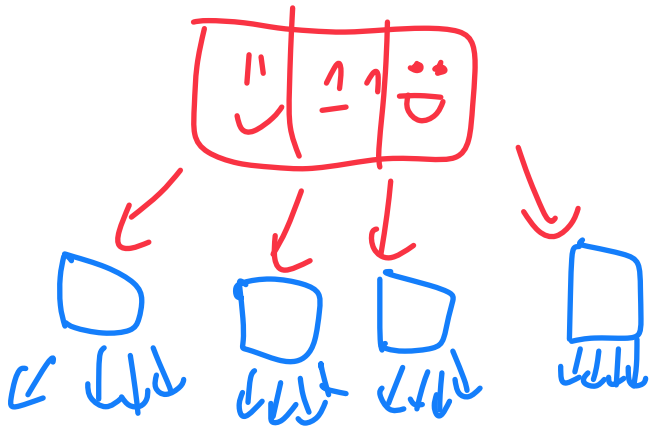
Data Structures

BTree Analysis

CS 225

Brad Solomon

March 6, 2026



Department of Computer Science

Announcements

Approved submission

[POTD EC Project](#) (1 + 5 EC Points)

↳ Propose & make a new POTD

↳ After approval, 2 weeks to complete

[EC Lab](#) (3 Points) → lab_trees!

↳ Bonus lab!

Learning Objectives

Review fundamentals of B Tree

Discuss the importance of M in a B Tree

Analyze the performance of the B Tree

Engineering vs Theory Efficiency

	Time x1 billion	Like
L1 cache reference	0.5 seconds	Heartbeat 💓
Branch mispredict	5 seconds	Yawn 😯
L2 cache reference	7 seconds	Long yawn 😯😯😯
Mutex lock/unlock	25 seconds	Make coffee ☕
Main memory reference	100 seconds	Brush teeth
Compress 1K bytes	50 minutes	TV show 📺
Send 2K bytes over 1 Gbps network	5.5 hours	(Brief) Night's sleep 🛌
SSD random read	1.7 days	Weekend
Read 1 MB sequentially from memory	2.9 days	Long weekend
Read 1 MB sequentially from SSD	11.6 days	2 weeks for delivery 📦
Disk seek	16.5 weeks	Semester
Read 1 MB sequentially from disk	7.8 months	Human gestation 🐣
Above two together	1 year	🌍☀️
Send packet CA->Netherlands->CA	4.8 years	Ph.D. 🎓

(Care of https://gist.github.com/hellerbarde/2843375)

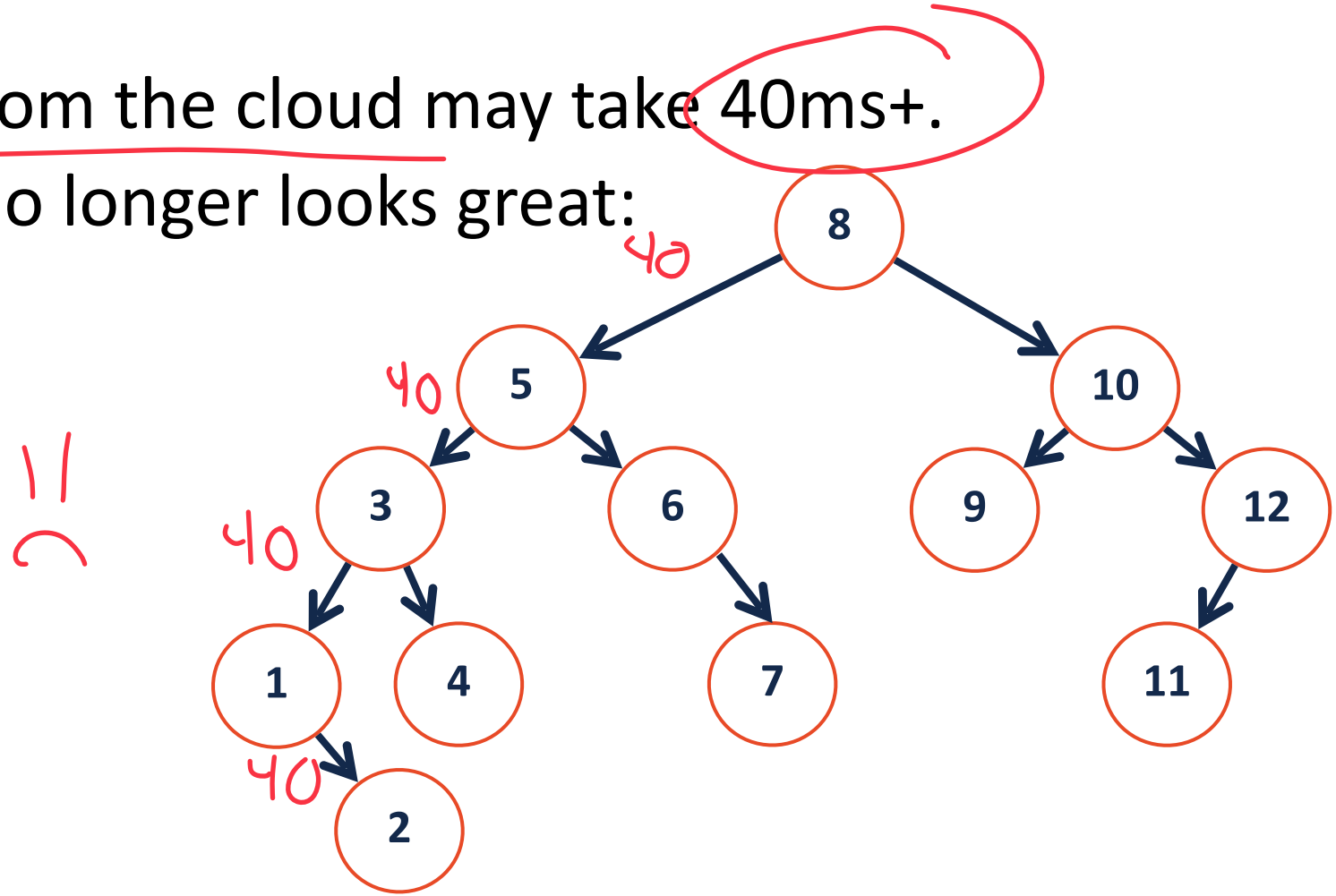
Engineering vs Theory Efficiency

In Big-O we have assumed uniform time for all operations, but this isn't always true.

