

Data Structures

BTree Analysis

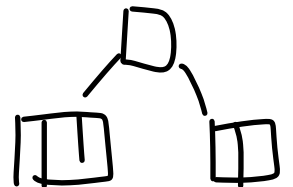
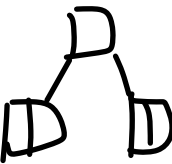
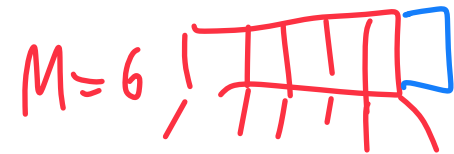
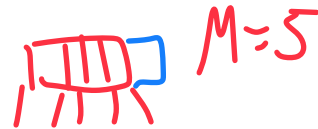
CS 225
Brad Solomon

October 9, 2026



UNIVERSITY OF
ILLINOIS
URBANA - CHAMPAIGN

Department of Computer Science



Next Week Mattox Lectures!

I will return around time to introduce graphs.

I will also join in on the exam 3 review lecture

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. 🎓

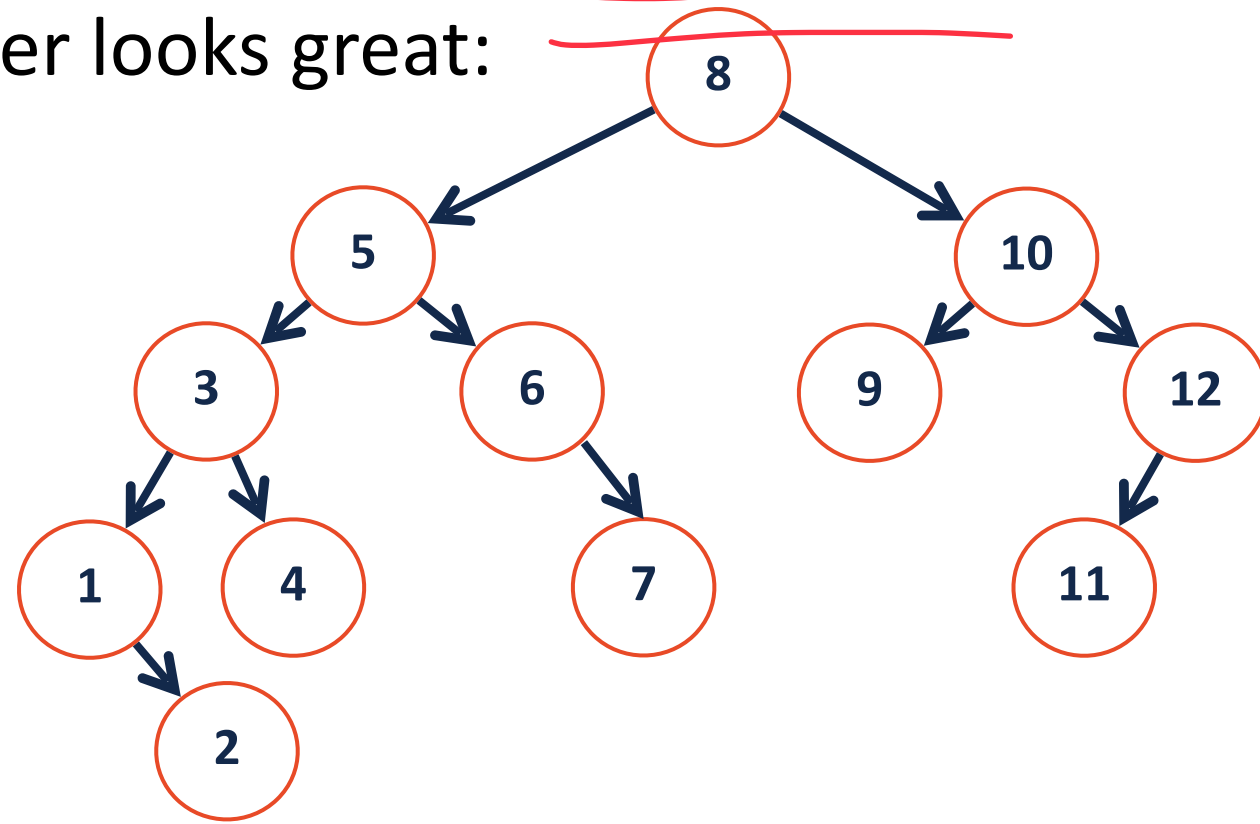
(Care of <https://gist.github.com/hellerbarde/2843375>)

Engineering vs Theory Efficiency

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

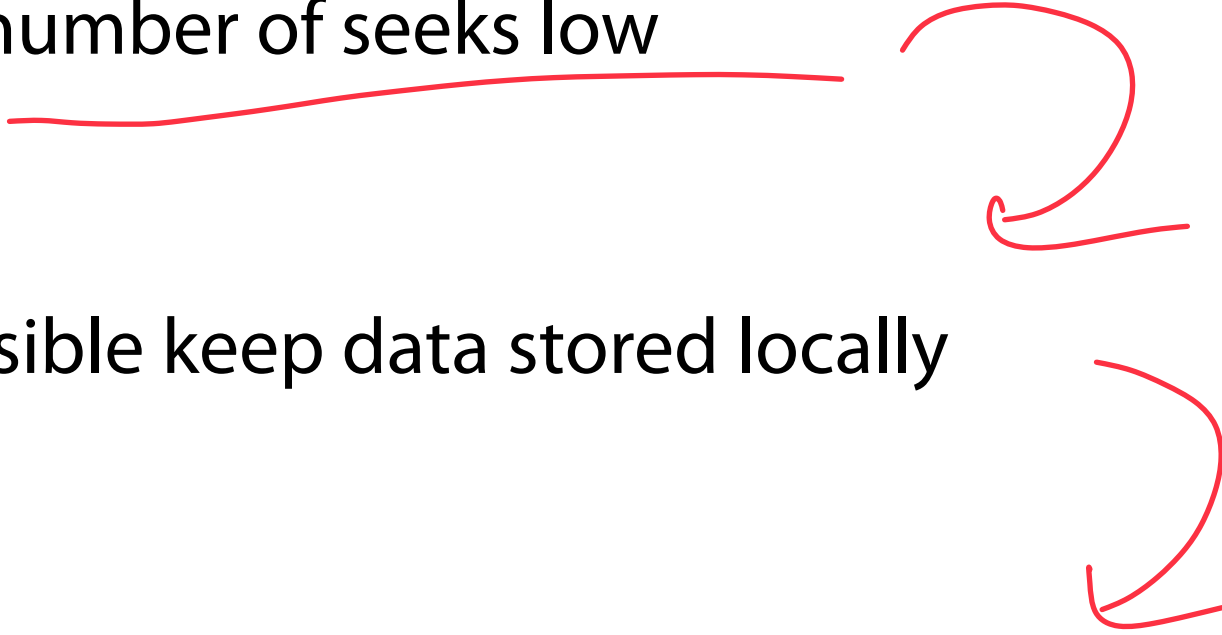
However, seeking data from the cloud may take 40ms+.

