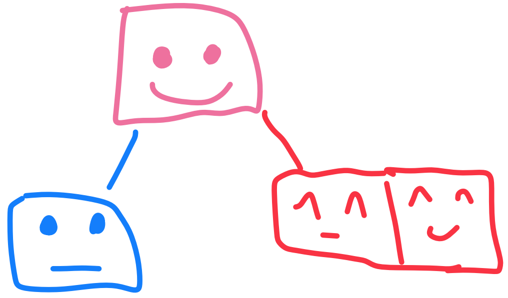
Data Structures
BTree

CS 225

October 4, 2023

Brad Solomon & G Carl Evans



UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN
Department of Computer Science

Exam 2 Summary → Very well!

Average: 87%

Median: 93%

30% of class got 100%

Most missed questions on exam 2 were concepts from exam 1

→ Exams are cumulative!

If you aren't performing as well as you would like…

Go to office hours (at off peak times)!

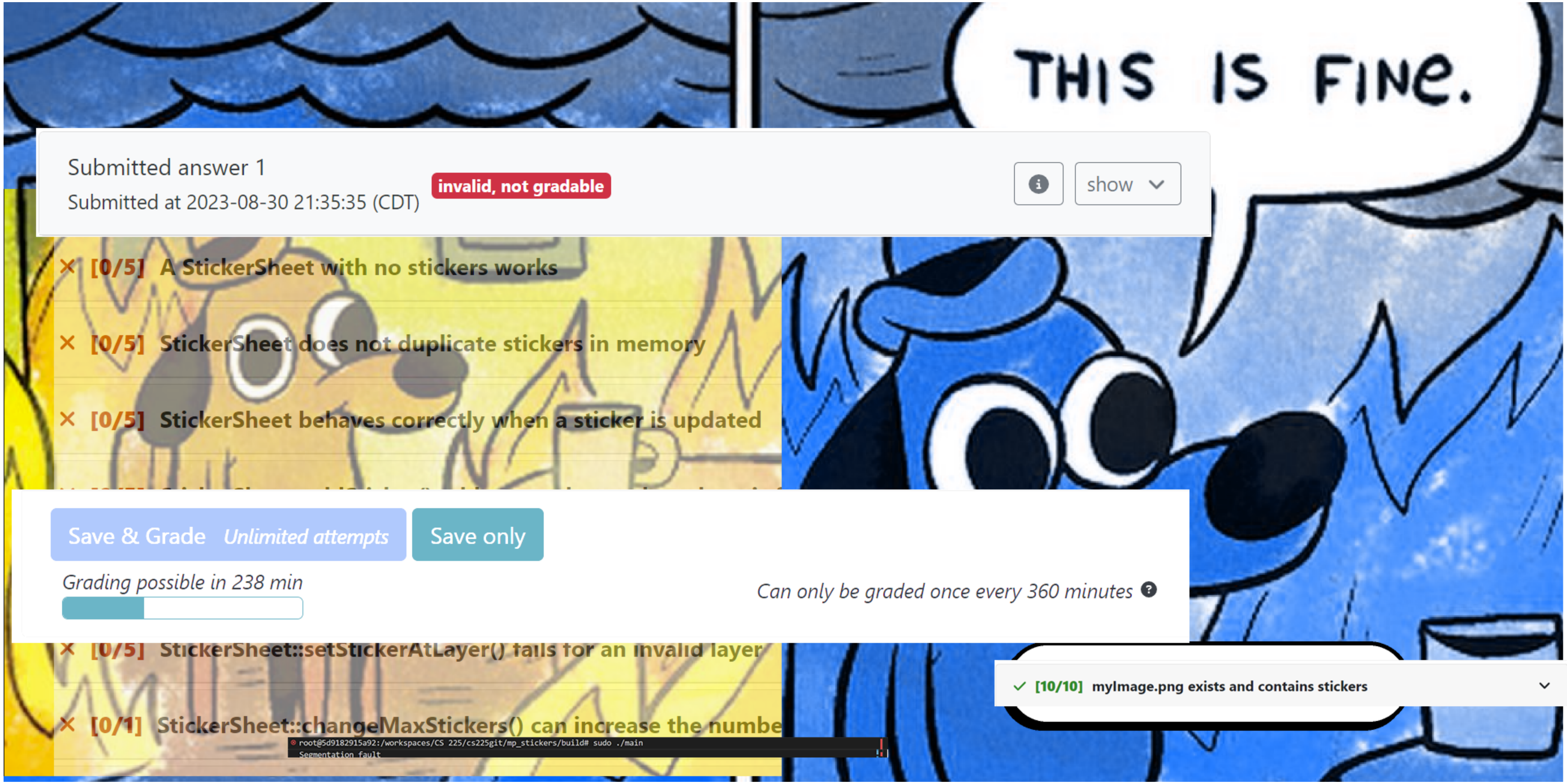
MP Mosaics Extra Credit closes today!

↳ Extended for this mp only

MP Art

All tests passed (409479 assertions in 35 test cases)
root@cf914a57bfac:/workspaces/cs225git/mp_stickers/build#
C++
BUT WHY are ALL THE ERRORS GONE?

MP Art


THIS IS FINE.
Submitted answer 1
Submitted at 2023-08-30 21:35:35 (CDT)
invalid, not gradable
show
✕ [0/5] A StickerSheet with no stickers works
✕ [0/5] StickerSheet does not duplicate stickers in memory
✕ [0/5] StickerSheet behaves correctly when a sticker is updated
Save & Grade Unlimited attempts
Save only
Grading possible in 238 min
Can only be graded once every 360 minutes
✕ [0/5] StickerSheet::setStickerAtLayer() fails for an invalid layer
✓ [10/10] myImage.png exists and contains stickers
✕ [0/1] StickerSheet::changeMaxStickers() can increase the numbe
root@5d9182915a92:/workspaces/CS 225/cs225git/mp_stickers/build# sudo ./main
Segmentation fault

MP Art

B/c mosaics has cool art too!

Learning Objectives

another tree

Discuss alternatives to AVL Trees (and BSTs)

Implement the BTree

Considering hardware limitations

Can we always fit our data in main memory?

↳ No!

Assume it

↳ Yes!

$X \to Y$

$O(1)$

Where else can we keep our data?

Hard drive

External drive

AWS / cloud / download :(

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

No!

