Data Structures

BTree Analysis

CS 225

Brad Solomon

October 9, 2024

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Department of Computer Science

SIEBEL SCHOOL UNDERGRADUATE INFORMATIONAL SESSIONS

Research · Internships · Graduate School

LOOKING FOR A LITTLE TRICK TO UNLOCK YOUR ACADEMIC FUTURE? **WE'VE GOT THE TREATS FOR YOU!**

RESEARCH 101

DIVE INTO THE MYSTERIES OF RESEARCH OPPORTUNITIES AT SIEBEL SCHOOL AND DISCOVER HOW YOU CAN GET INVOLVED! PRESENTED BY PROFESSOR BRAD SOLOMON W/ STUDENT Q & A PANEL.

October 10th, 1-2:30 PM, Siebel 2405

Sponsored by the Siebel School Undergraduate Programs Office

More questions? Contact

undergrad@siebelschool.illinois.edu

Learning Objectives

Discuss the importance of M in a B Tree

Analyze the performance of the B 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.

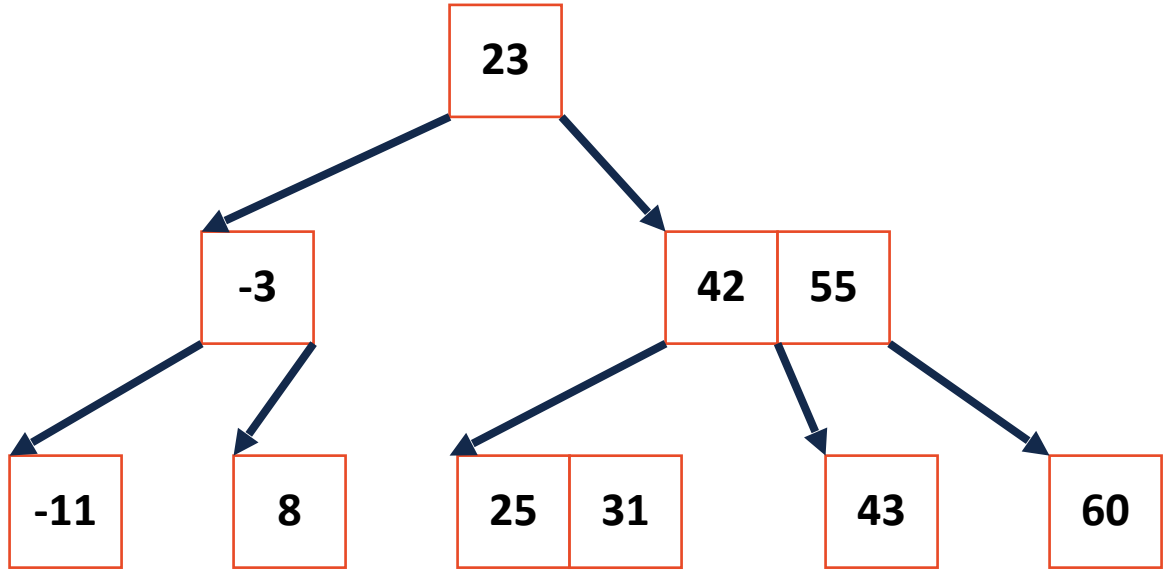
BTree Find

`Find(7)`

Base Case:

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 the appropriate BTreeNode, insert is array insert

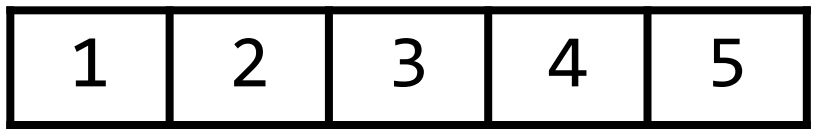
```
Insert(1)
Insert(2)
Insert(3)
Insert(4)
Insert(5)
Insert(6)
Insert(7)
Insert(8)
```

