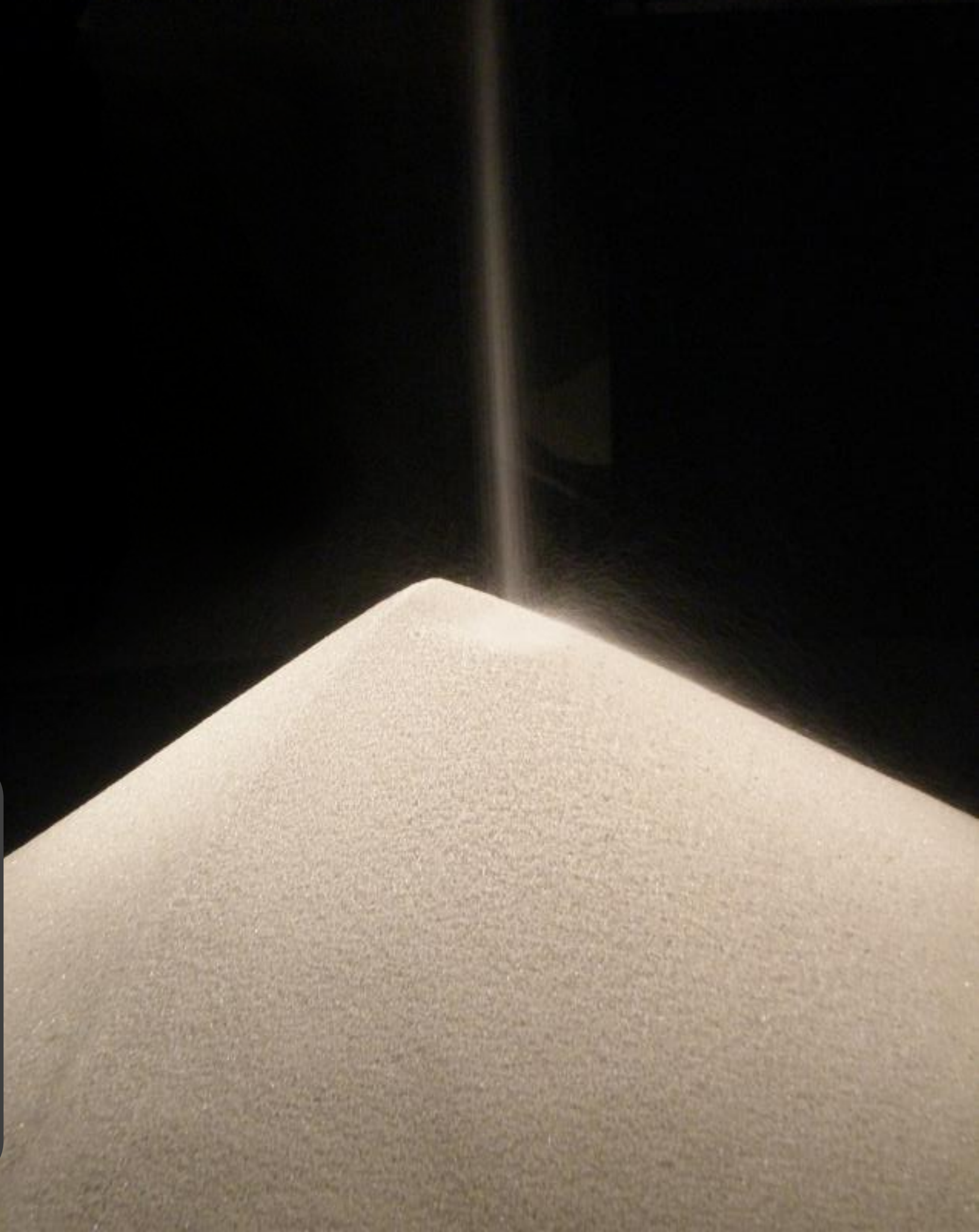
Carleton University

Alina Shaikhet

COMP 2402

Heaps

part 2

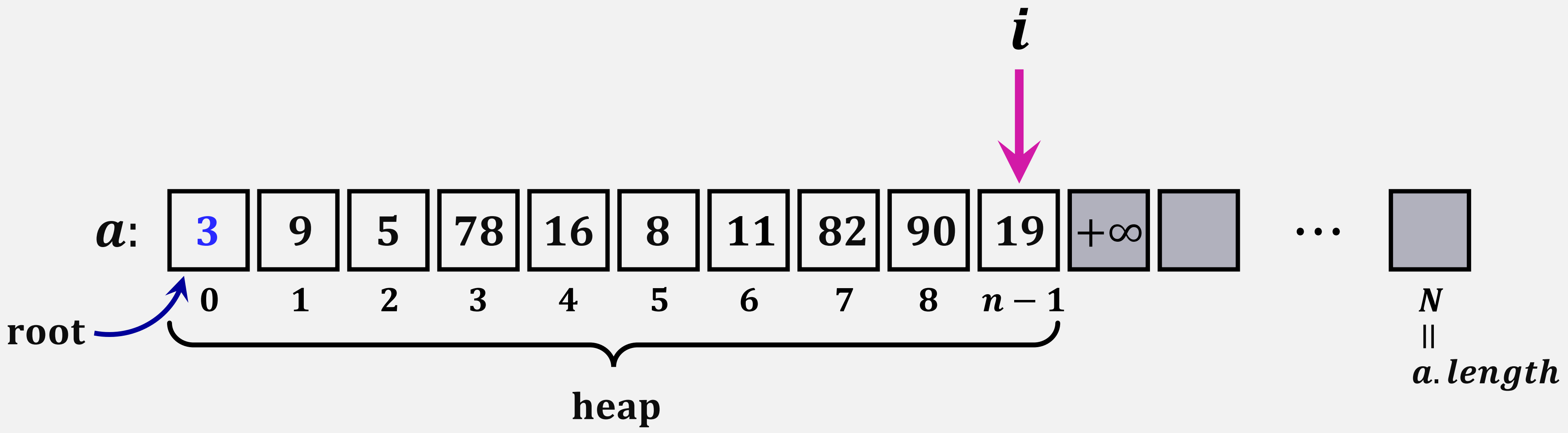


HeapSort

HeapSort is an in-place sorting algorithm

Given: min-heap as array $\boldsymbol{a}$ of $\boldsymbol{n}$ numbers

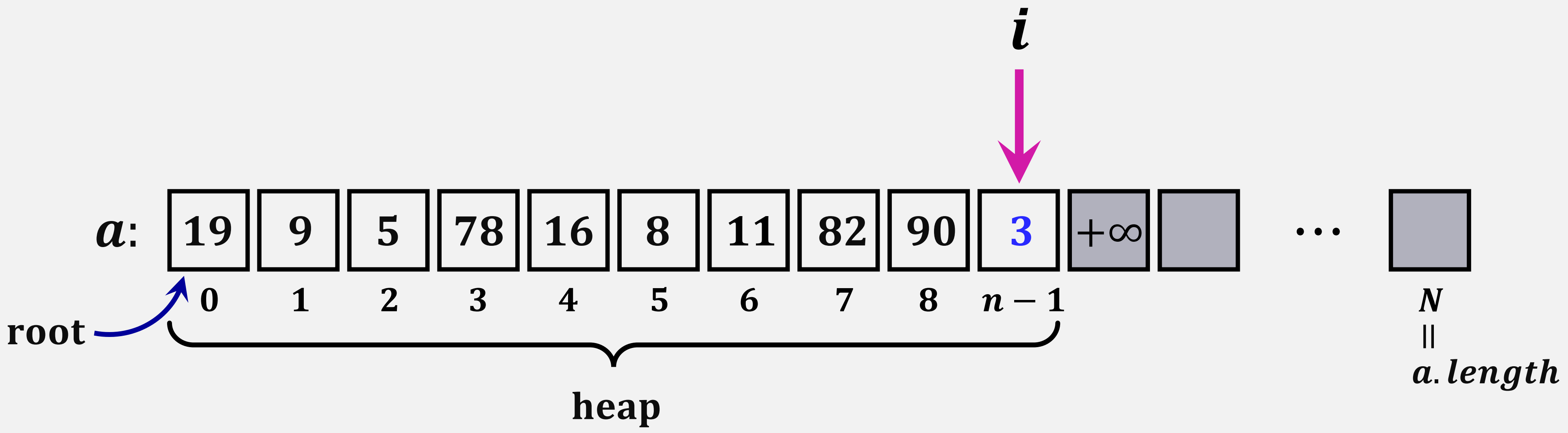
Goal: array $\boldsymbol{a}$, containing the same elements but in sorted order.



HeapSort

Given: min-heap as array a of n numbers

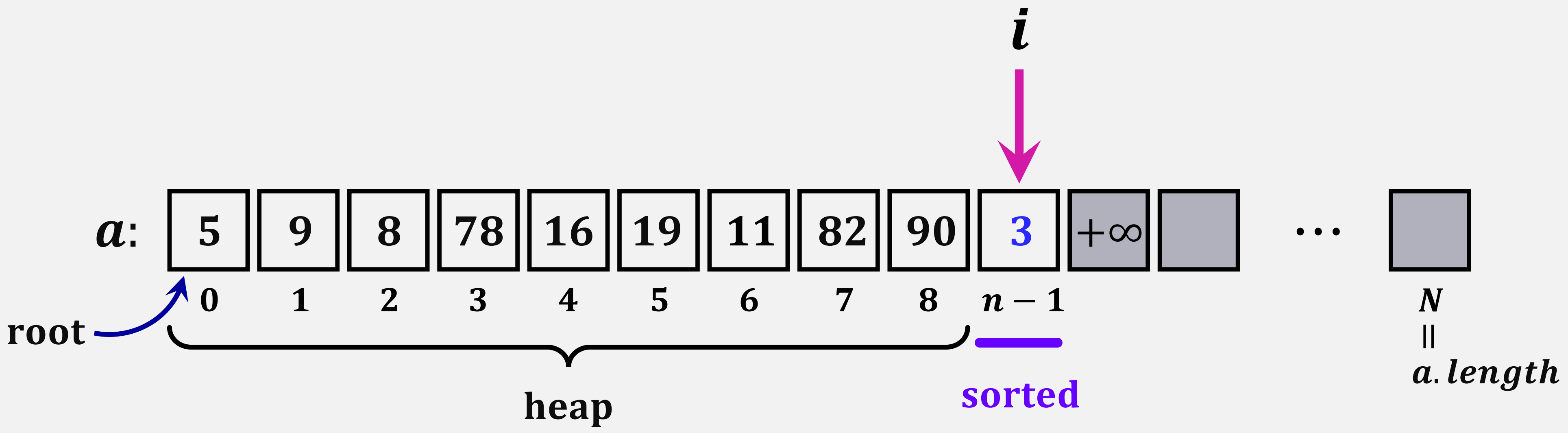
Goal: array a, containing the same elements but in sorted order.



HeapSort

Given: min-heap as array a of n numbers

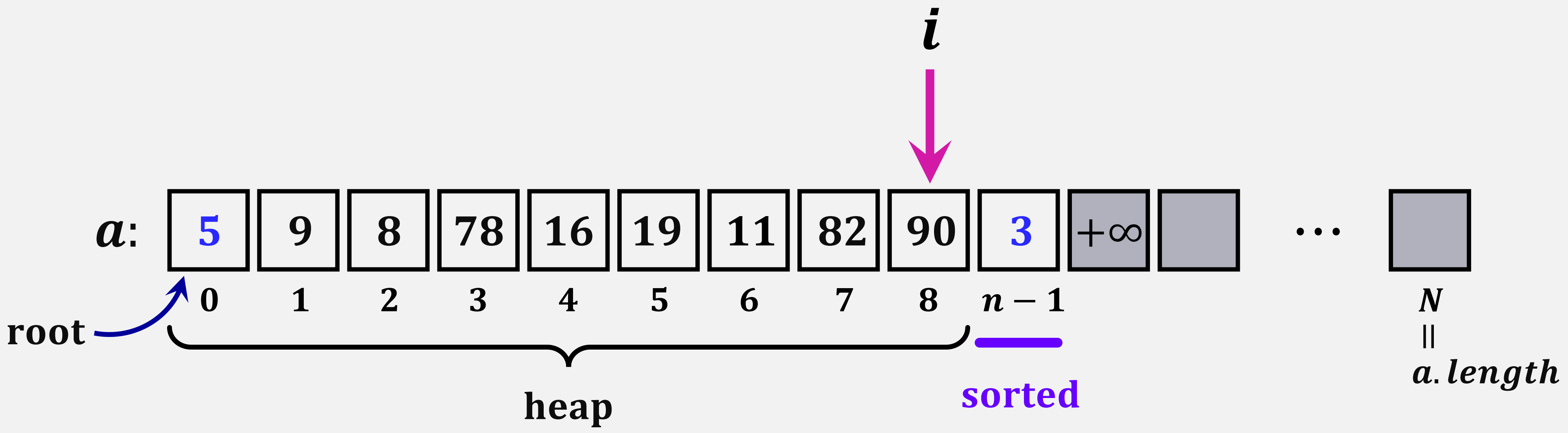
Goal: array a, containing the same elements but in sorted order.



