$$\Big($$

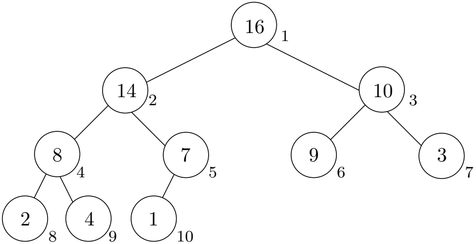
Heaps

An array $A[1..n]$ is called a *heap* if for all $i \geq 1$,

$$A[i] \geq A[2i] \qquad \text{if } 2i \leq n$$

and

$$A[i] \geq A[2i+1] \qquad \text{if } 2i+1 \leq n$$



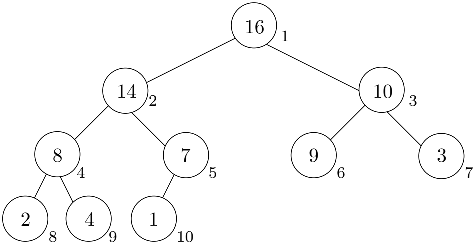
16	14	10	8	7	9	3	2	4	1

Root of the tree: $A[1]$

Consider a node with index i:

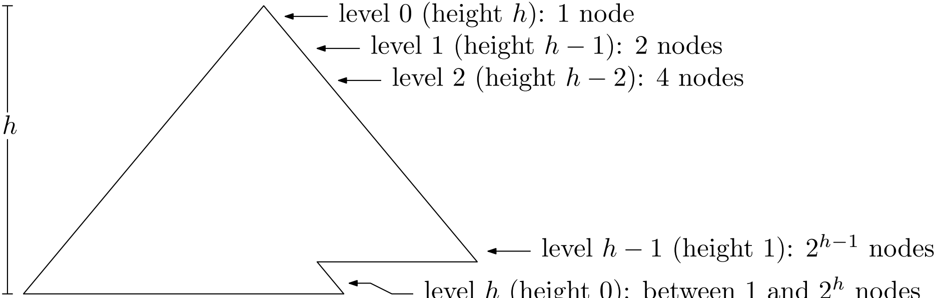
- Parent of i has index $\lfloor i/2 \rfloor$: $\text{parent}(i) = \lfloor i/2 \rfloor$
- Left child of i has index $2i$: $\text{left}(i) = 2i$
- Right child of i has index $2i+1$: $\text{right}(i) = 2i+1$

$A[1..n]$ is a heap if for all $1 < i \leq n$, $A[\text{parent}(i)] \geq A[i]$.

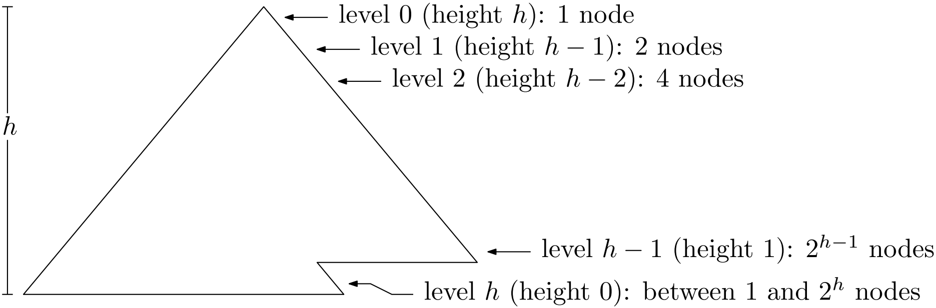


16	14	10	8	7	9	3	2	4	1

What is the height of a heap?

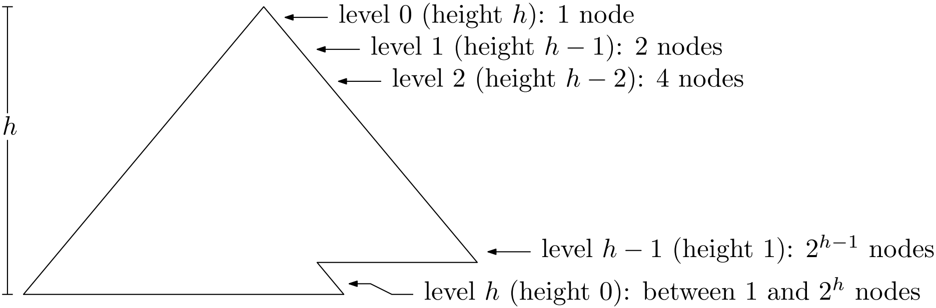


What is the height of a heap?



$$1 + 2 + 2^2 + \ldots + 2^{h-1} + 1 \leq n \leq 1 + 2 + 2^2 + \ldots + 2^{h-1} + 2^h$$