...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
 - 2) When possible keep data stored locally
 - 3) Make sure the data we look up is relevant!
- 

BTree Find

Find (~~21~~)
62

Base Cases:

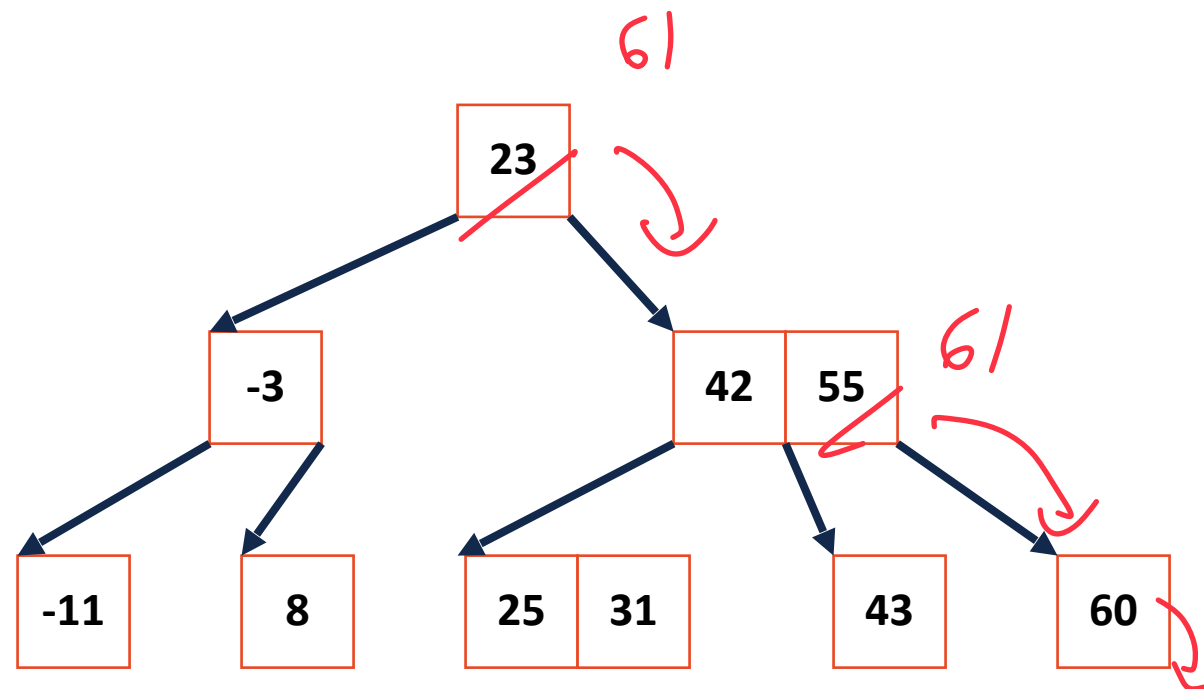
If root is empty, return

If leaf, do array find() and return

Recursive Step:

Array find() for match (then return) or first greater value (and recurse)

Recurse on appropriate child (index of first greater value or last child)



BTree Find

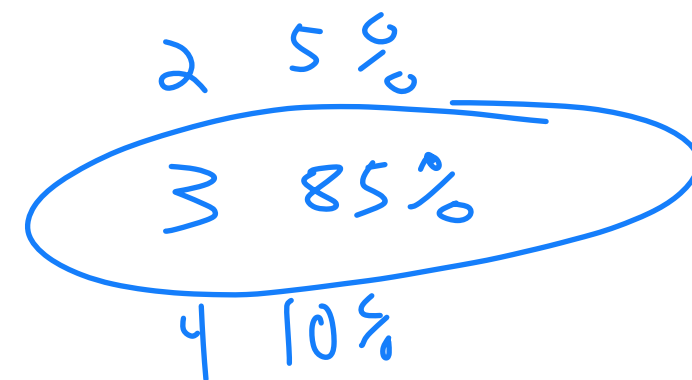
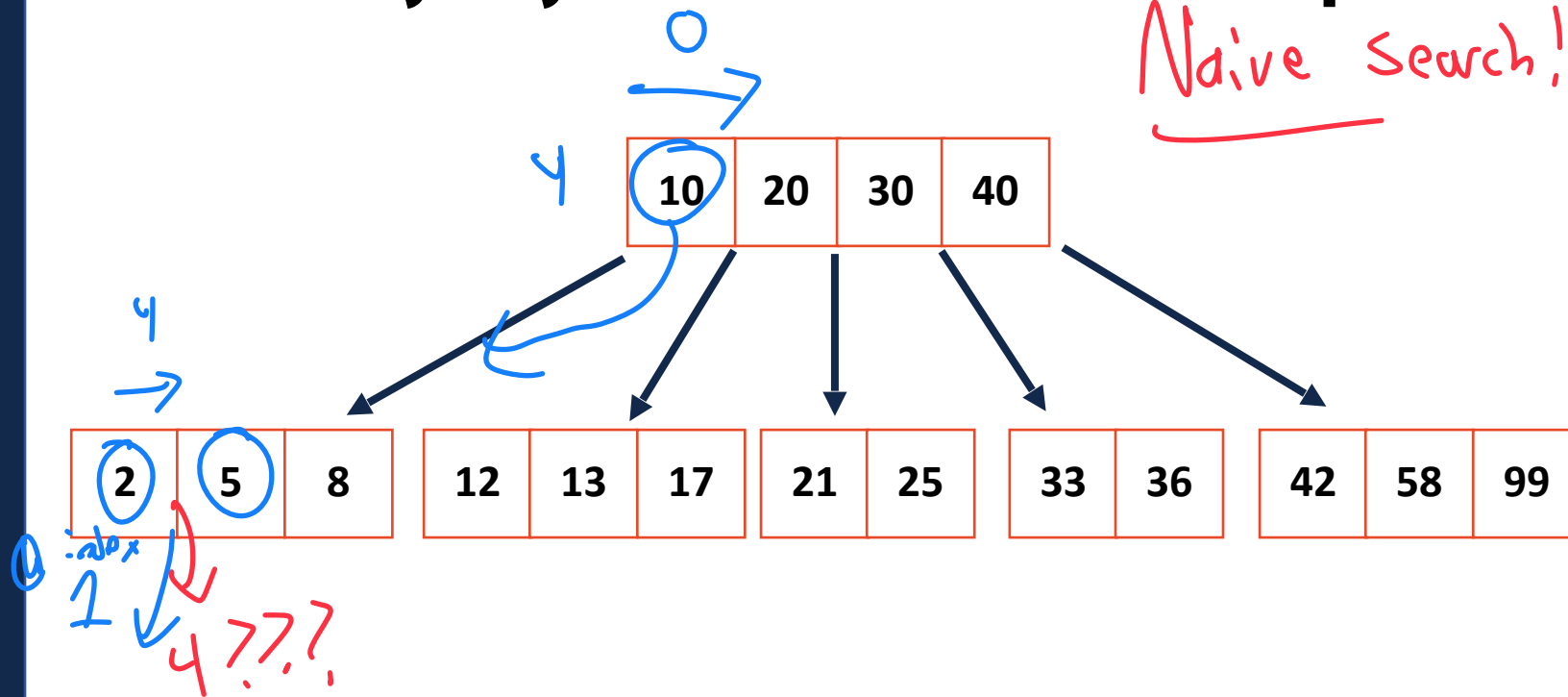
How many keys are looked at to complete this search?



