Twaalfde college algoritmiek

8 mei 2015

Heap, heapsort

Deze techniek lost een probleem op door transformatie naar een

- eenvoudiger/geschiktere instantie van hetzelfde probleem
- andere representatie van dezelfde instantie
- ander probleem waarvoor al een algoritme bestaat

Voorbeelden:

- presorting
- heapsort
- kleinste gemeenschappelijke veelvoud

Gegeven een array $A = A[0], A[1], \ldots, A[n-1]$ met gehele getallen (bijvoorbeeld).
Vraag: zijn alle waarden uit A verschillend?

Het voor de hand liggende brute force algoritme vergelijkt alle paren getallen $(A[i], A[j])$ met $i < j$ en is dus $O(n^2)$.

Het gaat beter als we eerst voorsorteren met een snel (d.w.z. $O(n \lg n)$) sorteeralgoritme zoals Mergesort. Vervolgens hoeven we alleen nog buurelementen te vergelijken:

```
for i := 0 to n-2 do
   if ( A[i] = A[i+1] )
        return false;
return true;
```

Dit algoritme is $O(n \lg n) + O(n) \in O(n \lg n)$!

Gegeven een array $A = A[0], A[1], \ldots, A[n-1]$ met gehele getallen (bijvoorbeeld) en een geheel getal K.
Probleem: vind de waarde K in A.

Lineair zoeken: $O(n)$.

Met presorting:
- eerst voorsorteren met een snel sorteeralgoritme zoals Mergesort: $O(n \lg n)$
- vervolgens binair zoeken: $O(\lg n)$

Samen: $O(n \lg n)$

Heeft voorsorteren hier wel zin?

De kleinste gemeenschappelijke veelvoud van twee gehele getallen $m > 0$ en $n > 0$, genoteerd als $\texttt{kgv}(m, n)$, is het kleinste gehele getal dat deelbaar is door zowel m als n.
Voorbeelden: $\texttt{kgv}(24, 60) = 120; \texttt{kgv}(77, 11) = 77;$
$\texttt{kgv}(13, 15) = 195\ (= 13 * 15)$

We kunnen dit probleem oplossen door de priemontbinding van m en n te bekijken: $24 = 2^3 * 3; 60 = 2^2 * 3 * 5$,
dus $\texttt{kgv}(m, n) = 2^3 * 3 * 5 = 120$. Lastig en niet efficiënt.

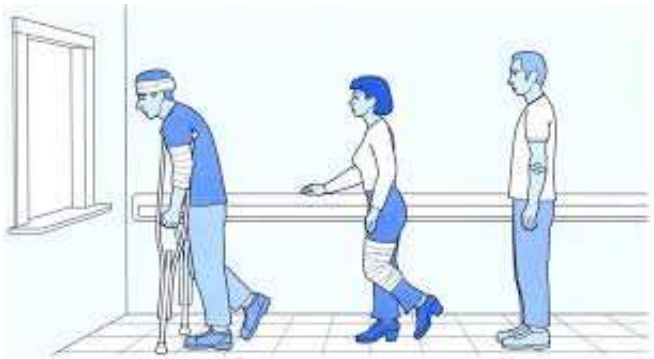
Beter: reduceer het probleem tot het vinden van $\texttt{ggd}(m, n)$ en los dit op met het algoritme van Euclides. Er geldt namelijk:

$$\texttt{kgv}(m, n) = \frac{m * n}{\texttt{ggd}(m, n)}$$

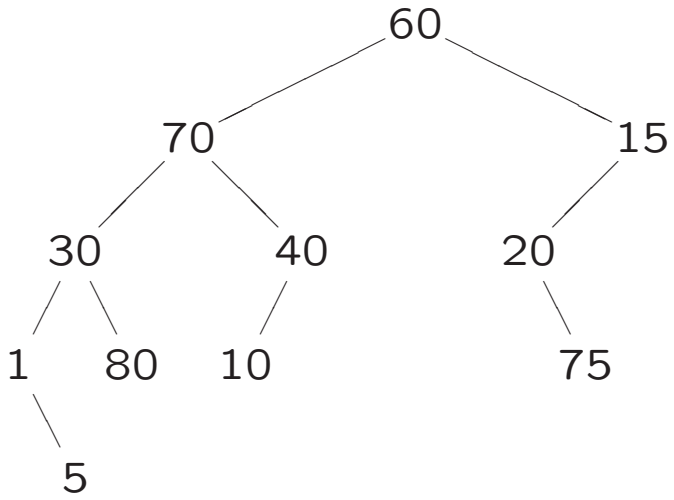
Een heap is een efficiënte manier om een priority queue te implementeren.