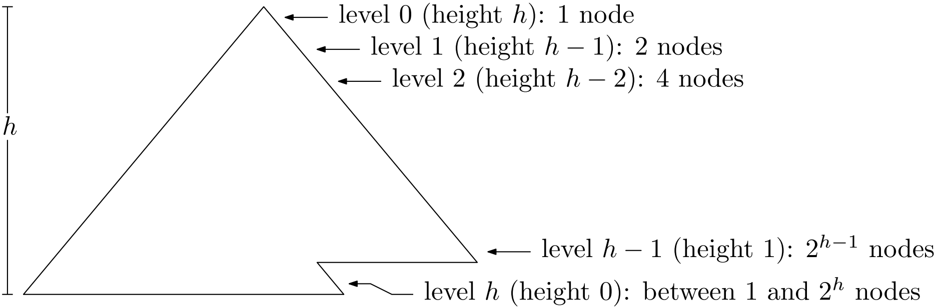
What is the height of a heap?



$$1 + 2 + 2^2 + \ldots + 2^{h-1} + 1 \leq n \leq 1 + 2 + 2^2 + \ldots + 2^{h-1} + 2^h$$

$$\left(2^h - 1\right) + 1 \leq n \leq 2^{h+1} - 1$$

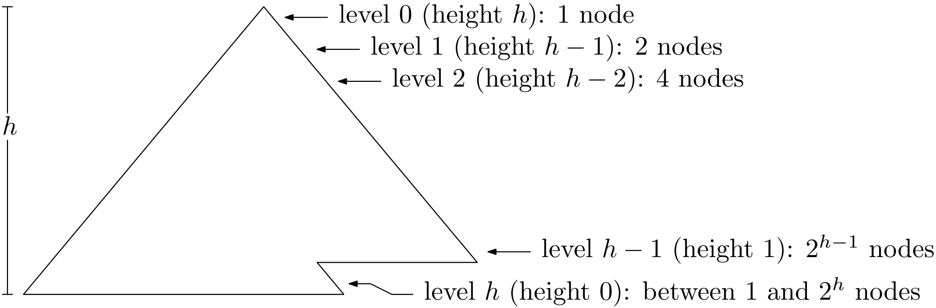
What is the height of a heap?



$$1 + 2 + 2^2 + \ldots + 2^{h-1} + 1 \leq n \leq 1 + 2 + 2^2 + \ldots + 2^{h-1} + 2^h$$

$$\left(2^h - 1\right) + 1 \leq n \leq 2^{h+1} - 1 < 2^{h+1}$$

What is the height of a heap?

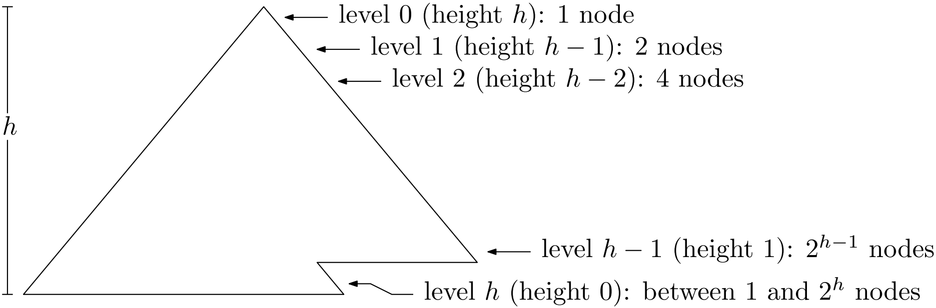


$$1+2+2^2+\ldots+2^{h-1}+1 \leq n \leq 1+2+2^2+\ldots+2^{h-1}+2^h$$

$$\left(2^h-1\right)+1 \leq n \leq 2^{h+1}-1 < 2^{h+1}$$

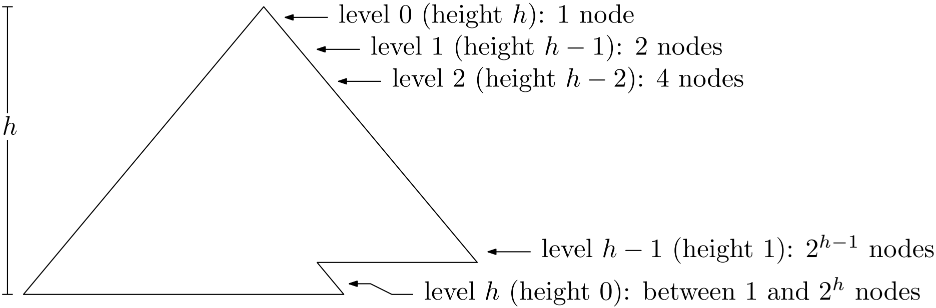
$$2^h \leq n < 2^{h+1}$$

What is the height of a heap?



$$
\begin{aligned}
1+2+2^2+\ldots+2^{h-1}+1 \leq\, & n \leq 1+2+2^2+\ldots+2^{h-1}+2^h \\
\left(2^h-1\right)+1 \leq\, & n \leq 2^{h+1}-1 < 2^{h+1} \\
2^h \leq\, & n < 2^{h+1} \\
h \leq\, & \log_2(n) < h+1
\end{aligned}
$$

What is the height of a heap?



$$
\begin{aligned}
1 + 2 + 2^2 + \ldots + 2^{h-1} + 1 &\leq n \leq 1 + 2 + 2^2 + \ldots + 2^{h-1} + 2^h \\
\left(2^h - 1\right) + 1 &\leq n \leq 2^{h+1} - 1 < 2^{h+1} \\
2^h &\leq n < 2^{h+1} \\
h &\leq \log_2(n) < h + 1 \\
h &= \lfloor \log_2(n) \rfloor
\end{aligned}
$$

From now on: we consider an array $A[1..N]$ with an integer n where $1 \leq n \leq N$.