HeapSort

Given: min-heap as array a of n numbers

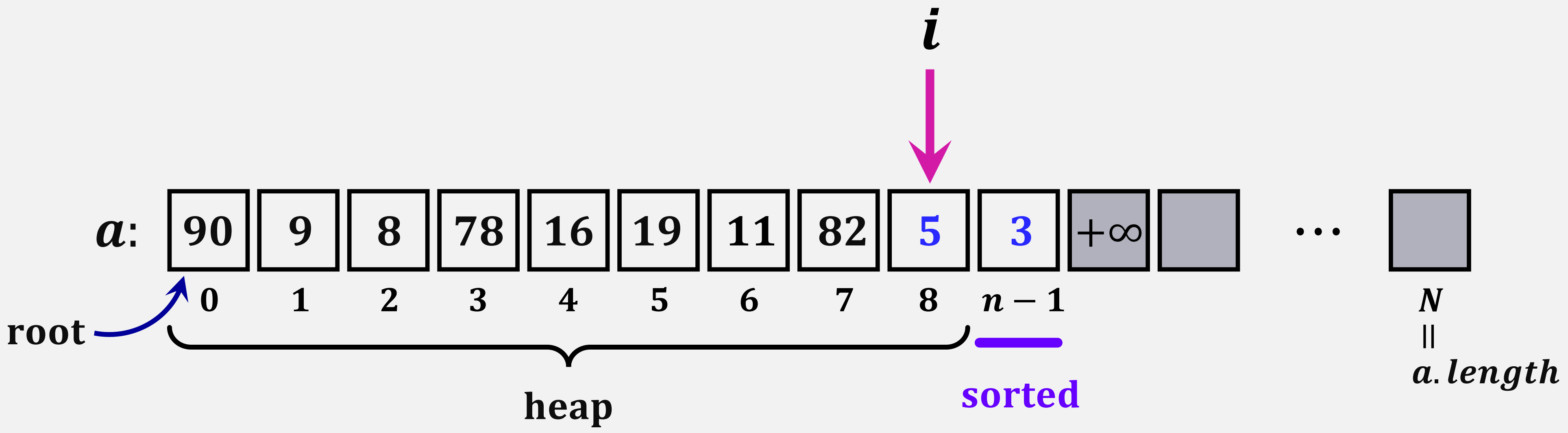
Goal: array a, containing the same elements but in sorted order.



HeapSort

Given: min-heap as array $\boldsymbol{a}$ of $\boldsymbol{n}$ numbers

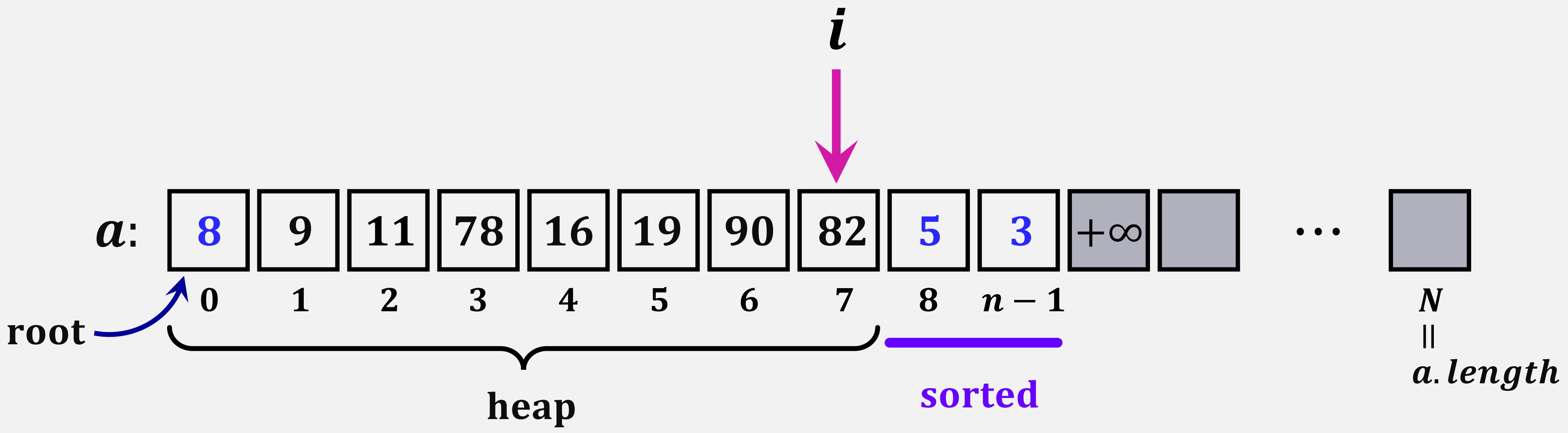
Goal: array $\boldsymbol{a}$, containing the same elements but in sorted order.



HeapSort

Given: min-heap as array a of n numbers

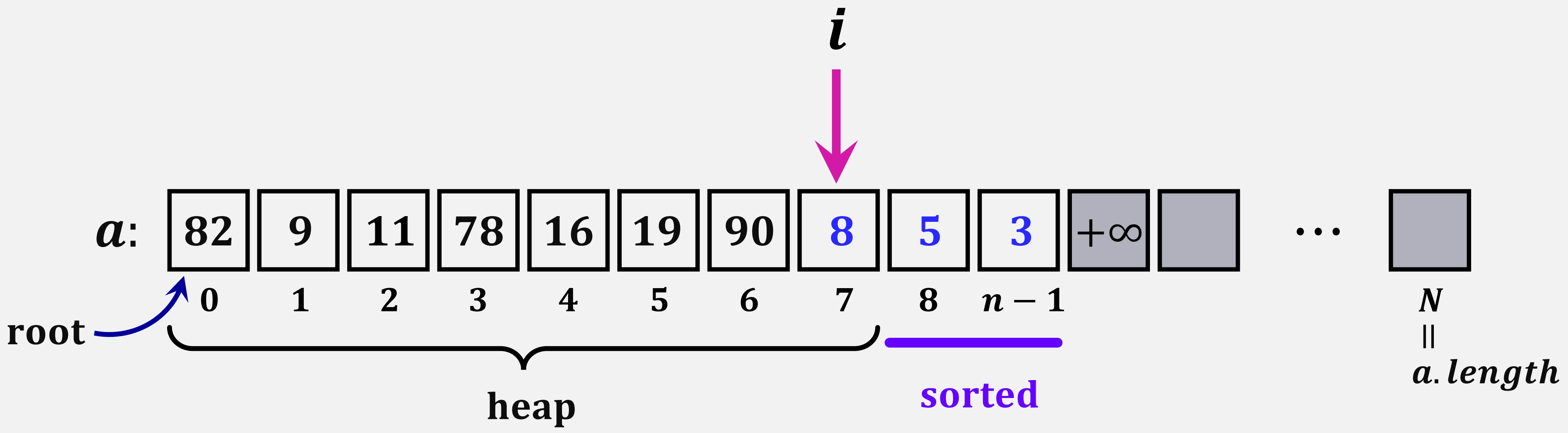
Goal: array a, containing the same elements but in sorted order.



HeapSort

Given: min-heap as array a of n numbers

Goal: array a, containing the same elements but in sorted order.

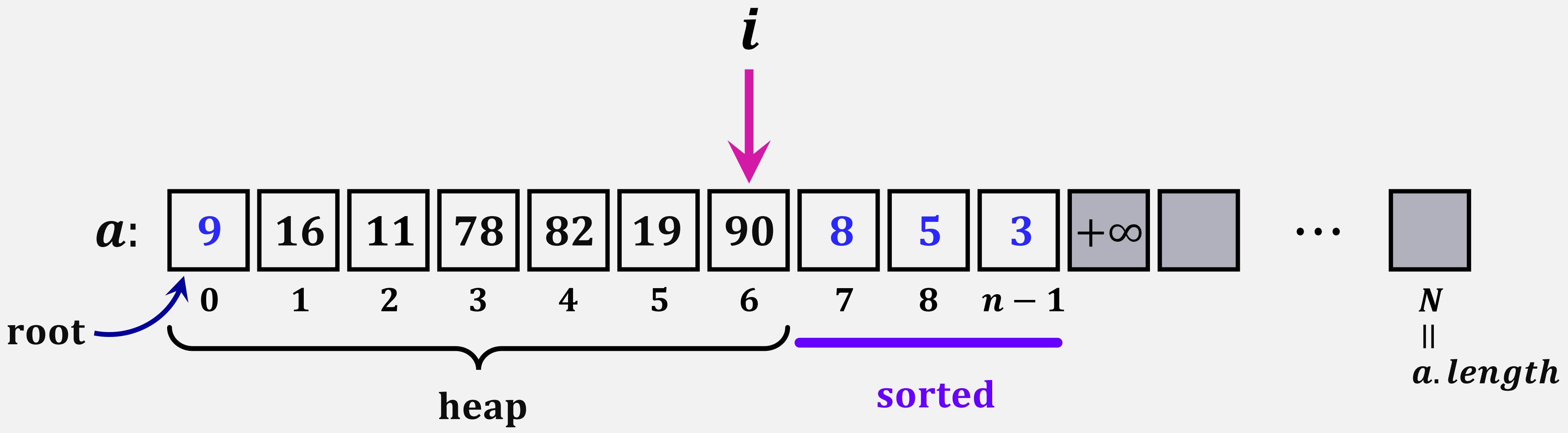


Alina Shaikhet – COMP 2402 – Carleton University

HeapSort

Given: min-heap as array a of n numbers

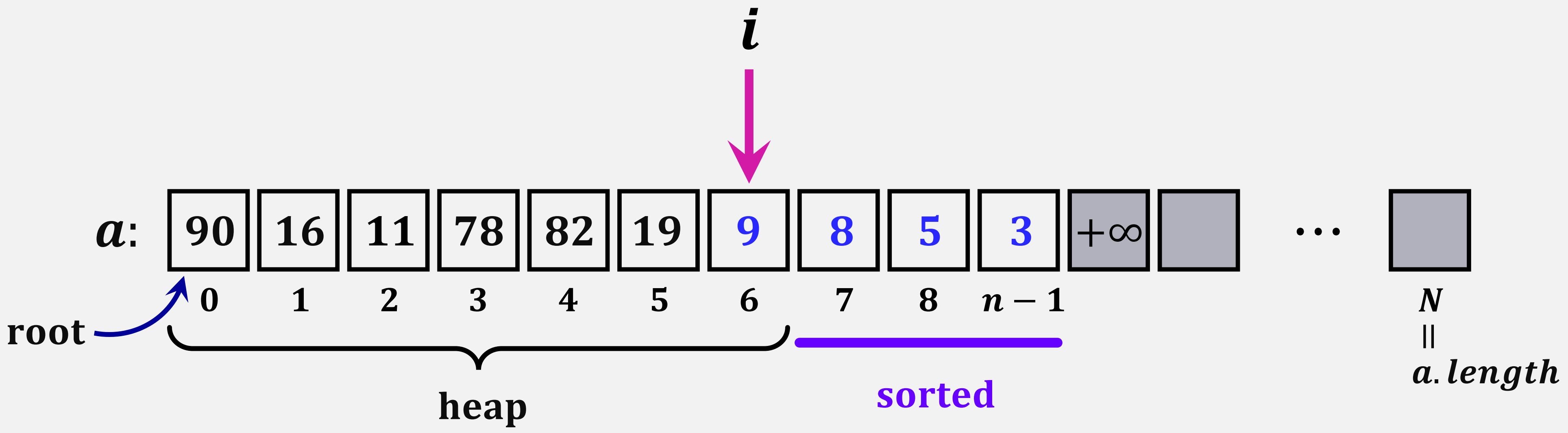
Goal: array a, containing the same elements but in sorted order.



HeapSort

Given: min-heap as array a of n numbers

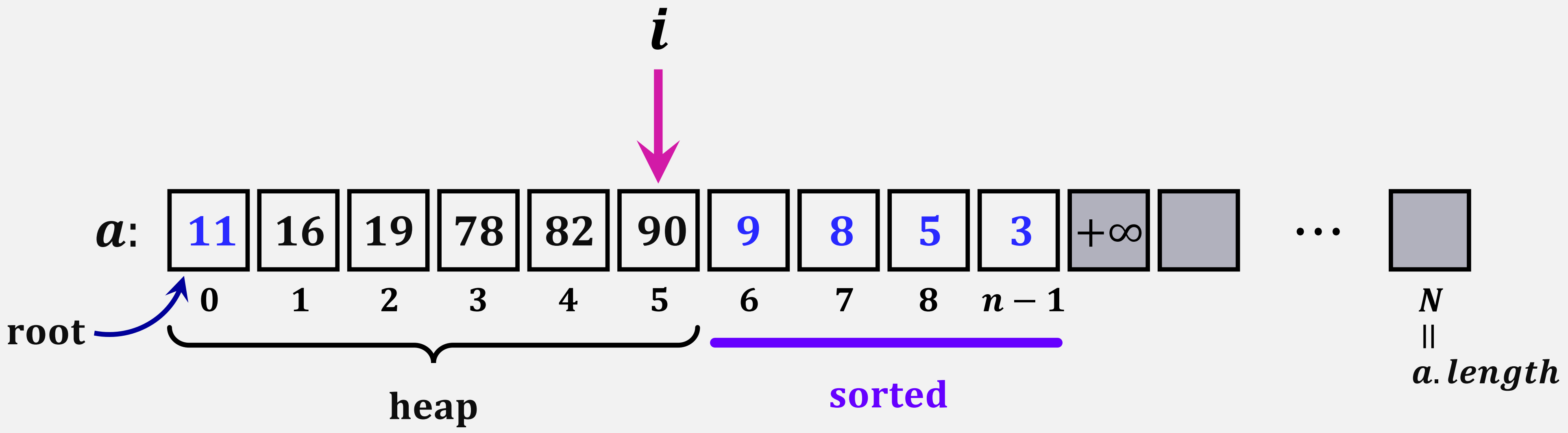
Goal: array a, containing the same elements but in sorted order.