0.0001 ms L1 cache
vs

However, seeking data from the cloud may take 40ms+.

…an O(lg(n)) AVL tree no longer looks great:



BTree Design Motivations

When large seek times become an issue, we address this by:

1) Keep the number of seeks low ← B Tree

2) When possible keep data stored locally → array

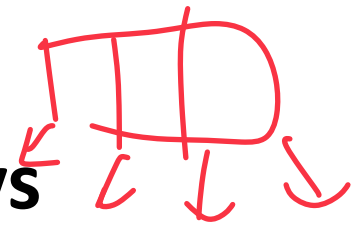
node qs

3) Make sure the data we look up is relevant! → Search tree

BTree Properties

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**
- All leaves in the tree are at the same level.



Today: How does the above lead to these improvements?

1) Keep the number of seeks low

2) When possible keep data stored locally

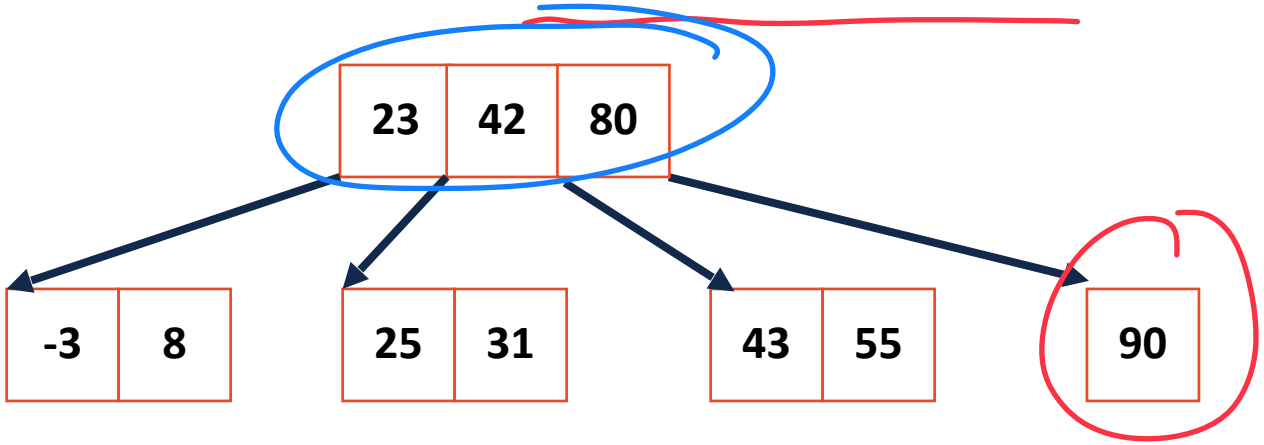
3) Make sure the data we look up is relevant!

Before we get there… lets review!

M = # children

Join Code: 225

What values of M are allowed in the following tree?



3] low

4

5

6] high

$M \leq 3$

Lower bound on M:

↳ Look for largest node

$M \geq 4$

upper bound on M:

↳ Look for smallest non-root node

$M \leq 4$

min # keys

$\lceil \frac{M}{2} \rceil - 1$

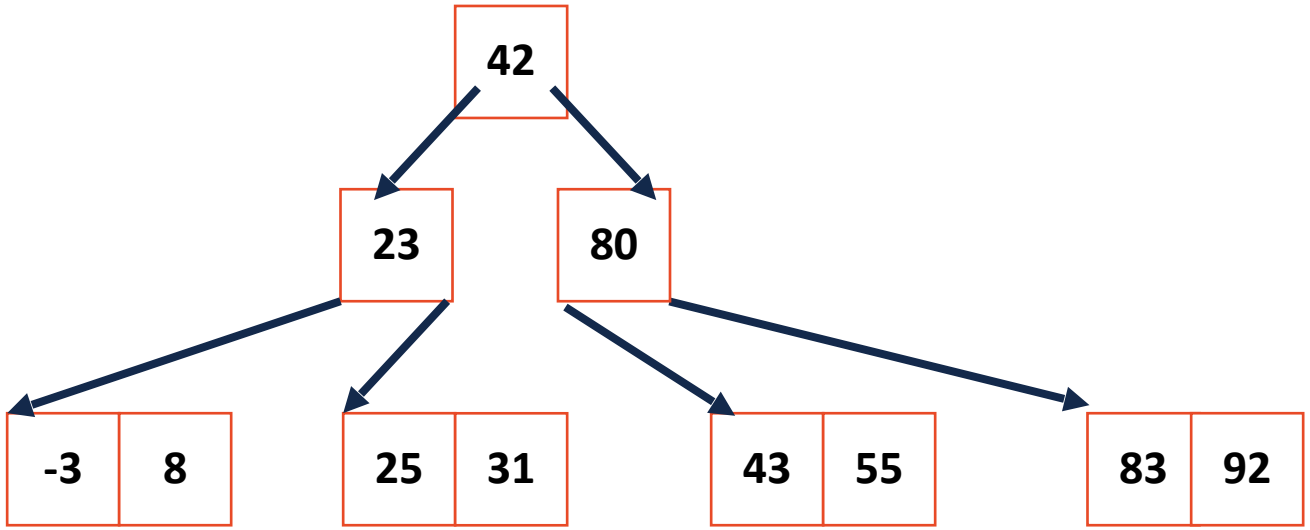
$\lceil \frac{M}{2} \rceil - 1 = 1$

$\lceil \frac{n}{2} \rceil = 2$

Before we get there… lets review!

Join Code: 225

What values of M are allowed in the following tree?



