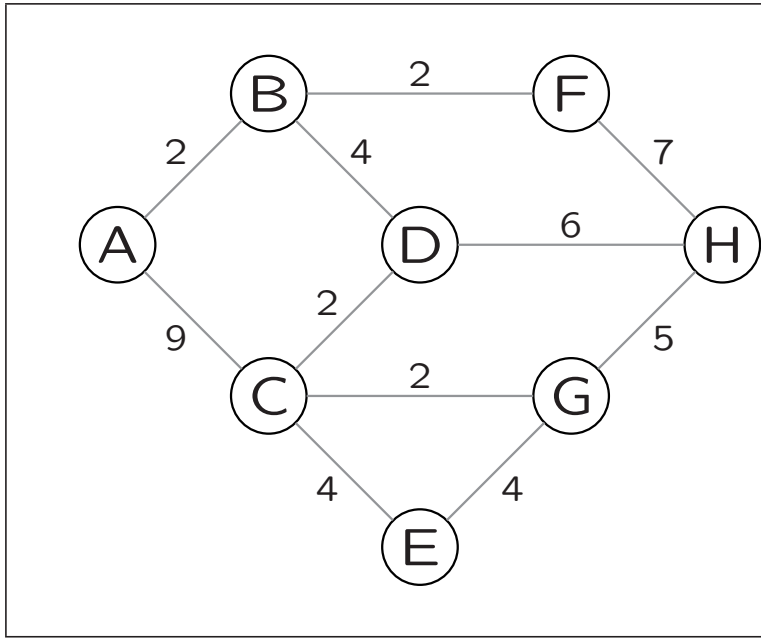
Twaalfde college algoritmiek

19 mei 2016

Heap, Heapsort & Heapify

Als je gewoon wilt doortekenen in één plaatje. . . (alternatief)

A	B	C	D	E	F	G	H	Actie
0	∞	∞	∞	∞	∞	∞	∞	Begin met A
–	2	9	∞	∞	∞	∞	∞	Kies B, vanaf A
–	–	9	6	∞	4	∞	∞	Kies F, vanaf B
–	–	9	6	∞	–	∞	11	Kies D, vanaf B
–	–	8	–	∞	–	∞	11	Kies C, vanaf D
–	–	–	–	12	–	10	11	Kies G, vanaf C
–	–	–	–	12	–	–	11	Kies H, vanaf F
–	–	–	–	12	–	–	–	Kies E, vanaf C



Een rij in de tabel komt overeen met het array `pad` in pseudo-code op volgende slide.

// invoer: samenhangende gewogen graaf $G = (V, E)$ en startknoop s
// uitvoer: array dat de lengtes van de kortste paden vanuit s bevat;
// na afloop is pad[v] = de lengte van een kortste pad van s naar v

for $v \in V$ **do**
    pad[v] := ∞;
od
pad[s] := 0;
$U := \emptyset$;

while ($U \neq V$) **do**
    vind knoop $v^* \in V \setminus U$ met pad[v^*] minimaal;
    $U := U \cup \{v^*\}$;
    for alle knopen v aangrenzend aan v^* **do**
        if pad[v^*] + gewicht(v^*, v) < pad[v] **then**
            pad[v] := pad[v^*] + gewicht(v^*, v);
        fi
    od
od

Complexiteit. . .

De volgorde waarin de knopen (deeloplossingen) worden uitgebreid hangt direct af van de berekende grenzen:

- er worden meerdere deeloplossingen tegelijk bijgehouden, dit in tegenstelling tot backtracking
- in elke stap wordt een van al deze deeloplossingen gekozen, en daarvan worden alle 1-staps-uitbreidingen (kinderen in de state space tree) bekeken en geëvalueerd (d.w.z. ondergrens/bovengrens bepaald)
- zinloze uitbreidingen worden meteen verworpen
- meestal wordt de deeloplossing gekozen die het meest veelbelovend lijkt: best-fit-first branch-and-bound
- bij minimalisatieproblemen (resp. maximalisatieproblemen) kiezen we de knoop met de laagste ondergrens (resp. hoogste bovengrens) als eerste

