

- Proble
- Insertion
- Extract Min
- Implementatio
- Pull Up
- Push Dow
- Analysis
- Construction
- Creation Tim
- Array Sizing

Sorting

Heaps

School of Computing and Data Science



- Problems
- Insertion Extract Min Implementatic Pull Up Push Down Analysis Construction Creation Time

Array Sizing Amortization

Sorting

Motivating Problems

Finding the min

Finding the min with insertions

Finding the min with insertions and deletions



- Problems
- Insertion Extract Min Implementatio Pull Up Push Down Analysis Construction Creation Time
- Array Sizing
- Sorting

Motivating Problems

Finding the min

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



- Problems
- Insertion Extract Min Implementatio Pull Up Push Down Analysis Construction Creation Time
- Array Sizing Amortization

Sorting

Motivating Problems

Finding the min

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



Heaps

Heaps Insertion Extract Min Implementation Pull Up Push Down Analysis Construction Creation Time Array Sizing Amortization

Sorting

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.)



Insertion

Heaps Problems Heaps Extract Min Implementation Pull Up Push Down Analysis Construction Creation Time Array Sizing Amortization

Sorting

1 insert at bottom of tree

re-establish invariant by pulling value up



Insertion

Heaps Problems Heaps Insertion Extract Min Implementation Pull Up Push Down Analysis Construction Creation Time Array Sizing Amortization

Sorting

1 insert at bottom of tree

2 re-establish invariant by pulling value up



Extract Min

- Heaps Problems
- Insertion

Extract Min

- Implementation Pull Up Push Down Analysis Construction Creation Time Array Sizing
- Sorting

1 remove root

2 move last node into root

re-establish invariant by pushing value down

heapsort



Extract Min

Heaps Problems Heaps

Extract Min

- Implementation Pull Up Push Down Analysis Construction Creation Time
- Array Sizing
- Sorting

1 remove root

- 2 move last node into root
- **3** re-establish invariant by pushing value down

heapsort



Heaps Problems Heaps Insertion Extract Min Implementation Pull Up Push Down Analysis

Construction Creation Time

Array Sizing

Sorting

Array Implementation of Tree

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



- Probler
- Heaps
- Insertion
- Extract Min
- Implementa
- Pull Up
- Push Down Analysis Construction
- Creation Tim
- Array Sizing
- Amortizatior
- Sorting

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: $pullup(parent_i)$

invariant: initialization, maintenance, terminination

School of Computing and Data Science



- Proble
- Heaps
- Insertion
- Extract Mir
- Implementat
- Pull Up
- Push Down
- Analysis
- Construction
- Array Sizing
- Amortization
- Sorting

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: $pushdown(min_i)$

invariant: initialization, maintenance, terminination



Analysis

Heaps

Problems Heaps Insertion Extract Min Implementati Pull Up

Analysis

Construction Creation Time Array Sizing Amortization

Sorting

Correctness

What is the space complexity?

What is the time complexity?



Construction

Heaps

- Heaps Insertion Extract Min Implementation Pull Up Push Down
- Analysis
- Construction Creation Time
- Array Sizing Amortization
- Sorting

Given an array, how do we construct a heap?

Can we do better than $O(n \lg n)$ time?

bttom 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



Construction

Heaps Problems Heaps Insertion Extract Min Implementation Pull Up Pull Down

Construction Creation Time Array Sizing Amortization

Sorting

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

ottom 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



Construction

Heaps Problems Heaps Insertion Extract Min Implementation Pull Up Push Down Analysis

Construction Creation Time Array Sizing Amortization

Sorting

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?



Problems Heaps Insertion Extract Min Implementatio Pull Up Push Down Analysis

Creation Time

Array Sizing Amortization

Sorting

Heap Creation Time Complexity

height of a node is maximum distance to a leaf $count_x$ is the number of nodes in a tree with height x

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

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

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

School of Computing and Data Science



More Time Complexity



- Probler
- Heaps
- Insertion
- Extract Mi
- Implementati
- Pull Up
- Push Dow
- Analysis
- Constructi
- Creation Time
- Array Sizing
- Sorting

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

= O(n)



Heaps Problems Heaps

Array Sizing

Sorting

Sizing The Array

resize by doubling array size

'amortized' analysis: the 'accounting method'

start with array half full, 0 accrued steps

- 2 each insertion takes 3 steps
 - a 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