Join Code: 225

Naive Search!

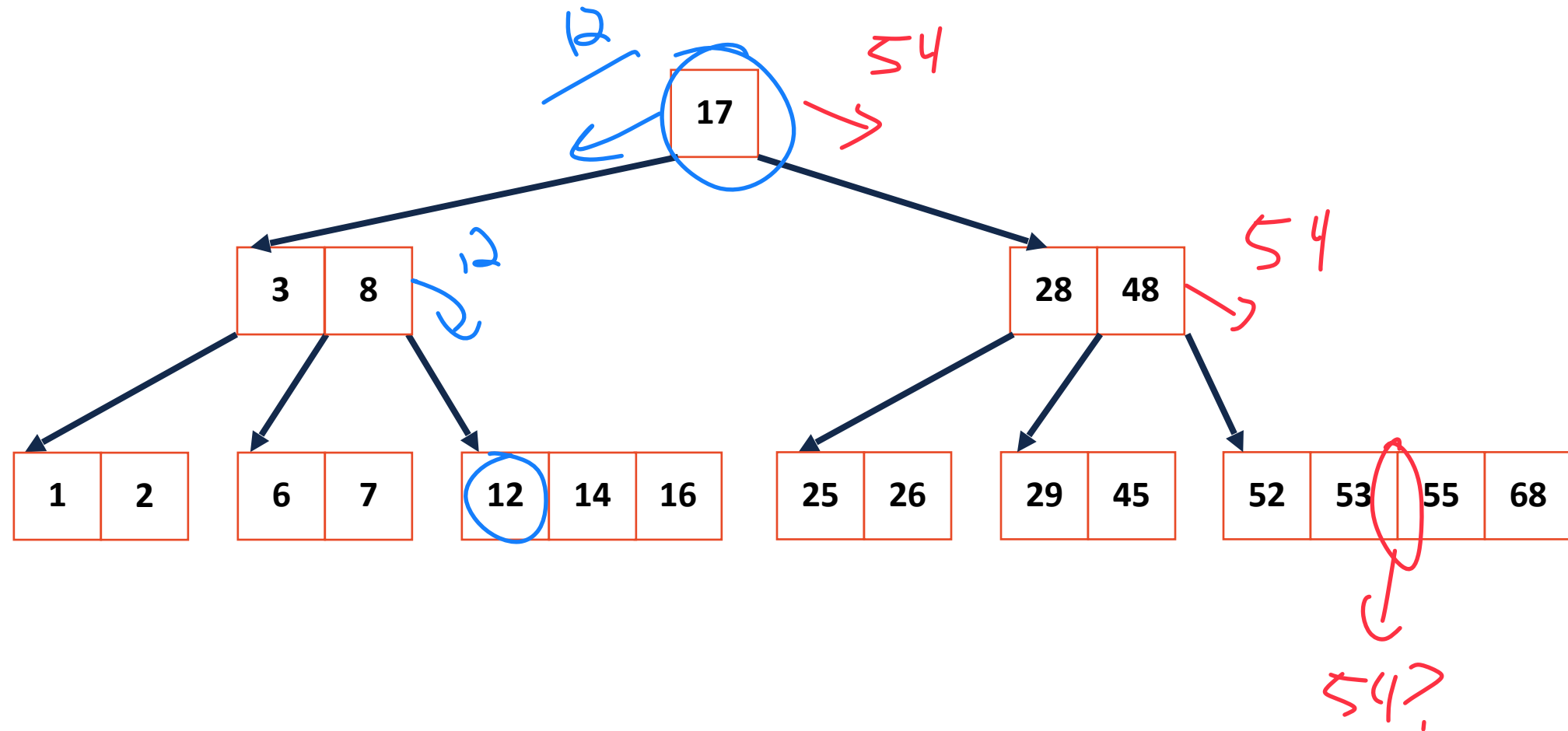
Find(4)



BTree

If $M=2$, we are a BST

What is the find path for 12? For 54?



BTree Insertion

Insert (56) , $M = 3$

1) Use BTree **find** to get correct array

(Node)

2) **Insert** item into array in sorted order

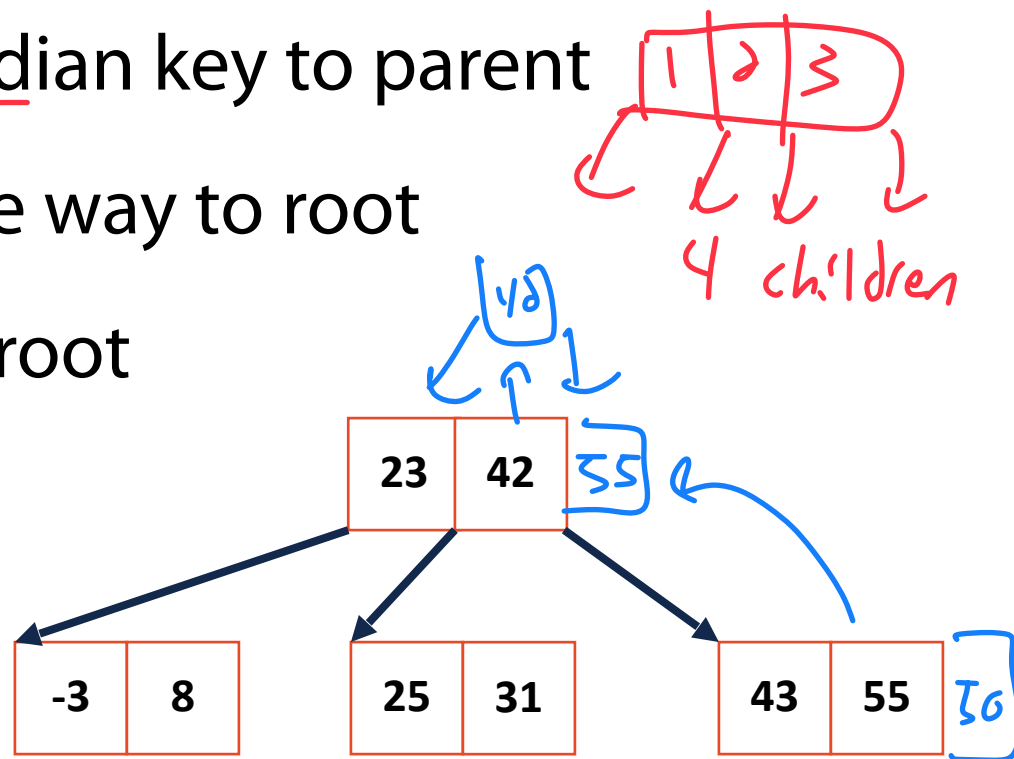
3) **Check** that array has at most $M - 1$ keys

