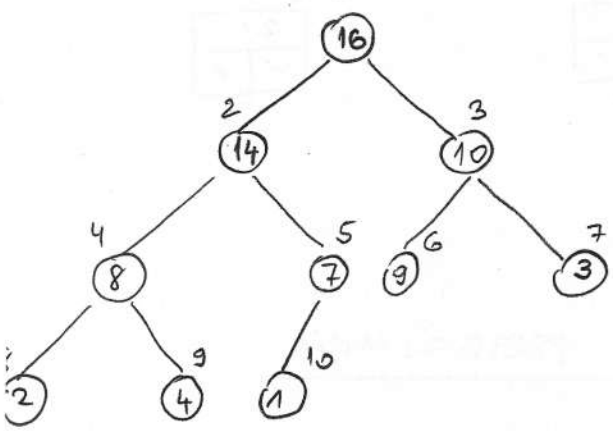
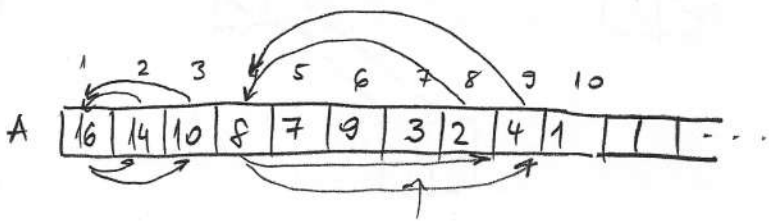
BINARNE HRPE

struktura podataka koja se može promatrati kao „skoro" potpuno binarno stablo (svaki čvor, osim nekolicine, ima točno dva djeteta)

koristi polje kao spremnik



↑
binarno stablo
(skoro potpuno)



Polje

length - duljina polja A
heap-size - broj elemenata u hrpi
} $0 \leq A.heap\text{-}size \leq A.length$

[1] - korijen stabla

PARENT (i) - indeks roditeljskog čvora od čvora s indeksom i u A

LEFT (i) - indeks lijevog djeteta od čvora s indeksom i u polju A

RIGHT (i) - indeks desnog djeteta od čvora s indeksom i u polju A

```
PARENT (i)
    if i = 1
        error: "Korijen nema roditelja"
    else then
        return ⌊i/2⌋
    end if
```

```
LEFT(i)
    if i > ⌊A.heap-size/2⌋
    |   error: "Čvor nema lijevo dijete
    else
    |   return 2i
    end if
```

```
RIGHT(i)
    if i > ⌊(A.heap-size - 1)/2⌋
    |   error: "Čvor nema desno dijete"
    else
    |   return 2i + 1
    end if
```

- razlikujemo dvije vrste binarnih hrpa:
  - * maksimalno orijentirane hrpe
  - * minimalno orijentirane hrpe

Svojstvo maksimalno orijentirane hrpe:

Za svaki $i = 2, \ldots, A.heap\text{-}size$

$A[\text{PARENT}(i)] \geq A[i]$

⇒ Najveći element u svakom podstablu nalazi se u korijenu tog podstabla

Svojstvo minimalno orijentirane hrpe:

Za svaki $i = 2, \ldots, A.heap\text{-}size$

$A[\text{PARENT}(i)] \leq A[i]$

⇒ Najmanji element u svakom podstablu nalazi se u korijenu tog podstabla

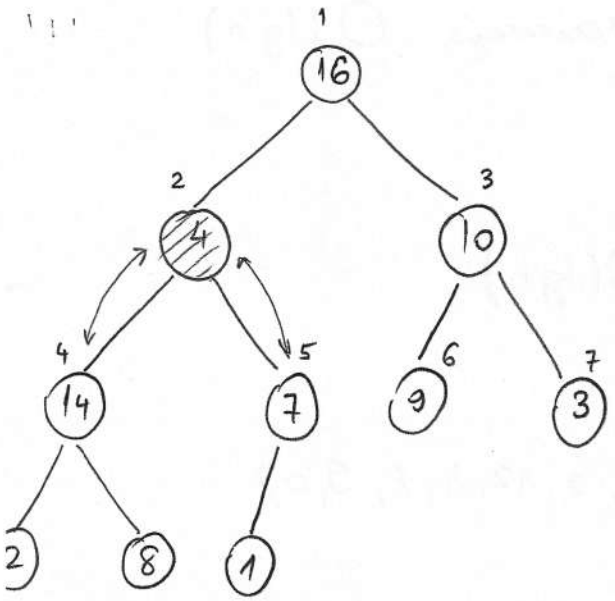
visina binarnog stabla kojim je implementirana binarna) hrpa od n elemenata iznosi $\Theta(\lg n)$
← zadaća

Razmotrit ćemo sljedeće operacije maksimalno orijentirane hrpe:

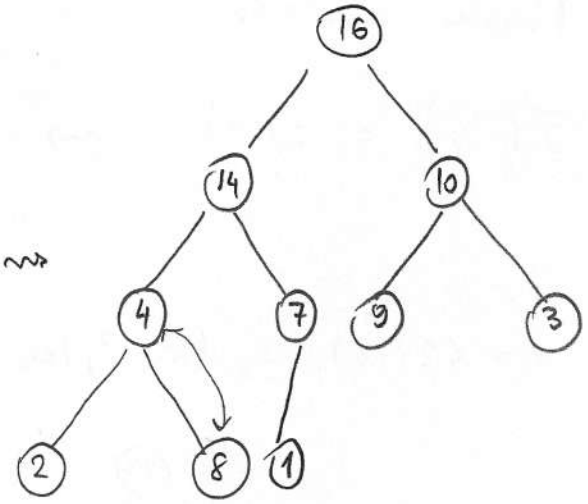
* MAX-HEAPIFY - služi za očuvanje svojstva maksimalno orijentirane hrpe
  Vrijeme: $O(\lg n)$
* BUILD-MAX-HEAP - služi za izgradnju maksimalno-orijentirane hrpe iz nesortiranog polja A
  Vrijeme: $O(n)$
* HEAPSORT - sortira polje A "in-place" (≤ "konstantno mnogo" dodatne memorije)
  Vrijeme: $O(n \lg n)$
* MAX-HEAP-INSERT - ubacivanje novog elementa
  Vrijeme: $O(\lg n)$
* HEAP-EXTRACT-MAX - brisanje najvećeg elementa
  Vrijeme: $O(\lg n)$
* HEAP-INCREASE-KEY - povećavanje vrijednosti ključa nekog elementa
  Vrijeme: $O(\lg n)$
* HEAP-MAXIMUM - uzimanje najvećeg elementa
  Vrijeme: $O(\lg n)$

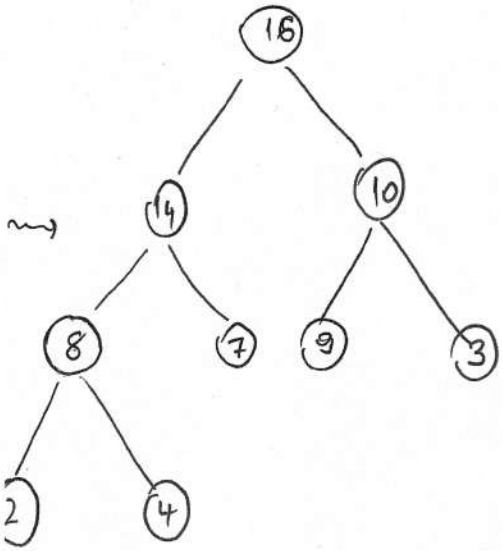
Održavanje svojstva hrpe

- ovdje ćemo implementirati funkciju MAX-HEAPIFY(A, i)
- pretpostavljamo da svojstvo može biti narušeno!



$A[\text{PARENT}(i)] \geq A[i]$






```
MAX-HEAPIFY(A, i)
l ← LEFT(i)
r ← RIGHT(i)
if l ≤ A.heap-size and A[l] > A[i]
    then largest ← l
    else largest ← i
end if
if r ≤ A.heap-size and A[r] > A[largest]
    then largest ← r
end if
if largest ≠ i
    then A[i] ↔ A[largest]  (zamjena)
         MAX-HEAPIFY(A, largest)
```

Vrijeme izvršavanja:

- sratnjeno visini binarnog stabla - $O(h)$

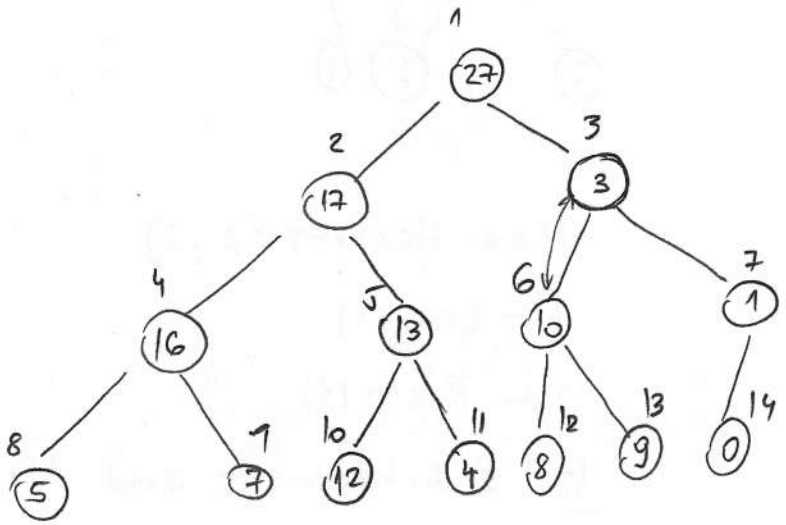
$\rightsquigarrow$ kako je visina bin. stabla $\Theta(\lg n)$

