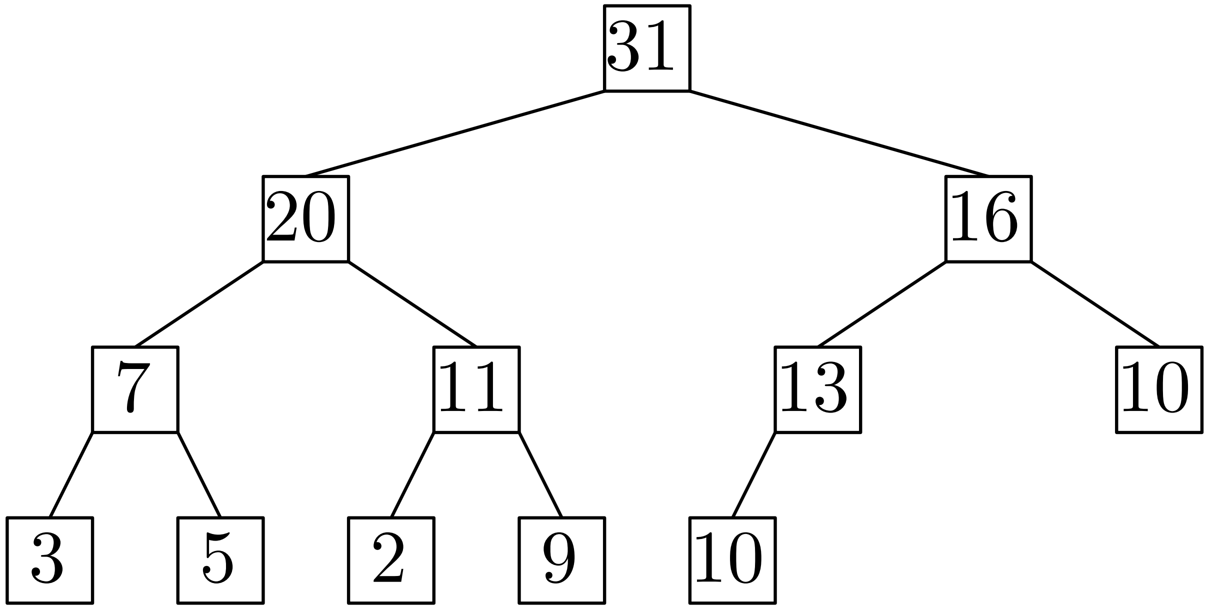
Prioritetskøer, hobe og heap sort

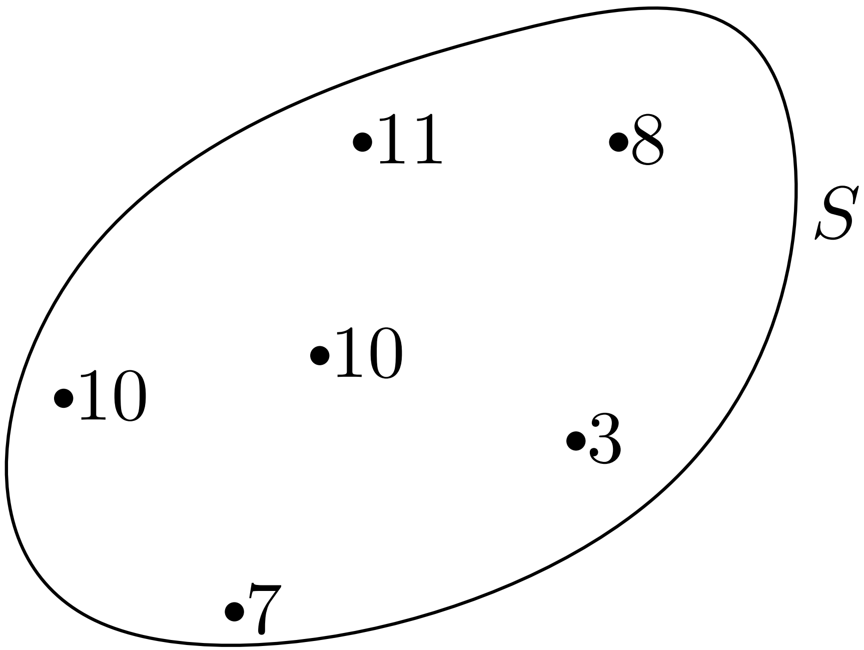


Mikkel Abrahamsen

Prioritetskø

Dynamisk multi-mængde S af *nøgler*.
To (vigtigste) operationer:

- Extract-Max(S): fjern og returnér største nøgle.
- Insert(S, k): tilføj k til S.

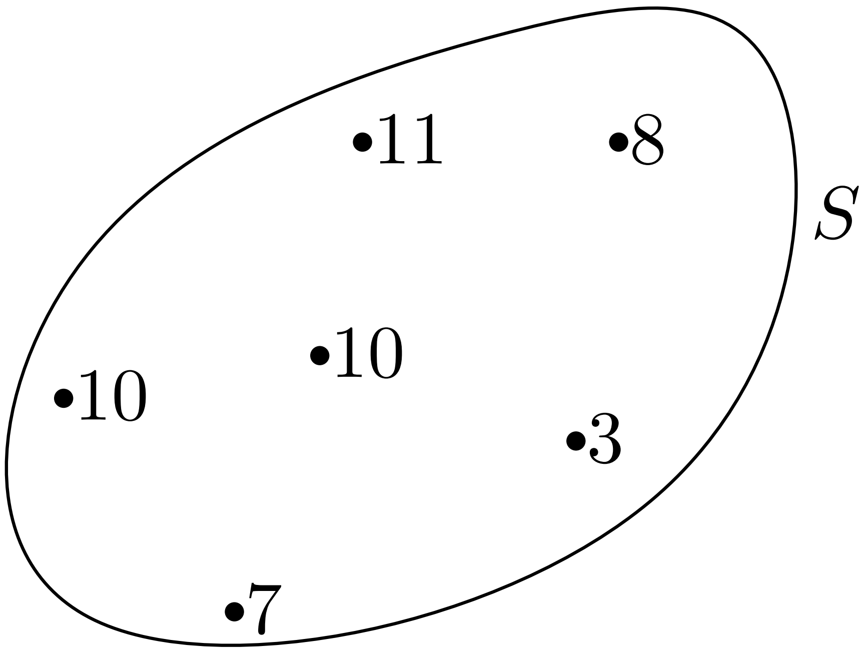


Prioritetskø

Dynamisk multi-mængde S af *nøgler*.
To (vigtigste) operationer:

- Extract-Max(S): fjern og returnér største nøgle.
- Insert(S, k): tilføj k til S.

Extract-Max(S)

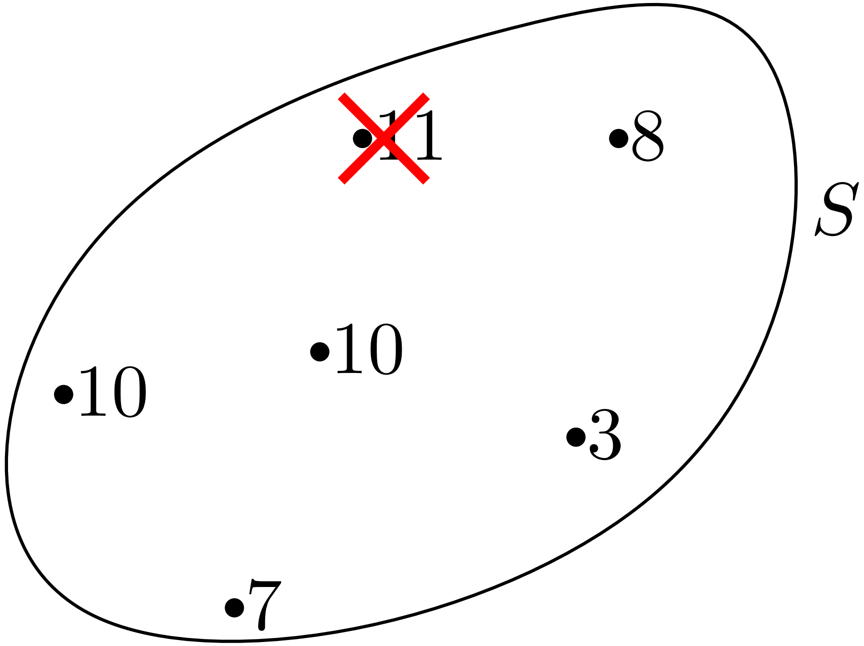


Prioritetskø

Dynamisk multi-mængde S af *nøgler*.
To (vigtigste) operationer:

- Extract-Max(S): fjern og returnér største nøgle.
- Insert(S, k): tilføj k til S.

Extract-Max(S) returnér 11



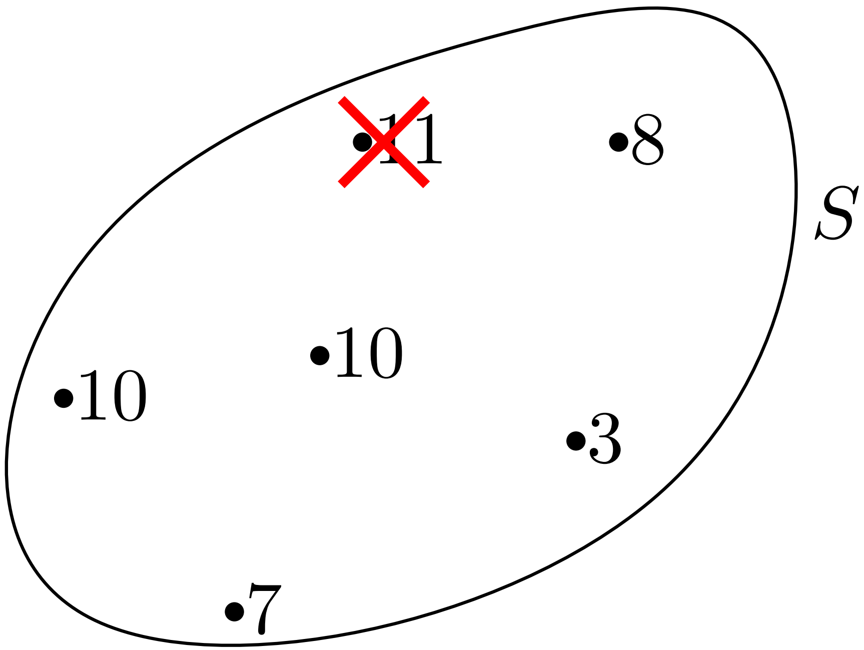
Prioritetskø

Dynamisk multi-mængde S af *nøgler*.
To (vigtigste) operationer:

- Extract-Max(S): fjern og returnér største nøgle.
- Insert(S, k): tilføj k til S.

Extract-Max(S) returnér 11

Extract-Max(S)



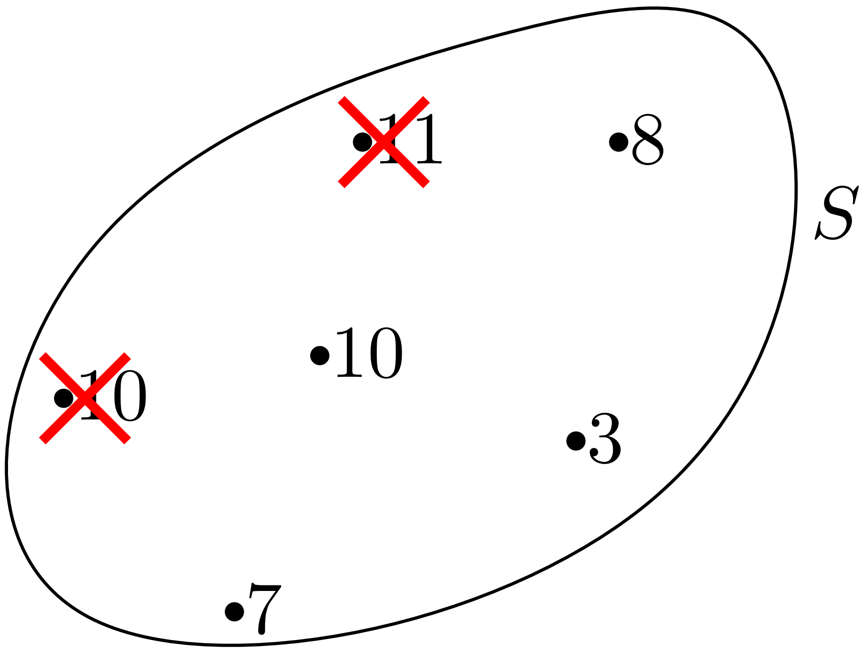
Prioritetskø

Dynamisk multi-mængde S af *nøgler*.
To (vigtigste) operationer:

- Extract-Max(S): fjern og returnér største nøgle.
- Insert(S, k): tilføj k til S.

Extract-Max(S) returnér 11

Extract-Max(S) returnér 10



Prioritetskø

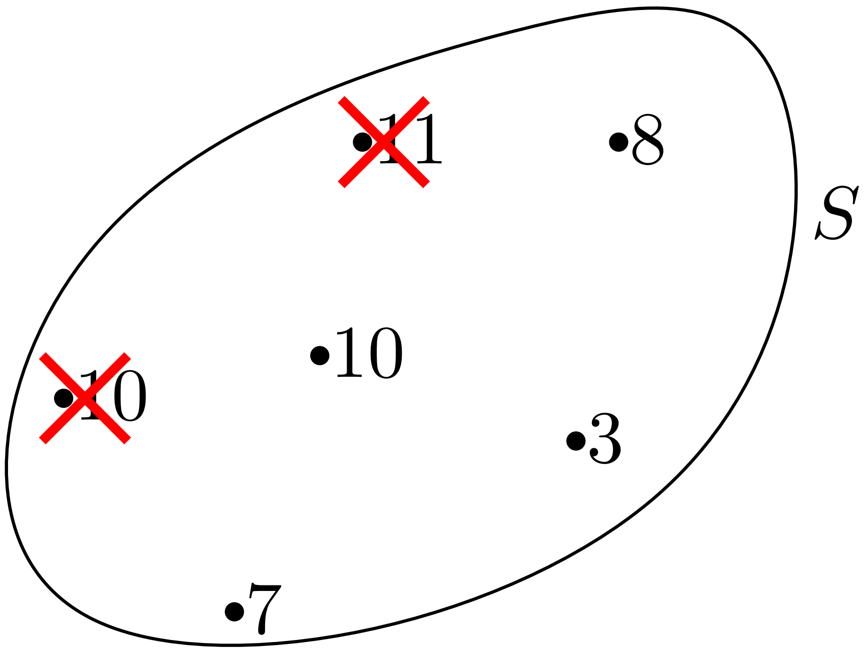
Dynamisk multi-mængde S af *nøgler*.
To (vigtigste) operationer:

- Extract-Max(S): fjern og returnér største nøgle.
- Insert(S, k): tilføj k til S.

Extract-Max(S) returnér 11

Extract-Max(S) returnér 10

Insert($S, 13$)



Prioritetskø

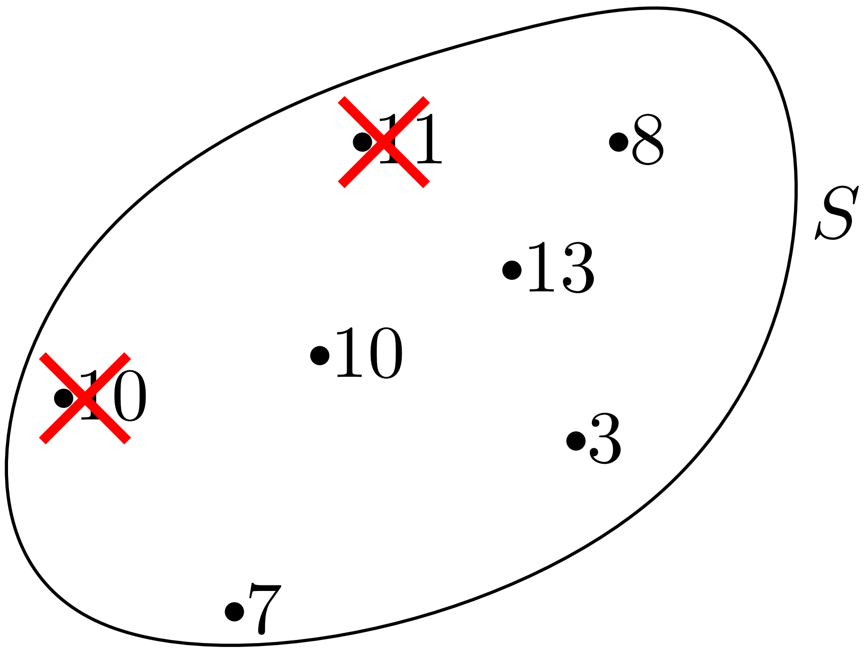
Dynamisk multi-mængde S af *nøgler*.
To (vigtigste) operationer:

- Extract-Max(S): fjern og returnér største nøgle.
- Insert(S, k): tilføj k til S.

Extract-Max(S) returnér 11

Extract-Max(S) returnér 10

Insert($S, 13$)



Prioritetskø

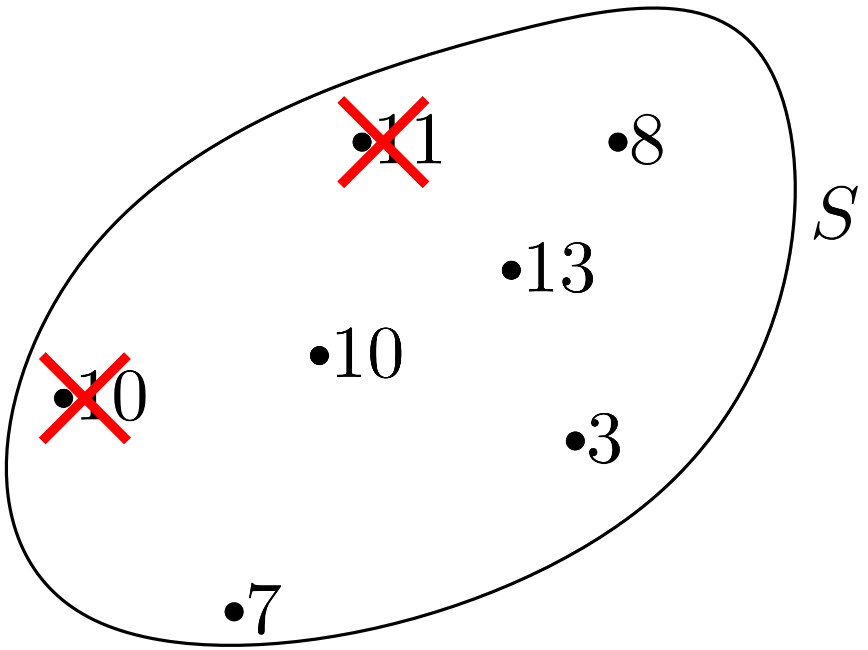
Dynamisk multi-mængde S af *nøgler*.
To (vigtigste) operationer:

- Extract-Max(S): fjern og returnér største nøgle.
- Insert(S, k): tilføj k til S.

Extract-Max(S) returnér 11

Extract-Max(S) returnér 10

Insert($S, 13$)



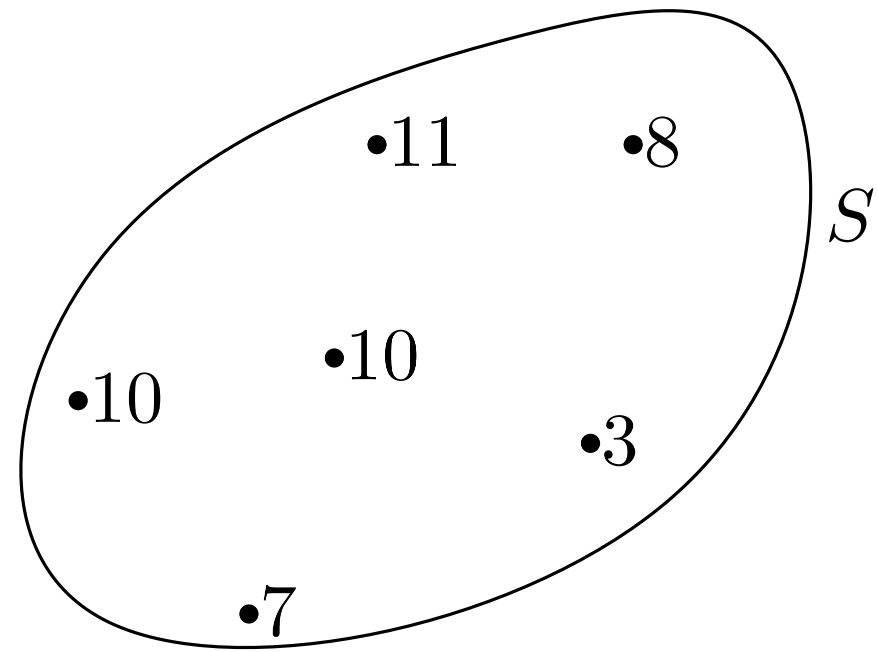
I praksis: Vi gemmer hver nøgle sammen med *sattelitdata*: (k,data)

Kendte teknikker

Array:

0	1	2	...					
3	10	7	11	8	10			

Extract-Max(S): $\Theta(n)$ tid.
Insert(S, k): $\Theta(1)$ tid.



Kendte teknikker

Array:

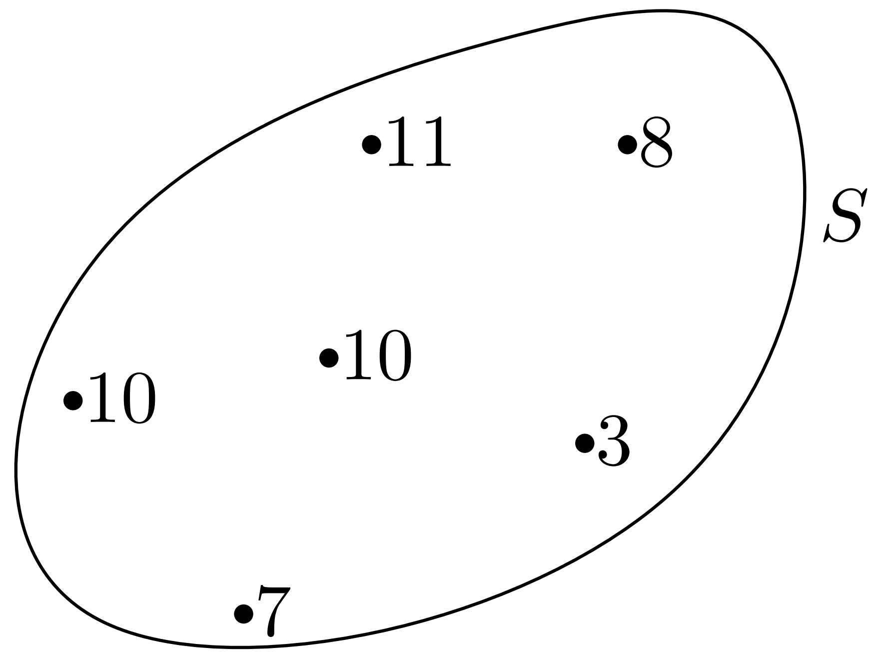
0	1	2	...					
3	10	7	11	8	10			

Extract-Max(S): $\Theta(n)$ tid.
Insert(S, k): $\Theta(1)$ tid.

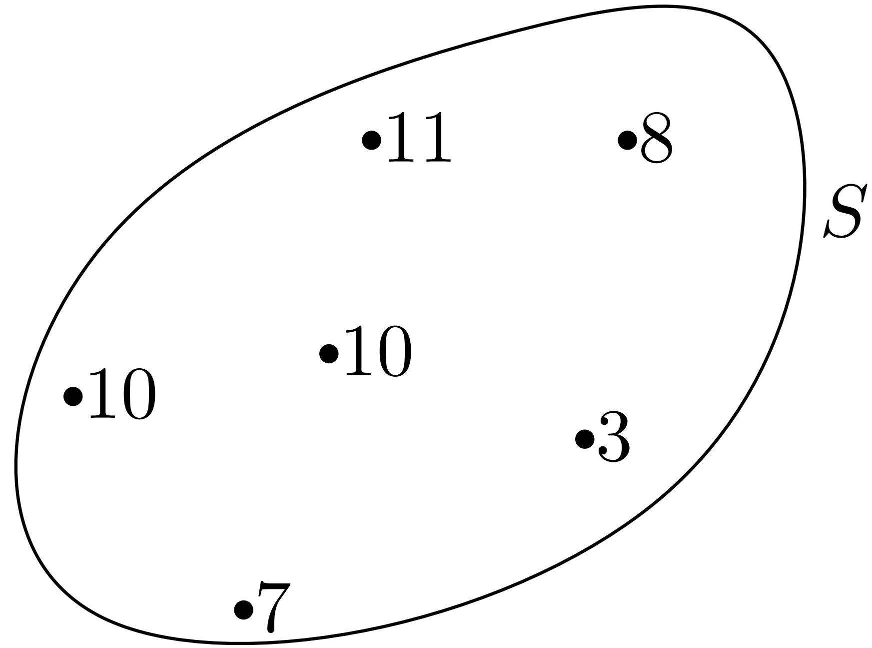
Sorteret array:

0	1	2	...					
3	7	8	10	10	11			

Extract-Max(S): $\Theta(1)$ tid.
Insert(S, k): $\Theta(n)$ tid.



Kendte teknikker



Array:

0	1	2	...					
3	10	7	11	8	10			

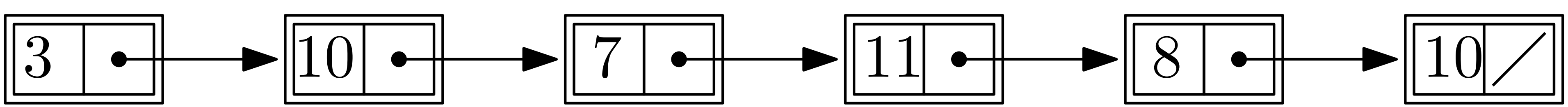
Extract-Max(S): $\Theta(n)$ tid.
Insert(S, k): $\Theta(1)$ tid.

Sorteret array:

0	1	2	...					
3	7	8	10	10	11			

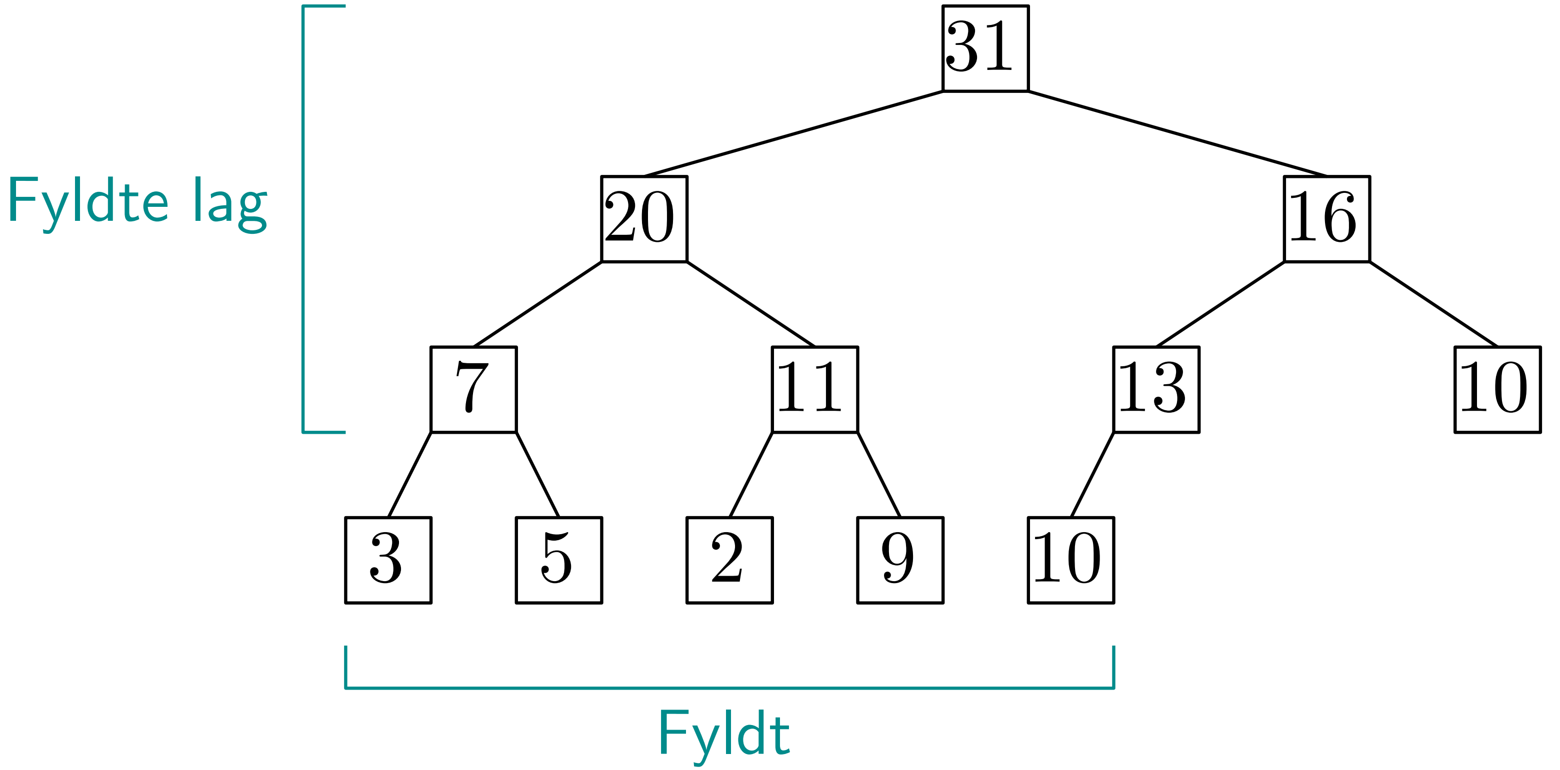
Extract-Max(S): $\Theta(1)$ tid.
Insert(S, k): $\Theta(n)$ tid.