3
4
high

5
6
low

Largest # keys in node: 2

↳ $M \geq 3$

$3 \leq M \leq 4$

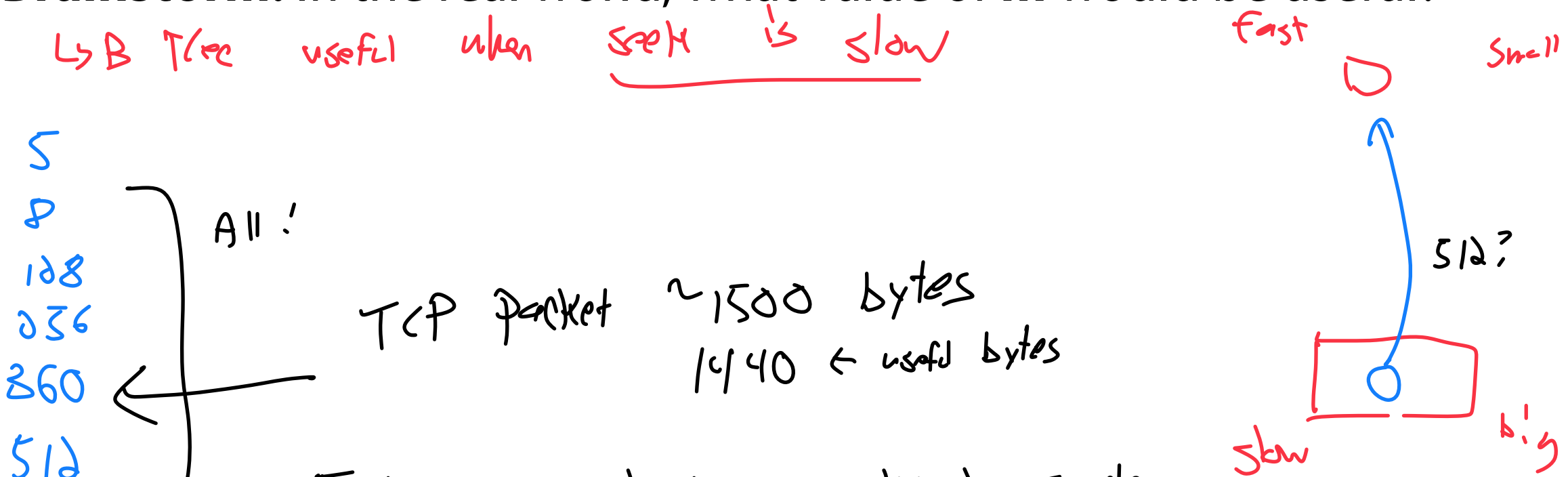
Smallest # keys in non-root: 1

↳

$\lceil \frac{M}{2} \rceil - 1 = 1 \rightarrow M \leq 4$

BTree of Order M (In Practice)

Brainstorm: In the real world, what value of **m** would be useful?



Join Code: 225

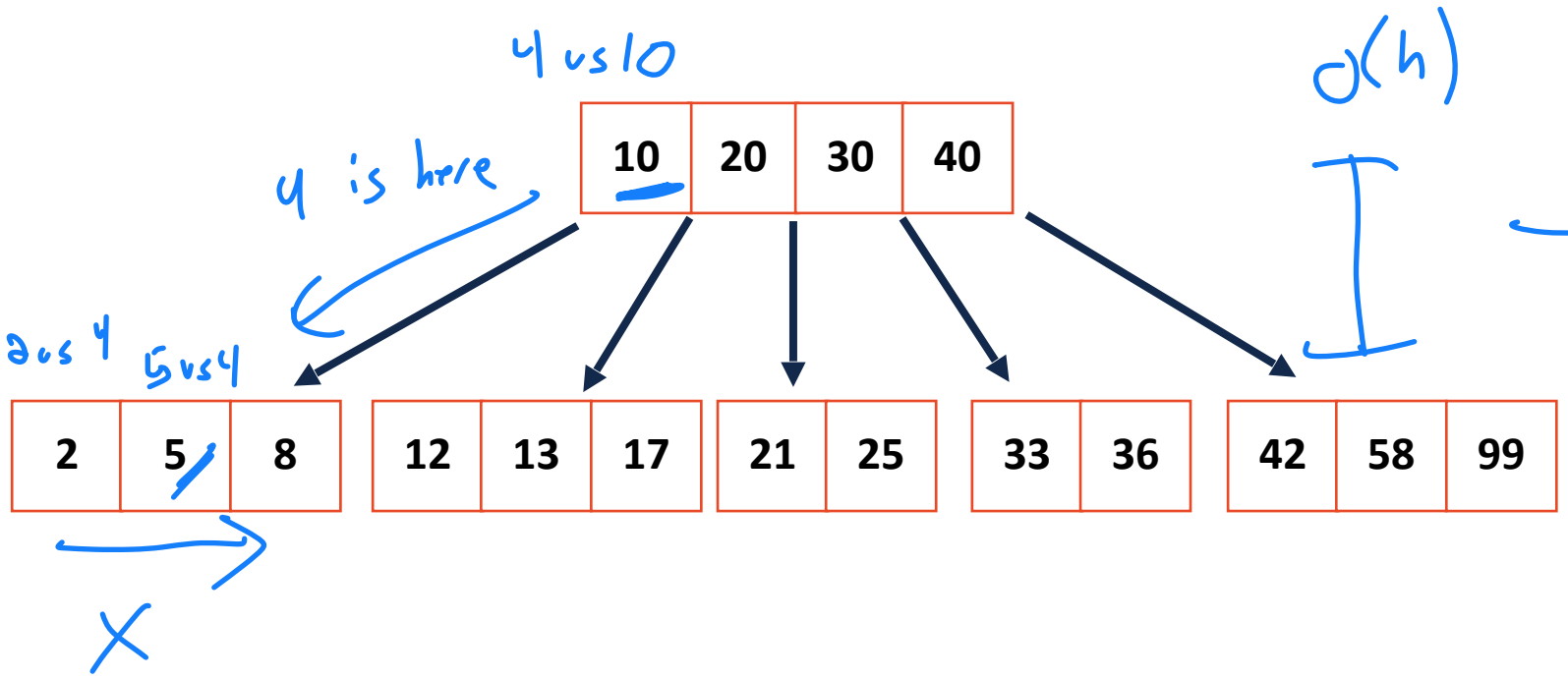
BTree of Order M

Questions about M?