Priority queue: datatype dat de volgende operaties ondersteunt:

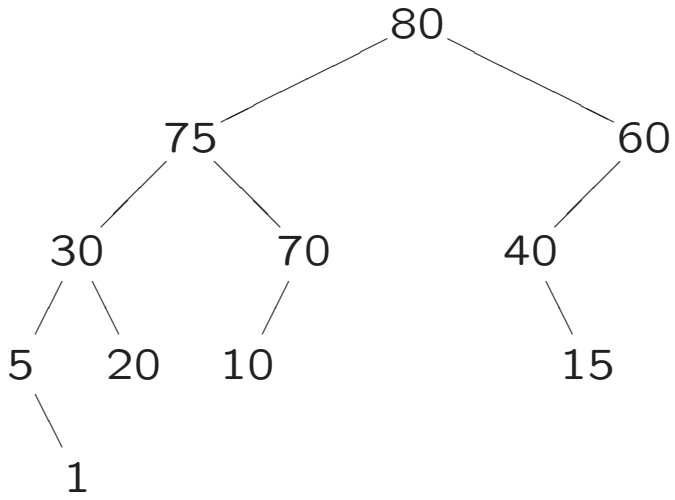
- voeg een element met bijbehorende prioriteit toe
- vind het element met de hoogste prioriteit / waarde
- verwijder het element met de hoogste prioriteit / waarde



Toepassingen:
- algoritme van Dijkstra
- Branch & bound
- ...



binaire boom



binaire boom met *heapeigenschap*

```
class knoop {  // een struct mag ook
  public:
    knoop ( ) {  // constructor
      info = 0; links = NULL; rechts = NULL; }
    int info;
    knoop* links;
    knoop* rechts;
};  // knoop
```

De binaire boom wordt gerepresenteerd door middel van een pointer naar de wortel:

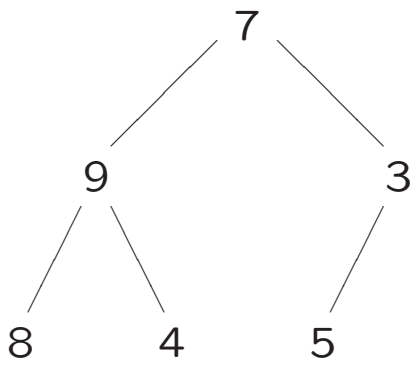
```
  knoop* wortel; // de ingang tot de binaire boom
```

Netter om een klasse te gebruiken: zie Programmeermethoden

Complete binaire boom

- Een complete binaire boom is een binaire boom waarbij alle niveaus geheel vol zitten, behalve eventueel het onderste. Op het onderste niveau mogen alleen de meest rechter knopen missen.

- Voorbeeld:



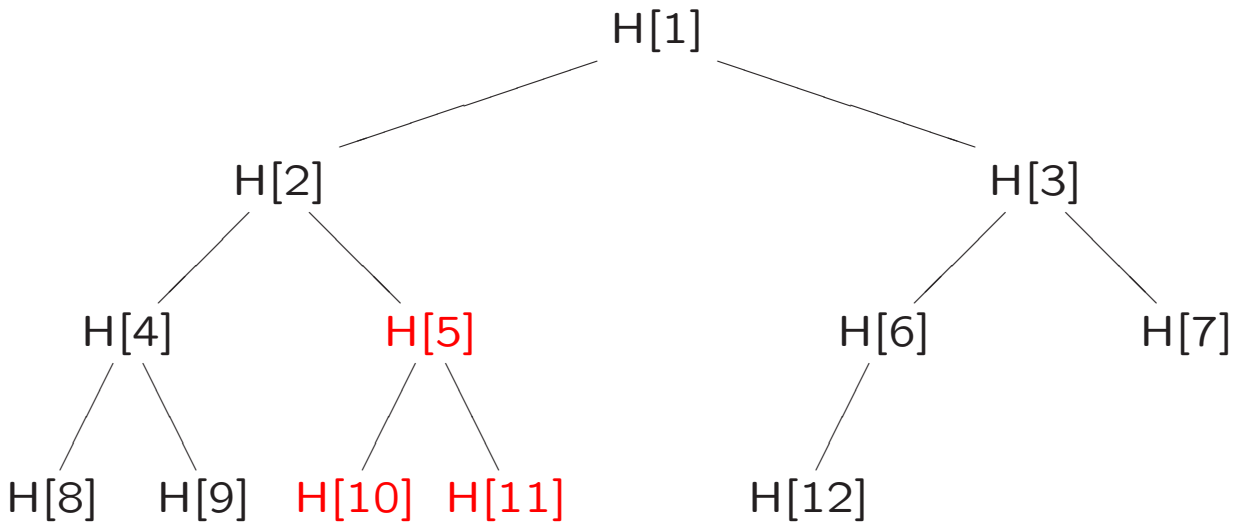
$\Longleftrightarrow$

7 9 3 8 4 5

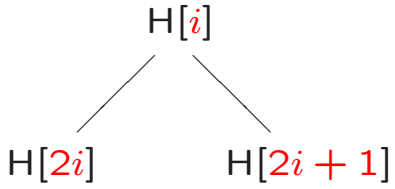
representatie als array

complete binaire boom

Boom $\leftrightarrow$ array



$\Updownarrow$ niveau-orde



H[1] H[2] H[3] H[4] H[5] H[6] H[7] H[8] H[9] H[10] H[11] H[12]

Een heap (hoopstructuur) is een binaire boom met de volgende twee eigenschappen:

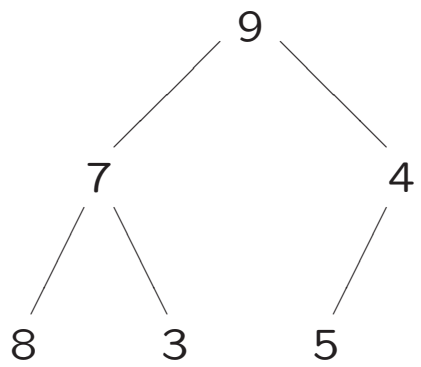
1. Vorm: de binaire boom is compleet

2. Inhoud: de heap-eigenschap geldt, d.w.z. in elke knoop geldt dat de waarde opgeslagen in die knoop groter dan of gelijk is aan* de waarde in zijn kinderen

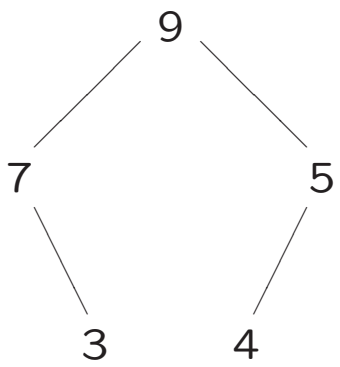
Langs elk pad van de wortel tot een blad zijn de sleutels in de knopen dus van groot naar klein geordend.

De heap is een efficiënte implementatie van een priority queue, en ook de basis van de sorteermethode heapsort.

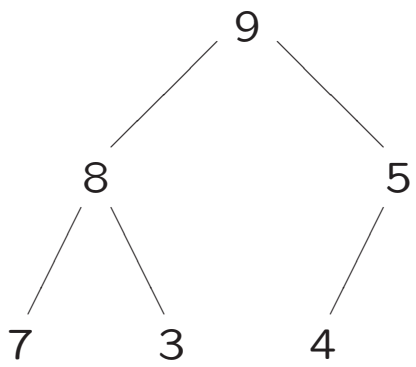
*we spreken dan wel van een max-heap; een min-heap wordt analoog gedefinieerd: waarde knoop $\leq$ waarde kinderen



1. geen heap



2. geen heap



3. wel heap

1. ouder $\geq$ kinderen geldt niet in elke knoop

2. niet compleet

3. compleet en ouder $\geq$ kinderen

1. Gegeven n, dan bestaat er precies één **complete** binaire boom met n knopen. Deze heeft hoogte $\lfloor \lg n \rfloor$.