BTree Design Motivations

Node

child

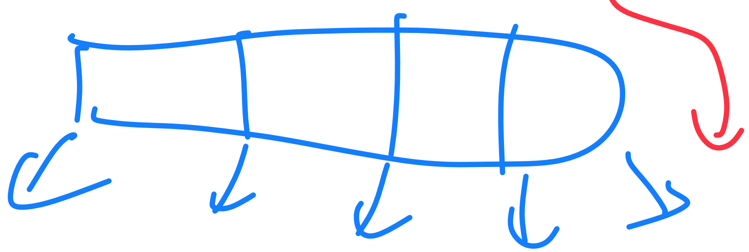
is slow

(key, value)

entry

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

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



2. We want to have a wider tree

↳ More children for every node

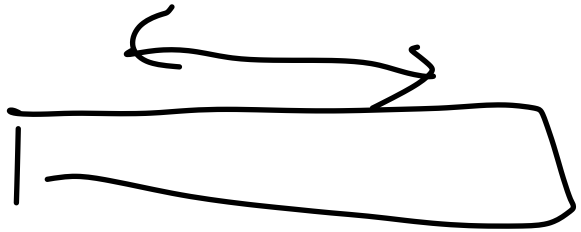
↳ My tree height will be smaller

wide!

3. Make sure my tree is ordered/sorted! (Make data relevant)

only certain trees can do this (easily)

Store my tree as an array in local memory



BTree

O(log n)

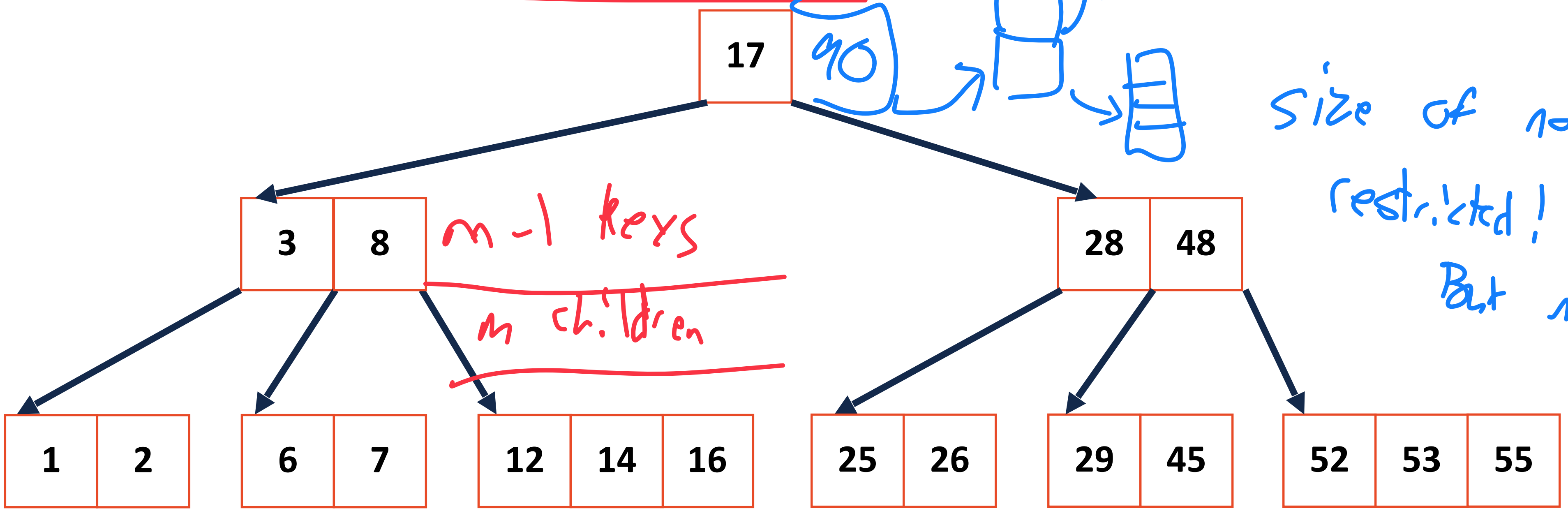
A BTree (of order m) is a m-ary tree

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

All leaves in a BTree are on the same level

of order 2 → Binary tree

m is the # of children



m-1 keys

m children

why?

m-1 keys

size of nodes are restricted! But not by level.

$m \geq 4$

BTree Node (of order m)

4 keys

M ≥ 5

Smallest m-1 Keys largest

-3	5	8	13

an array of values

an array of children

what values

smaller than -3

(-∞, -3]

-3<x<5

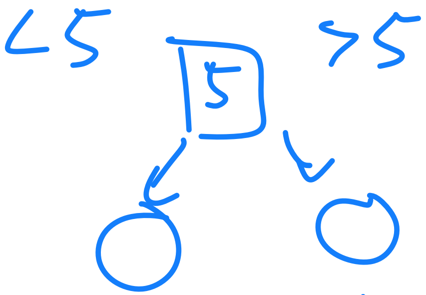
m different "regions"

```
class BTree Node {
    vector <key, value pair> objects;
              ↳ Data Pair
    vector <BTree Node *> children;
}
```

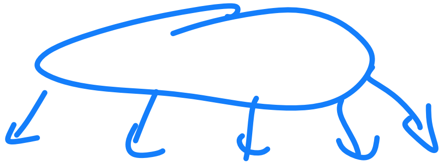
Tradeoff!

↳ faster if prealloc everything

↳ cost is size!



N objects 1 per



→ N objects m per shorter

BTree Node (of order m)

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

Motivation is memory

1 byte == 8 bits

4 bytes = 32 bits

↳ we want to determine this based on need or architecture

Disk Block [L1] ~ RAM

faster seek slower

x | y | Hard Disk | z

obj

↳ Let m equal at most one of these blocks

↳ 512 bytes or 4096 bytes

↳ int is 32 ~~bytes~~ bits!

