<span id="page-0-0"></span>

- [Problems](#page-1-0)
- [Insertion](#page-5-0)
- [Extract Min](#page-7-0)
- [Implementation](#page-9-0)
- 
- [Push Down](#page-11-0)
- [Analysis](#page-12-0)
- [Construction](#page-13-0)
- [Creation Time](#page-16-0)
- 
- [Amortization](#page-19-0)

#### [Sorting](#page-20-0)

### **[Heaps](#page-0-0)**

School of Computing and Data Science - 1/16 - Frank Kreimendahl | *kreimendahlf@wit.edu* 



- [Problems](#page-1-0)
- **[Heaps](#page-4-0)** [Insertion](#page-5-0) [Extract Min](#page-7-0) [Implementation](#page-9-0) [Push Down](#page-11-0) [Analysis](#page-12-0)
- [Construction](#page-13-0) [Creation Time](#page-16-0)
- [Array Sizing](#page-18-0)
- [Amortization](#page-19-0)
- [Sorting](#page-20-0)

# <span id="page-1-0"></span>Motivating Problems

#### **1** Finding the min



- [Problems](#page-1-0)
- [Insertion](#page-5-0) [Extract Min](#page-7-0) [Implementation](#page-9-0) [Push Down](#page-11-0) [Analysis](#page-12-0) [Construction](#page-13-0)
- [Creation Time](#page-16-0)
- [Array Sizing](#page-18-0) [Amortization](#page-19-0)
- [Sorting](#page-20-0)

## Motivating Problems

#### **1** Finding the min

- 2 Finding the min with insertions
	-



- [Problems](#page-1-0)
- [Insertion](#page-5-0) [Extract Min](#page-7-0) [Implementation](#page-9-0) [Push Down](#page-11-0) [Analysis](#page-12-0) [Construction](#page-13-0)
- [Creation Time](#page-16-0)
- [Array Sizing](#page-18-0) [Amortization](#page-19-0)
- [Sorting](#page-20-0)

# Motivating Problems

- **1** Finding the min
- 2 Finding the min with insertions
- 3 Finding the min with insertions and deletions

<span id="page-4-0"></span>

#### [Heaps](#page-0-0)

[Problems](#page-1-0) [Heaps](#page-4-0) [Insertion](#page-5-0) [Extract Min](#page-7-0) [Implementation](#page-9-0) [Push Down](#page-11-0) [Analysis](#page-12-0) **[Construction](#page-13-0)** [Creation Time](#page-16-0) [Array Sizing](#page-18-0) [Amortization](#page-19-0)

[Sorting](#page-20-0)

Solve using a binary tree data structure

Heap invariant property: parent is smaller than (or equal to) both children

(This is specifically a min-heap. The max-heap invariant flips the inequality.)



### <span id="page-5-0"></span>Insertion



[Pull Up](#page-10-0) [Push Down](#page-11-0)

[Analysis](#page-12-0)

[Construction](#page-13-0)

[Creation Time](#page-16-0) [Array Sizing](#page-18-0)

[Amortization](#page-19-0)

[Sorting](#page-20-0)

#### 1 insert at bottom of tree



### Insertion



[Sorting](#page-20-0)

#### 1 insert at bottom of tree

2 re-establish invariant by pulling value up



## <span id="page-7-0"></span>Extract Min

- [Heaps](#page-0-0)
- [Problems](#page-1-0)
- [Heaps](#page-4-0)
- [Insertion](#page-5-0) [Extract Min](#page-7-0)
- [Implementation](#page-9-0)
- 
- [Push Down](#page-11-0)
- [Analysis](#page-12-0)
- [Construction](#page-13-0)
- [Creation Time](#page-16-0)
- [Array Sizing](#page-18-0) [Amortization](#page-19-0)
- [Sorting](#page-20-0)