2. De wortel van een heap bevat altijd de grootste waarde.

3. Voor elke knoop van een heap geldt: de subboom met die knoop als wortel is weer een heap.

4. Een heap wordt gerepresenteerd door een eendimensionaal array H, met de inhoud van de n knopen op posities 1 t/m n.

   - ouderknopen (interne knopen) corresponderen met de posities 1 t/m $\lfloor \frac{n}{2} \rfloor$; bladeren met $\lfloor \frac{n}{2} \rfloor + 1$ t/m n.

   - de kinderen van $H[i]$ $(i = 1, \ldots, \lfloor \frac{n}{2} \rfloor)$ zijn $H[2i]$ en $H[2i+1]$; de ouder van $H[i]$ $(i = 2, \ldots, n)$ is $H[\ \lfloor i/2 \rfloor\ ]$.

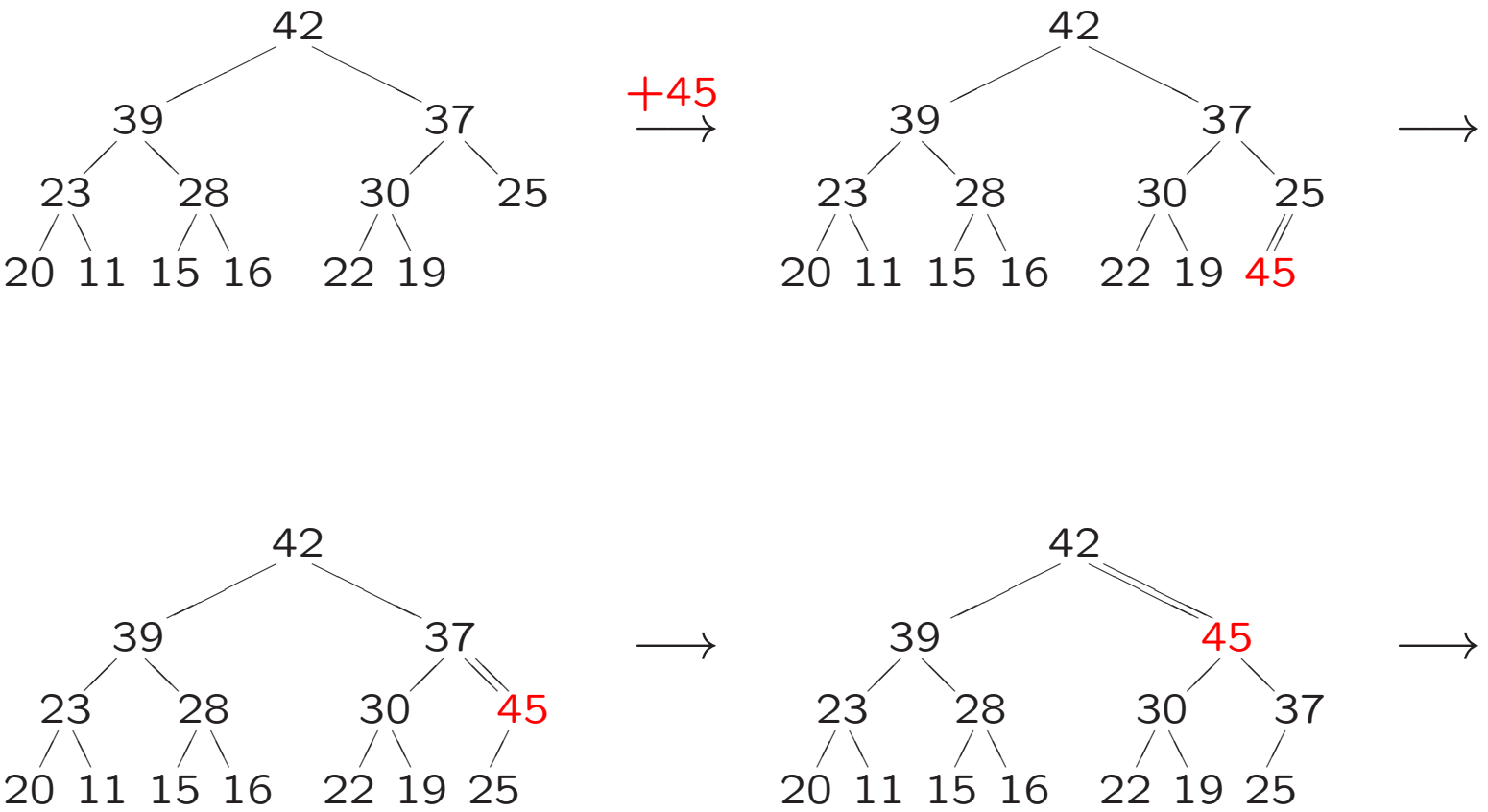
Wanneer de waarde in een knoop verandert (of een waarde wordt verwijderd/toegevoegd) zal i.h.a. de heap-eigenschap niet meer gelden. Er zijn twee manieren (beide $O(\lg n)$, met n het aantal knopen van de heap) om die weer te herstellen, afhankelijk van de situatie.