Hægtet liste:

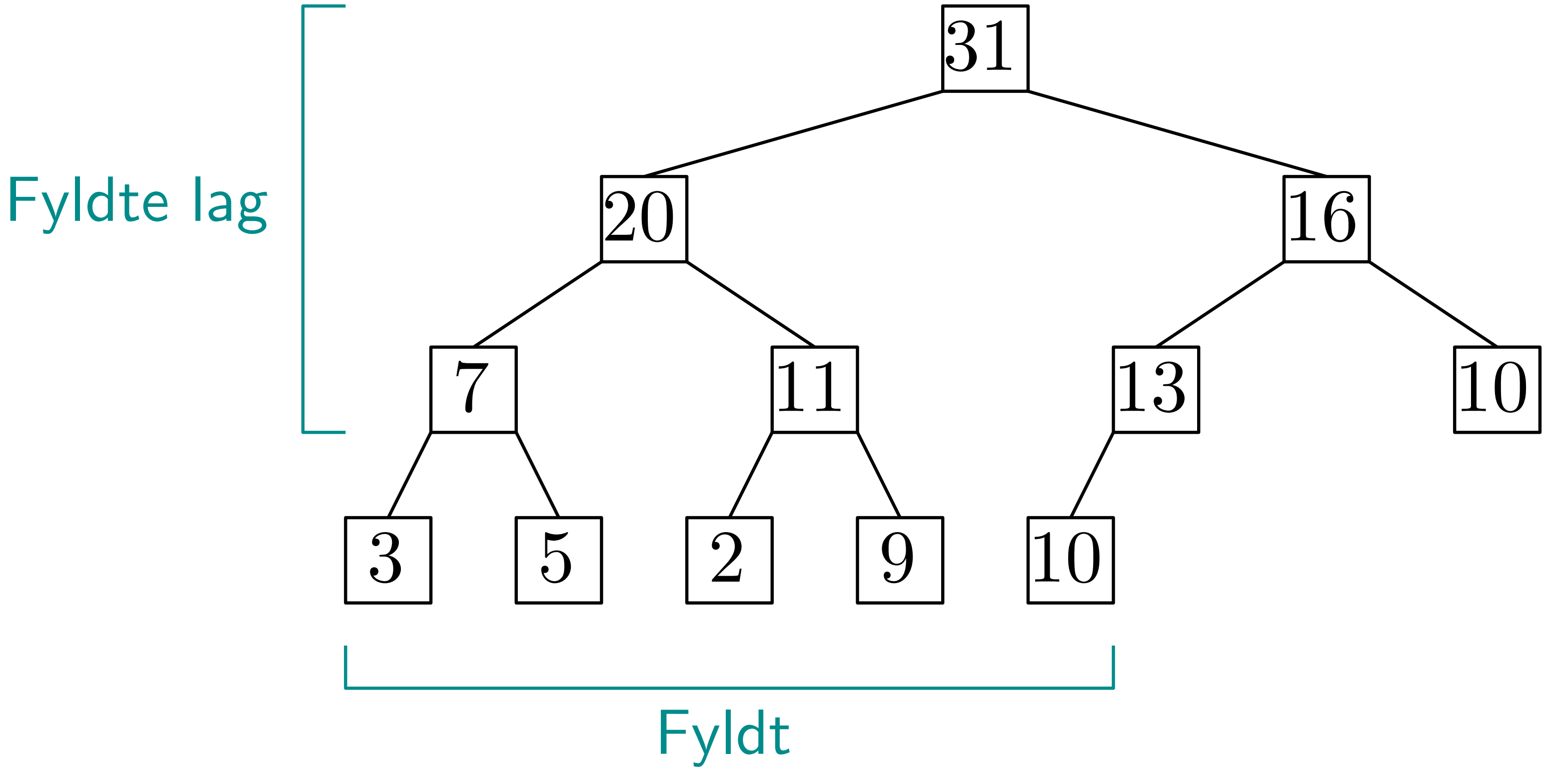


Extract-Max(S): $\Theta(n)$ tid.
Insert(S, k): $\Theta(1)$ tid.

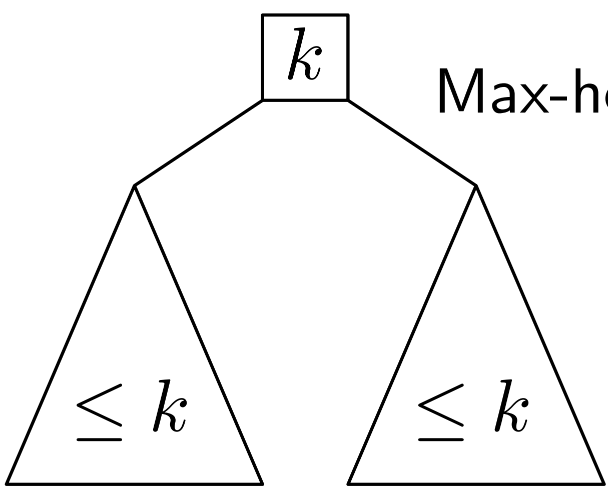
Hob



Hob

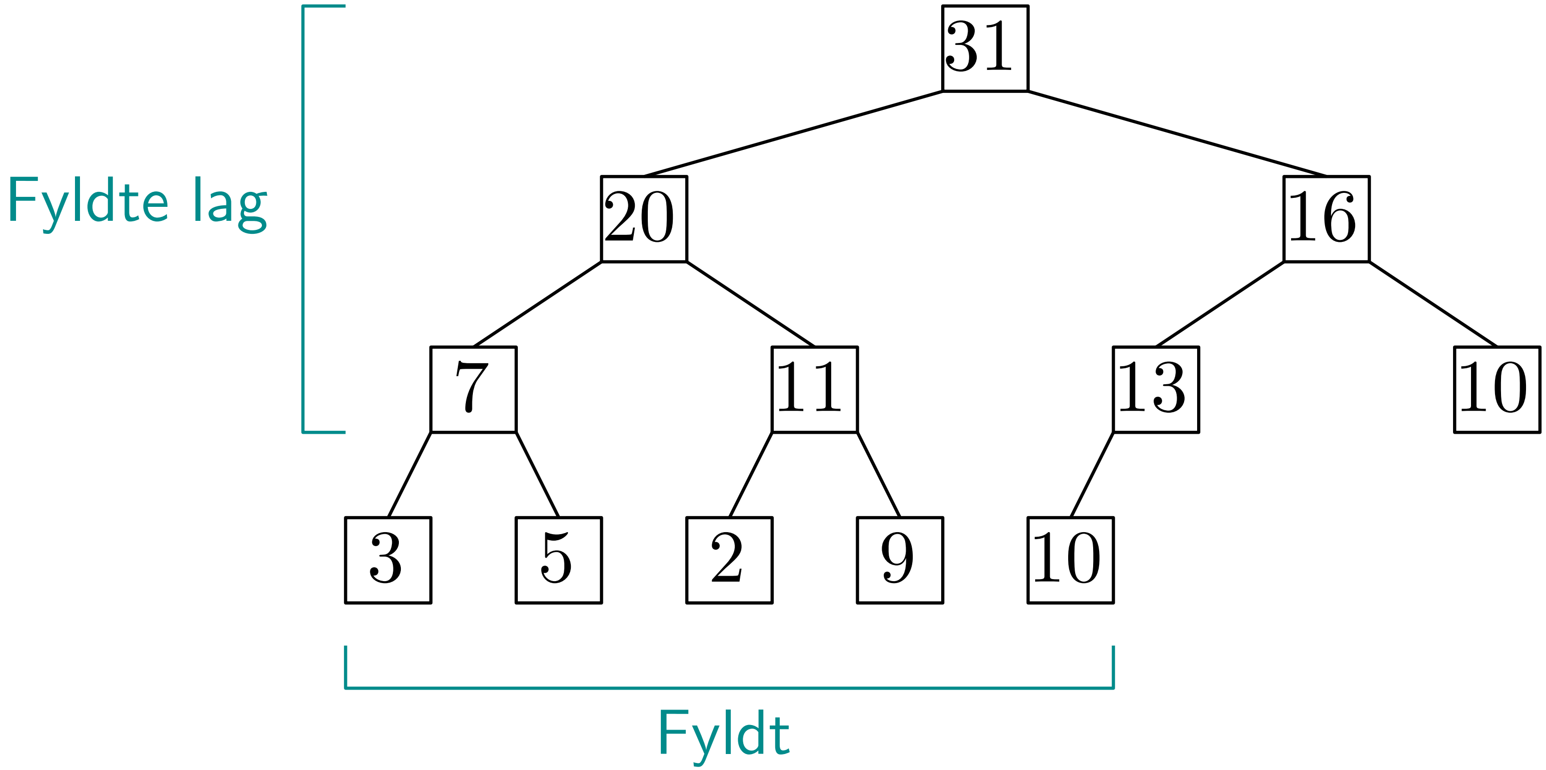


Hobeordenen:



Max-hob

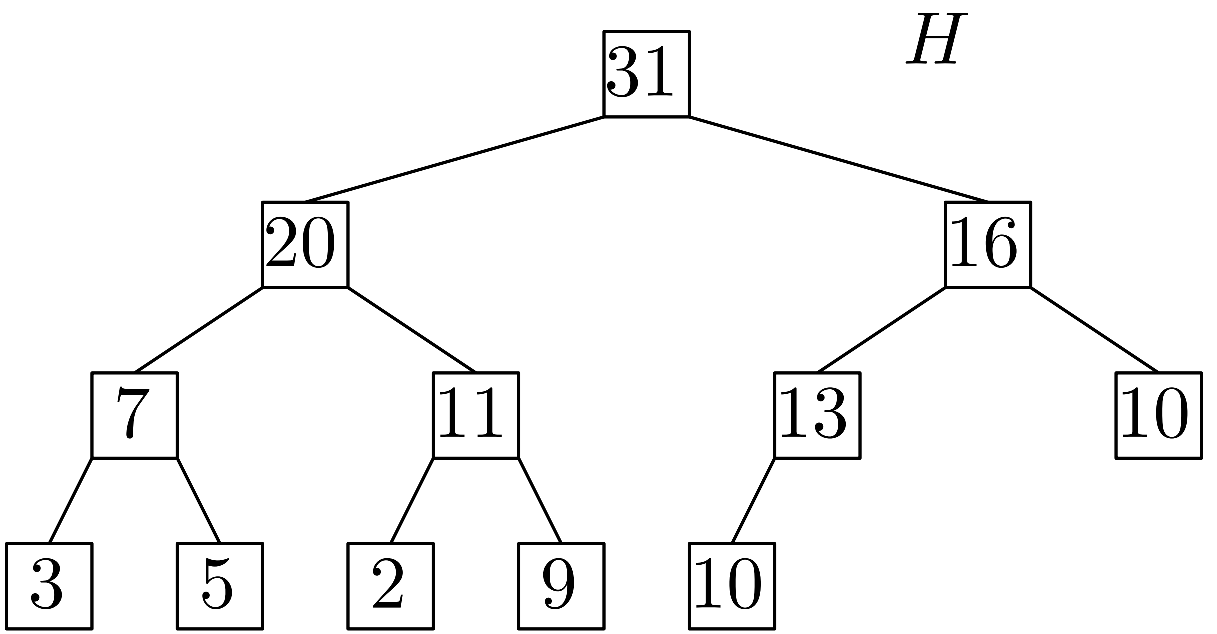
Hob



Hobeordenen:

Extract-Max

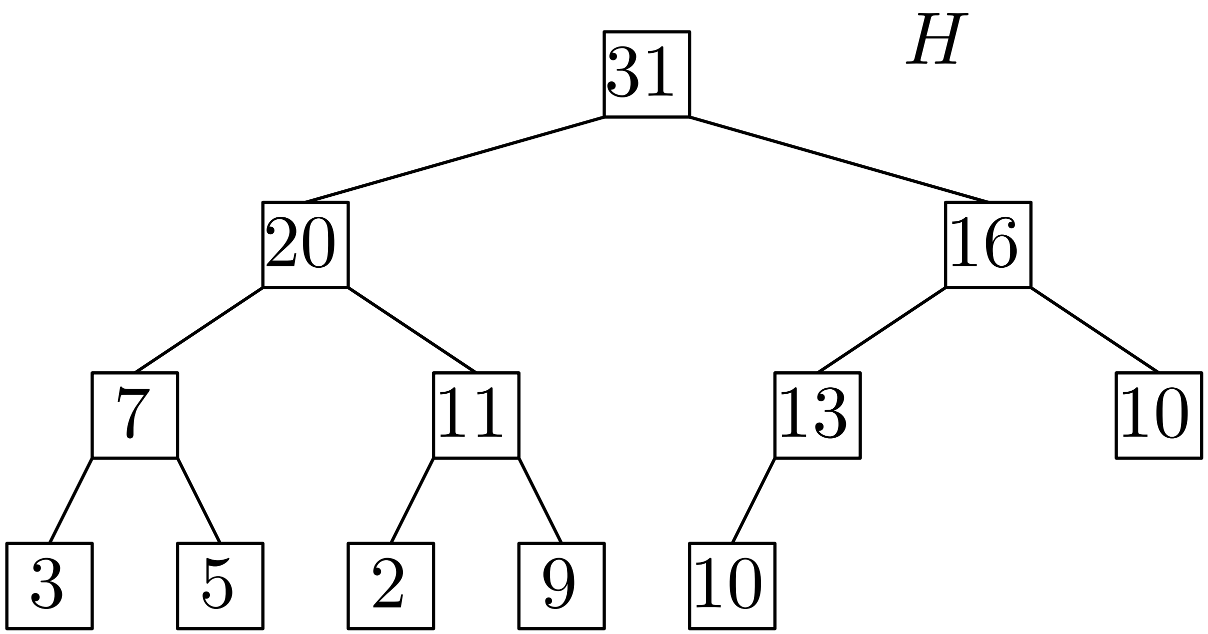
Extract-Max(H)



Extract-Max

Extract-Max(H)

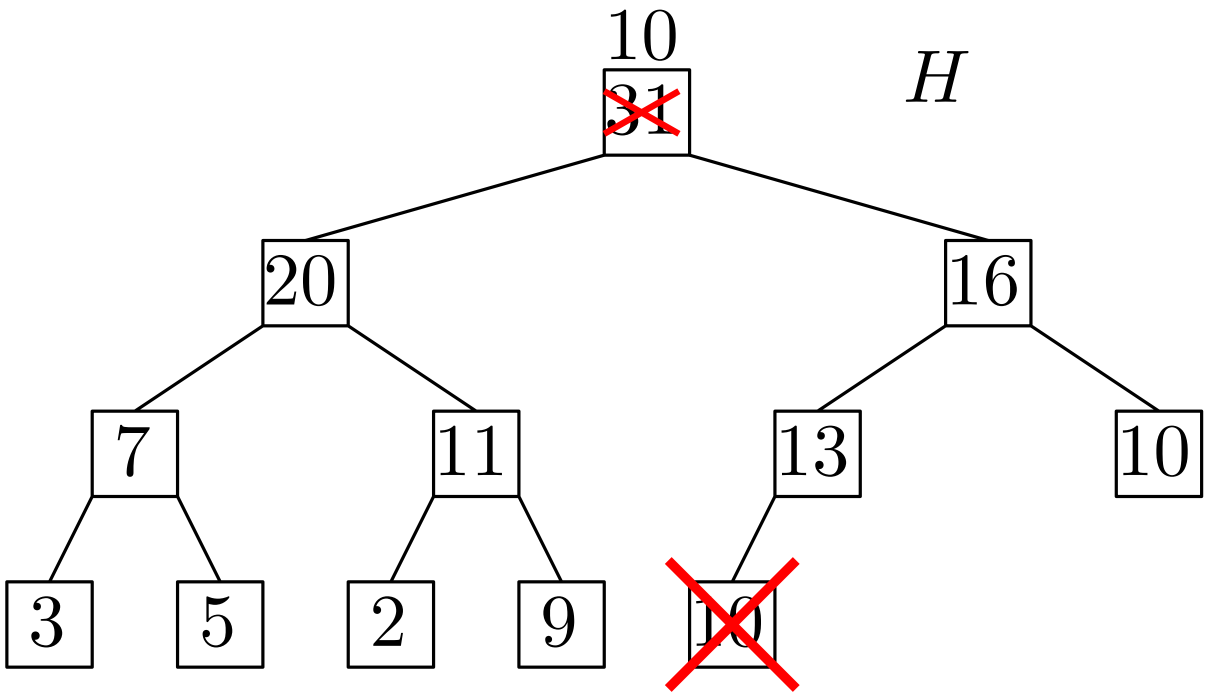
$max = 31$



Extract-Max

Extract-Max(H)

$max = 31$

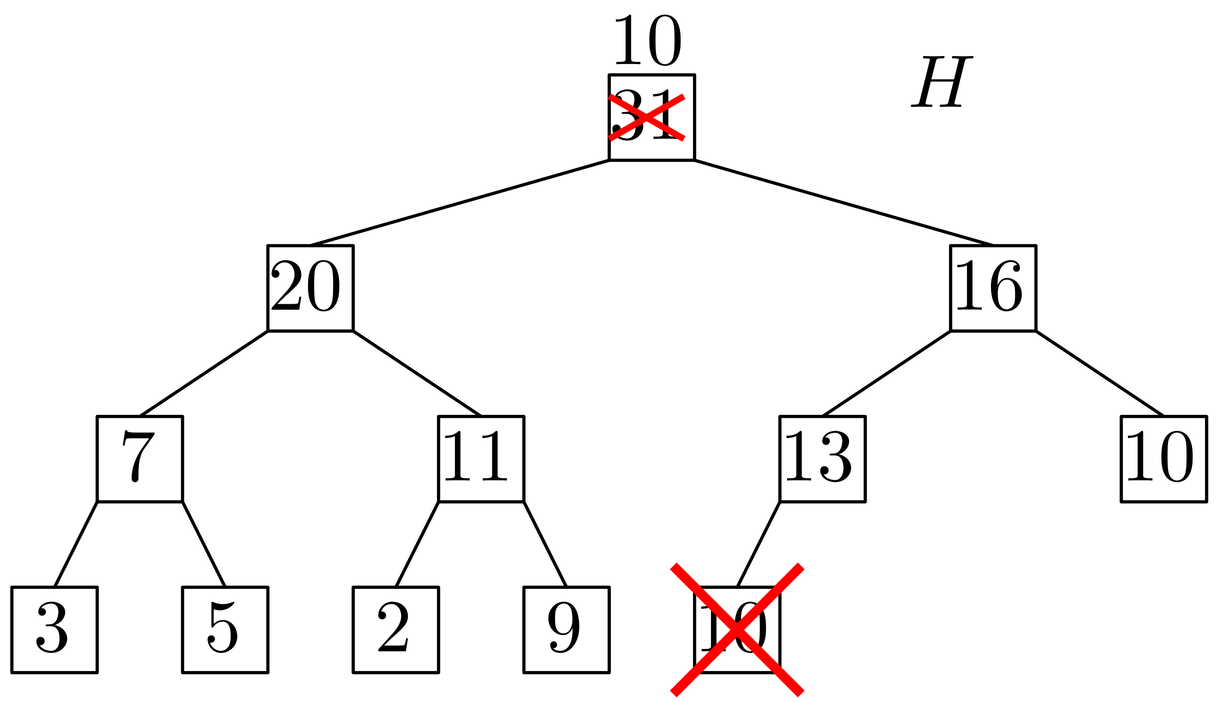


Extract-Max

Extract-Max(H)

$max = 31$

10 "bobler ned"
(Max-Heapify)

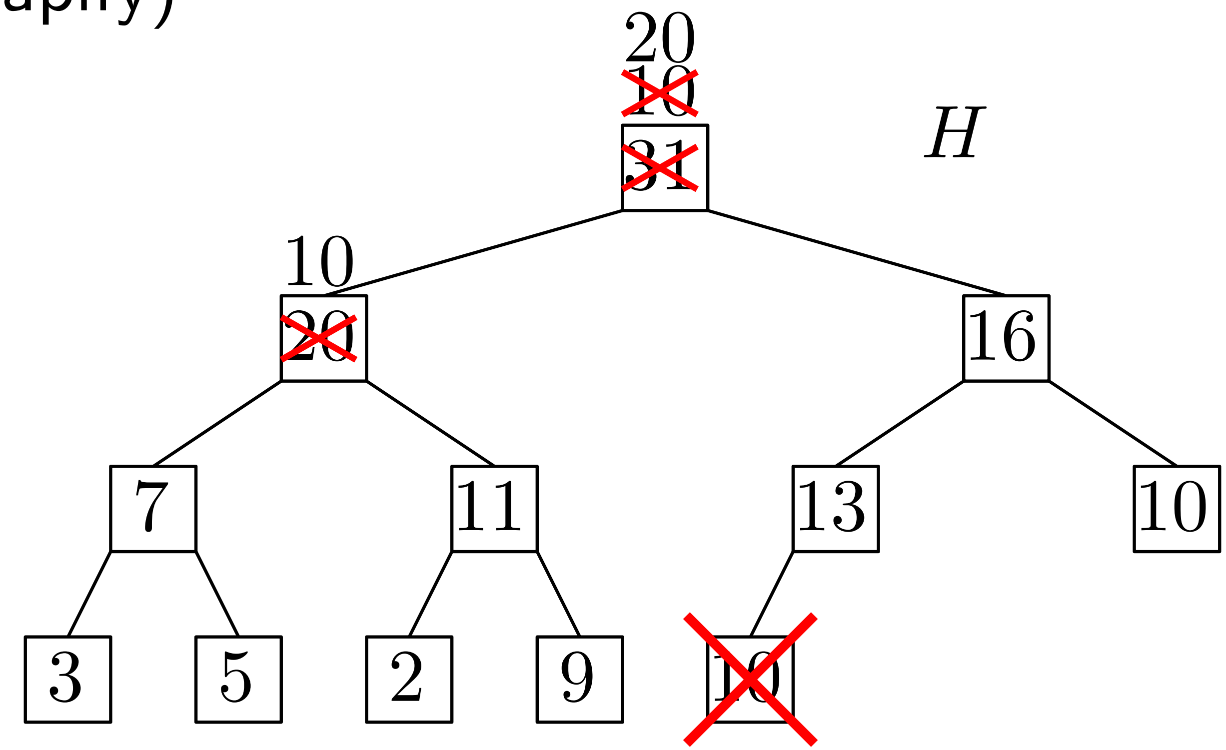


Extract-Max

Extract-Max(H)

$max = 31$

10 "bobler ned" (Max-Heapify)

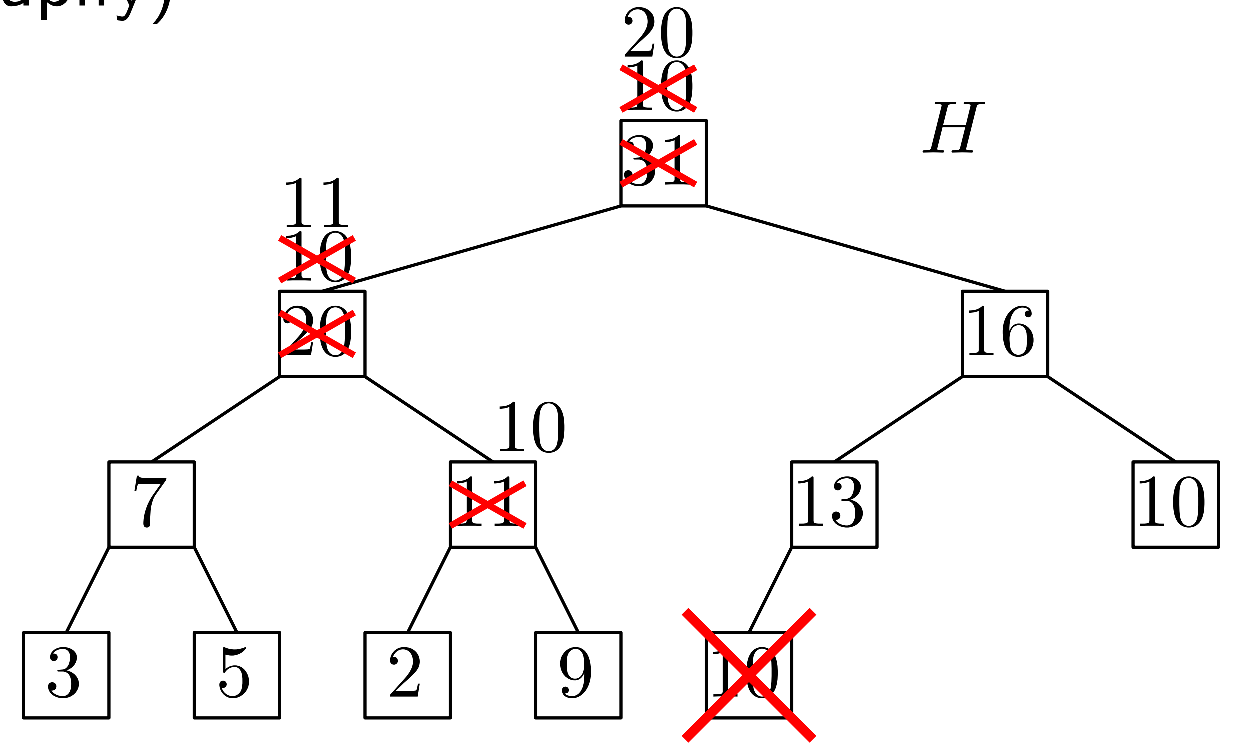


Extract-Max

Extract-Max(H)

$max = 31$

10 "bobler ned"
(Max-Heapify)



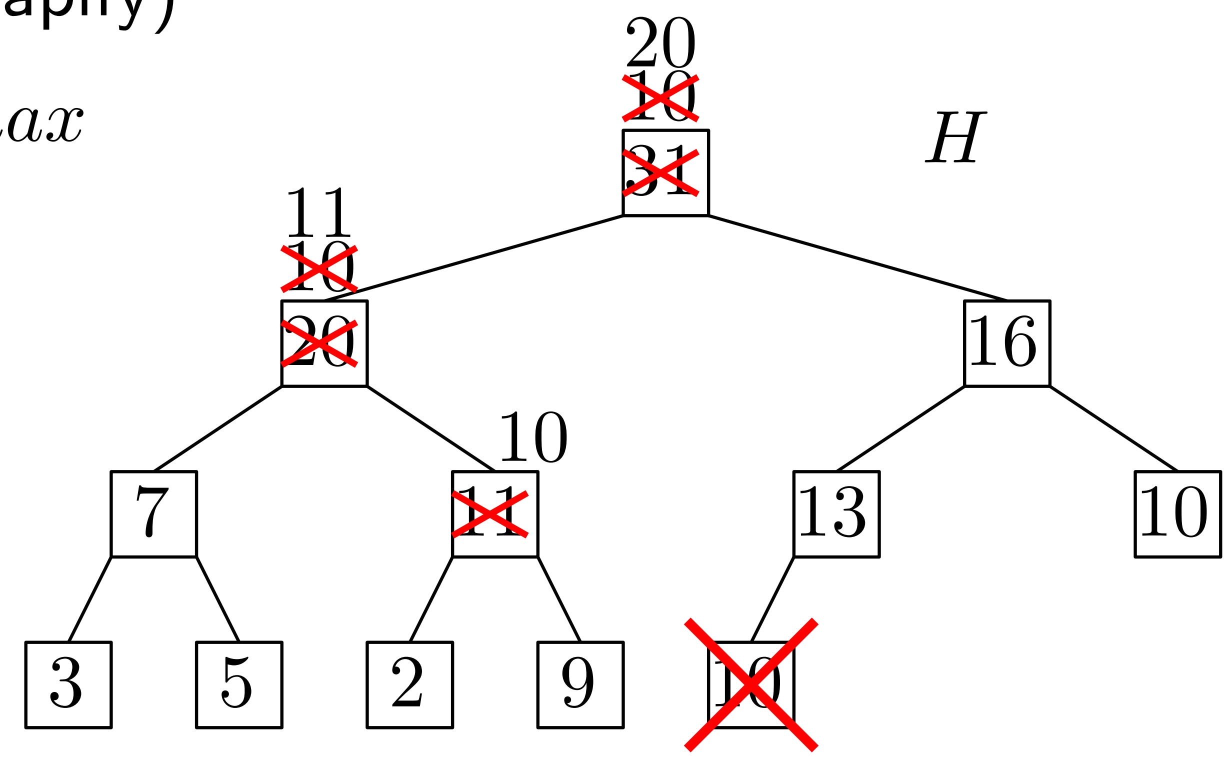
Extract-Max

Extract-Max(H)

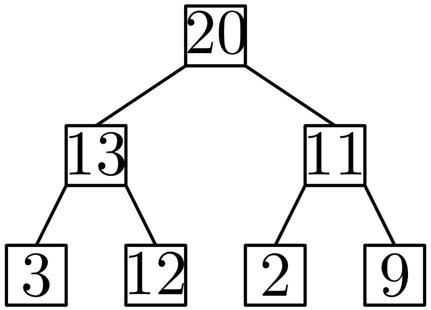
$max = 31$

10 “bobler ned” (Max-Heapify)

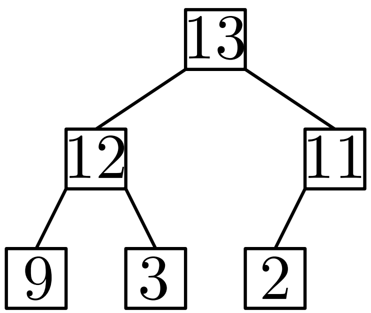
return max



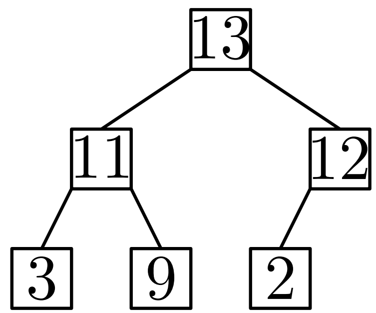
Hvordan ser hoben ud til sidst?