zaključujemo da je vrijeme izvršavanja $O(\lg n)$

i preko Master metode

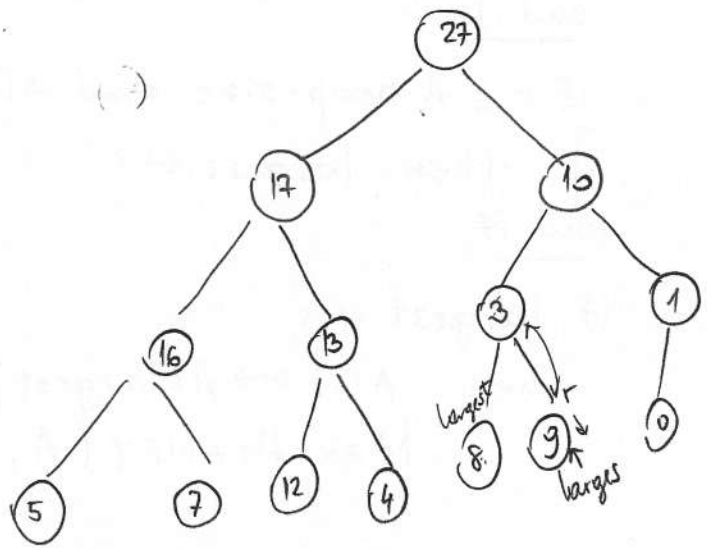
$$T(n) = T(n/2) + \Theta(1) \quad \rightsquigarrow \quad \Theta(\lg n)$$

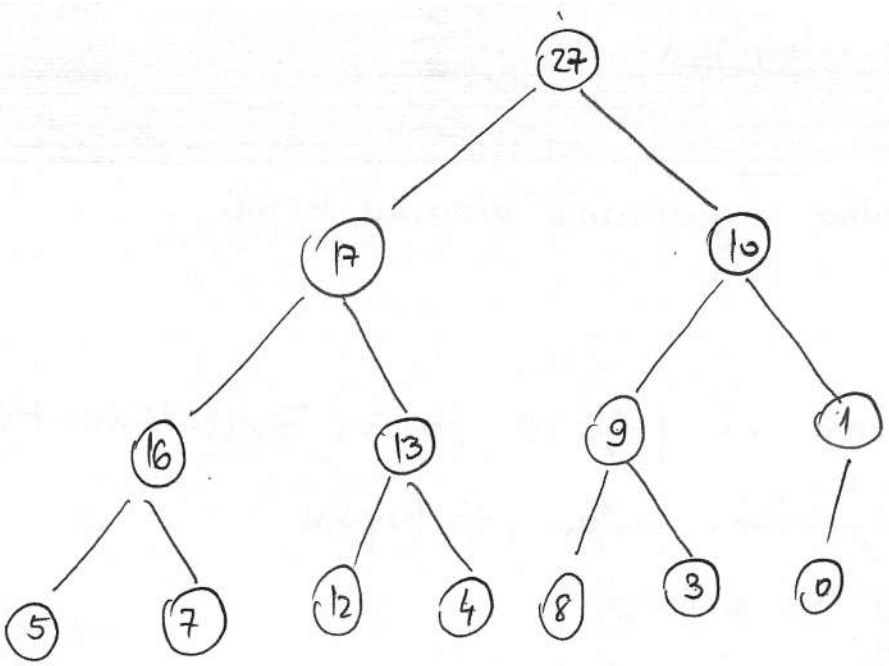
Primjer: $A = \langle 27, 17, 3, 16, 13, 10, 1, 5, 7, 12, 4, 8, 9, 0 \rangle$



MAX-HEAPIFY (A, 3)

$\rightsquigarrow$





STOP.

Izgradnja maksimalno orijentirane hrpe

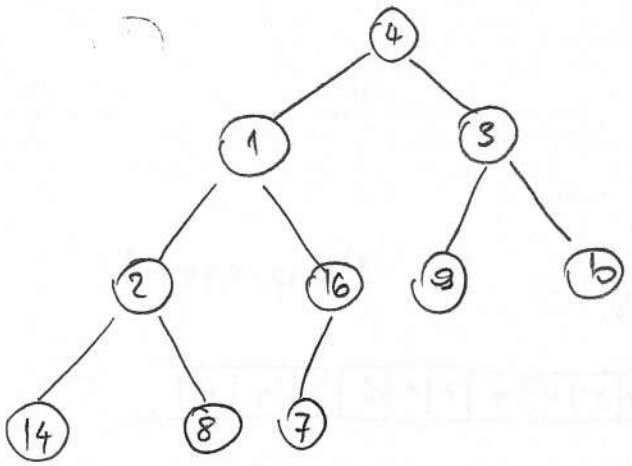
PROBLEM: Pretvoriti polje $A[1, \ldots, n]$, $n = A.length$ u maksimalno orijentiranu hrpu

TVRDNJA:

* Listovi se nalaze u potpolju $A[(\lfloor n/2 \rfloor + 1), \ldots, n]$
* Hrpa od n elemenata ima visinu $\lfloor \lg n \rfloor$