Bij Dijkstra: kies knoop buiten boom met laagste kandidaatwaarde

Bij Branch & Bound: kies deeloplossing met beste ondergrens/bovengrens

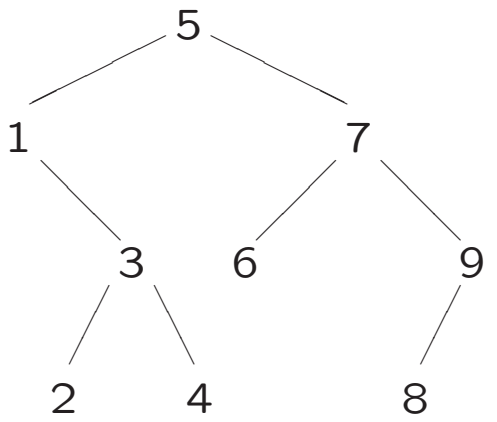
Priority Queue:

- objecten met prioriteit (en info)
- insert
- findmax (findmin)
- deletemax (deletemin)
- (optioneel) changepriority

Binaire zoekboom:

LWR

1 2 3 4 5 6 7 8 9



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

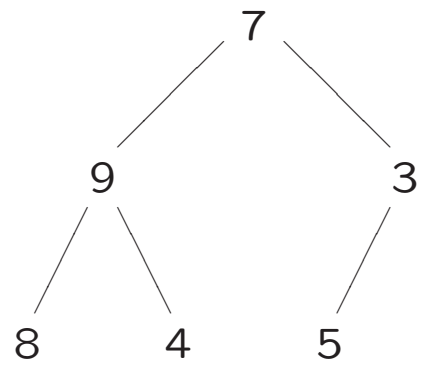
De binaire boom wordt gerepresenteerd door middel van een pointer naar de wortel:

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

Netter om een klasse te gebruiken: zie Programmeermethoden

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

- Voorbeeld:

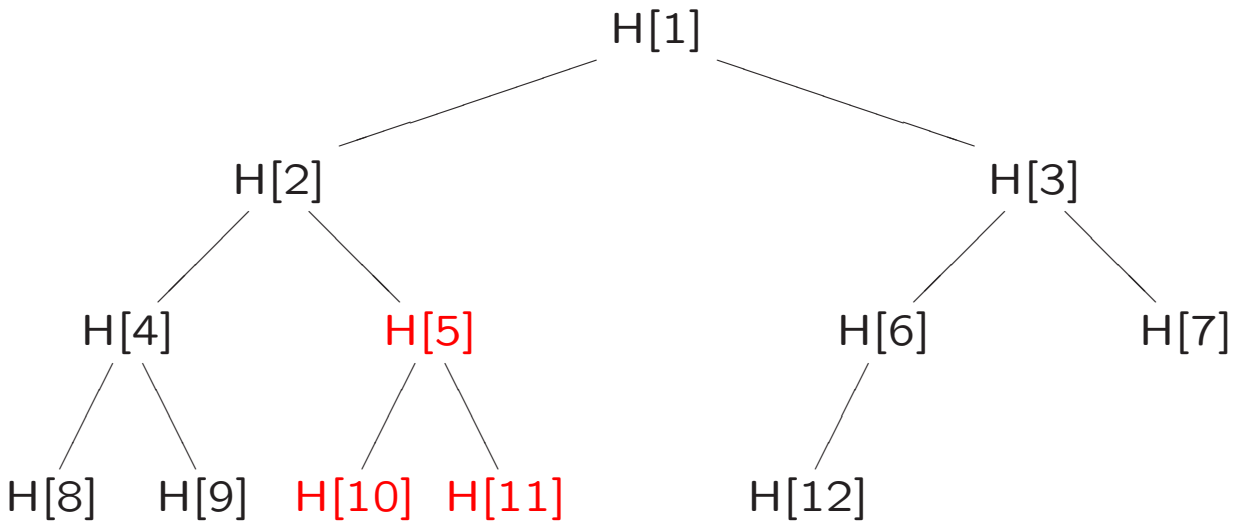


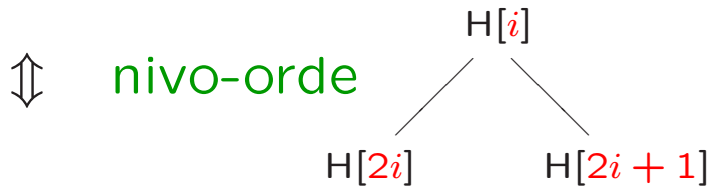
complete binaire boom

$\Longleftrightarrow$

7 9 3 8 4 5

representatie als array





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

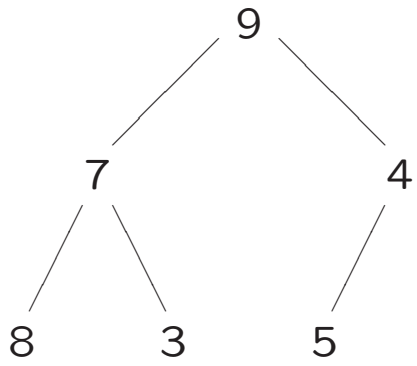
Definitie

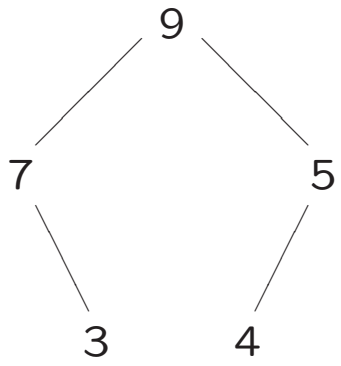
Een heap (hoopstructuur) is een binaire boom met de volgende twee eigenschappen:

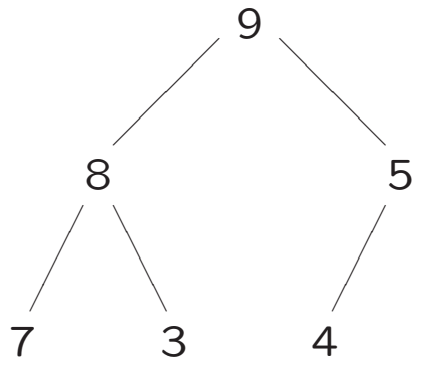
1. Vorm: de binaire boom is compleet
2. Inhoud: de heap-eigenschap geldt, d.w.z. in elke knoop geldt dat de waarde opgeslagen in die knoop groter dan of gelijk is aan(*) de waarde in zijn kinderen

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

(*) we spreken dan wel van een max-heap; een min-heap wordt analoog gedefinieerd







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.

