# F10 - Prioritetskö, hög

## 5DV149 Datastrukturer och algoritmer
## Kapitel 14.5–14.8

Niclas Börlin
niclas.borlin@cs.umu.se

2020-02-20 Thu

# Innehåll

- Prioritetskö:
  - Modell.
  - Organisation.
  - Konstruktioner
    - Listor.
    - *Heap* (hög).

# Prioritetskö

- Modell: Patienterna på en akutmottagning kommer in i en viss tidsordning men behandlas utifrån en annan ordning.
- Organisation: En mängd vars grundmängd är linjärt ordnad av en *prioritetsordning*:
  - Avläsningar och borttagningar görs endast på det element som har högst prioritet.
  - Andra mängdoperationer är inte aktuella.

# Informell specifikation av prioritetskö

```
abstract datatype  Pqueue(val, R)
  Empty()                                  → Pqueue(val, R)
  Insert(v:val, p:Pqueue(val, R))          → Pqueue(val, R)
  Isempty(p:Pqueue(val, R))                → Bool
  Inspect-first(p:Pqueue(val, R))          → val
  Delete-first(p:Pqueue(val, R))           → Pqueue(val, R)
```

- R är relationen för prioritetsordningen.
  - Om t.ex. R är "$<$" så är "$a$ R $b$" sann om $a < b$.
- Ibland slås `Inspect-first` och `Delete-first` ihop.
- Gränsytan ovan förutsätter *statisk* prioritet.
- Vill man ha *dynamisk* prioritet måste en `Update`-metod finnas som byter ut relationen.

# Fråga

- Hur hanteras element med *samma* prioritet?

# Formell specifikation av prioritetskö

**OBS! Fel i boken!**

Ax 1 Isempty (Empty)
Ax 2 ¬Isempty (Insert (v, p))
Ax 3 Inspect-first (Insert (v, Empty)) = v
Ax 4 Inspect-first (Insert (v1, Insert (v2, p))) =
**if** v1 R v2
**then** Inspect-first (Insert (v1, p))
**else** Inspect-first (Insert (v2, p))
Ax 5 Delete-first (Insert (v, Empty)) = Empty
Ax 6 Delete-first (Insert (v1, Insert (v2, p))) =
**if** v1 R v2
**then** Insert (v2, Delete-first (Insert (v1, p)))
**else** Insert (v1, Delete-first (Insert (v2, p)))

# Formell specifikation av prioritetskö

**OBS! Fel i boken!**

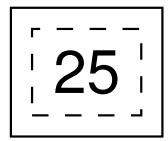
Ax 1 Isempty (Empty)
Ax 2 ¬Isempty (Insert (v, p))
Ax 3 Inspect-first (Insert (v, Empty)) = v
Ax 4 Inspect-first (Insert (v1, Insert (v2, p))) =
**if** v1 R v2
**then** Inspect-first (Insert (v1, p))
**else** Inspect-first (Insert (v2, p))
Ax 5 Delete-first (Insert (v, Empty)) = Empty
Ax 6 Delete-first (Insert (v1, Insert (v2, p))) =
**if** v1 R v2
**then** Insert (v2, Delete-first (Insert (v1, p)))
**else** Insert (v1, Delete-first (Insert (v2, p)))

- Frågor:
  - Om R är "$<$" och två lika värden stoppas in, vilket plockas ut först?
  - Dito om R är "$\leq$".

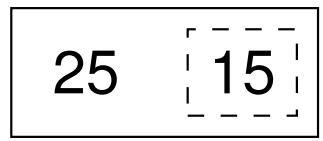
# Exempel (1)

- För val=heltal, R=$<$, dvs. “$a$ R $b$” är sann om $a < b$:
- `p` $\leftarrow$ `Empty()`

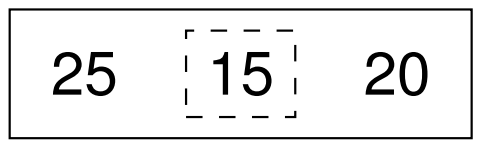
- `p` $\leftarrow$ `Insert(25,p)`



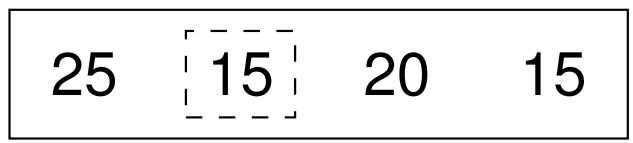

- `p` $\leftarrow$ `Insert(15,p)`




- `p` $\leftarrow$ `Insert(20,p)`

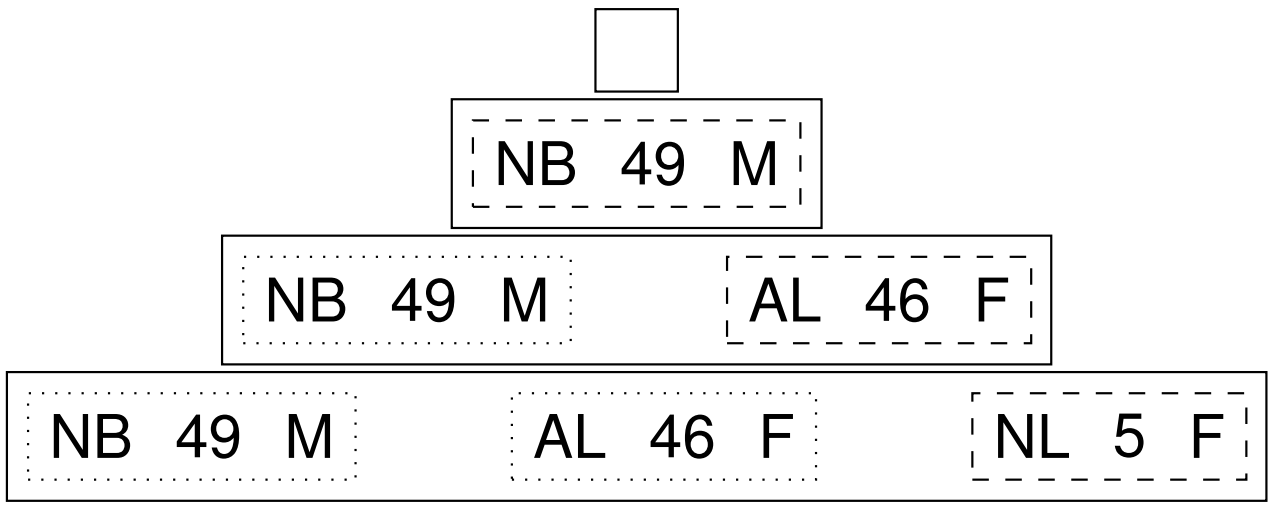



- `p` $\leftarrow$ `Insert(15,p)`



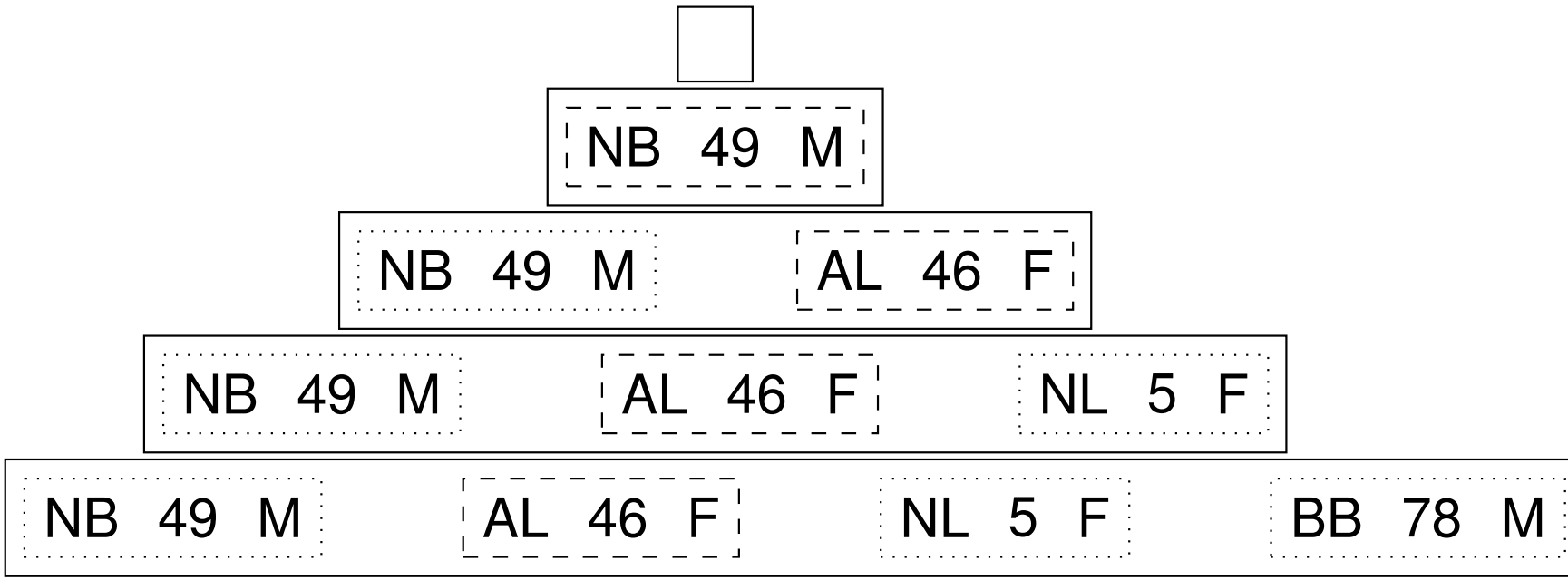
# Exempel (2)

- För val=3-tippel med (name, age, sex), R= `a.age < b.age`:



# Exempel (3)

- För val=3-tippel med (name, age, sex), R= `F R M` (kvinnor prioriteras före män):



# Stack och kö som specialfall

- Om R är en strikt partiell ordning, t.ex. $>$, kommer lika element behandlas som en *kö*.
- Om R är icke-strikt, t.ex. $\geq$, behandlas lika element som en *stack*.
- Om R är den *totala relationen*, dvs. sann för alla par av värden blir prioritetskön en *stack*.
- Om R är den *tomma relationen*, dvs. falsk för alla par av värden, blir prioritetskön en *kö*.

# Fråga

- Hur lagras elementen internt i prioritetskön?

# Konstruktioner av Prioritetskö

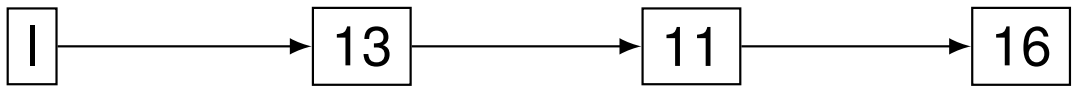
- Utgår ofta från konstruktioner av:
  - Mängd
  - Lexikon
  - Lista eller
  - Hög.