$$m = \left\lfloor \frac{\text{size of chunk}}{\text{size of object}} \right\rfloor$$

sending info by cloud

TCP Network Packet → ~1500 bytes

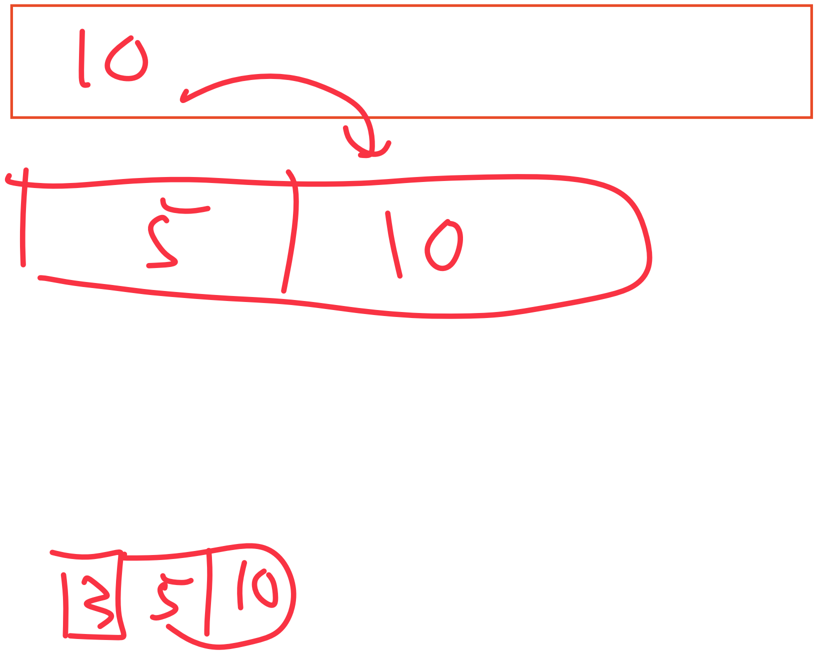
BTree Insertion

M = 5

↳ 4 keys

↳ 5 children

All keys within a BTree node are ordered (sorted)



Insert(10)

Insert (5)

(3)

Array insert is slow!

↳ in a sorted array

BTree Insertion

All keys within a BTree are ordered

M = 5

At most M-1

Allocate M

Allocate M space for each node*

5	7	9	10

insert(2)

10				

Insert(10)

↳ 1 op

5	10			

Insert(5)

2 op

5	7	10		

Insert(7)

3 op

5	7	9	10	

Insert(9)

4 op

2	5	7	9	10

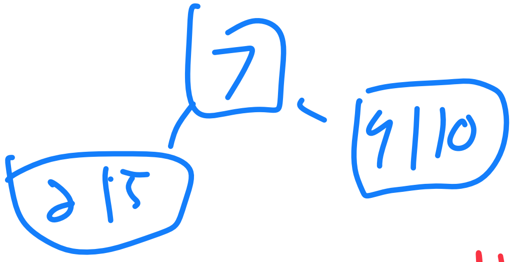
Insert(2)

...

resize when size == M

↳ too big!

↳ We will split array



Know the median element!

M is always odd*

↳ clean median

BTree Insertion

M = 5

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

1) Find median value

2) Split by "throwing "up" the median

↳ we make a new parent

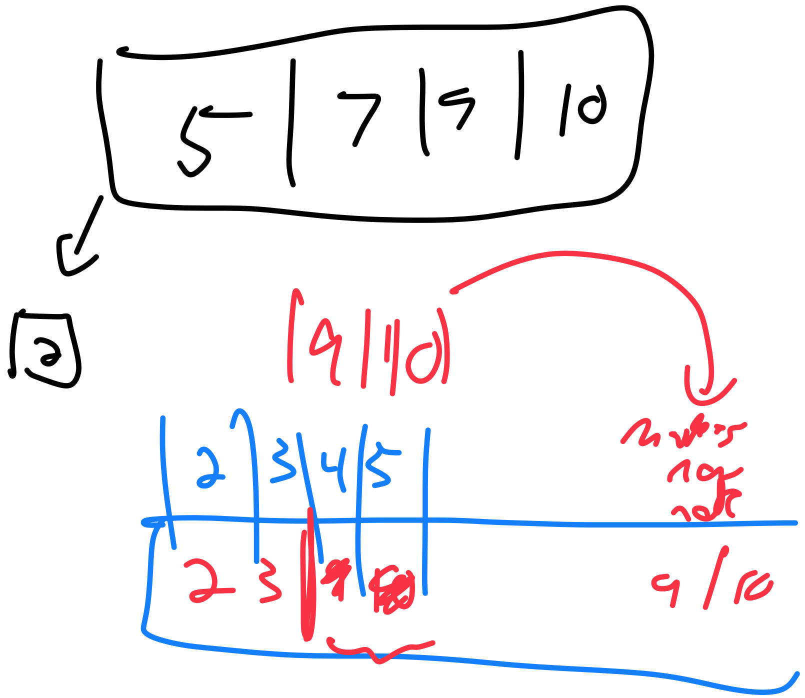
↳ Assume no duplicates

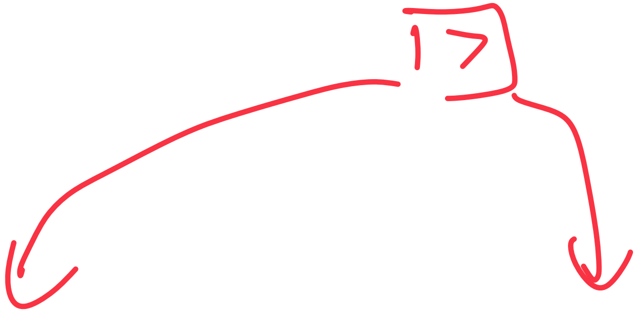
Key

6

vector<vals>

No!