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)
```

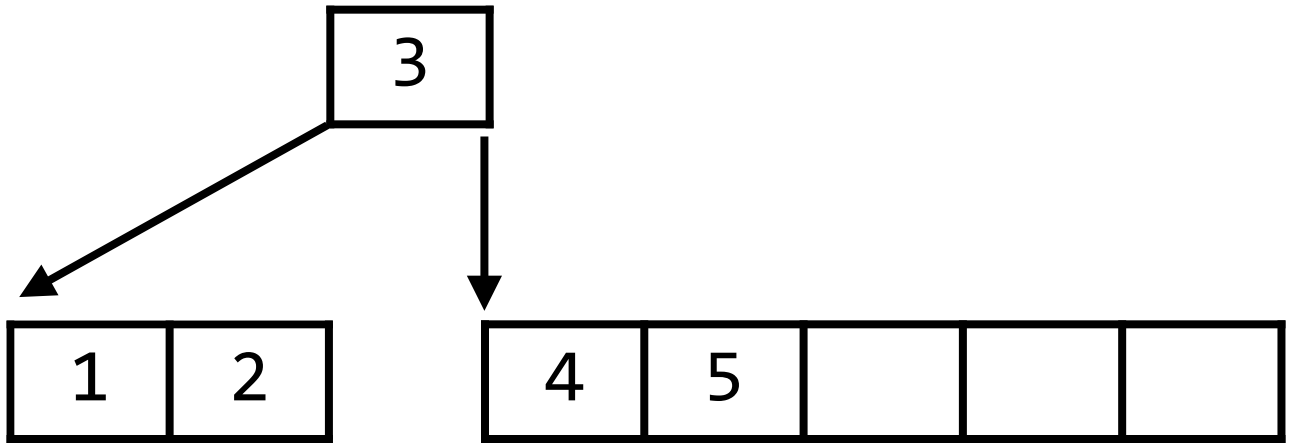


BTree Insertion

M = 5

"Given appropriate BTreeNode" == Find()

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

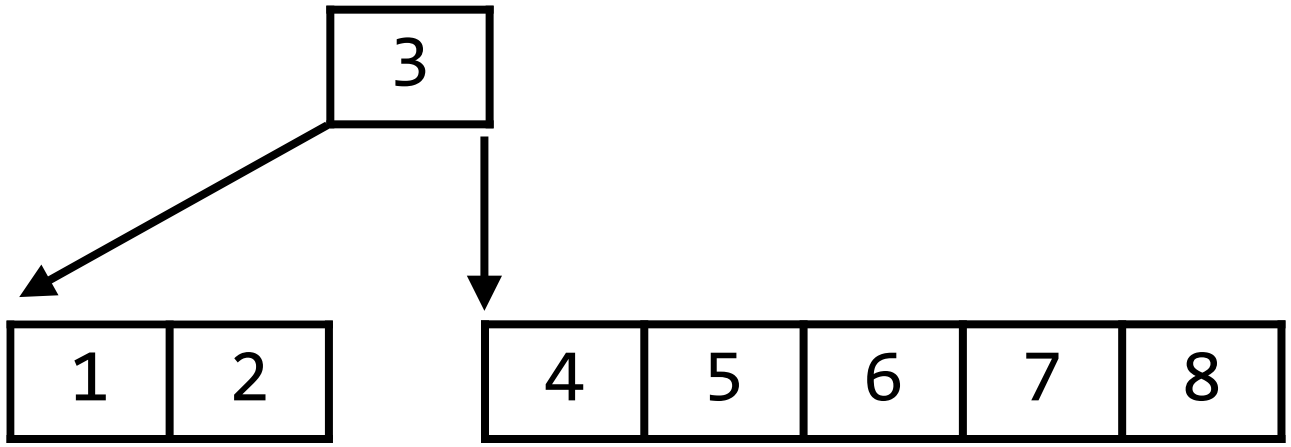


BTree Insertion

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)
```