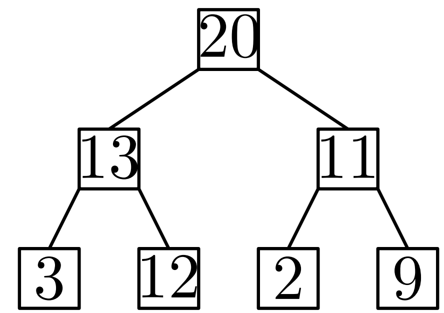
Extract-Max(H)



A

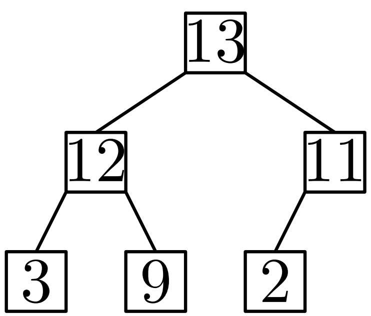


B

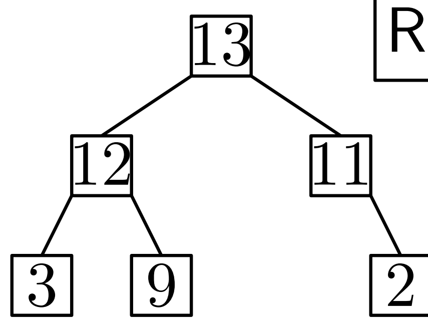


C

socrative.com $\longrightarrow$ Student login,
Room name: ABRAHAMSEN3464



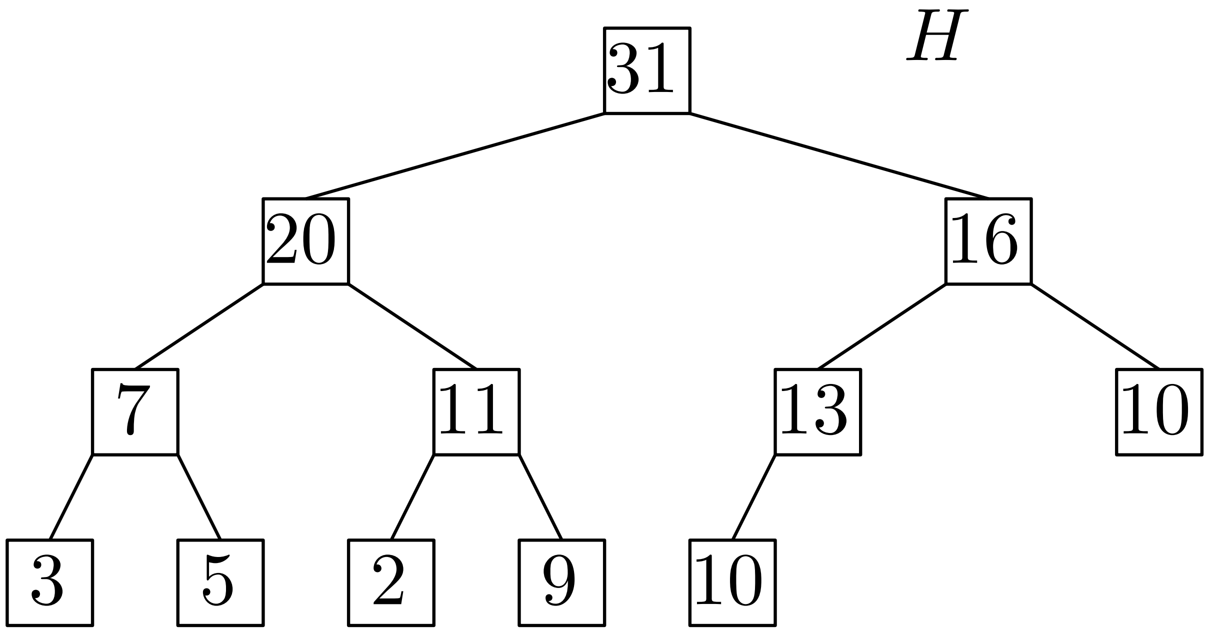
D



E

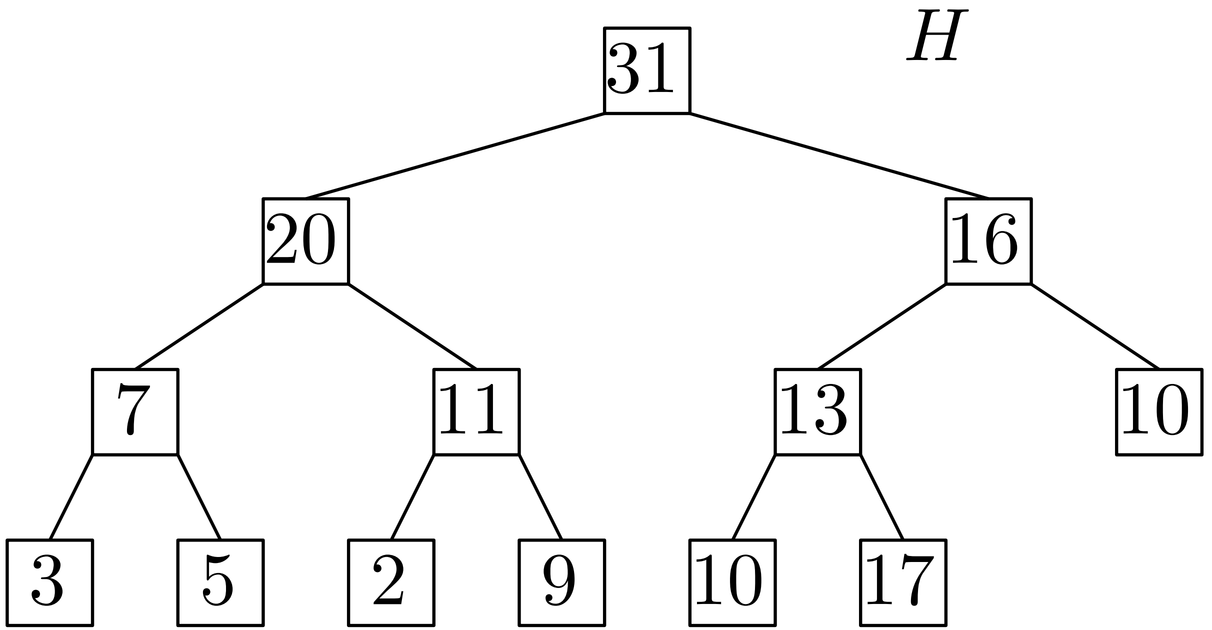
Insert

Insert$(H, 17)$



Insert

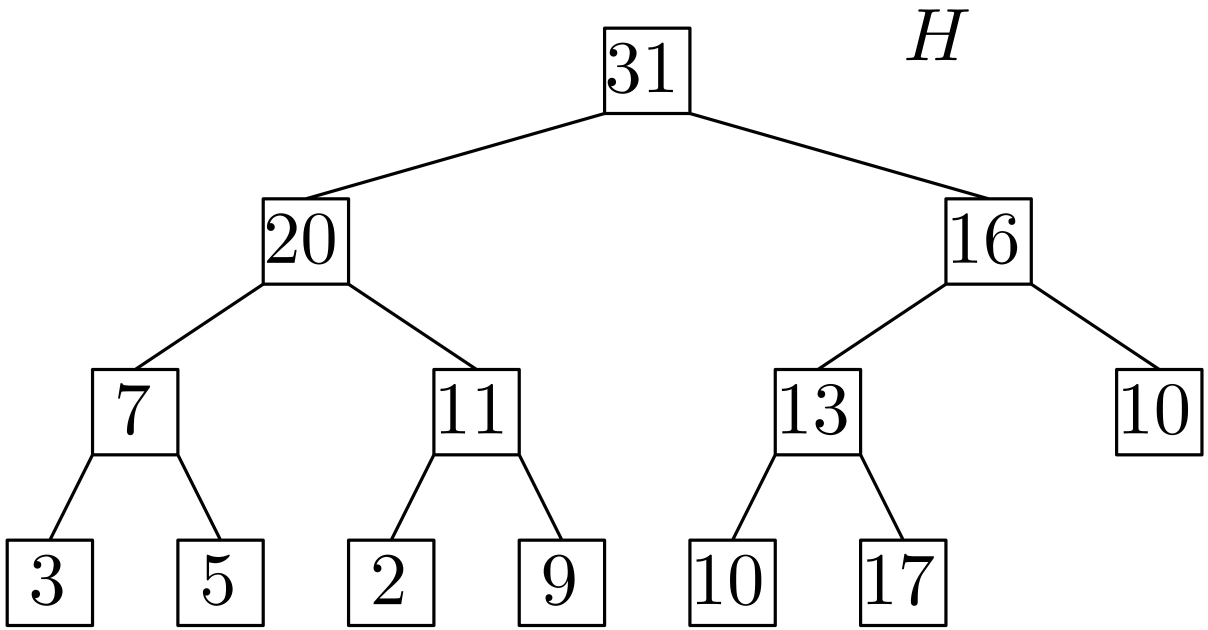
Insert$(H, 17)$



Insert

Insert$(H, 17)$

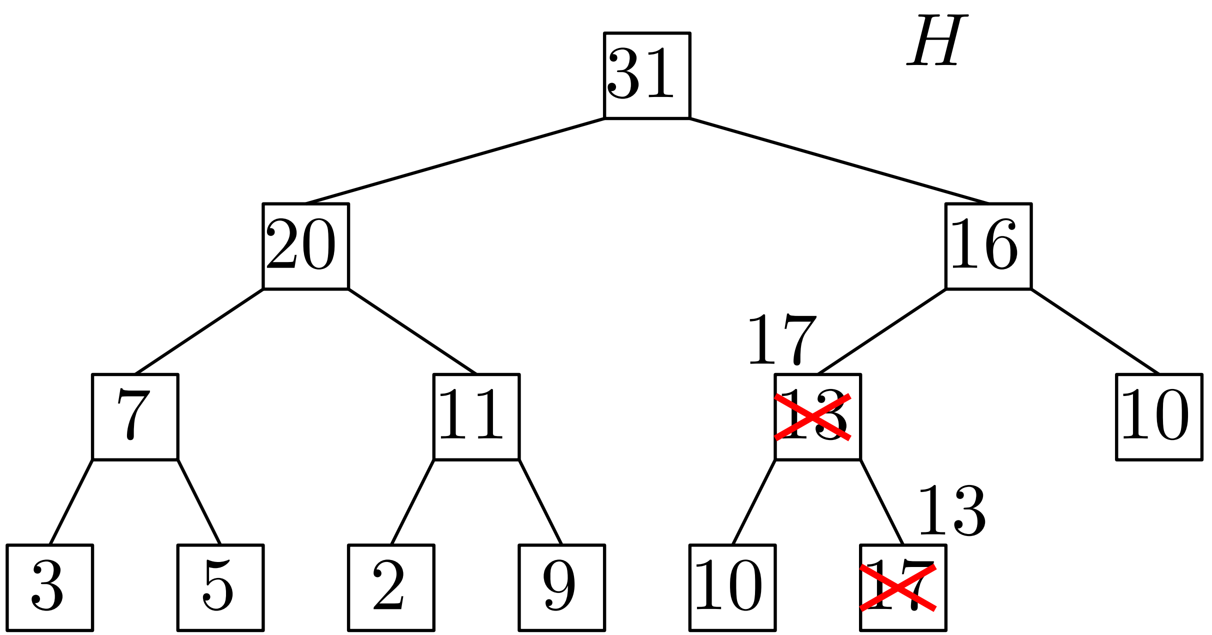
17 "bobler op"



Insert

Insert$(H, 17)$

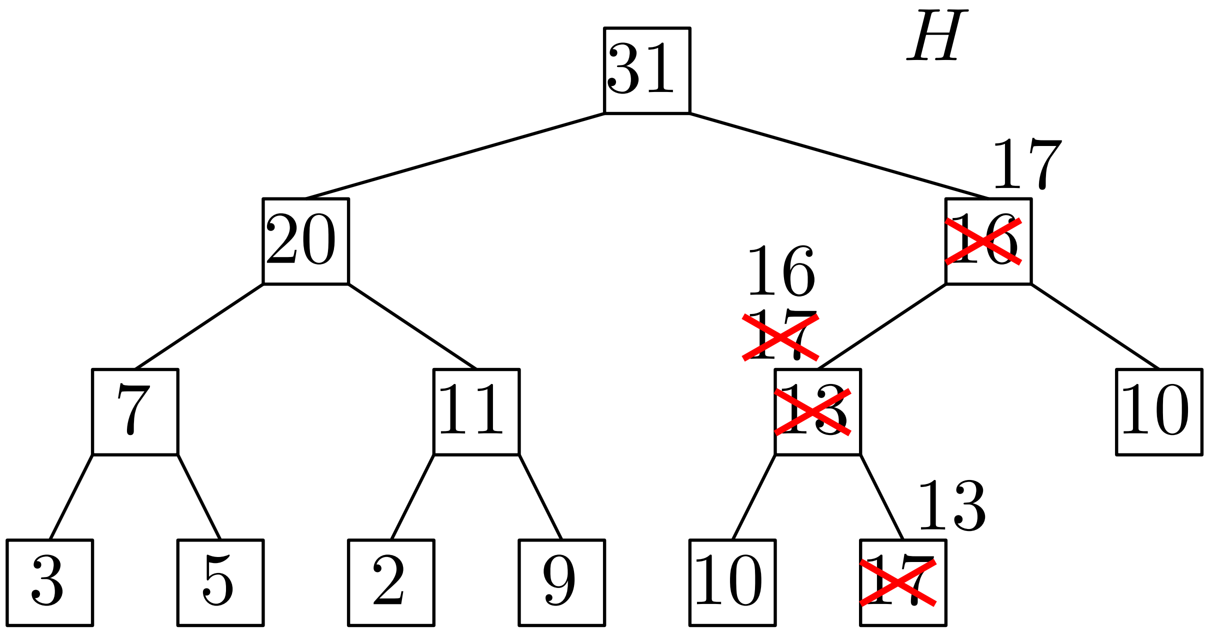
17 “bobler op”



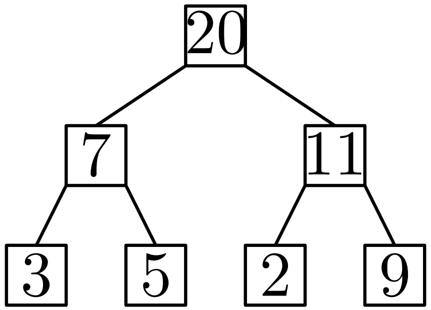
Insert

Insert$(H, 17)$

17 "bobler op"

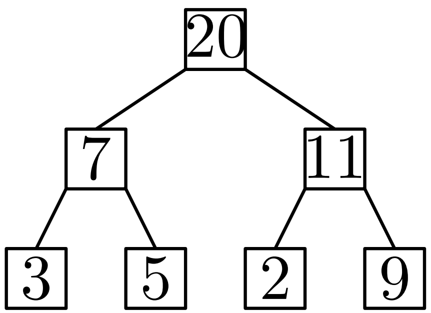


Hvordan ser hoben ud til sidst?

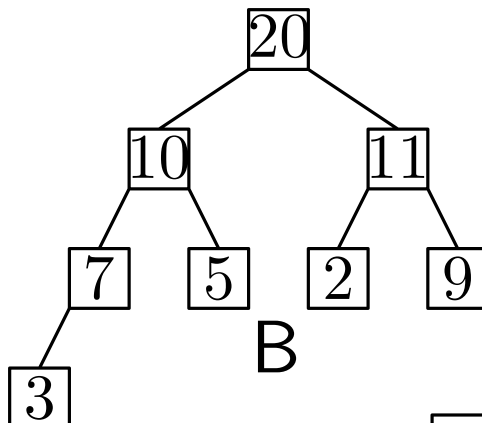


Insert($H, 10$)

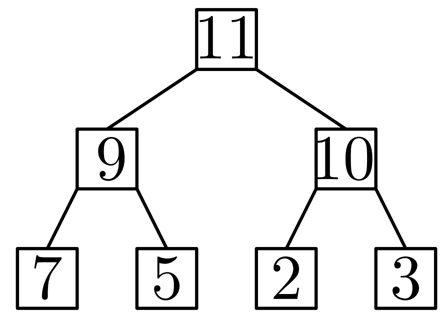
Extract-Max(H)



A

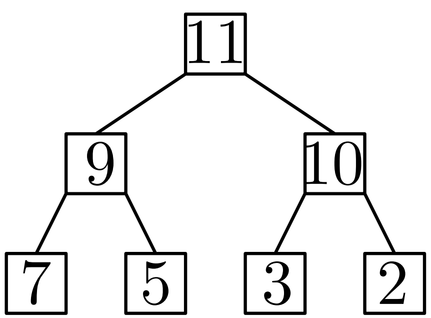


B

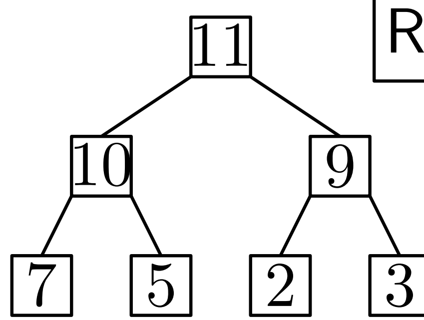


C

socrative.com $\longrightarrow$ Student login,
Room name: ABRAHAMSEN3464



D



E

Køretider

Insert: Boble op

Extract-Max: Boble ned

Køretider

Insert: Boble op

Extract-Max: Boble ned

I begge tilfælde er $T(n) = \Theta(h)$, hvor h er højden af hoben.

Køretider

Insert: Boble op

Extract-Max: Boble ned

I begge tilfælde er $T(n) = \Theta(h)$, hvor h er højden af hoben.

$$n = 2^0 = 1$$
$$h = 0$$

Køretider

Insert: Boble op

Extract-Max: Boble ned

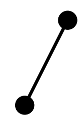
I begge tilfælde er $T(n) = \Theta(h)$, hvor h er højden af hoben.

$n = 2^0 = 1$

$h = 0$

$n = 2^1 = 2$

$h = 1$



Køretider

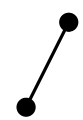
Insert: Boble op

Extract-Max: Boble ned

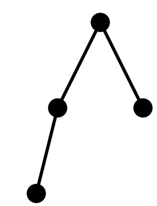
I begge tilfælde er $T(n) = \Theta(h)$, hvor h er højden af hoben.

$n = 2^0 = 1$
$h = 0$

$n = 2^1 = 2$
$h = 1$



$n = 2^2 = 4$
$h = 2$



Køretider

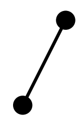
Insert: Boble op

Extract-Max: Boble ned

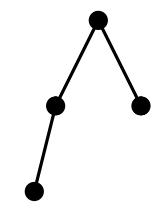
I begge tilfælde er $T(n) = \Theta(h)$, hvor h er højden af hoben.

$n = 2^0 = 1$
$h = 0$

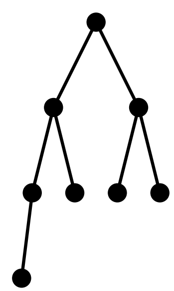
$n = 2^1 = 2$
$h = 1$



$n = 2^2 = 4$
$h = 2$



$n = 2^3 = 8$
$h = 3$



Køretider

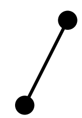
Insert: Boble op

Extract-Max: Boble ned

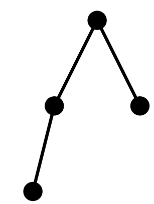
I begge tilfælde er $T(n) = \Theta(h)$, hvor h er højden af hoben.

$n = 2^0 = 1$
$h = 0$

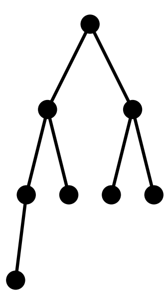
$n = 2^1 = 2$
$h = 1$



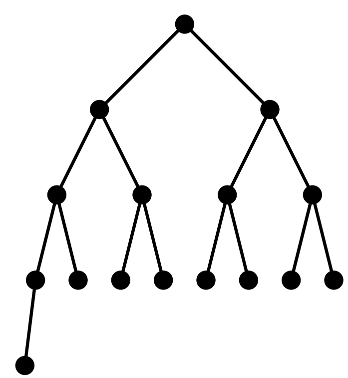
$n = 2^2 = 4$
$h = 2$



$n = 2^3 = 8$
$h = 3$



$n = 2^4 = 16$
$h = 4$



Køretider

Insert: Boble op

Extract-Max: Boble ned

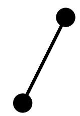
I begge tilfælde er $T(n) = \Theta(h)$, hvor h er højden af hoben.

$n = 2^0 = 1$

$h = 0$

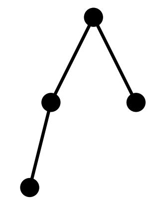
$n = 2^1 = 2$

$h = 1$



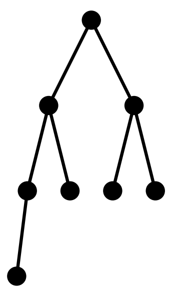
$n = 2^2 = 4$

$h = 2$



$n = 2^3 = 8$

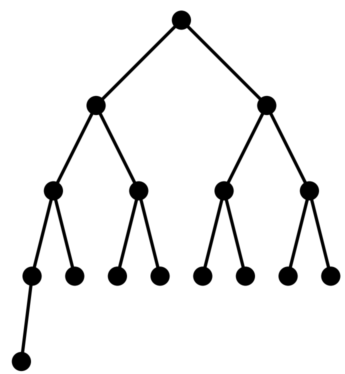
$h = 3$



$1 + 1 + 2 + 4 + \ldots + 2^{h-1}$

$n = 2^4 = 16$

$h = 4$



Køretider

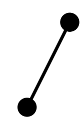
Insert: Boble op

Extract-Max: Boble ned

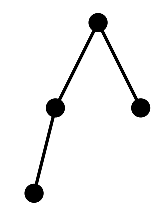
I begge tilfælde er $T(n) = \Theta(h)$, hvor h er højden af hoben.

$n = 2^0 = 1$
$h = 0$

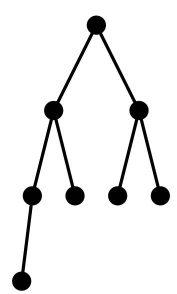
$n = 2^1 = 2$
$h = 1$



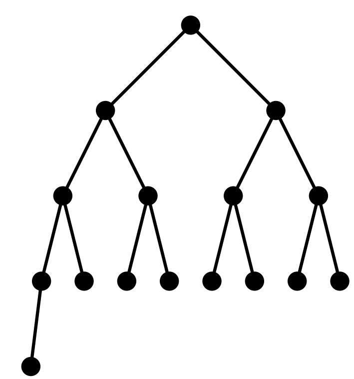
$n = 2^2 = 4$
$h = 2$



$n = 2^3 = 8$
$h = 3$



$n = 2^4 = 16$
$h = 4$



$$\underbrace{1 + 1}_{2} + 2 + 4 + \ldots + 2^{h-1}$$

Køretider

Insert: Boble op

Extract-Max: Boble ned

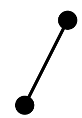
I begge tilfælde er $T(n) = \Theta(h)$, hvor h er højden af hoben.

$n = 2^0 = 1$

$h = 0$

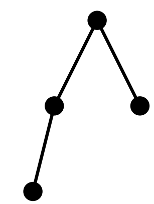
$n = 2^1 = 2$

$h = 1$



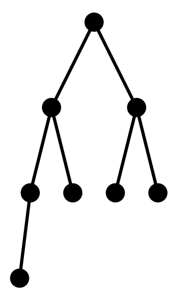
$n = 2^2 = 4$

$h = 2$



$n = 2^3 = 8$

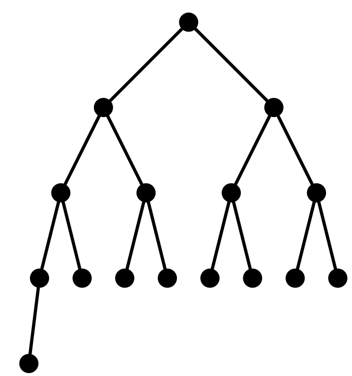
$h = 3$



$$\underbrace{\underbrace{1 + 1}_{2} + 2}_{4} + 4 + \ldots + 2^{h-1}$$

$n = 2^4 = 16$

$h = 4$



Køretider

Insert: Boble op

Extract-Max: Boble ned

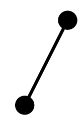
I begge tilfælde er $T(n) = \Theta(h)$, hvor h er højden af hoben.

$n = 2^0 = 1$

$h = 0$

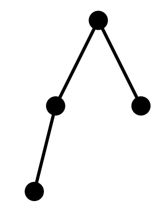
$n = 2^1 = 2$

$h = 1$



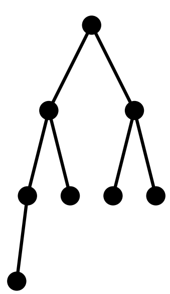
$n = 2^2 = 4$

$h = 2$



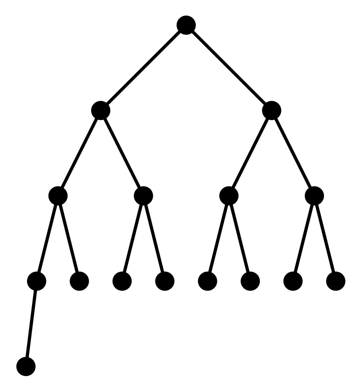
$n = 2^3 = 8$

$h = 3$



$n = 2^4 = 16$

$h = 4$



$$\underbrace{\underbrace{\underbrace{1+1}_{2}+2}_{4}+4}_{8}+\ldots+2^{h-1}$$

Køretider

Insert: Boble op

Extract-Max: Boble ned

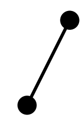
I begge tilfælde er $T(n) = \Theta(h)$, hvor h er højden af hoben.

$n = 2^0 = 1$

$h = 0$

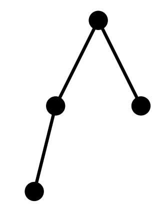
$n = 2^1 = 2$

$h = 1$



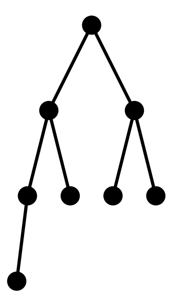
$n = 2^2 = 4$

$h = 2$



$n = 2^3 = 8$

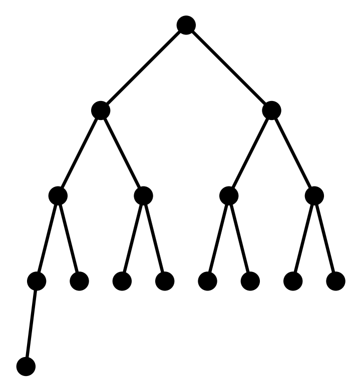
$h = 3$



$$\underbrace{\underbrace{\underbrace{\underbrace{1 + 1}_{2} + 2}_{4} + 4}_{8} + \ldots + 2^{h-1}}_{2^h}$$

$n = 2^4 = 16$

$h = 4$



Køretider

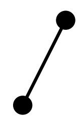
Insert: Boble op

Extract-Max: Boble ned

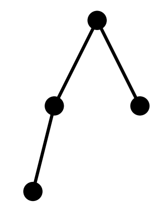
I begge tilfælde er $T(n) = \Theta(h)$, hvor h er højden af hoben.

$n = 2^0 = 1$
$h = 0$

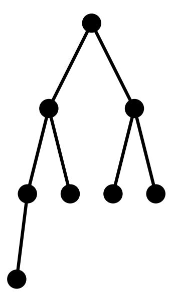
$n = 2^1 = 2$
$h = 1$



$n = 2^2 = 4$
$h = 2$

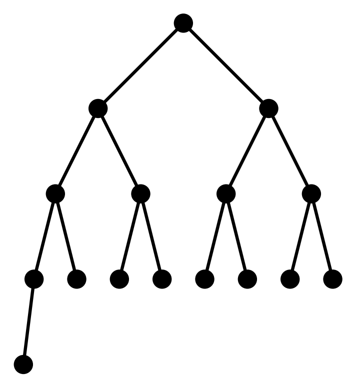


$n = 2^3 = 8$
$h = 3$



$$\underbrace{\underbrace{\underbrace{\underbrace{1 + 1}_{2} + 2}_{4} + 4}_{8} + \ldots + 2^{h-1}}_{2^h}$$

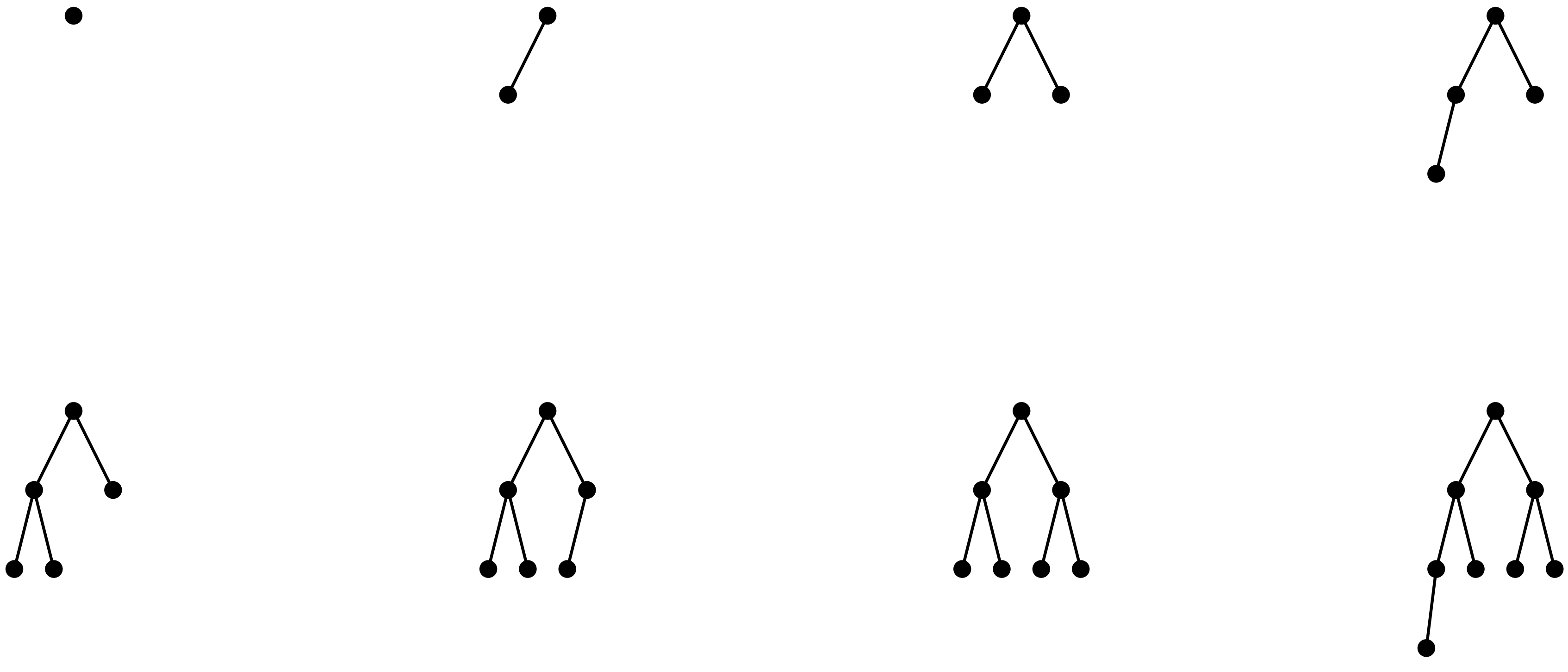
$n = 2^4 = 16$
$h = 4$



$h = \lfloor \lg n \rfloor$, så
$T(n) = \Theta(\log n)$.

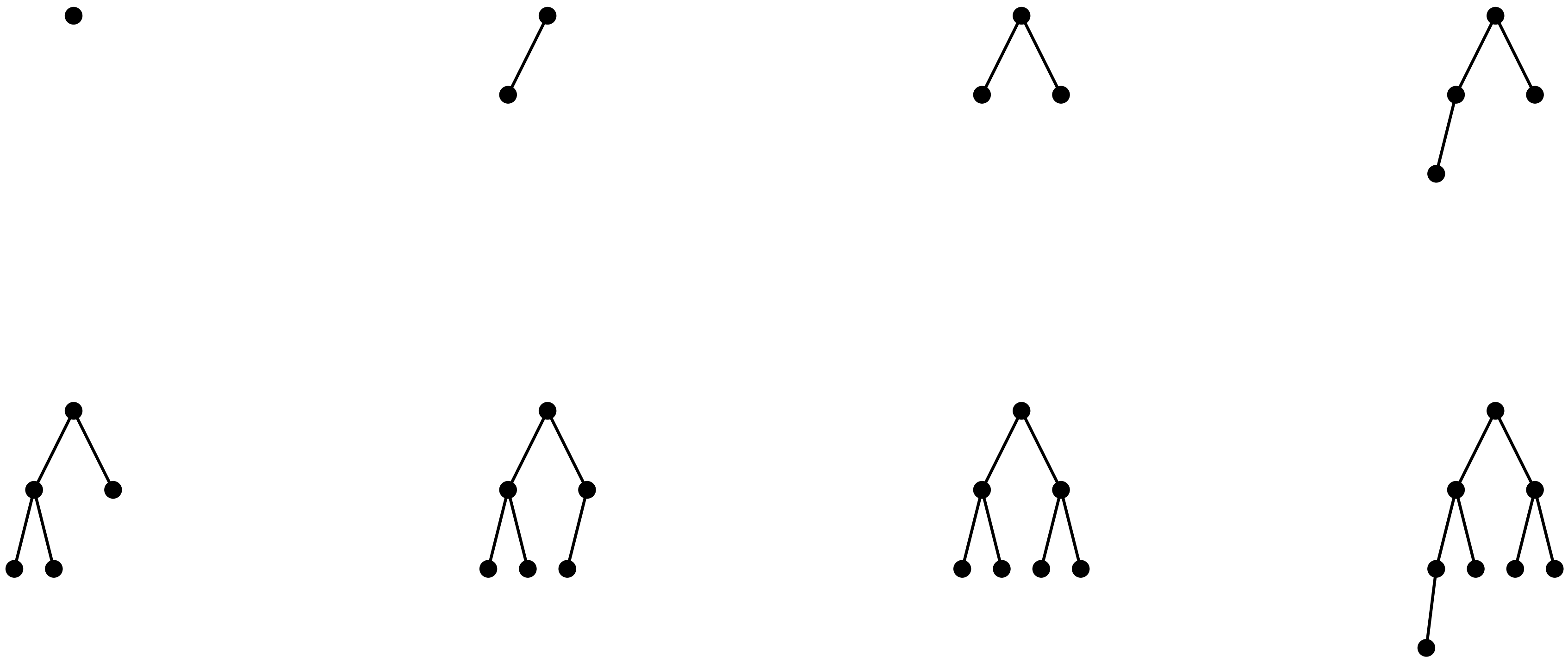
Hvor mange blade er der?