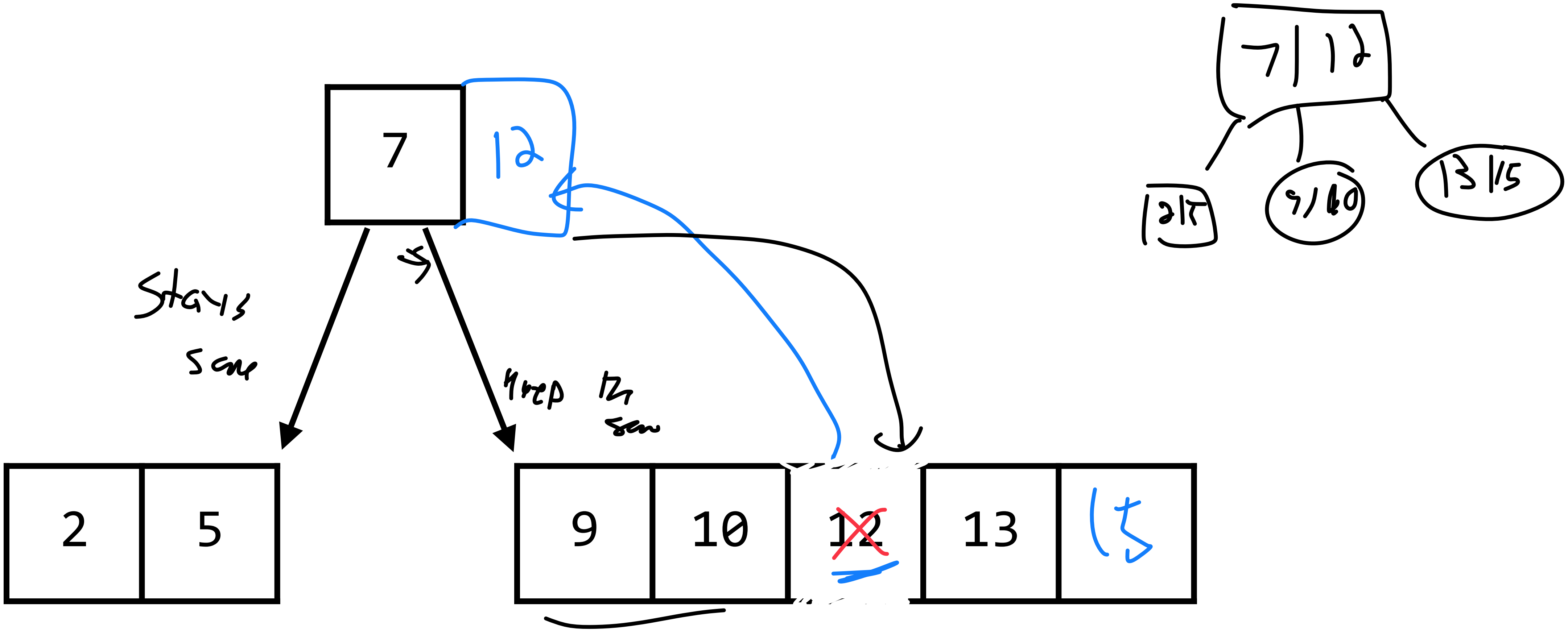
2	5	~~7~~	9	10

BTree Insertion

M = 5

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

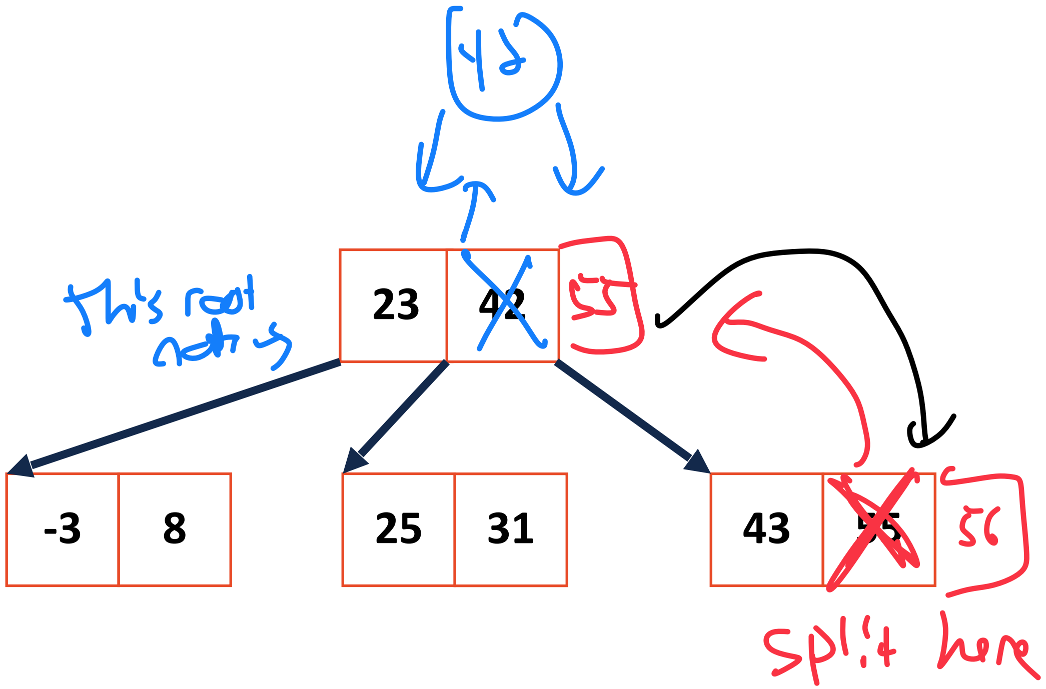
1) Find median

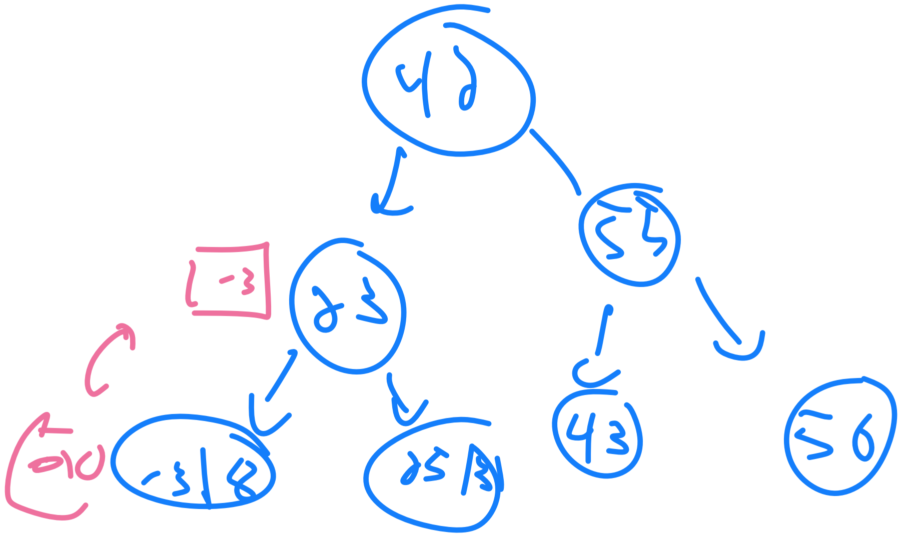


BTree Recursive Insert

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

play around w/ this visualizer!