# Prioritetskö som osorterad lista

- val=heltal, R=”$<$”:

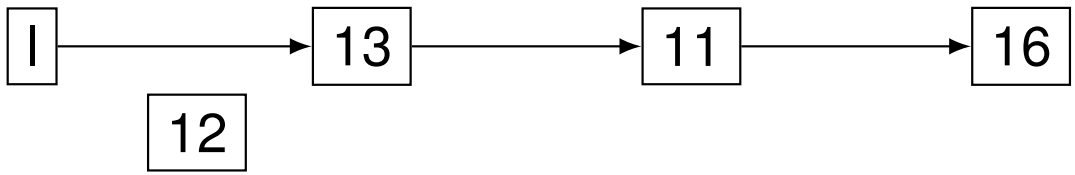
# Prioritetskö som osorterad lista

- val=heltal, R=”$<$”:
  - Insert $O(1)$:



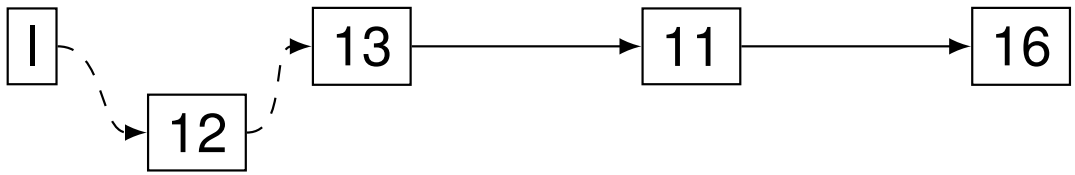
# Prioritetskö som osorterad lista

- val=heltal, R=”$<$”:
  - Insert $O(1)$:



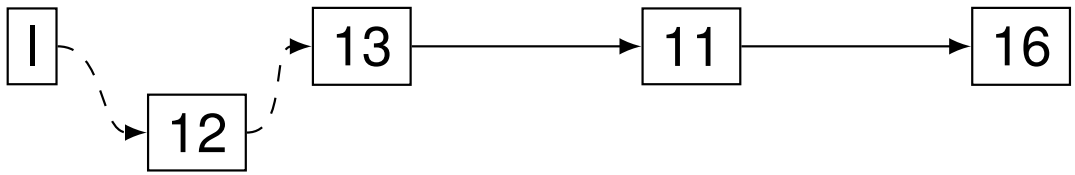
# Prioritetskö som osorterad lista

- val=heltal, R=”$<$”:
  - Insert $O(1)$:

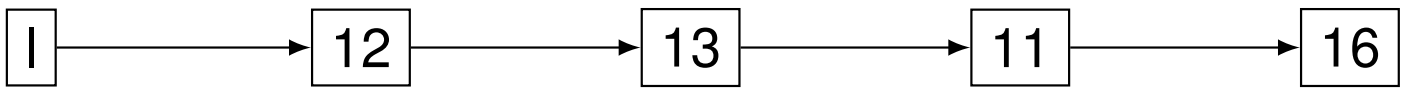


# Prioritetskö som osorterad lista

- val=heltal, R=”$<$”:
  - Insert $O(1)$:

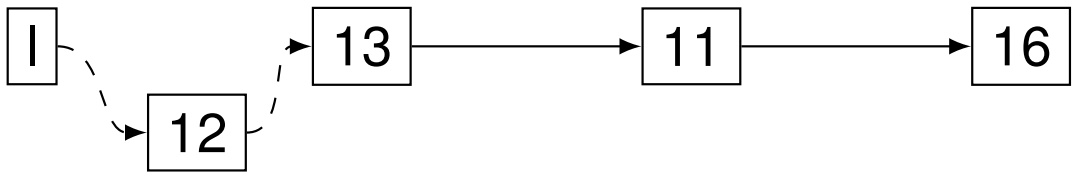



  - Inspect-First, Delete-first $O(n)$:

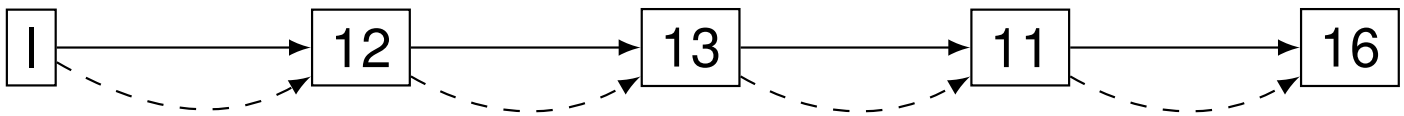


# Prioritetskö som osorterad lista

- val=heltal, R=”$<$”:
  - Insert $O(1)$:

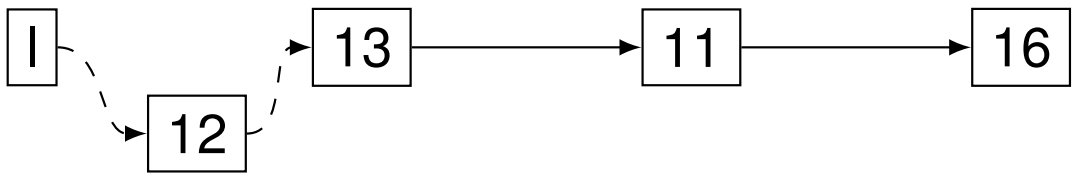



  - Inspect-First, Delete-first $O(n)$:

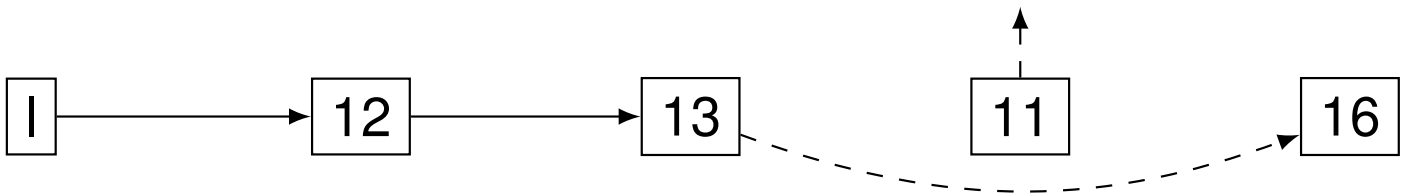


# Prioritetskö som osorterad lista

- val=heltal, R=”$<$”:
  - Insert $O(1)$:

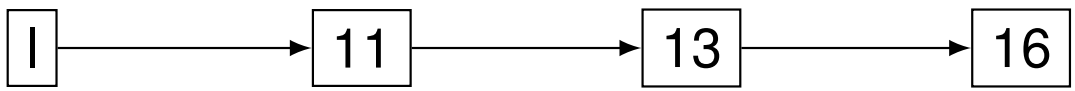



  - Inspect-First, Delete-first $O(n)$:



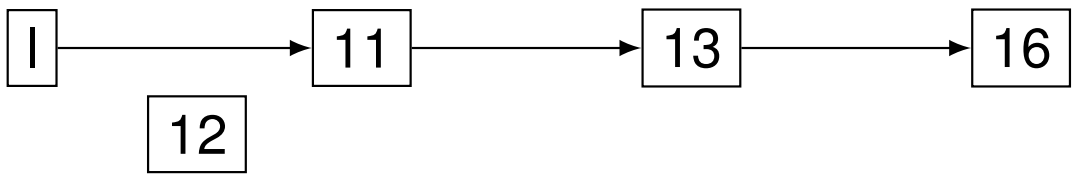
# Prioritetskö som sorterad lista

- val=heltal, R=”$<$”:
  - Insert $O(n)$:



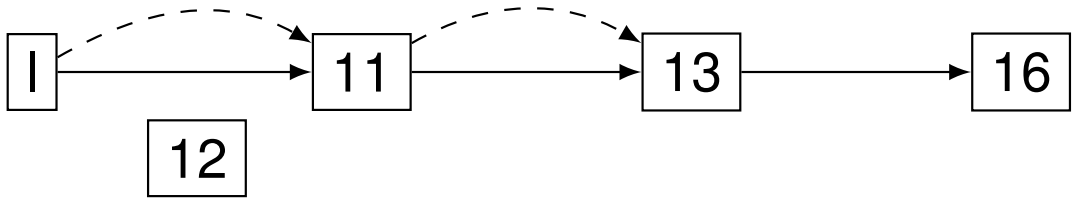
# Prioritetskö som sorterad lista

- val=heltal, R=”$<$”:
  - Insert $O(n)$:



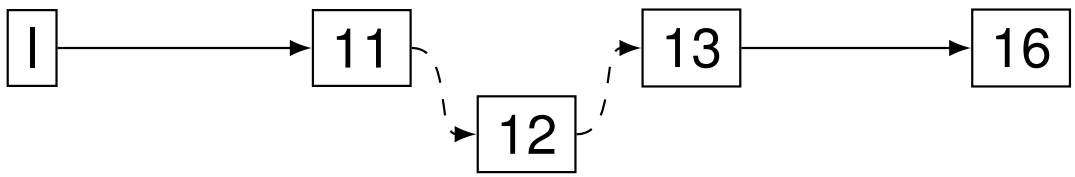
# Prioritetskö som sorterad lista

- val=heltal, R=”$<$”:
  - Insert $O(n)$:



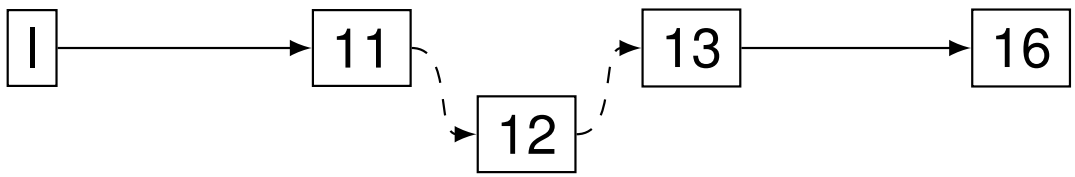
# Prioritetskö som sorterad lista

- val=heltal, R=”$<$”:
  - Insert $O(n)$:

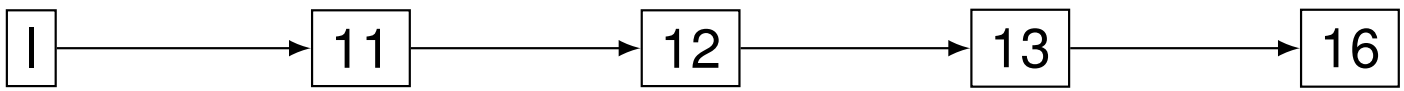


# Prioritetskö som sorterad lista

- val=heltal, R=”$<$”:
  - Insert $O(n)$:

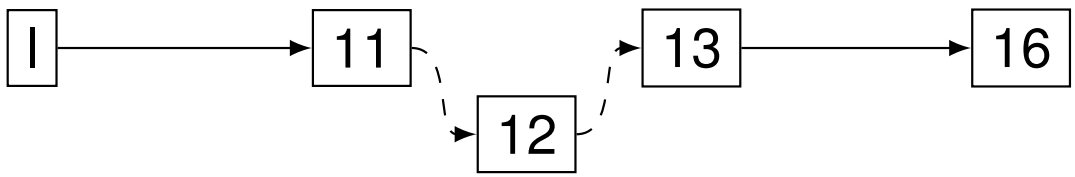



  - Inspect-first, Delete-first $O(1)$:

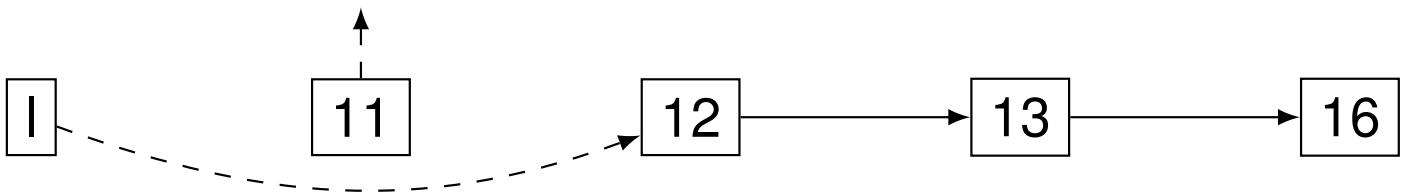


# Prioritetskö som sorterad lista

- val=heltal, R=”$<$”:
  - Insert $O(n)$:

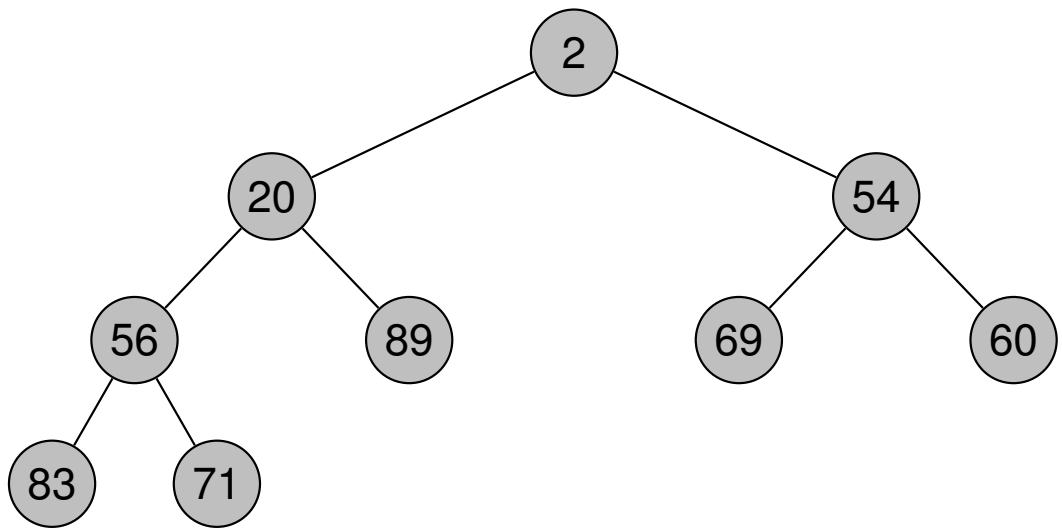



  - Inspect-first, Delete-first $O(1)$:



# Prioritetskö som hög (*heap*)

- En hög (*heap*) är ett partiellt sorterat binärt träd.
- Trädet *är en hög* eller *har hög-egenskapen* för en relation R om och endast om:
  - Trädet är sorterat så att etiketterna för alla föräldra-barn-par uppfyller $a$ R $b$, där $a$ är föräldraetiketten och $b$ är barnetiketten.
  - Exempel: Heap med R $<$:

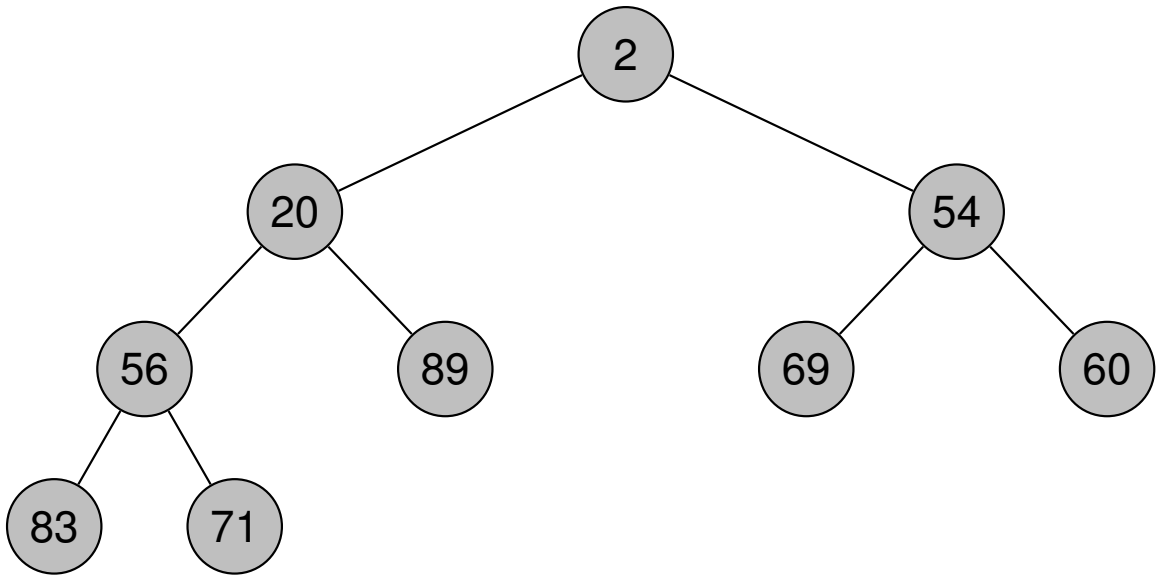



- Insättningar och borttagningar blir effektiva om dom görs så att trädet hålls komplett.

# Heap — Algoritm för insättning

1. Sätt in det nya elementet på den första lediga platsen.
2. Sortera om grenen tills trädet är en hög.

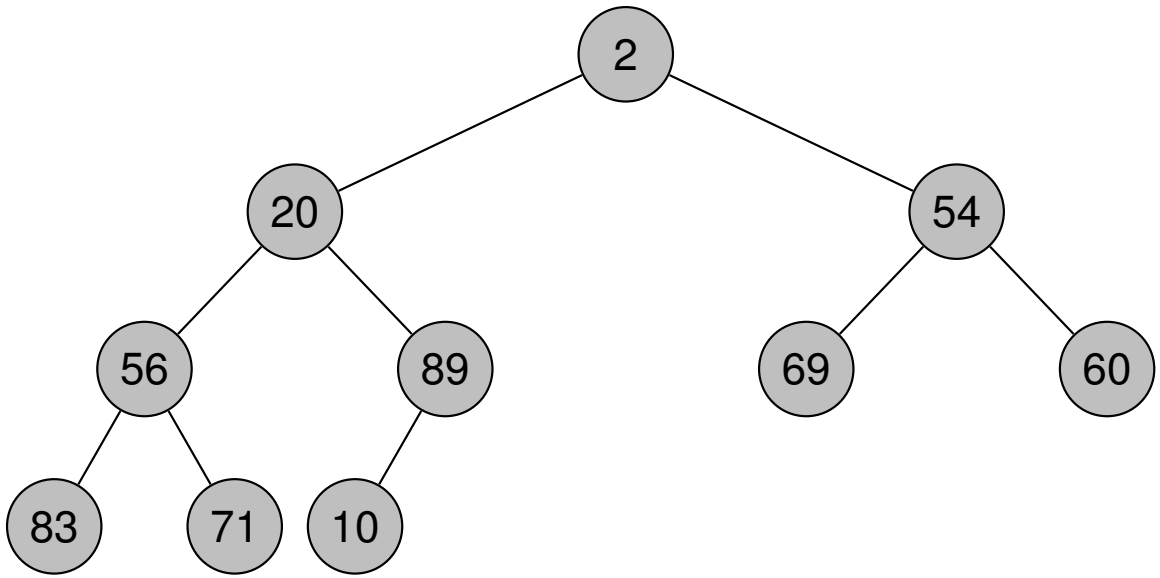
- Exempel: Sortera in 10:



# Heap — Algoritm för insättning

1. Sätt in det nya elementet på den första lediga platsen.
2. Sortera om grenen tills trädet är en hög.

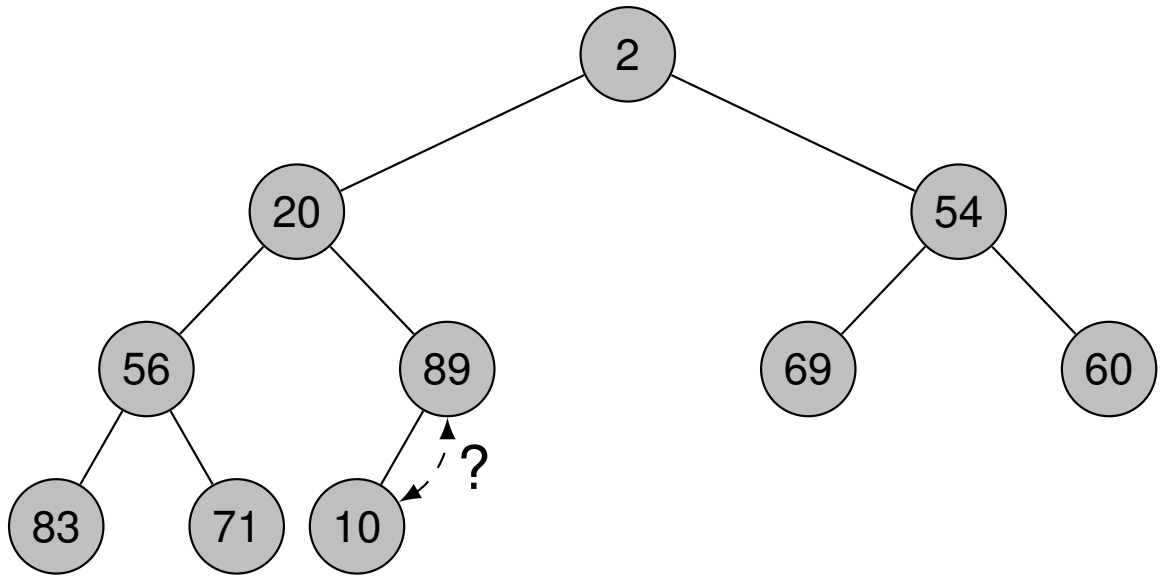
- Exempel: Sortera in 10:



# Heap — Algoritm för insättning

1. Sätt in det nya elementet på den första lediga platsen.
2. Sortera om grenen tills trädet är en hög.

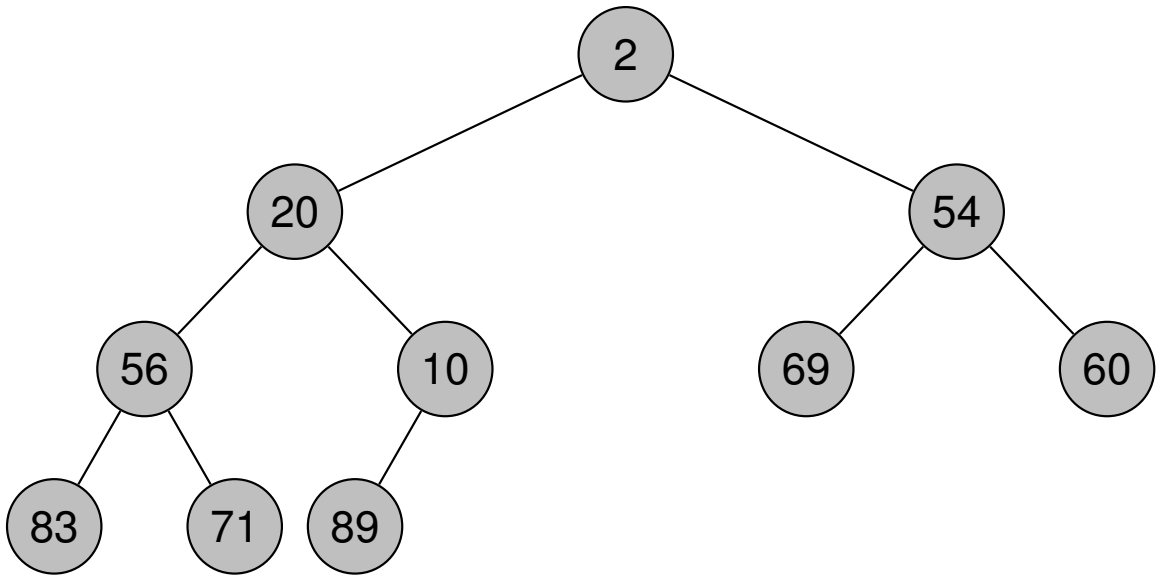
- Exempel: Sortera in 10:



# Heap — Algoritm för insättning

1. Sätt in det nya elementet på den första lediga platsen.
2. Sortera om grenen tills trädet är en hög.

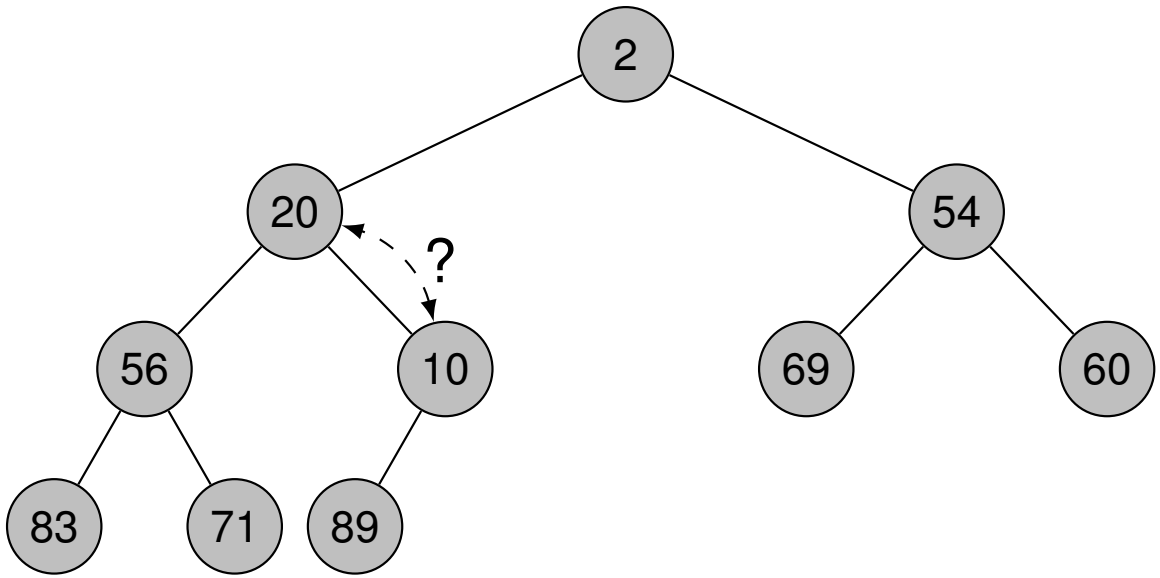
- Exempel: Sortera in 10:



# Heap — Algoritm för insättning

1. Sätt in det nya elementet på den första lediga platsen.
2. Sortera om grenen tills trädet är en hög.

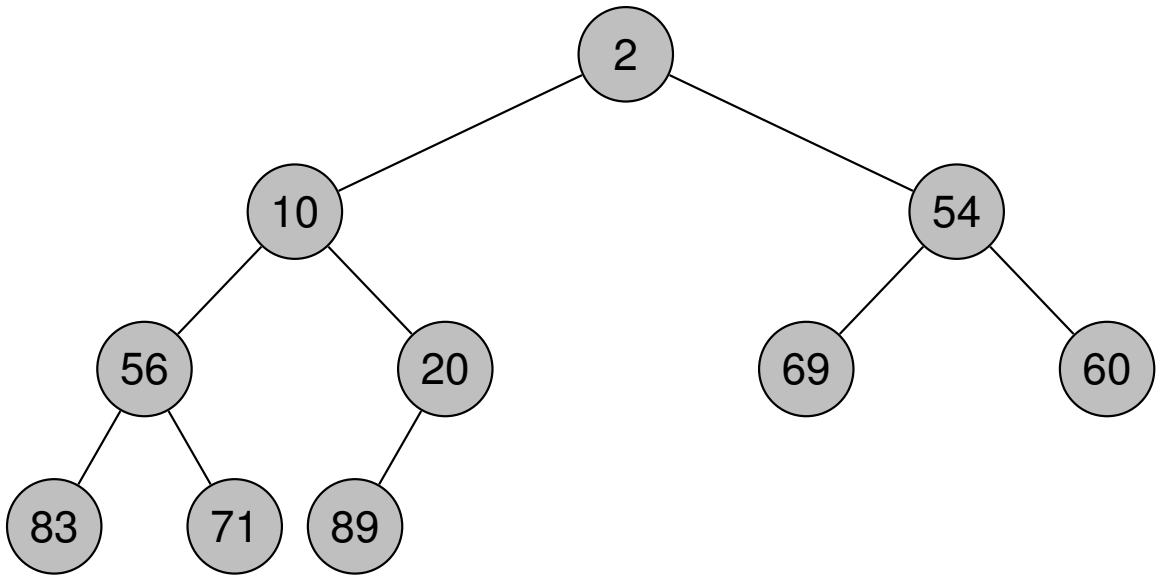
- Exempel: Sortera in 10:



# Heap — Algoritm för insättning

1. Sätt in det nya elementet på den första lediga platsen.
2. Sortera om grenen tills trädet är en hög.

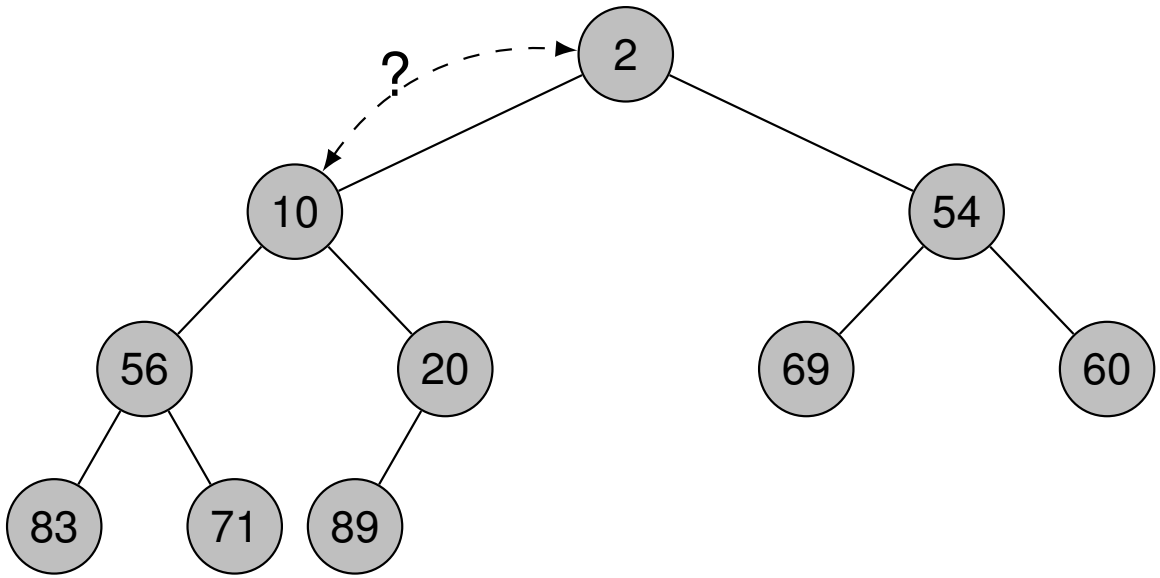
- Exempel: Sortera in 10:



# Heap — Algoritm för insättning

1. Sätt in det nya elementet på den första lediga platsen.
2. Sortera om grenen tills trädet är en hög.

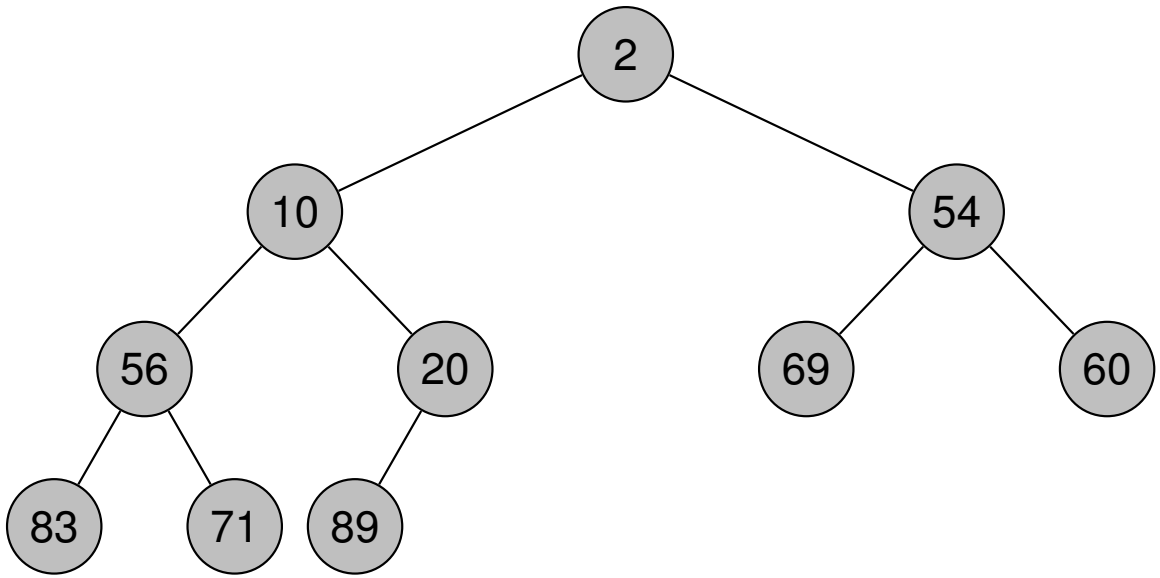
- Exempel: Sortera in 10:



# Heap — Algoritm för insättning

1. Sätt in det nya elementet på den första lediga platsen.
2. Sortera om grenen tills trädet är en hög.

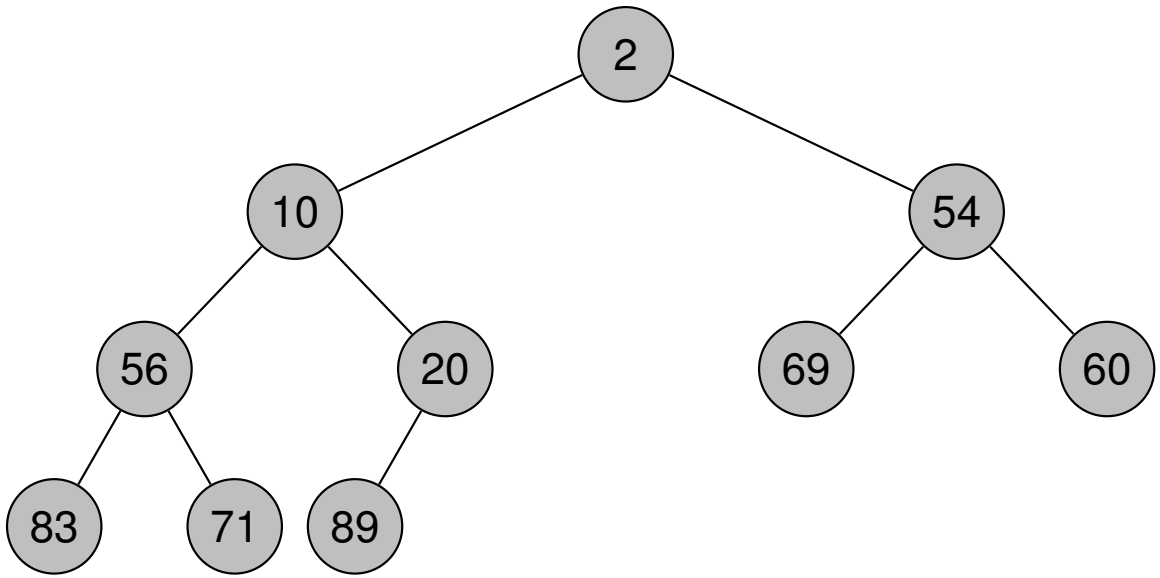
- Exempel: Sortera in 10:



# Heap — Algoritm för insättning

1. Sätt in det nya elementet på den första lediga platsen.
2. Sortera om grenen tills trädet är en hög.

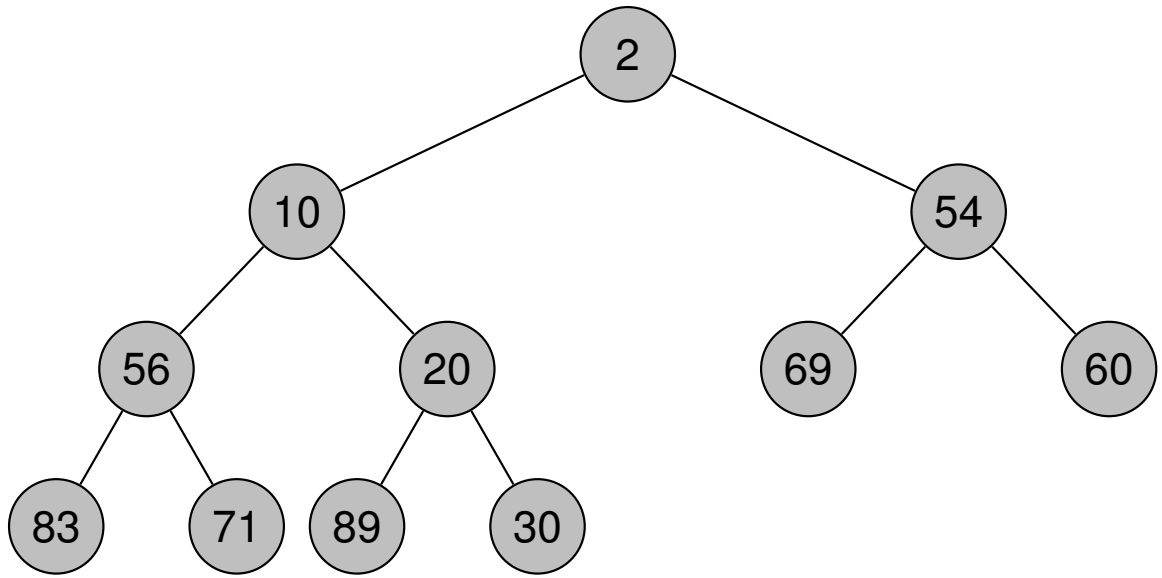
- Exempel: Sortera in 10:
- Exempel: Sortera in 30:



# Heap — Algoritm för insättning

1. Sätt in det nya elementet på den första lediga platsen.
2. Sortera om grenen tills trädet är en hög.

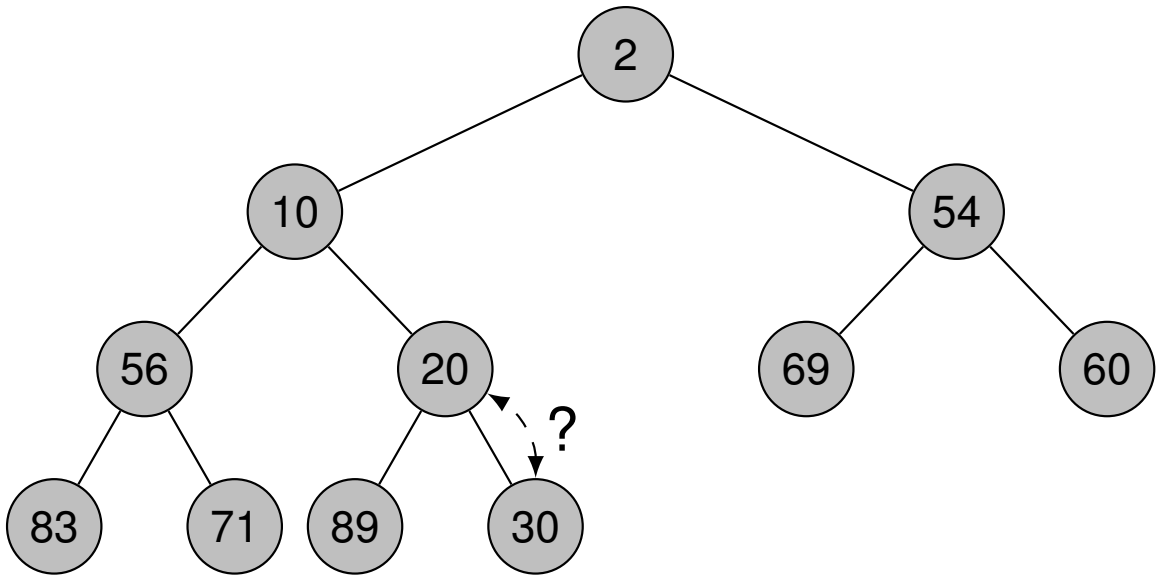
- Exempel: Sortera in 10:
- Exempel: Sortera in 30:



# Heap — Algoritm för insättning

1. Sätt in det nya elementet på den första lediga platsen.
2. Sortera om grenen tills trädet är en hög.

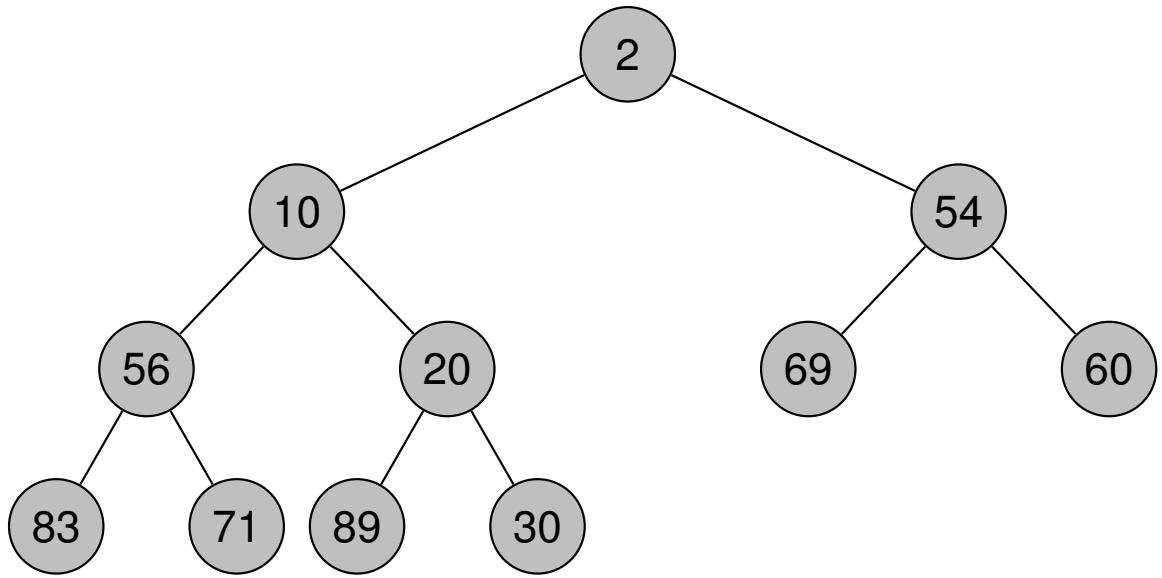
- Exempel: Sortera in 10:
- Exempel: Sortera in 30:



# Heap — Algoritm för insättning

1. Sätt in det nya elementet på den första lediga platsen.
2. Sortera om grenen tills trädet är en hög.

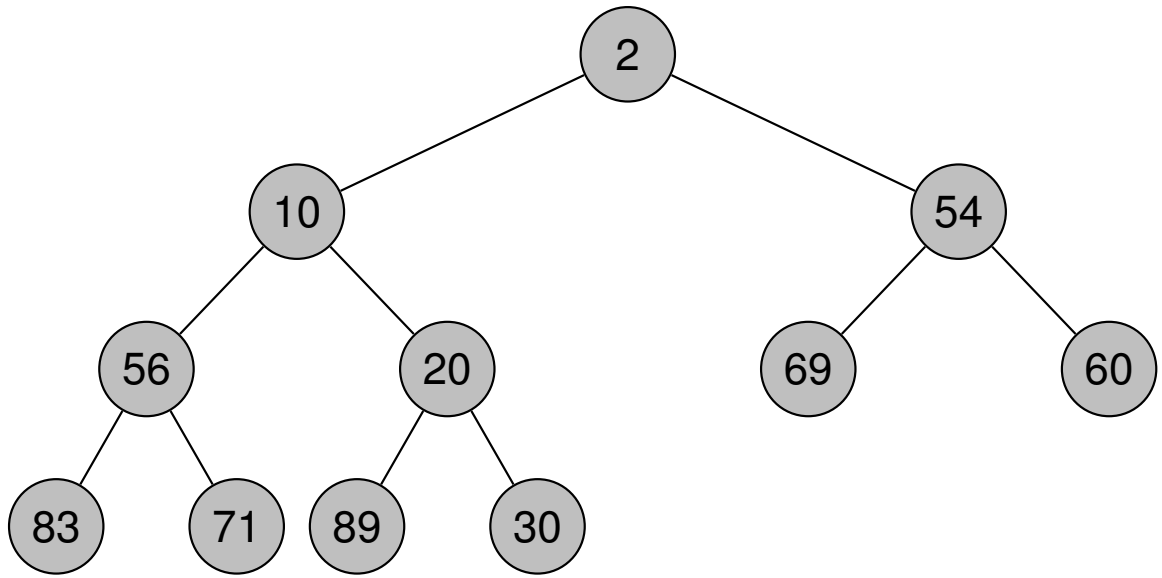
- Exempel: Sortera in 10:
- Exempel: Sortera in 30:



# Heap — Algoritm för insättning

1. Sätt in det nya elementet på den första lediga platsen.
2. Sortera om grenen tills trädet är en hög.

- Exempel: Sortera in 10:
- Exempel: Sortera in 30:

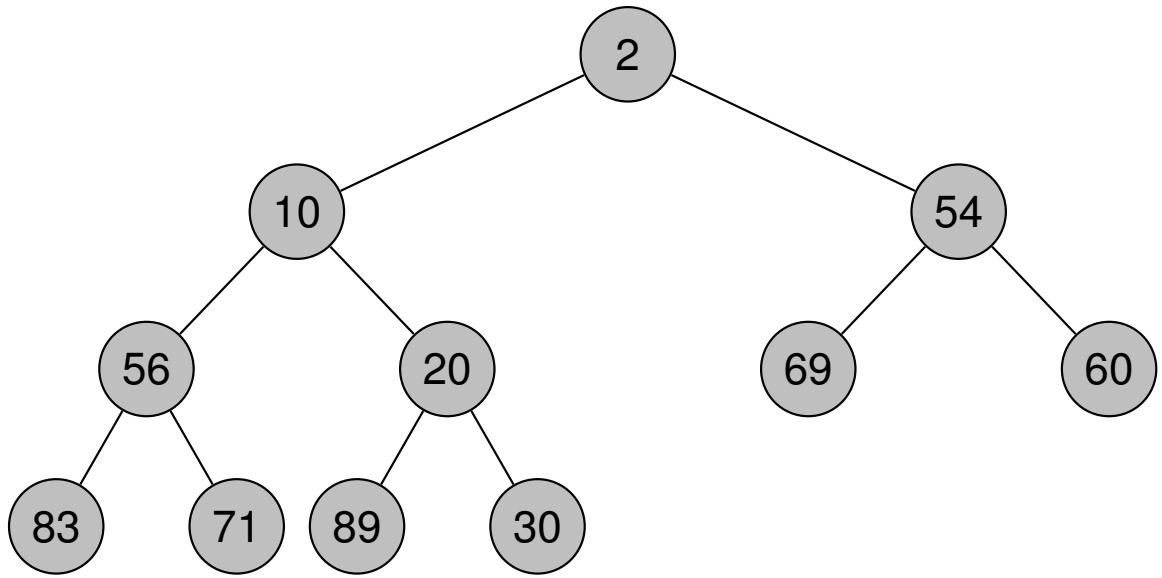



- Komplexitet för insättning av ett element?