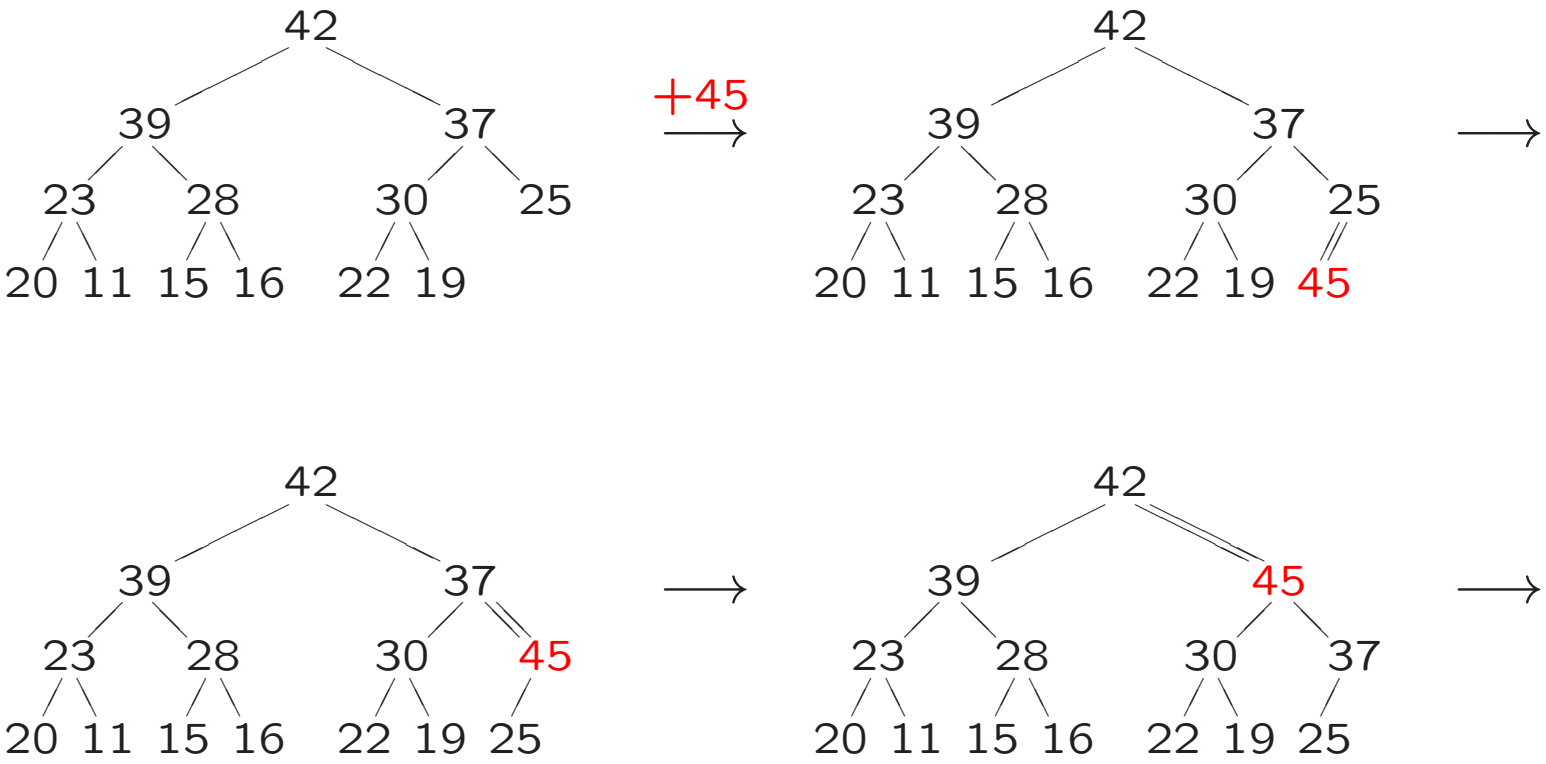
   - de kinderen van $H[i]$ $(i = 1, \ldots, \lfloor \frac{n}{2} \rfloor)$ zijn $H[2i]$ en $H[2i+1]$; de ouder van $H[i]$ $(i = 2, \ldots, n)$ is $H[\ \lfloor i/2 \rfloor\ ]$.

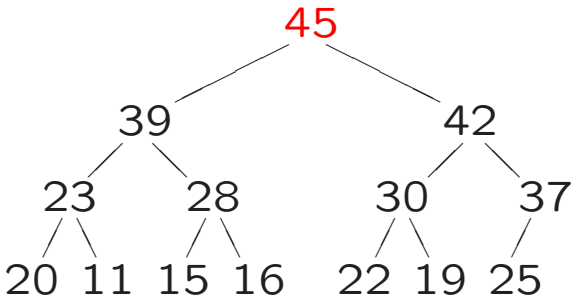
Wanneer de waarde in een knoop verandert (of een waarde wordt verwijderd/toegevoegd) zal i.h.a. de heap-eigenschap niet meer gelden. Er zijn twee manieren (beide $O(\lg n)$, met n het aantal knopen van de heap) om die weer te herstellen, afhankelijk van de situatie.