HeapSort

Given: min-heap as array $\boldsymbol{a}$ of $\boldsymbol{n}$ numbers

Goal: array $\boldsymbol{a}$, containing the same elements but in sorted order.



HeapSort is an **in-place** algorithm!

HeapSort(a)

a: | heap part | sorted part | |

0 $\quad i \quad$ $n-1$

Input: array a of n numbers

Output: array a, containing the same elements in sorted order.

buildMinHeap(a);
$i = n - 1;$
while $i \geq 1$ do:
    swap $a[0]$ and $a[i]$;
    $i--;$ $n--;$
    minHeapify(0);

$a[0 \ldots i]$ is a heap,
$a[i+1 \ldots n-1]$ contains the $n-i-1$ smallest elements in sorted order

HeapSort($\boldsymbol{a}$)

$\boldsymbol{a}$:

heap part	sorted part	

$\boldsymbol{0}$ $\quad$ $\boldsymbol{i}$ $\quad$ $\boldsymbol{n-1}$

Input: array $\boldsymbol{a}$ of $\boldsymbol{n}$ numbers

Output: array $\boldsymbol{a}$, containing the same elements in sorted order.

```
buildMinHeap(a);

while n ≥ 1 do:
      swap a[0] and a[n − 1];
      n − −;
      minHeapify(0);
```

$\boldsymbol{a}[\mathbf{0}\ldots \boldsymbol{i}]$ is a heap,
$\boldsymbol{a}[\boldsymbol{i}+\mathbf{1}\ldots \boldsymbol{n}-\mathbf{1}]$ contains the $\boldsymbol{n}-\boldsymbol{i}-\mathbf{1}$ smallest elements in sorted order

$$\underbrace{O(n)}_{\text{build heap}} + \underbrace{O(\log(n-1)+\log(n-2)+\cdots+\log 3+\log 2)}_{\text{while loop}} = O(n\log n)$$

HeapSort(a)

a: | heap part | sorted part | |
0 $\quad$ i $\quad$ $n-1$

Input: array a of n numbers

Output: array a, containing the same elements in sorted order.

buildMinHeap(a);

while $n \geq 1$ do:
    swap $a[0]$ and $a[n-1]$;
    $n--$;
    minHeapify(0);

buildMinHeap(a);
for ($j = 0; j < n; i++$) do:
    $x =$ **removeMin**();
    $a[n-1-j] = x$;

$$\underbrace{O(n)}_{\text{build heap}} + \underbrace{O(\log(n-1) + \log(n-2) + \cdots + \log 3 + \log 2)}_{\text{while loop}} = O(n \log n)$$

Theorem 11.4

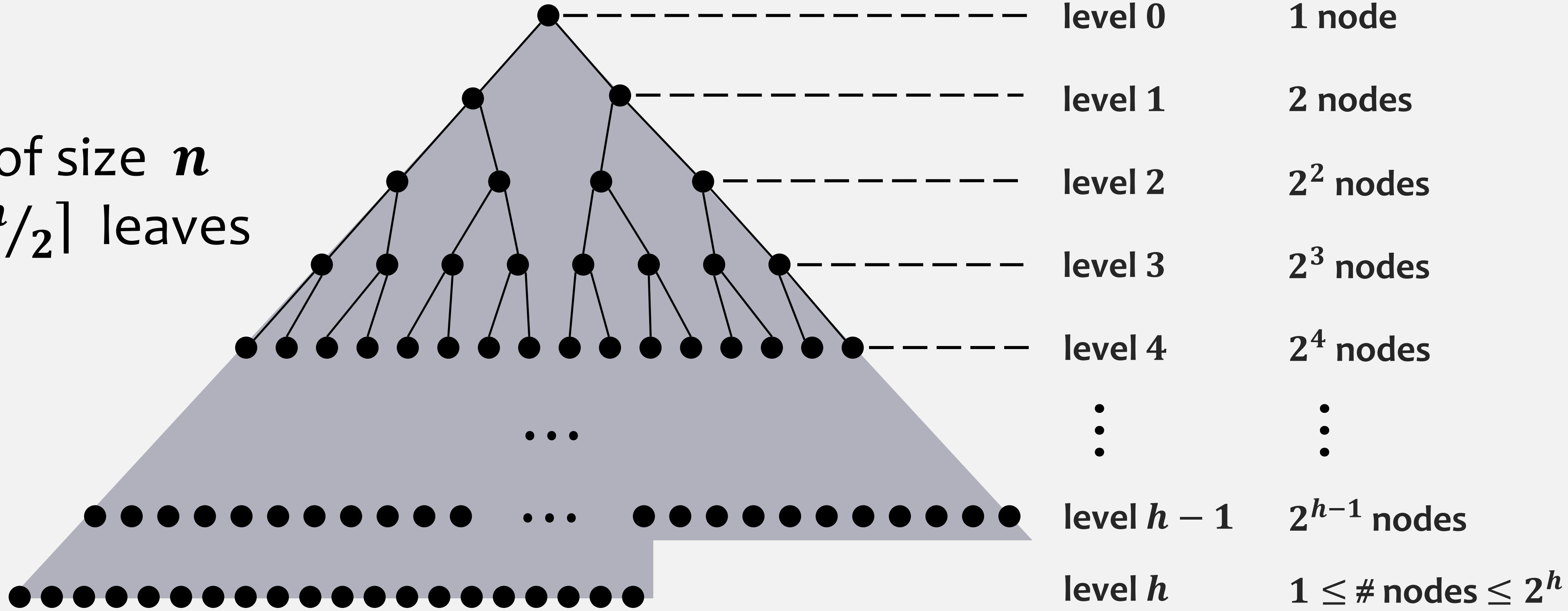
The **HeapSort** algorithm sorts an array containing $\boldsymbol{n}$ elements in $\boldsymbol{O}(\boldsymbol{n}\log\boldsymbol{n})$ worst-case time and performs at most $\boldsymbol{2n}\log\boldsymbol{n}+\boldsymbol{O}(\boldsymbol{n})$ comparisons.

How to build a heap in $O(n)$ time?

$\boldsymbol{a}$: ●●● … ●●●●●●●●●●●

$$\mathbf{2^0 + 2^1 + 2^2 + \cdots + 2^k = 2^{k+1} - 1}$$

Heap of size $\boldsymbol{n}$ has $\lceil n/2 \rceil$ leaves



How to build a heap in $O(n)$ time?

All $\boldsymbol{a[i]}$, $\lfloor {}^{\boldsymbol{n+1}}/_{\boldsymbol{2}} \rfloor \leq \boldsymbol{i} \leq \boldsymbol{n-1}$,
are leaves.

height of the root is $\lfloor \log n \rfloor$

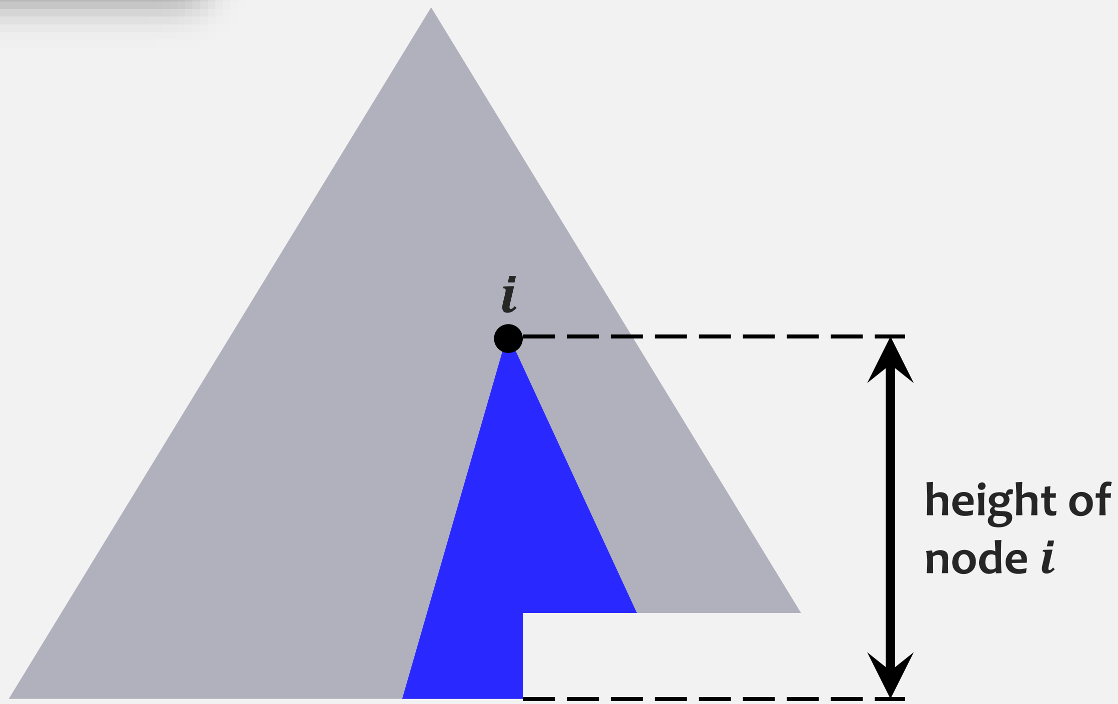
height of a leaf is $\mathbf{0}$

minHeapify($\boldsymbol{i}$) $\boxed{\boldsymbol{O}(\text{height of } \boldsymbol{i})}$

Heap of size $\boldsymbol{n}$ has $\lceil {}^{\boldsymbol{n}}/_{\boldsymbol{2}} \rceil$ leaves.

Every node $\boldsymbol{a[i]}$, where
$\lfloor {}^{\boldsymbol{n+1}}/_{\boldsymbol{2}} \rfloor \leq \boldsymbol{i} \leq \boldsymbol{n-1}$, is a leaf

Most of the nodes have
small height.



buildMinHeap(a)

A subtree rooted at a leaf is a heap!
All $\boldsymbol{a[i]}$, $\lfloor {}^{n+1}/_{2} \rfloor \le \boldsymbol{i} \le \boldsymbol{n-1}$, are leaves.

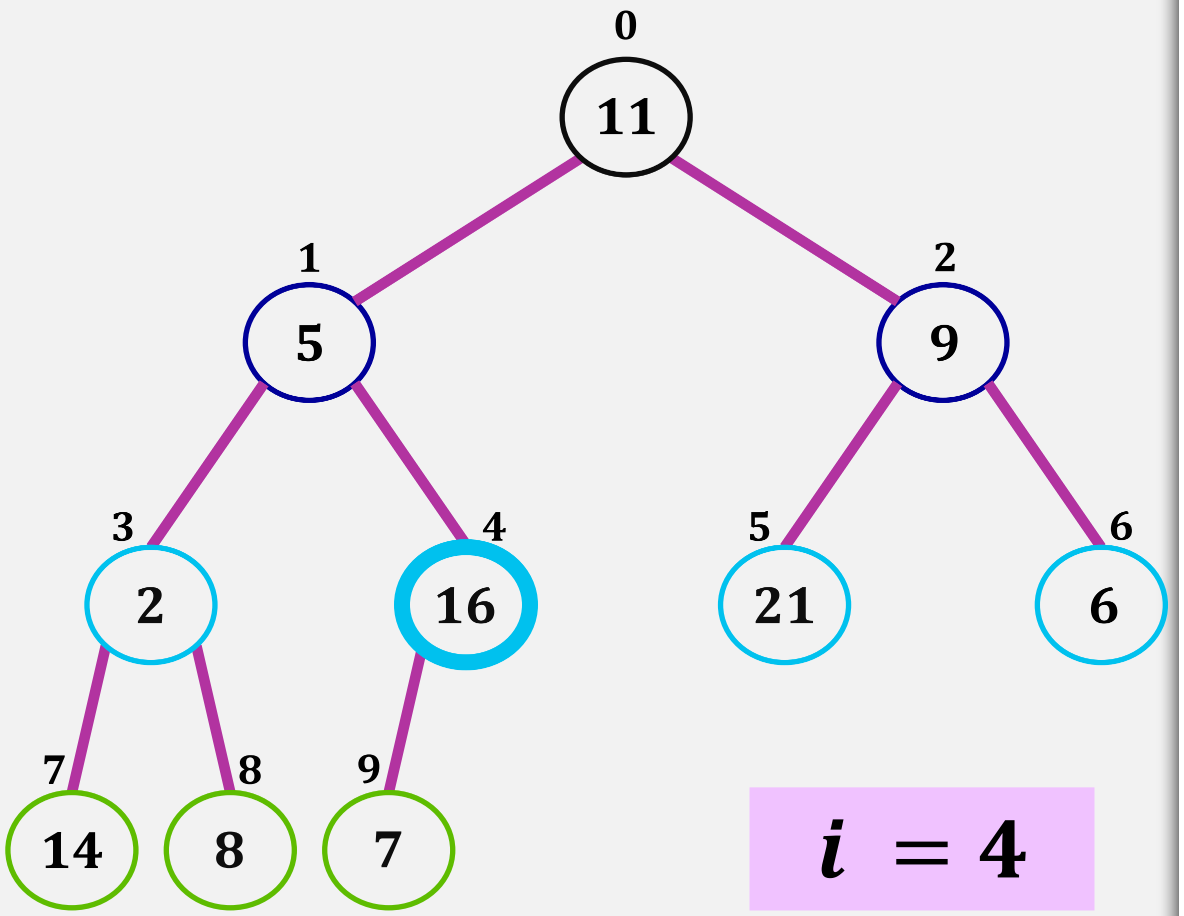
Input: array $\boldsymbol{a}$ with $\boldsymbol{n}$ elements.

