Data Structures

BTree Analysis (and Heaps)

CS 225
Brad Solomon & G Carl Evans

October 9, 2023

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN
Department of Computer Science

Exam 3 (10/16 — 10/18)

Sign up now on Prairietest!

↳ Practice exam 's up

Cumulative content through end of BTrees (today)

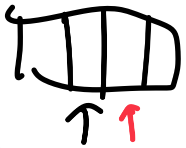
Coding question based on trees (know your tree labs!)

tree bst avl

mosaics

Learning Objectives

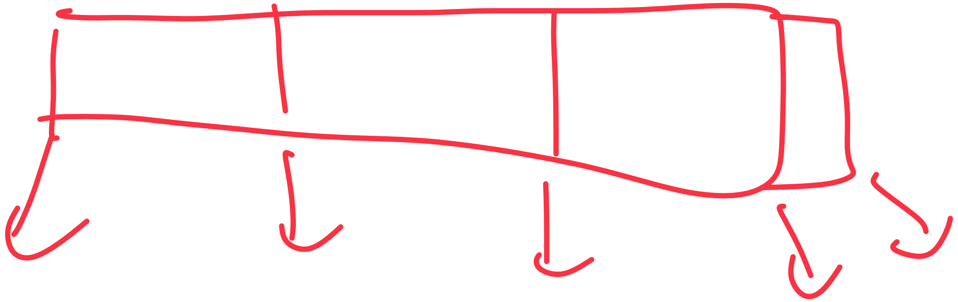
er



choose small or large

Analyze the performance of the BTree

Introduce a specialized data structure (discuss tradeoffs)



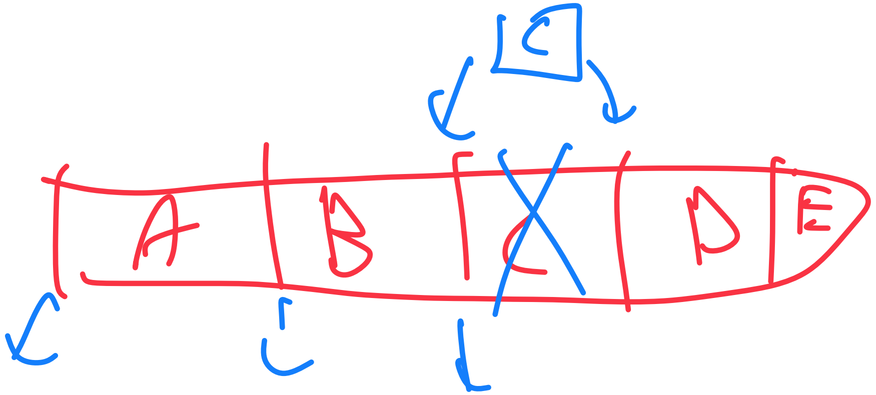
4 keys

5 children

$m = \cancel{4}\ 5$

yes split when

keys = 4



$\lceil \frac{m}{2} \rceil - 1$ keys

$\lceil m/2 \rceil$ children

Very important property!

BTree Properties

Minimize seek operations

A **BTrees** of order **m** is an m-ary tree and by definition:

- All keys within a node are ordered
- All nodes contain no more than **m-1** keys.
- All internal nodes have exactly **one more child than keys**

max

children = keys + 1

Root nodes can be a leaf or have **[2, m]** children.

All non-root, internal nodes have **[ceil(m/2), m]** children.

All leaves in the tree are at the same level.

BTree Analysis

of order 3

$\lceil m/2 \rceil - 1$ keys

$\lceil m/2 \rceil$ children for internal nodes

Let **n** be the number of keys in a BTree of order **m**.