$A[1..n]$ is a heap which contains n integers and $A[(n+1)..N]$ contains garbage.

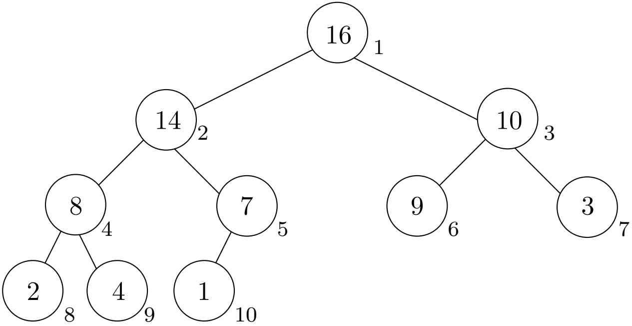
heap	garbage
1 ... n	$(n+1)$ ... N

1. Finding a maximum element

$$\text{Maximum}(A)\text{: returns } A[1]$$

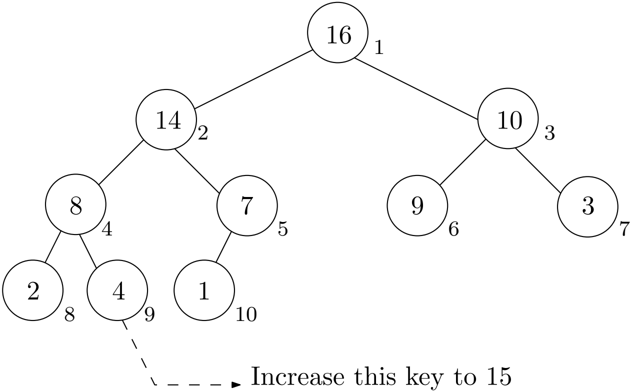
This takes $O(1)$ time.

2. Increase the value of $A[i]$ to x.



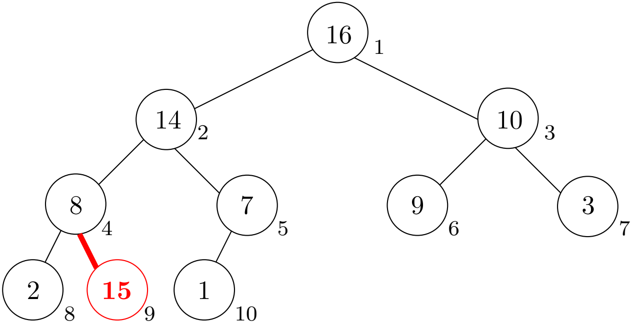
2. Increase the value of $A[i]$ to x.

Increase_key$(A, 9, 15)$



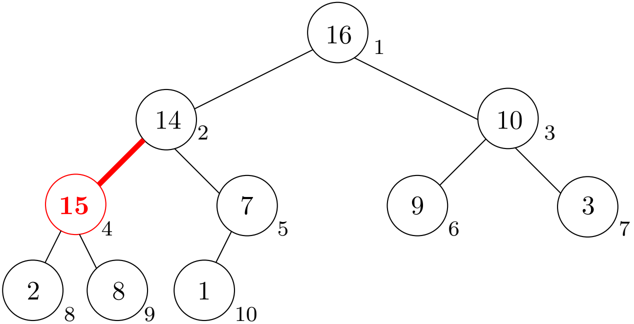
2. Increase the value of $A[i]$ to x.

$\text{Increase\_key}(A, 9, 15)$



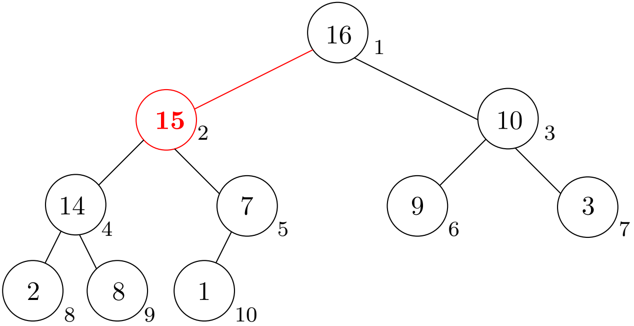
2. Increase the value of $A[i]$ to x.

