Data Structures

BTree

CS 225

Brad Solomon

October 7, 2024

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Department of Computer Science

Classify: New & Improved Course Explorer

Key Features:

- Average GPA for each course
- Professor on a class page links to Rate My Professor
- Class locations on a class page linked to Google Maps
- Page that ranks Gen-Eds in a category by average GPA
- UI Improvements over old course explorer
- More features in development

Find Us:

- @ClassifyUIUC on Twitter & Instagram
- Visit: https://classify-x.vercel.app/

Feedback Survey EC

MP_stickers: ~74% participation (Great work!)

MP_lists: ~48% participation (Still time to get those points)

IEF: ~55% participation (Oh no! Deadline 10/14!)

Surveys are important! Please do them and get points!

Exam FRQ Regrade Requests

For now: Email cs225admin@lists.cs.illinois.edu

Include ‘Regrade Request’ and Exam number in subject

Write a clear explanation for why you disagree with grade

Exam 1 regrade request deadline: 10/21/24

Exam 2 regrade request deadline: TBD

Learning Objectives

Remind ourselves one (engineering) issue with trees

Introduce (and implement) the B Tree!

Summary of Balanced BST

Pros:

O(log N) for insert, find, remove

Optimal range queries in 1D

Cons:

O(log N) isn't that great

Large in-memory requirement

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. 🎓

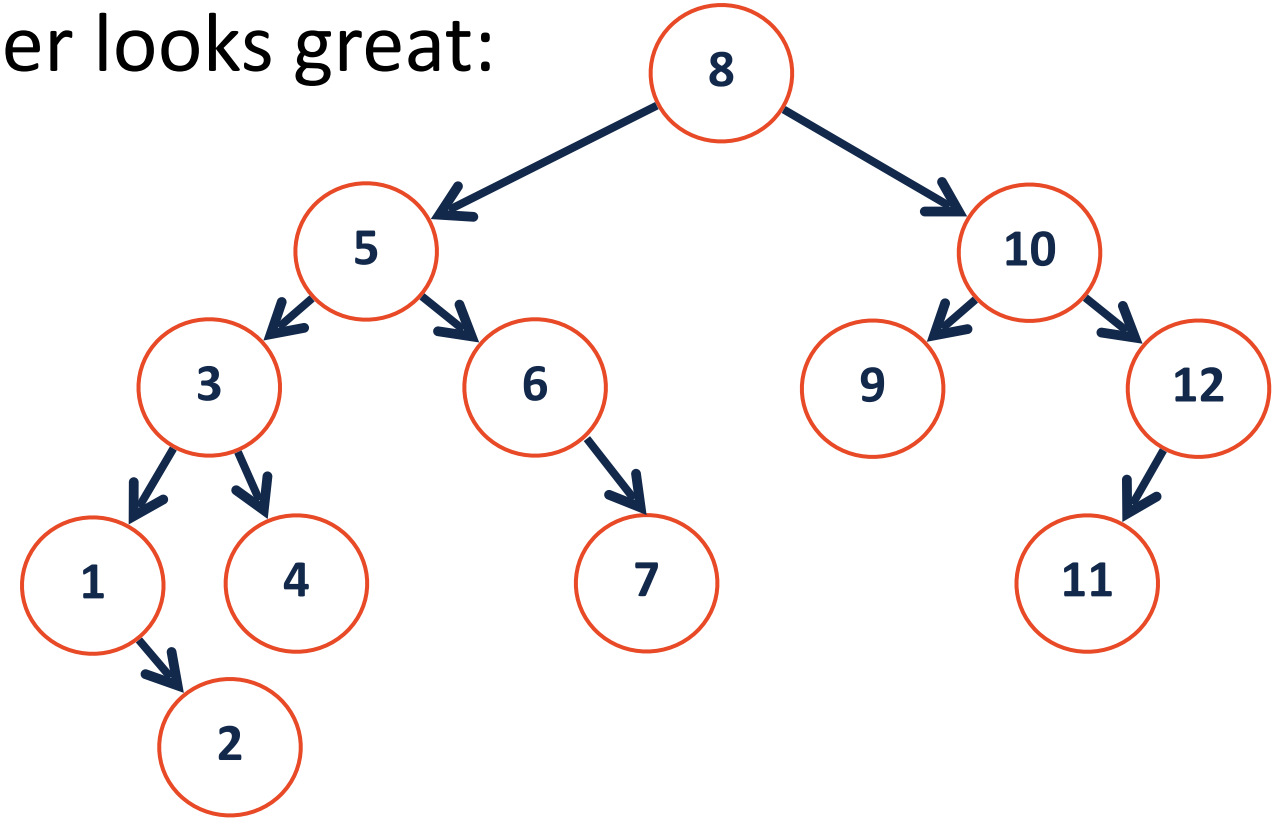
(Care of https://gist.github.com/hellerbarde/2843375)

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:



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)?