What is our best approximation for the runtime for find? For insert?

m is constant that we set

$O(m)$

$O(\log m)$

size of nodes

m - 1

# of nodes is not # keys

$n = \#\text{ nodes} \cdot (m-1)$ [max]

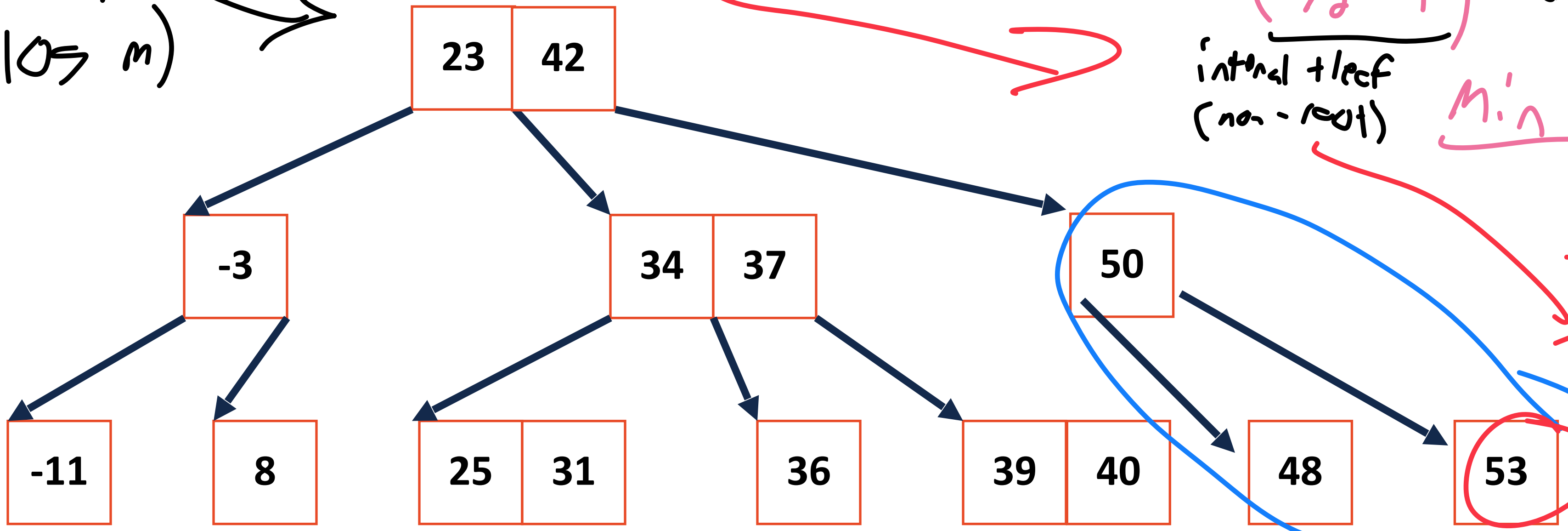
$\#\text{ nodes} \cdot (\lceil m/2 \rceil - 1)$

non-root

internal + leaf (non-root)

min

$O(h)$



BTree Analysis

or AVL

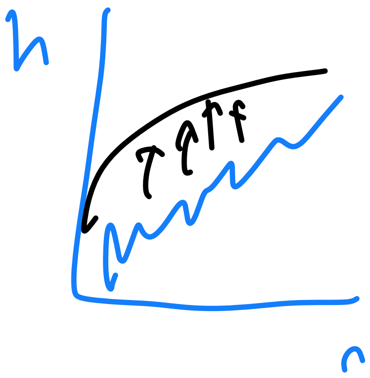
Like the BST, BTree height determines the runtime of our operations!

Claim: The BTree structure limits our height to $O(log_m(n))$

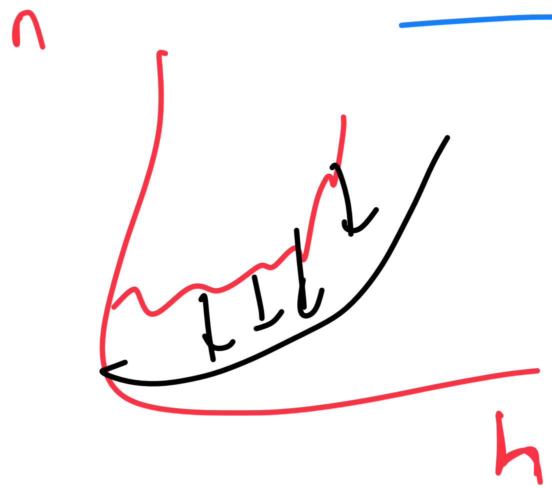
Proof: We want to find a relationship for BTrees between the number of keys (**n**) and the height (**h**).

estimate min n given h

estimating height given nodes is hard :(



Invert



BTree Analysis

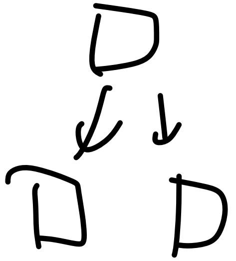
nodes have multiple keys

Strategy:

We will first count the number of nodes, level by level.