Increase_key$(A, 9, 15)$



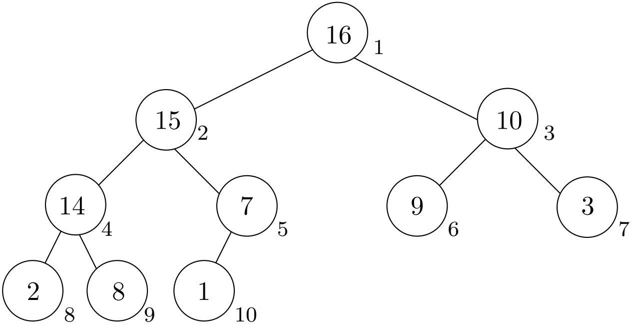
2. Increase the value of $A[i]$ to x.

$\text{Increase\_key}(A, 9, 15)$



2. Increase the value of $A[i]$ to x.

$\text{Increase\_key}(A, 9, 15)$



Algorithm 1 Increase_key(A, i, x)

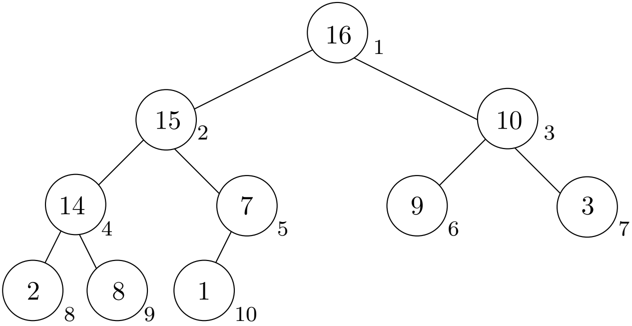
Require: $x \geq A[i]$

```
A[i] = x
// The following while-loop is sometimes called percolate.
while i > 1 and A[parent(i)] < A[i] do
    swap A[i] and A[parent(i)]
    i = parent(i)
end while
```

This takes $O(h) = O(\log(n))$ time.

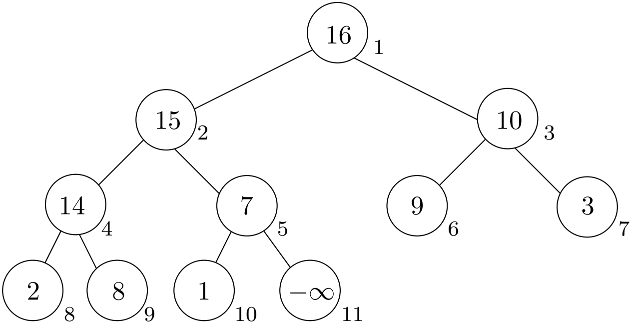
3. Insert a new value in A.

$\text{Insert}(A, 13)$



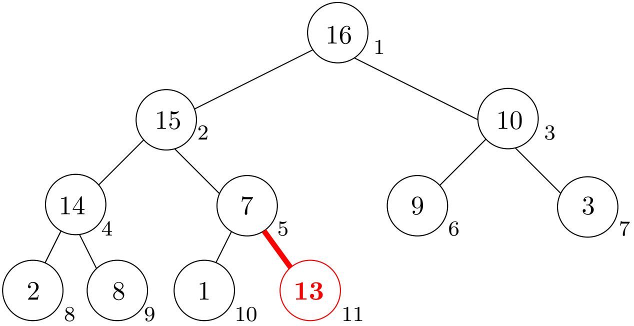
3. Insert a new value in A.

$\text{Insert}(A, 13)$



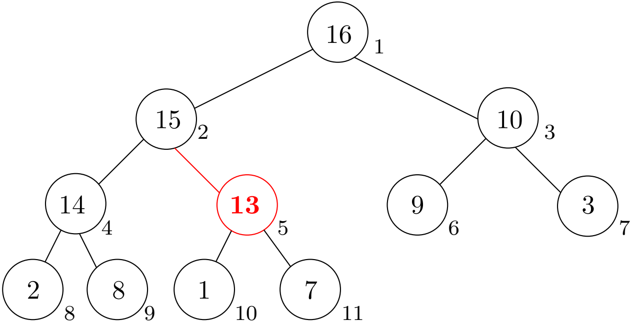
3. Insert a new value in A.

$\text{Insert}(A, 13)$



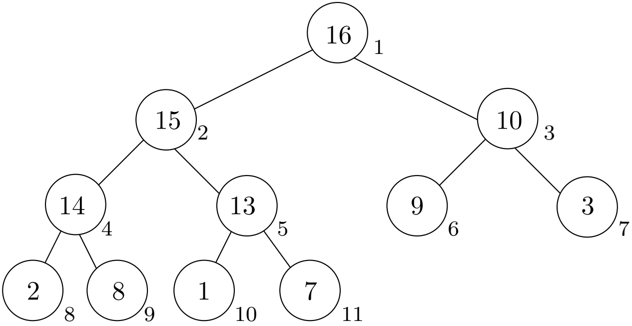
3. Insert a new value in A.