Output: heap $\boldsymbol{a}$ of size $\boldsymbol{n}$, containing the same elements

$\boldsymbol{a} = [\mathbf{11, 5, 9, 2, 16, 21, 6, 14, 8, 7}]$, $\boldsymbol{n} = \mathbf{10}$

$\boldsymbol{i}$ starts at $\lfloor {}^{n+1}/_{2} \rfloor - \mathbf{1} = \lfloor {}^{n-1}/_{2} \rfloor = \mathbf{4}$

for $(\boldsymbol{i} = \lfloor {}^{n-1}/_{2} \rfloor$ downto $\mathbf{0})$:
    minHeapify$(\boldsymbol{i})$



$\boldsymbol{i} = \mathbf{4}$

Alina Shaikhet – COMP 2402 – Carleton University

buildMinHeap(a)

A subtree rooted at a leaf is a heap!
All $\boldsymbol{a[i]}$, $\lfloor {}^{n+1}/_2 \rfloor \le i \le n-1$, are leaves.

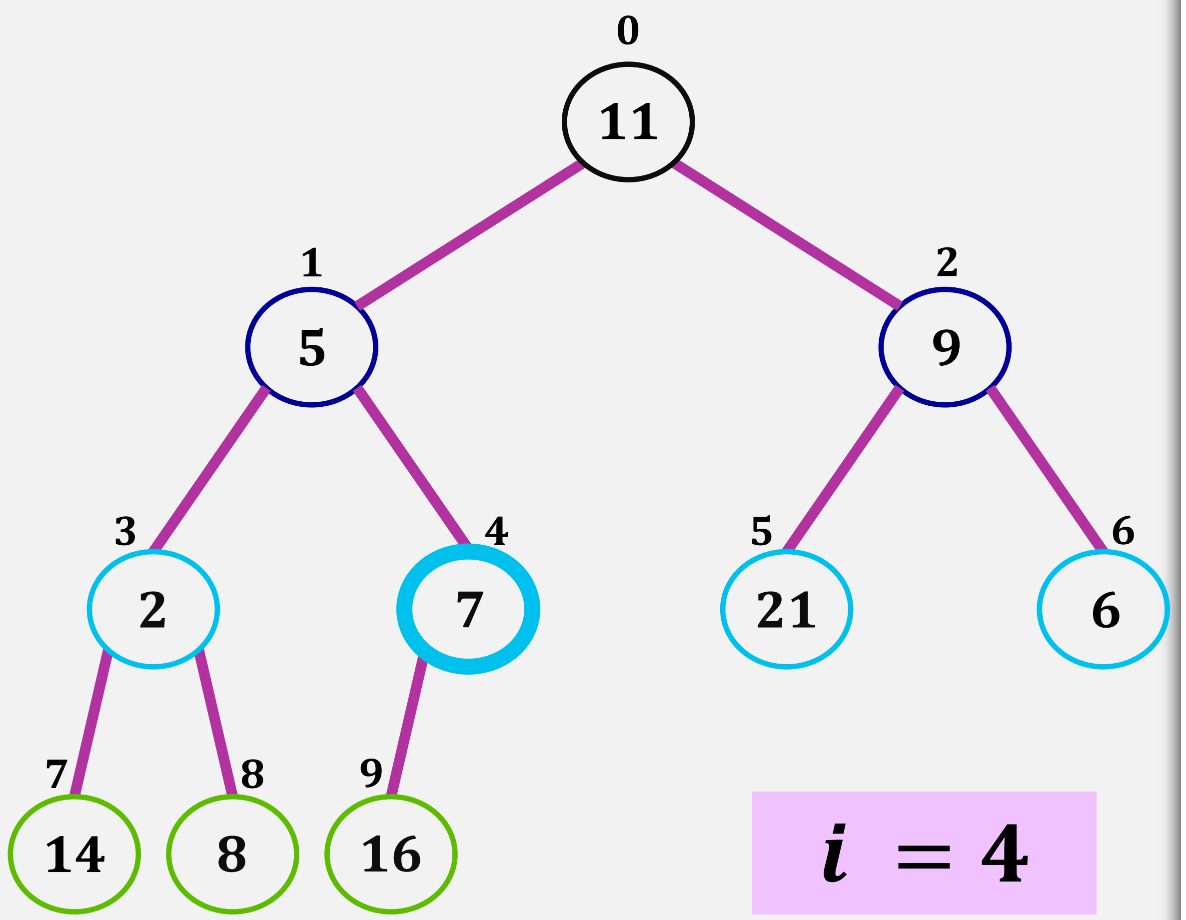
Input: array $\boldsymbol{a}$ with $\boldsymbol{n}$ elements.

Output: heap $\boldsymbol{a}$ of size $\boldsymbol{n}$, containing the same elements

$\boldsymbol{a} = [11, 5, 9, 2, 16, 21, 6, 14, 8, 7]$, $\boldsymbol{n} = 10$

$\boldsymbol{i}$ starts at $\lfloor {}^{n+1}/_2 \rfloor - 1 = \lfloor {}^{n-1}/_2 \rfloor = 4$

for $(\boldsymbol{i} = \lfloor {}^{n-1}/_2 \rfloor$ downto $\boldsymbol{0})$:
minHeapify($\boldsymbol{i}$)



$\boldsymbol{i} = 4$

Alina Shaikhet – COMP 2402 – Carleton University

buildMinHeap(a)

A subtree rooted at a leaf is a heap!
All $a[i]$, $\lfloor {}^{n+1}/_2 \rfloor \le i \le n-1$, are leaves.

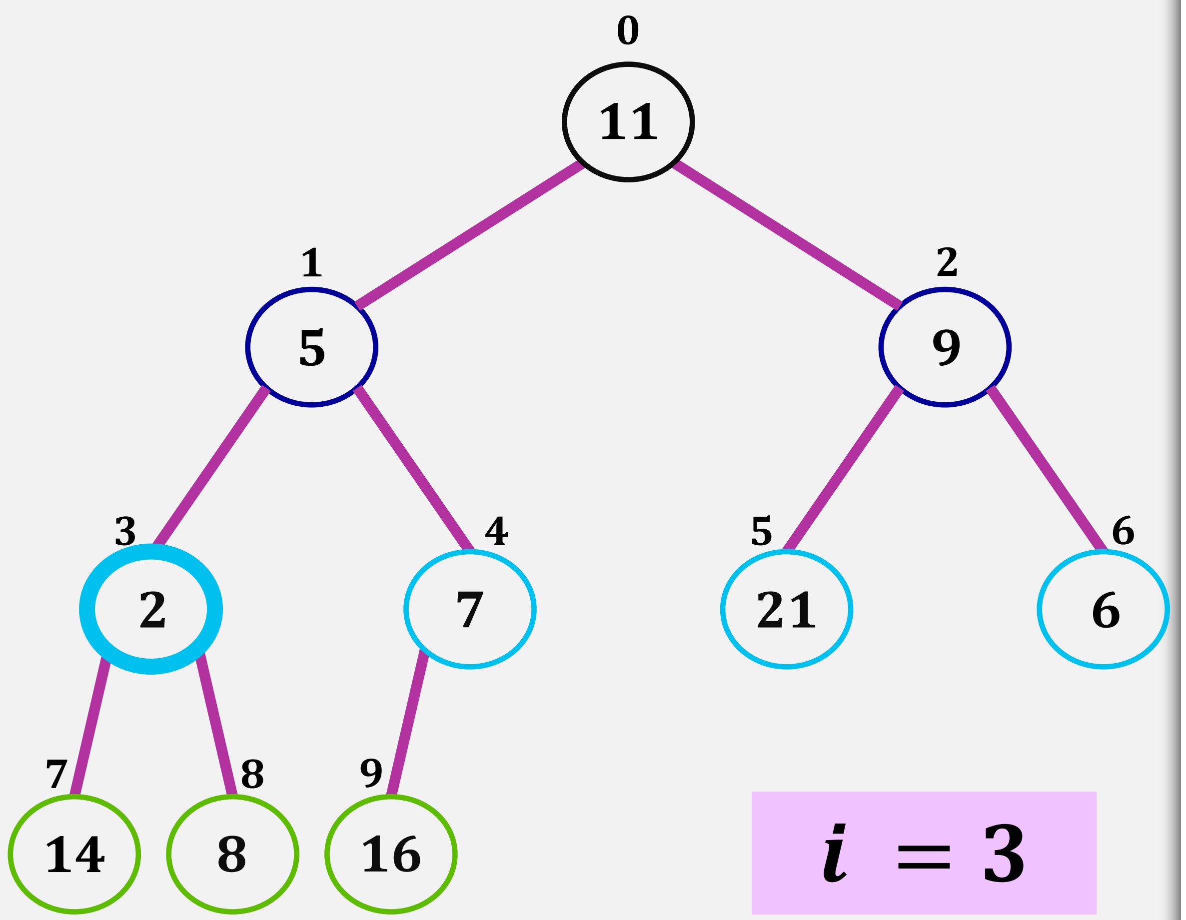
Input: array a with n elements.

Output: heap a of size n, containing the same elements

$a = [11, 5, 9, 2, 16, 21, 6, 14, 8, 7]$, $n = 10$

i starts at $\lfloor {}^{n+1}/_2 \rfloor - 1 = \lfloor {}^{n-1}/_2 \rfloor = 4$

for $(i = \lfloor {}^{n-1}/_2 \rfloor$ downto $0)$:
  minHeapify(i)



$i = 3$

buildMinHeap(a)

A subtree rooted at a leaf is a heap!

All $a[i]$, $\lfloor^{n+1}/_2\rfloor \le i \le n-1$, are leaves.

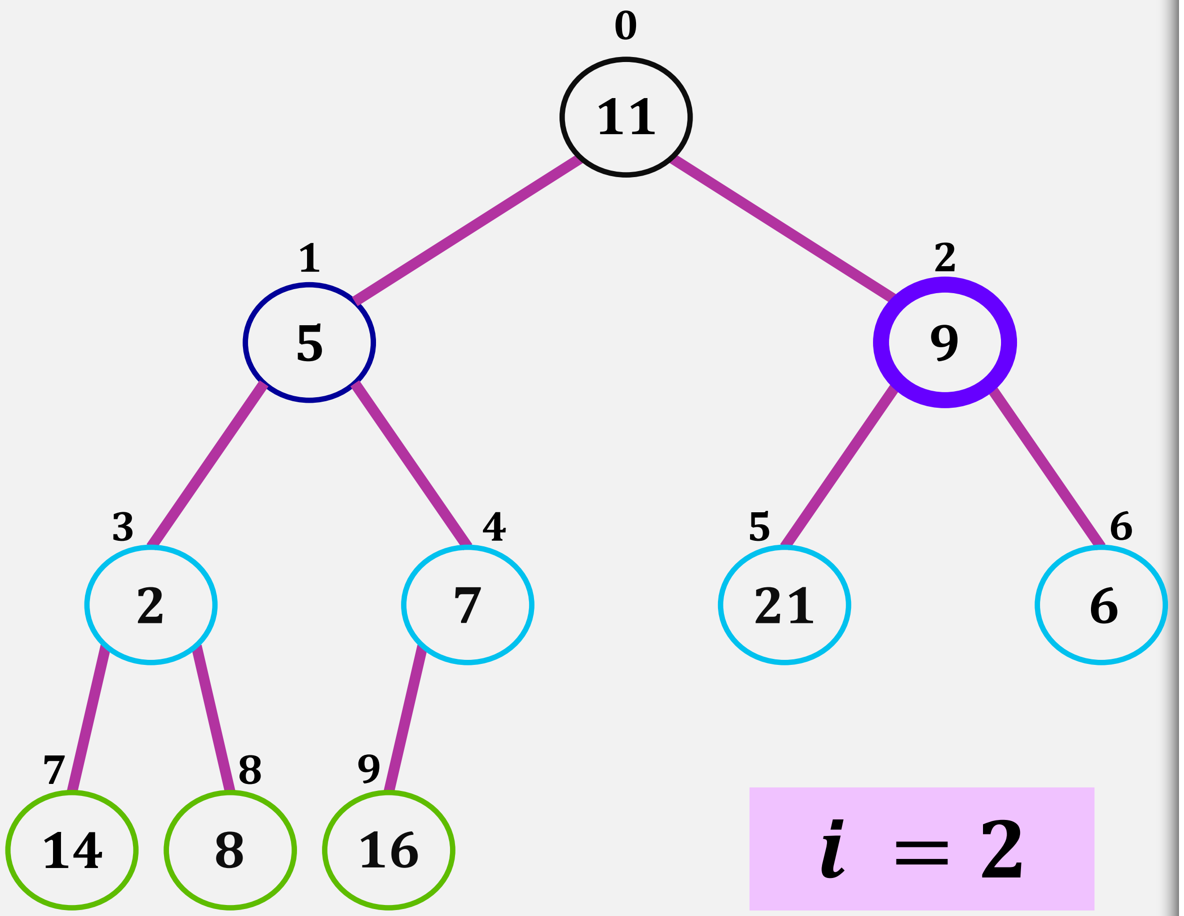
Input: array a with n elements.