Then, we will add the minimum number of keys per node (**n**).

The minimum number of nodes will tell us the largest possible height (**h**), allowing us to find an upper-bound on height.



2 children == 1 key

Key Facts:

Root nodes can be a leaf or have **[2, m]** children.

All non-root, internal nodes have **[ceil(m/2), m]** children.

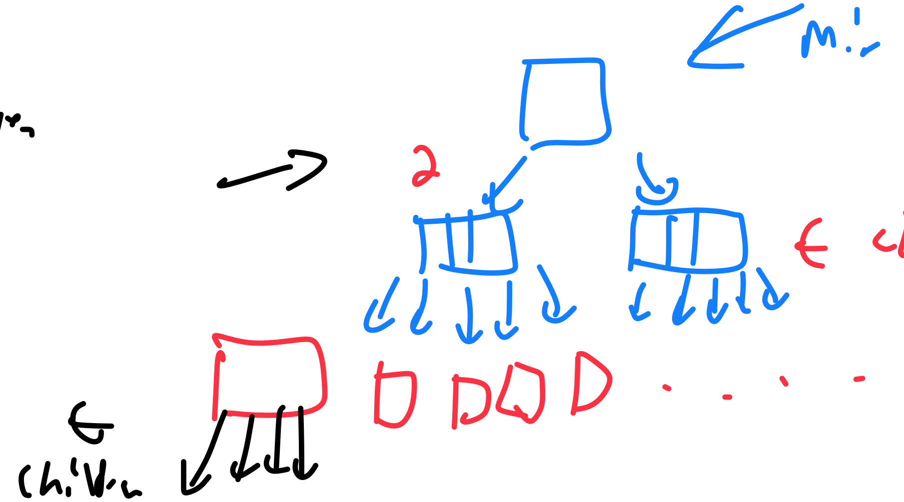
BTree Analysis

$t = \lceil \frac{m}{2} \rceil$ (# of children for all internal nodes)

(max)

Minimum number of **nodes** for a BTree of order m **at each level:** $m-1$

has one key → 2 children



t children each

2t children will have t children

Root: 1

Level 1: 2

Level 2: $2t$

Level 3: $2t^2$

Level h: $2t^{h-1}$

total # of nodes: $1 + \sum_{i=0}^{h-1} 2t^i$

$2 + 2t + 2t^2 + \ldots$

BTree Analysis

$1 \leq t = \lceil \frac{m}{2} \rceil$

The min **total number of nodes** is the sum of all the levels:

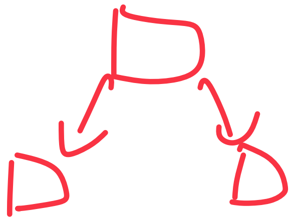
$$1 + 2\sum_{k=0}^{h-1} t^k = 1 + 2\left(\frac{t^h - 1}{t - 1}\right)$$

Summation identity

$$\sum_{i=0}^{n-1} x^i = \frac{x^n - 1}{x - 1}$$

↳ How can we relate # nodes to # keys?

BTree of order 2 is a BST



Note this doesn't work for m=2

m > 2

As long as m=3, BTree is efficient!

BTree Analysis

min

The **total number of nodes:**

root, leaf + internal

$$1 + 2\frac{t^h - 1}{t - 1}$$

$$t = \lceil \frac{m}{2} \rceil$$

The **total number of keys:**

root has how many keys? 1

internal nodes: $\lceil m/2 \rceil - 1 = t - 1$

leaf nodes: $\lceil m/2 \rceil - 1 = t - 1$

$$1 + 2\left(\frac{t^h - 1}{t - 1}\right) \cdot t - 1$$

$$= 1 + 2t^h - 2$$

$$= 2t^h - 1$$

min # keys in btree of height h

BTree Analysis

→ engine.. :)