BTree Find

How many keys are looked at to complete this search?

Join Code: 225



Find(4)

1) Start @ root

↳ sorted array search for 4 or first key value

3 total comparisons

↳ 4 not in BTree!

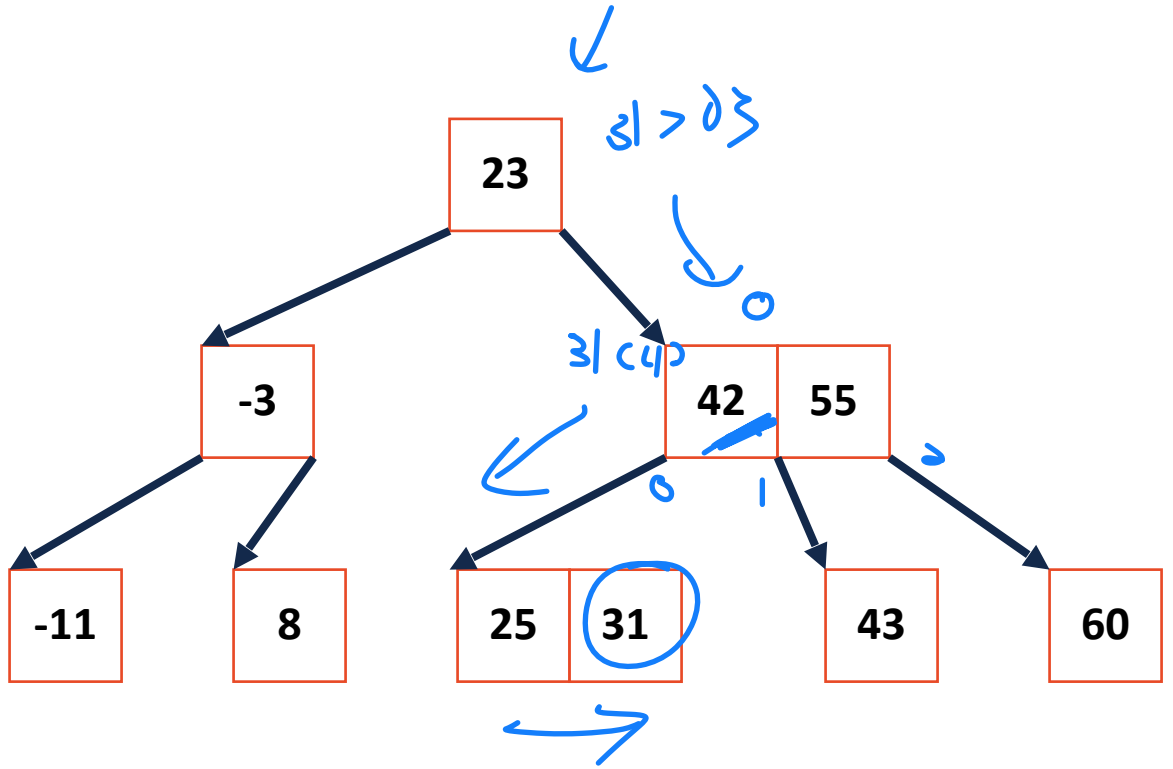
BTree Find

Find(31)

Base Cases:

If root is empty, return

If leaf, do array find() and return



Recursive Step:

Array find() for match or first greater value

Recurse on appropriate child

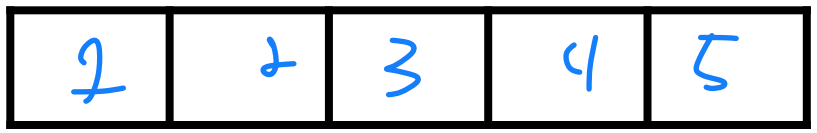
Tip: Index of first greater value is index of child we want to visit!

BTree Insertion

M = 5

Given appropriate leaf, insert is sorted array insert

```
Insert(1)
Insert(2)
Insert(3)
Insert(4)
Insert(5)
Insert(6)
Insert(7)
Insert(8)
```



BTree Insertion

M = 5

Given appropriate leaf, insert is sorted array insert

Once Mth item is inserted, raise up and on value

Insert(1)

Insert(2)

Insert(3)

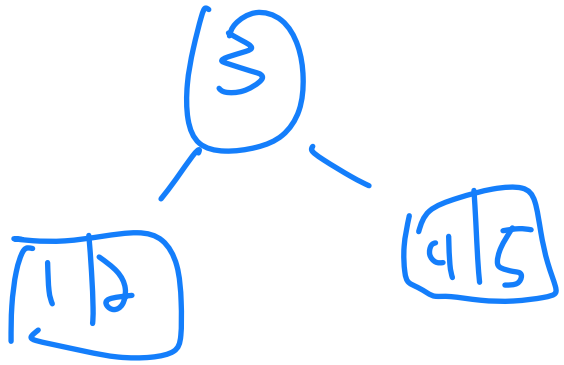
Insert(4)

Insert(5)

Insert(6)

Insert(7)

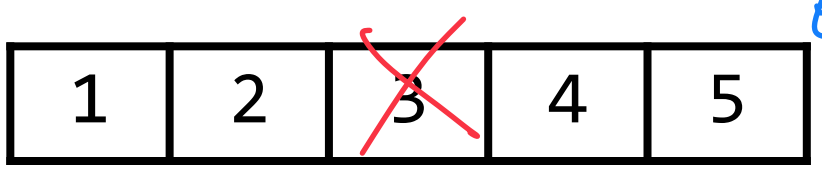
Insert(8)



$\lceil \frac{M}{2} \rceil - 1$ Keys thing

↳ we don't split until M items

non-optimized design decision!



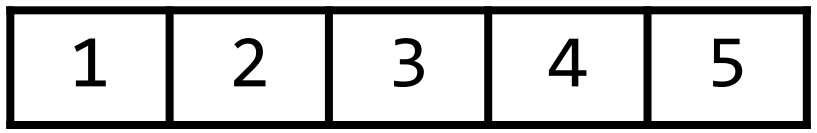
For LAB: Why is this size 5 and not size 4?