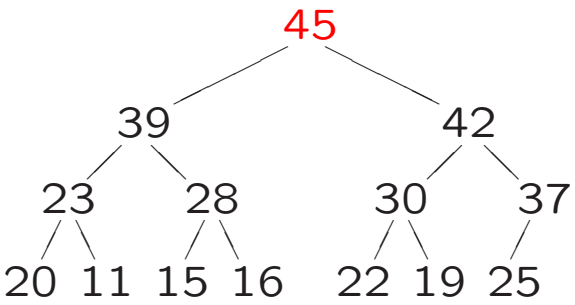
Stel nu dat de waarde in één knoop veranderd wordt. Dan zijn er twee mogelijkheden:

1. waarde in knoop $>$ waarde in ouder: herhaald verwisselen met ouder totdat de heap-eigenschap hersteld is (waarde borrelt omhoog)

2. waarde knoop $<$ waarde van (ten minste een der) kinderen: herhaald verwisselen met grootste kind totdat de heap-eigenschap hersteld is (waarde zakt omlaag)

Toevoegen -1-


42
39
37
23
28
30
25
20 11 15 16
22 19
+45
42
39
37
23
28
30
25
20 11 15 16
22 19 45
42
39
37
23
28
30
45
20 11 15 16
22 19 25
42
39
45
23
28
30
37
20 11 15 16
22 19 25



