Data Structures
BTree

CS 225 October 4, 2023

Brad Solomon & G Carl Evans

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Department of Computer Science

Learning Objectives

Discuss alternatives to AVL Trees (and BSTs)

Implement the BTree

Considering hardware limitations

Can we always fit our data in main memory?

Where else can we keep our data?

Does this match our assumption that all memory lookups are O(1)?

BTree Design Motivations

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

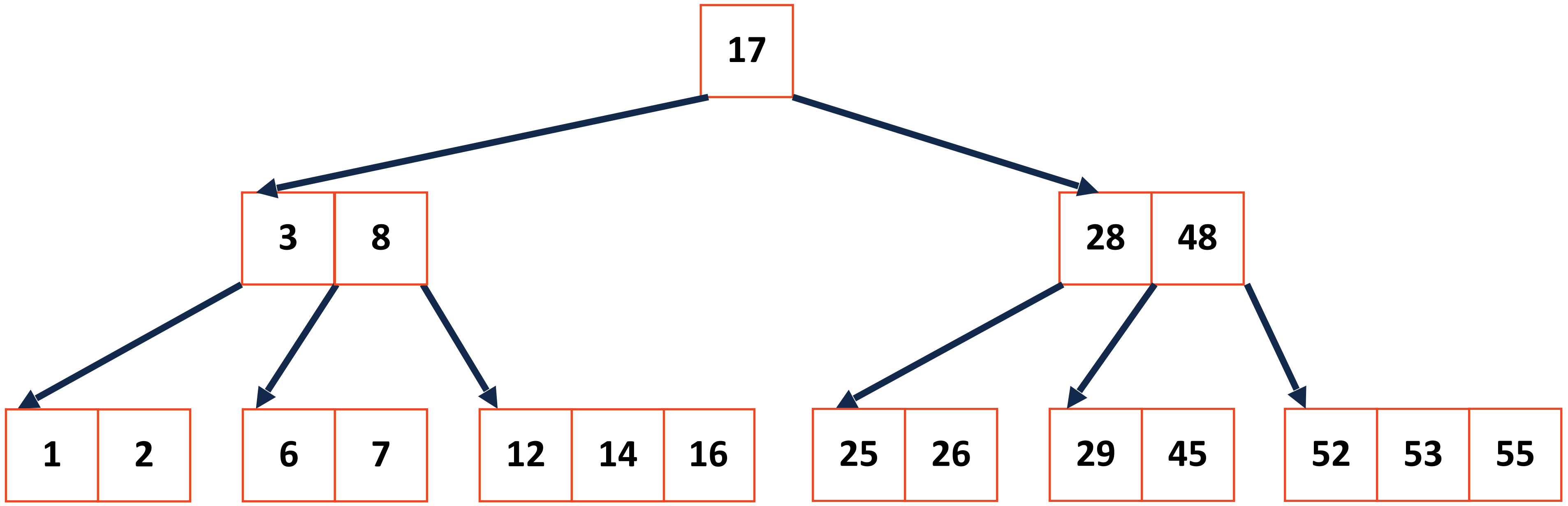
1. “Pack a node with more data” — **store many keys in each node**

BTree

A BTree (of order m) is a m-ary tree

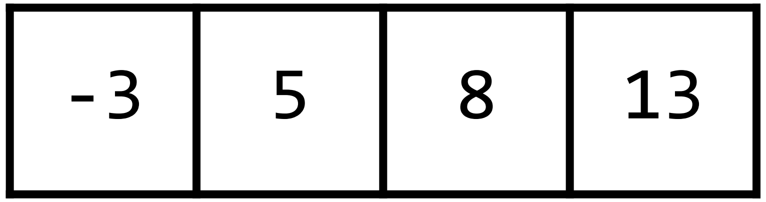
Nodes contain up to **m-1** keys and have **|keys|+1** children

All leaves in a BTree are on the same level



BTree Node (of order m)

$M \geq 5$



BTree Node (of order m)

What value of **m** should we be using?