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)


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

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

⇓

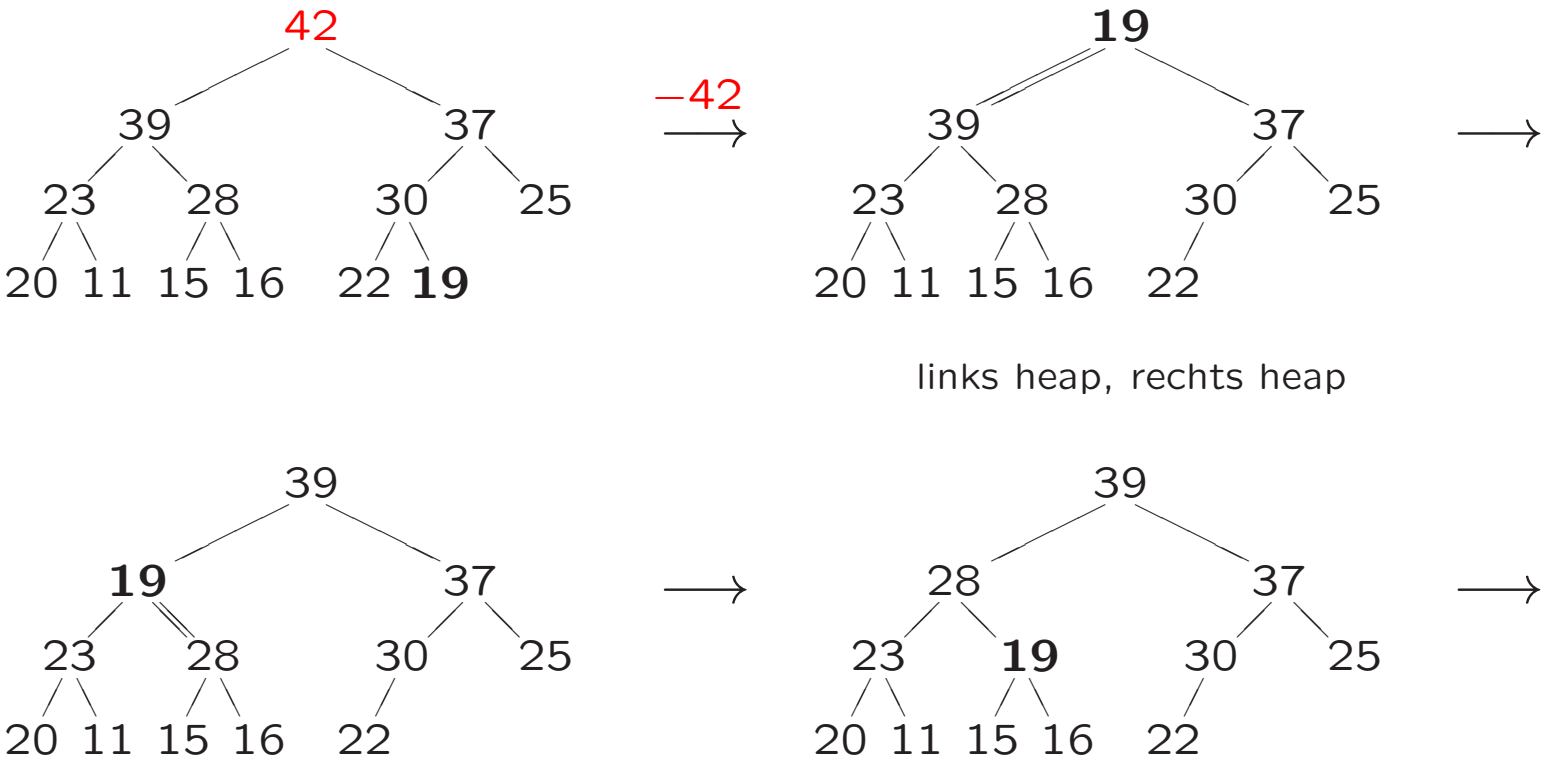
42 39 37 23 28 30 45 20 11 15 16 22 19 25