Blad: knude uden børn



Hvor mange blade er der?

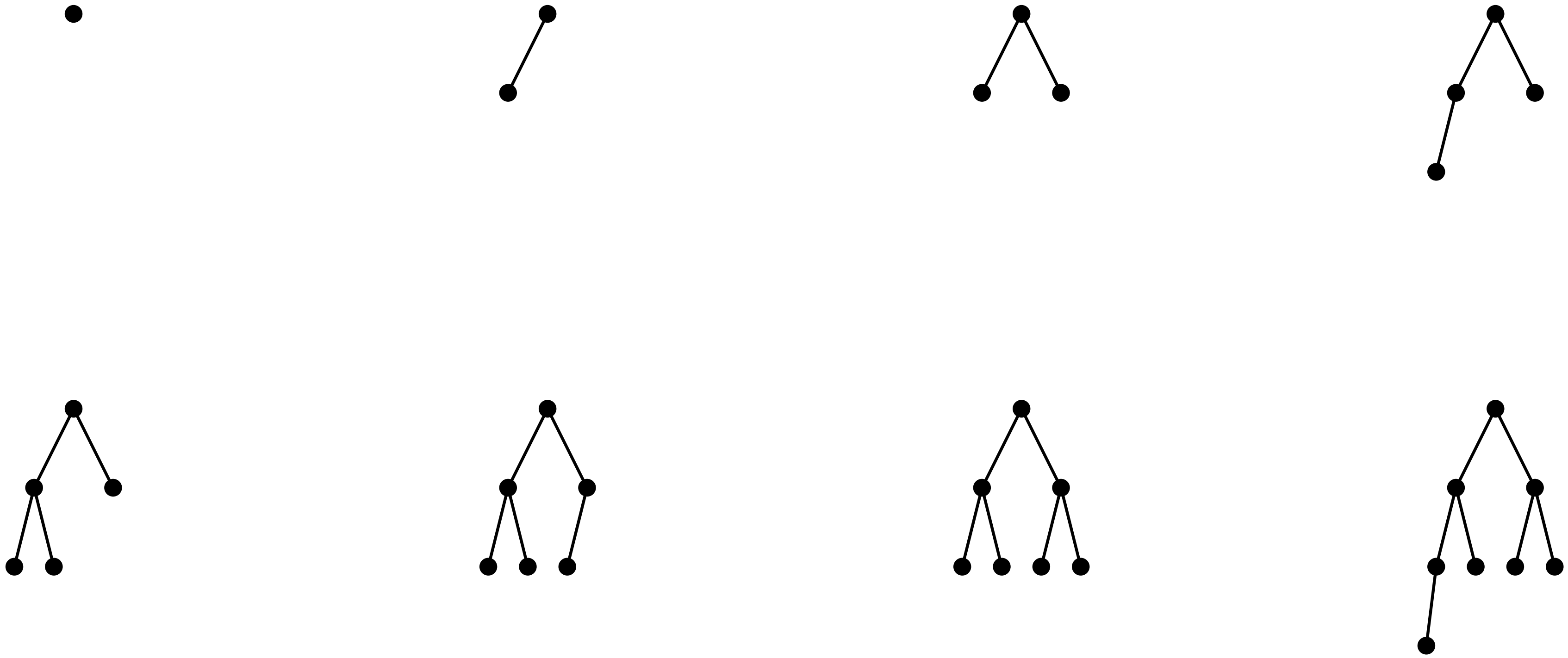
Blad: knude uden børn



Hver anden gang vi tilføjer en knude er antallet uændret, de andre gange vokser det med én. Derfor: $\#\text{blade} = \lceil \frac{n}{2} \rceil$.

Hvor mange blade er der?

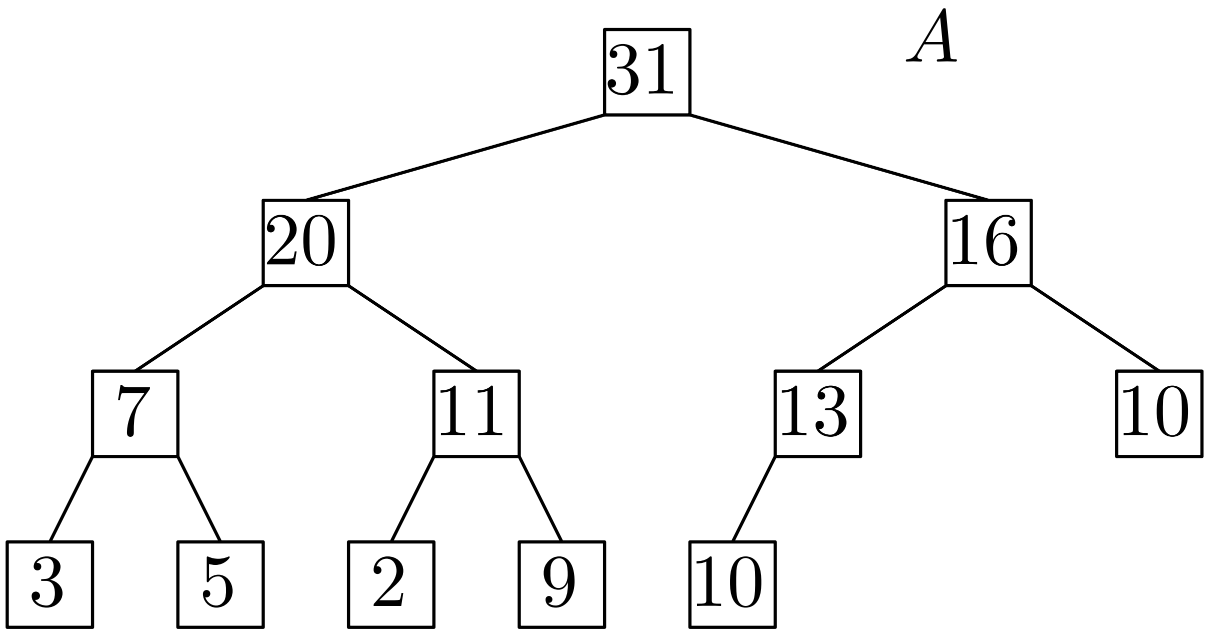
Blad: knude uden børn



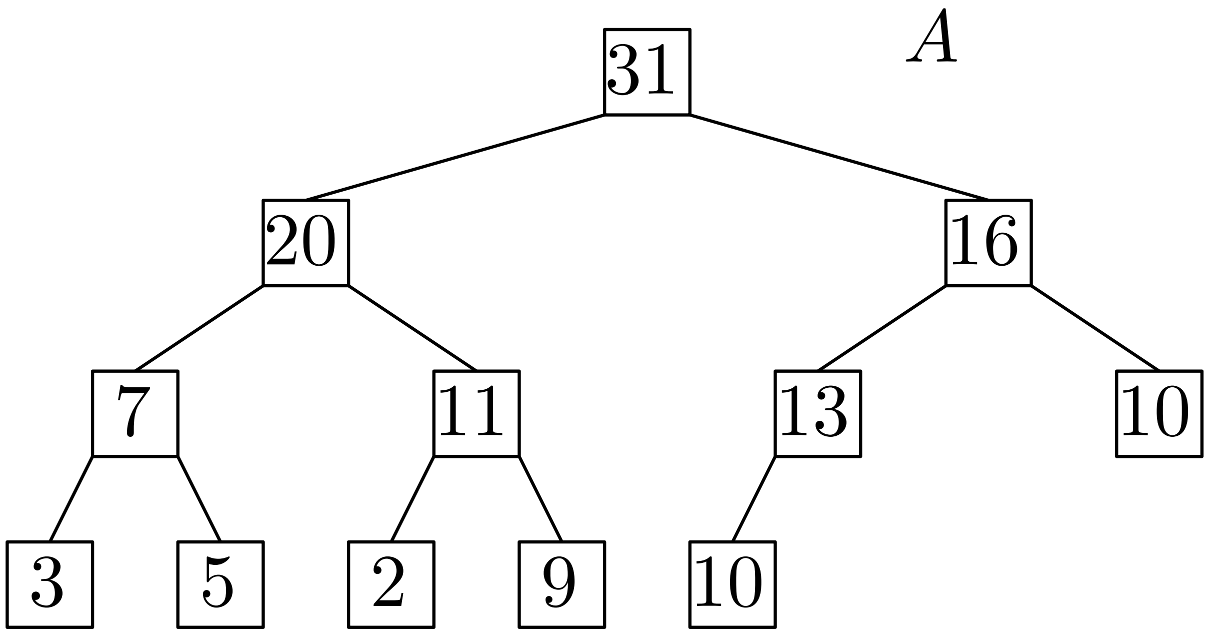
Hver anden gang vi tilføjer en knude er antallet uændret, de andre gange vokser det med én. Derfor: $\#\text{blade} = \lceil \frac{n}{2} \rceil$.

Højde $= \lfloor \lg n \rfloor$
Blade $= \lceil \frac{n}{2} \rceil$

Repræsentation med array

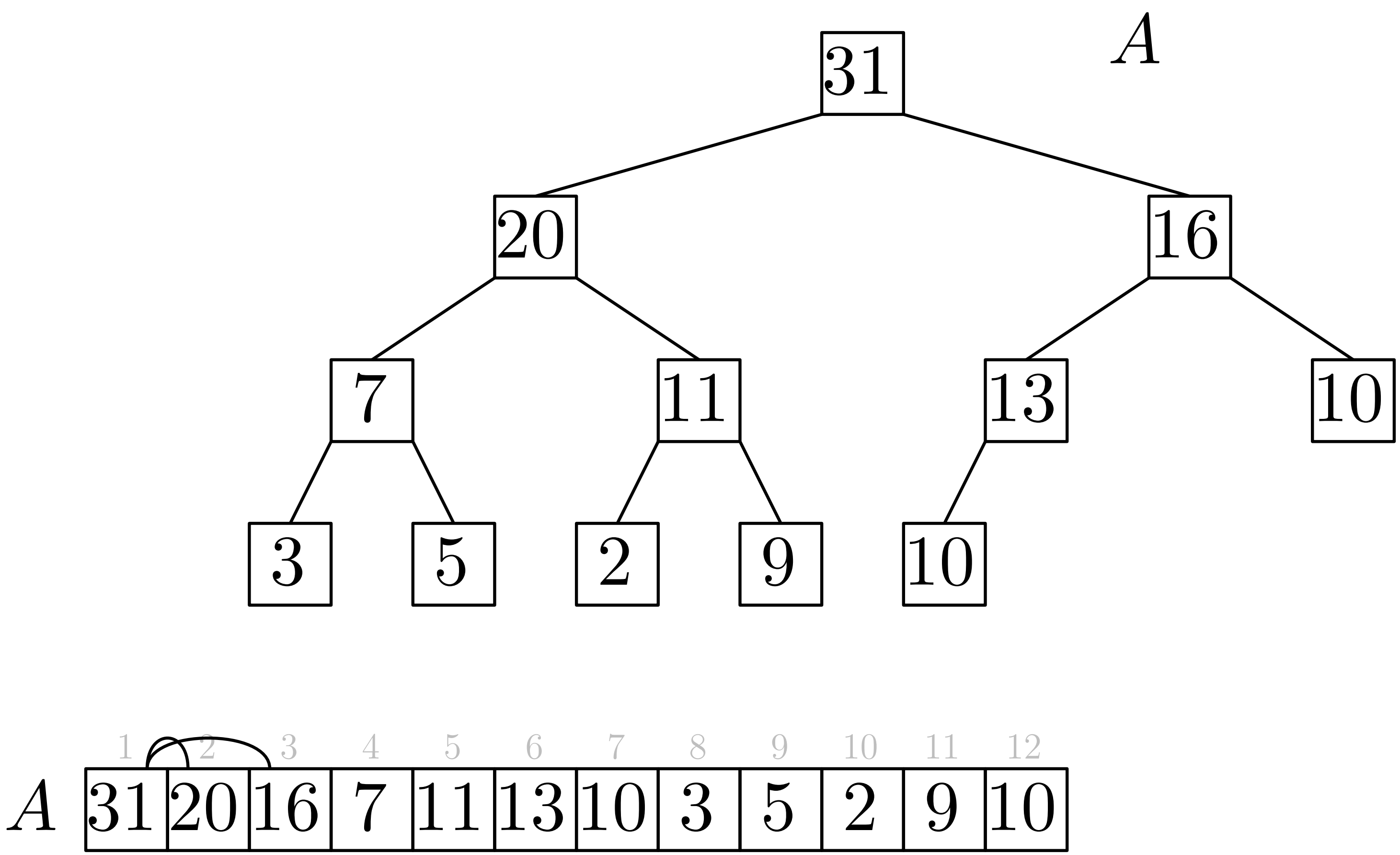


Repræsentation med array

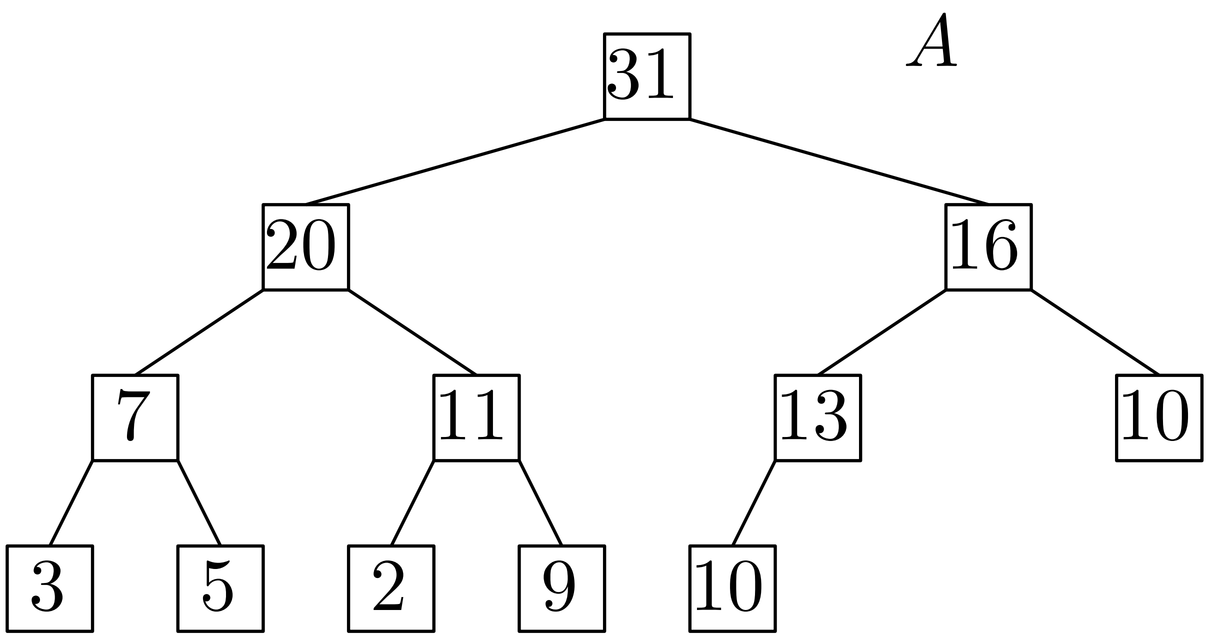


	1	2	3	4	5	6	7	8	9	10	11	12
A	31	20	16	7	11	13	10	3	5	2	9	10

Repræsentation med array

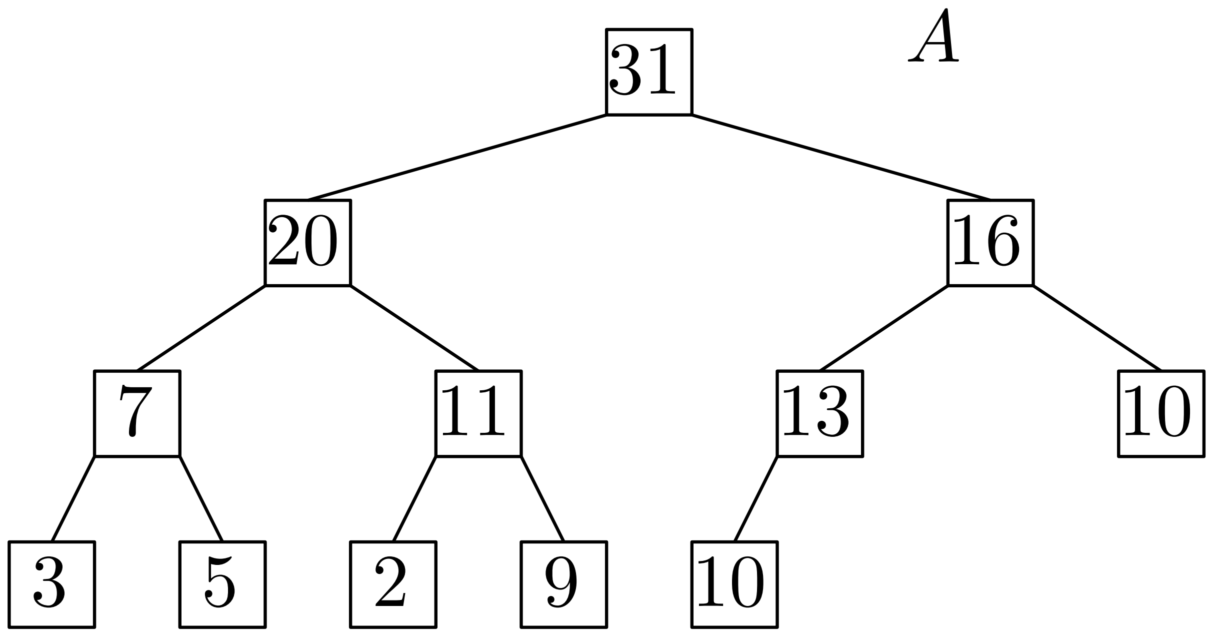


Repræsentation med array



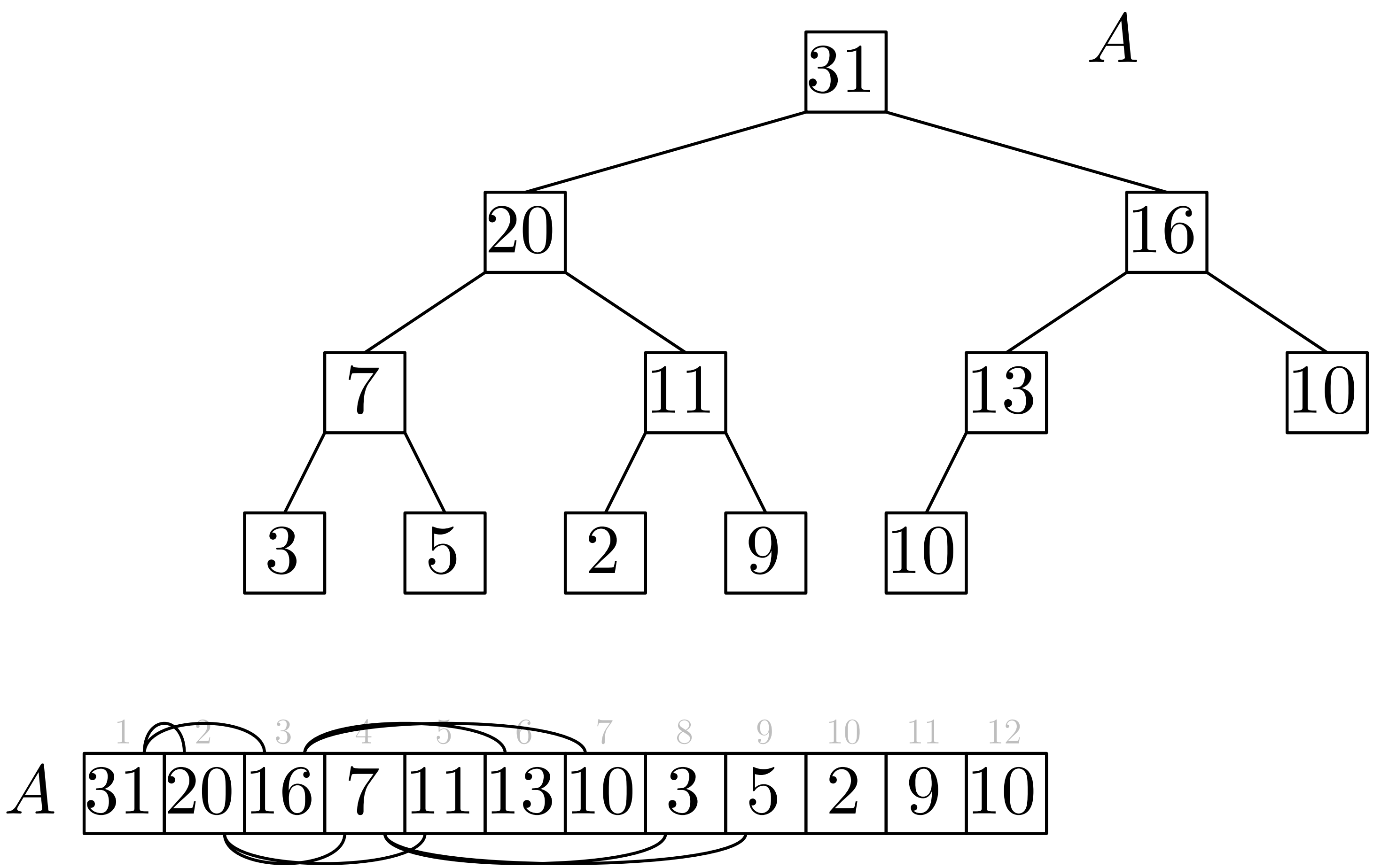
	1	2	3	4	5	6	7	8	9	10	11	12
A	31	20	16	7	11	13	10	3	5	2	9	10