BTree Insertion

M = 5

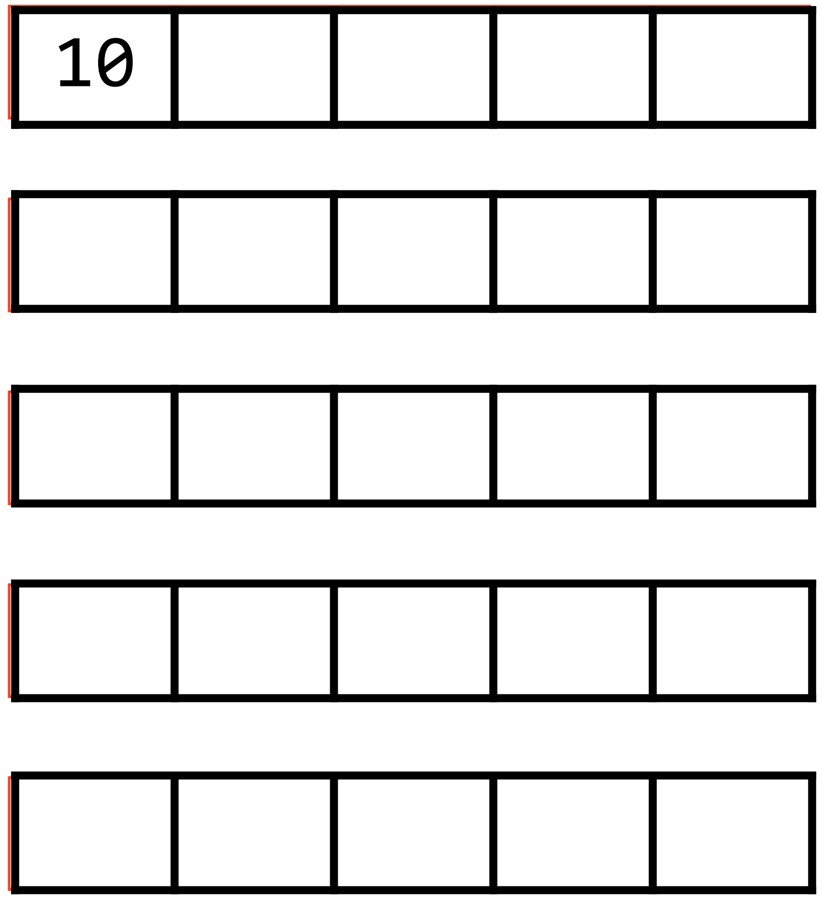
All keys within a BTree are ordered

Insert(10)

BTree Insertion

M = 5

All keys within a BTree are ordered



Insert(10)

Insert(5)

Insert(7)

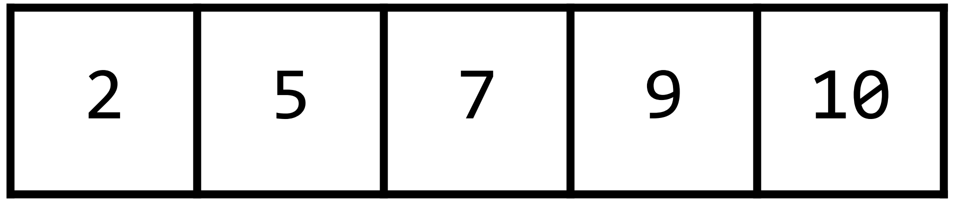
Insert(9)

Insert(2)

BTree Insertion

M = 5

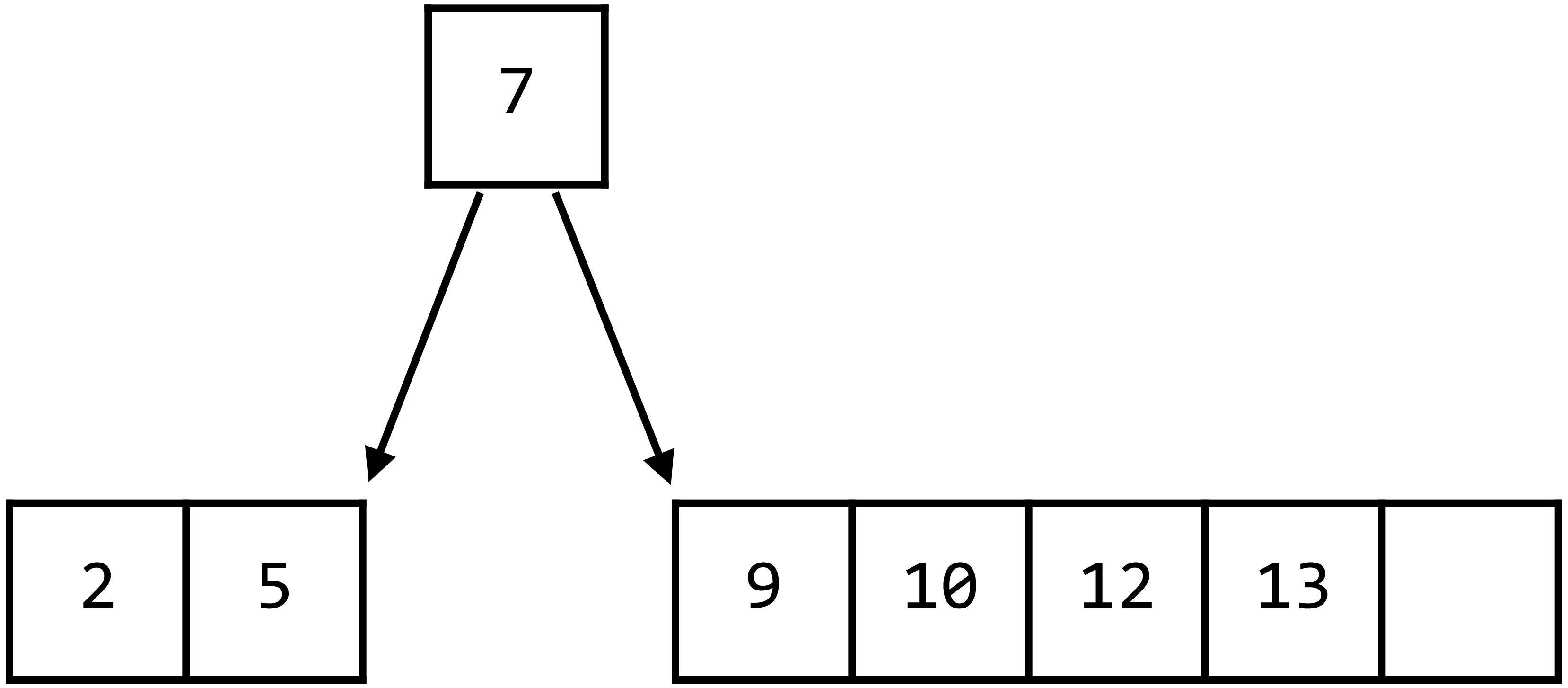
When a BTree node reaches **m** keys:



BTree Insertion

M = 5

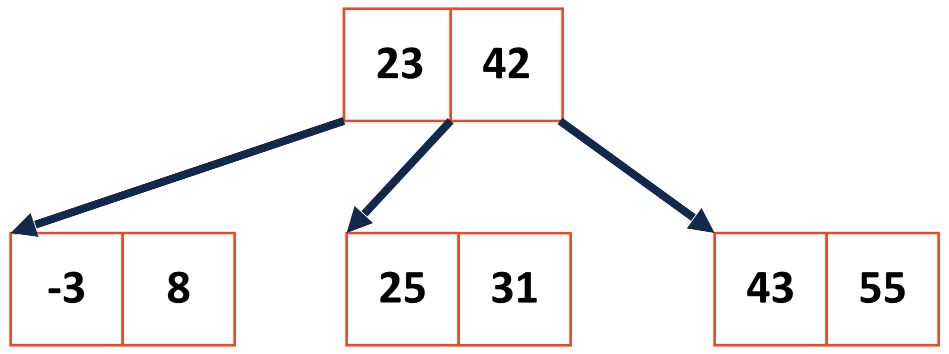
When a BTree node reaches **m** keys, **split and make a new parent.**



BTree Recursive Insert

Insert(56), M = 3

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



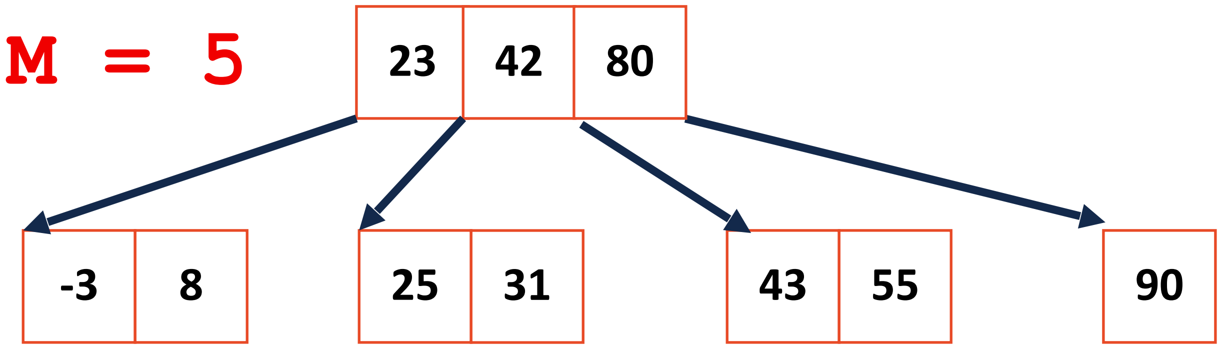
BTree Visualization/Tool

https://www.cs.usfca.edu/~galles/visualization/BTree.html