$\mathrm{Insert}(A, 13)$



3. Insert a new value in A.

$\text{Insert}(A, 13)$



Algorithm 2 Insert(A, x)

Require:

- $1 \leq n < N$
- and $A[1..n]$ is a heap.

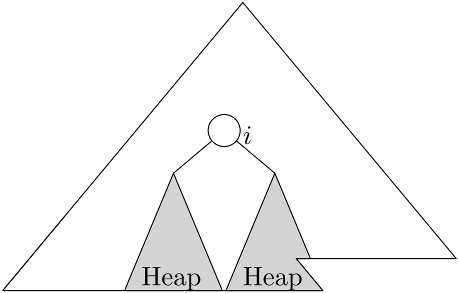
```
n = n + 1
A[n] = −∞
Increase_key(A, n, x)
```

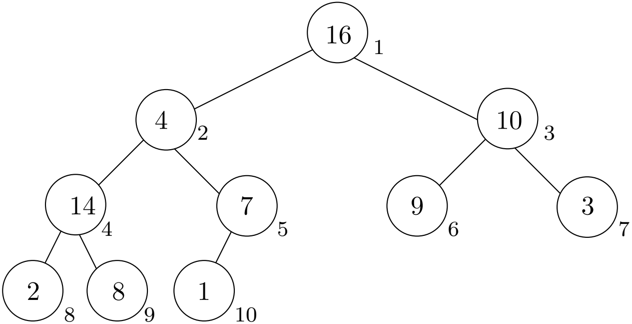
This takes $O(h) = O(\log(n))$ time.

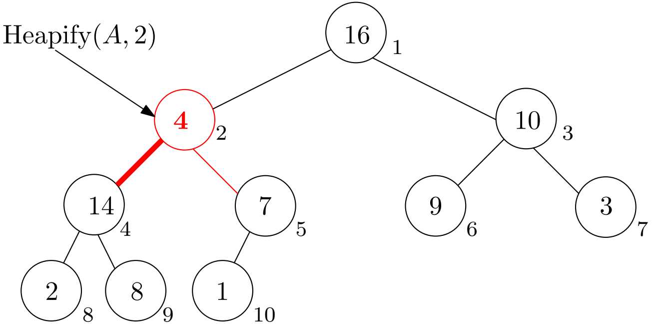
4. Heapify(A, i) (sometimes called *sift-down*): this operation assumes that

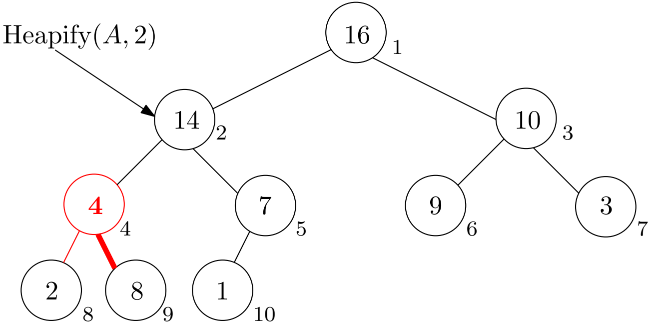
- $1 \leq i \leq n$,
- the subtree rooted at left(i) is a heap
- and the subtree rooted at right(i) is a heap.

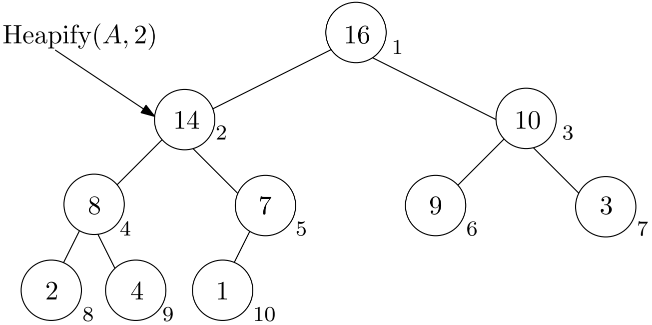
At termination, the subtree rooted at i is a heap.




16
1
4
2
10
3
14
4
7
5
9
6
3
7
2
8
8
9
1
10


Heapify$(A, 2)$
16
1
4
2
10
3
14
4
7
5
9
6
3
7
2
8
8
9
1
10


Heapify(A, 2)
16
1
14
2
10
3
4
4
7
5
9
6
3
7
2
8
8
9
1
10


Heapify$(A, 2)$
16
1
14
2
10
3
8
4
7
5
9
6
3
7
2
8
4
9
1
10

Algorithm 3 Heapify(A, i)

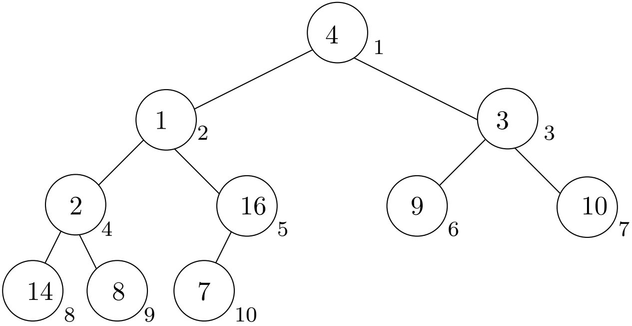
Require:

- $1 \leq i \leq n$,
- the subtree rooted at left(i) is a heap
- and the subtree rooted at right(i) is a heap.