Repræsentation med array

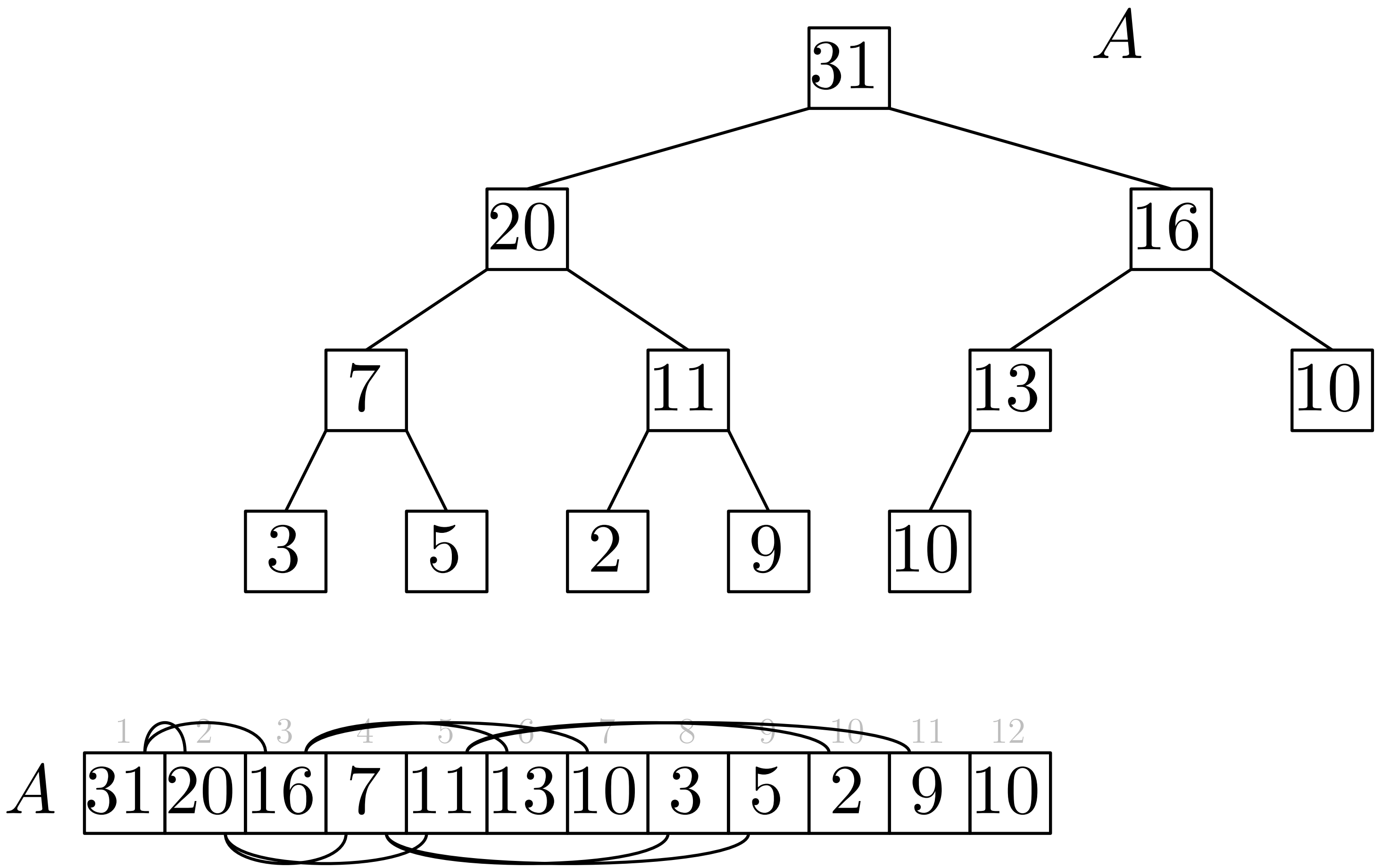


	1	2	3	4	5	6	7	8	9	10	11	12
A	31	20	16	7	11	13	10	3	5	2	9	10

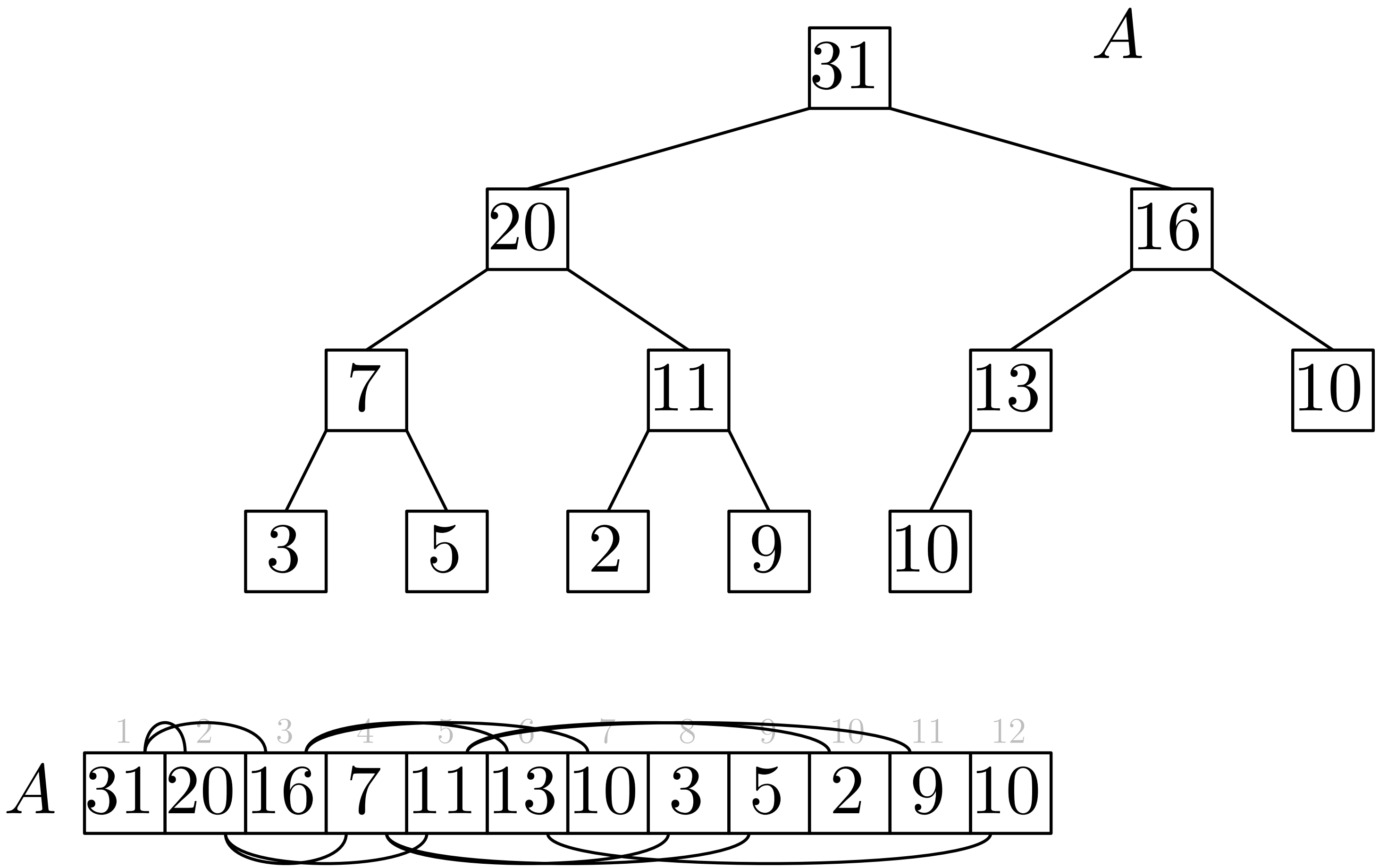
Repræsentation med array



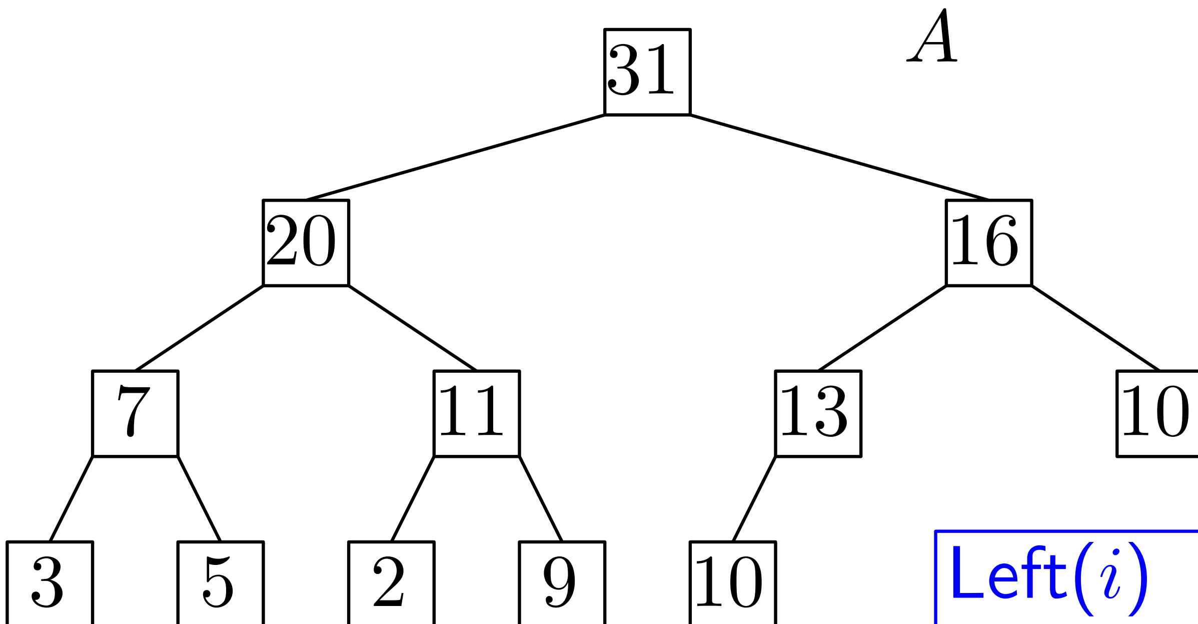
Repræsentation med array

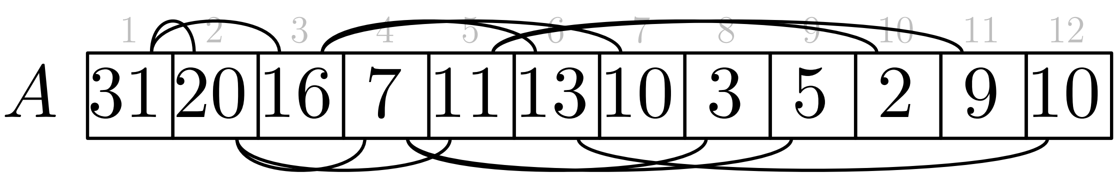


Repræsentation med array



Repræsentation med array

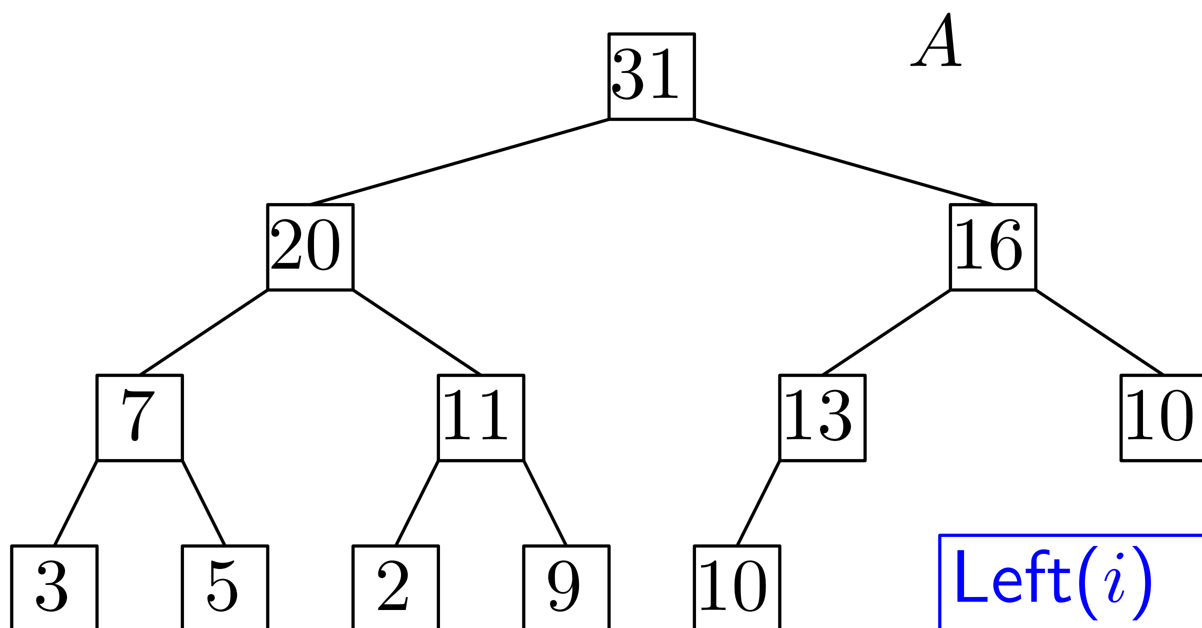


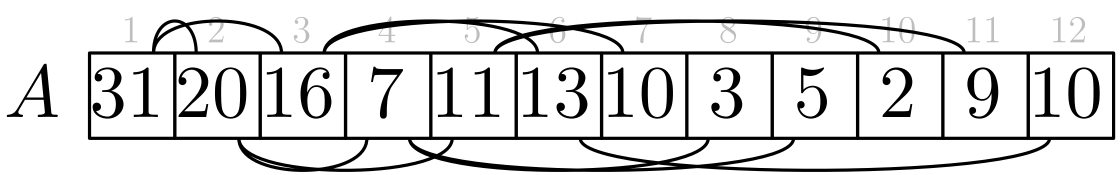


Left(i)
 return $2i$

Right(i)
 return $2i + 1$

Repræsentation med array





Left(i)
    return $2i$

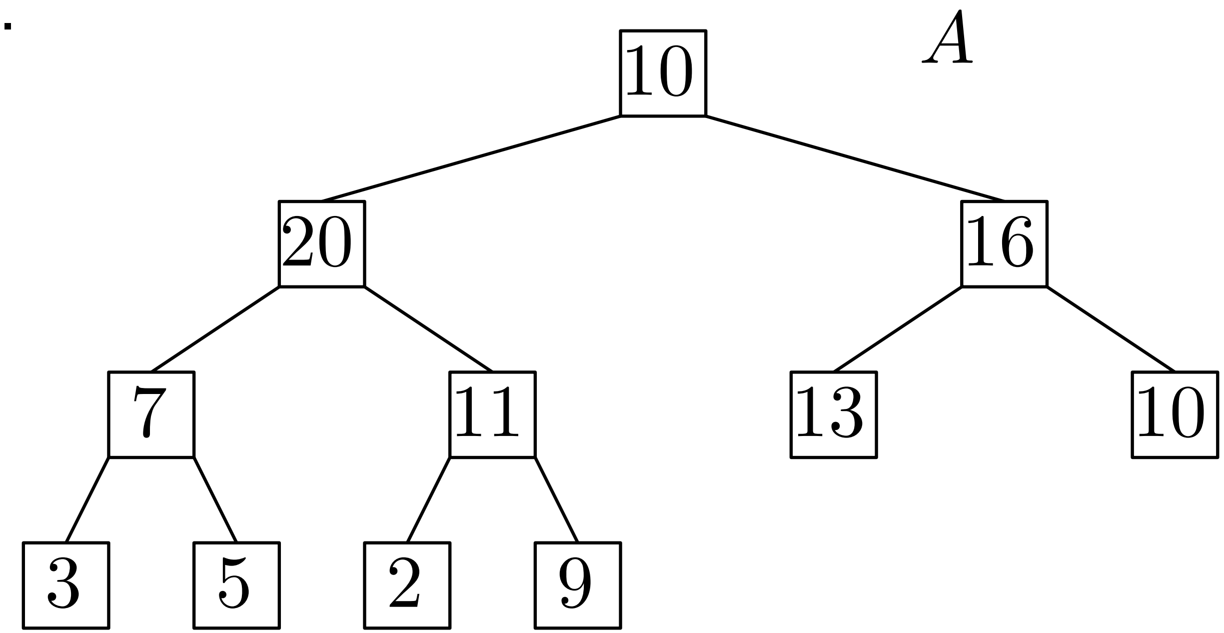
Right(i)
    return $2i + 1$

Parent(i)
    return $\lfloor \frac{i}{2} \rfloor$

Max-Heapify

Vi lader 10 "boble ned".

Max-Heapify$(A, 1)$



Max-Heapify(A, i)
    l = Left(i)
    r = Right(i)
    $largest = i$
    if $l \leq A.heap\text{-}size$ and $A[l] > A[largest]$
        $largest = l$
    if $r \leq A.heap\text{-}size$ and $A[r] > A[largest]$
        $largest = r$
    if $largest \neq i$
        swap $A[i]$ and $A[largest]$
        Max-Heapify$(A, largest)$

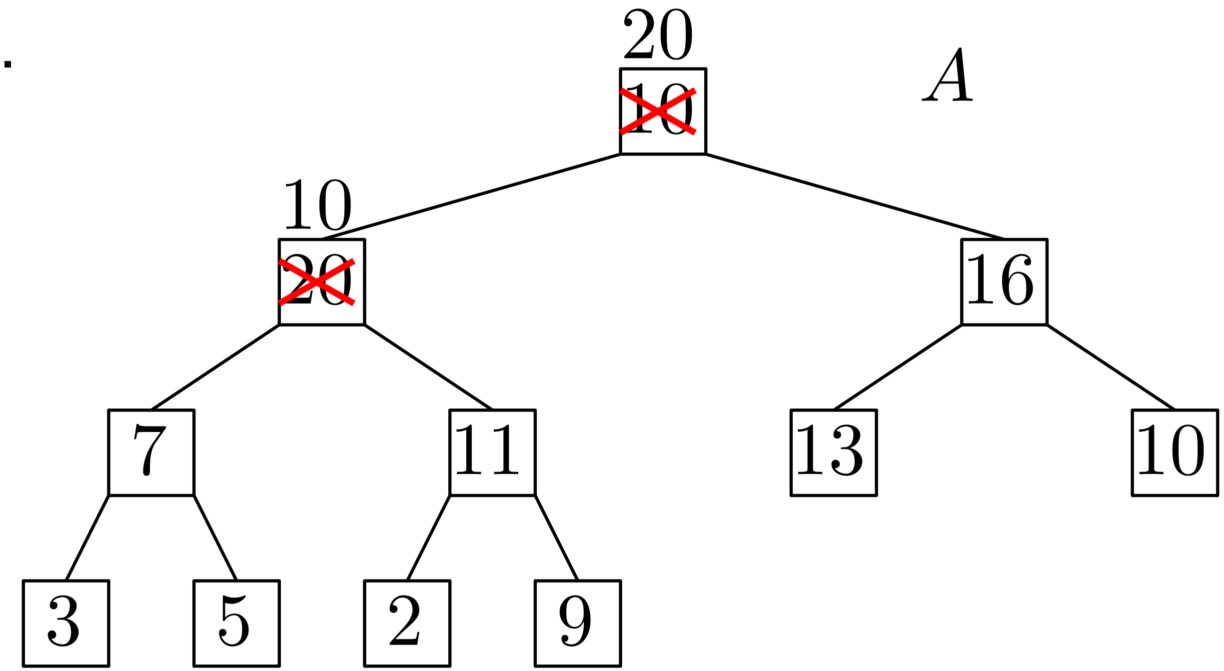
Left(i)
    return $2i$

Right(i)
    return $2i + 1$

Max-Heapify

Vi lader 10 "boble ned".

Max-Heapify$(A, 1)$



Max-Heapify(A, i)
    $l =$ Left(i)
    $r =$ Right(i)
    $largest = i$
    if $l \leq A.heap\text{-}size$ and $A[l] > A[largest]$
        $largest = l$
    if $r \leq A.heap\text{-}size$ and $A[r] > A[largest]$
        $largest = r$
    if $largest \neq i$
        swap $A[i]$ and $A[largest]$
        Max-Heapify$(A, largest)$

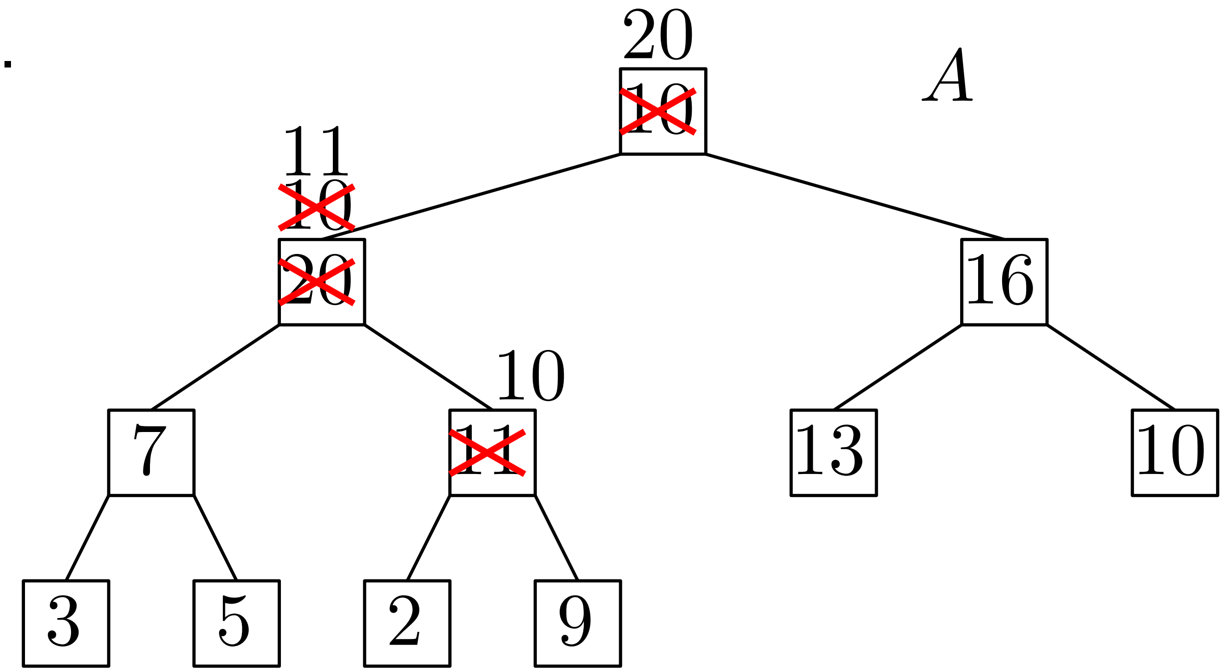
Left(i)
    return $2i$

Right(i)
    return $2i + 1$

Max-Heapify

Vi lader 10 “boble ned”.

Max-Heapify$(A, 1)$



Max-Heapify(A, i)
  $l =$ Left(i)
  $r =$ Right(i)
  $largest = i$
  if $l \leq A.heap\text{-}size$ and $A[l] > A[largest]$
    $largest = l$
  if $r \leq A.heap\text{-}size$ and $A[r] > A[largest]$
    $largest = r$
  if $largest \neq i$
    swap $A[i]$ and $A[largest]$
    Max-Heapify$(A, largest)$

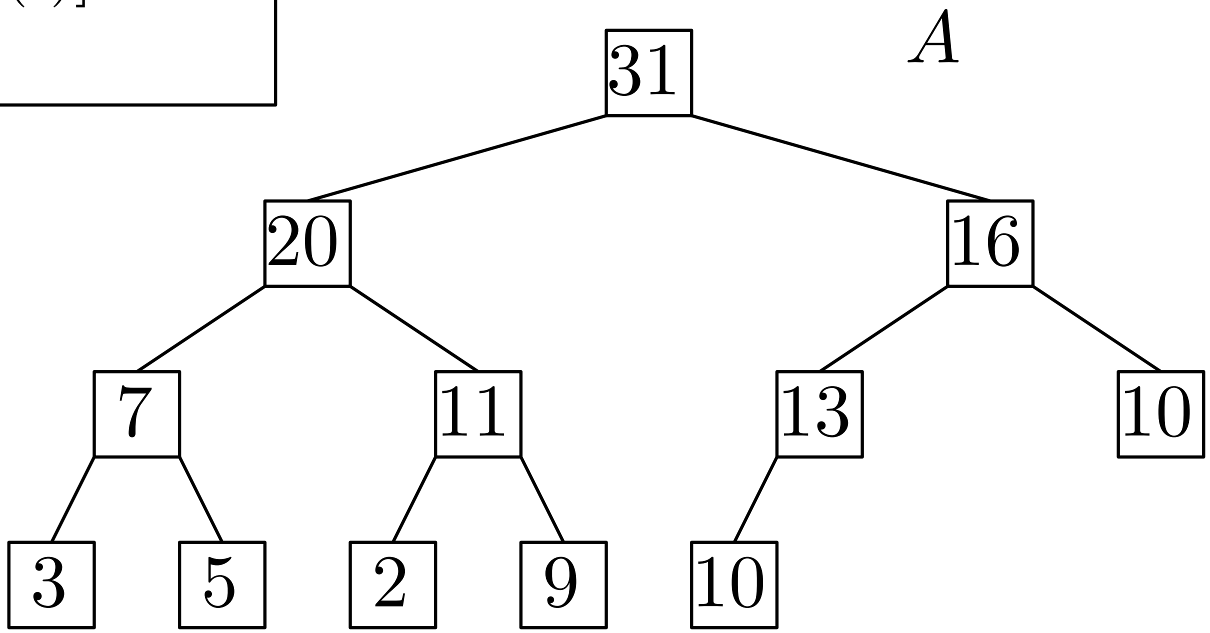
Left(i)
  return $2i$

Right(i)
  return $2i + 1$

Insert

Insert(A, k)
  $A.heap\text{-}size = A.heap\text{-}size + 1$
  $i = A.heap\text{-}size$
  $A[i] = k$
  while $i > 1$ and $A[\text{Parent}(i)] < A[i]$
    swap $A[i]$ and $A[\text{Parent}(i)]$
    $i = \text{Parent}(i)$

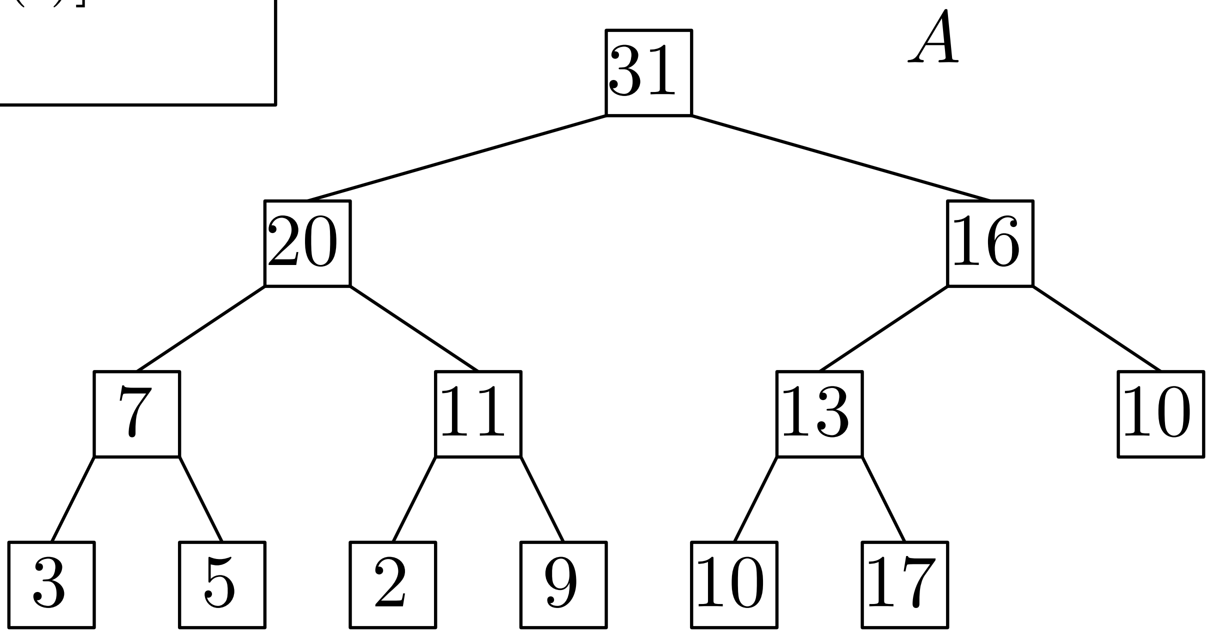
Parent(i)
  return $\lfloor \frac{i}{2} \rfloor$



Insert

```
Insert(A, k)
    A.heap-size = A.heap-size + 1
    i = A.heap-size
    A[i] = k
    while i > 1 and A[Parent(i)] < A[i]
        swap A[i] and A[Parent(i)]
        i = Parent(i)
```

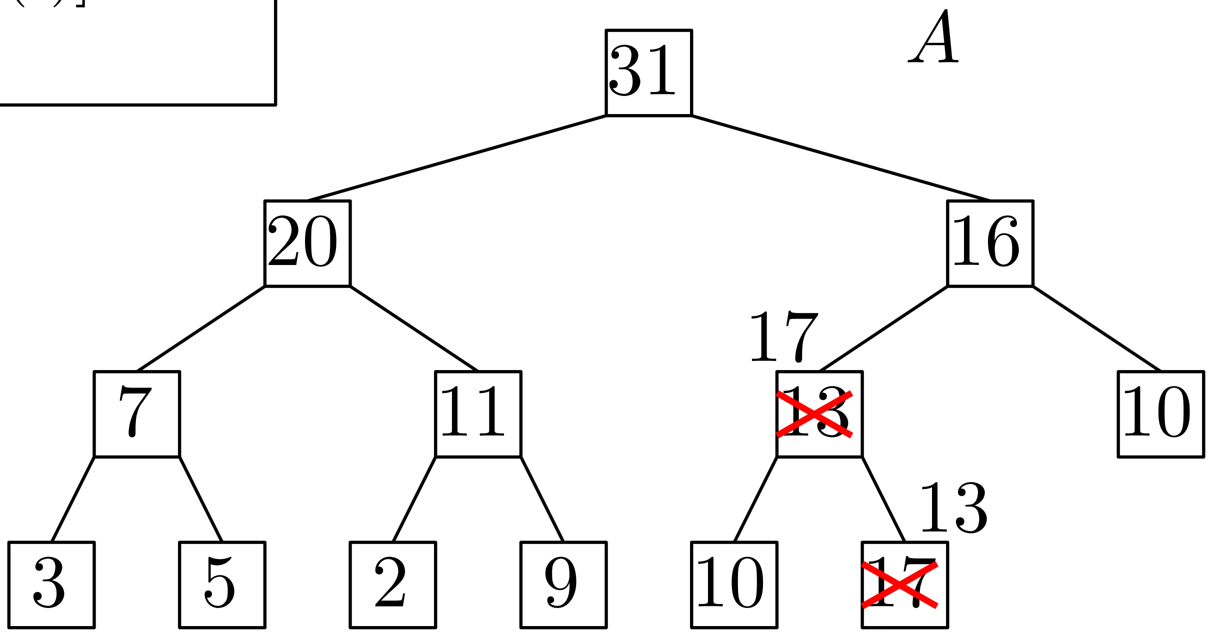
```
Parent(i)
    return ⌊i/2⌋
```



Insert

Insert(A, k)
  $A.heap\text{-}size = A.heap\text{-}size + 1$
  $i = A.heap\text{-}size$
  $A[i] = k$
  while $i > 1$ and $A[\text{Parent}(i)] < A[i]$
    swap $A[i]$ and $A[\text{Parent}(i)]$
    $i = \text{Parent}(i)$

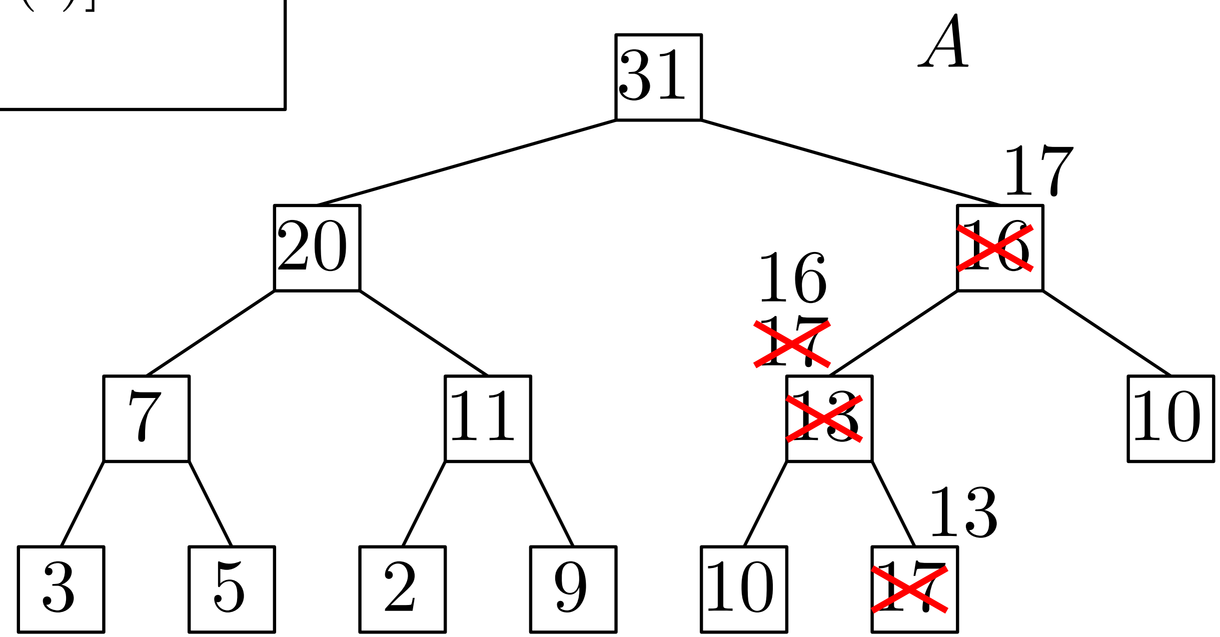
Parent(i)
  return $\lfloor \frac{i}{2} \rfloor$



Insert

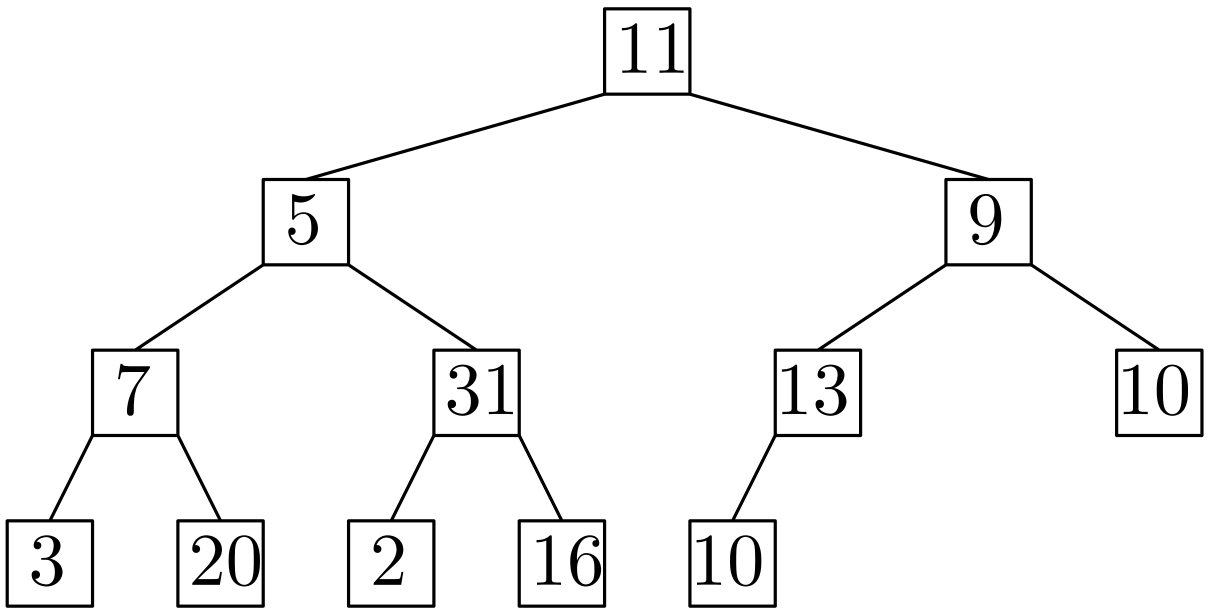
Insert(A, k)
  $A.heap\text{-}size = A.heap\text{-}size + 1$
  $i = A.heap\text{-}size$
  $A[i] = k$
  while $i > 1$ and $A[\text{Parent}(i)] < A[i]$
    swap $A[i]$ and $A[\text{Parent}(i)]$
    $i = \text{Parent}(i)$

Parent(i)
  return $\lfloor \frac{i}{2} \rfloor$



Konstruktion af hob

Build-Max-Heap(A)
  $A.heap\text{-}size = A.length$
  for $i = \lfloor \frac{A.length}{2} \rfloor$ downto 1
    Max-Heapify(A, i) // Lad $A[i]$ boble ned



11	5	9	7	31	13	10	3	20	2	16	10

Konstruktion af hob

```
Build-Max-Heap(A)
    A.heap-size = A.length
    for i = ⌊A.length/2⌋ downto 1
        Max-Heapify(A, i) // Lad A[i] boble ned
```

Build-Max-Heap(A)

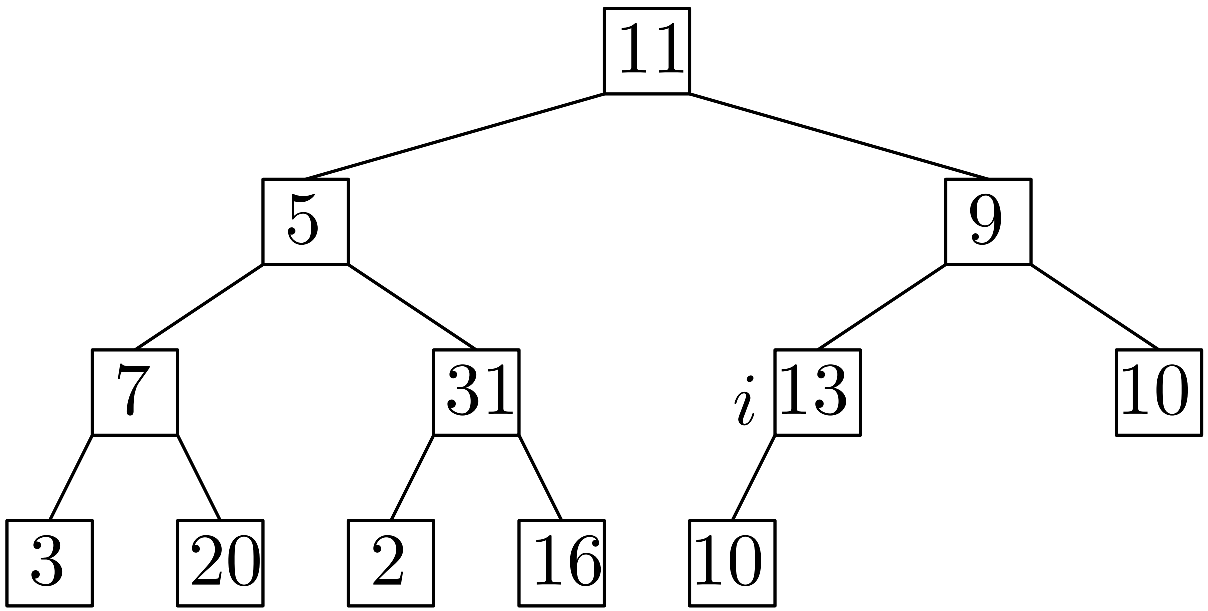
$A.heap\text{-}size = A.length$

for $i = \boxed{\left\lfloor \frac{A.length}{2} \right\rfloor}$ downto 1

Max-Heapify(A, i) // Lad $A[i]$ boble ned

$= A.length - \left\lceil \frac{A.length}{2} \right\rceil$

Sidste knude med et barn

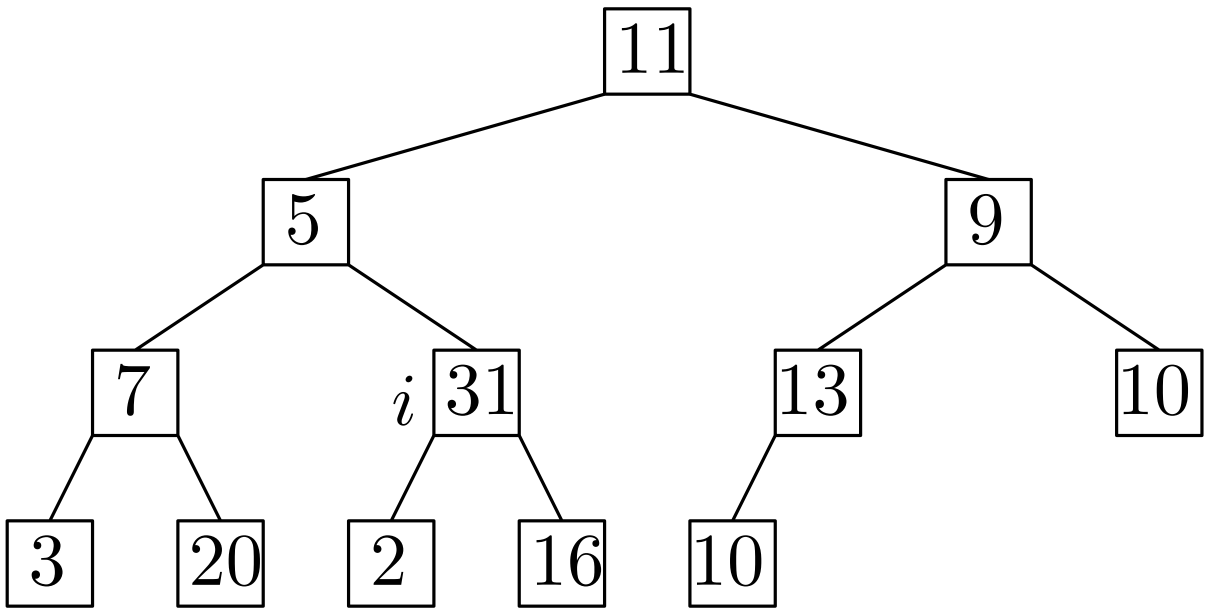


11	5	9	7	31	13	10	3	20	2	16	10
					i						

Konstruktion af hob

Build-Max-Heap(A)
    $A.heap\text{-}size = A.length$
    for $i = \boxed{\left\lfloor \frac{A.length}{2} \right\rfloor}$ downto 1
        Max-Heapify(A, i) // Lad $A[i]$ boble ned