Primjer: $A =$

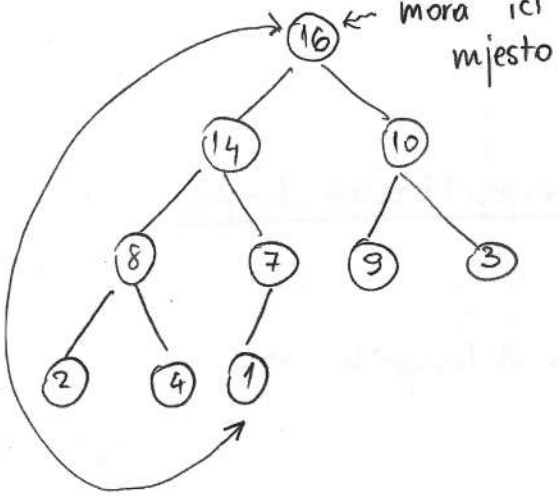
4	1	3	2	16	9	10	14	8	7



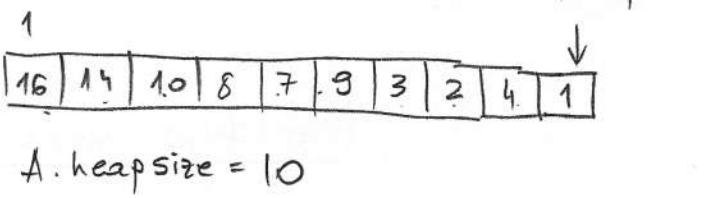
HEAPSORT algoritam

deja:
- izgraditi maksimalno orijentiranu binarnu hrpu iz polja A

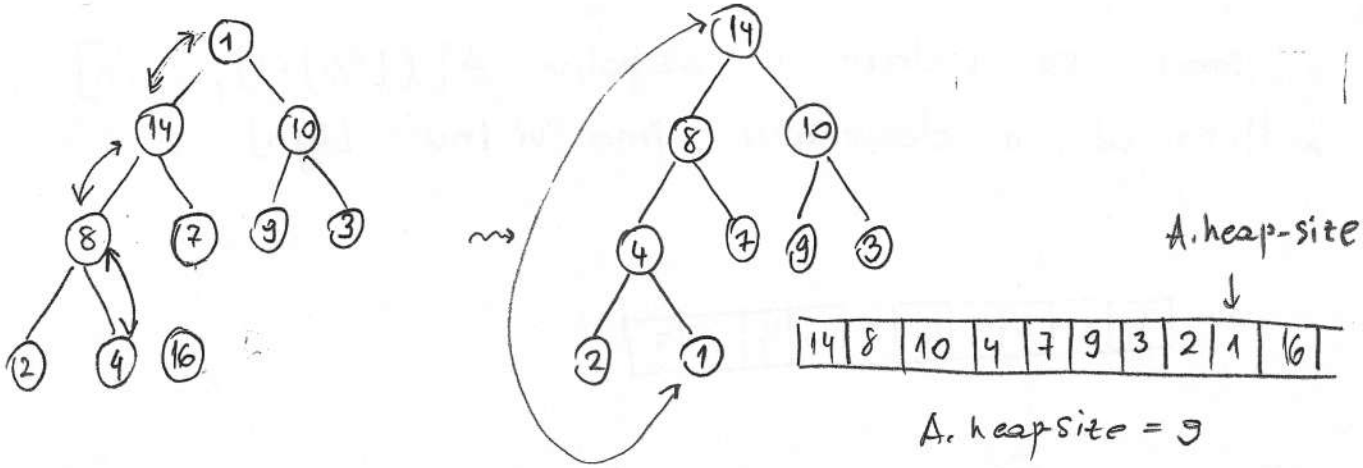
rimjer: pretpostavimo da smo na polju A pozvali BUILD-MAX-HEAP to je rezultiralo sljedećim poljem, tj. hrpom