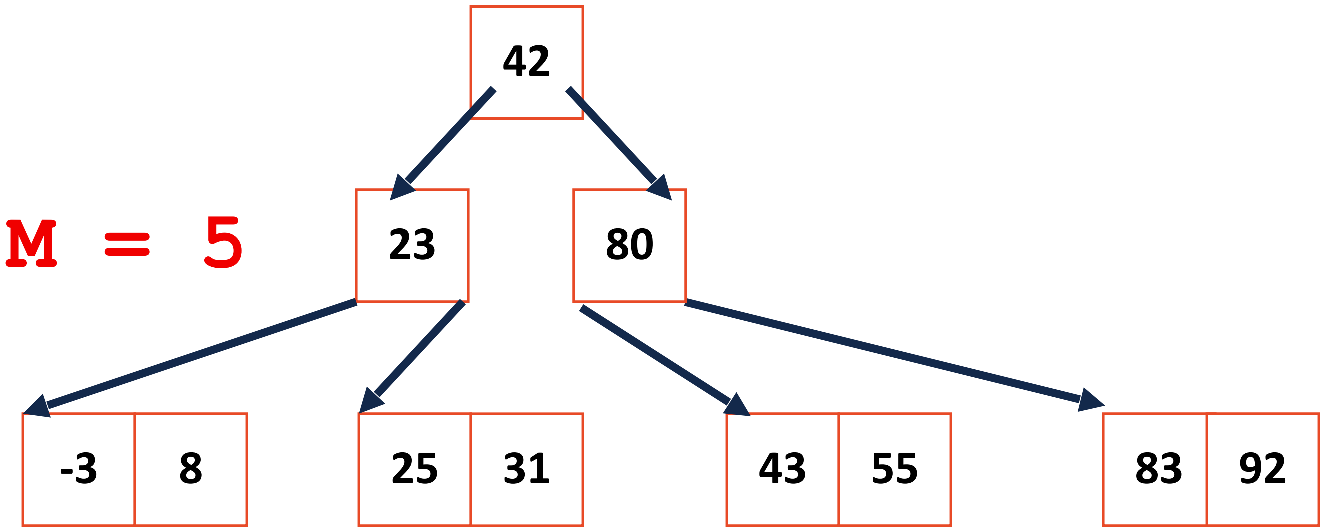
BTree Size Restrictions

By definition we have max, but do we have min? Are these trees valid?

M = 5



M = 5



BTree Properties

A **BTrees** 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**

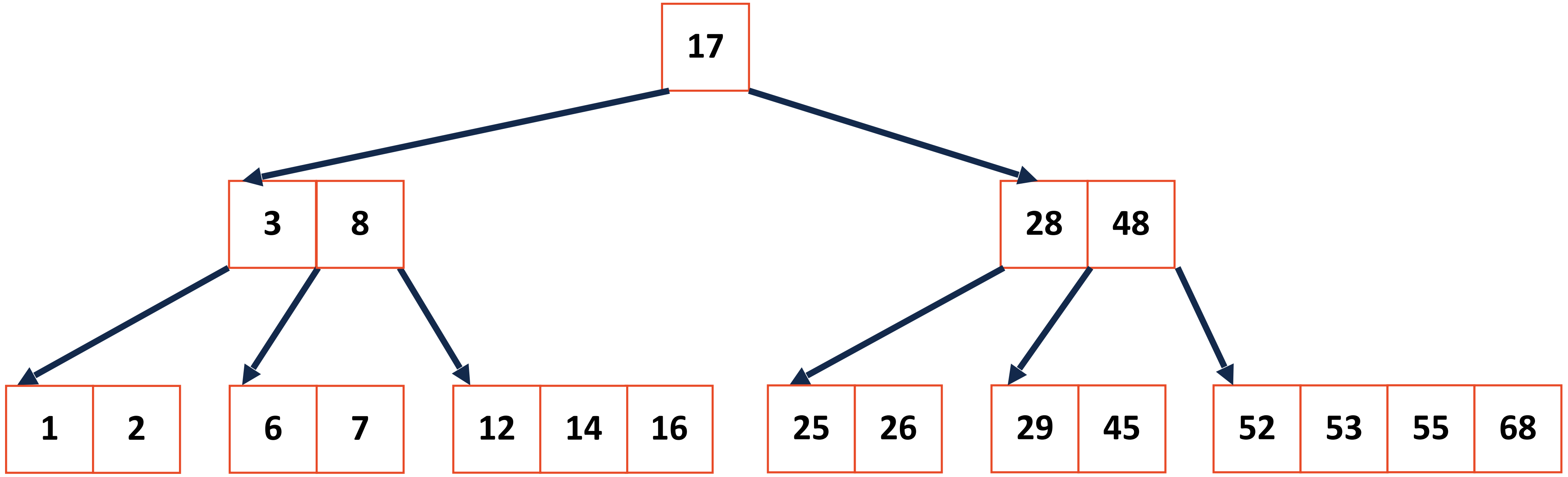
Root nodes can be a leaf or have ______________ children.