$= A.length - \lceil \frac{A.length}{2} \rceil$
Sidste knude med et barn

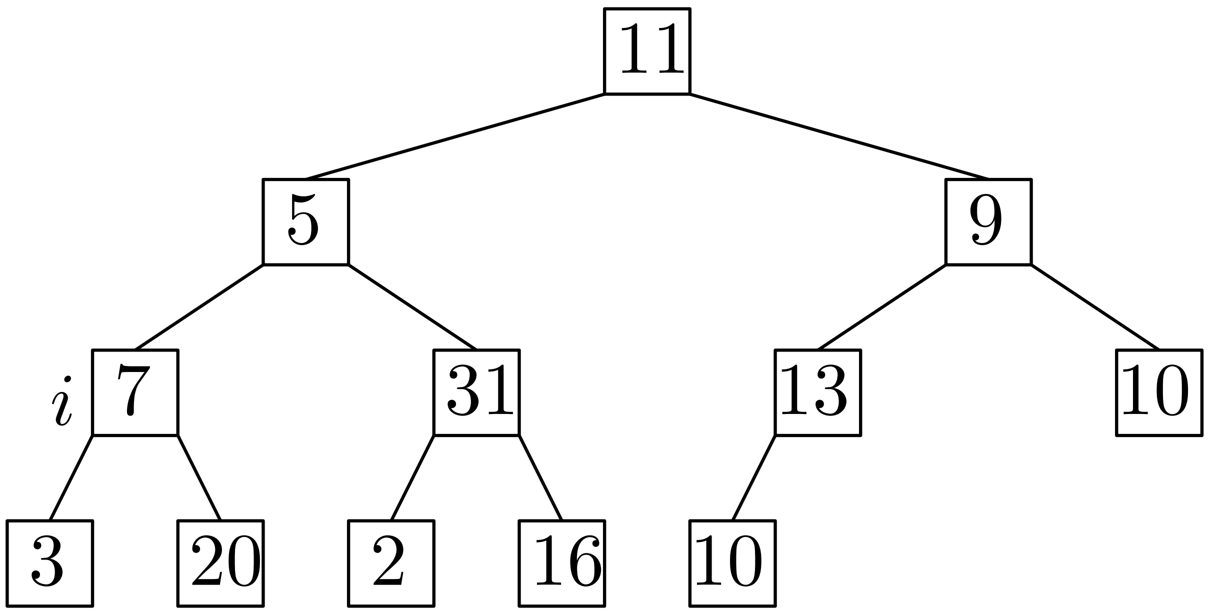


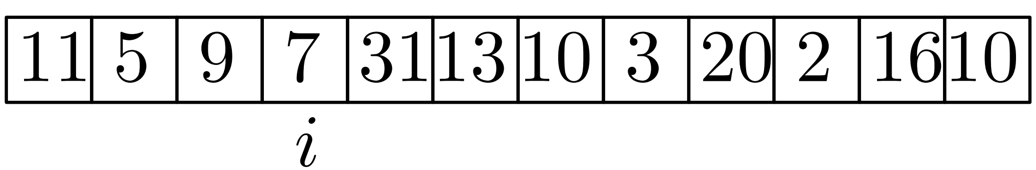
11	5	9	7	31	13	10	3	20	2	16	10
				i							

Konstruktion af hob

Build-Max-Heap(A)
  $A.heap\text{-}size = A.length$
  for $i = \boxed{\lfloor \frac{A.length}{2} \rfloor}$ downto 1
    Max-Heapify(A, i) // Lad $A[i]$ boble ned

$= A.length - \lceil \frac{A.length}{2} \rceil$
Sidste knude med et barn





Konstruktion af hob

Build-Max-Heap(A)

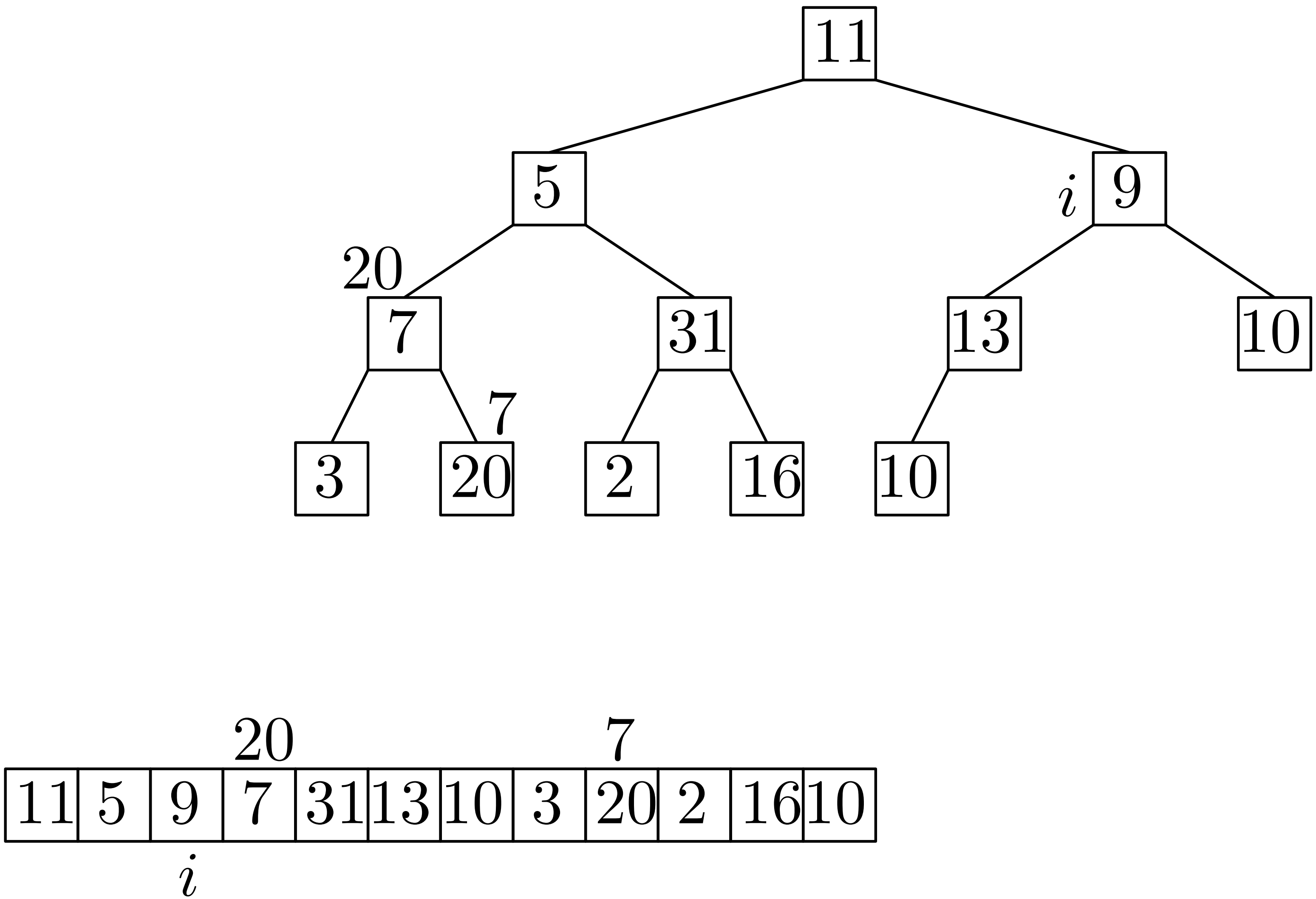
  $A.heap\text{-}size = A.length$

  for $i = \boxed{\left\lfloor \frac{A.length}{2} \right\rfloor}$ downto 1

    Max-Heapify(A, i) // Lad $A[i]$ boble ned

$= A.length - \lceil \frac{A.length}{2} \rceil$

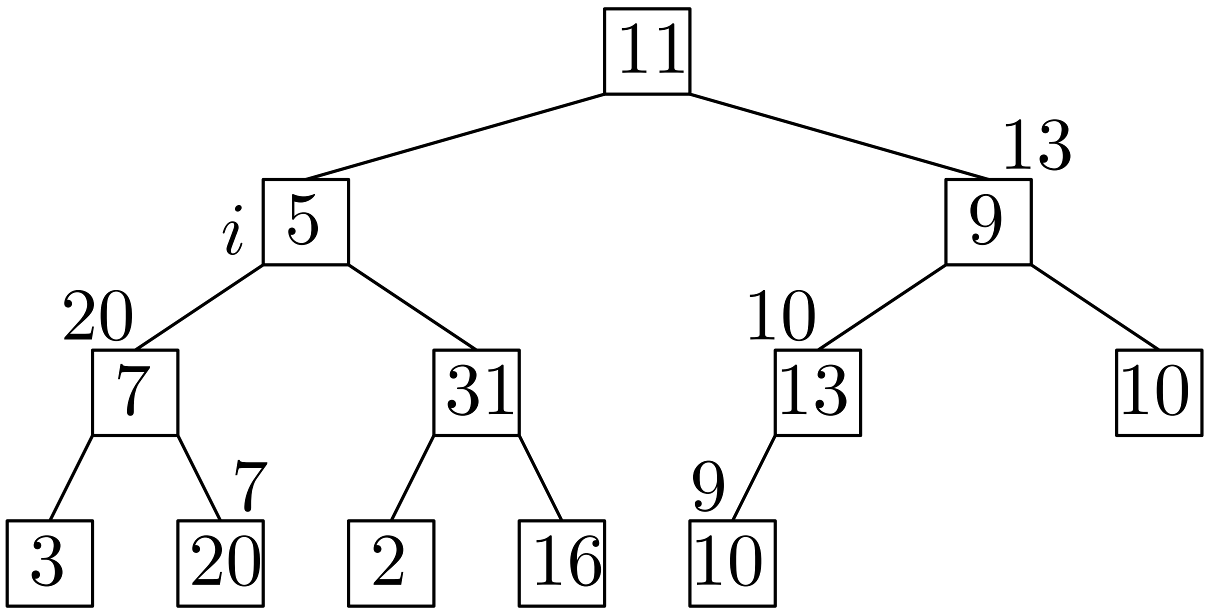
Sidste knude med et barn

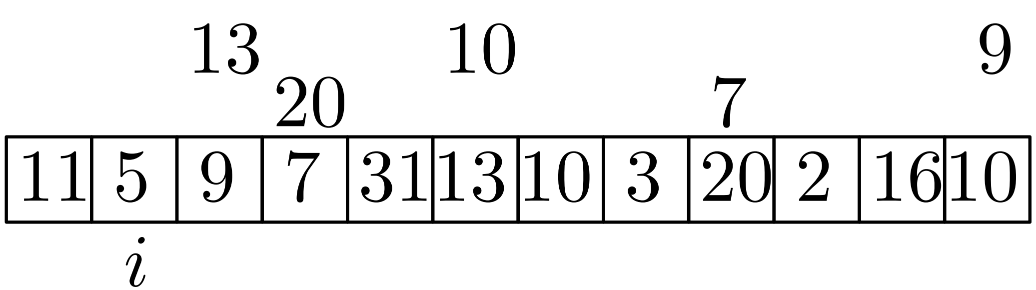


Konstruktion af hob

Build-Max-Heap(A)
    $A.heap\text{-}size = A.length$
    for $i = \boxed{\lfloor \frac{A.length}{2} \rfloor}$ downto 1
        Max-Heapify(A, i) // Lad $A[i]$ boble ned

$= A.length - \lceil \frac{A.length}{2} \rceil$
Sidste knude med et barn

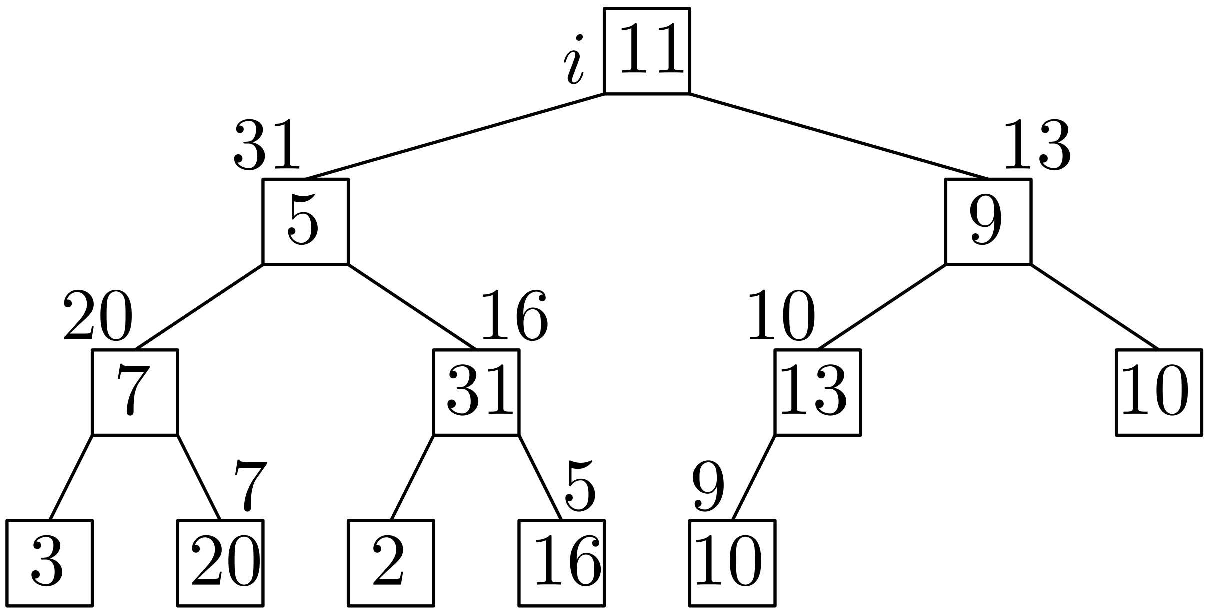


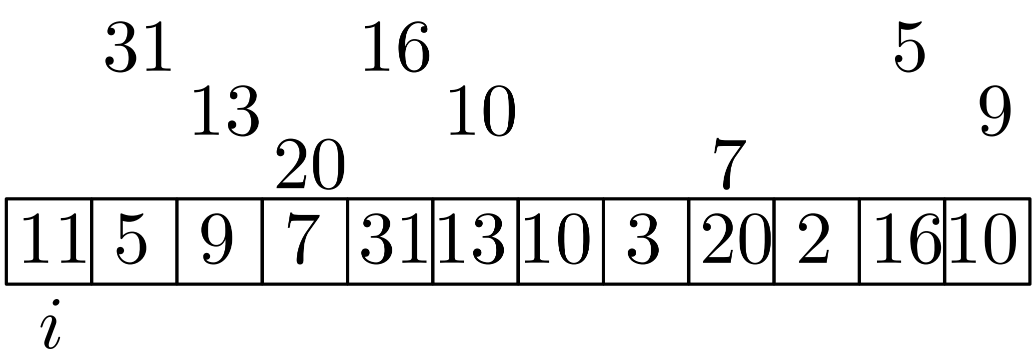


Konstruktion af hob

Build-Max-Heap(A)
    $A.heap\text{-}size = A.length$
    for $i = \boxed{\lfloor \frac{A.length}{2} \rfloor}$ downto 1
        Max-Heapify(A, i) // Lad $A[i]$ boble ned

$= A.length - \lceil \frac{A.length}{2} \rceil$
Sidste knude med et barn

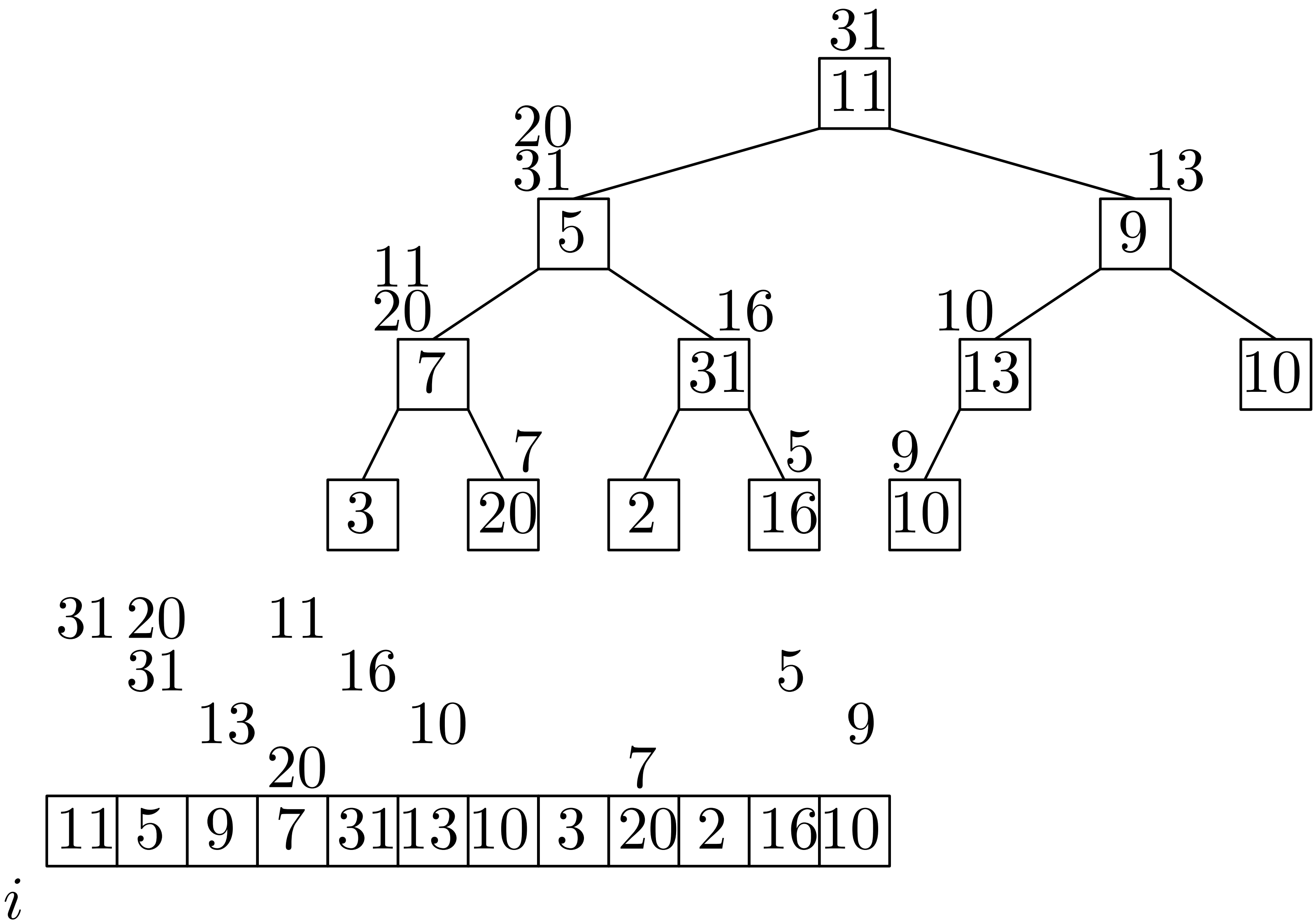




Konstruktion af hob

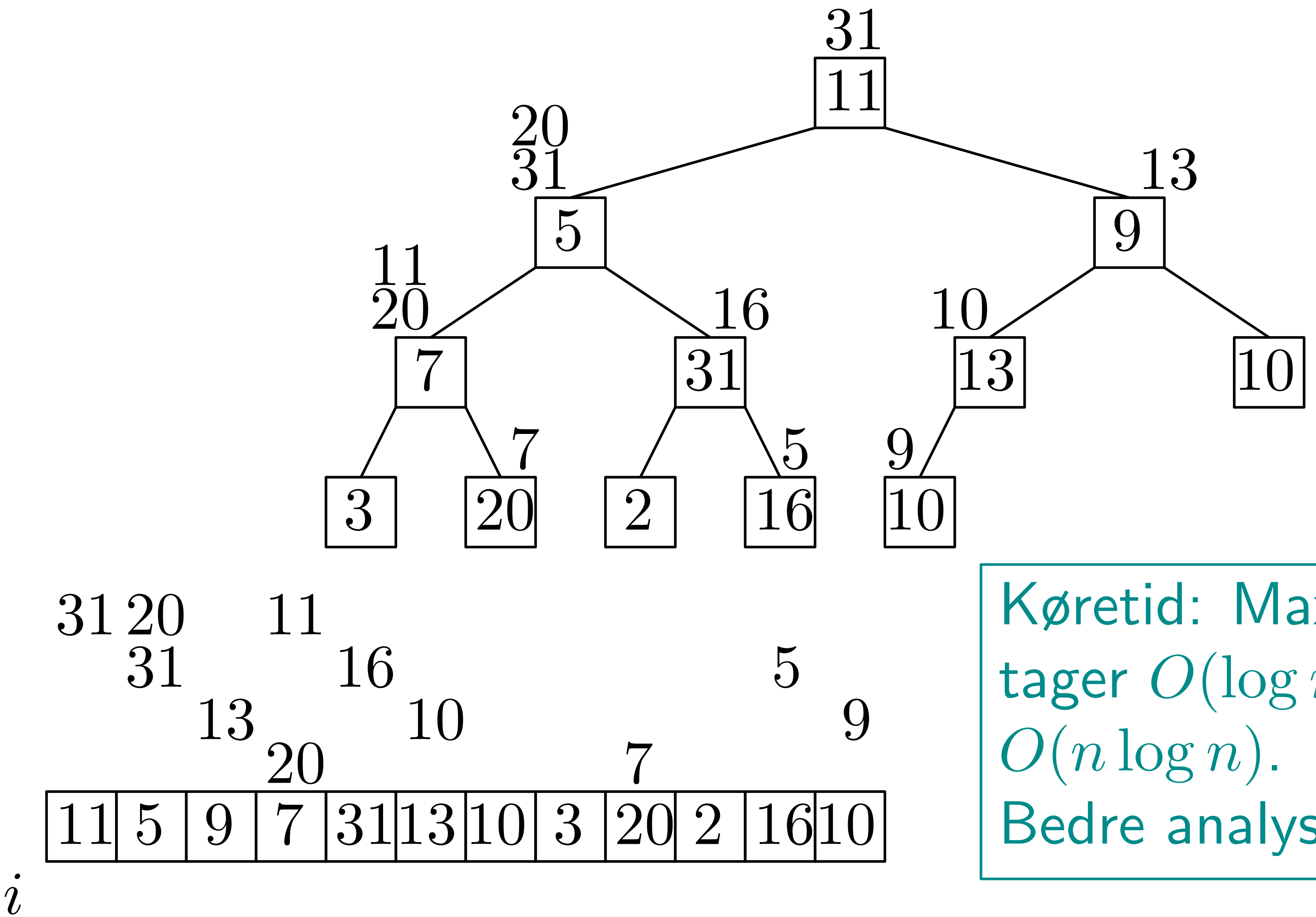
Build-Max-Heap(A)
  $A.heap\text{-}size = A.length$
  for $i = \boxed{\lfloor \frac{A.length}{2} \rfloor}$ downto 1
    Max-Heapify(A, i) // Lad $A[i]$ boble ned

$= A.length - \lceil \frac{A.length}{2} \rceil$
Sidste knude med et barn



Konstruktion af hob

Build-Max-Heap(A)
  $A.heap\text{-}size = A.length$
  for $i = \boxed{\lfloor \frac{A.length}{2} \rfloor}$ downto 1 $= A.length - \lceil \frac{A.length}{2} \rceil$ Sidste knude med et barn
    Max-Heapify(A, i) // Lad $A[i]$ boble ned



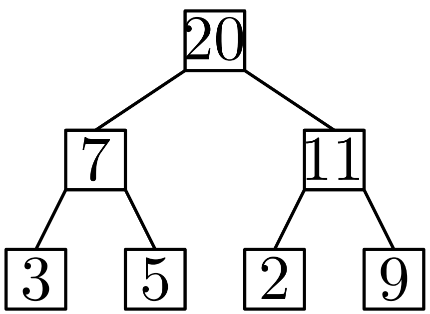
Køretid: Max-Heapify tager $O(\log n)$ tid. I alt $O(n \log n)$.
Bedre analyse: $\Theta(n)$.

Hvordan ser hoben ud til sidst?

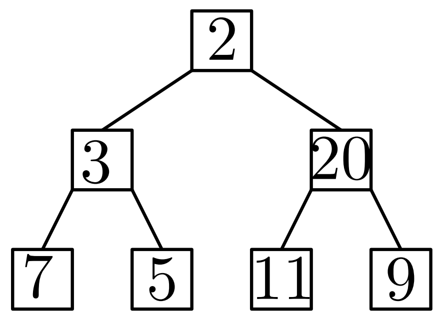
Build-Max-Heap(A)

A :

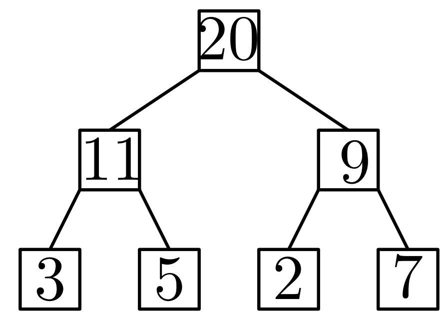
2	3	20	7	5	11	9



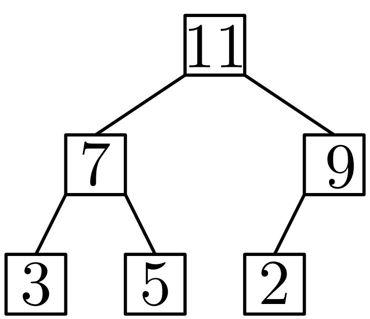
A



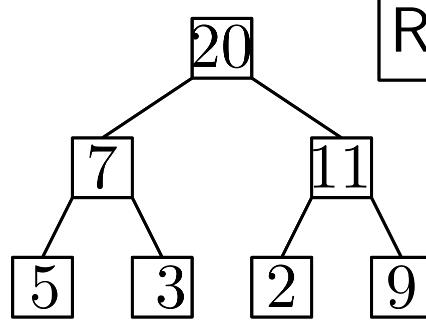
B



C



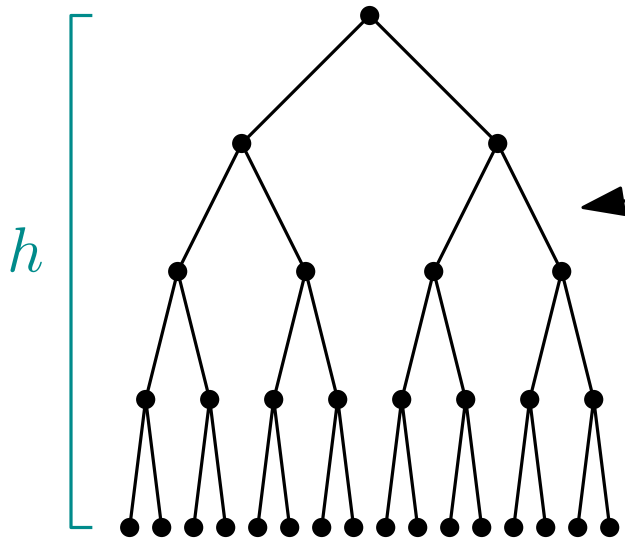
D



E

socrative.com $\rightarrow$ Student login,
Room name: ABRAHAMSEN3464

Bedre analyse af Build-Max-Heap

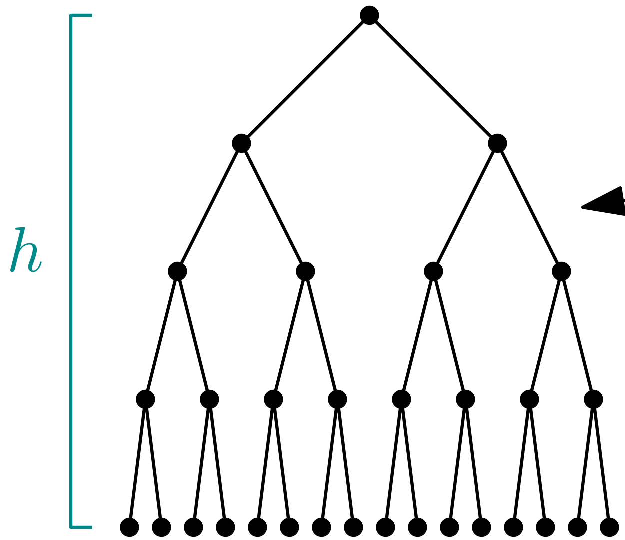


$$n = 2^{h+1} - 1$$

Her: $h = 4$, $n = 2^5 - 1 = 31$.

$$B = 2^{h-1} \cdot 1 + 2^{h-2} \cdot 2 + 2^{h-3} \cdot 3 + \ldots + 2^0 \cdot h$$

Bedre analyse af Build-Max-Heap



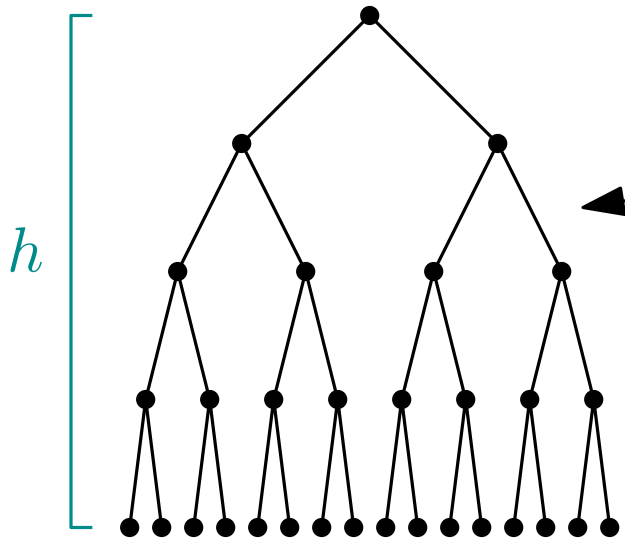
$$n = 2^{h+1} - 1$$

Her: $h = 4$, $n = 2^5 - 1 = 31$.