At most M children
 $M - 1$ keys

4) If M keys, **split** node and **push up** median key to parent

Check (and push up as needed) all the way to root

If root splits, median becomes a new root



BTree Insertion

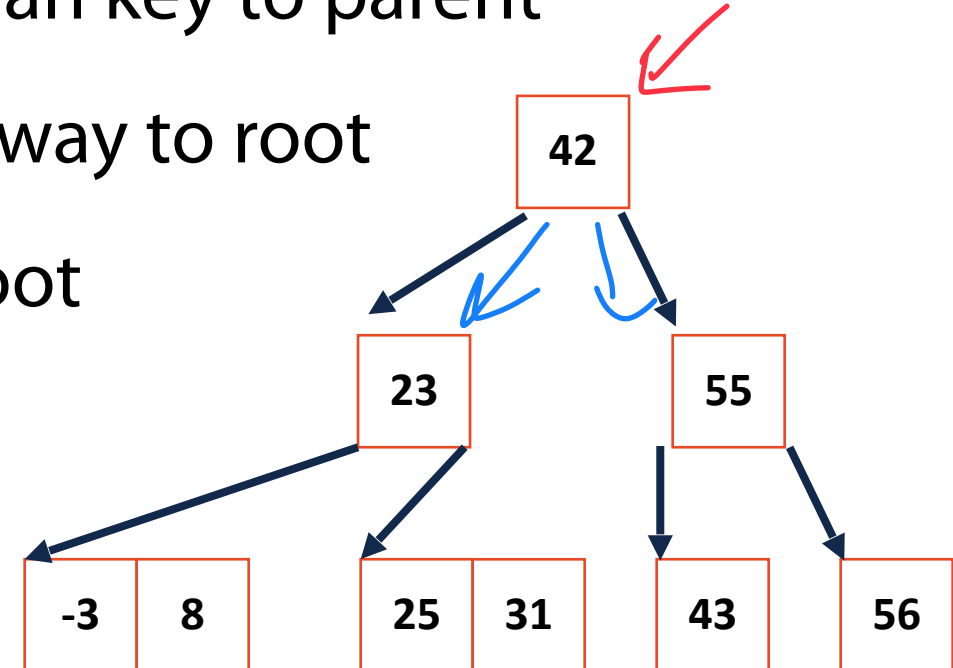
Insert(56), $M = 3$

- 1) Use BTree **find** to get correct array
- 2) **Insert** item into array in sorted order
- 3) **Check** that array has at most $M - 1$ keys
- 4) If M keys, **split** node and **push up** median key to parent

Check (and push up as needed) all the way to root

If root splits, median becomes a new root

All leaves on
same level



BTree Remove

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

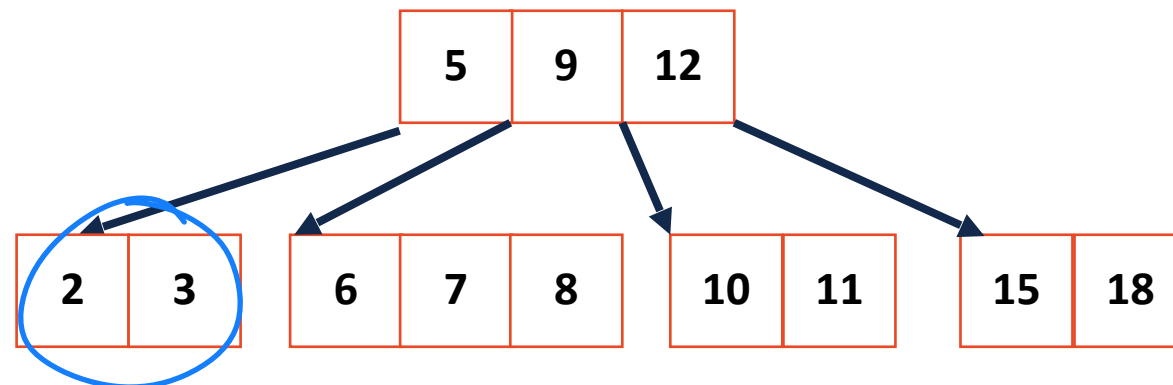
There's a POTD about it if interested

IMO, remove is boring

If ———, do this

If ———, do

...



↑ minimum size for split
More interesting concept!

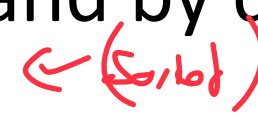
BTree Implementation

Any questions about BTree implementation details?



BTree Properties

A **BTree** 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? 