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 value of m?



BTree ADT

Constructor

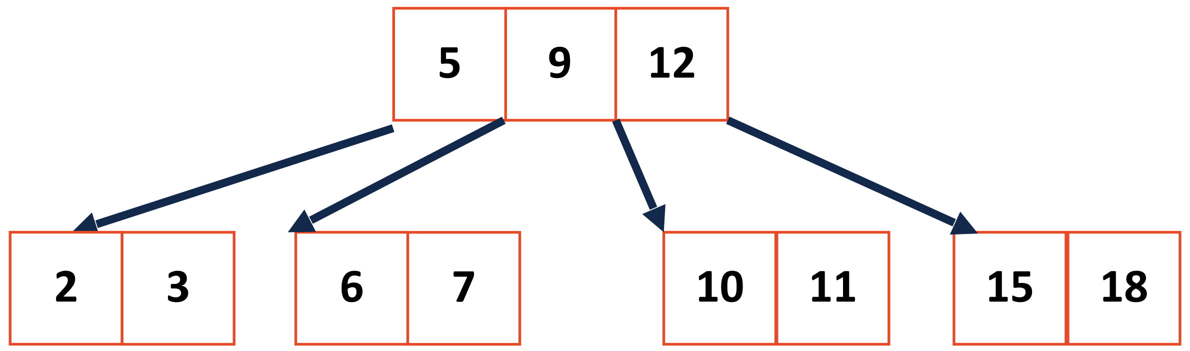
Insert

Find

Delete

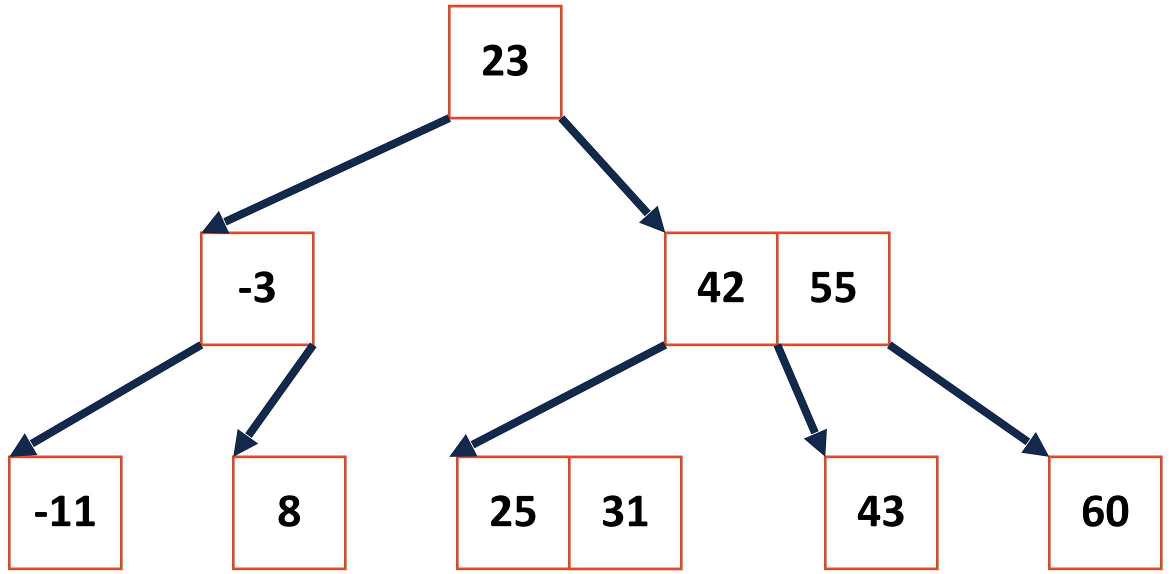
BTree Find

Find(7)