# Heap — Algoritm för insättning

1. Sätt in det nya elementet på den första lediga platsen.
2. Sortera om grenen tills trädet är en hög.

- Exempel: Sortera in 10:
- Exempel: Sortera in 30:

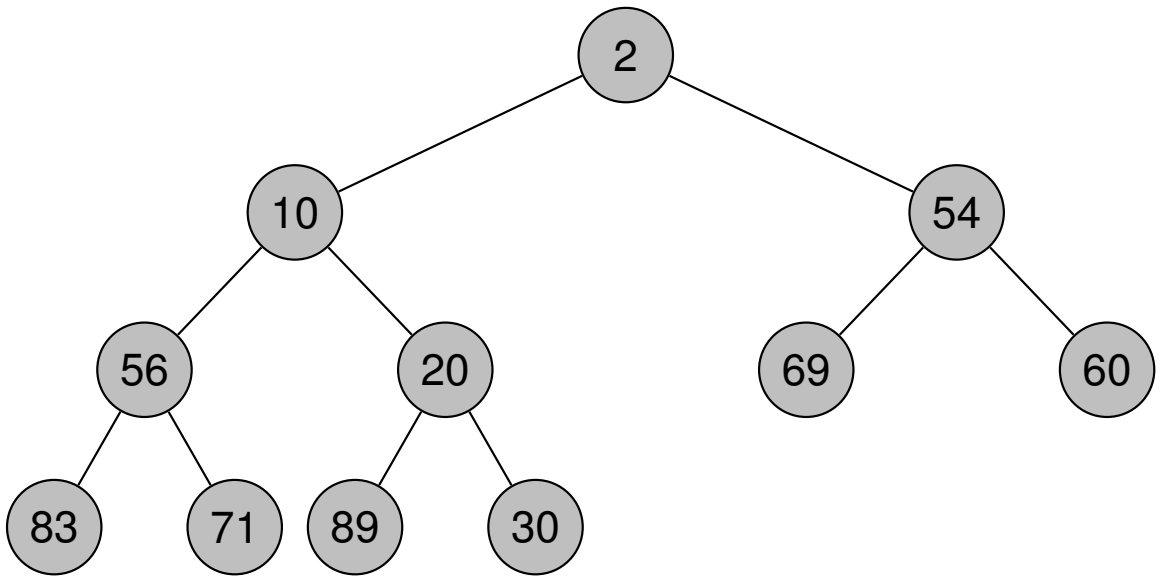



- Komplexitet för insättning av ett element?
  - $O(\log n)$.

# Heap — Algoritm för borttagning

1. Ta bort toppelementet.
2. Flytta sista elementet till toppen.
3. Om nödvändigt,
   - 3.1 Byt ut toppelementet mot det minsta av dess barn.
   - 3.2 Fortsätt nedåt i den påverkade grenen.

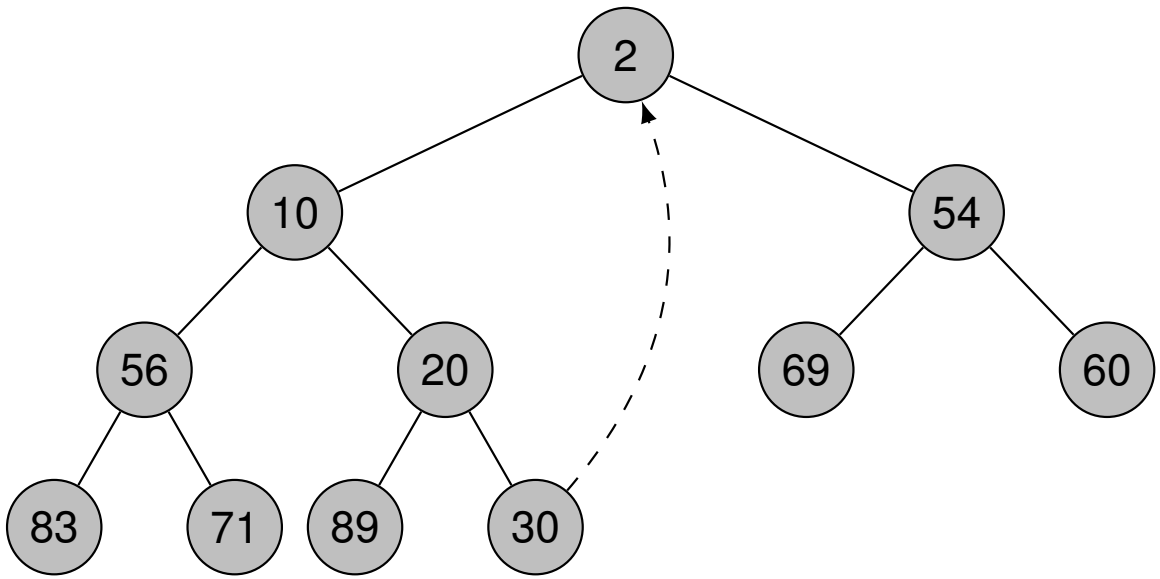
- Exempel: Remove-first:



# Heap — Algoritm för borttagning

1. Ta bort toppelementet.
2. Flytta sista elementet till toppen.
3. Om nödvändigt,
   - 3.1 Byt ut toppelementet mot det minsta av dess barn.
   - 3.2 Fortsätt nedåt i den påverkade grenen.

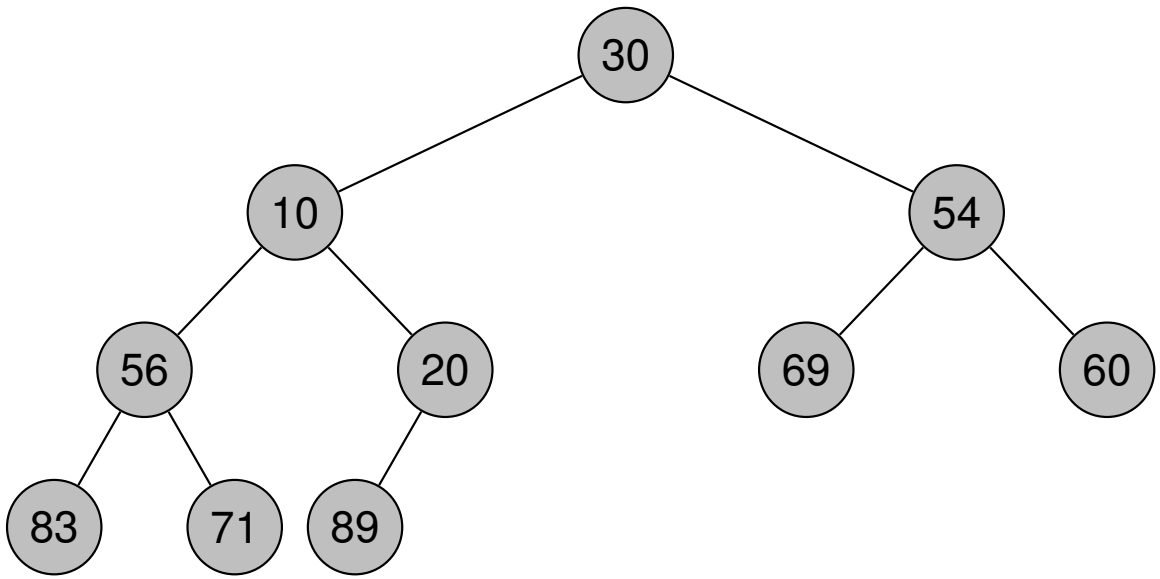
- Exempel: Remove-first:



# Heap — Algoritm för borttagning

1. Ta bort toppelementet.
2. Flytta sista elementet till toppen.
3. Om nödvändigt,
   - 3.1 Byt ut toppelementet mot det minsta av dess barn.
   - 3.2 Fortsätt nedåt i den påverkade grenen.

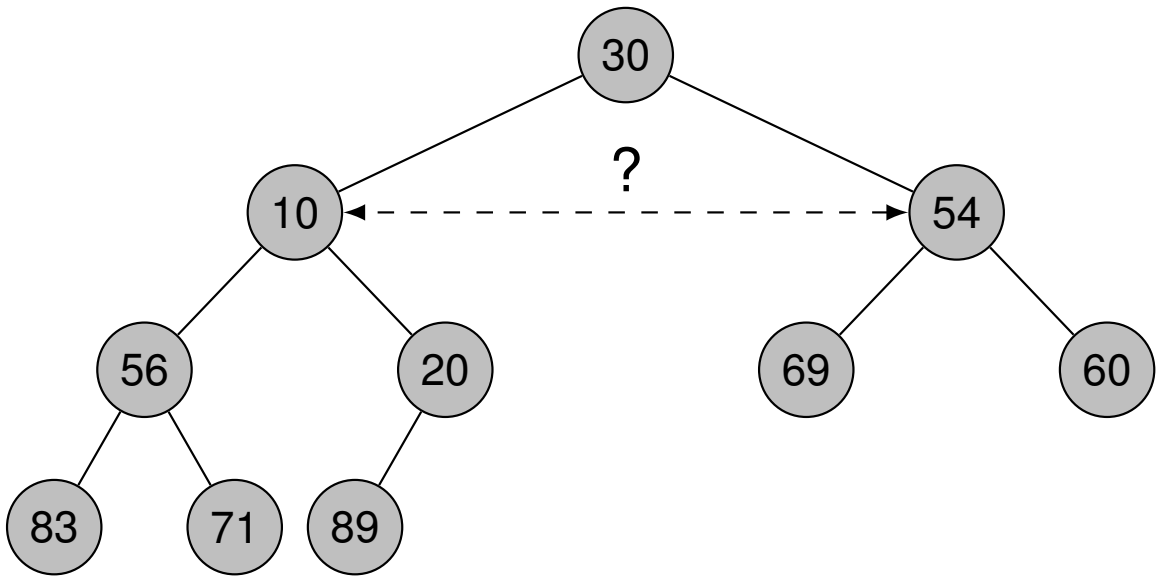
- Exempel: Remove-first:



# Heap — Algoritm för borttagning

1. Ta bort toppelementet.
2. Flytta sista elementet till toppen.
3. Om nödvändigt,
   - 3.1 Byt ut toppelementet mot det minsta av dess barn.
   - 3.2 Fortsätt nedåt i den påverkade grenen.

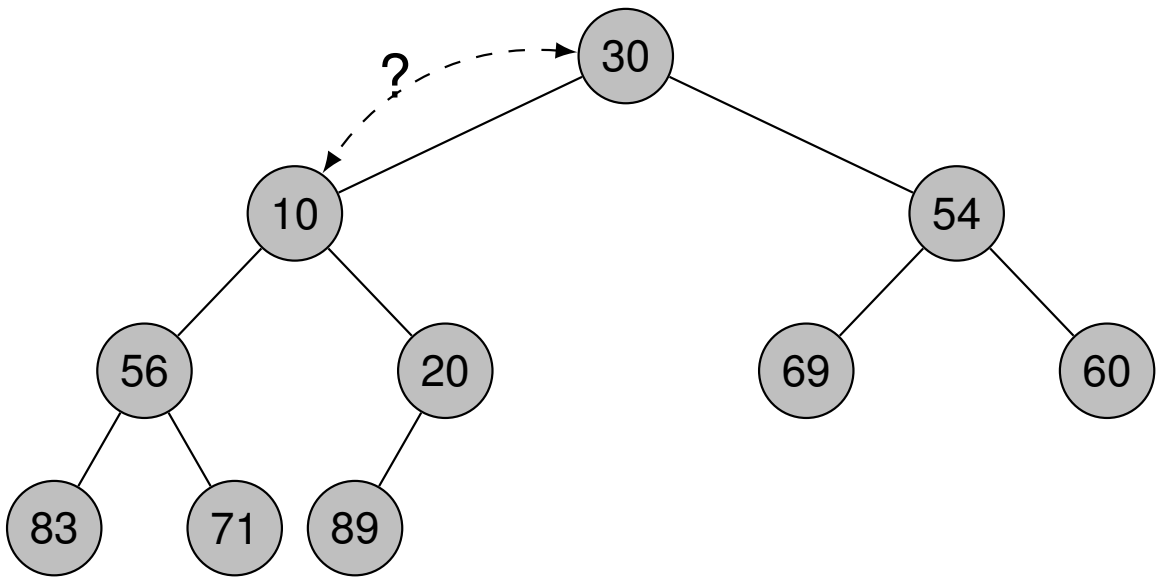
- Exempel: Remove-first:



# Heap — Algoritm för borttagning

1. Ta bort toppelementet.
2. Flytta sista elementet till toppen.
3. Om nödvändigt,
   - 3.1 Byt ut toppelementet mot det minsta av dess barn.
   - 3.2 Fortsätt nedåt i den påverkade grenen.