Heap-eigenschap weer hersteld:
45 is omhoog geborreld

42 39 37 23 28 30 25 20 11 15 16 22 19

⇓

42 39 37 23 28 30 25 20 11 15 16 22 19 45

⇓

42 39 37 23 28 30 45 20 11 15 16 22 19 25

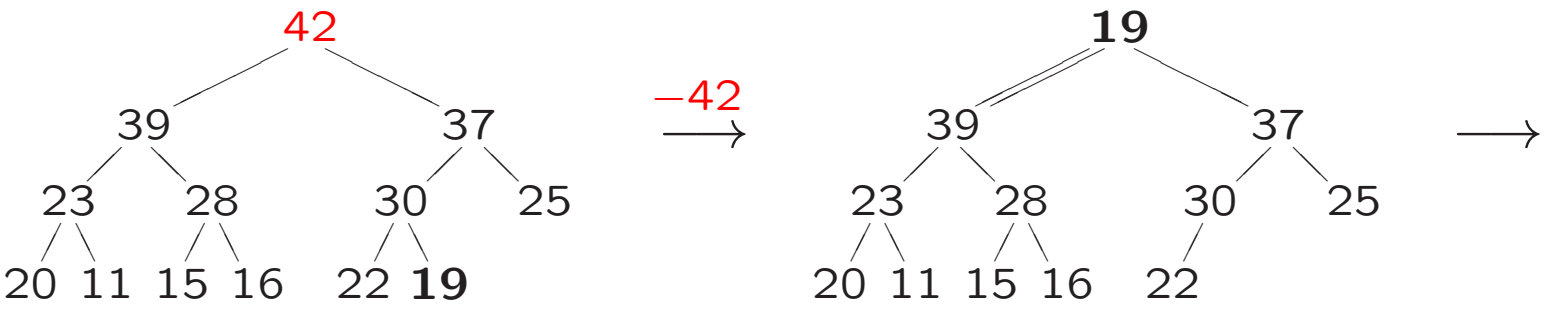
⇓

42 39 45 23 28 30 37 20 11 15 16 22 19 25

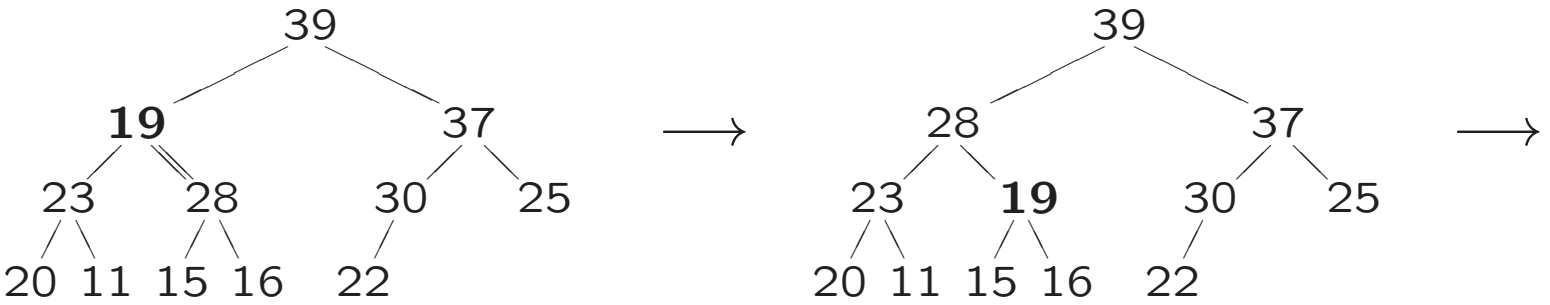
⇓

45 39 42 23 28 30 37 20 11 15 16 22 19 25

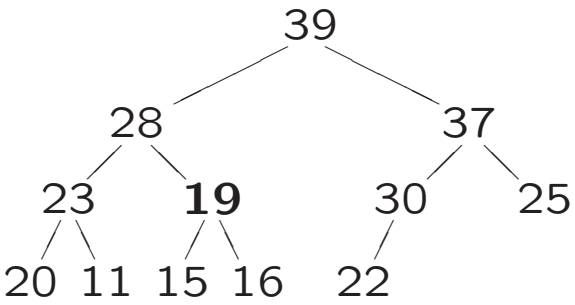
Verwijderen maximum -1-



links heap, rechts heap



Verwijderen maximum -2-



Heap-eigenschap weer hersteld:
19 is omlaag gezakt

42 39 37 23 28 30 25 20 11 15 16 22 **19**

⇓

19 39 37 23 28 30 25 20 11 15 16 22

⇓

39 **19** 37 23 28 30 25 20 11 15 16 22

⇓

39 28 37 23 **19** 30 25 20 11 15 16 22

⇓

39 28 37 23 **19** 30 25 20 11 15 16 22

Operaties O(lg n)

1. Zoeken grootste waarde:
   - zit in de wortel
   - complexiteit O(1)

2. Toevoegen nieuwe waarde:
   - achteraan toevoegen
   - herhaald verwisselen met ouder
   - complexiteit O(h)*, dus O(lg n)

3. Verwijderen grootste waarde (uit de wortel):
   - achterste waarde $\longrightarrow$ wortel
   - herhaald verwisselen met grootste kind
   - complexiteit O(h), dus O(lg n)

*h is de hoogte van de boom

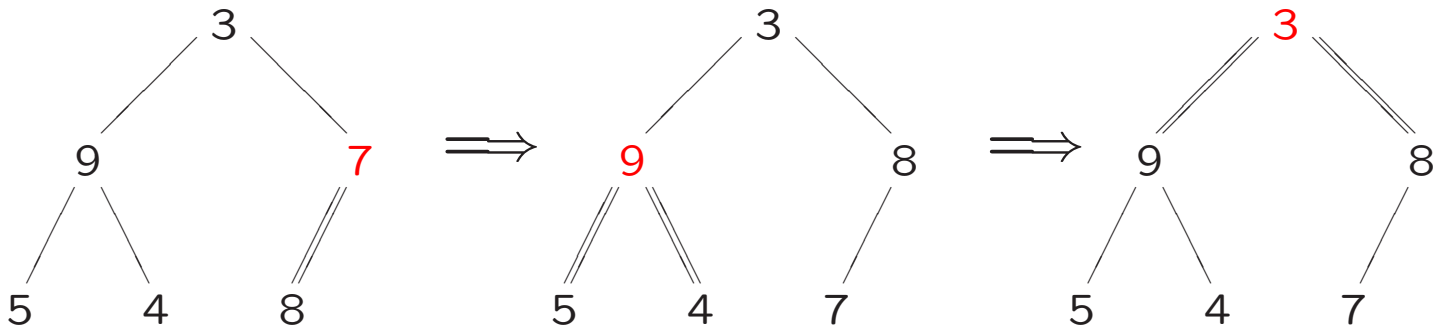
Constructie van een heap uit een gegeven rij (array H) sleutels (getallen bijv.):