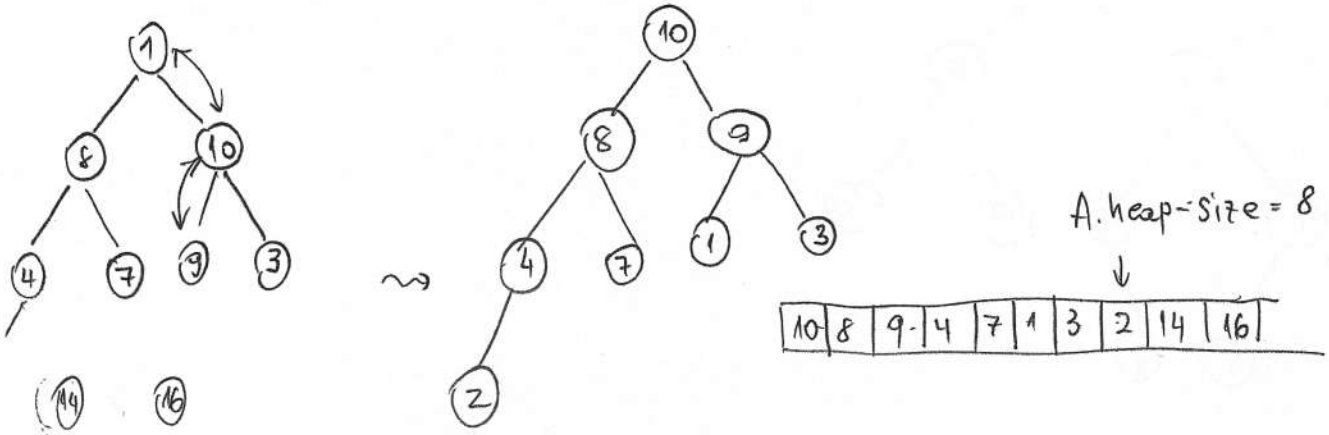
← mora ići na posljednje mjesto



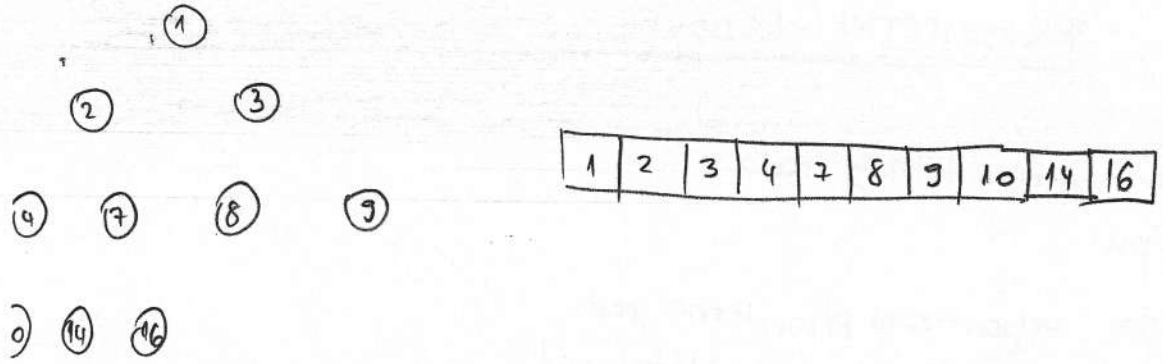
A.heapsize = 10



A.heapsite = 9



nastavljamo sve dok ne ispraznimo hrpu



HEAPSORT(A)

BUILD-MAX-HEAP(A) → O(n)

for i ← A.length downto 2

do A[1] ↔ A[i]

A.heap-size ← A.heap-size - 1

MAX-HEAPIFY(A, 1)

end for

Vrijeme izvršavanja $T(n) = O(n) + n \cdot O(\lg n) = O(n \lg n)$

PRIORITETNI REDOVI

najvažnija primjena prioritetnog reda

razlikujemo dva tipa:

* maksimalno orijentirani prioritetni red
* minimalno orijentirani prioritetni red

prioritetni red: - red u kojem svaki element ima ključ
- ključ predstavlja prioritet elemenata u redu

maksimalno orijentirani prioritetni red:

⇒ element s najvećim ključem prvi „ide van"

minimalno orijentirani prioritetni red:

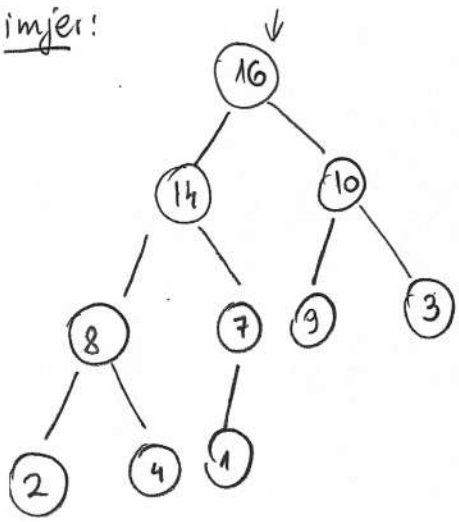
⇒ element s najmanjim ključem prvi „ide van"

operacije:

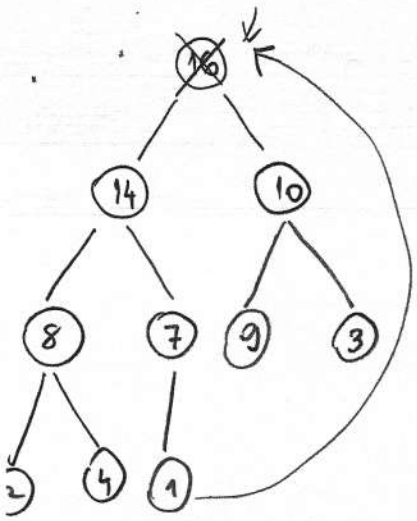
* MAXIMUM
* EXTRACT-MAX

* INCREASE-KEY
* INSERT

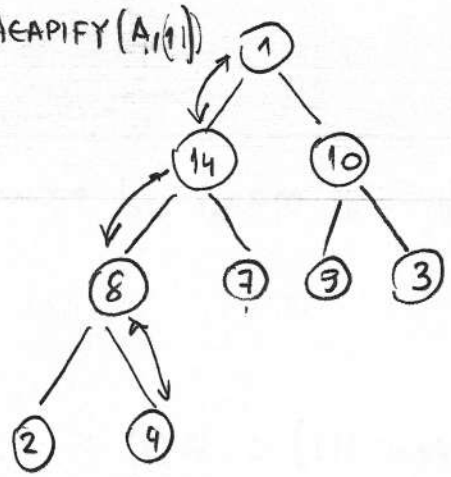
primjer:

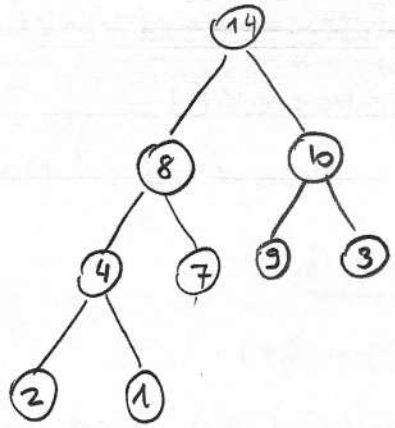



```
MAXIMUM (S)
    return A[1]
```



MAX-HEAPIFY(A,1)

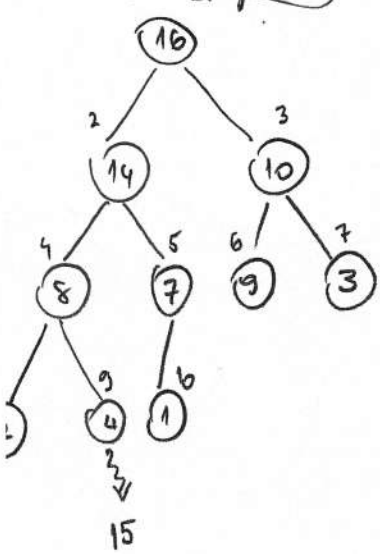


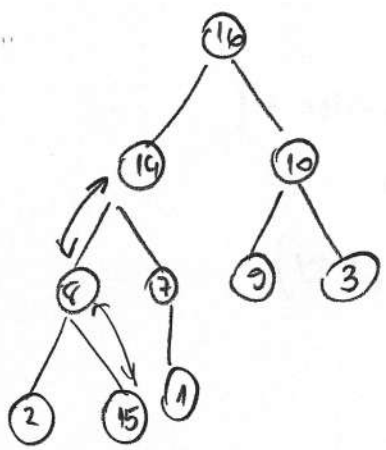


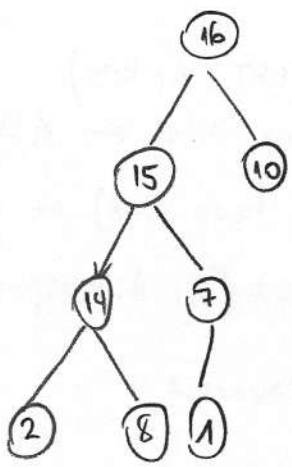
```
EXTRACT-MAX (A)
if  A.heap-size < 1
|   then error "heap underflow"
end if
max ← A[1]
A[1] ← A[A.heap-size]
A.heap-size ← A.heap-size -1
MAX-HEAPIFY (A, 1)
return max
```

Ubacivanje elementa u prioritetni red

Povećanje (vrijednost ključa / prioriteta)

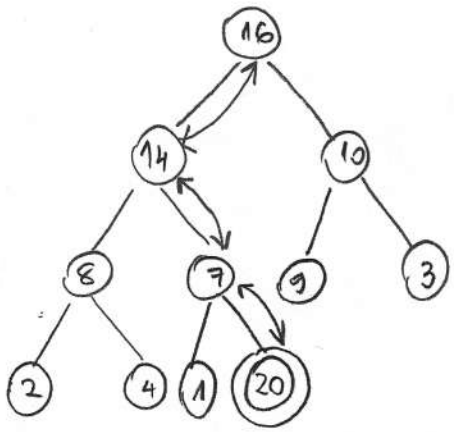


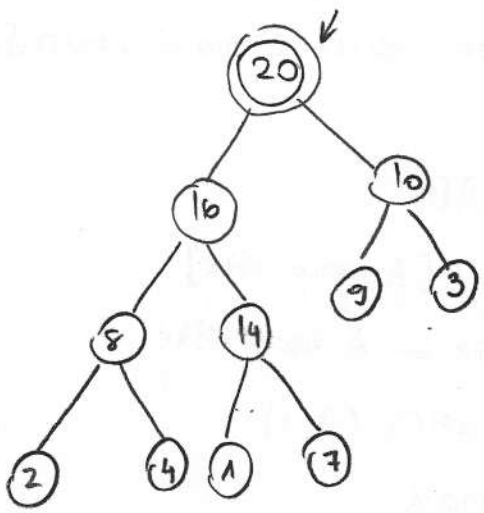




- ideja: praviti zamjene s roditeljskim čvorom sve dok je svojstvo maksimalno orijentirane hrpe narušeno

```
INCREASE-KEY(A, i, key)
if key < A[i]
   then error: "Novi ključ je manji od trenutnog"
end if
A[i] ← key
while i > 1 and A[PARENT(i)] < A[i]
      do exchange A[i] ↔ A[PARENT(i)]
         i ← PARENT(i)
```






```
INSERT(A, key)
A.heap-size ← A.heap-size + 1
A[A.heap-size] ← -∞
INCREASE-KEY(A, A.heap-size, key)
```

VRIJEME IZVRŠAVANJA:

MAXIMUM	$O(1)$
EXTRACT-MAX	$O(\lg n)$
INCREASE-KEY	$O(\lg n)$
INSERT	$O(\lg n)$

Primjeri: visualgo.net

Zadatak 4. Neka je A binarna hrpa maksimalnog usmjerenja koja koja je inicijalno prazna.

a) Učinite sljedeće operacije nad binarnom hrpom A.