$$B = 2^{h-1} \cdot 1 + 2^{h-2} \cdot 2 + 2^{h-3} \cdot 3 + \ldots + 2^0 \cdot h$$

$$B/2 = B - B/2$$

Bedre analyse af Build-Max-Heap



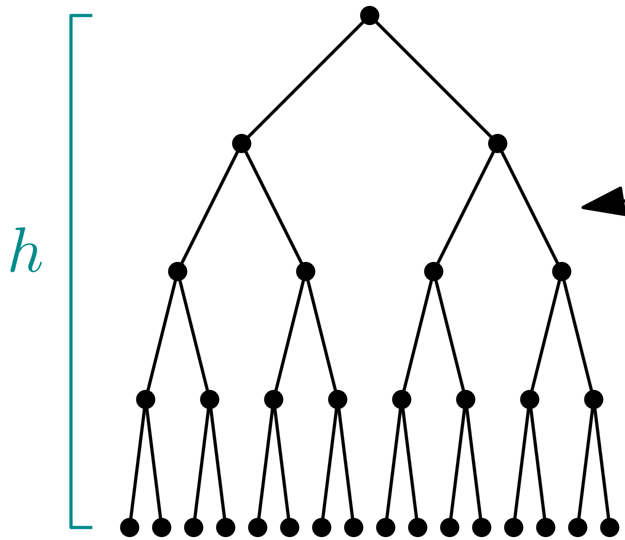
$$n = 2^{h+1} - 1$$

Her: $h = 4$, $n = 2^5 - 1 = 31$.

$$B = 2^{h-1} \cdot 1 + 2^{h-2} \cdot 2 + 2^{h-3} \cdot 3 + \ldots + 2^0 \cdot h$$

$$\begin{aligned} B/2 = B - B/2 \\ &= 2^{h-1} \cdot 1 + 2^{h-2} \cdot 2 + 2^{h-3} \cdot 3 + \ldots + 2^0 \cdot h \\ &\quad - 2^{h-2} \cdot 1 - 2^{h-3} \cdot 2 - \ldots - 2^0 \cdot (h-1) - 2^{-1} \cdot h \end{aligned}$$

Bedre analyse af Build-Max-Heap



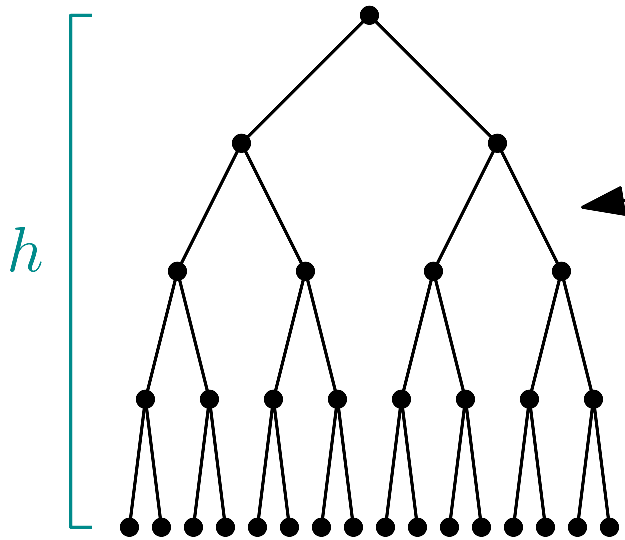
$$n = 2^{h+1} - 1$$

Her: $h = 4$, $n = 2^5 - 1 = 31$.

$$B = 2^{h-1} \cdot 1 + 2^{h-2} \cdot 2 + 2^{h-3} \cdot 3 + \ldots + 2^0 \cdot h$$

$$\begin{aligned} B/2 = B - B/2 \\ &= 2^{h-1} \cdot 1 + 2^{h-2} \cdot 2 + 2^{h-3} \cdot 3 + \ldots + 2^0 \cdot h \\ &\qquad - 2^{h-2} \cdot 1 - 2^{h-3} \cdot 2 - \ldots - 2^0 \cdot (h-1) - 2^{-1} \cdot h \\ &= 2^{h-1} + 2^{h-2} + \ldots + 2^0 - h/2 \end{aligned}$$

Bedre analyse af Build-Max-Heap

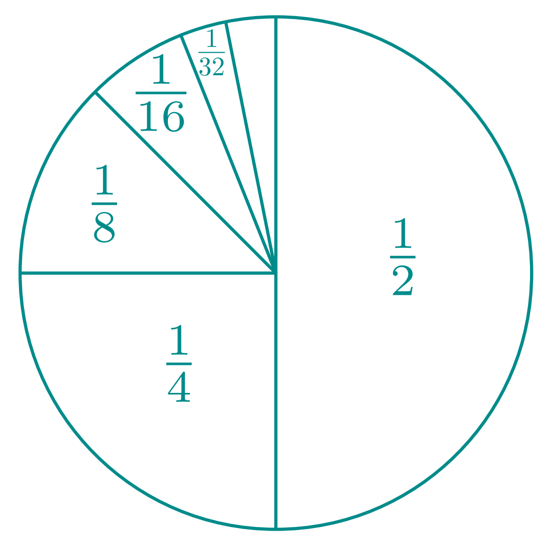


$$n = 2^{h+1} - 1$$

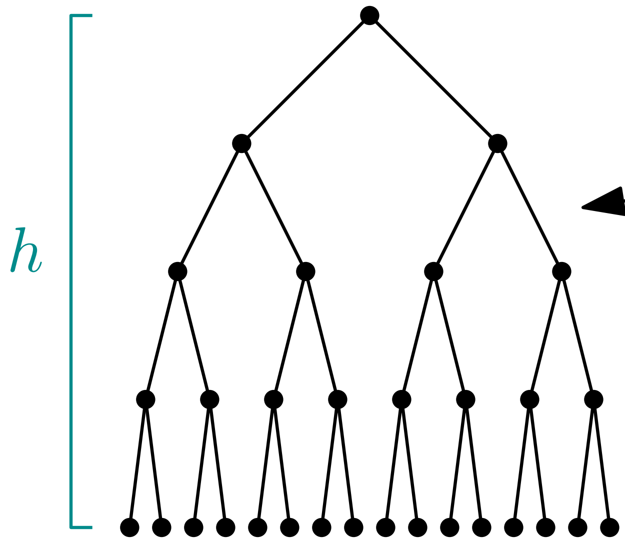
Her: $h = 4$, $n = 2^5 - 1 = 31$.

$$B = 2^{h-1} \cdot 1 + 2^{h-2} \cdot 2 + 2^{h-3} \cdot 3 + \ldots + 2^0 \cdot h$$

$$\begin{aligned}
B/2 = B - B/2 \\
&= 2^{h-1} \cdot 1 + 2^{h-2} \cdot 2 + 2^{h-3} \cdot 3 + \ldots + 2^0 \cdot h \\
&\qquad - 2^{h-2} \cdot 1 - 2^{h-3} \cdot 2 - \ldots - 2^0 \cdot (h-1) - 2^{-1} \cdot h \\
&= 2^{h-1} + 2^{h-2} + \ldots + 2^0 - h/2 \\
&< 2^h - h/2
\end{aligned}$$



Bedre analyse af Build-Max-Heap

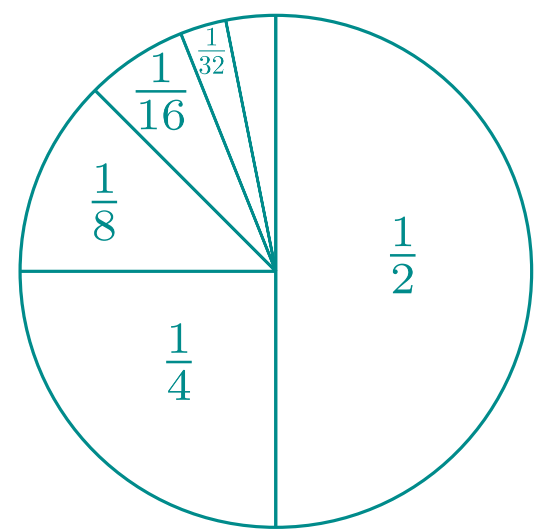


$$n = 2^{h+1} - 1$$

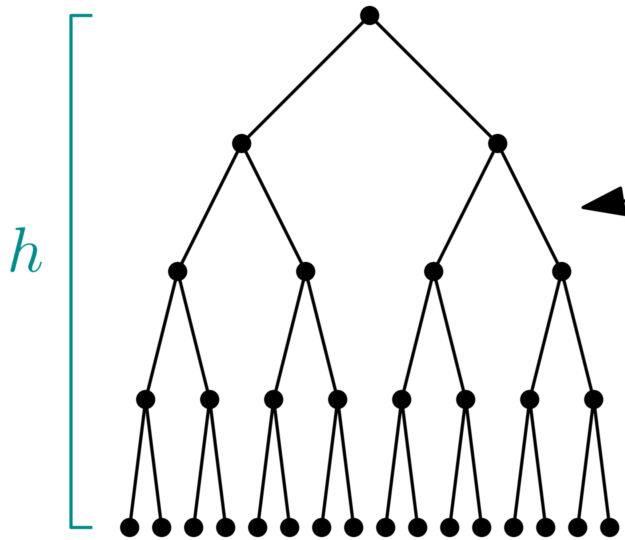
Her: $h = 4$, $n = 2^5 - 1 = 31$.

$$B = 2^{h-1} \cdot 1 + 2^{h-2} \cdot 2 + 2^{h-3} \cdot 3 + \ldots + 2^0 \cdot h$$

$$\begin{aligned} B/2 = B - B/2 \\ &= 2^{h-1} \cdot 1 + 2^{h-2} \cdot 2 + 2^{h-3} \cdot 3 + \ldots + 2^0 \cdot h \\ &\qquad - 2^{h-2} \cdot 1 - 2^{h-3} \cdot 2 - \ldots - 2^0 \cdot (h-1) - 2^{-1} \cdot h \\ &= 2^{h-1} + 2^{h-2} + \ldots + 2^0 - h/2 \\ &< 2^h - h/2 < 2^h \end{aligned}$$



Bedre analyse af Build-Max-Heap

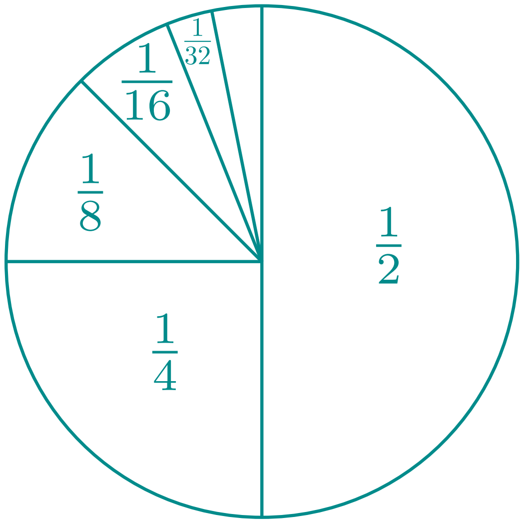


$n = 2^{h+1} - 1$

Her: $h = 4$, $n = 2^5 - 1 = 31$.

$$B = 2^{h-1} \cdot 1 + 2^{h-2} \cdot 2 + 2^{h-3} \cdot 3 + \ldots + 2^0 \cdot h$$

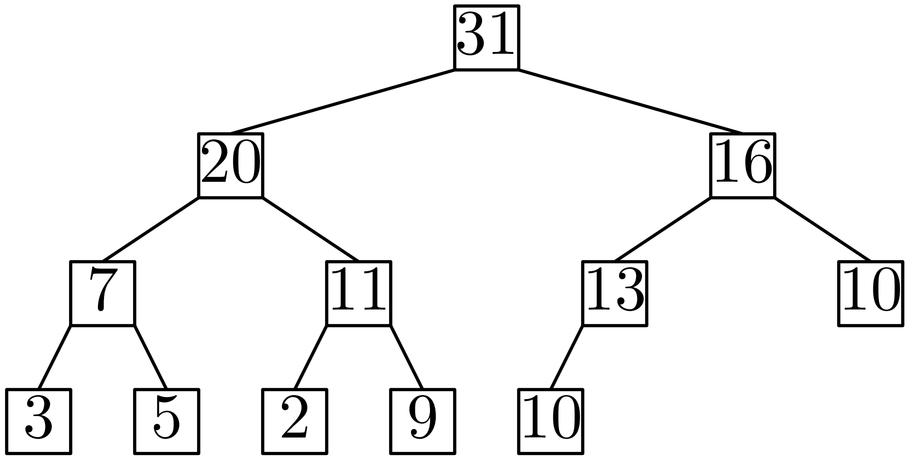
$$\begin{aligned} B/2 &= B - B/2 \\ &= 2^{h-1} \cdot 1 + 2^{h-2} \cdot 2 + 2^{h-3} \cdot 3 + \ldots + 2^0 \cdot h \\ &\qquad - 2^{h-2} \cdot 1 - 2^{h-3} \cdot 2 - \ldots - 2^0 \cdot (h-1) - 2^{-1} \cdot h \\ &= 2^{h-1} + 2^{h-2} + \ldots + 2^0 - h/2 \\ &< 2^h - h/2 < 2^h \end{aligned}$$



Konklusion: $B < 2^{h+1}$, så $B \leq 2^{h+1} - 1 = n$.

Køretid for Build-Max-Heap: $\Theta(n)$.

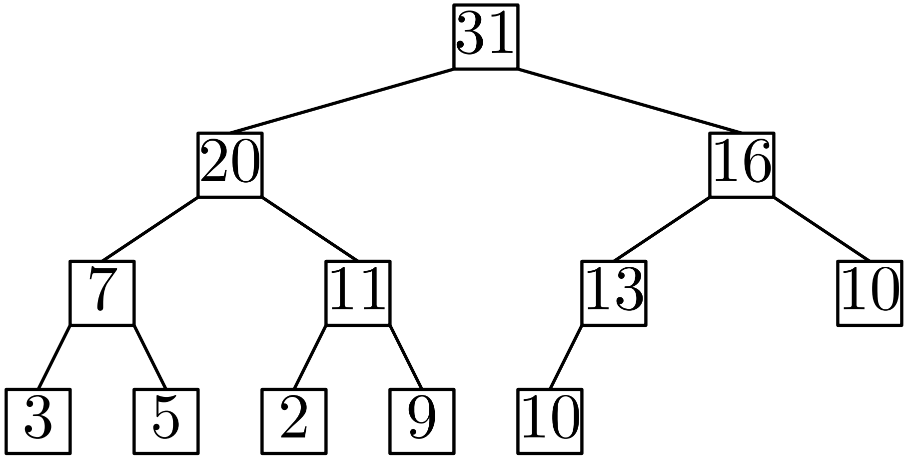
Opsummering



"Fyldt op"

Hobeordenen

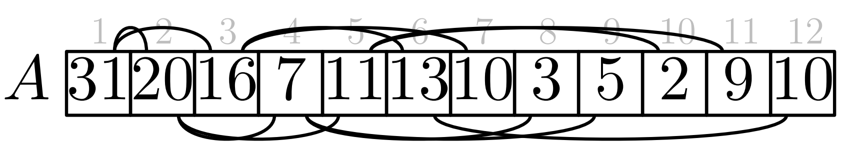
Opsummering



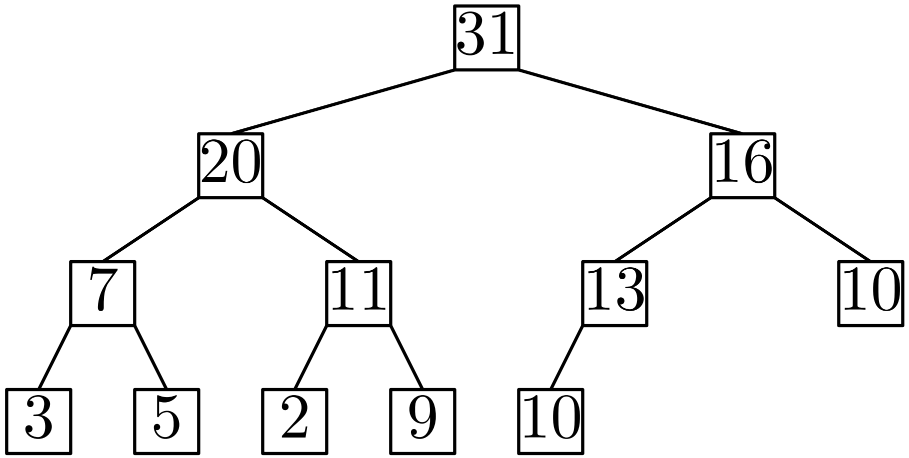
“Fyldt op”

Hobeordenen

Som array:



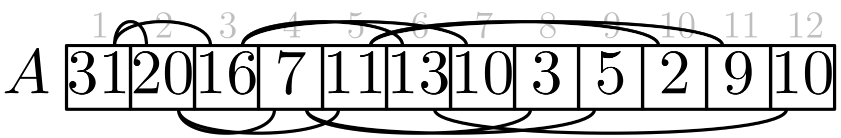
Opsummering



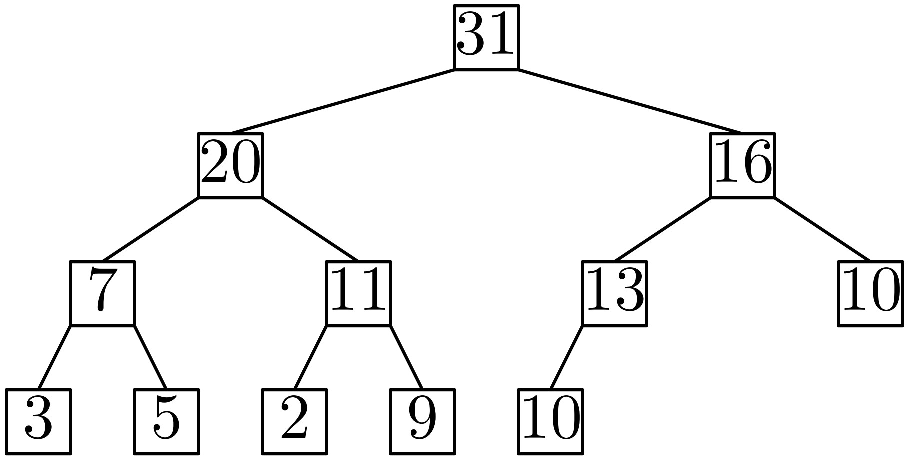
Extract-Max: Overskrive rod med sidste tal. Boble ned (**Max-Heapify**).
Insert: Tilføje tal til sidst. Boble op.
Begge: $\Theta(\log n)$ tid.

"Fyldt op"
Hobeordenen

Som array:



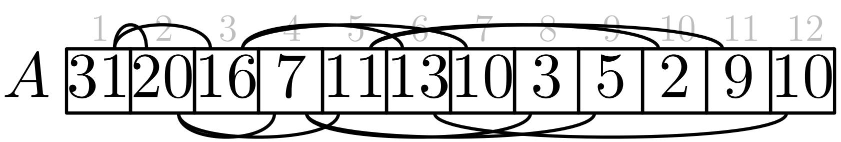
Opsummering



"Fyldt op"

Hobeordenen

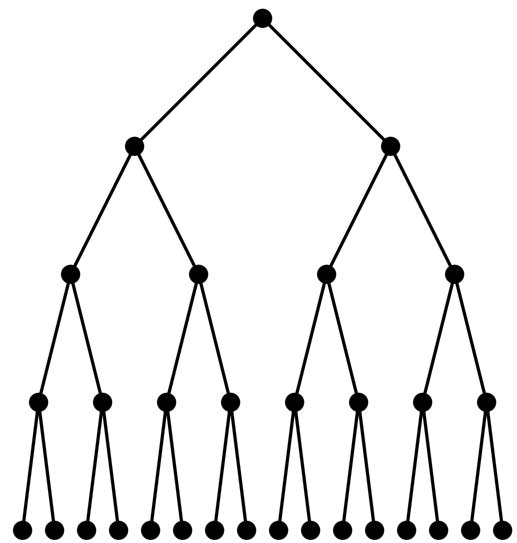
Som array:



Extract-Max: Overskrive rod med sidste tal. Boble ned (**Max-Heapify**).

Insert: Tilføje tal til sidst. Boble op.

Begge: $\Theta(\log n)$ tid.



Build-Max-Heap: Brug

Max-Heapify fra $\lfloor n/2 \rfloor$ ned til 1.

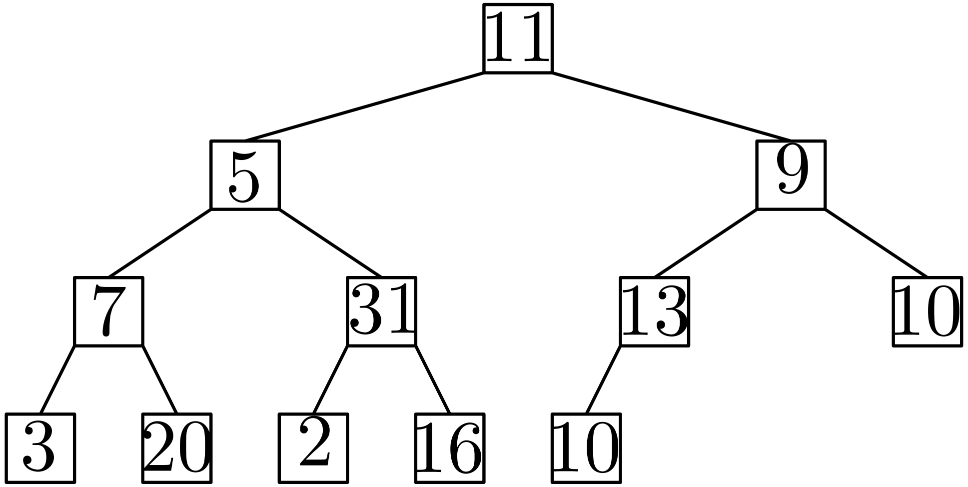
Nem grænse for køretid: $O(n \log n)$.

Mere omhyggelig: $\Theta(n)$.

Alternativ algoritme: Brug Insert n gange giver $\Theta(n \log n)$.

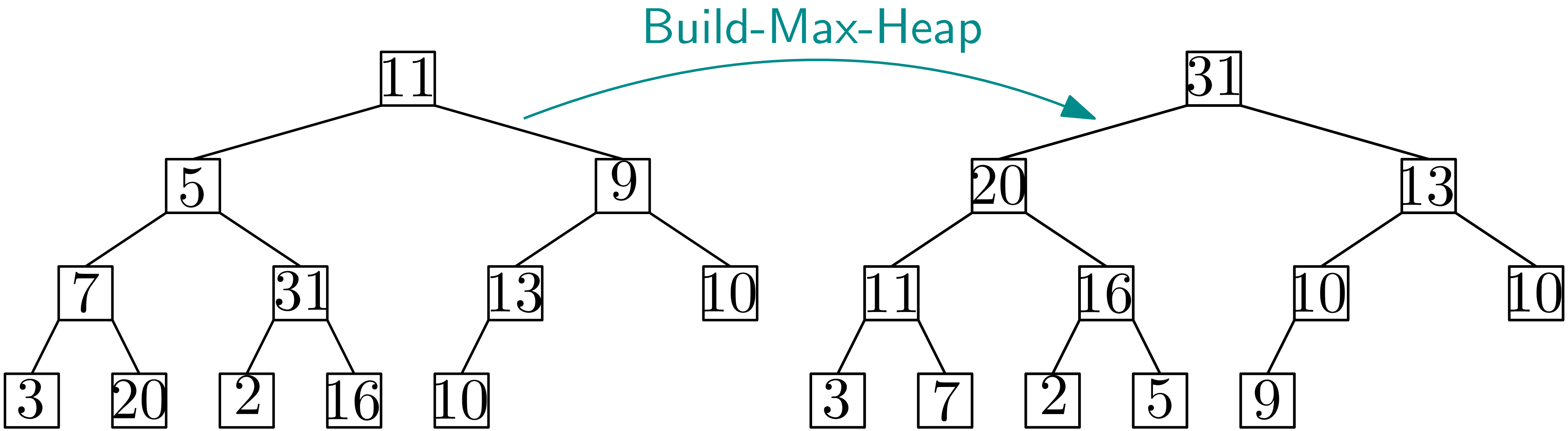
Heapsort

Heapsort(A)
  Build-Max-Heap(A)
  for $i = A.length$ downto 2
    swap $A[1]$ and $A[i]$
    $A.heap\text{-}size = A.heap\text{-}size - 1$
    Max-Heapify($A, 1$)



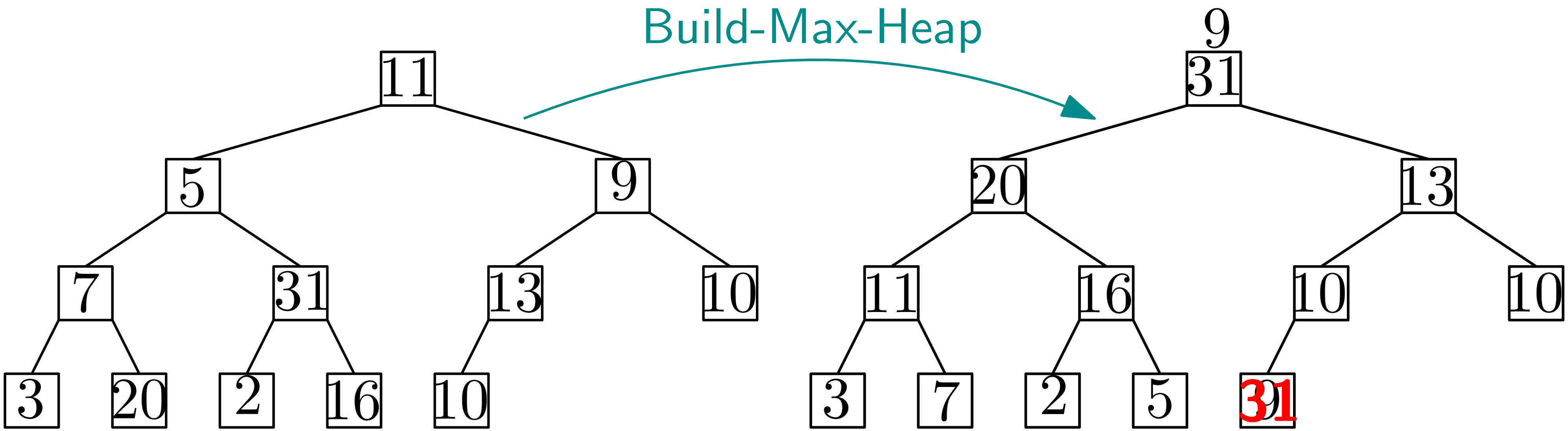
Heapsort

Heapsort(A)
  Build-Max-Heap(A)
  for $i = A.length$ downto 2
    swap $A[1]$ and $A[i]$
    $A.heap\text{-}size = A.heap\text{-}size - 1$
    Max-Heapify($A, 1$)



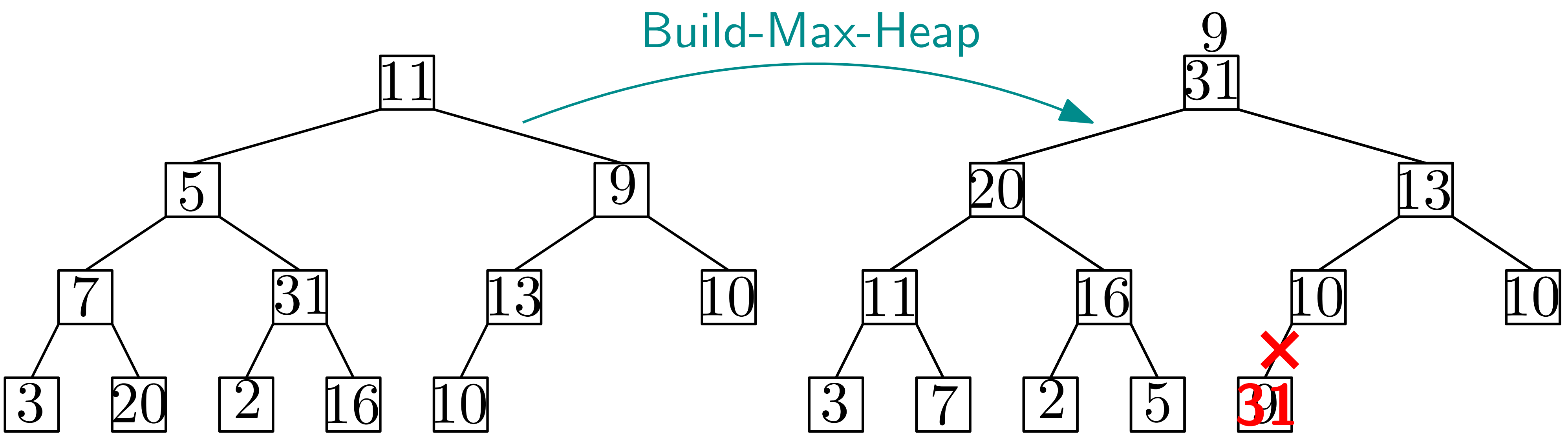
Heapsort

Heapsort(A)
  Build-Max-Heap(A)
  for $i = A.length$ downto 2
    swap $A[1]$ and $A[i]$
    $A.heap\text{-}size = A.heap\text{-}size - 1$
    Max-Heapify($A, 1$)



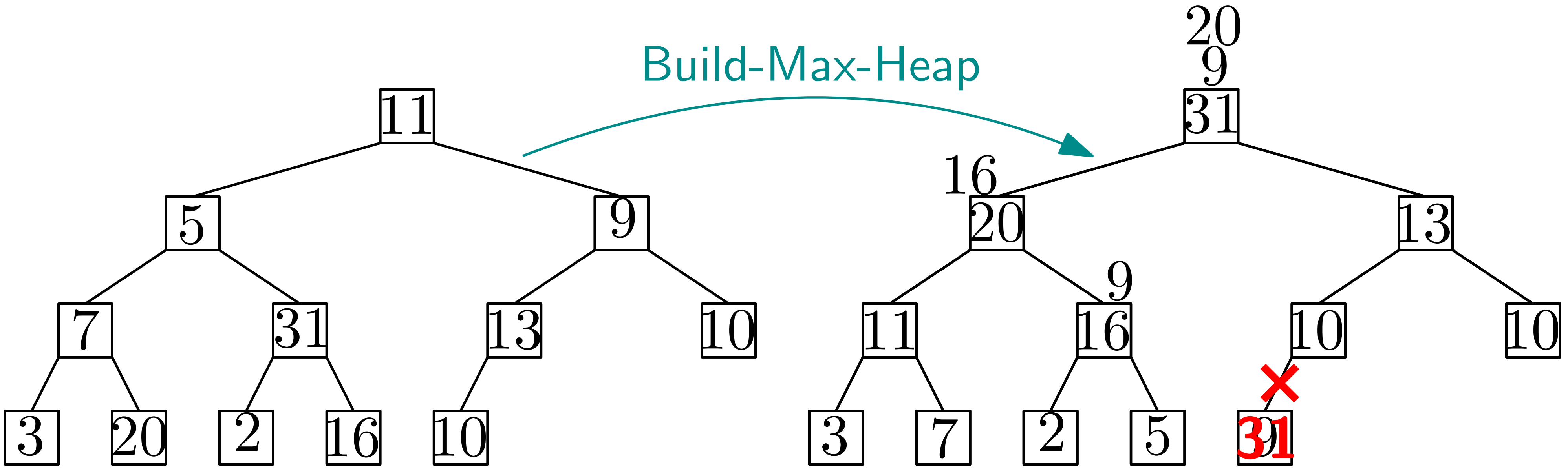
Heapsort

Heapsort(A)
  Build-Max-Heap(A)
  for $i = A.length$ downto 2
    swap $A[1]$ and $A[i]$
    $A.heap\text{-}size = A.heap\text{-}size - 1$
    Max-Heapify($A, 1$)