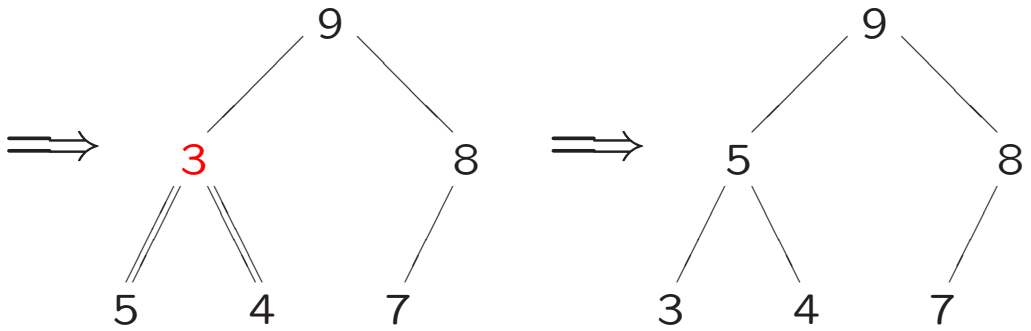
- Bottom up (heapify): beginnend bij $H[\lfloor \frac{n}{2} \rfloor]$, de laatste ouderknoop, en zo teruglopend, wordt in alle subbomen met de ouderknopen als wortel de heap-eigenschap hersteld via omlaag zakken van de (inhoud van die) ouderknoop

- Top down: beginnend bij de wortel en door telkens een knoop meer bij de heap te betrekken wordt de heap-eigenschap steeds hersteld via omhoog borrelen van de (inhoud van de) nieuwe knoop

- Heapify is $O(n)$; de andere methode is $O(n \lg n)$

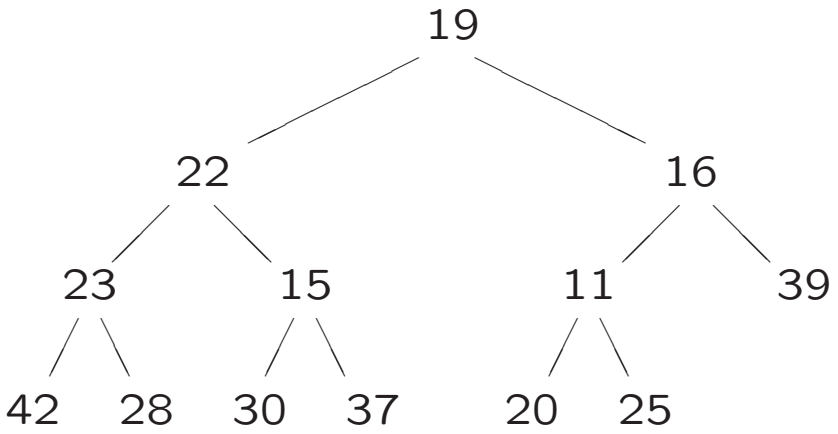
Heapify





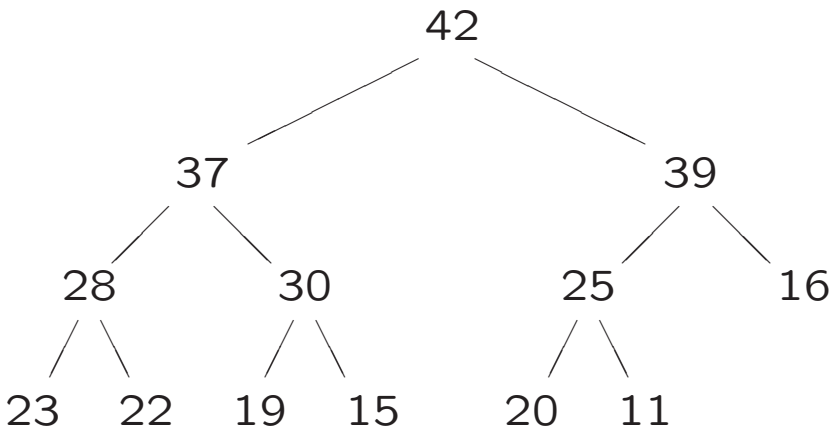
Heap!