1. INSERT (A,20), INSERT(A,26), INSERT (A,13), INSERT (A,5), INSERT(9
   INSERT (A,4), INSERT (A,2), INSERT (A,17), INSERT (A,3)
2. EXTRACT MAX (A), EXTRACT MAX (A)
3. INCREASE KEY (A 4, 11), INCREASE KEY (A, 1, 31)
4. EXTRACT MAX (A)

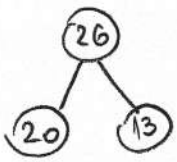
b) Objasnite kako možemo sortirati u $O(n \lg n)$ vremenu koristeći operacije na binarnoj hrpi.

Rješenje:

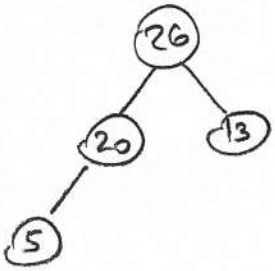
INSERT (A,20)

INSERT (A,26)

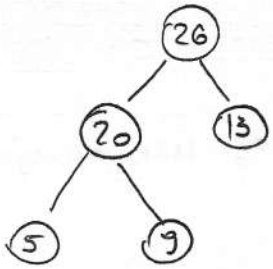
INSERT (A,13)



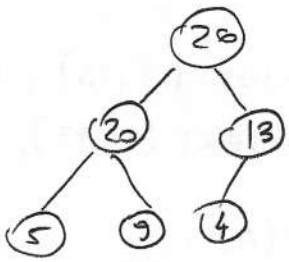
INSERT (A, 5)



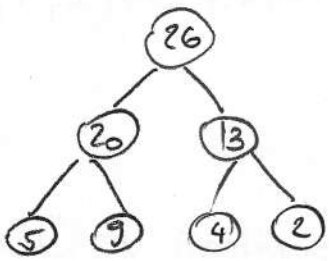
NSERT (A, 9)



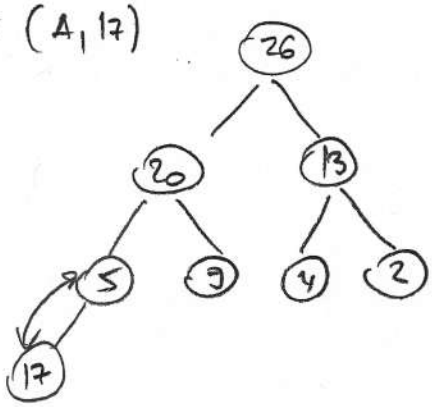
NSERT (A, 4)

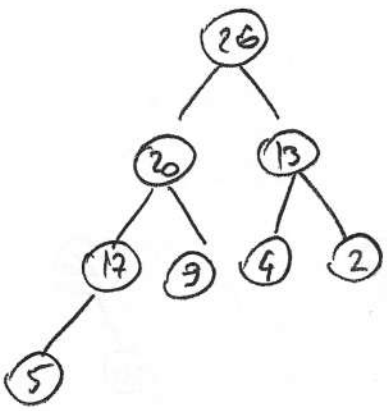


SERT (A, 2)

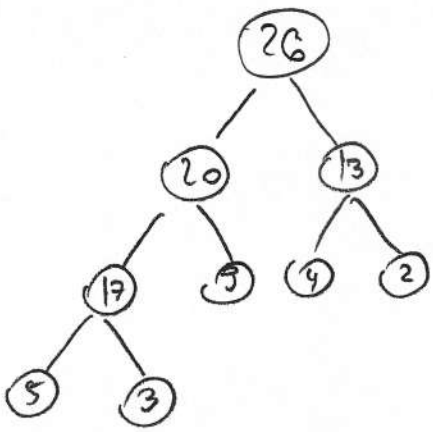


SERT (A, 17)