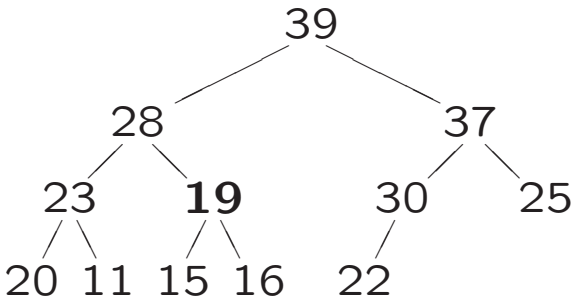
⇓

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


42
39
37
23
28
30
25
20 11 15 16 22 19
−42
19
39
37
23
28
30
25
20 11 15 16 22
links heap, rechts heap
39
19
37
23
28
30
25
20 11 15 16 22
39
28
37
23
19
30
25
20 11 15 16 22



Heap-eigenschap weer hersteld

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

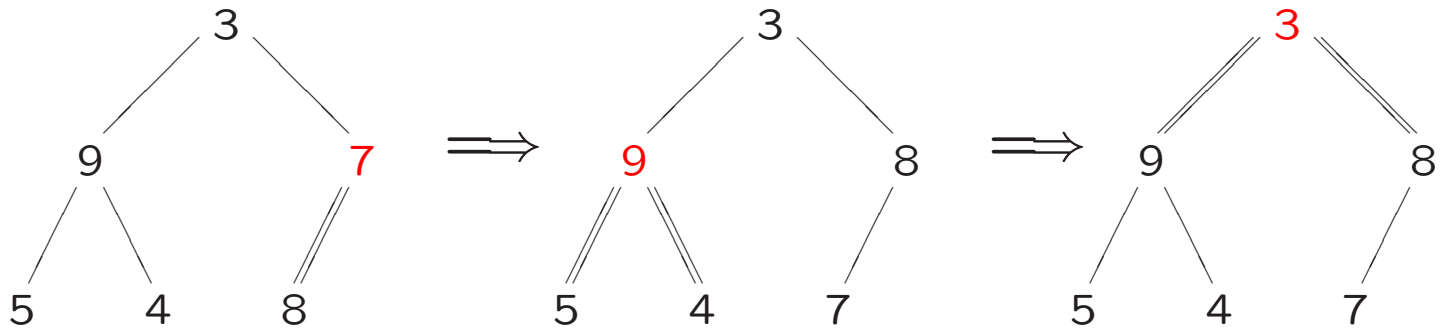
⇓

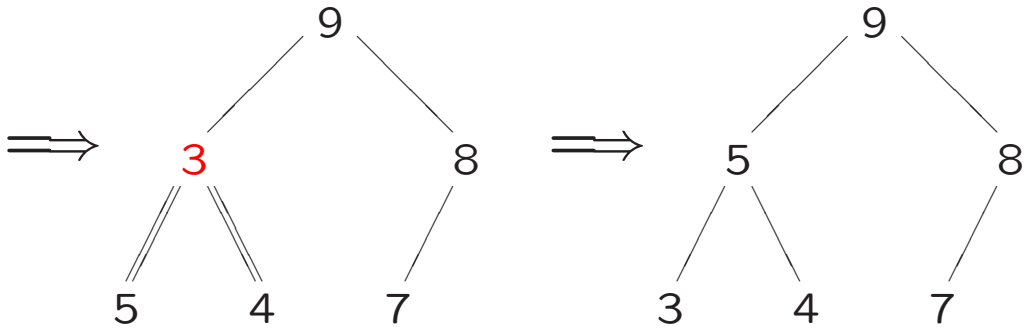
39 28 37 23 **19** 30 25 20 11 15 16 22

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

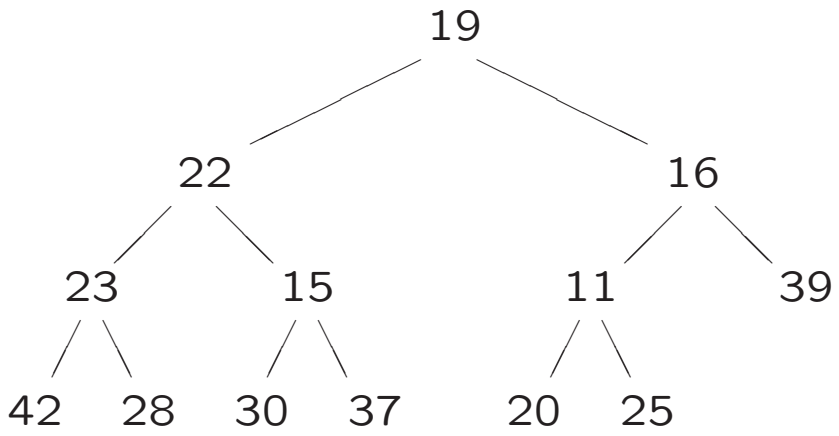
- Bottom up (heapify): beginnend bij $H[\lfloor \frac{n}{2} \rfloor]$, de laatste ouderknoop, en zo teruglopend, wordt in alle subbomen met de ouderknopen als wortel de heap-eigenschap hersteld via omlaag zakken van de (inhoud van die) ouder-knoop

- 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



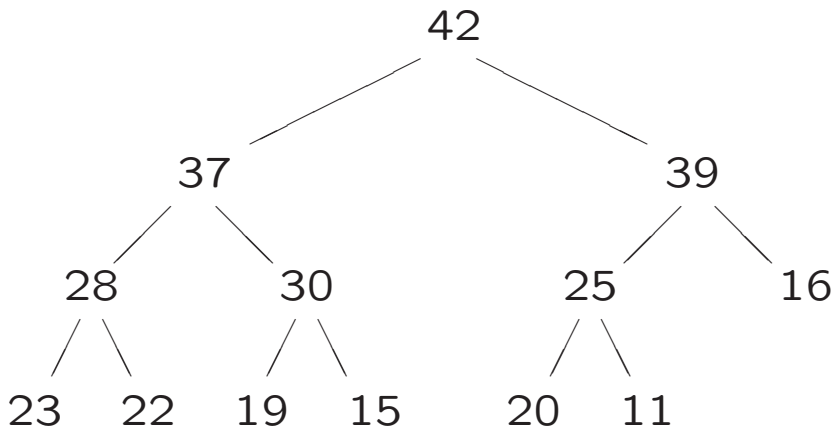