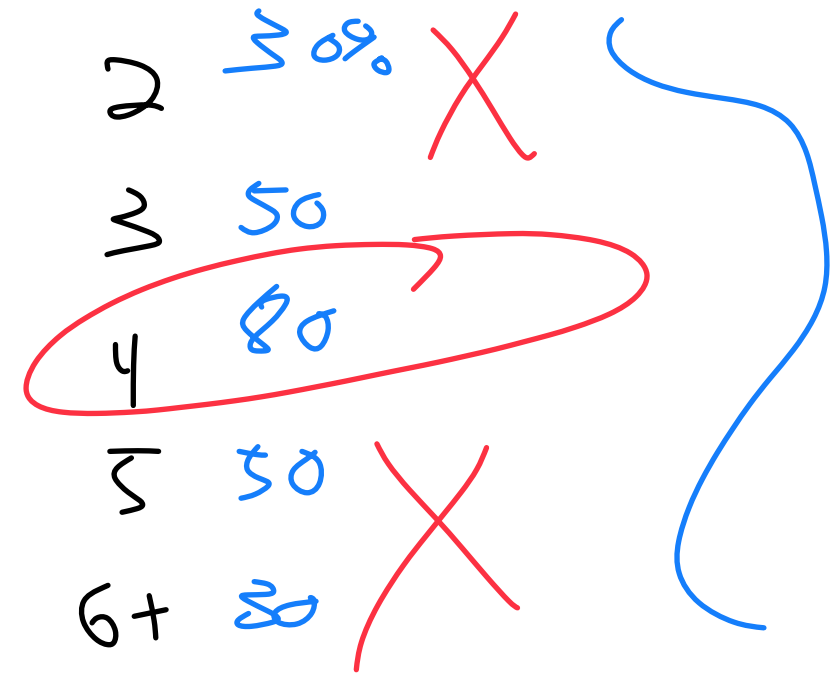
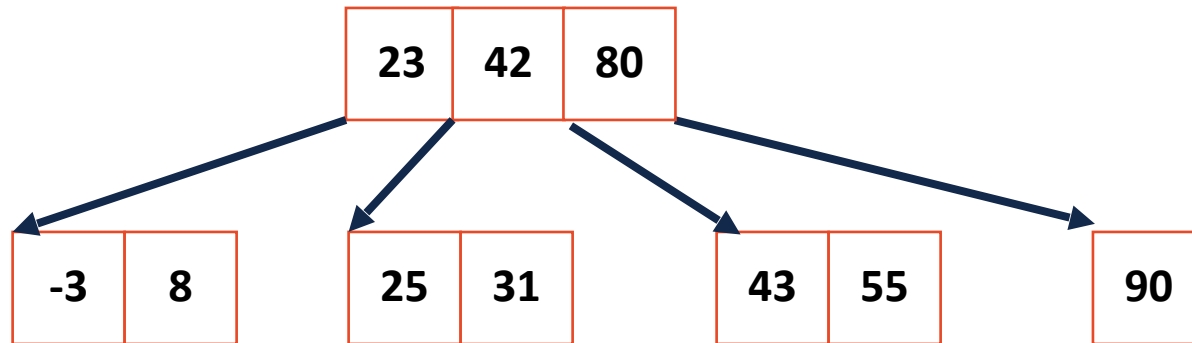
- 1) Keep the number of seeks low
- 2) When possible keep data stored locally
- 3) Make sure the data we look up is relevant!

Before we get there... a pretest!



Join Code: 225

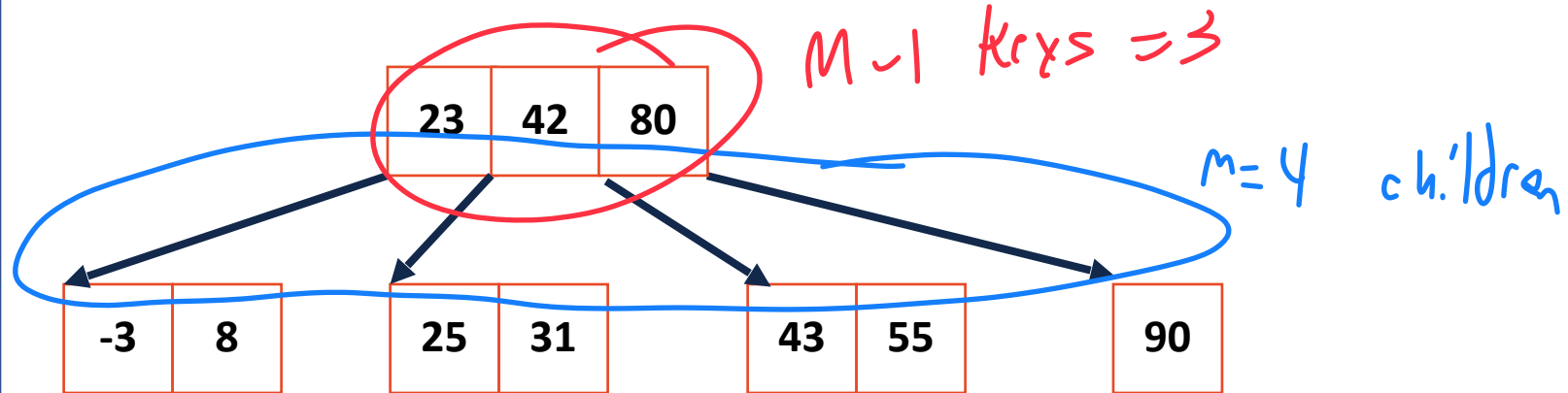
What values of M are allowed in the following tree?



Hint: M defines max but also... min # keys (children)

BTree Size Restrictions

M defines the maximum number of **children**...



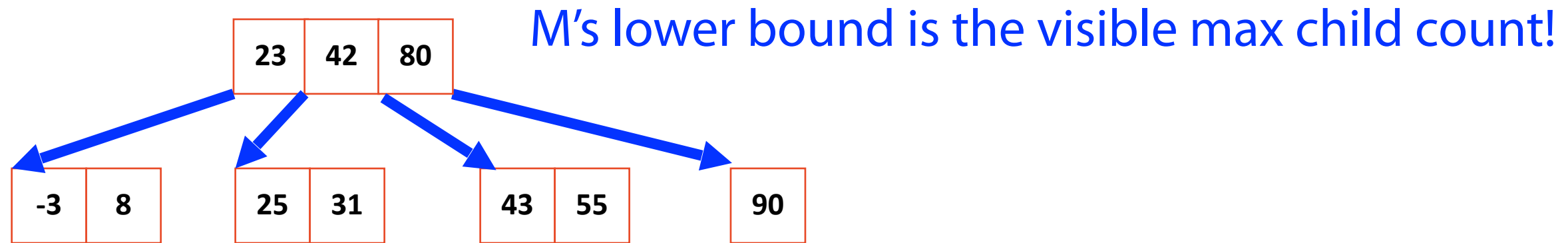
We are discussing
bounds on M

Look for node w/ greatest # keys

M is 1 or more

BTree Size Restrictions

M defines the maximum number of **children**...