For height h

The **smallest total number of keys** is: $2t^h - 1$

$t = \lceil \frac{m}{2} \rceil$

$t = \frac{m}{2}$

So an inequality about **n**, the total number of keys:

$$\log\left(\frac{n+1}{2}\right) \geq \log\left(\frac{2t^h - 1 + 1}{2}\right)$$

$$\log_t\left(\frac{n+1}{2}\right) \geq h$$

$$\log_{\frac{m}{2}}\left(\frac{n+1}{2}\right) \geq h$$

Solving for **h**, since **h** is the max number of seek operations:

It's ok to be imbalanced

the min is still efficient

$$h = \boxed{O(\log_m n)}$$

!!

m does matter but not in theory

BTree Analysis

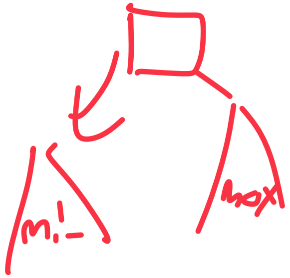
h=4 min vs max h=4

is ok!

12.5 m keys

Given **m=101**, a tree of height **h=4** has:

Minimum Keys: $2t^h - 1 = 2\lceil \frac{m}{2} \rceil^h - 1$



?? slightly larger

$2 \cdot 5\cancel{0}1^4 - 1 = \cancel{12}13.5$ million

Maximum Keys: Same logic but $t = m$

+ root is not 1 key but m-1 keys

$1 + (m + m^2 + m^3 + \ldots\ldots)$ nodes

keys

$m^{h+1} - 1$ keys

10.5 Billion

Somewhere between this

BTree

The BTree is still used heavily today!

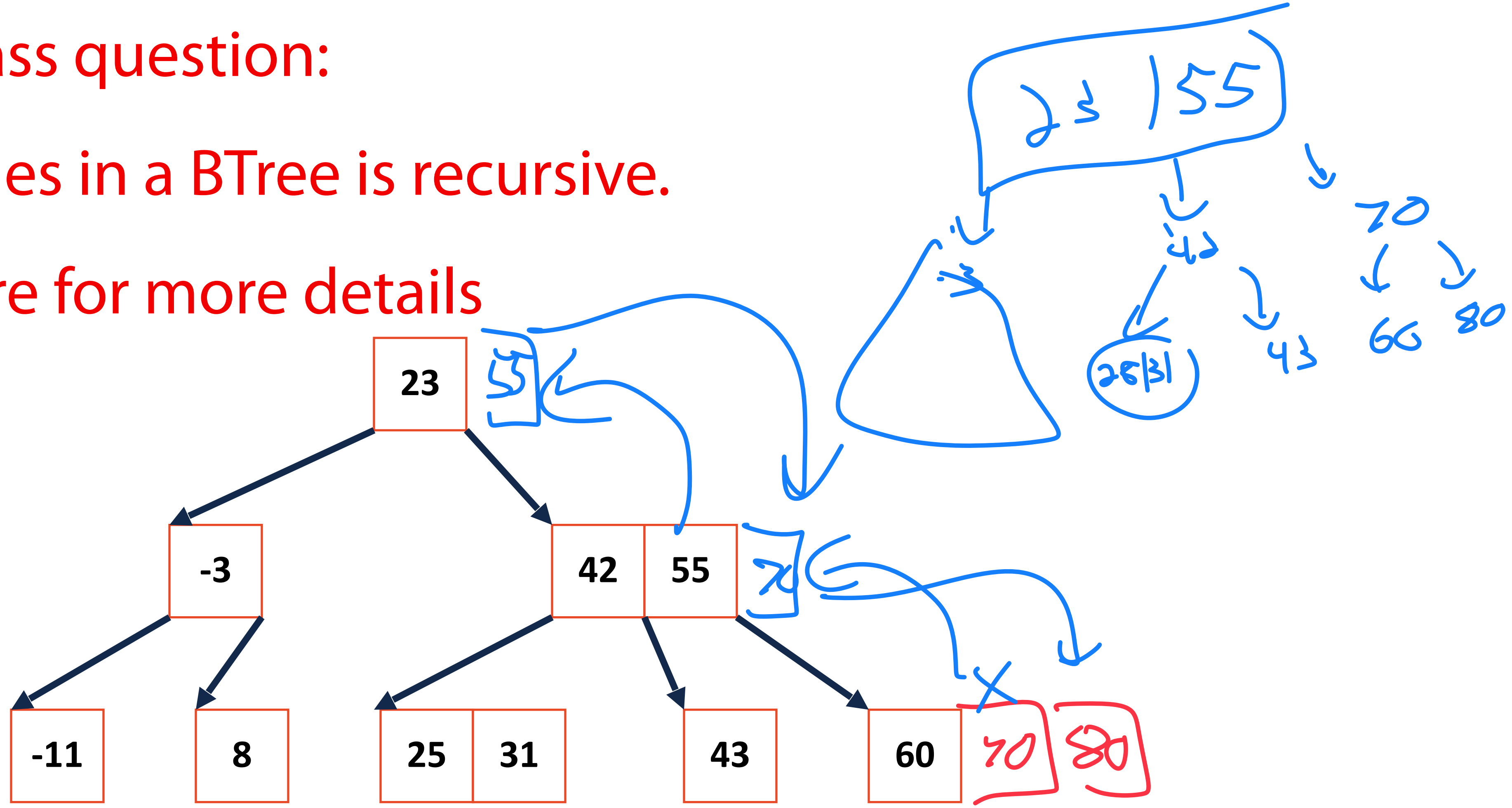
Improvements such as B+Tree and B*Tree exist far outside class scope