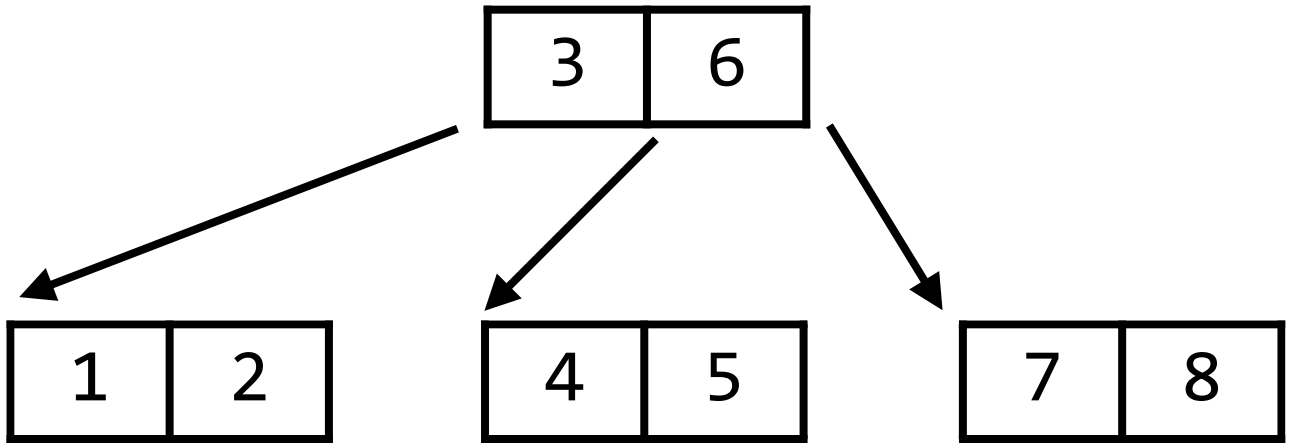


BTree Insertion

M = 5

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

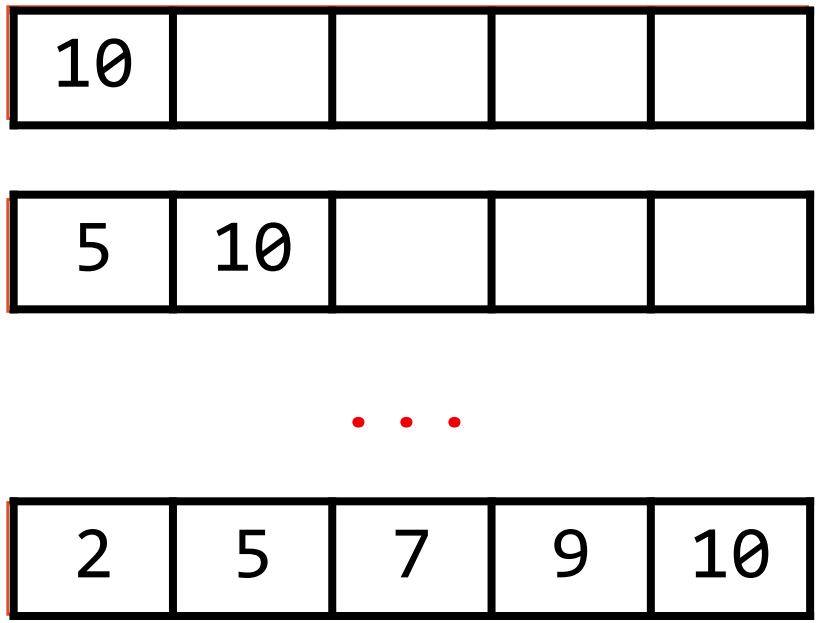
```
Insert(1)
Insert(2)
Insert(3)
Insert(4)
Insert(5)
Insert(6)
Insert(7)
Insert(8)
```