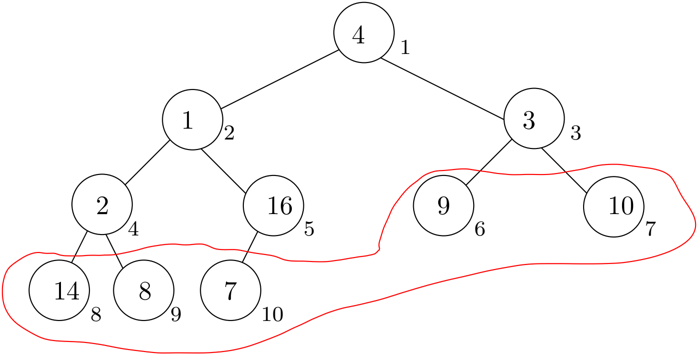
$\ell = \text{left}(i)$
$r = \text{right}(i)$
if $\ell \leq n$ and $A[\ell] > A[i]$ **then**
    $\max = \ell$
else
    $\max = i$
end if
if $r \leq n$ and $A[r] > A[\max]$ **then**
    $\max = r$
end if
if $\max \neq i$ **then**
    swap $A[i]$ and $A[\max]$
    Heapify($A, \max$)
end if

This takes $O(h) = O(\log(n))$ time.

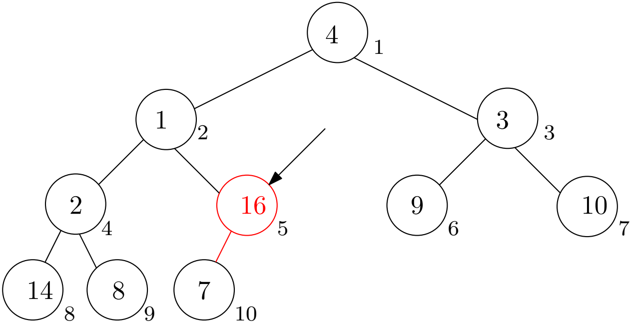
5. Build_heap(A)



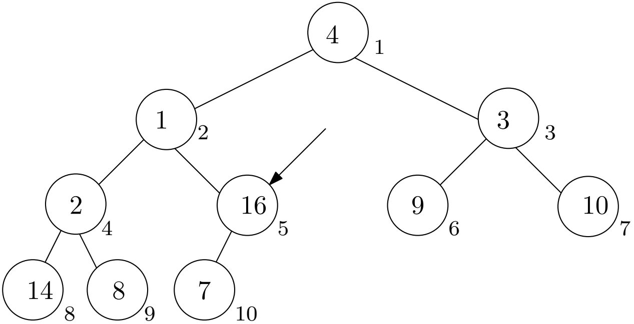
5. Build_heap(A)



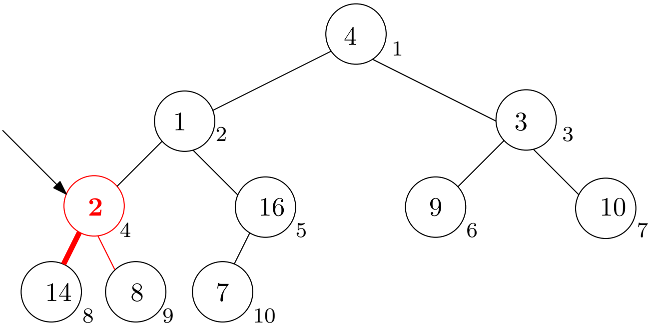
5. Build_heap(A)



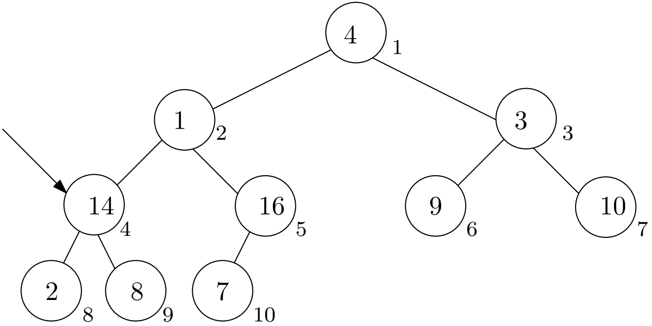
5. Build_heap(A)