#### 1 remove root

#### 2 move last node into root



## Extract Min

- [Heaps](#page-0-0) [Problems](#page-1-0) [Heaps](#page-4-0)
- [Insertion](#page-5-0) [Extract Min](#page-7-0)
- [Implementation](#page-9-0)
- 
- [Push Down](#page-11-0)
- [Analysis](#page-12-0)
- [Construction](#page-13-0) [Creation Time](#page-16-0)
- [Array Sizing](#page-18-0)
- [Amortization](#page-19-0)
- [Sorting](#page-20-0)
- 1 remove root
- 2 move last node into root
- 3 re-establish invariant by pushing value down

#### heapsort



#### [Heaps](#page-0-0) [Problems](#page-1-0) **[Heaps](#page-4-0)** [Insertion](#page-5-0) [Extract Min](#page-7-0) [Implementation](#page-9-0) [Push Down](#page-11-0)

[Analysis](#page-12-0) [Construction](#page-13-0) [Creation Time](#page-16-0) [Array Sizing](#page-18-0) [Amortization](#page-19-0)

[Sorting](#page-20-0)

# <span id="page-9-0"></span>Array Implementation of Tree

Rules for array index: parent of  $i = \lfloor \frac{i-1}{2} \rfloor$  $\frac{-1}{2}$ left child of  $i = 2i + 1$ right child of  $i = 2i + 2$ automatically balanced!



- Problem
- **[Heaps](#page-4-0)**
- [Insertion](#page-5-0)
- [Extract Min](#page-7-0)
- [Implementation](#page-9-0)
- [Pull Up](#page-10-0)
- [Push Down](#page-11-0)
- [Construction](#page-13-0)
- [Creation Time](#page-16-0)
- [Array Sizing](#page-18-0)
- [Amortization](#page-19-0)
- [Sorting](#page-20-0)

# <span id="page-10-0"></span>Pull Up – Insertion Helper

#### assume heap except at element *i*, *A*[*i*] might be too small

#### pullup(*i*)

- 1: **if**  $A[i] < A[parent_i]$  then
- 2: swap  $A[i]$  with  $A[parent_i]$
- 3: pullup(*parenti*)

#### invariant: initialization, maintenance, terminination



- Problem
- **[Heaps](#page-4-0)**
- [Insertion](#page-5-0)
- [Extract Min](#page-7-0)
- [Implementation](#page-9-0)
- [Pull Up](#page-10-0)
- [Push Down](#page-11-0)
- [Analysis](#page-12-0)
- [Construction](#page-13-0)
- [Creation Time](#page-16-0)
- [Array Sizing](#page-18-0)
- [Amortization](#page-19-0)
- [Sorting](#page-20-0)

# <span id="page-11-0"></span>Push Down – Extraction Helper

assume heap except at element *i*, *A*[*i*] might be too large

#### pushdown(*i*)

- 1:  $min_i \leftarrow$  index of smallest among *i* and valid children of *i*
- 2: **if**  $min_i \neq i$  **then**
- 3: swap  $A[i]$  with  $A[min_i]$
- 4: pushdown(*mini*)

#### invariant: initialization, maintenance, terminination



### <span id="page-12-0"></span>Analysis

- [Heaps](#page-0-0)
- [Problems](#page-1-0) [Heaps](#page-4-0) [Insertion](#page-5-0) [Extract Min](#page-7-0)
- [Implementation](#page-9-0)
- 
- [Push Down](#page-11-0) [Analysis](#page-12-0)
- [Construction](#page-13-0) [Creation Time](#page-16-0) [Array Sizing](#page-18-0) [Amortization](#page-19-0)
- [Sorting](#page-20-0)

#### **Correctness**

What is the space complexity?

What is the time complexity?