Output: heap a of size n, containing the same elements

$a = [11, 5, 9, 2, 16, 21, 6, 14, 8, 7]$, $n = 10$

i starts at $\lfloor^{n+1}/_2\rfloor - 1 = \lfloor^{n-1}/_2\rfloor = 4$

for $(i = \lfloor^{n-1}/_2\rfloor$ downto $0)$:
  minHeapify(i)



$i = 2$

buildMinHeap(a)

A subtree rooted at a leaf is a heap!

All $\boldsymbol{a[i]}$, $\lfloor {}^{n+1}/_{2} \rfloor \leq \boldsymbol{i} \leq \boldsymbol{n-1}$, are leaves.

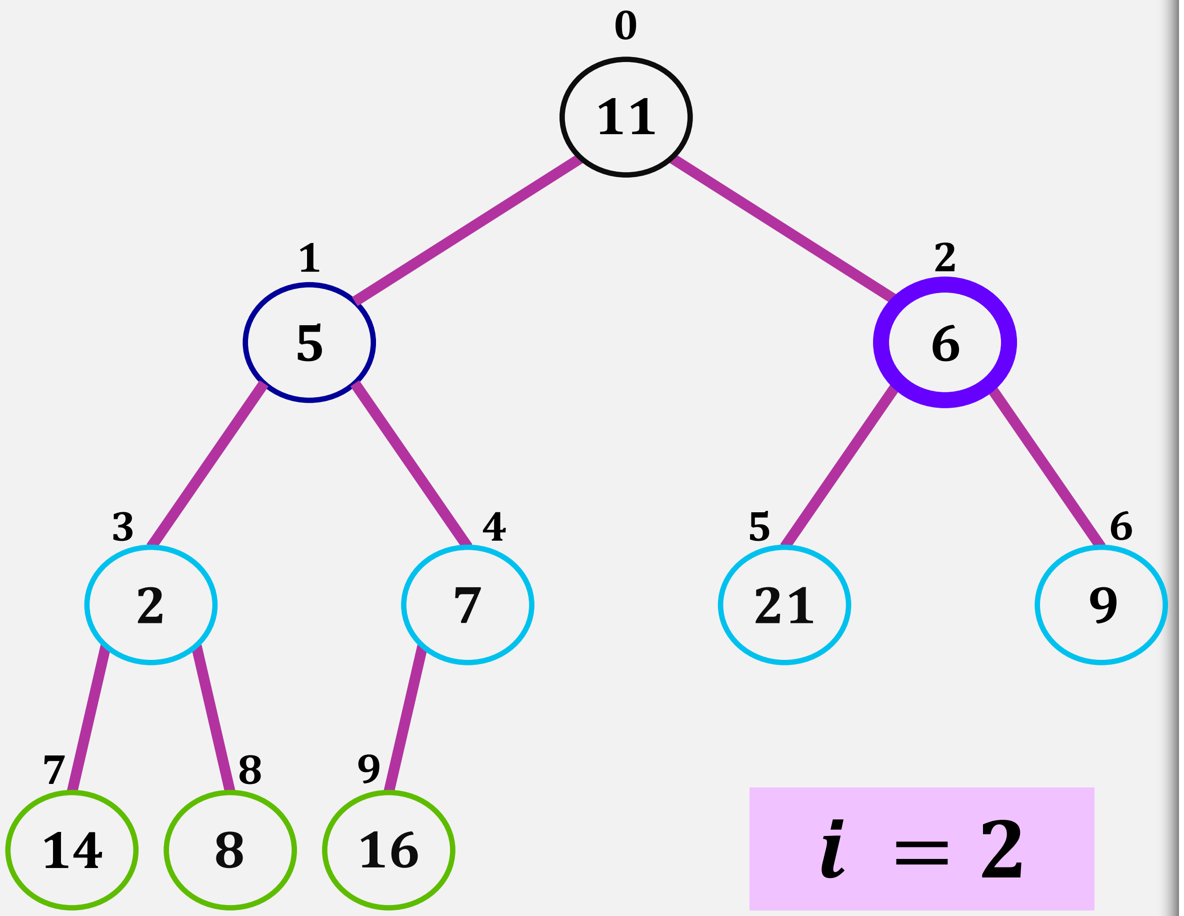
Input: array $\boldsymbol{a}$ with $\boldsymbol{n}$ elements.

Output: heap $\boldsymbol{a}$ of size $\boldsymbol{n}$, containing the same elements

$\boldsymbol{a} = [11, 5, 9, 2, 16, 21, 6, 14, 8, 7]$, $\boldsymbol{n} = 10$

$\boldsymbol{i}$ starts at $\lfloor {}^{n+1}/_{2} \rfloor - 1 = \lfloor {}^{n-1}/_{2} \rfloor = 4$

for ($\boldsymbol{i} = \lfloor {}^{n-1}/_{2} \rfloor$ downto $\boldsymbol{0}$):
    minHeapify($\boldsymbol{i}$)



$\boldsymbol{i} = 2$

buildMinHeap(a)

A subtree rooted at a leaf is a heap!
All $\boldsymbol{a[i]}$, $\lfloor {}^{\boldsymbol{n+1}}/_{\boldsymbol{2}} \rfloor \le \boldsymbol{i} \le \boldsymbol{n-1}$,
are leaves.

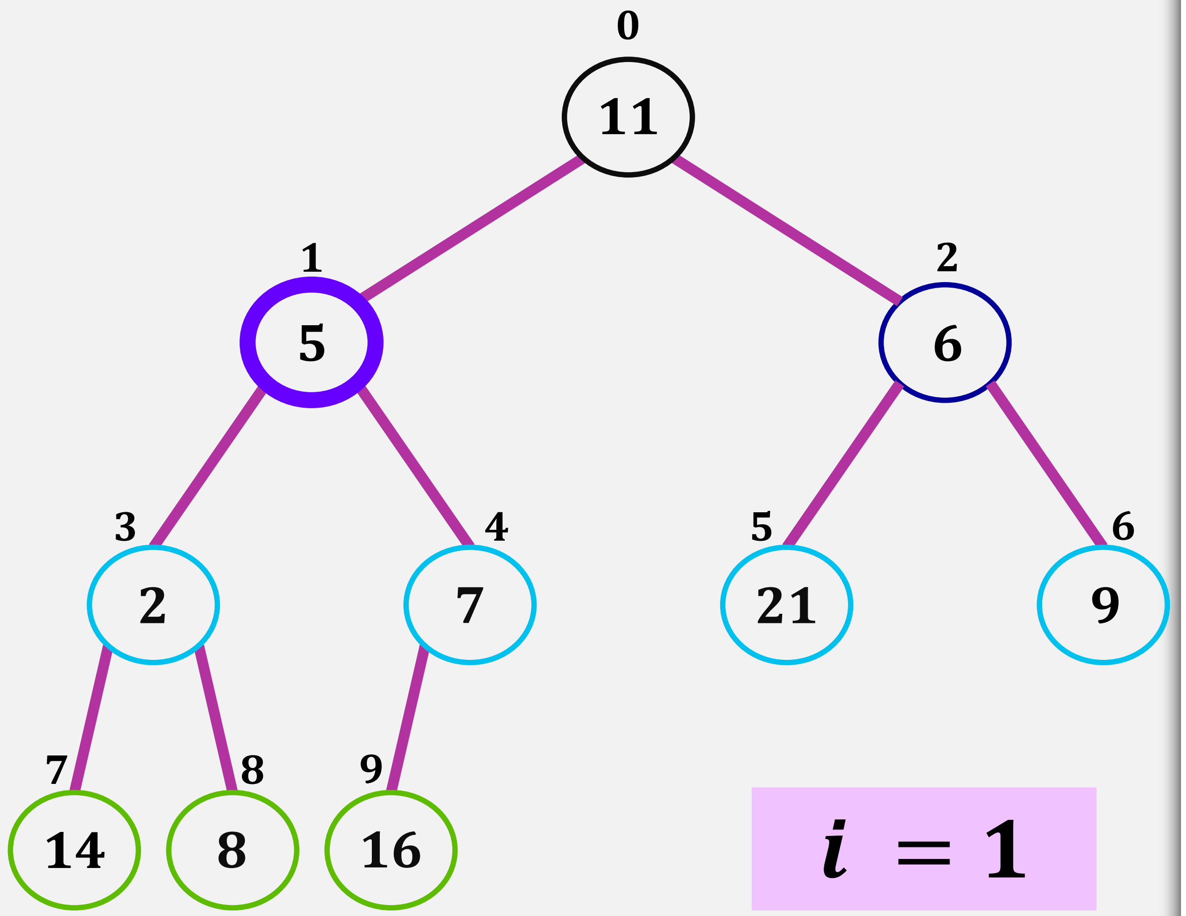
Input: array $\boldsymbol{a}$ with $\boldsymbol{n}$ elements.

Output: heap $\boldsymbol{a}$ of size $\boldsymbol{n}$, containing the same elements

$\boldsymbol{a = [11, 5, 9, 2, 16, 21, 6, 14, 8, 7]}$, $\boldsymbol{n = 10}$

$\boldsymbol{i}$ starts at $\lfloor {}^{\boldsymbol{n+1}}/_{\boldsymbol{2}} \rfloor - \boldsymbol{1} = \lfloor {}^{\boldsymbol{n-1}}/_{\boldsymbol{2}} \rfloor = \boldsymbol{4}$

for ($\boldsymbol{i} = \lfloor {}^{\boldsymbol{n-1}}/_{\boldsymbol{2}} \rfloor$ downto $\boldsymbol{0}$):
  minHeapify($\boldsymbol{i}$)



$\boldsymbol{i = 1}$

Alina Shaikhet – COMP 2402 – Carleton University

buildMinHeap(a)

A subtree rooted at a leaf is a heap!
All $a[i]$, $\lfloor {}^{n+1}/_2 \rfloor \le i \le n-1$, are leaves.

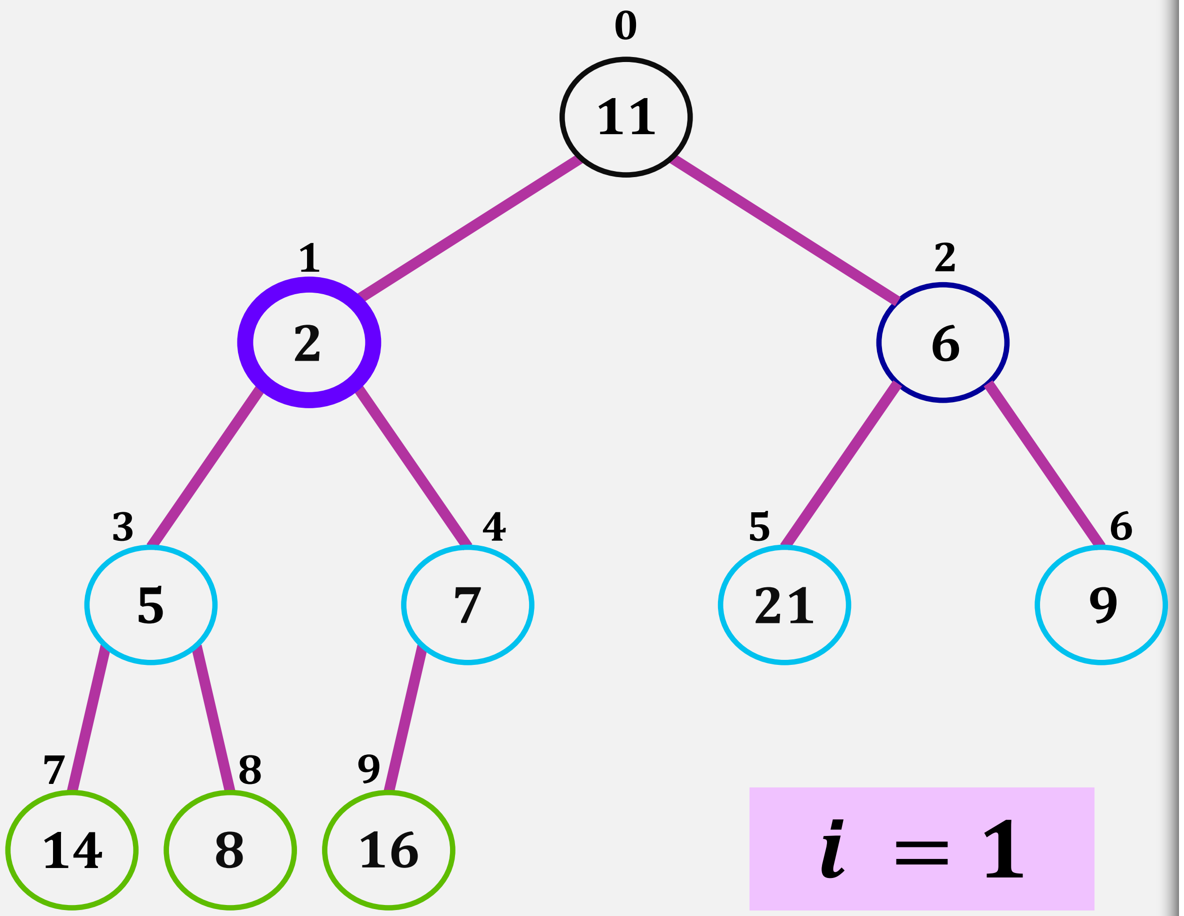
Input: array a with n elements.