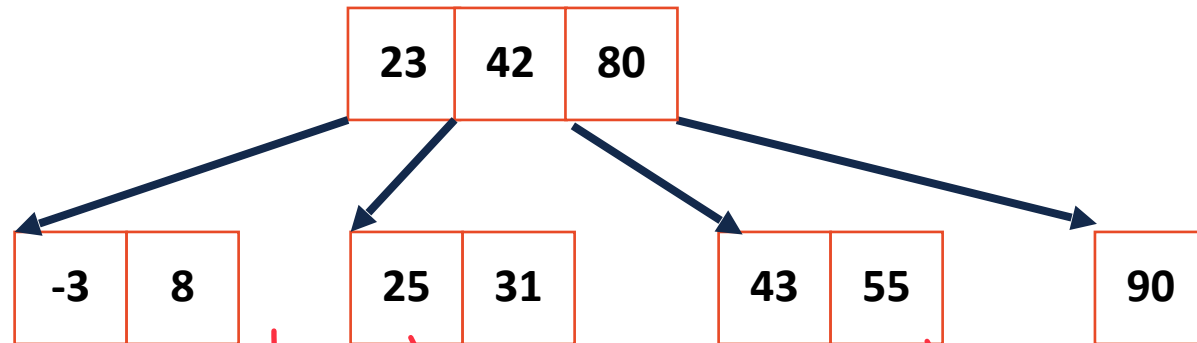


Easy to identify — find the single largest number of keys (plus 1)

BTree Size Restrictions

$M - 1$ keys OK
 → Not OK!

BTree insert splits one node with M keys by pushing up median...



Min # of keys in any non-leaf node

internal

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

children

$$2$$



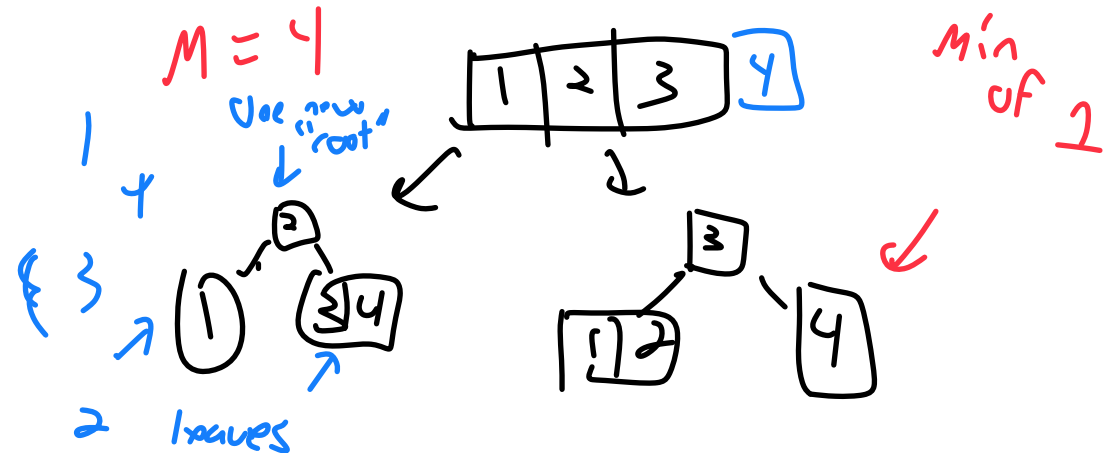
every non-leaf

OR

leaf →

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

keys



$M = 5$

1 | 2 | 3 | 4 | 5

$M = 6$

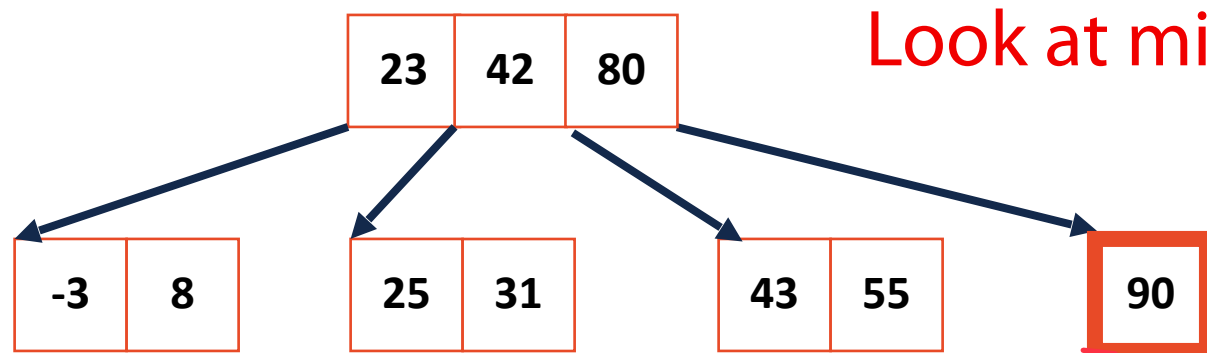
← min of 2 keys

1 | 2 | 3 | 4 | 5 | 6

1 | 2 | 3 | 4 | 5 | 6

BTree Size Restrictions

BTree insert splits one node with M keys by pushing up median...



Look at minimum size of non-root nodes!

If $M = 6$ (this can be 6)

This can be $M = 4$ only

Each resulting non-root node must have at least $\lceil \frac{m}{2} \rceil$ children!

Each resulting non-root node must have at least $\lceil \frac{m}{2} \rceil - 1$ keys!



BTree Properties

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

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

Root nodes can be a leaf or have $[2, m]$ children. $[1, m-1]$ keys

All non-root, internal nodes have $[\lceil \frac{m}{2} \rceil, m]$ children.

All leaves in the tree are at the same level.
 when we split we have got least this!
 All non root nodes have $[\lceil \frac{m}{2} \rceil - 1, m-1]$ keys