BTree Insertion

M = 5

When we hit **M** items, split into three nodes!

```
Insert(1)
Insert(2)
Insert(3)
Insert(4)
Insert(5)
Insert(6)
Insert(7)
Insert(8)
```

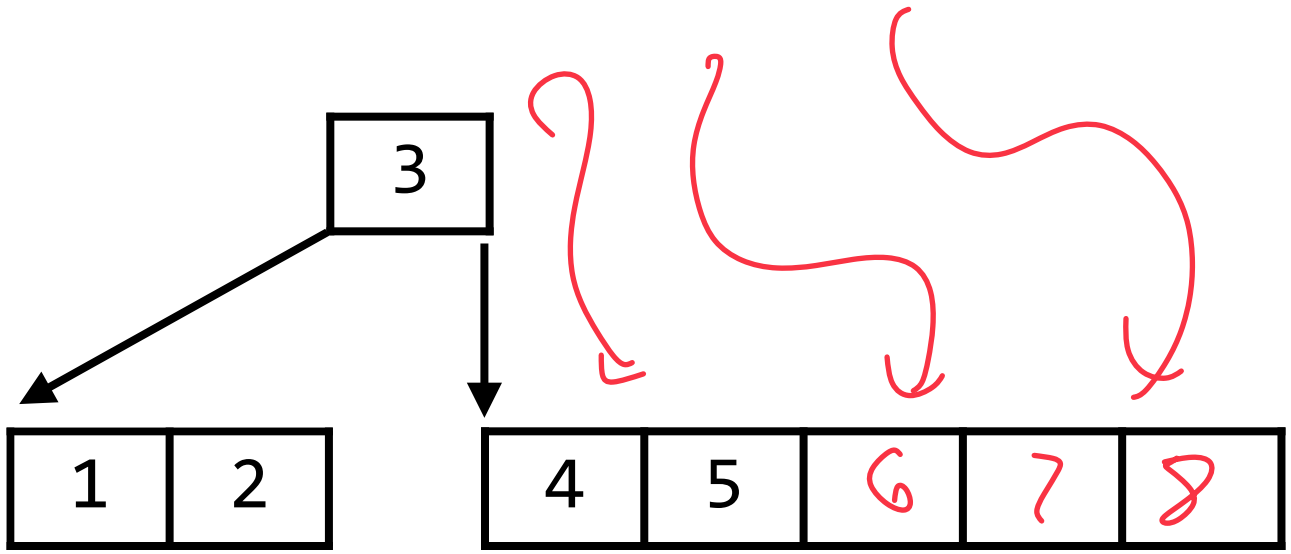


BTree Insertion

M = 5

In larger trees, use BTree Find() to get appropriate leaf

```
Insert(1)
Insert(2)
Insert(3)
Insert(4)
Insert(5)
Insert(6)
Insert(7)
Insert(8)
```

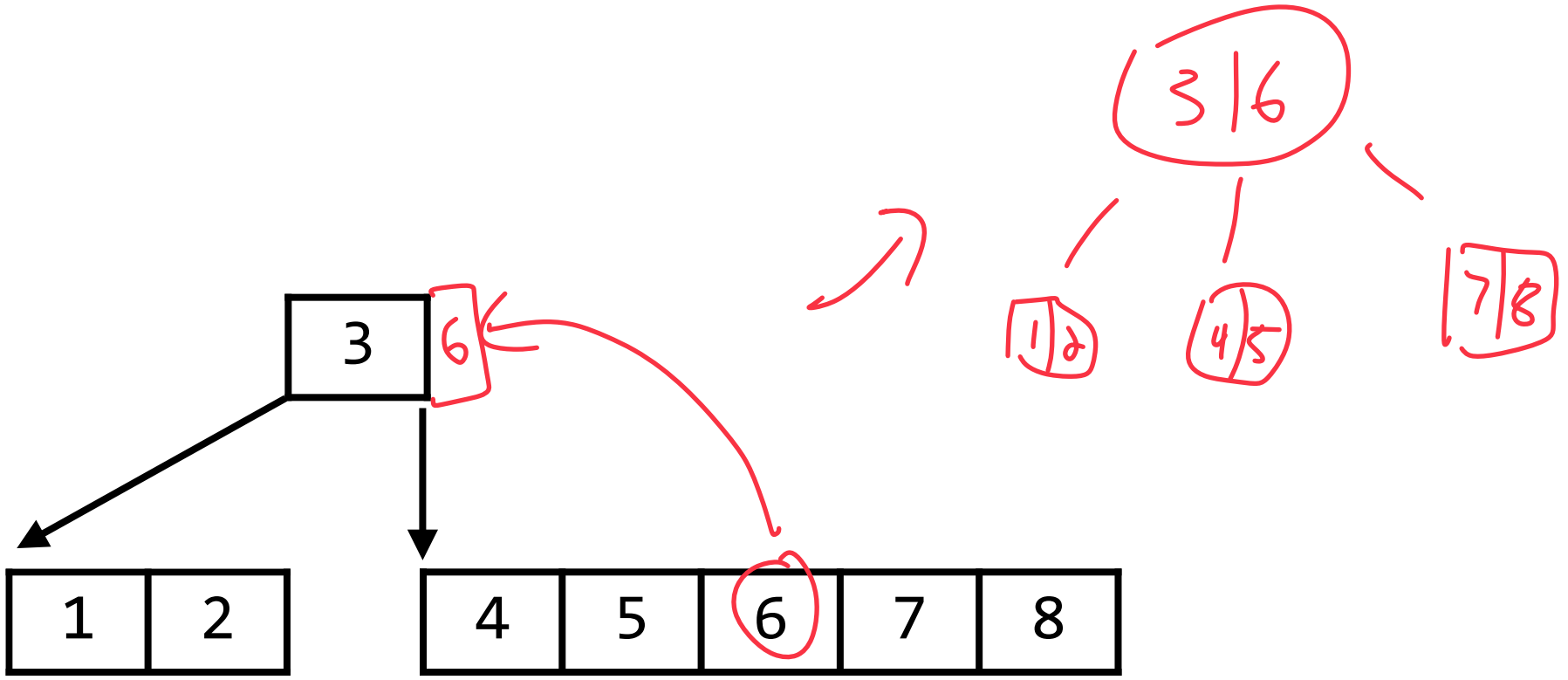


BTree Insertion

M = 5

If parent node already exists, split adds new key.

```
Insert(1)
Insert(2)
Insert(3)
Insert(4)
Insert(5)
Insert(6)
Insert(7)
Insert(8)
```