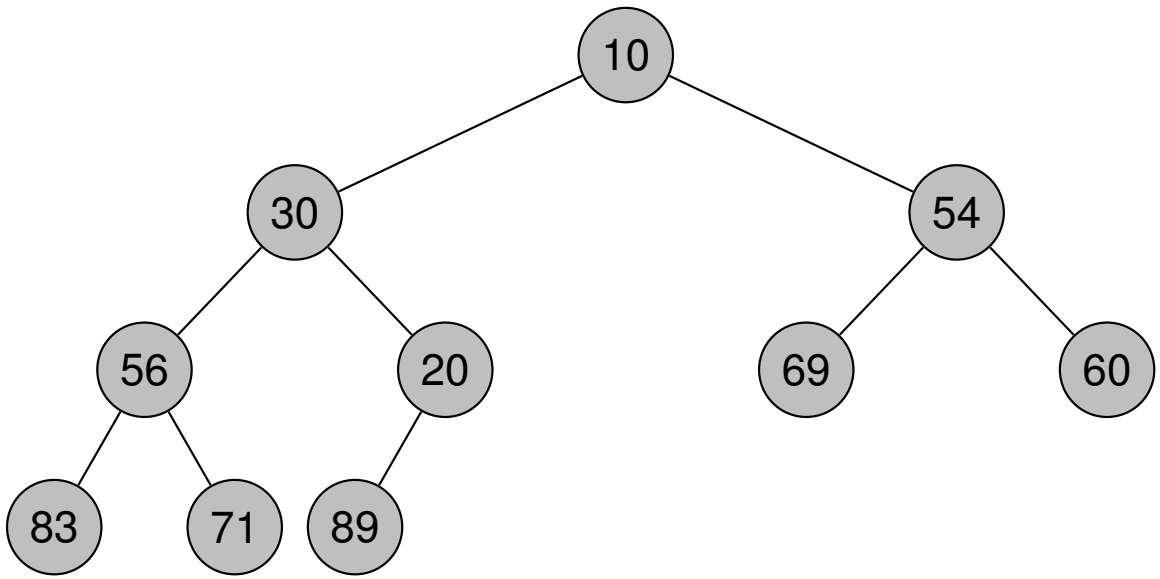
- Exempel: Remove-first:



# Heap — Algoritm för borttagning

1. Ta bort toppelementet.
2. Flytta sista elementet till toppen.
3. Om nödvändigt,
   - 3.1 Byt ut toppelementet mot det minsta av dess barn.
   - 3.2 Fortsätt nedåt i den påverkade grenen.

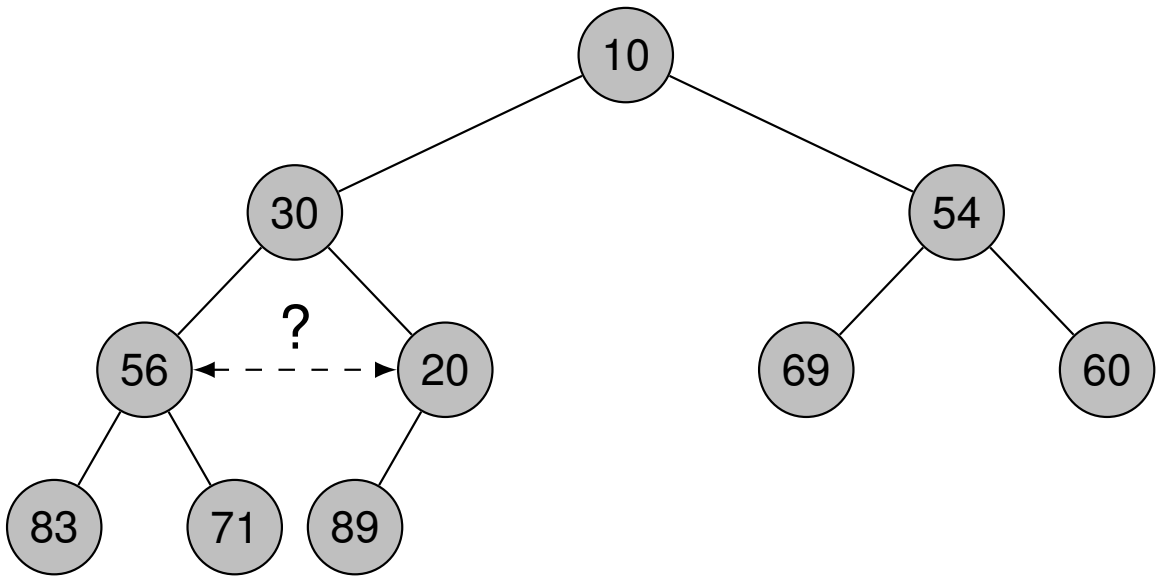
- Exempel: Remove-first:



# Heap — Algoritm för borttagning

1. Ta bort toppelementet.
2. Flytta sista elementet till toppen.
3. Om nödvändigt,
   - 3.1 Byt ut toppelementet mot det minsta av dess barn.
   - 3.2 Fortsätt nedåt i den påverkade grenen.

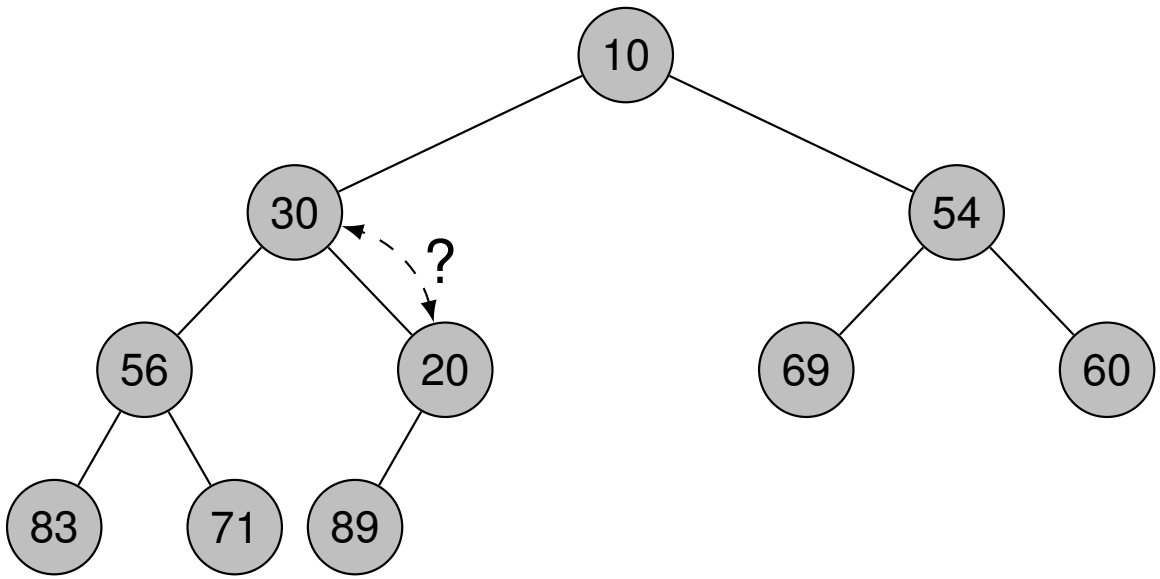
- Exempel: Remove-first:



# Heap — Algoritm för borttagning

1. Ta bort toppelementet.
2. Flytta sista elementet till toppen.
3. Om nödvändigt,
   - 3.1 Byt ut toppelementet mot det minsta av dess barn.
   - 3.2 Fortsätt nedåt i den påverkade grenen.

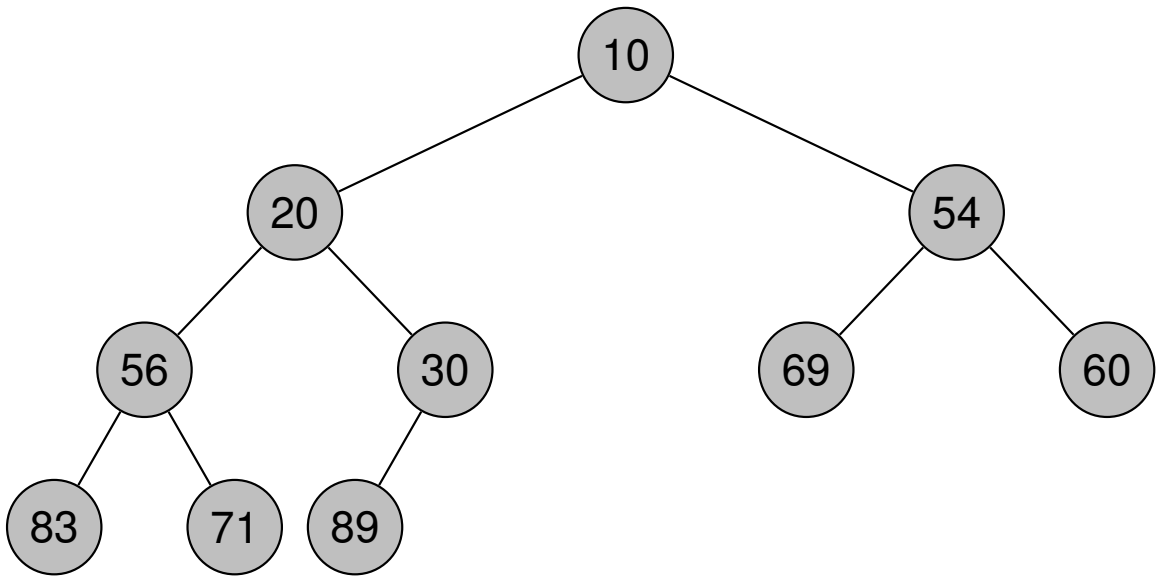
- Exempel: Remove-first:



# Heap — Algoritm för borttagning

1. Ta bort toppelementet.
2. Flytta sista elementet till toppen.
3. Om nödvändigt,
   - 3.1 Byt ut toppelementet mot det minsta av dess barn.
   - 3.2 Fortsätt nedåt i den påverkade grenen.