BTree Size Restriction Practice

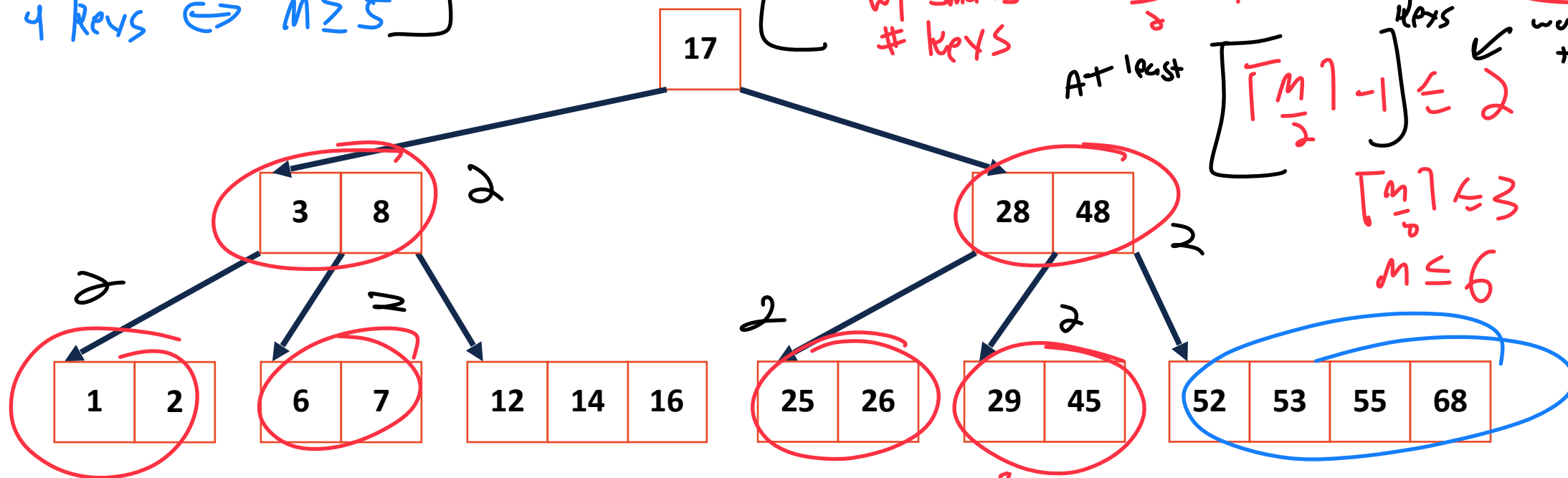


If I tell you this is a valid BTree, what is the value of m?

1) Find largest Node
4 keys $\Rightarrow m \geq 5$

2) The non root Node w/ smallest # keys $\lceil \frac{m}{2} \rceil - 1$ keys as min
At least $\lceil \frac{m}{2} \rceil - 1 \leq 2$ keys were just told can't be from

$\lceil \frac{m}{2} \rceil \leq 3$
 $m \leq 6$



5%

10%

20%

80%

50%

$m \geq$

2
~~2~~

3
~~3~~

4
~~4~~

5 6 ~~6~~

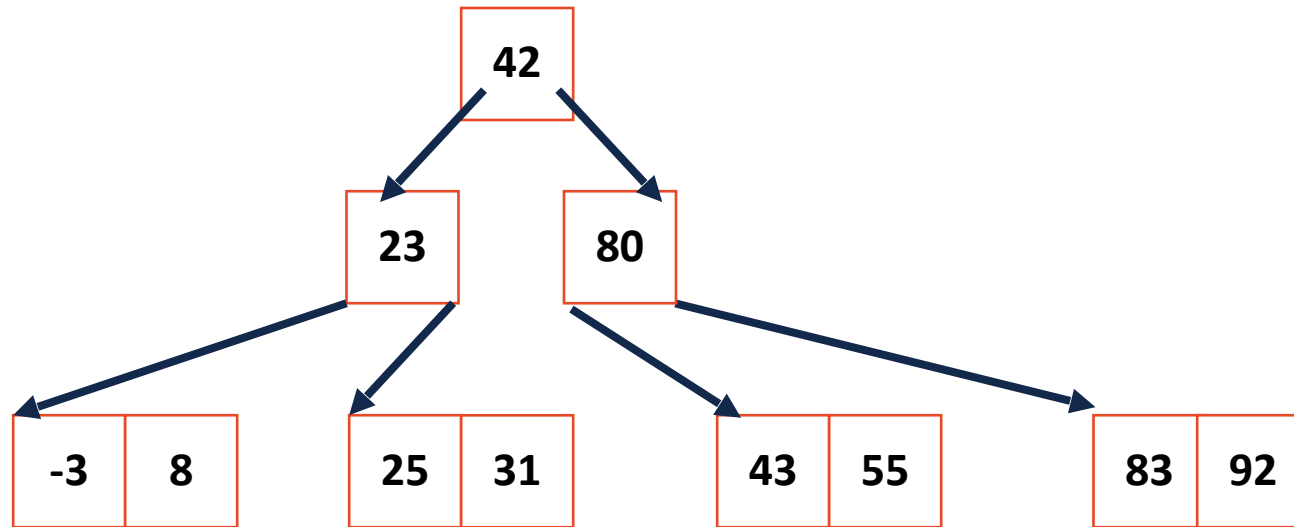
~~7~~
8

Lets review!



Join Code: 225

What values of M are allowed in the following tree?



Added after class

Largest: 2 keys

$$M \geq 3$$

Smallest non-root: 1 key

$$\left\lceil \frac{M}{2} \right\rceil - 1 \leq 1$$

$$\rightarrow \left\lceil \frac{M}{2} \right\rceil \leq 2$$

$$M \leq 4$$

$$M = 3 \text{ or } 4$$

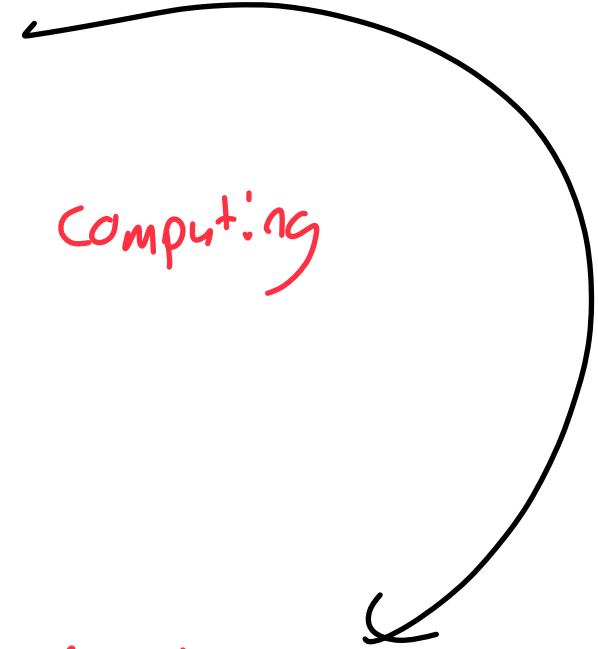
BTree of Order M (In Practice)

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

↳ 1500 byte Network packet

↳ 512
4096 block size of modern computing

□ Each node should have meaning



BTree of Order M

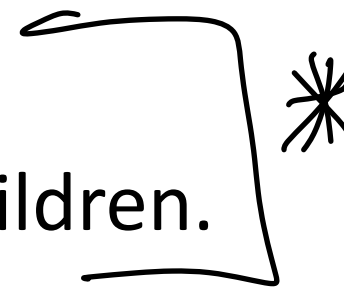
Questions about M?



BTree Properties

A **BTree** 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.
- Root nodes can be a leaf or have **[2, m]** children.
- All non-root, internal nodes have **[ceil(m/2), m]** children.



What can we prove about our runtime based on these properties?

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).

↑
This is Not n nodes!