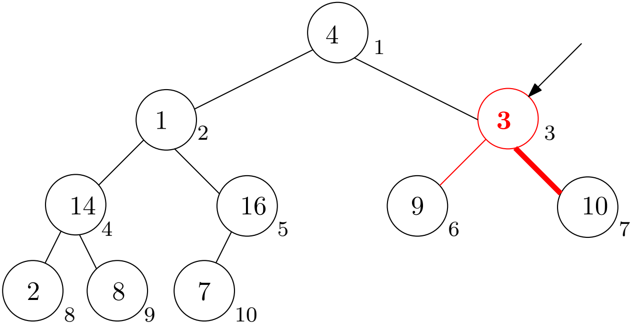
5. Build_heap(A)



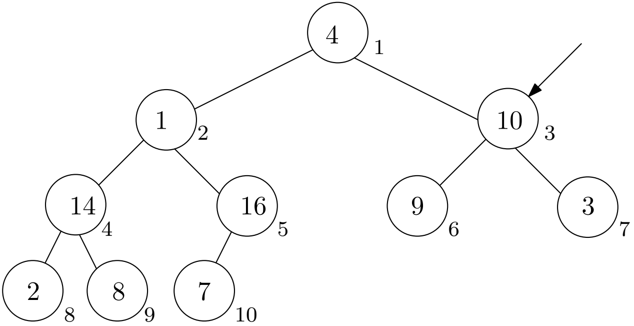
5. Build_heap(A)



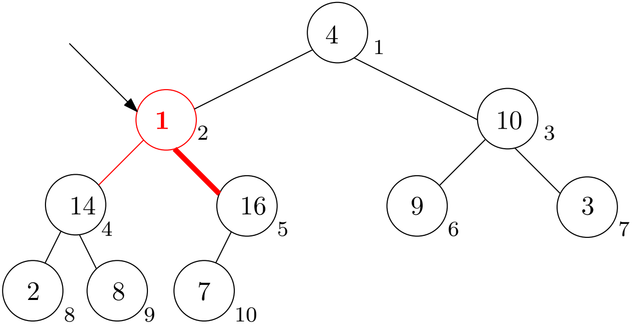
5. Build_heap(A)



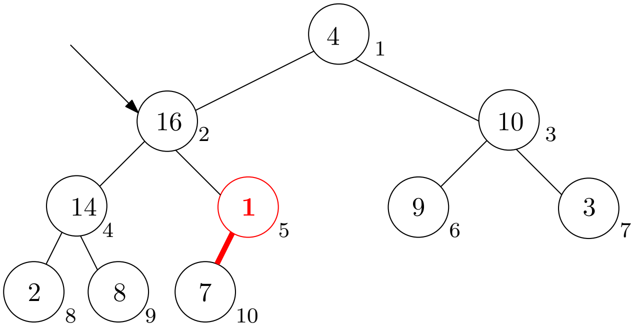
5. Build_heap(A)



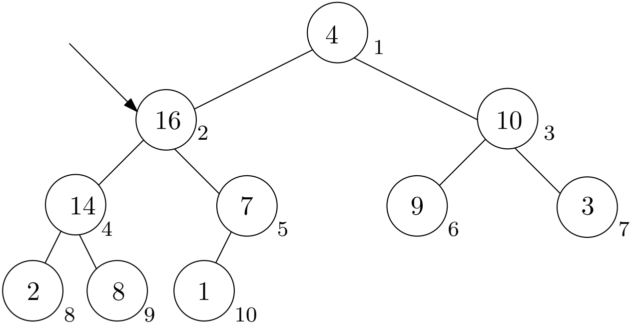
5. Build_heap(A)



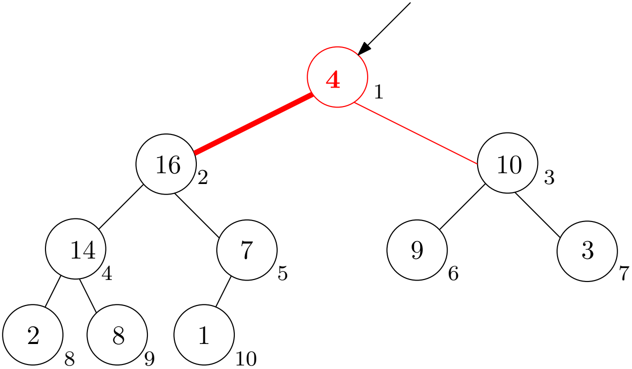
5. Build_heap(A)



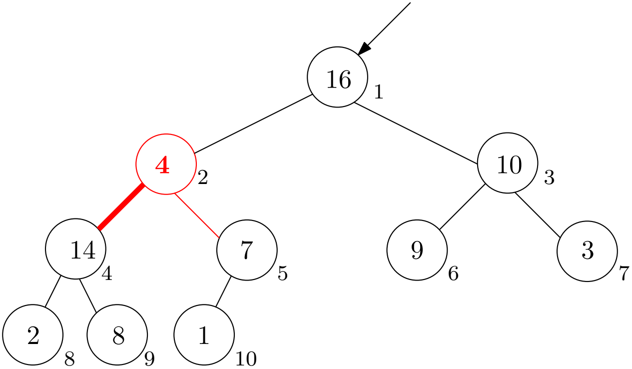
5. Build_heap(A)