BTree Insertion

M = 5

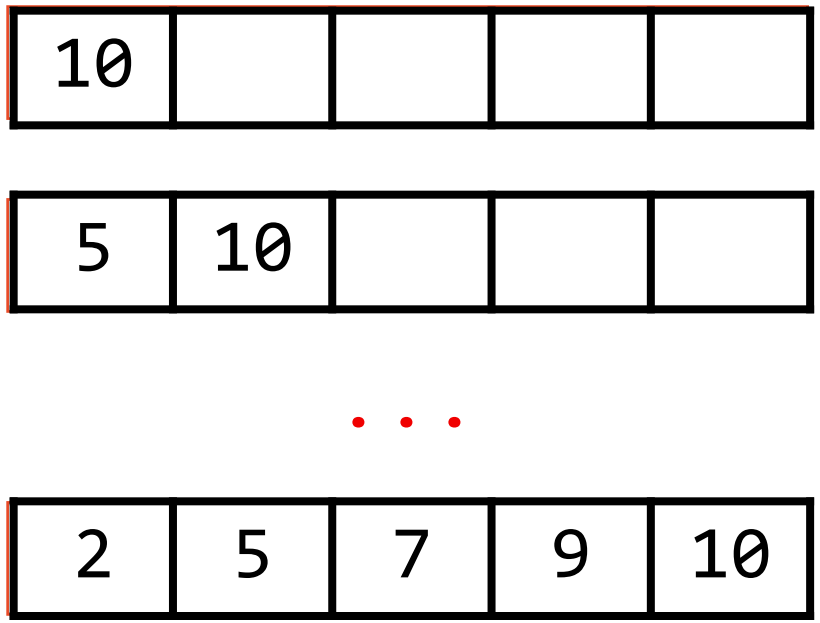
Problem 3: I need to find median value AFTER inserting the **M**th value



BTree Insertion

M = 5

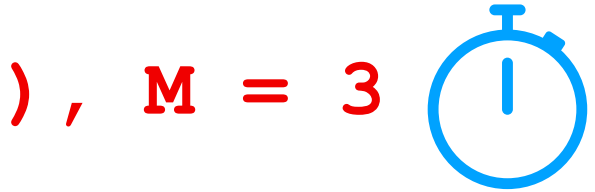
Problem 3: I need to find median value AFTER inserting the **M**th value



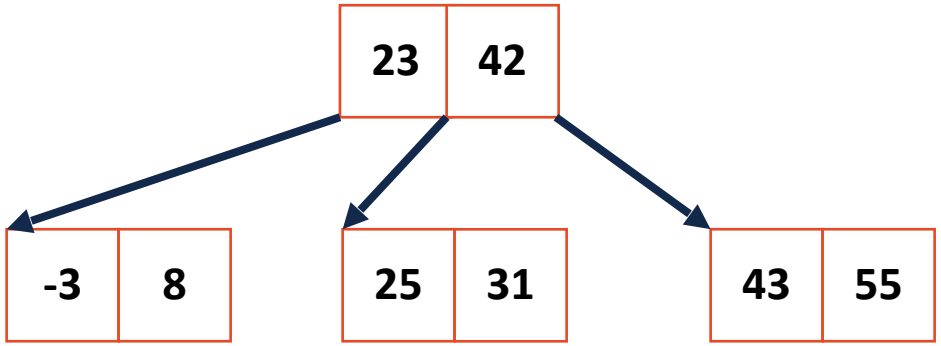
Non-Optimal Solution: Pre-allocate **M** size arrays for every node!

BTree Recursive Insert

`Insert(56), M = 3`

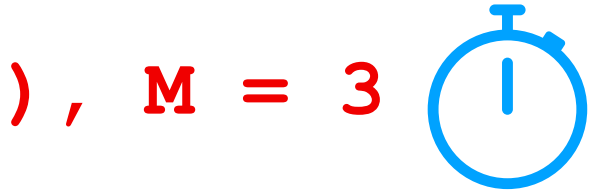


Insert always starts at a leaf but can propagate up repeatedly.