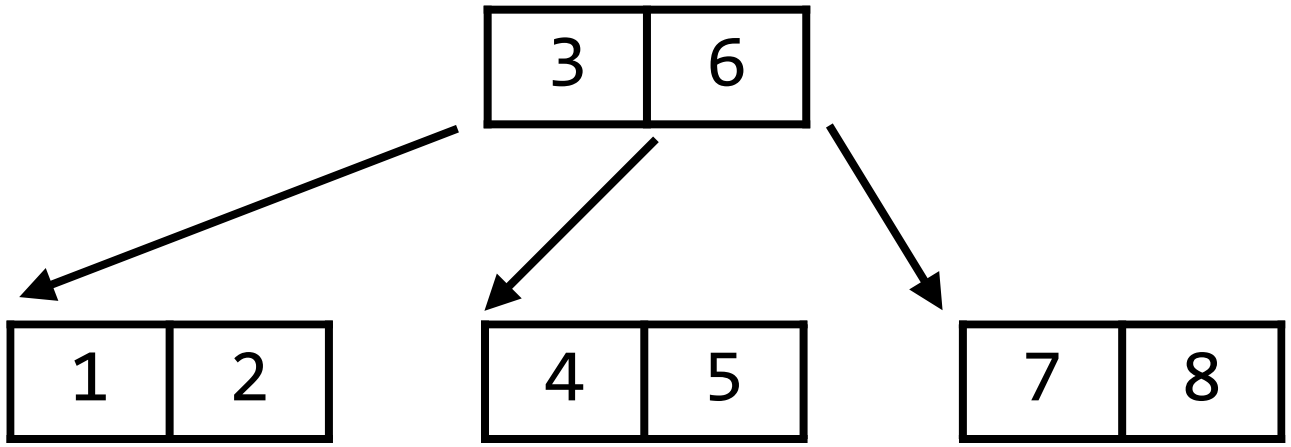


BTree Insertion

M = 5

If parent node already exists, split instead adds new key.

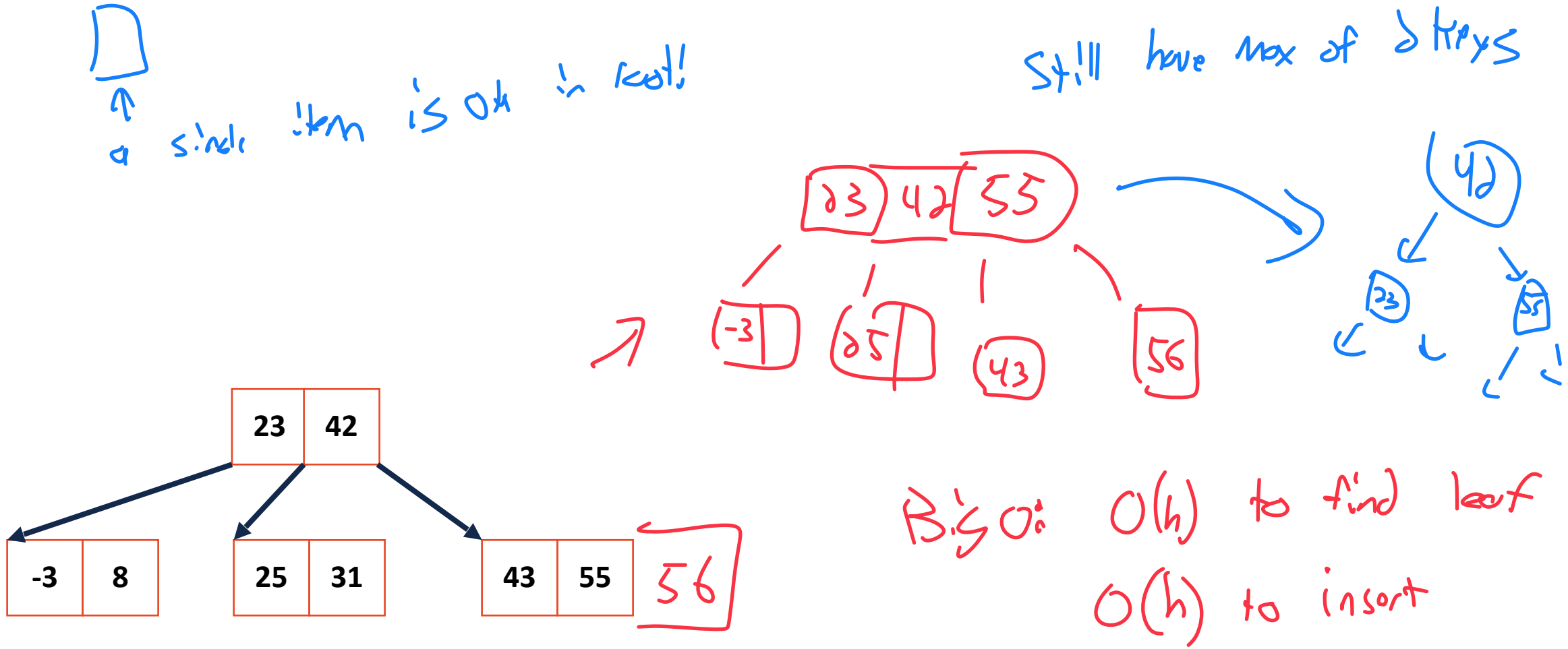
```
Insert(1)
Insert(2)
Insert(3)
Insert(4)
Insert(5)
Insert(6)
Insert(7)
Insert(8)
```