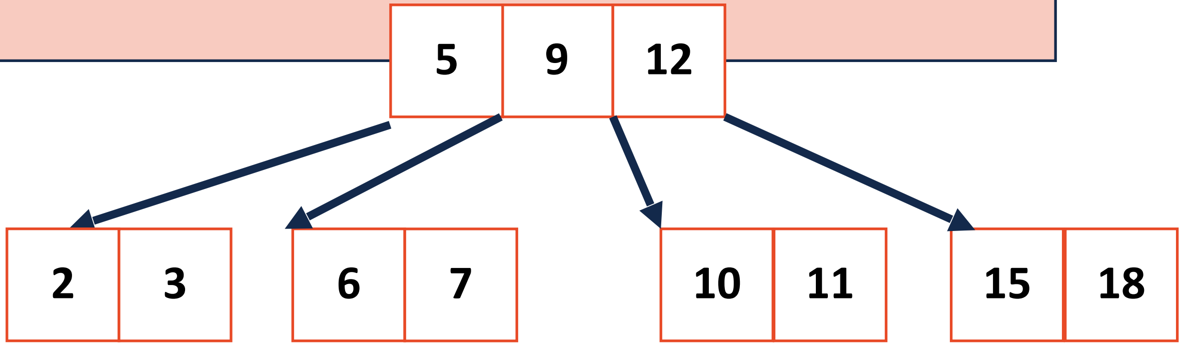
BTree Find

`Find(0)`



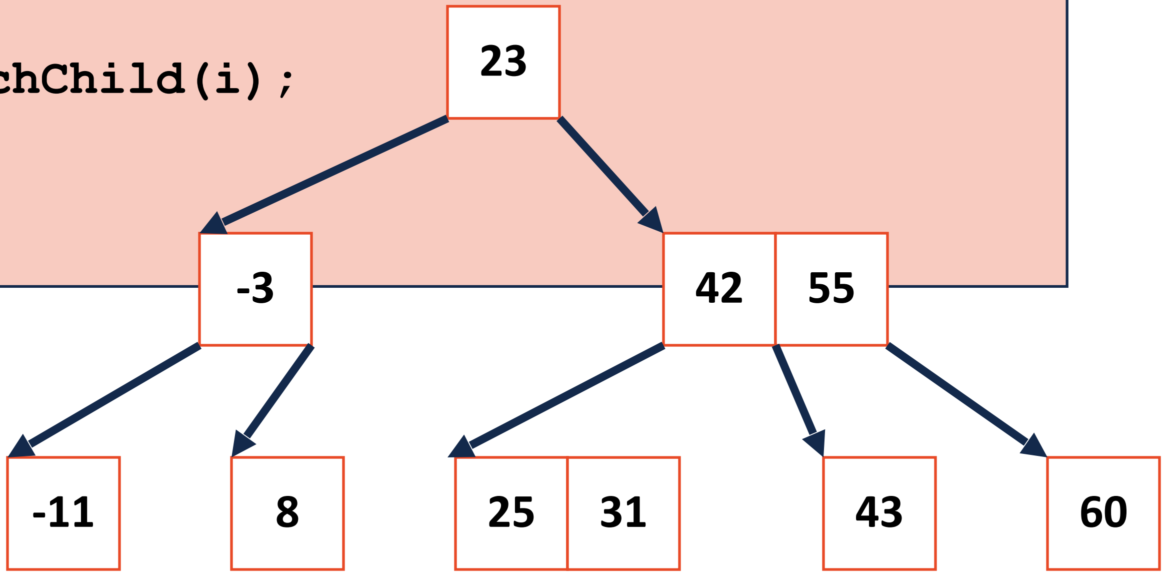
BTree *Exists*

```
bool Btree::_exists(BTreeNode & node, const K & key) {

  unsigned i;
  for ( i = 0; i < node.keycount_ && key > node.keys_[i]; i++) { }

  if ( i < node.keycount_ && key == node.keys_[i] ) {
    return true;
  }

  if ( node.isLeaf() ) {
    return false;
  } else {
    BTreeNode nextChild = node._fetchChild(i);
    return _exists(nextChild, key);
  }
}
```



BTree *Exists*

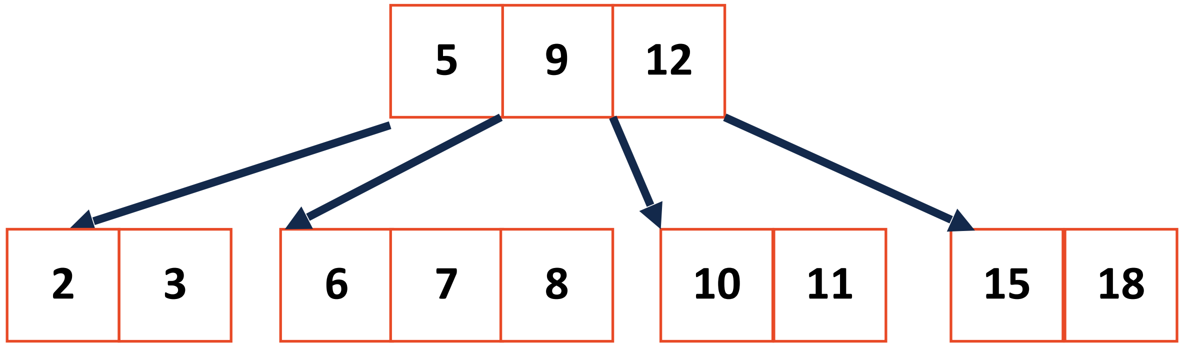
```
bool Btree::_exists(BTreeNode & node, const K & key) {

  unsigned i;
  for ( i = 0; i < node.keycount_ && key > node.keys_[i]; i++) { }

  if ( i < node.keycount_ && key == node.keys_[i] ) {
    return true;
  }

  if ( node.isLeaf() ) {
    return false;
  } else {
    BTreeNode nextChild = node._fetchChild(i);
    return _exists(nextChild, key);
  }
}
```