Output: heap a of size n, containing the same elements

$a = [11, 5, 9, 2, 16, 21, 6, 14, 8, 7]$, $n = 10$

i starts at $\lfloor {}^{n+1}/_2 \rfloor - 1 = \lfloor {}^{n-1}/_2 \rfloor = 4$

for $(i = \lfloor {}^{n-1}/_2 \rfloor$ downto $0)$:
  minHeapify(i)



$i = 1$

buildMinHeap(a)

Input: array $\boldsymbol{a}$ with $\boldsymbol{n}$ elements.

Output: heap $\boldsymbol{a}$ of size $\boldsymbol{n}$, containing the same elements

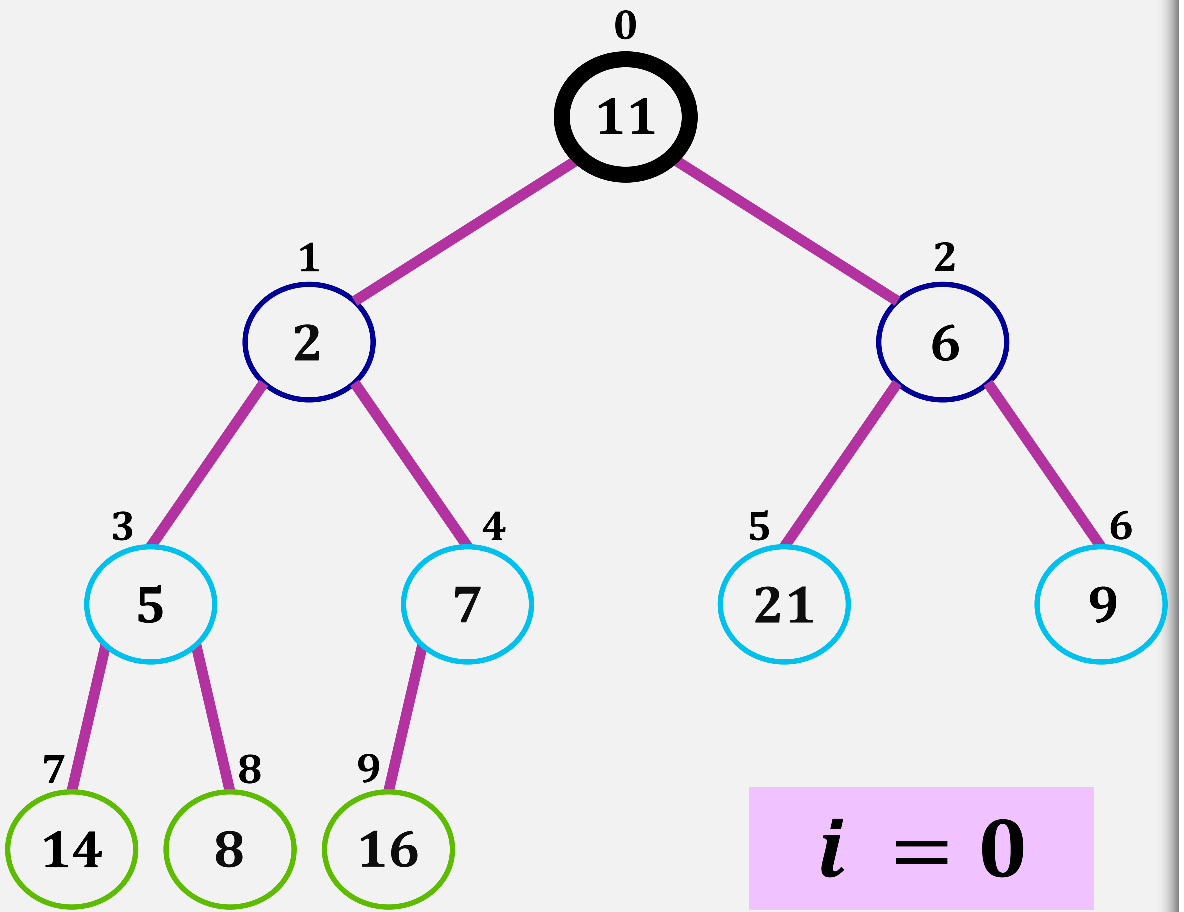
A subtree rooted at a leaf is a heap!

All $\boldsymbol{a[i]}$, $\lfloor {}^{n+1}/_{2} \rfloor \leq \boldsymbol{i} \leq \boldsymbol{n} - \mathbf{1}$, are leaves.

$\boldsymbol{a} = [\mathbf{11, 5, 9, 2, 16, 21, 6, 14, 8, 7}]$, $\boldsymbol{n} = \mathbf{10}$

$\boldsymbol{i}$ starts at $\lfloor {}^{n+1}/_{2} \rfloor - \mathbf{1} = \lfloor {}^{n-1}/_{2} \rfloor = \mathbf{4}$

for $(\boldsymbol{i} = \lfloor {}^{n-1}/_{2} \rfloor$ downto $\mathbf{0})$:
    minHeapify$(\boldsymbol{i})$



$\boldsymbol{i} = \mathbf{0}$

buildMinHeap(a)

A subtree rooted at a leaf is a heap!

All $\boldsymbol{a[i]}$, $\lfloor {}^{n+1}/_{2} \rfloor \leq \boldsymbol{i} \leq \boldsymbol{n-1}$,
are leaves.

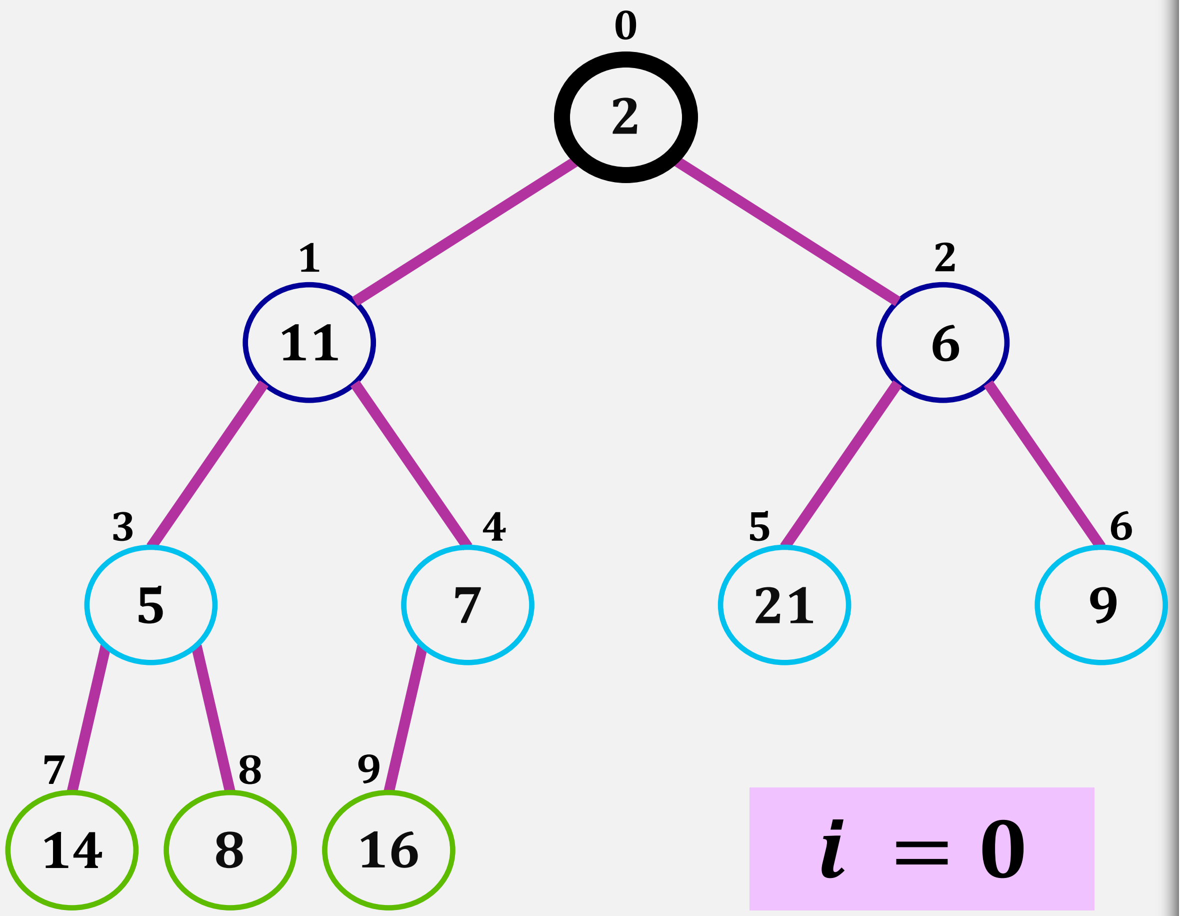
Input: array $\boldsymbol{a}$ with $\boldsymbol{n}$ elements.

Output: heap $\boldsymbol{a}$ of size $\boldsymbol{n}$, containing the same elements

$\boldsymbol{a = [11, 5, 9, 2, 16, 21, 6, 14, 8, 7], \; n = 10}$

$\boldsymbol{i}$ starts at $\lfloor {}^{n+1}/_{2} \rfloor - \boldsymbol{1} = \lfloor {}^{n-1}/_{2} \rfloor = \boldsymbol{4}$

for $(\boldsymbol{i} = \lfloor {}^{n-1}/_{2} \rfloor$ downto $\boldsymbol{0})$:
  minHeapify($\boldsymbol{i}$)



$\boldsymbol{i = 0}$

Alina Shaikhet – COMP 2402 – Carleton University

buildMinHeap(a)

A subtree rooted at a leaf is a heap!
All $a[i]$, $\lfloor {}^{n+1}/_2 \rfloor \leq i \leq n-1$, are leaves.