BTree Recursive Insert

`Insert(56), M = 3`

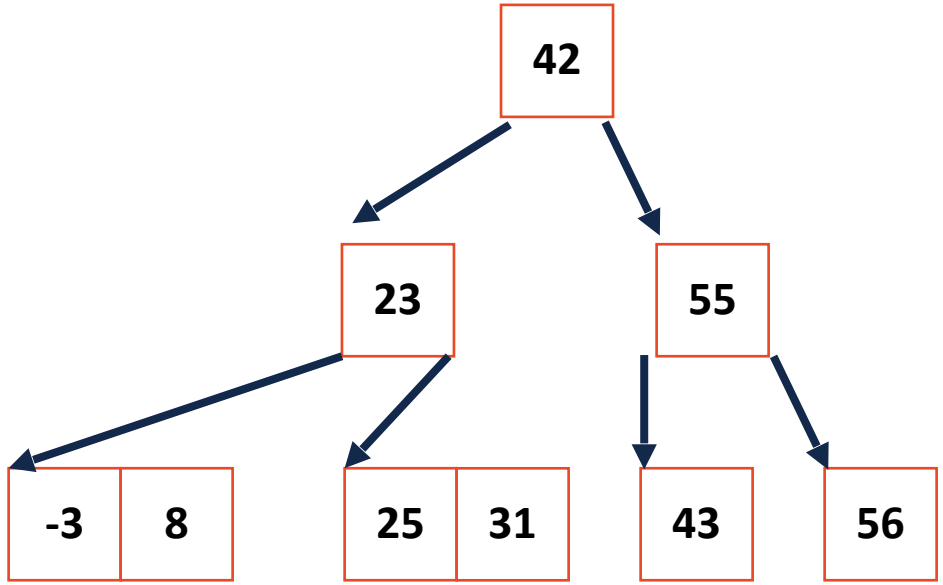
Insert always starts at a leaf but can propagate up repeatedly.



BTree Recursive Insert

`Insert(56), M = 3`

Insert always starts at a leaf but can propagate up repeatedly.



BTree Remove

BTree removal is complicated! **It won't be part of the lab.**

If we have time at the end of the day today we will discuss it

If

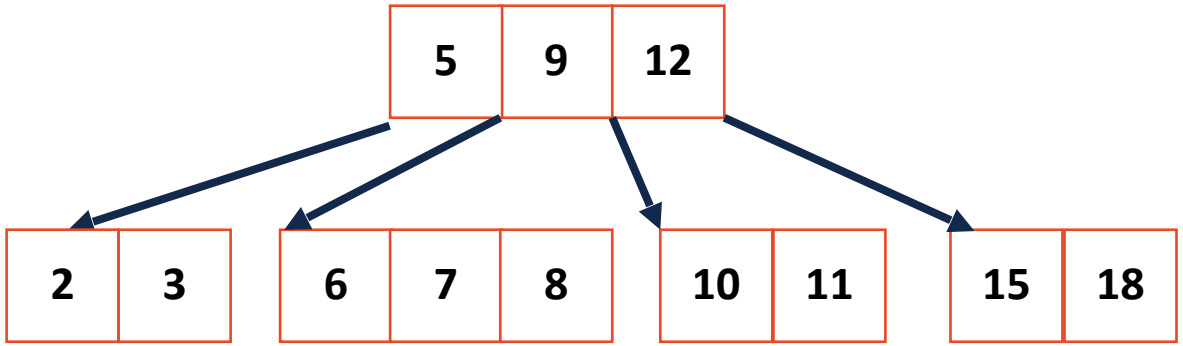
If

If

If

If

Boring!



BTree Implementation

Any questions about BTree implementation details?

BTree Properties

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**
- All leaves in the tree are at the same level.

Today: How does the above lead to these improvements?

1) Keep the number of seeks low

2) When possible keep data stored locally

3) Make sure the data we look up is relevant!

BTree Analysis

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**).

BTree Analysis

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.

Key Facts:

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

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

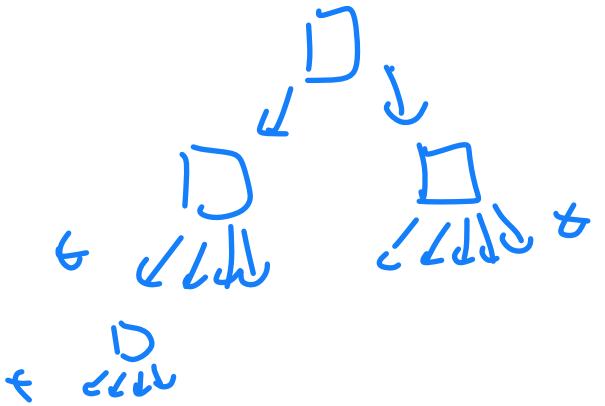
BTree Analysis

height h

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

min # of children for any non root internal node

Min number of **nodes** for a BTree of order m **at each level:**



Root: 1

Level 1: 2

Level 2: $2t$

Level 3: $2t^2$

...

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

BTree Analysis

Min number of **nodes** for a BTree of order m **at each level:** $t = \lceil \frac{m}{2} \rceil$

Root: 1

Level 1: 2 (Each of these will have t children)

Level 2: $2t$ (Each of these will have t children)

Level 3: $2t^2$ (Each of these will have t children)

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

BTree Analysis

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

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

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

root

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

this summation equation

(Don't memorize)

BTree Analysis

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

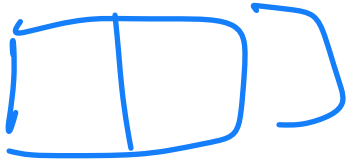
The **min total number of nodes** is the sum of all the levels:

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

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

total # nodes in BTree

Target why root not 1 child?



insert splits one to three Nodes

BTree Analysis

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

The **min total number of nodes**: $1 + 2\frac{t^h - 1}{t - 1}$

The **min total number of keys**:

BTree Analysis

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

The **min total number of nodes**: $1 + 2\frac{t^h - 1}{t - 1}$

The **min total number of keys**:

Root has how many keys? **1**

Internal nodes? $\lceil \frac{m}{2} \rceil - 1 = t - 1$

Leaf nodes? $\lceil \frac{m}{2} \rceil - 1 = t - 1$

So we can multiply the fraction by $t - 1$

$$= 1 + 2\frac{t^h - 1}{t - 1} * (t - 1)$$

$$= 2t^h - 1$$

min # keys in BTree

BTree Analysis

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

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

n & h

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

$n = \#\ keys \geq 2t^h - 1$

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

BTree Analysis

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

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

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

$$n \geq 2t^h - 1$$

+1

$$n + 1 \geq 2t^h$$

plug in for t

$\log_m$

$$\log_m (n+1) \geq \log_m \left(2\lceil \frac{m}{2} \rceil^h\right) = \log_m (m^h) = h$$

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

$$h = O(\log_m n)$$

BTree Analysis

This is very powerful!