Heap!

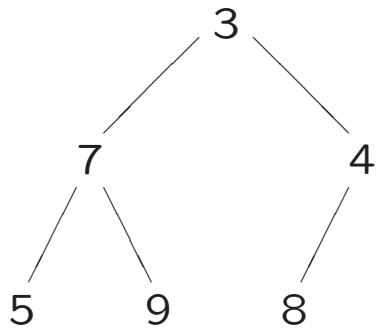


heapify

$\longrightarrow \cdots \longrightarrow$



```
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
```


3
7
4
5
9
8

- Naam bottom-up – top-down
- Worst case bottom-up (heapify). . .

  Worst case complexiteit. . .
- Worst case top-down. . .

  Worst case complexiteit. . .

- Naam bottom-up – top-down

- Worst case bottom-up (heapify)...

  Worst case complexiteit: $O(n)$

- Worst case top-down...

  Worst case complexiteit: $O(n \lg n)$

  `http://oeis.org/A036799`

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

- **Lezen/leren bij dit college:**
  paragraaf 6.4

- **Werkcollege:**
  donderdag 19 mei 2016, 13:45–15:30, in zaal Benoordenhout

- **Opgaven:**
  zie `http://www.liacs.leidenuniv.nl/~vlietrvan1/algoritmiek`

- **Volgend 'college':**
  26 mei 2016: tentamen oefenen

- **Tentamen:**
  dinsdag 7 juni 2016, 14.00–17.00

- **Vragenuur?**