BTree Remove

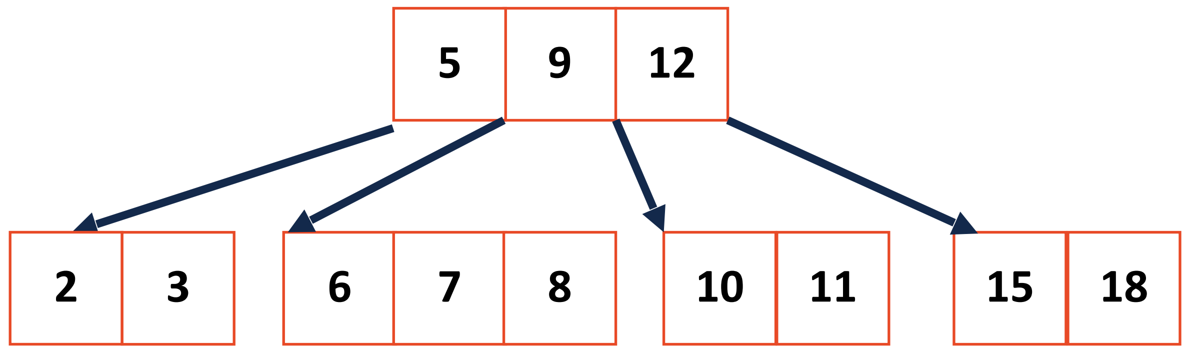
BTree removal is complicated! **It won't be part of the lab.**

However lets consider how we would handle the following cases…



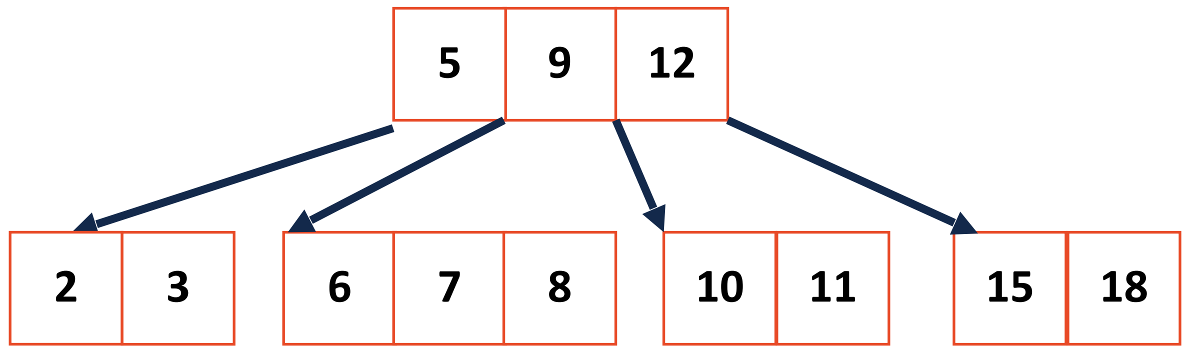
BTree Remove

Remove(8)



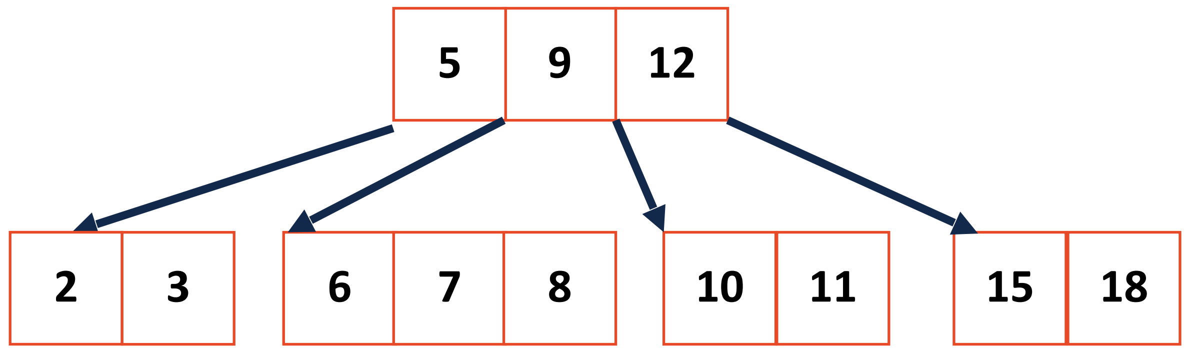
BTree Remove

Remove(2)