# <span id="page-13-0"></span>**Construction**

#### [Heaps](#page-0-0) [Problems](#page-1-0)

- [Heaps](#page-4-0) [Insertion](#page-5-0) [Extract Min](#page-7-0) [Implementation](#page-9-0)
- [Push Down](#page-11-0)
- [Analysis](#page-12-0)
- [Construction](#page-13-0) [Creation Time](#page-16-0)
- [Array Sizing](#page-18-0) [Amortization](#page-19-0)
- [Sorting](#page-20-0)

#### Given an array, how do we construct a heap?

School of Computing and Data Science - 10/16 - Frank Kreimendahl | *kreimendahlf@wit.edu* 



# **Construction**

#### [Heaps](#page-0-0) [Problems](#page-1-0) **[Heaps](#page-4-0)** [Insertion](#page-5-0) [Extract Min](#page-7-0) [Implementation](#page-9-0) [Push Down](#page-11-0) [Analysis](#page-12-0)

[Construction](#page-13-0) [Creation Time](#page-16-0) [Array Sizing](#page-18-0)

[Amortization](#page-19-0)

[Sorting](#page-20-0)

Given an array, how do we construct a heap? Can we do better than  $O(n \lg n)$  time?

School of Computing and Data Science - 10/16 - Frank Kreimendahl | *kreimendahlf@wit.edu* 



# **Construction**

[Heaps](#page-0-0) [Problems](#page-1-0) **[Heaps](#page-4-0)** [Insertion](#page-5-0) [Extract Min](#page-7-0) [Implementation](#page-9-0) [Pull Up](#page-10-0) [Push Down](#page-11-0) [Analysis](#page-12-0)

[Construction](#page-13-0) [Creation Time](#page-16-0) [Array Sizing](#page-18-0) [Amortization](#page-19-0)

[Sorting](#page-20-0)

Given an array, how do we construct a heap? Can we do better than  $O(n \lg n)$  time?

bottom up creation: **for** *i* from  $\frac{length}{2} - 1$  to 0 **do** pushdown(*i*)

what is this time complexity?

School of Computing and Data Science - 10/16 - Frank Kreimendahl | *kreimendahlf@wit.edu* 



Problem **[Heaps](#page-4-0)** [Insertion](#page-5-0) [Extract Min](#page-7-0) [Implementation](#page-9-0) [Pull Up](#page-10-0) [Push Down](#page-11-0)

[Construction](#page-13-0) [Creation Time](#page-16-0) [Array Sizing](#page-18-0) [Amortization](#page-19-0)

[Sorting](#page-20-0)

# <span id="page-16-0"></span>Heap Creation Time Complexity

*height* of a node is maximum distance to a leaf *count<sub>x</sub>* is the number of nodes in a tree with height  $x$ 

$$
\sum_{h=0}^{\lg n} (O(h) \times count_h)
$$

There are  $\frac{n}{2^{h+1}}$  nodes with height *h* 

$$
\sum_{h=0}^{\lg n} O(h) \frac{n}{2^{h+1}} = O\left(n \sum_{h=0}^{\lg n} \frac{h}{2^{h+1}}\right)
$$



## More Time Complexity



- [Problems](#page-1-0)
- [Heaps](#page-4-0)
- [Insertion](#page-5-0)
- [Extract Min](#page-7-0)
- [Implementation](#page-9-0)
- 
- [Push Down](#page-11-0)
- [Analysis](#page-12-0)
- [Construction](#page-13-0)
- [Creation Time](#page-16-0)
- [Array Sizing](#page-18-0)
- [Amortization](#page-19-0)
- [Sorting](#page-20-0)

∞ ∑ *h*=0 *h*  $\frac{n}{2^h} = 2$ , so lg*n* ∑ *h*=0 *h*  $\frac{n}{2^h}$  < 2

$$
O\left(n\sum_{h=0}^{\lg n}\frac{h}{2^{h+1}}\right) = O\left(n\sum_{h=0}^{\infty}\frac{h}{2^h}\right) = O(n)
$$



# <span id="page-18-0"></span>Sizing The Array

[Problems](#page-1-0) **[Heaps](#page-4-0)** [Insertion](#page-5-0) [Extract Min](#page-7-0)