As long as I am *at least* minimally sized, we are O(log n)!

BTree Analysis

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

Minimum Keys:

Maximum Keys:

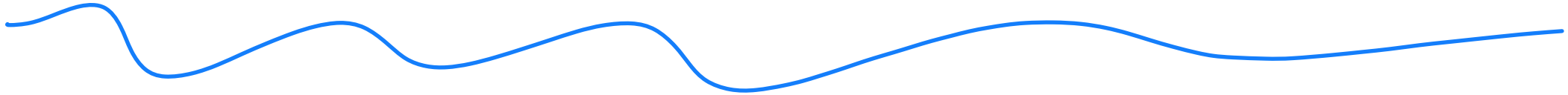
BTree

The BTree is still used heavily today!

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

Thinking conceptually: Sorting a queue

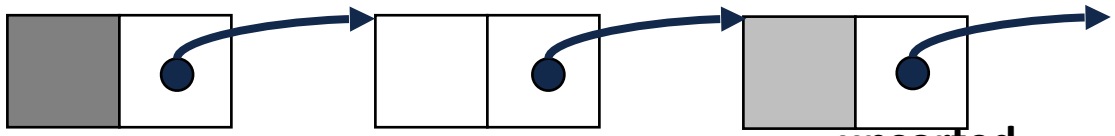
How might we build a ‘queue’ in which our front element is the min?



Priority Queue Implementation

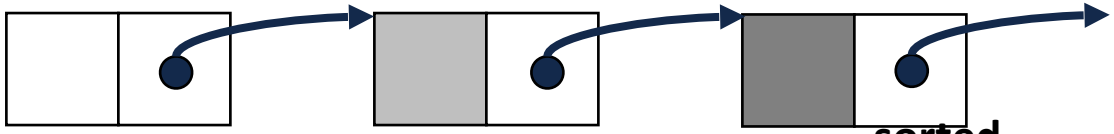
insert	removeMin
O(n)	O(n)
O(1)	O(n)
O(n)	O(1)
O(n)	O(1)

unsorted



unsorted

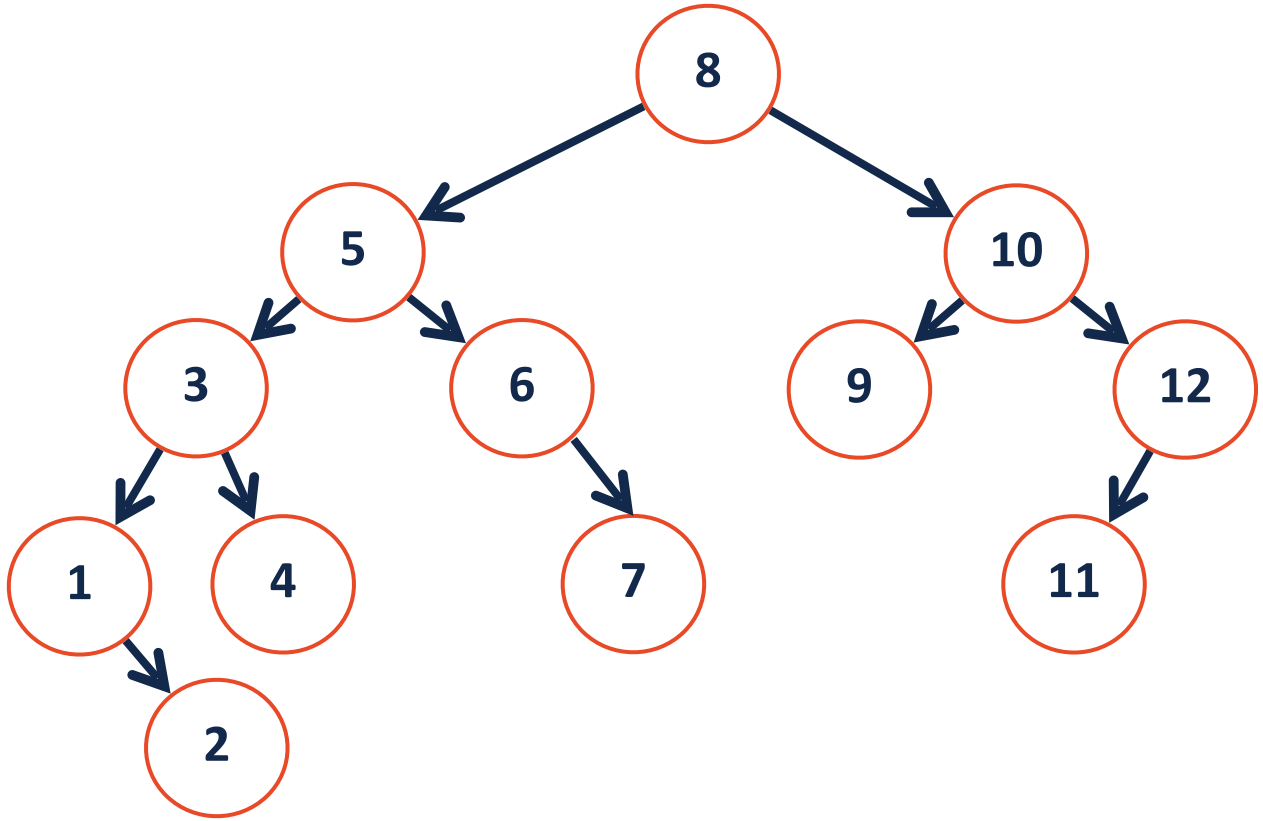
sorted



sorted

Priority Queue Implementation

insert	removeMin



Thinking conceptually: A tree without pointers

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