Amortization

Heaps

Problems Heaps Insertion Extract Min Implementa Pull Up

Push Dov

Analysis

Constituention

creation rin

Amortization

Sorting

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

$$\sum_{i=1}^{n} c_i \le n + \sum_{j=0}^{\lg n} 2^j$$
$$< n+2n$$
$$< 3n$$

School of Computing and Data Science



Heap

Sorting

Lower Bounds



School of Computing and Data Science



Sorting 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: $n! = \sqrt{2\pi n} (\frac{n}{2})^n (1 + \Theta(\frac{1}{n}))$ so:

$$\begin{split} &\lg(n!) = \lg(\sqrt{2\pi n} (\frac{n}{e})^n (1 + \Theta(\frac{1}{n}))) \\ &= \lg\sqrt{2\pi} + \lg\sqrt{n} + \lg\left((\frac{n}{e})^n\right) + \lg\left(1 + \Theta(\frac{1}{n})\right) \\ &= \Theta\left(\lg\sqrt{n} + n\lg(\frac{n}{e}) + \lg\left(1 + \Theta(\frac{1}{n})\right)\right) \\ &= \Theta\left(n\lg n\right) \\ &\text{so comparison-based sorting takes } \Omega(n\lg n) \text{ tir} \end{split}$$



Lower Bounds

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!)Stirling approximation: $n! = \sqrt{2\pi n} \left(\frac{n}{2}\right)^n \left(1 + \Theta\left(\frac{1}{n}\right)\right)$ so: $\log(n!) = \log(\sqrt{2\pi n} \left(\frac{n}{2}\right)^n \left(1 + \Theta\left(\frac{1}{n}\right)\right)$

$$\begin{split} &\lg(n!) = \lg(\sqrt{2\pi}n(\frac{n}{e})^n(1+\Theta(\frac{1}{n}))) \\ &= \lg\sqrt{2\pi} + \lg\sqrt{n} + \lg\left((\frac{n}{e})^n\right) + \lg\left(1+\Theta(\frac{1}{n})\right) \\ &= \Theta\left(\lg\sqrt{n} + n\lg(\frac{n}{e}) + \lg\left(1+\Theta(\frac{1}{n})\right)\right) \\ &= \Theta\left(n\lg n\right) \\ &\text{so comparison-based sorting takes } \Omega(n\lg n) \text{ tim} \end{split}$$



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

so: $lg(n!) = lg(\sqrt{2\pi n}(\frac{n}{e})^n(1+\Theta(\frac{1}{n})))$ $= lg\sqrt{2\pi} + lg\sqrt{n} + lg((\frac{n}{e})^n) + lg(1+\Theta(\frac{1}{n}))$ $= \Theta(lg\sqrt{n} + nlg(\frac{n}{e}) + lg(1+\Theta(\frac{1}{n})))$ $= \Theta(nlgn)$ so comparison-based sorting takes $\Omega(nlgn)$ time



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

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

so comparison-based sorting takes $\Omega(n \lg n)$ time



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

Heaps Sorting Lower Bounds

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

$$\begin{split} &\lg(n!) = \lg(\sqrt{2\pi n} \left(\frac{n}{e}\right)^n (1 + \Theta(\frac{1}{n}))) \\ &= \lg\sqrt{2\pi} + \lg\sqrt{n} + \lg\left(\left(\frac{n}{e}\right)^n\right) + \lg\left(1 + \Theta(\frac{1}{n})\right) \\ &= \Theta\left(\lg\sqrt{n} + n\lg\left(\frac{n}{e}\right) + \lg\left(1 + \Theta(\frac{1}{n})\right)\right) \\ &= \Theta\left(n\lg n\right) \end{split}$$

so comparison-based sorting takes $\Omega(n \lg n)$ time

Heaps Sorting Lower Bounds

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

$$\begin{split} &\lg(n!) = \lg(\sqrt{2\pi n} (\frac{n}{e})^n (1 + \Theta(\frac{1}{n}))) \\ &= \lg\sqrt{2\pi} + \lg\sqrt{n} + \lg\left((\frac{n}{e})^n\right) + \lg\left(1 + \Theta(\frac{1}{n})\right) \\ &= \Theta\left(\lg\sqrt{n} + n\lg(\frac{n}{e}) + \lg\left(1 + \Theta(\frac{1}{n})\right)\right) \\ &= \Theta\left(n\lg n\right) \end{split}$$

so comparison-based sorting takes $\Omega(n \lg n)$ time

Heaps Sorting Lower Bounds

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

$$\begin{split} &\lg(n!) = \lg(\sqrt{2\pi n} (\frac{n}{e})^n (1 + \Theta(\frac{1}{n}))) \\ &= \lg\sqrt{2\pi} + \lg\sqrt{n} + \lg\left((\frac{n}{e})^n\right) + \lg\left(1 + \Theta(\frac{1}{n})\right) \\ &= \Theta\left(\lg\sqrt{n} + n\lg(\frac{n}{e}) + \lg\left(1 + \Theta(\frac{1}{n})\right)\right) \\ &= \Theta\left(n\lg n\right) \end{split}$$

so comparison-based sorting takes $\Omega(n \lg n)$ time