[Heaps](#page-0-0)

[Implementation](#page-9-0)

[Pull Up](#page-10-0)

[Push Down](#page-11-0)

[Analysis](#page-12-0) [Construction](#page-13-0)

[Creation Time](#page-16-0)

[Array Sizing](#page-18-0) [Amortization](#page-19-0)

[Sorting](#page-20-0)

resize by doubling array size

'amortized' analysis: the 'accounting method'

1 start with array half full, 0 accrued steps

2 each insertion takes 3 steps

- insert self now
- b move self when array full
- c move an existing element when array full
- 3 when array is full, we have 1 move step for each item
- 4 after move, array is half full, 0 accrued steps

School of Computing and Data Science - 13/16 - Frank Kreimendahl | *kreimendahlf@wit.edu* 



### <span id="page-19-0"></span>Amortization

#### [Heaps](#page-0-0)

- [Problems](#page-1-0) **[Heaps](#page-4-0)** [Insertion](#page-5-0) [Extract Min](#page-7-0)
- [Implementation](#page-9-0)
- 
- [Push Down](#page-11-0) [Analysis](#page-12-0)
- [Construction](#page-13-0)
- [Creation Time](#page-16-0) [Array Sizing](#page-18-0)
- [Amortization](#page-19-0)

[Sorting](#page-20-0)

'amortized' analysis: the 'aggregate method' Let  $c_i = i$  if  $i - 1$  is a power of 2, 1 otherwise

$$
\sum_{i=1}^{n} c_i \le n + \sum_{j=0}^{\lg n} 2^j
$$
  

$$
< n + 2n
$$
  

$$
< 3n
$$

<span id="page-20-0"></span>

[Sorting](#page-20-0)

[Lower Bounds](#page-21-0)



School of Computing and Data Science - 15/16 - Frank Kreimendahl | *kreimendahlf@wit.edu* 

<span id="page-21-0"></span>

[Heaps](#page-0-0) [Sorting](#page-20-0)

[Lower Bounds](#page-21-0)

How many possible outputs are there for arranging *n* items?

School of Computing and Data Science - 16/16 - Frank Kreimendahl | *kreimendahlf@wit.edu* 



#### [Sorting](#page-20-0) [Lower Bounds](#page-21-0)

How many possible outputs are there for arranging *n* items? Binary decision tree with *n*! leaves



How many possible outputs are there for arranging *n* items? Binary decision tree with *n*! leaves has height at least  $lg(n!)$ 



How many possible outputs are there for arranging *n* items? Binary decision tree with *n*! leaves has height at least  $lg(n!)$ Stirling approximation:  $n! = \sqrt{2\pi n}(\frac{n}{e})$  $\frac{n}{e}$ <sup>n</sup>(1+ $\Theta(\frac{1}{n})$ )



How many possible outputs are there for arranging *n* items? Binary decision tree with *n*! leaves has height at least  $lg(n!)$ Stirling approximation:  $n! = \sqrt{2\pi n}(\frac{n}{e})$  $\frac{n}{e}$ <sup>n</sup>(1+ $\Theta(\frac{1}{n})$ ) so: √

 $lg(n!) = lg($  $\sqrt{2\pi n}(\frac{n}{e})$  $\frac{n}{e}$ <sup>n</sup>(1+ $\Theta(\frac{1}{n}))$ )  $\mathbf{u}(\mathbf{u}) = \mathbf{u}(\mathbf{v}^2 + \mathbf{v}^2)$  (1-<br>=  $\mathbf{u}(\sqrt{2\pi} + \mathbf{u})$   $\sqrt{n} + \mathbf{u}(\frac{\pi}{e})$ 