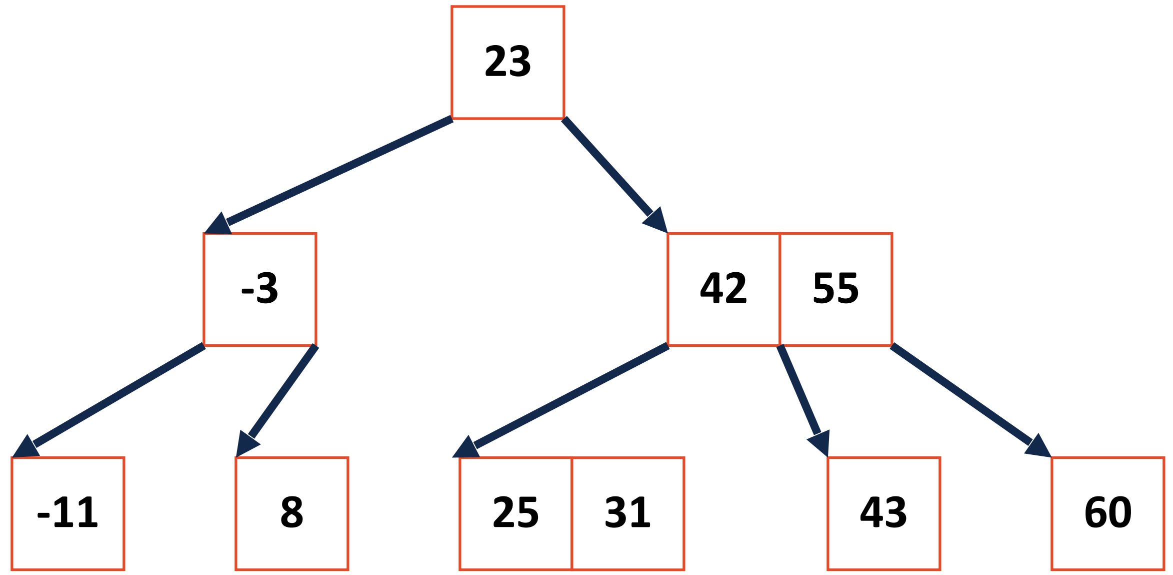
BTree Remove

Remove(15)



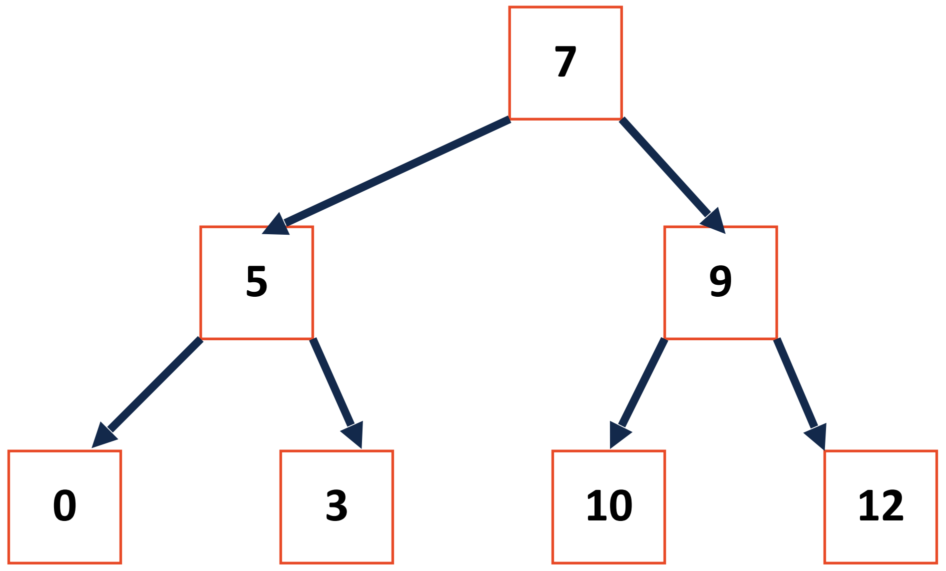
BTree Remove

Remove(42)



BTree Remove

Remove(5)



For next time: BTree Analysis

We’ve seen the ADT

What is the runtime for our BTree operations?