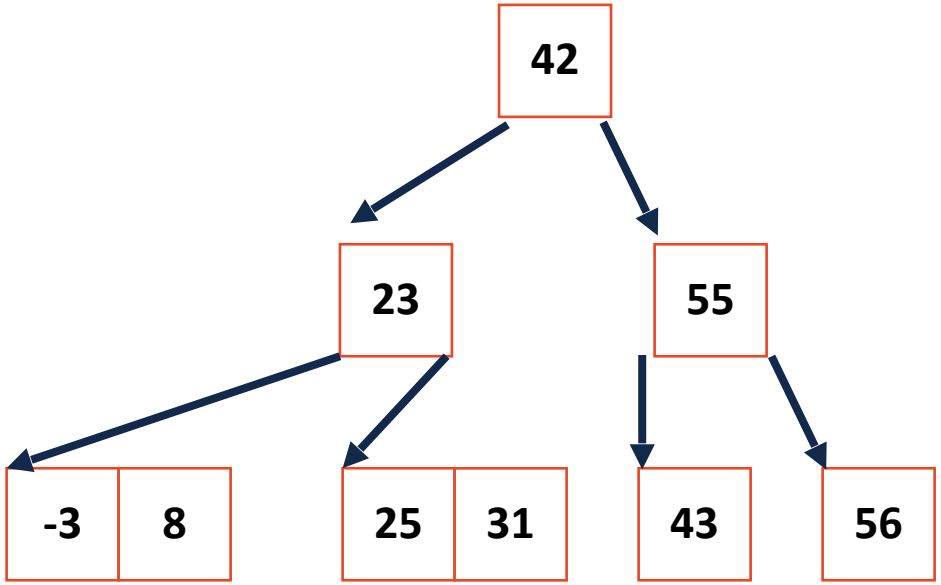


BTree Recursive Insert

`Insert(56), M = 3`



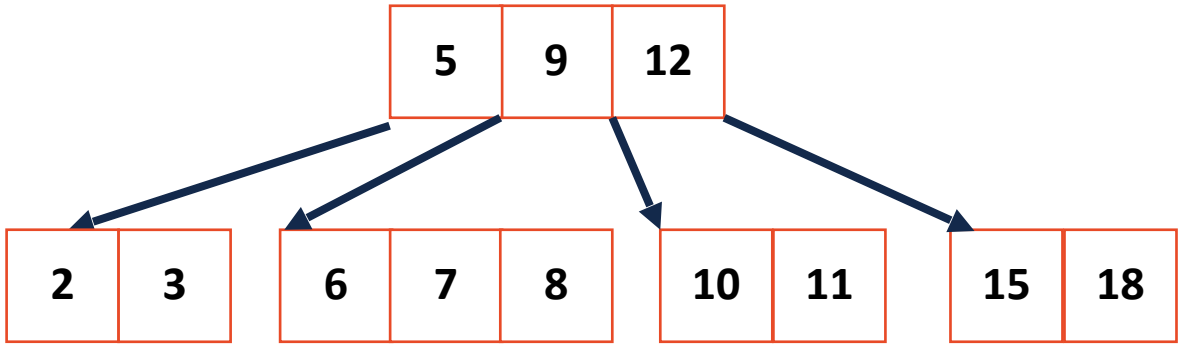
Insert always starts at a leaf but can propagate up repeatedly.



BTree Remove

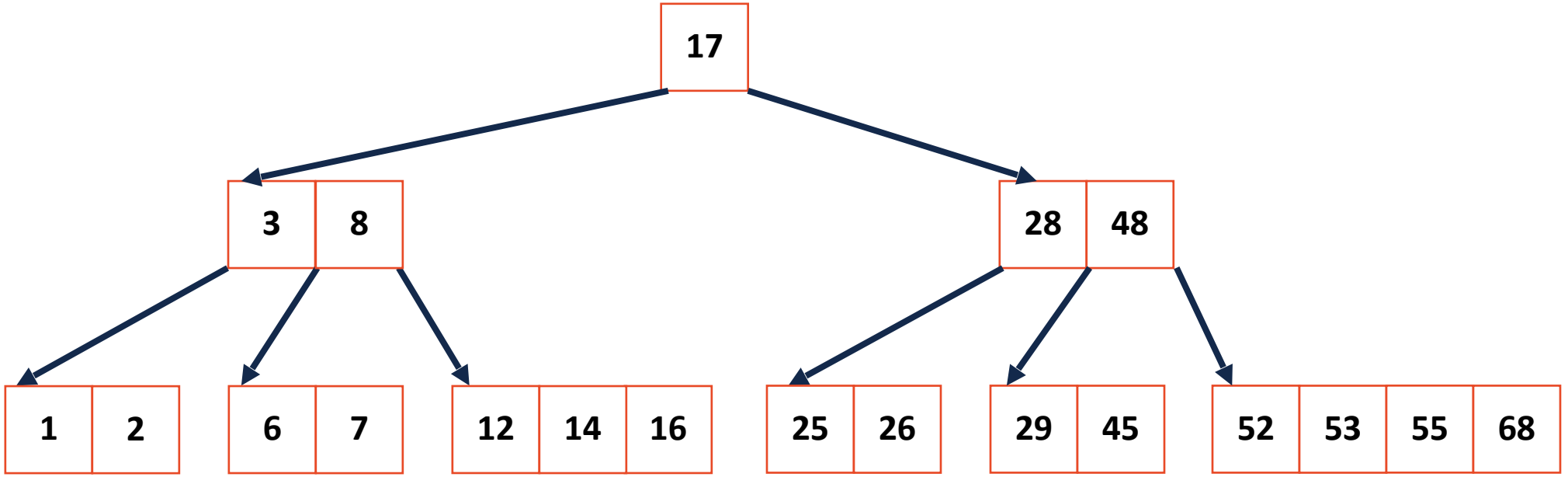
BTree removal is complicated! **It won't be part of the lab.**