### Lower Bounds

How many possible outputs are there for arranging *n* items? Binary decision tree with *n*! leaves has height at least  $lg(n!)$ Stirling approximation:  $n! = \sqrt{2\pi n}(\frac{n}{e})$  $\frac{n}{e}$ <sup>n</sup>(1+ $\Theta(\frac{1}{n})$ ) so: √

$$
\mathbf{lg}(n!) = \mathbf{lg}(\sqrt{2\pi n}(\frac{n}{e})^n(1+\Theta(\frac{1}{n})))
$$
  
= 
$$
\mathbf{lg}\sqrt{2\pi} + \mathbf{lg}\sqrt{n} + \mathbf{lg}((\frac{n}{e})^n) + \mathbf{lg}(1+\Theta(\frac{1}{n}))
$$
  
= 
$$
\Theta(\mathbf{lg}\sqrt{n} + n\mathbf{lg}(\frac{n}{e}) + \mathbf{lg}(1+\Theta(\frac{1}{n})))
$$
  
= 
$$
\Theta(n\mathbf{lg}n)
$$
  
so comparison-based sorting takes 
$$
O(n\mathbf{lg}n)
$$
 tin

### Lower Bounds

How many possible outputs are there for arranging *n* items? Binary decision tree with *n*! leaves has height at least  $lg(n!)$ Stirling approximation:  $n! = \sqrt{2\pi n}(\frac{n}{e})$  $\frac{n}{e}$ <sup>n</sup>(1+ $\Theta(\frac{1}{n})$ ) so: √

$$
lg(n!) = lg(\sqrt{2\pi n}(\frac{n}{e})^n(1+\Theta(\frac{1}{n})))
$$
  
= lg $\sqrt{2\pi}$ +lg $\sqrt{n}$ +lg((\frac{n}{e})^n) + lg(1+\Theta(\frac{1}{n}))  
=  $\Theta$  (lg $\sqrt{n}$ +nlg( $\frac{n}{e}$ ) + lg(1+\Theta(\frac{1}{n})))  
=  $\Theta$  (nlgn)

## Lower Bounds

How many possible outputs are there for arranging *n* items? Binary decision tree with *n*! leaves has height at least  $lg(n!)$ Stirling approximation:  $n! = \sqrt{2\pi n}(\frac{n}{e})$  $\frac{n}{e}$ <sup>n</sup>(1+ $\Theta(\frac{1}{n})$ ) so: √

$$
lg(n!) = lg(\sqrt{2\pi n}(\frac{n}{e})^n(1+\Theta(\frac{1}{n})))
$$
  
= lg $\sqrt{2\pi}$ +lg $\sqrt{n}$ +lg((\frac{n}{e})^n)+lg(1+\Theta(\frac{1}{n}))  
=  $\Theta$ (lg $\sqrt{n}$ +nlg( $\frac{n}{e}$ ) + lg(1+\Theta(\frac{1}{n})))  
=  $\Theta(nlg n)$ 

School of Computing and Data Science - 16/16 - Frank Kreimendahl | *kreimendahlf@wit.edu* 

### Lower Bounds

How many possible outputs are there for arranging *n* items? Binary decision tree with *n*! leaves has height at least  $lg(n!)$ Stirling approximation:  $n! = \sqrt{2\pi n}(\frac{n}{e})$  $\frac{n}{e}$ <sup>n</sup>(1+ $\Theta(\frac{1}{n})$ ) so: √

$$
lg(n!) = lg(\sqrt{2\pi n}(\frac{n}{e})^n(1+\Theta(\frac{1}{n})))
$$
  
= lg $\sqrt{2\pi}$ +lg $\sqrt{n}$ +lg((\frac{n}{e})^n) + lg(1+\Theta(\frac{1}{n}))  
=  $\Theta$  (lg $\sqrt{n}$ +nlg( $\frac{n}{e}$ ) + lg(1+\Theta(\frac{1}{n})))  
=  $\Theta(nlg n)$ 

so comparison-based sorting takes Ω(*n*lg*n*) time