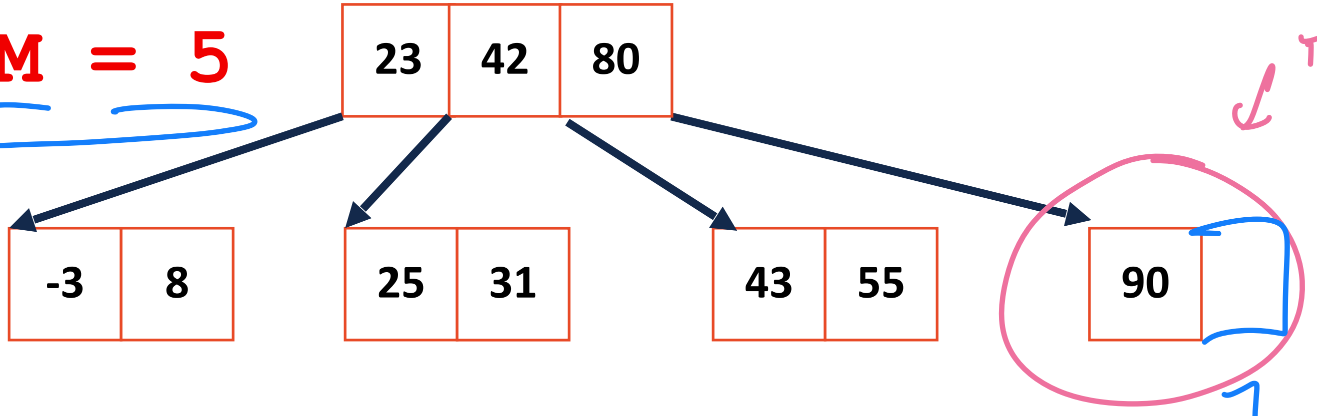
BTree Size Restrictions

$m-1$ on keys (of order m)

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

This is a problem!

M = 5

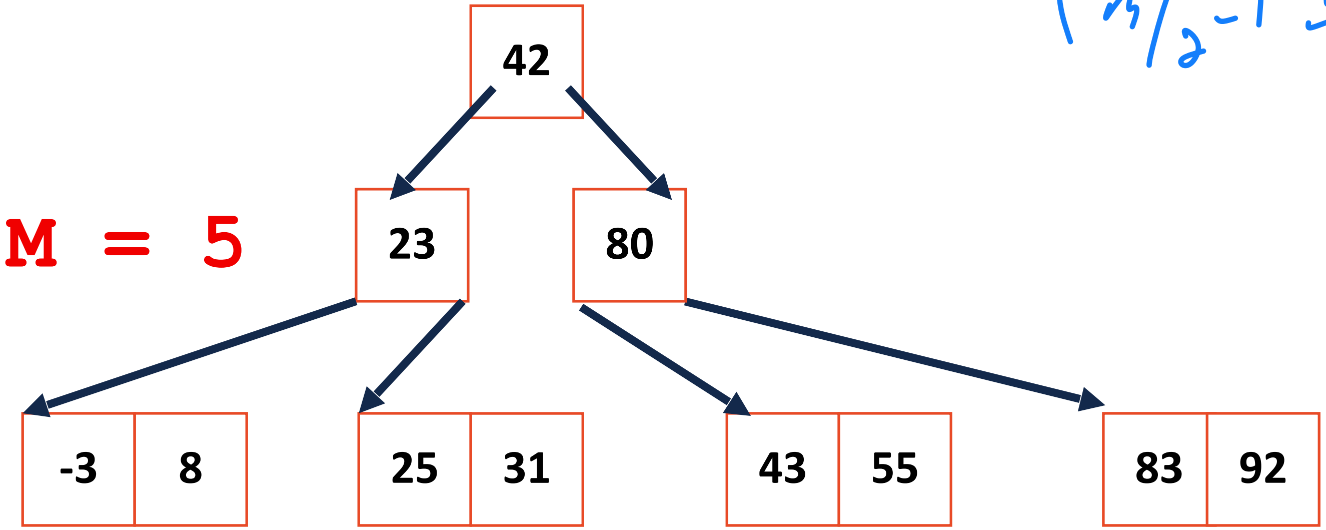


$\lceil m/2 - 1 \rceil \geq 1.5$

Add to array until it splits to this

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

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 nodes contain no more than **m-1** keys.
- All internal nodes have exactly **one more child than keys**

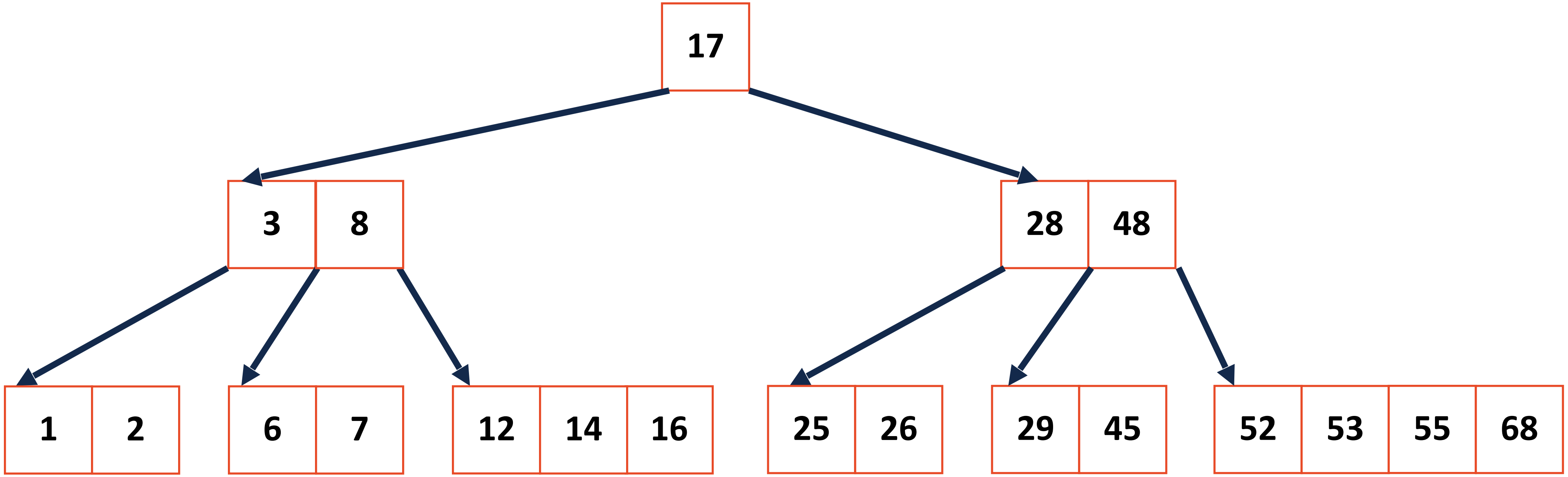
Root nodes can be a leaf or have ______________ children.