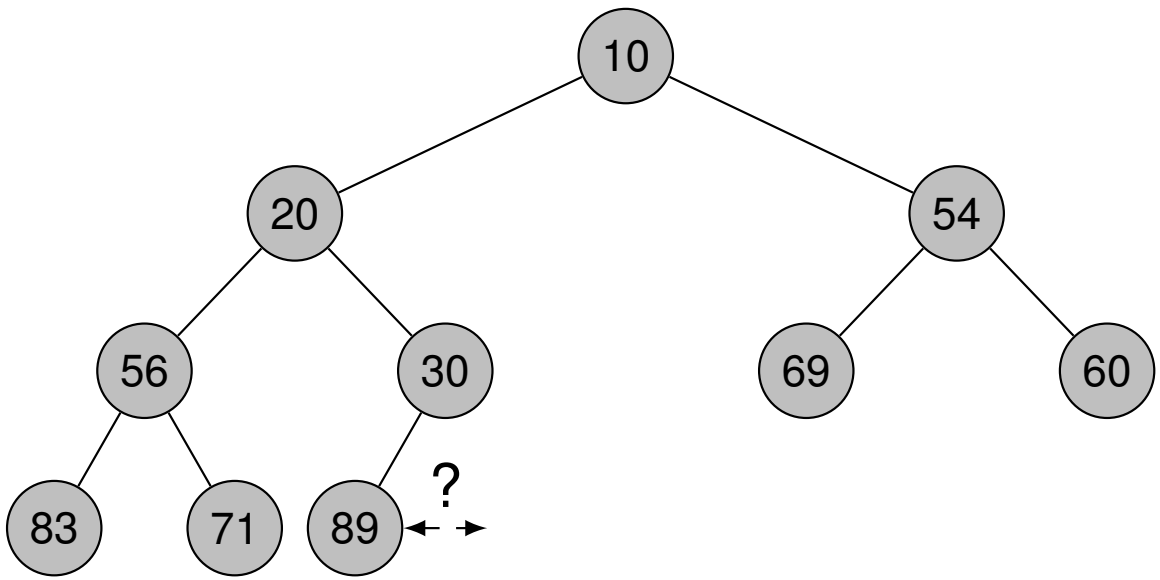
- Exempel: Remove-first:



# Heap — Algoritm för borttagning

1. Ta bort toppelementet.
2. Flytta sista elementet till toppen.
3. Om nödvändigt,
   - 3.1 Byt ut toppelementet mot det minsta av dess barn.
   - 3.2 Fortsätt nedåt i den påverkade grenen.

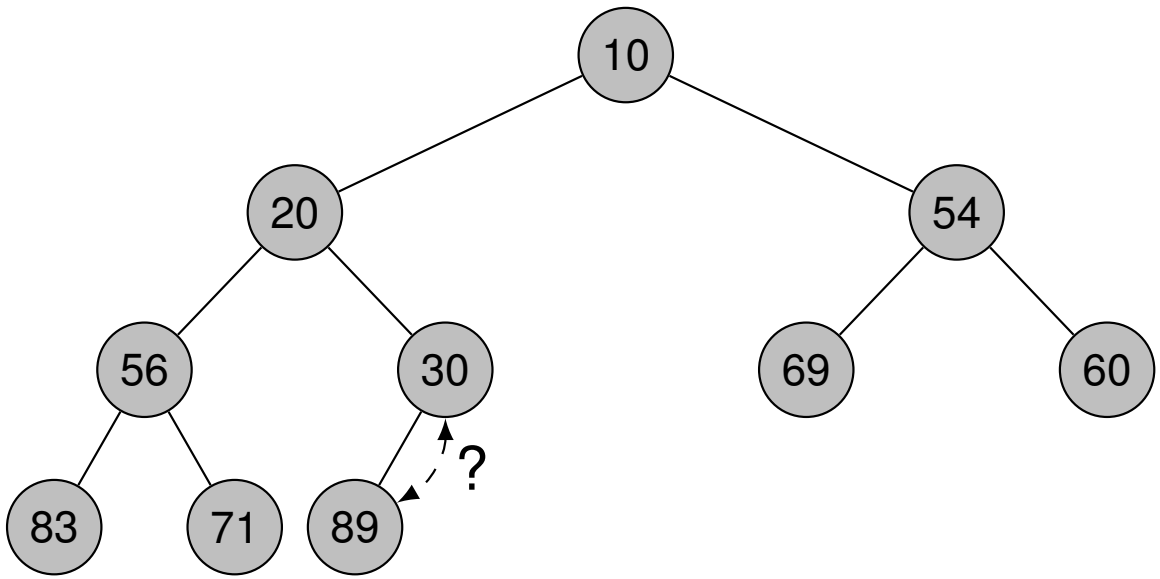
- Exempel: Remove-first:



# Heap — Algoritm för borttagning

1. Ta bort toppelementet.
2. Flytta sista elementet till toppen.
3. Om nödvändigt,
   - 3.1 Byt ut toppelementet mot det minsta av dess barn.
   - 3.2 Fortsätt nedåt i den påverkade grenen.

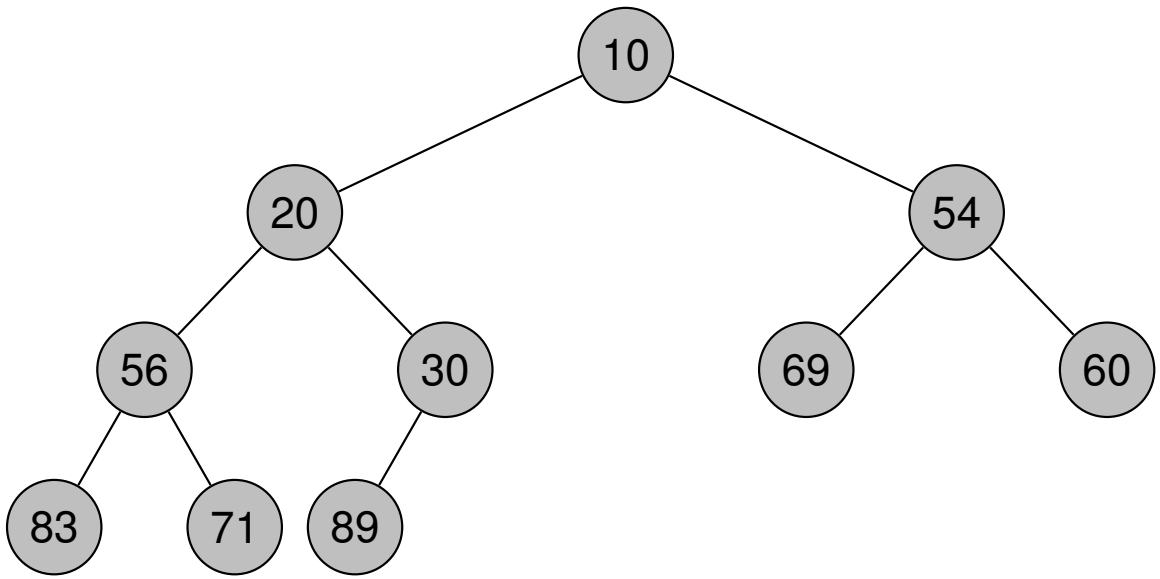
- Exempel: Remove-first:



# Heap — Algoritm för borttagning

1. Ta bort toppelementet.
2. Flytta sista elementet till toppen.
3. Om nödvändigt,
   - 3.1 Byt ut toppelementet mot det minsta av dess barn.
   - 3.2 Fortsätt nedåt i den påverkade grenen.

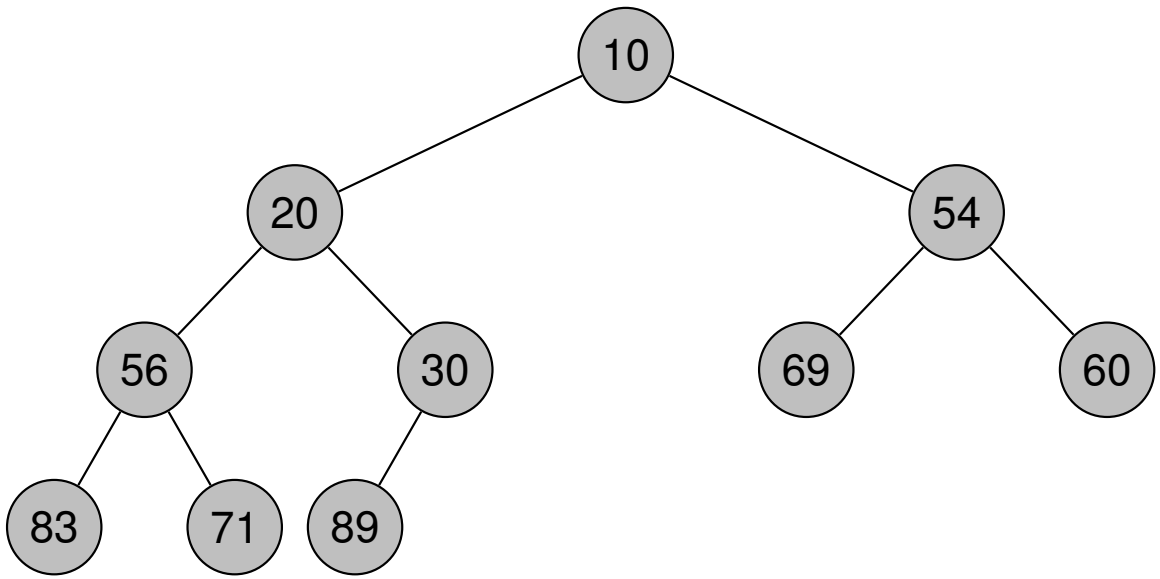
- Exempel: Remove-first:



# Heap — Algoritm för borttagning

1. Ta bort toppelementet.
2. Flytta sista elementet till toppen.
3. Om nödvändigt,
   - 3.1 Byt ut toppelementet mot det minsta av dess barn.
   - 3.2 Fortsätt nedåt i den påverkade grenen.

- Exempel: Remove-first:

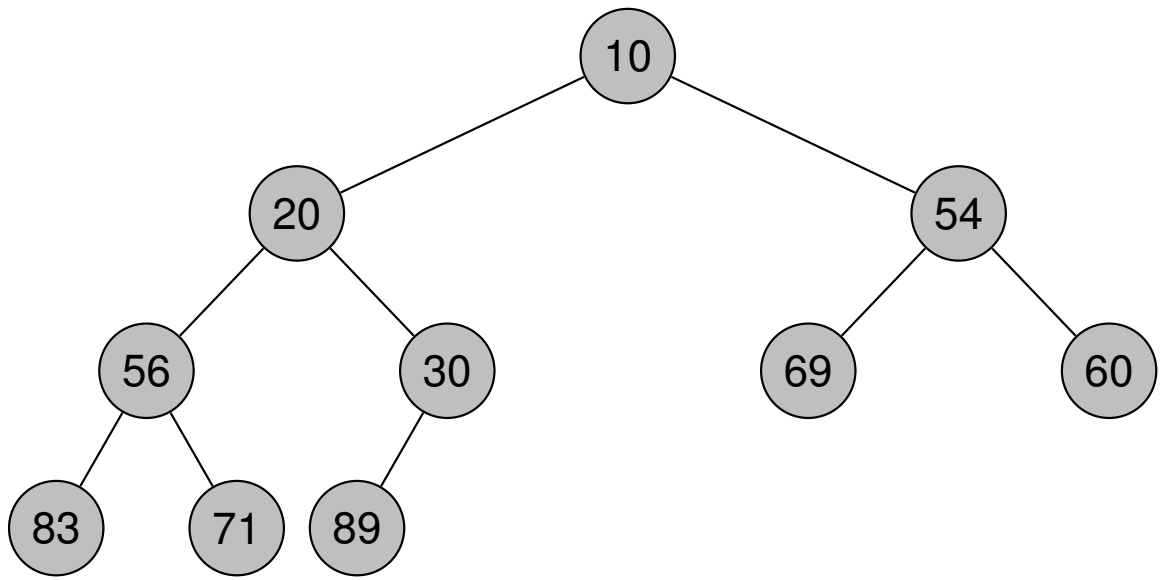



- Komplexitet för borttagning av ett element?

# Heap — Algoritm för borttagning

1. Ta bort toppelementet.
2. Flytta sista elementet till toppen.
3. Om nödvändigt,
   - 3.1 Byt ut toppelementet mot det minsta av dess barn.
   - 3.2 Fortsätt nedåt i den påverkade grenen.

- Exempel: Remove-first:




- Komplexitet för borttagning av ett element?
  - $O(\log n)$.

# Tillämpningar

- Operativsystem som fördelar jobb mellan olika processer.
- Enkelt sätt att sortera något:
  - Stoppa in allt i en prioritetskö och plocka ut det igen — *heapsort*.
- Hjälpmedel vid traversering av graf:
  - Jämför att stack och kö används vid traversering av träd.