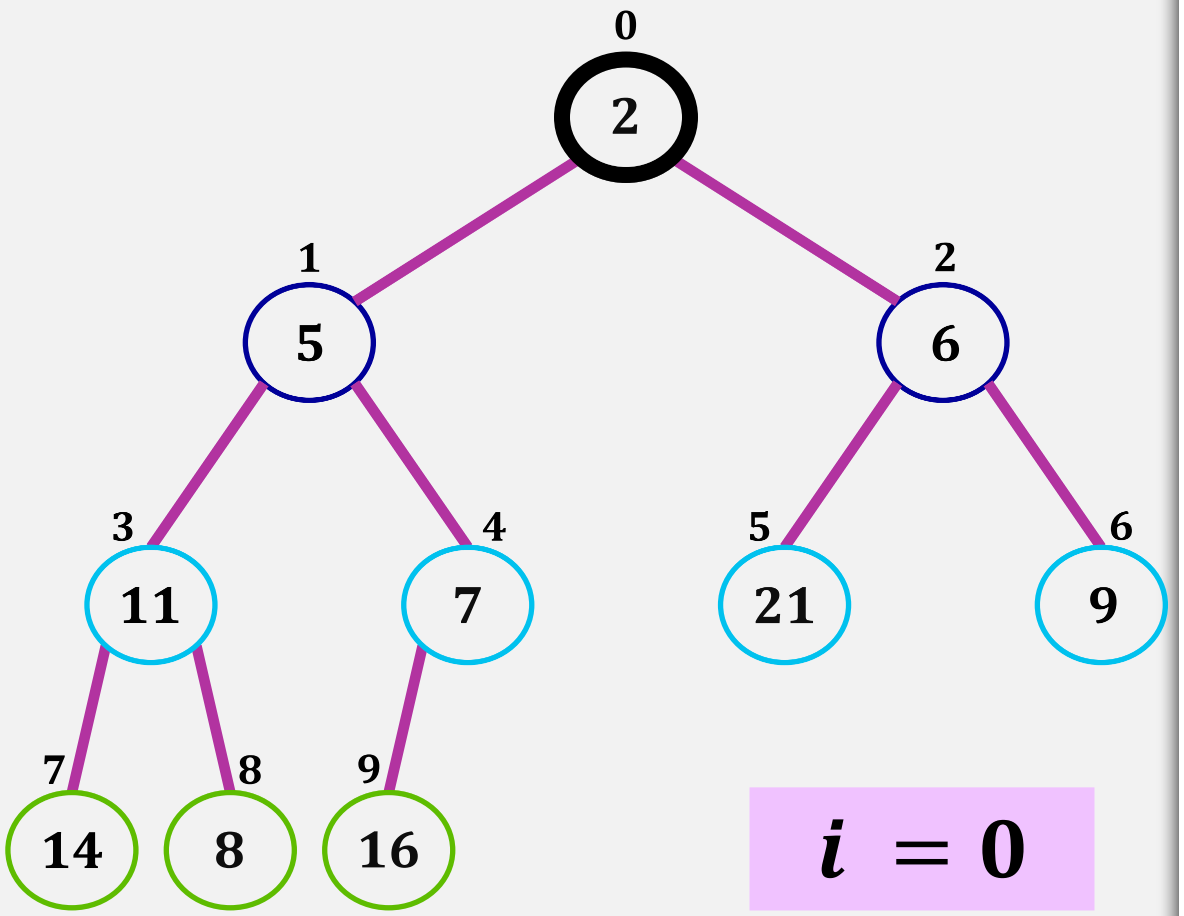
Input: array a with n elements.

Output: heap a of size n, containing the same elements

$a = [11, 5, 9, 2, 16, 21, 6, 14, 8, 7]$, $n = 10$

i starts at $\lfloor {}^{n+1}/_2 \rfloor - 1 = \lfloor {}^{n-1}/_2 \rfloor = 4$

for $(i = \lfloor {}^{n-1}/_2 \rfloor$ downto $0)$:
  minHeapify(i)



$i = 0$

buildMinHeap(a)

A subtree rooted at a leaf is a heap!
All $\boldsymbol{a[i]},\ \lfloor {}^{n+1}/_{2} \rfloor \le \boldsymbol{i} \le \boldsymbol{n-1},$
are leaves.

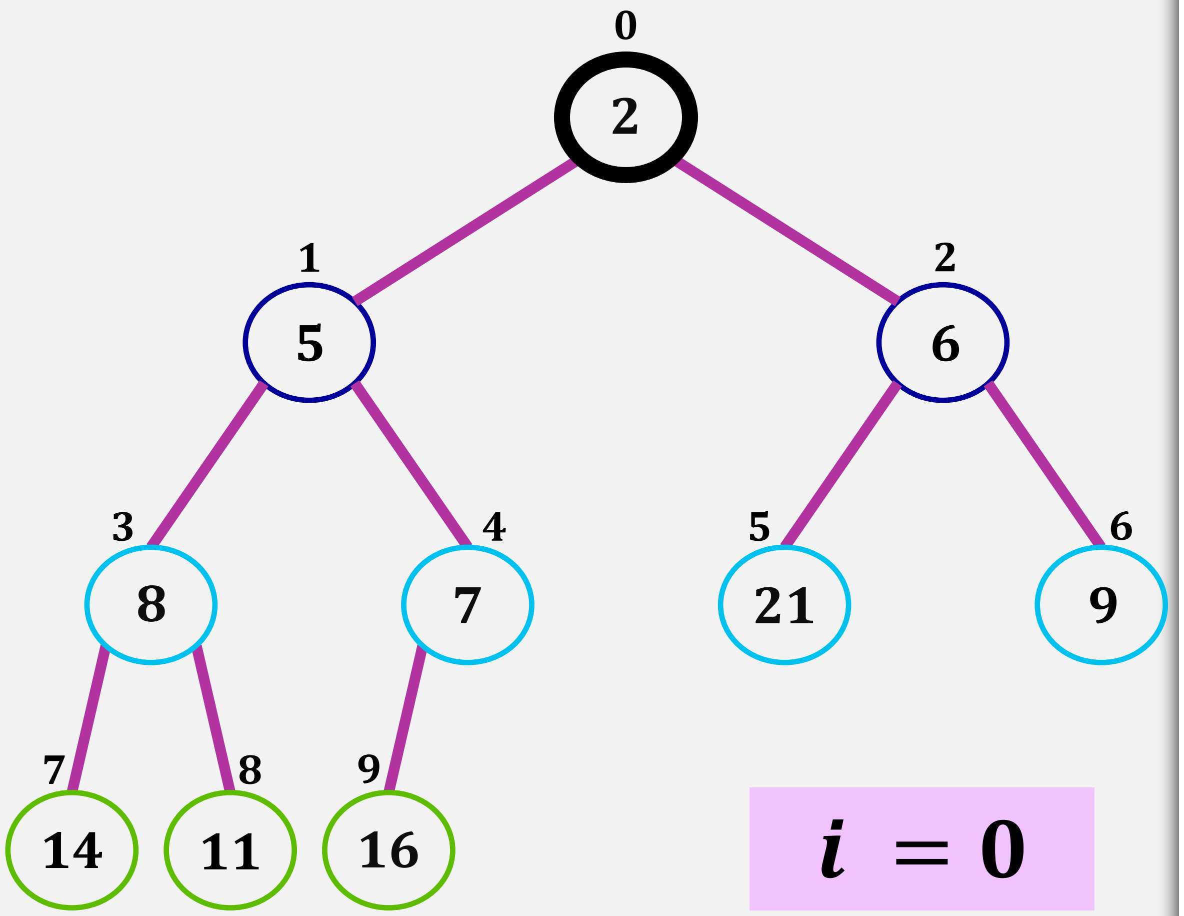
Input: array $\boldsymbol{a}$ with $\boldsymbol{n}$ elements.

Output: heap $\boldsymbol{a}$ of size $\boldsymbol{n}$, containing the same elements

$\boldsymbol{a} = [\mathbf{11, 5, 9, 2, 16, 21, 6, 14, 8, 7}],\ \boldsymbol{n} = \mathbf{10}$

$\boldsymbol{i}$ starts at $\lfloor {}^{n+1}/_{2} \rfloor - \mathbf{1} = \lfloor {}^{n-1}/_{2} \rfloor = \mathbf{4}$

for $(\boldsymbol{i} = \lfloor {}^{n-1}/_{2} \rfloor$ downto $\mathbf{0})$:
    minHeapify$(\boldsymbol{i})$



$\boldsymbol{i} = \mathbf{0}$

Running Time of buildMinHeap(a)

minHeapify(i) $\boxed{O(\text{height of } i)}$

$$T(n) \le 1 \cdot h + 2(h-1) + 2^2(h-2) + \cdots + 2^{h-2} \cdot 2 + 2^{h-1} \cdot 1 =$$

$$= \sum_{i=1}^{h} 2^{h-i} \cdot i = 2^h \sum_{i=1}^{h} i \cdot \left(\frac{1}{2}\right)^i$$

$$\le n \underbrace{\sum_{i=1}^{\infty} i \cdot \left(\frac{1}{2}\right)^i}_{\text{constant}} \le O(n)$$

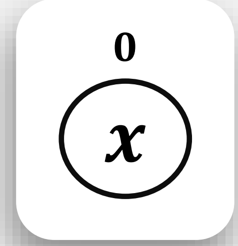
$$h = \lfloor \log n \rfloor \le \log n$$
$$2^h \le 2^{\log n} = n$$

level	height	# of nodes
0	h	1
1	$h-1$	2
2	$h-2$	2^2
3	$h-3$	2^3
…	…	…
$h-1$	1	2^{h-1}
h	0	$\le 2^h$

Randomized Meldable Heap

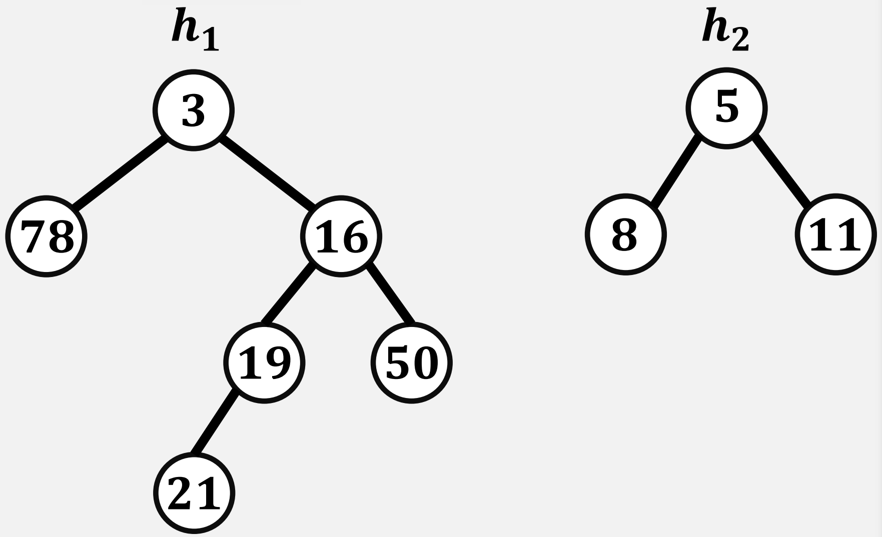
MeldableHeap is a priority **Queue** implementation in which the underlying structure is a heap-ordered binary tree with no restrictions on its shape.

- makeHeap($\boldsymbol{x}$) – returns a heap containing only $\boldsymbol{x}$



$\boxed{O(1)}$

- merge($\boldsymbol{h_1}, \boldsymbol{h_2}$) – returns a heap that contains all the elements in $\boldsymbol{h_1}$ and $\boldsymbol{h_2}$



merge($\boldsymbol{h_1}, \boldsymbol{h_2}$)

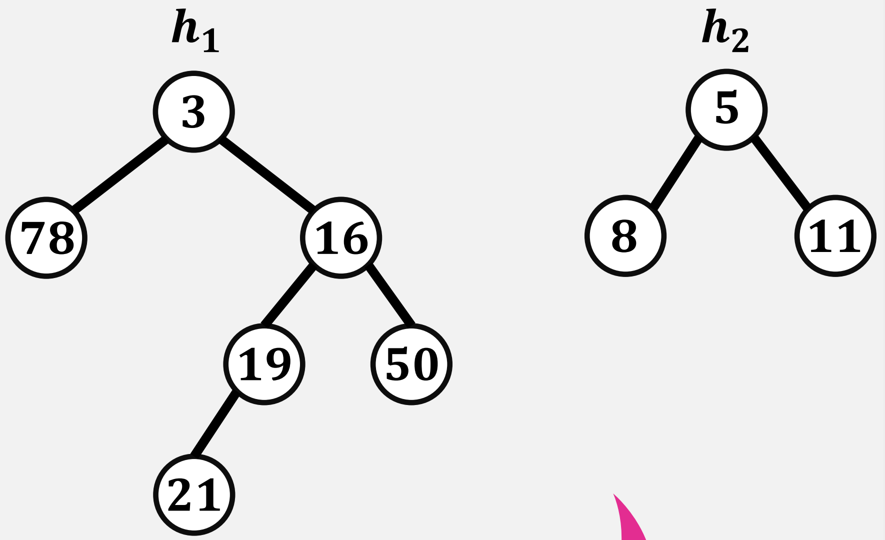
This operation can be defined recursively.