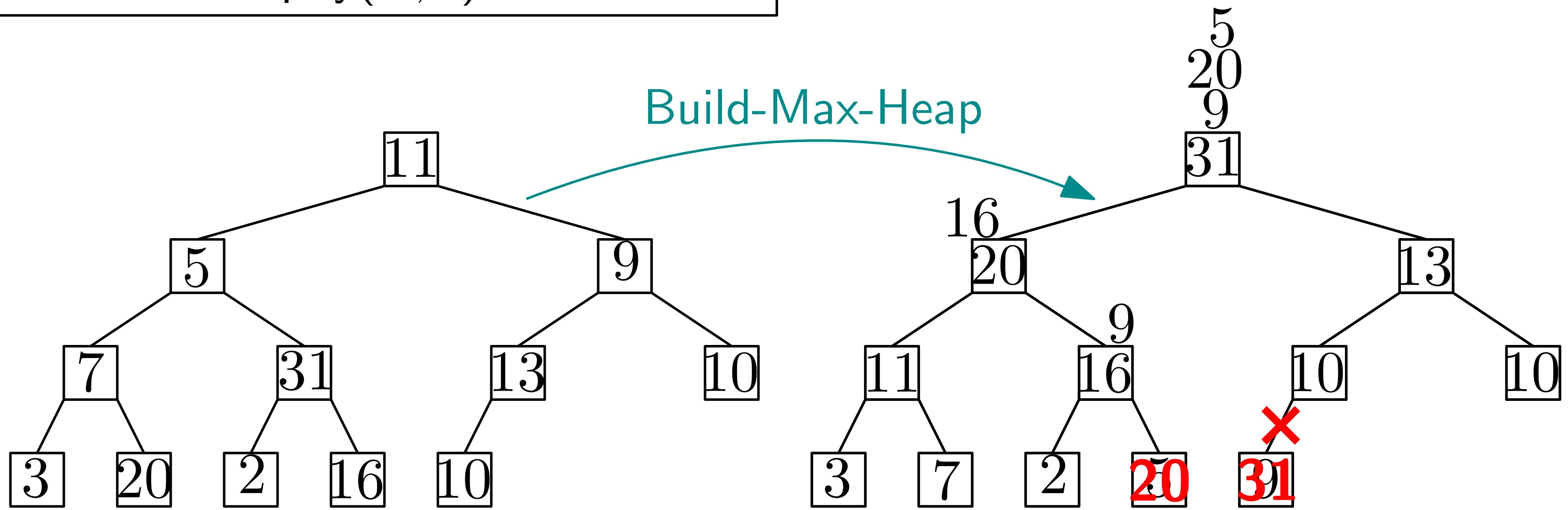
Heapsort

Heapsort(A)
  Build-Max-Heap(A)
  for $i = A.length$ downto 2
    swap $A[1]$ and $A[i]$
    $A.heap\text{-}size = A.heap\text{-}size - 1$
    Max-Heapify($A, 1$)



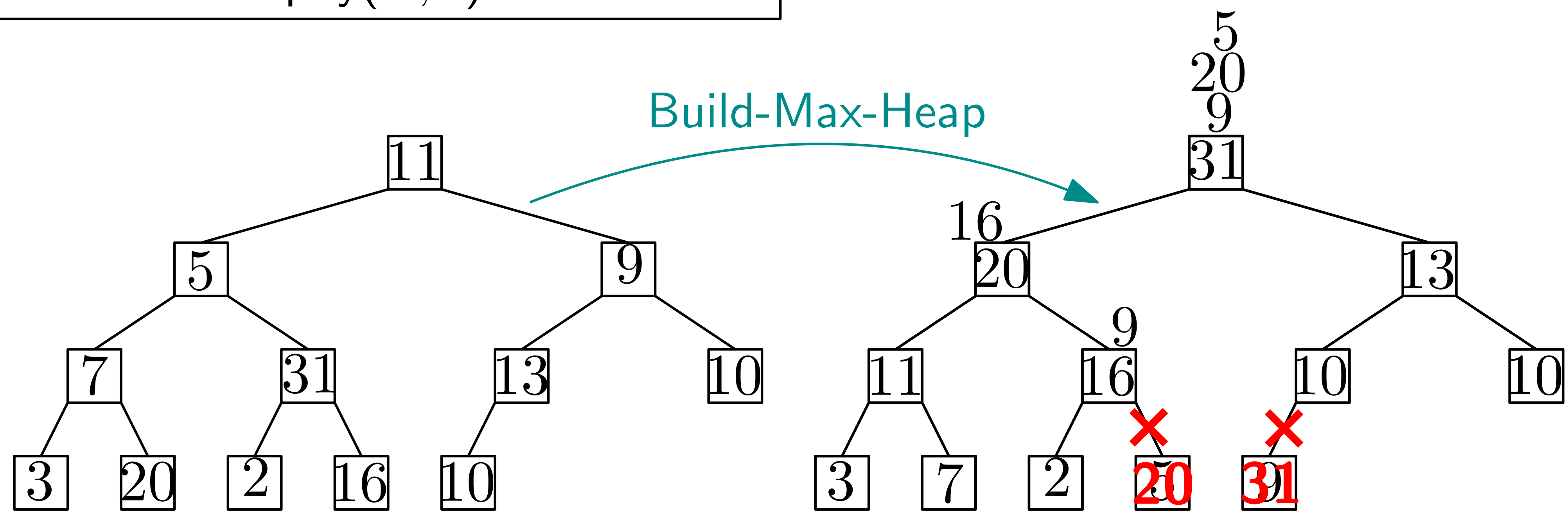
Heapsort

Heapsort(A)
  Build-Max-Heap(A)
  for $i = A.length$ downto 2
    swap $A[1]$ and $A[i]$
    $A.heap\text{-}size = A.heap\text{-}size - 1$
    Max-Heapify($A, 1$)



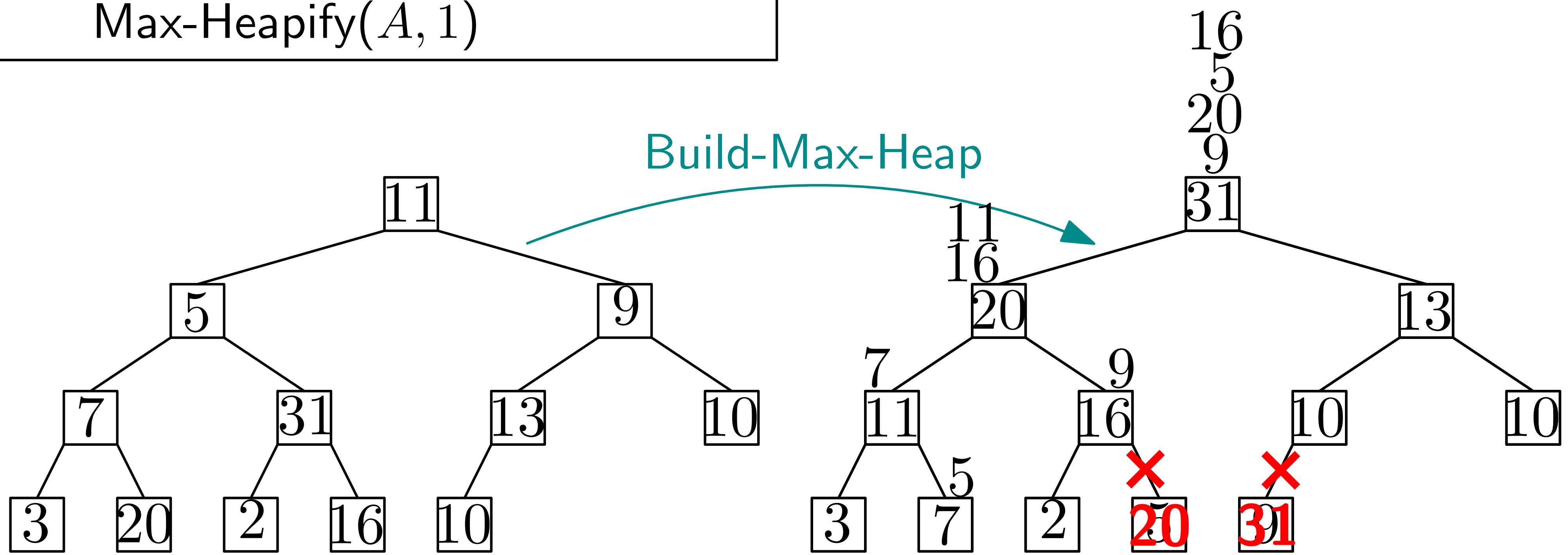
Heapsort

Heapsort(A)
  Build-Max-Heap(A)
  for $i = A.length$ downto 2
    swap $A[1]$ and $A[i]$
    $A.heap\text{-}size = A.heap\text{-}size - 1$
    Max-Heapify($A, 1$)



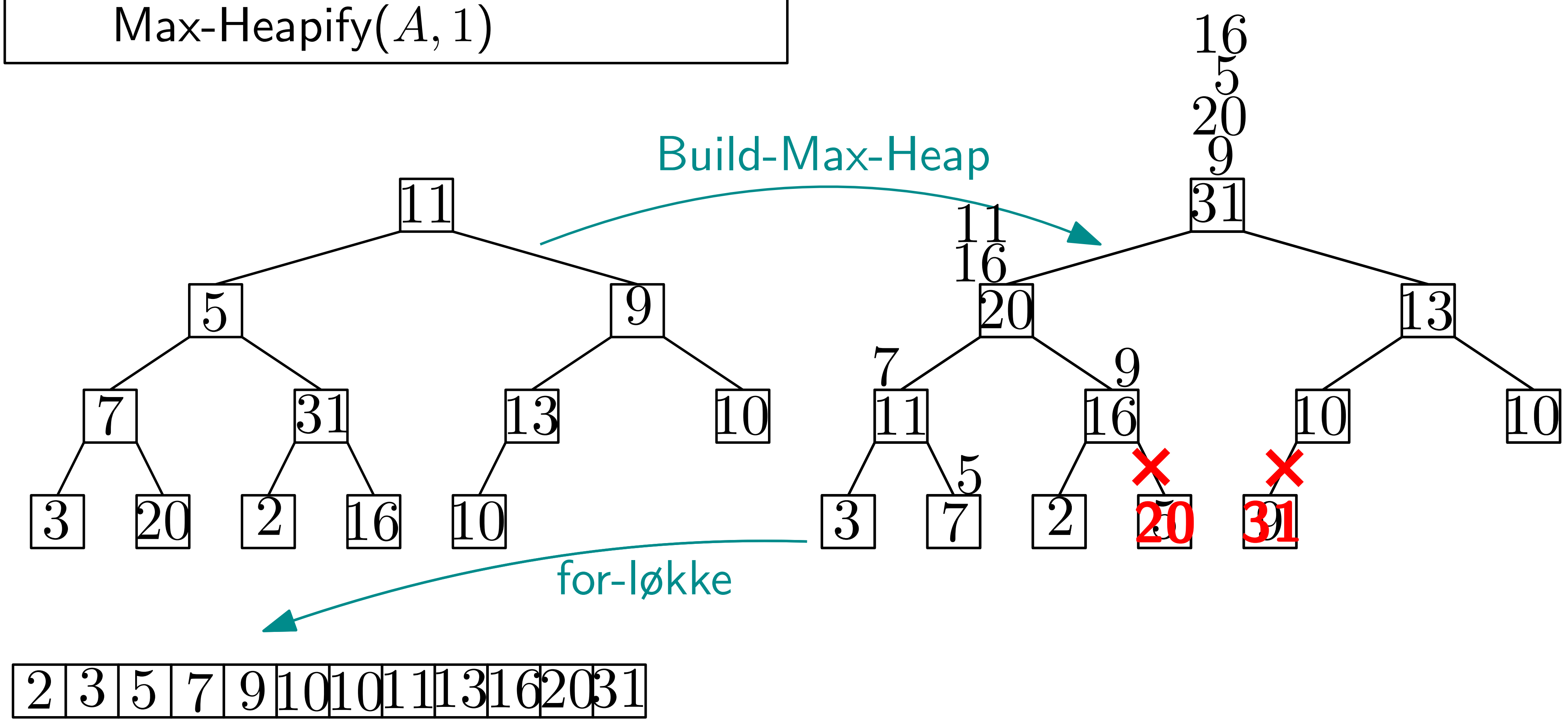
Heapsort

Heapsort(A)
  Build-Max-Heap(A)
  for $i = A.length$ downto 2
    swap $A[1]$ and $A[i]$
    $A.heap\text{-}size = A.heap\text{-}size - 1$
    Max-Heapify($A, 1$)



Heapsort

Heapsort(A)
    Build-Max-Heap(A)
    for $i = A.length$ downto 2
        swap $A[1]$ and $A[i]$
        $A.heap\text{-}size = A.heap\text{-}size - 1$
        Max-Heapify($A, 1$)



Køretid

Heapsort(A)	Tid	Gange
Build-Max-Heap(A)	$\Theta(n)$	1
for $i = A.length$ downto 2 swap $A[1]$ and $A[i]$ $A.heap\text{-}size = A.heap\text{-}size - 1$	$\Theta(1)$	n
Max-Heapify($A, 1$)	$O(\log n)$	n

Køretid

Heapsort(A)	Tid	Gange
Build-Max-Heap(A)	$\Theta(n)$	1
for $i = A.length$ downto 2 swap $A[1]$ and $A[i]$ $A.heap\text{-}size = A.heap\text{-}size - 1$	$\Theta(1)$	n
Max-Heapify($A, 1$)	$O(\log n)$	n

Køretid: $T(n) = O(n \cdot 1) + O(1 \cdot n) + O(\log n \cdot n) = O(n \log n)$.

Der gælder også $T(n) = \Omega(n \log n)$, så $T(n) = \Theta(n \log n)$.

Køretid

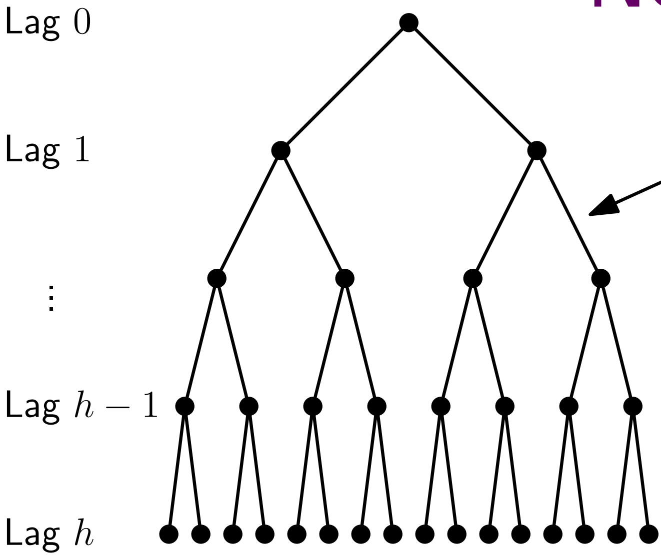
Heapsort(A)	Tid	Gange
Build-Max-Heap(A)	$\Theta(n)$	1
for $i = A.length$ downto 2 swap $A[1]$ and $A[i]$ $A.heap\text{-}size = A.heap\text{-}size - 1$	$\Theta(1)$	n
Max-Heapify($A, 1$)	$O(\log n)$	n

Køretid: $T(n) = O(n \cdot 1) + O(1 \cdot n) + O(\log n \cdot n) = O(n \log n)$.

Der gælder også $T(n) = \Omega(n \log n)$, så $T(n) = \Theta(n \log n)$.

Ekstra plads: $\Theta(1)$.
Bemærk: Merge-Sort bruger $\Theta(n)$ ekstra plads!

Nedre grænse for køretid



$n = 2^{h+1} - 1$

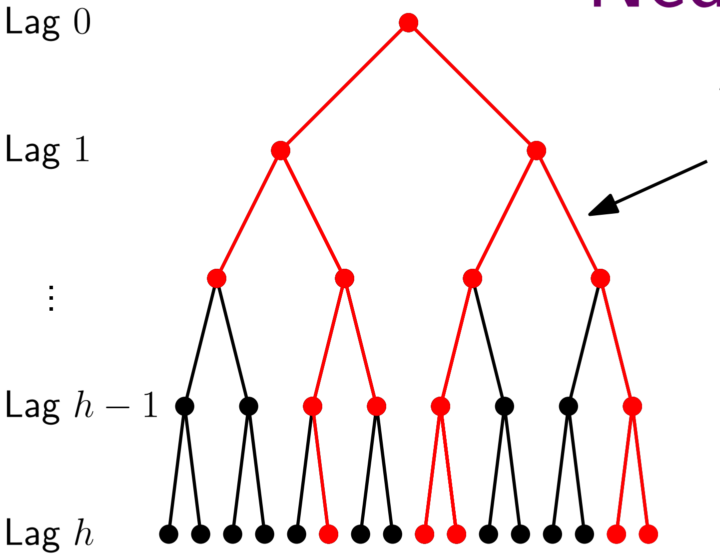
Her: $h = 4$, $n = 2^5 - 1 = 31$.

Nøgleværdier: $\{1, 2, \ldots, n\}$, én af hver.

Største $n/2$ værdier kaldes *store*.

```
Heapsort(A)
  Build-Max-Heap(A)
  for i = A.length downto 2
    swap A[1] and A[i]
    A.heap-size = A.heap-size - 1
    Max-Heapify(A, 1)
```

Nedre grænse for køretid



$n = 2^{h+1} - 1$

Her: $h = 4$, $n = 2^5 - 1 = 31$.

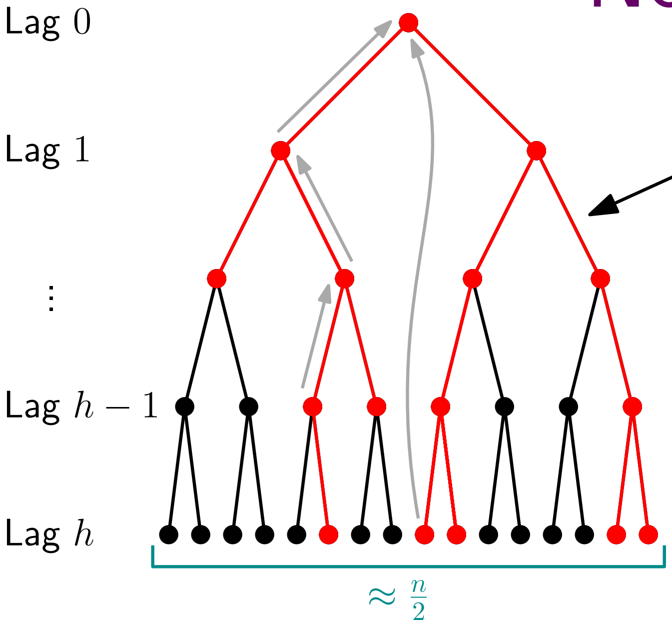
Nøgleværdier: $\{1, 2, \ldots, n\}$, én af hver.

Største $n/2$ værdier kaldes *store*.

Observation 1: Efter Build-Max-Heap danner de store værdier et (sammenhængende) træ.

```
Heapsort(A)
   Build-Max-Heap(A)
   for i = A.length downto 2
      swap A[1] and A[i]
      A.heap-size = A.heap-size − 1
      Max-Heapify(A, 1)
```

Nedre grænse for køretid




```
Heapsort(A)
  Build-Max-Heap(A)
  for i = A.length downto 2
    swap A[1] and A[i]
    A.heap-size = A.heap-size − 1
    Max-Heapify(A, 1)
```

$n = 2^{h+1} - 1$

Her: $h = 4$, $n = 2^5 - 1 = 31$.

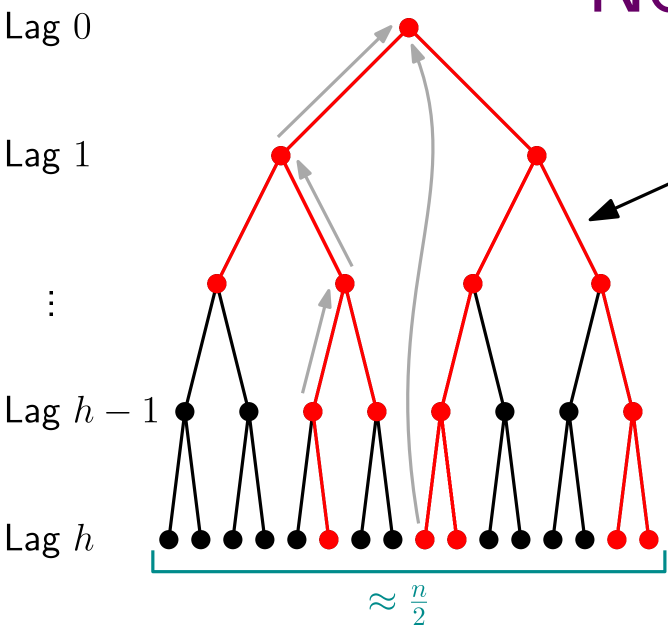
Nøgleværdier: $\{1, 2, \ldots, n\}$, én af hver.

Største $n/2$ værdier kaldes *store*.

Observation 1: Efter Build-Max-Heap danner de store værdier et (sammenhængende) træ.

Observation 2: Enhver stor værdi der starter i lag $h-1$, bevæger sig til roden og bliver fjernet i løbet af $n/2$ iterationer. Dvs. $\Omega(\log n)$ skridt op.

Nedre grænse for køretid



$$n = 2^{h+1} - 1$$

Her: $h = 4$, $n = 2^5 - 1 = 31$.

Nøgleværdier: $\{1, 2, \ldots, n\}$, én af hver.

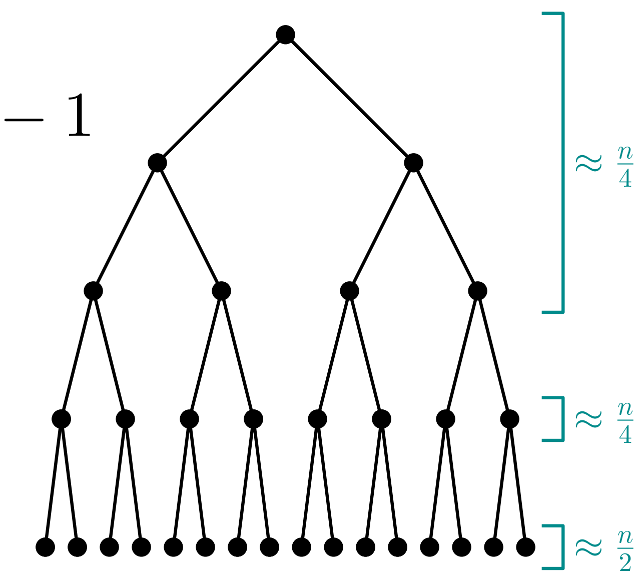
Største $n/2$ værdier kaldes *store*.

Observation 1: Efter Build-Max-Heap danner de store værdier et (sammenhængende) træ.

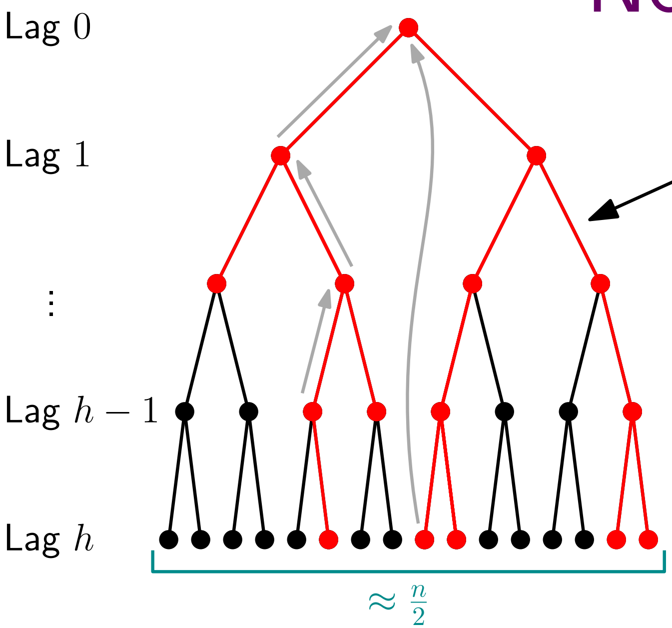
Observation 2: Enhver stor værdi der starter i lag $h-1$, bevæger sig til roden og bliver fjernet i løbet af $n/2$ iterationer. Dvs. $\Omega(\log n)$ skridt op.

```
Heapsort(A)
  Build-Max-Heap(A)
  for i = A.length downto 2
    swap A[1] and A[i]
    A.heap-size = A.heap-size − 1
    Max-Heapify(A, 1)
```

Observation 3: Mindst $n/4$ store værdier i lag $h-1$ og h tilsammen.



Nedre grænse for køretid



$n = 2^{h+1} - 1$

Her: $h = 4$, $n = 2^5 - 1 = 31$.

Nøgleværdier: $\{1, 2, \ldots, n\}$, én af hver.

Største $n/2$ værdier kaldes *store*.

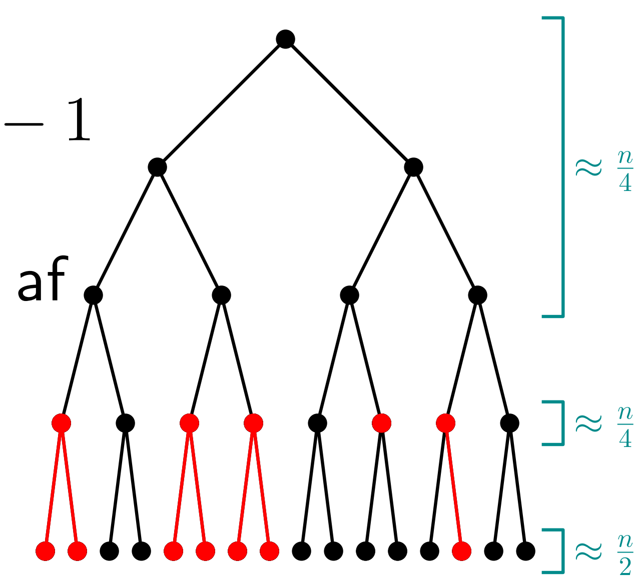
Observation 1: Efter Build-Max-Heap danner de store værdier et (sammenhængende) træ.

Observation 2: Enhver stor værdi der starter i lag $h-1$, bevæger sig til roden og bliver fjernet i løbet af $n/2$ iterationer. Dvs. $\Omega(\log n)$ skridt op.

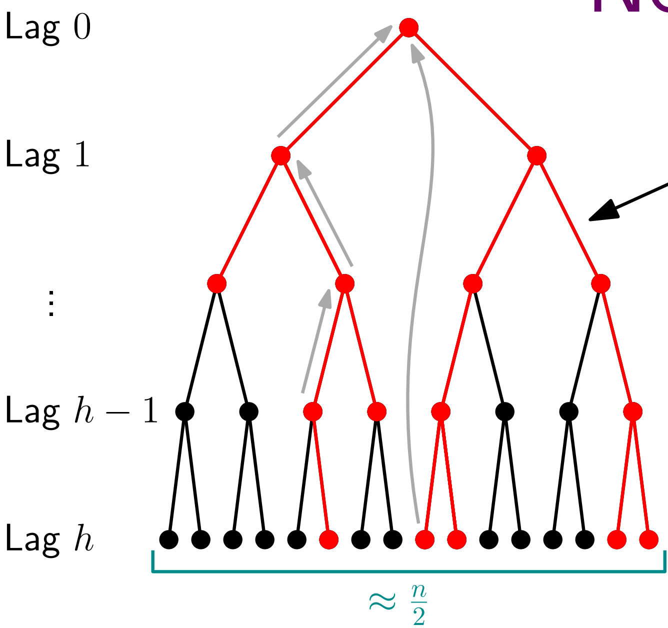
```
Heapsort(A)
  Build-Max-Heap(A)
  for i = A.length downto 2
    swap A[1] and A[i]
    A.heap-size = A.heap-size − 1
    Max-Heapify(A, 1)
```

Observation 3: Mindst $n/4$ store værdier i lag $h-1$ og h tilsammen.

Observation 4: Lag $h-1$ indeholder mindst $1/3$ af alle store værdier i lag $h-1$ og $h \Rightarrow$ mindst $n/12 = \Omega(n)$ store værdier i lag $h-1$.



Nedre grænse for køretid



$n = 2^{h+1} - 1$

Her: $h = 4$, $n = 2^5 - 1 = 31$.

Nøgleværdier: $\{1, 2, \ldots, n\}$, én af hver.

Største $n/2$ værdier kaldes *store*.

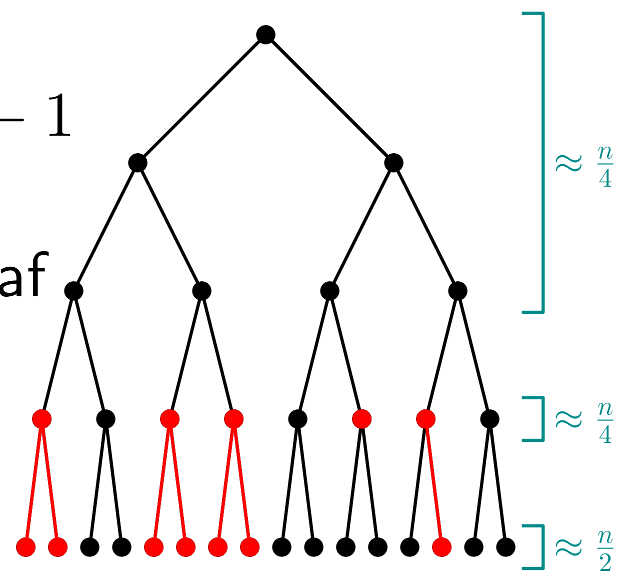
Observation 1: Efter Build-Max-Heap danner de store værdier et (sammenhængende) træ.

Observation 2: Enhver stor værdi der starter i lag $h-1$, bevæger sig til roden og bliver fjernet i løbet af $n/2$ iterationer. Dvs. $\Omega(\log n)$ skridt op.

```
Heapsort(A)
   Build-Max-Heap(A)
   for i = A.length downto 2
      swap A[1] and A[i]
      A.heap-size = A.heap-size − 1
      Max-Heapify(A, 1)
```

Observation 3: Mindst $n/4$ store værdier i lag $h-1$ og h tilsammen.

Observation 4: Lag $h-1$ indeholder mindst $1/3$ af alle store værdier i lag $h-1$ og h $\Rightarrow$ mindst $n/12 = \Omega(n)$ store værdier i lag $h-1$.



Konklusion: Observation 2+4 giver $\Omega(n \log n)$ tid.

Hvor meget forstod du af beviset?

socrative.com $\rightarrow$ Student login,
Room name: ABRAHAMSEN3464

Det hele.
A

Det meste.
B

Noget.
C

Kun en lille smule.
D

Ingenting.
E