↳ Not be a final project!

Answer to in-class question:

Pushing up values in a BTree is recursive.

See insert lecture for more details



Thinking conceptually: Sorting a queue

How might we build a 'queue' in which our front element is the min?

After list we saw stack & queue → special case list!
↳ Tradeoff for speed
Lose random access

Build w/ unsorted list!
↳ Insert by append to end of list O(n) or O(1)
↳ Remove I have to find my next min and swap w/ front
↳ O(n)

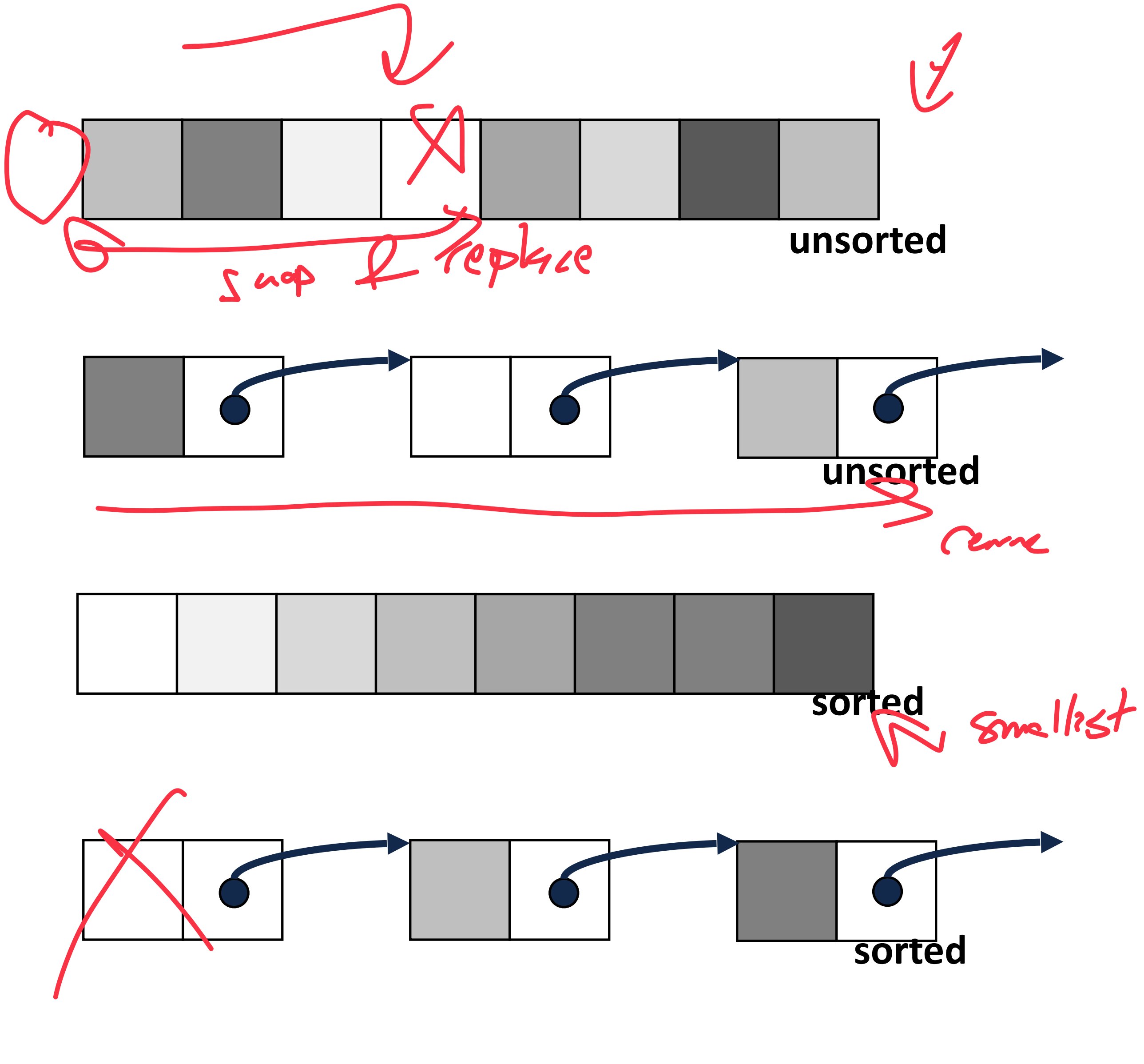
BTree / AVL tree (Sorted tree - ~~BST~~)
↳ Remove in log n
↳ Insert in log n
Find in (log n)