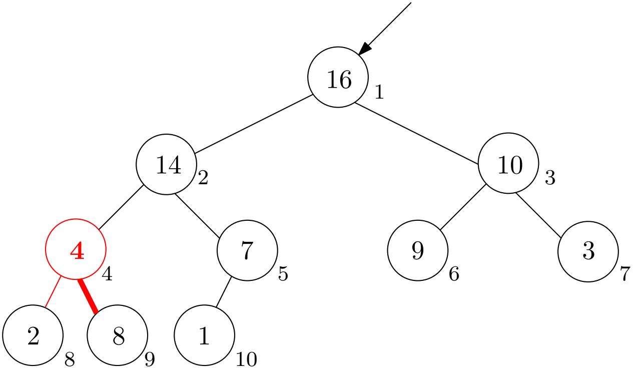
5. Build_heap(A)



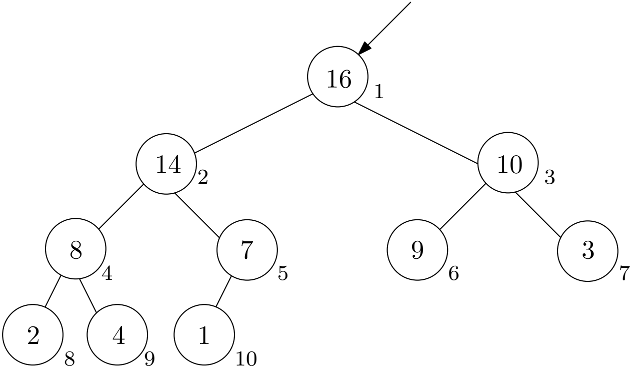
5. Build_heap(A)



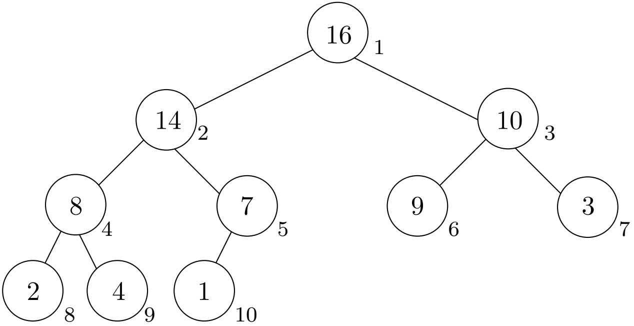
5. Build_heap(A)



5. Build_heap(A)



5. Build_heap(A)



Algorithm 4 Build_heap(A)

```
for i = ⌊n/2⌋ to 1 do
    Heapify(A, i)
end for
```

Algorithm 4 Build_heap(A)

for $i = \lfloor n/2 \rfloor$ to 1 **do**
  Heapify(A, i)
end for

How many steps to build a heap?

$$\begin{gathered}\text{There is } 2^0 \text{ node at height } h. \\ \text{There are } 2^1 \text{ nodes at height } h-1. \\ \text{There are } 2^2 \text{ nodes at height } h-2. \\ \vdots \\ \text{There are } 2^{h-1} \text{ nodes at height } 1.\end{gathered}$$

Algorithm 4 Build_heap(A)

```
for i = ⌊n/2⌋ to 1 do
    Heapify(A, i)
end for
```

How many steps to build a heap?

$$\begin{gathered}\text{There is } 2^0 \text{ node at height } h. \\ \text{There are } 2^1 \text{ nodes at height } h-1. \\ \text{There are } 2^2 \text{ nodes at height } h-2. \\ \vdots \\ \text{There are } 2^{h-1} \text{ nodes at height } 1.\end{gathered}$$

$$\begin{aligned} & h \cdot 2^0 + (h-1) \cdot 2^1 + (h-2) \cdot 2^2 + \ldots + 1 \cdot 2^{h-1} \\ = & \sum_{i=0}^{h-1} (h-i) \cdot 2^i \end{aligned}$$

$$
\begin{aligned}
& \sum_{i=0}^{h-1} (h-i) \cdot 2^i \\
= \;& \sum_{i=0}^{h-1} h \cdot 2^i - \sum_{i=0}^{h-1} i \cdot 2^i \\
= \;& h \sum_{i=0}^{h-1} 2^i - \sum_{i=0}^{h-1} i \cdot 2^i \\
= \;& h\left(2^h - 1\right) - \left((h-2) \cdot 2^h + 2\right) \\
= \;& 2 \cdot 2^h - h - 2 \\
= \;& O(n)
\end{aligned}
$$

6. Extract max A

Algorithm 5 Extract_max(A)

max $= A[1]$
$A[1] = A[n]$
$n = n - 1$
Heapify$(A, 1)$
return max

This takes $O(h) = O(\log(n))$ time.

We have discussed max-heaps.

There is their symmetric counterpart: min-heaps,

where $A[\text{parent}(i)] \leq A[i]$ $(1 < i \leq n)$.

$$\Bigg)$$