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 value of m?



BTree ADT

Constructor

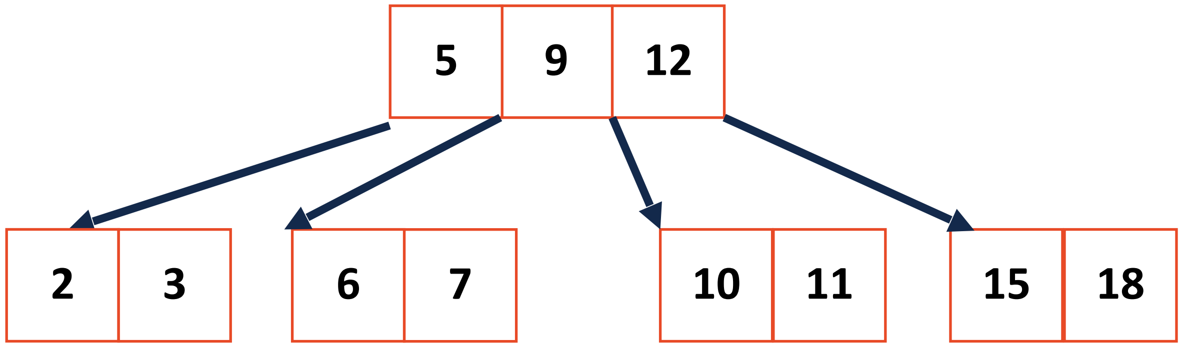
Insert

Find

Delete

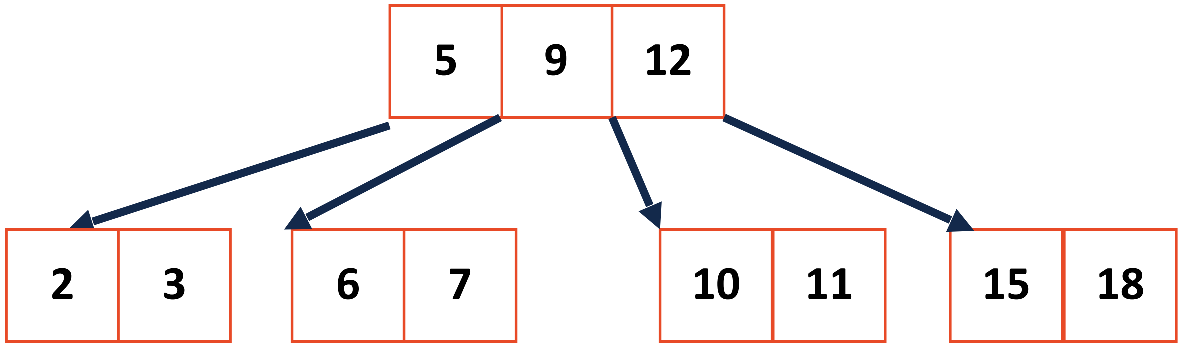
BTree Find

`Find(12)`



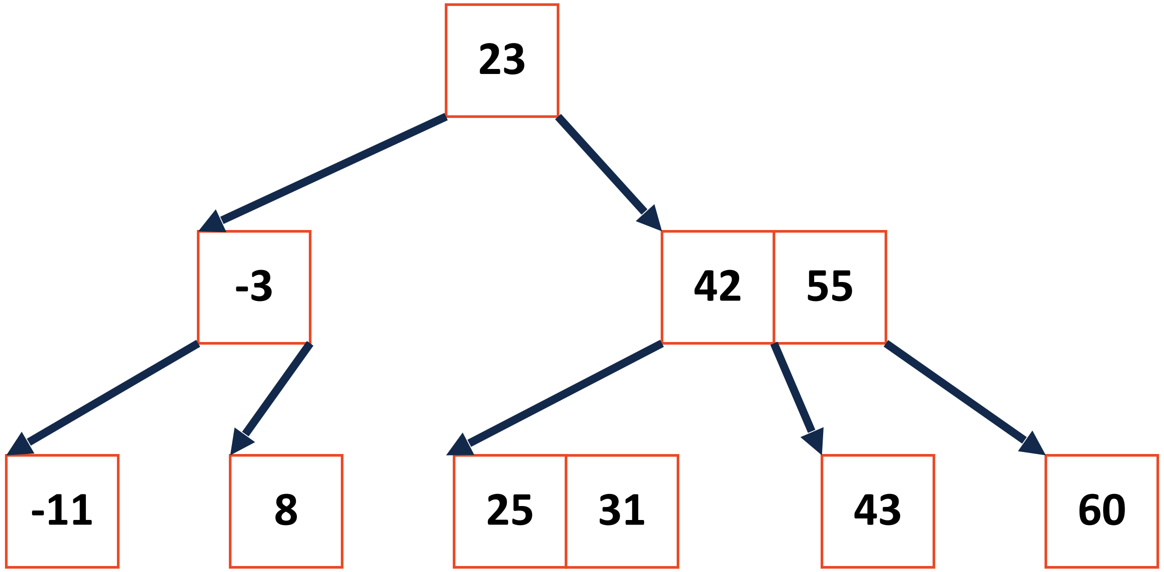
BTree Find

`Find(7)`



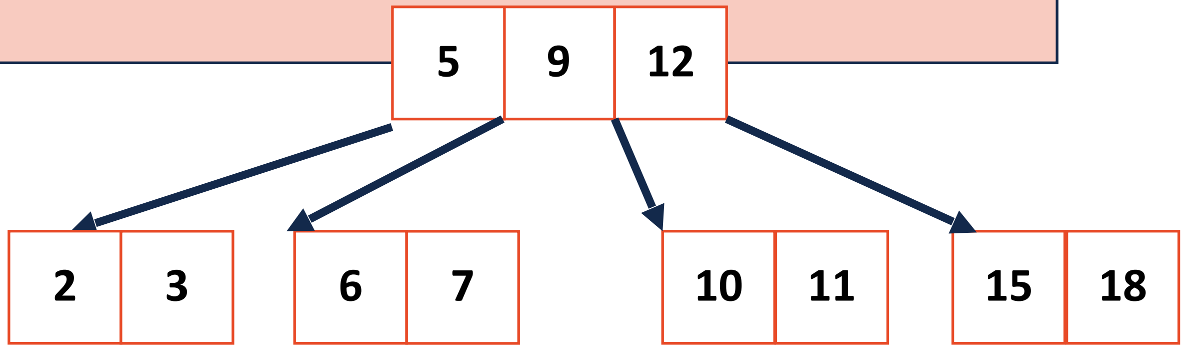
BTree Find

`Find(70)`



