\rfloor - 2 \leq \left\lfloor \frac{2n-2}{4} \right\rfloor$ , completing the proof of the lemma.  $\square$