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



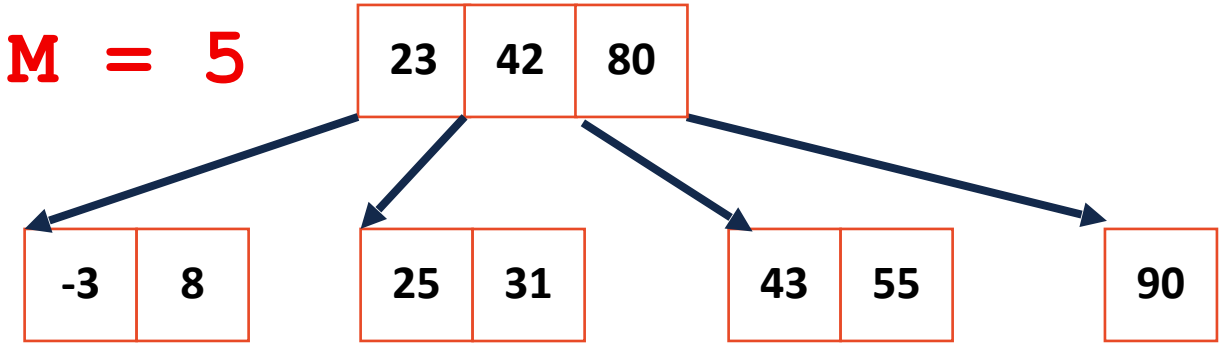
BTree *of Order M*

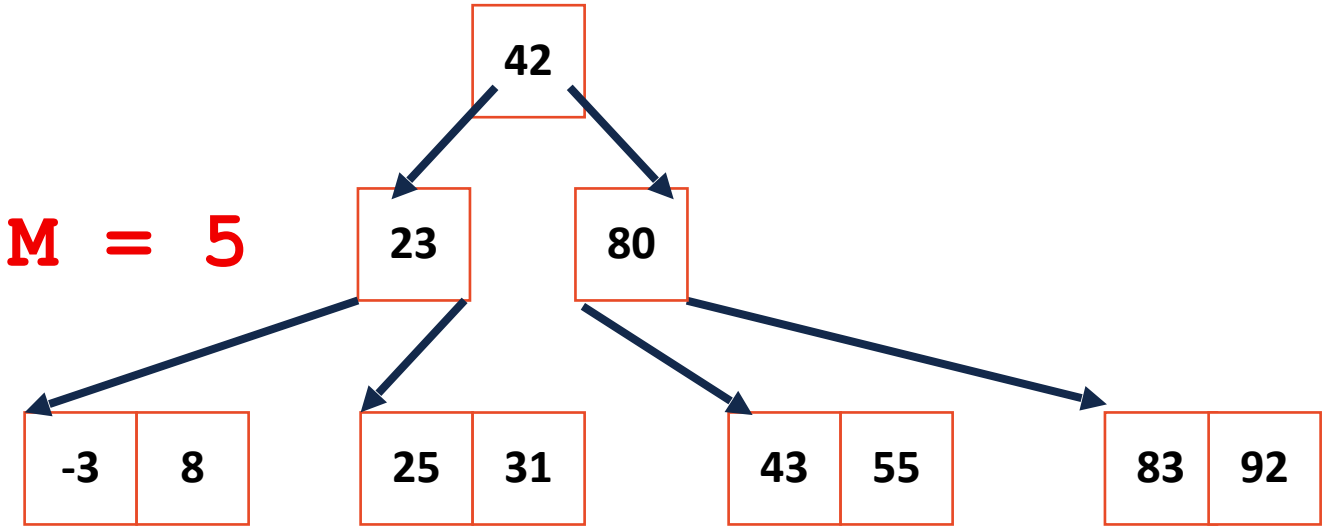
If I tell you this is a valid BTree, what is the **lower bound** of **M**?