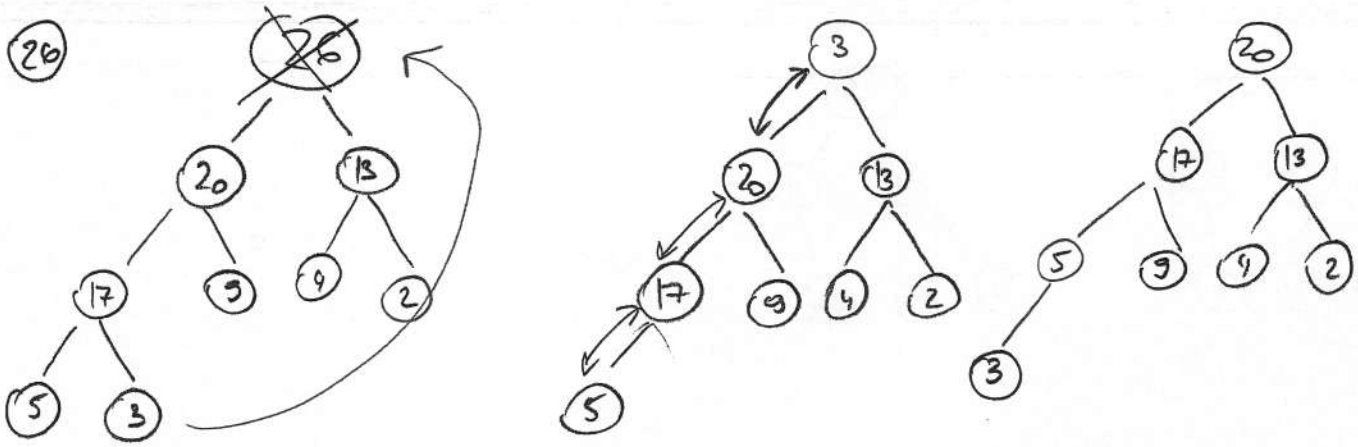




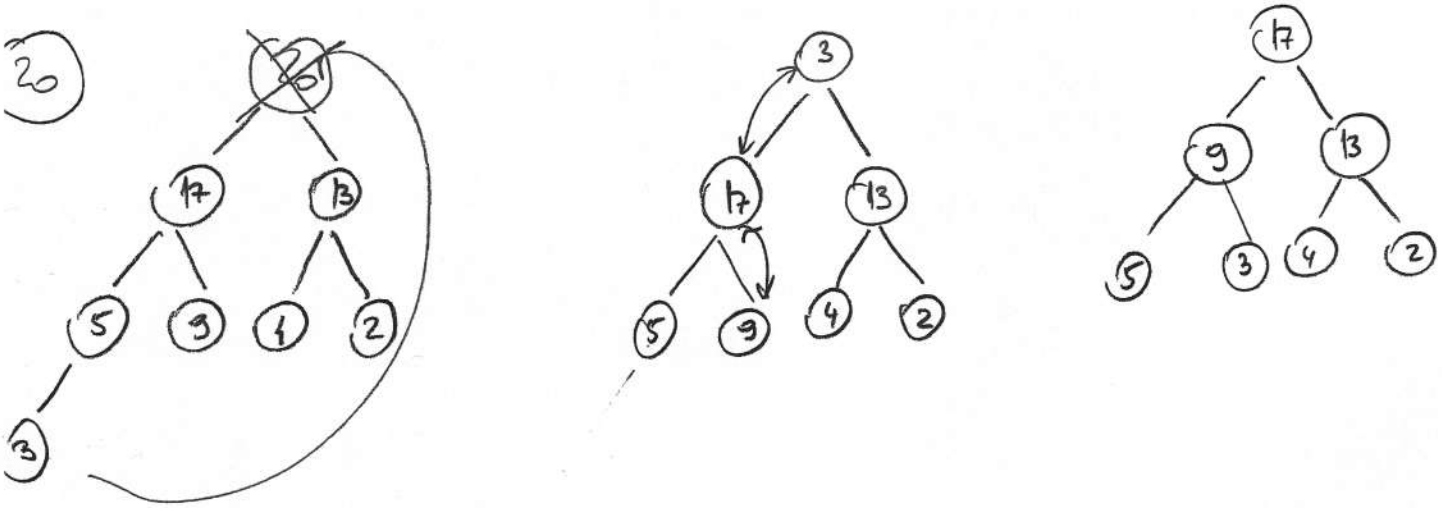
NSERT (A, 3)



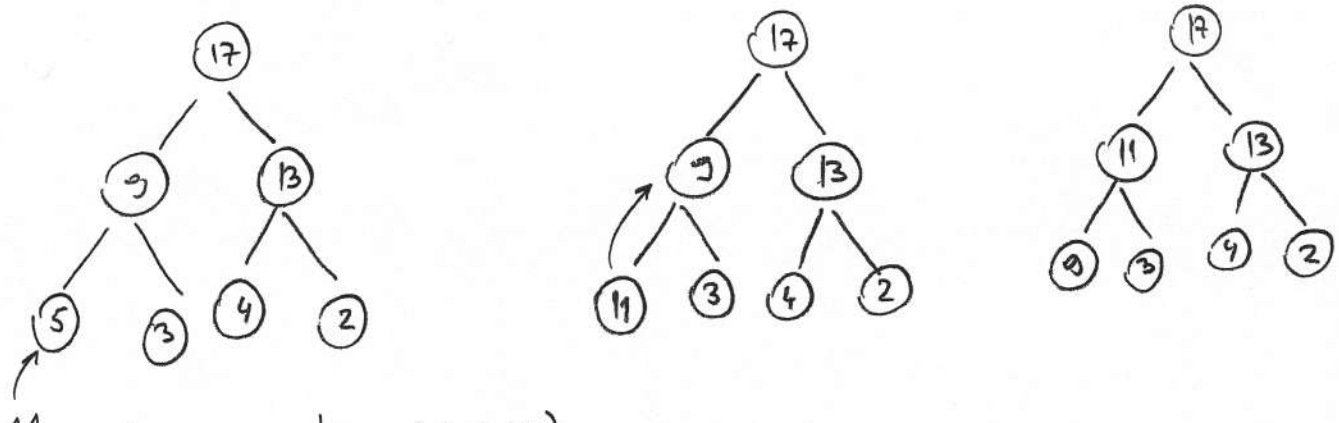
2. EXTRACT-MAX (A)



EXTRACT MAX (A)



3. INCREASE-KEY (A, 4, 11)



INCREASE-KEY (A, 1, 31)

EXTRACT-MAX (A)

(3)