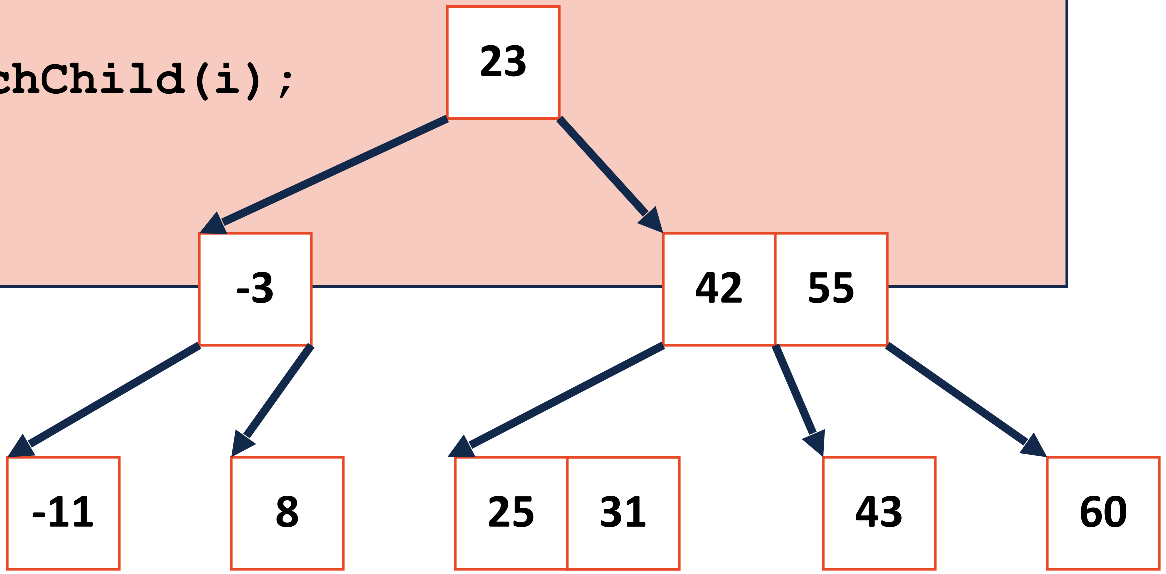
BTree *Exists*

```
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*

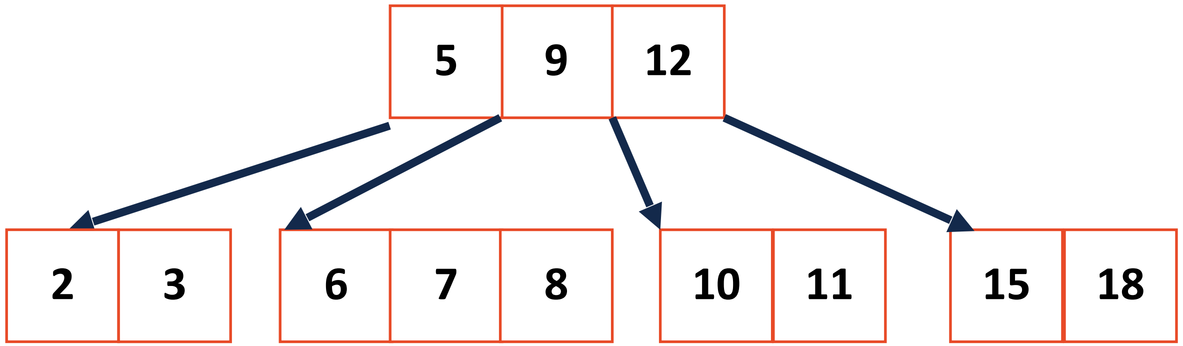
```
bool Btree::_exists(BTreeNode & node, const K & key) {

  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

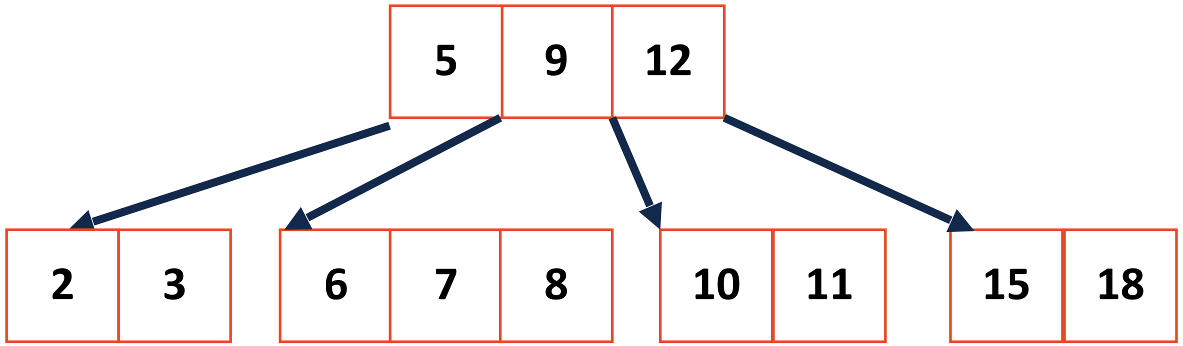
BTree removal is complicated! **It won't be part of the lab.**