:)

Priority Queue Implementation

insert	removeMin
O(n)*	O(n)
O(1)	O(n)
O(n)	O(1)
O(n)	O(1)

slow fast

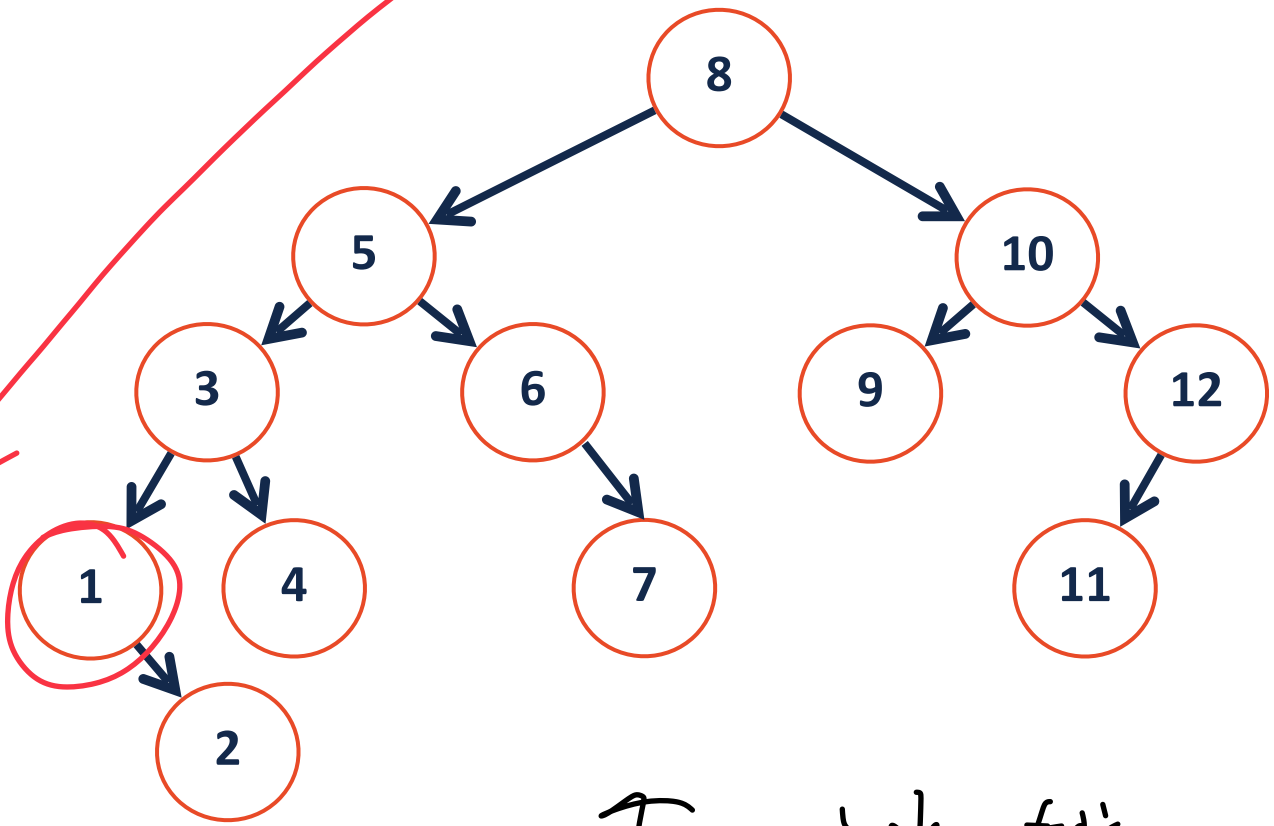


Priority Queue Implementation

insert	removeMin
O(log n)	O(log n)

1) Tree size in storage !!

2) I hate pointers



I say this object is a queue

↳ You (the user) can only insert/remove "front"

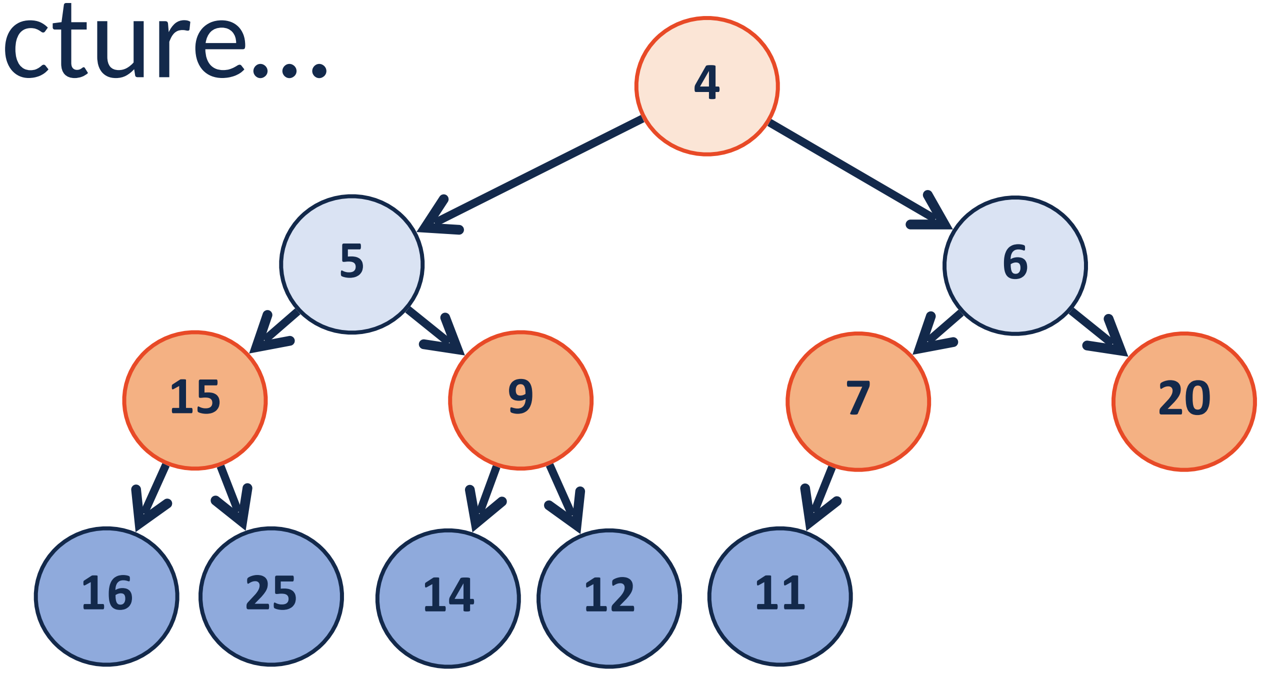
removeMin

our implementation can be a tree

Thinking conceptually: A tree without pointers

What class of (non-trivial) trees can we describe without pointers?

Another possibly structure…



(min)Heap

A complete binary tree T is a min-heap if:

- **T = {}** or
- **T = {r, T_L, T_R}**, where **r** is less than the roots of {**T_L, T_R**} and {**T_L, T_R**} are min-heaps.