BTree Size Restrictions

We have max on nodes, but do we have min? Are these trees valid?





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**

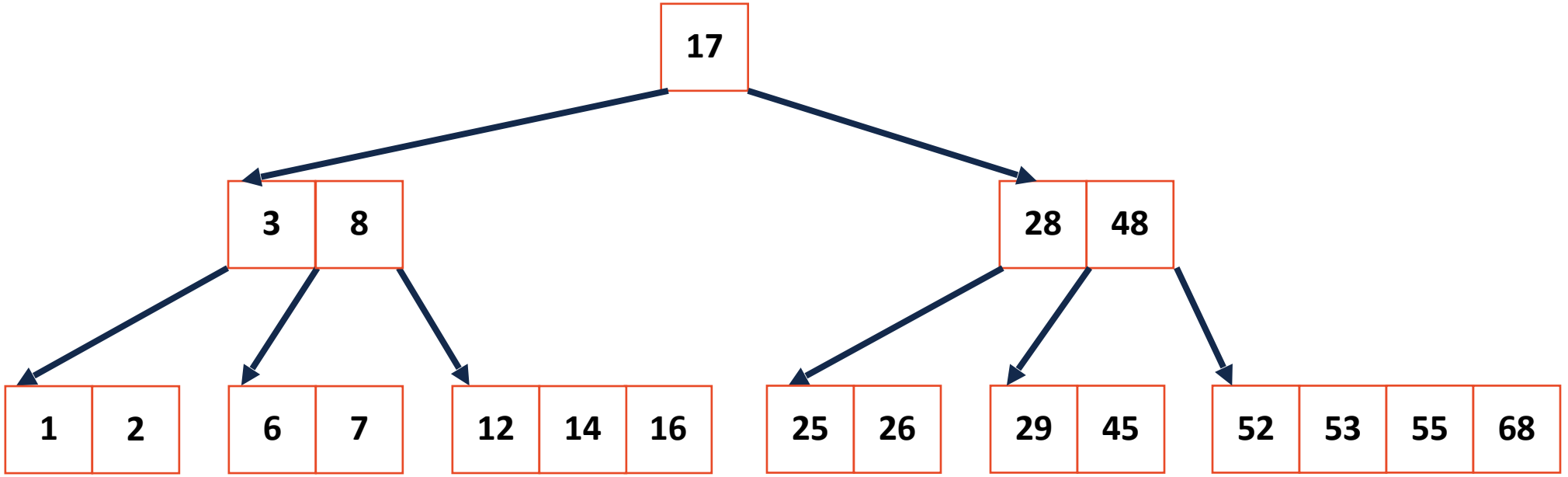
Root nodes can be a leaf or have _____ children.

All non-root, internal nodes have _____ children.

All leaves in the tree are at the same level.

BTree

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



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

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

Root:

Level 1:

Level 2:

Level 3:

Level h:

BTree Analysis

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

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

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

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

BTree Analysis

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

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

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 $t - 1$

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

$$= 2t^h - 1$$

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:

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:

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

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

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

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

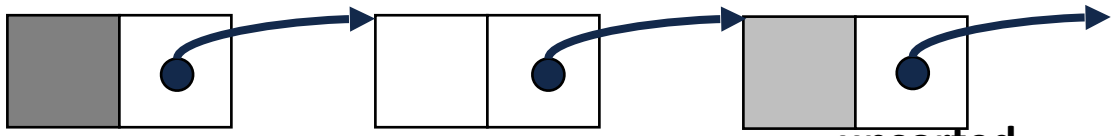
Thinking conceptually: Sorting a queue

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

Priority Queue Implementation

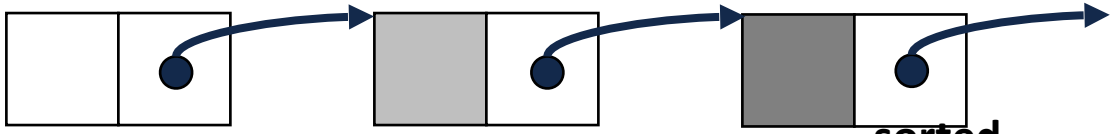
insert	removeMin
O(n)	O(n)
O(1)	O(n)
O(n)	O(1)
O(n)	O(1)