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

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

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

↑
Lower bound on
children

Root:

1 } root has 2 children

Level 1:

2 } all other nodes have t children

Level 2:

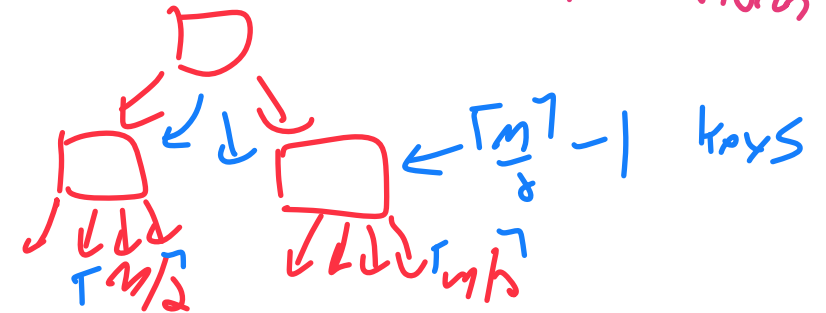
$2t$

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:

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

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

BTree Analysis

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

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

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

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

BTree Analysis

The **min total number of nodes:**

Diagram illustrating the formula for the minimum total number of nodes in a B-tree:

root

Sum of non root

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

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

$t - 1$ keys per node

The **min total number of keys:**

How many keys in root? 1

How many keys in internal node?

How many keys in leaf node?

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

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

$$t - 1$$

$$1 + 2(t^h - 1)$$

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**:

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

Root has how many keys? **1**

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

$$= 2(t^h - 1)$$

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

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

BTree Analysis

(for given height) $t = \lceil \frac{m}{2} \rceil$

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

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

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

$$\log_t(n+1) \geq \log_t(2t^h)$$

$$\boxed{\log_m(n) \geq h}$$

Proof is similar to AVL proof!
Hard to write relation for height given n
Easy to write min nodes given height
 $\hookrightarrow$ Add complexity (multiply node by $\frac{n}{\text{keys}}$)

Pink added after class :)

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:

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

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

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

2 is a constant

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

The BTree is still used heavily today!

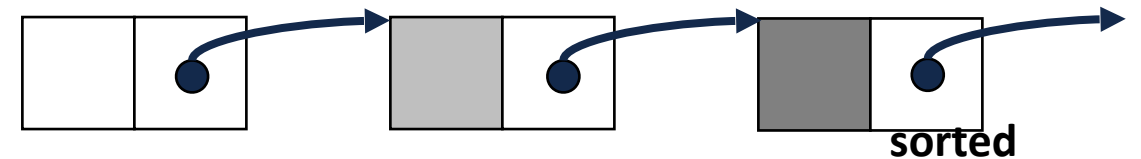
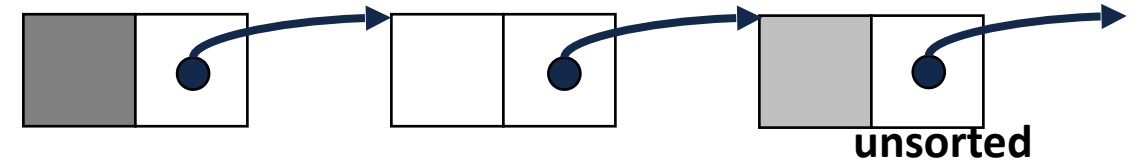
Improvements such as B+Tree and B*Tree exist far outside class scope

Thinking conceptually: Sorting a queue

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

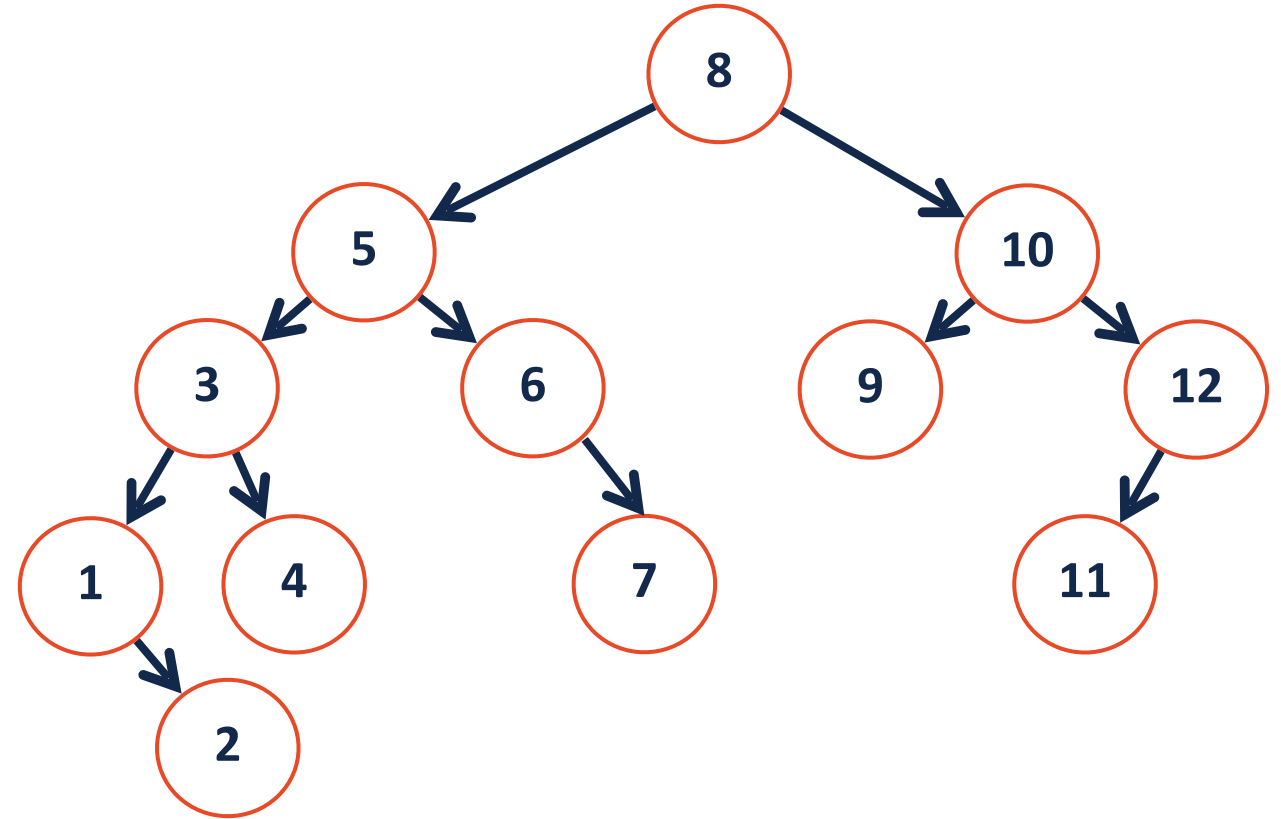
Priority Queue Implementation

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



Priority Queue Implementation

insert	removeMin



Thinking conceptually: A tree without pointers

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