Groter voorbeeld



heapify

$\longrightarrow \cdots \longrightarrow$



```
for i := ⌊ n/2 ⌋ downto 1 do
        k := i; v := H[k];
        heap := false;
        while not heap and 2 * k ≤ n do // geen blad
            j := 2 * k; // linkerkind
            if j < n then // 2 kinderen
                if H[j] < H[j + 1] then
                    j := j + 1; fi // selecteer grootste kind
            fi
            if  v ≥ H[j] then
                heap := true;
            else
                H[k] := H[j]; k := j; fi
        od
        H[k] := v;
od
```

1. Maak een heap van het gegeven array

2. Verwijder nu herhaald ($n-1$ keer) de grootste waarde uit de wortel:

   - verwissel deze met de laatste waarde uit de heap
   - verlaag de grootte van de heap met 1
   - zorg dat overal de heap-eigenschap weer geldt door de nieuwe waarde uit de wortel te laten zakken

Het array wordt zo oplopend gesorteerd.
Complexiteit: $O(n \lg n)$.

Sorteer het array 3 9 7 5 4 8 met heapsort.

Fase 1

3 9 7 5 4 8 $\longrightarrow$
3 9 8 5 4 7 $\longrightarrow$
9 3 8 5 4 7 $\longrightarrow$
9 5 8 3 4 7

Fase 2

9 5 8 3 4 7 $\longrightarrow$
7 5 8 3 4 |9 $\longrightarrow$
8 5 7 3 4 |9 $\longrightarrow$
4 5 7 3 |8 |9 $\longrightarrow$
7 5 4 3 |8 |9 $\longrightarrow$
3 5 4 |7 |8 |9 $\longrightarrow$
5 3 4 |7 |8 |9 $\longrightarrow$
4 3 |5 |7 |8 |9 $\longrightarrow$
3 |4 |5 |7 |8 |9

Heapsort gezien als een voorbeeld van Transform-and-Conquer (H6, Levitin): transformeer het array eerst naar een heap, en doe dan iedere keer de volgende stappen:

- Verwissel eerste en laatste getal (transformatie)
- Haal laatste getal uit de heap (conquer)
- Herstel de heap (transformatie)

Overigens ... als je de heap niet ziet in het array, is het algoritme moeilijk te begrijpen.

- **Lezen/leren bij dit college:** Paragraaf 6.4
- **Laatste college:**
  vrijdag 22 mei 2015, 11.15-13.00, zaal B2 (oud tentamen)
- **Laatste werkcollege:**
  donderdag 21 mei 2015, 13:45–15:30 uur, zaal B2
- **Opgaven/oude tentamens/testvoorbeelden:**
  zie http://liacs.leidenuniv.nl/~graafjmde/ALGO/
- **Inleveren programmeeropdracht 3:** 21 mei 2015
- **Tentamen:**
  dinsdag 9 juni 2015, 14:00–17:00 uur
- **Vragenuur:**
  vrijdag 5 juni 2015, 14:00 uur ????

De rij edelstenen: 65 50 72 52 45 52 48 59, met totale waarde 443

Het array M met $M[i][j]$ = maximale waarde die Bonnie kan krijgen als edelstenen nummer i t/m j in die volgorde op de grond liggen:

0	65	65	137	117	182	165	230
0	0	72	72	117	117	165	165
0	0	0	72	52	124	104	176
0	0	0	0	52	52	104	104
0	0	0	0	0	52	52	111
0	0	0	0	0	0	52	52
0	0	0	0	0	0	0	59
0	0	0	0	0	0	0	0

De rij edelstenen: 65 50 72 52 45 52 48 59, met totale waarde 443

Het bijbehorende array met de optimale keuzes (hoeven niet uniek te zijn):

65	65	72	65	65	65	65	65
0	50	72	50	45	52	48	59
0	0	72	72	72	72	72	72
0	0	0	52	52	52	52	59
0	0	0	0	45	52	45	59
0	0	0	0	0	52	52	59
0	0	0	0	0	0	48	59
0	0	0	0	0	0	0	59

Bonnie krijgt: 65 48 45 72 (totaal: 230)
Clyde krijgt: 59 52 52 50 (totaal: 213)