unsorted



unsorted

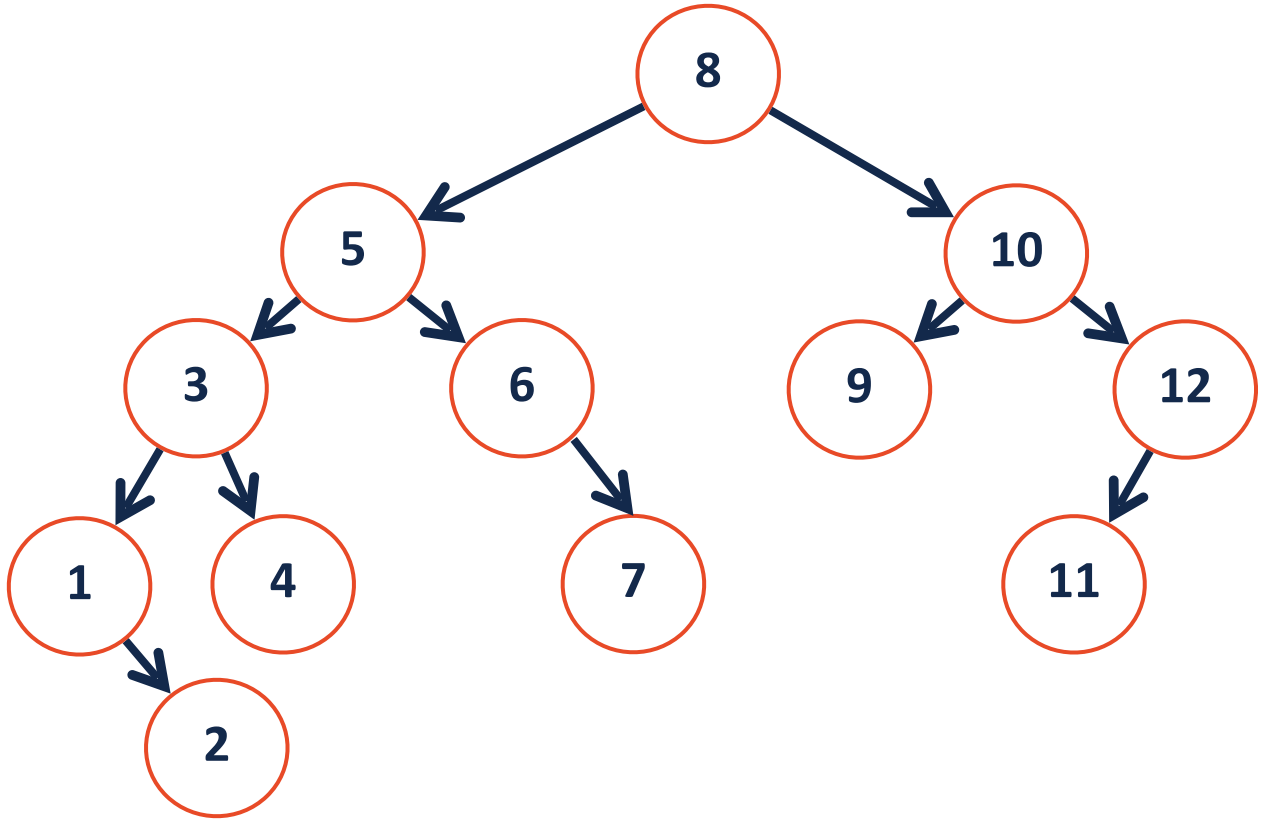
sorted



sorted

Priority Queue Implementation

insert	removeMin



Thinking conceptually: A tree without pointers

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