However lets consider how we would handle the following cases…



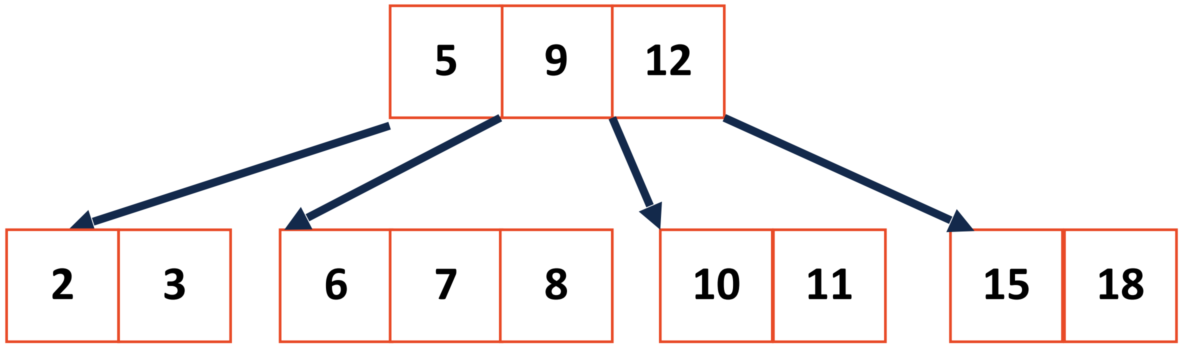
BTree Remove

Remove(8)



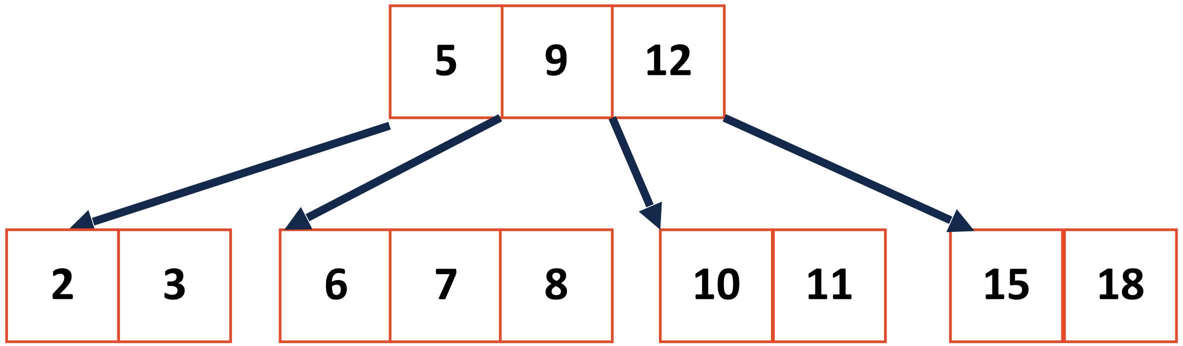
BTree Remove

`Remove(2)`



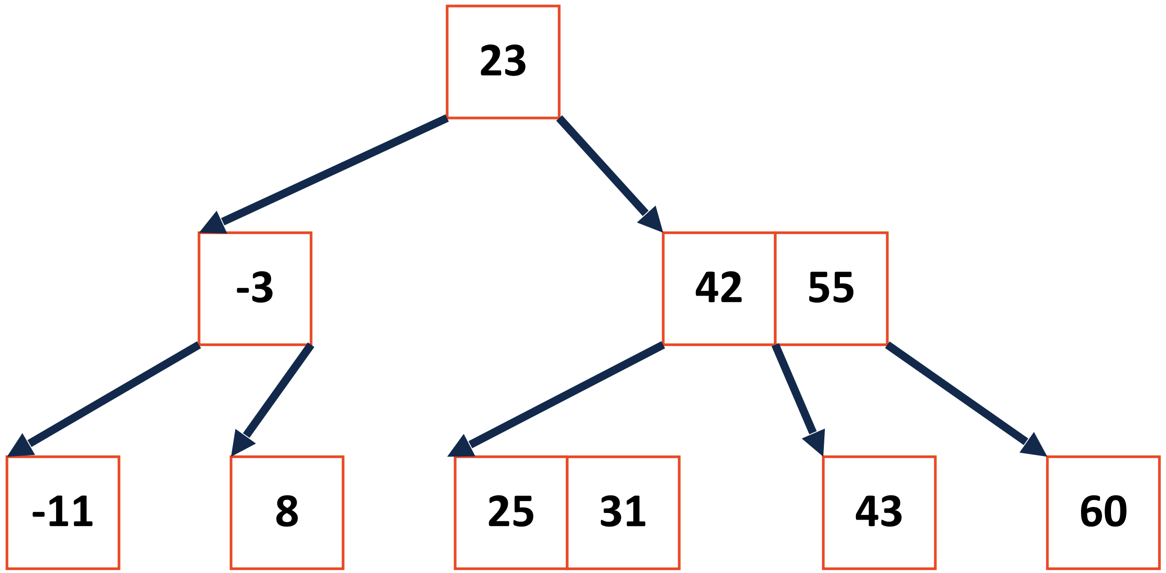
BTree Remove

`Remove(15)`



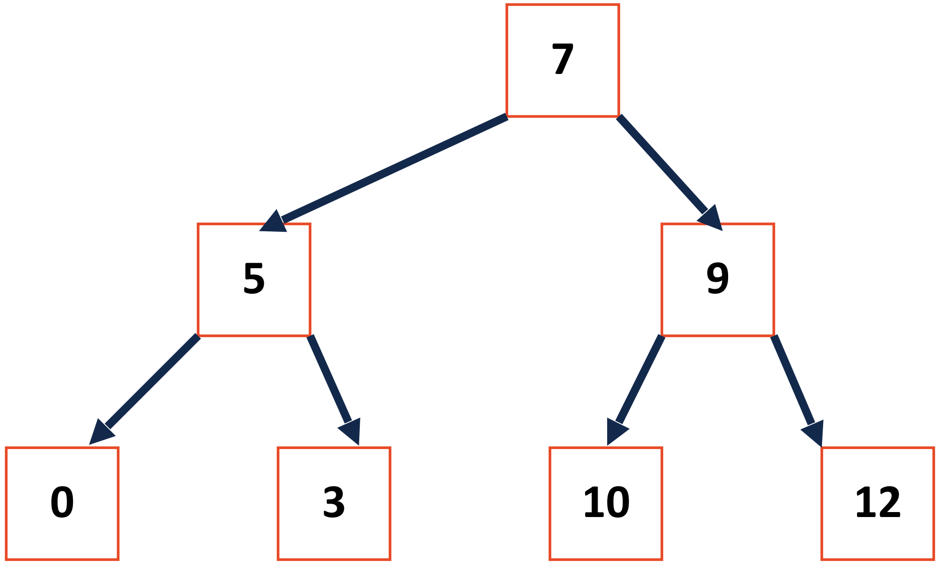
BTree Remove

Remove(42)



BTree Remove

Remove(5)



For next time: BTree Analysis

We’ve seen the ADT

What is the runtime for our BTree operations?