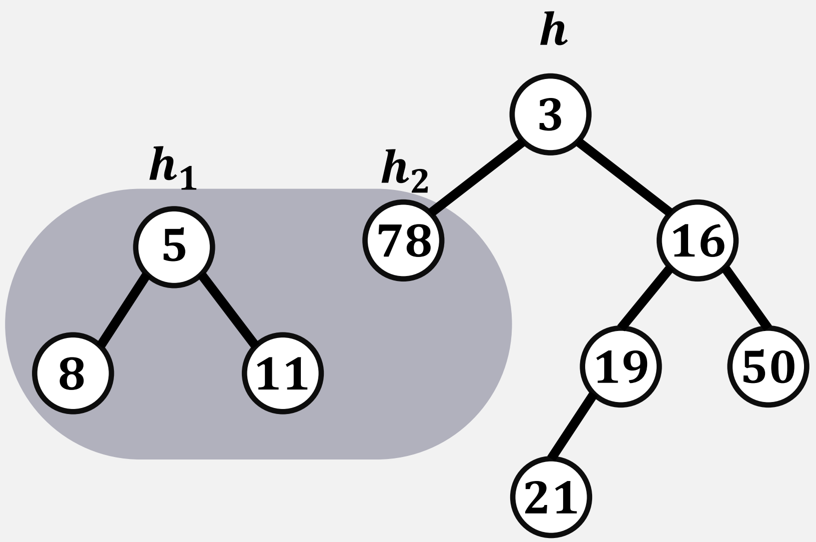
- If either $\boldsymbol{h_1}$ or $\boldsymbol{h_2}$ is null, then we are merging with an empty set, so we return $\boldsymbol{h_2}$ or $\boldsymbol{h_1}$, respectively.
- Otherwise, assume $\boldsymbol{h_1.x} \le \boldsymbol{h_2.x}$.

if $\boldsymbol{h_1.x} > \boldsymbol{h_2.x}$ then
swap $\boldsymbol{h_1} \leftrightarrow \boldsymbol{h_2}$

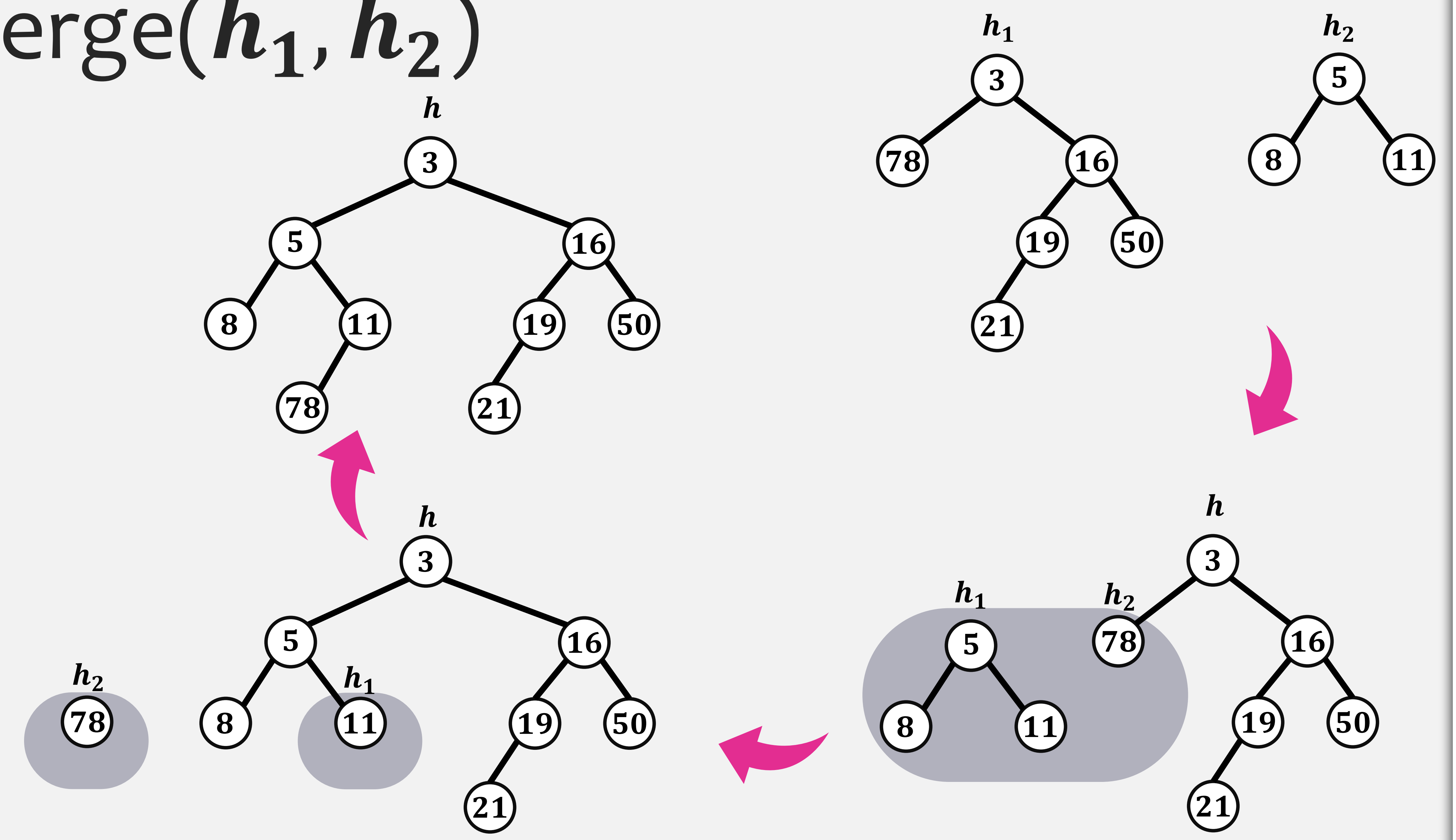
- The root of the merged heap will contain $\boldsymbol{h_1.x}$
- **Recursively** merge $\boldsymbol{h_2}$ with $\boldsymbol{h_1}$**.left** or $\boldsymbol{h_1}$**.right** , as we wish.

to decide we toss a coin





merge(h_1, h_2)



h
3
5
16
8
11
19
50
78
21
h1
h2
3
5
78
16
8
11
19
50
21
h
3
h1
h2
5
78
16
8
11
19
50
21
h
h2
78
3
5
16
8
h1
11
19
50
21

merge($\boldsymbol{h_1}, \boldsymbol{h_2}$)

merge($\boldsymbol{h_1}, \boldsymbol{h_2}$):

```
    if ($h_1$ = null) then return $h_2$;
```

merge($\boldsymbol{h_1}, \boldsymbol{h_2}$):

- if ($\boldsymbol{h_1} = \text{null}$) then return $\boldsymbol{h_2}$;
- if ($\boldsymbol{h_2} = \text{null}$) then return $\boldsymbol{h_1}$;
- if ($\boldsymbol{h_1.x} > \boldsymbol{h_2.x}$) then swap $\boldsymbol{h_1} \leftrightarrow \boldsymbol{h_2}$;
- if (coin comes up heads) then
  - $\boldsymbol{h_1}$.left = **merge**($\boldsymbol{h_1}$.left, $\boldsymbol{h_2}$);
  - $\boldsymbol{h_1}$.left.parent = $\boldsymbol{h_1}$;
- else
  - $\boldsymbol{h_1}$.right = **merge**($\boldsymbol{h_1}$.right, $\boldsymbol{h_2}$);
  - $\boldsymbol{h_1}$.right.parent = $\boldsymbol{h_1}$;
- return $\boldsymbol{h_1}$;

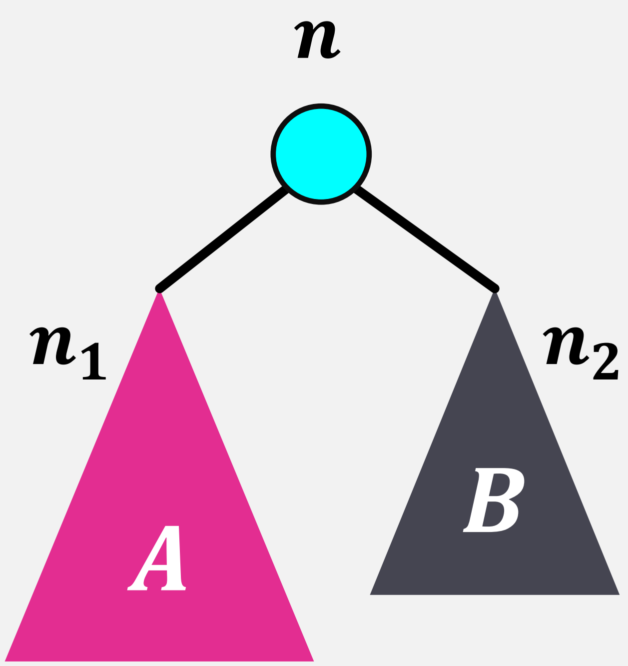
$\boldsymbol{O}(\log \boldsymbol{n})$

Analysis of merge($\boldsymbol{h_1}, \boldsymbol{h_2}$)

A **random walk** in a binary tree

- starts at the root of the tree.
- at each step a coin is tossed and, depending on the result, the walk proceeds to the **left** or to the **right** child of the current node.
- the walk ends when it falls off the tree

> Lemma 10.1:
> The **expected length** of a **random walk** in a binary tree with $\boldsymbol{n}$ nodes is at most $\mathbf{log}(\boldsymbol{n}+\mathbf{1})$.



$$\boldsymbol{n_1 + n_2 = n - 1}$$

add($\boldsymbol{x}$)

We create a new node $\boldsymbol{u}$ containing $\boldsymbol{x}$ and then merge $\boldsymbol{u}$ with the root of our heap

```
boolean add(x):
    Node<T> u = newNode();
    u.x = x;
    r = merge(u, r);
    r.parent = null;
    n++;
    return true;
```

$\boldsymbol{O}(\log \boldsymbol{n})$ expected time

removeMin()

The node we want to remove is the root, so we just merge its two children and make the result the root:

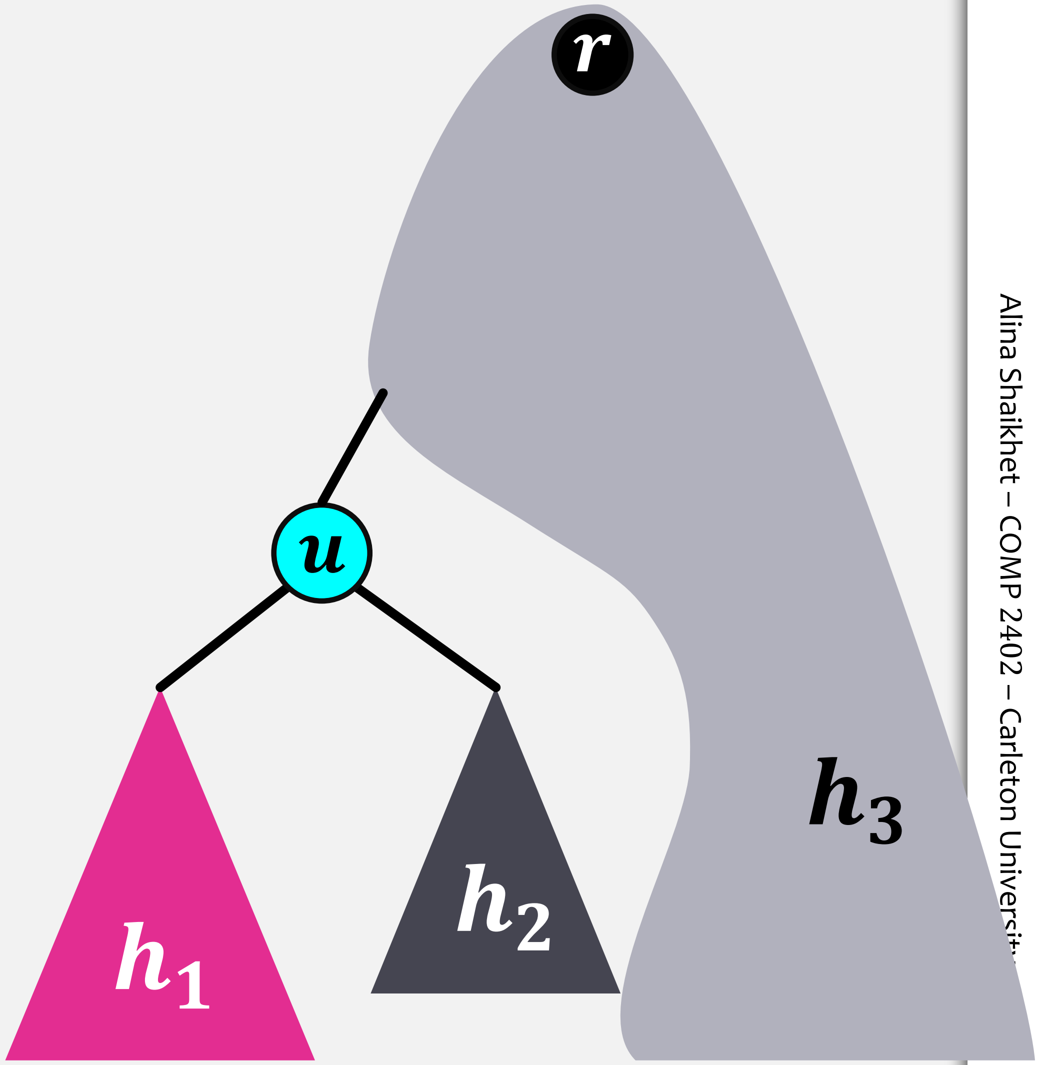
```
T removeMin():
    T x = r.x;
    r = merge(r.left, r.right);
    if (r ≠ null) then
        r.parent = null;
    n − −;
    return x;
```

$\boldsymbol{O}(\log \boldsymbol{n})$ expected time

remove($\boldsymbol{u}$)

Remove the node $\boldsymbol{u}$ (and its key $\boldsymbol{u.x}$) from the heap:

```
T remove (u):
    T x = u.x;
    Node h = merge(u.left, u.right);
    delete(u);
    r = merge(r, h);
    if (r ≠ null) then
        r.parent = null;
    n − −;
    return x;
```



$O(\log n)$ expected time

Theorem 10.2

A **MeldableHeap** implements the (priority) **Queue** interface.
A **MeldableHeap** supports the operations add($\boldsymbol{x}$) and removeMin() in $\boldsymbol{O}(\log \boldsymbol{n})$ **expected** time per operation.