Exam FRQ Regrade Requests

For now: Email cs225admin@lists.cs.illinois.edu

Include ‘Regrade Request’ and Exam number in subject

Write a clear explanation for why you disagree with grade

Exam 1 regrade request deadline: 10/21/24

Exam 2 regrade request deadline: TBD

BTree Design Motivations

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

1) Keep the number of seeks low

BTree Design Motivations

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

2) When possible keep data stored locally

BTree Design Motivations

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

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

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 Design Motivations

1) Keep the number of seeks low

Make a tree that is wide and short by…

2) When possible keep data stored locally

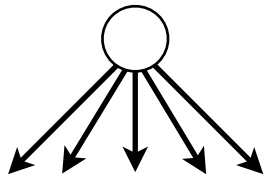
Store more than one key in each node

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

Make sure our tree is still ordered

BTree

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

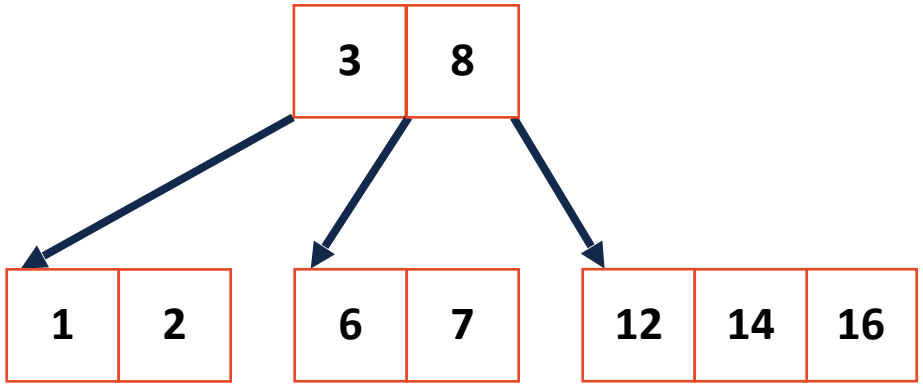


BTree

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

Nodes contain up to **m-1** keys

An internal node of **k** keys has **k+1** children

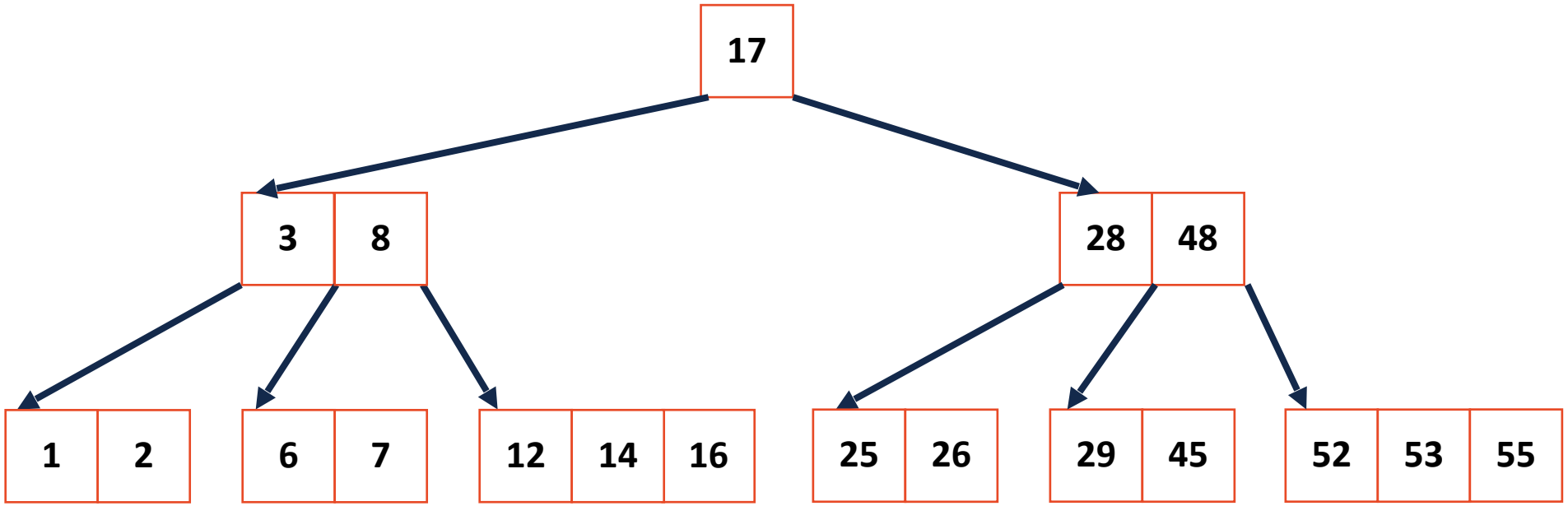


BTree

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

Nodes are ordered with up to **m-1** keys and **|keys|+1** children

All leaves in a BTree are on the same level



BTree ADT

Constructor

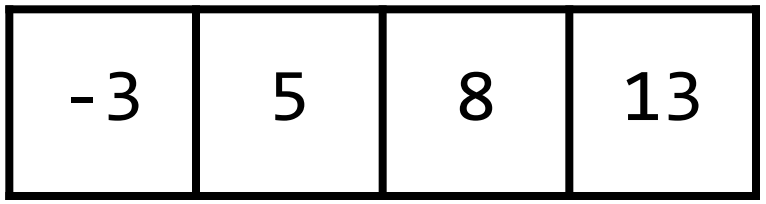
Insert

Find

Delete

BTree Node (of order m)

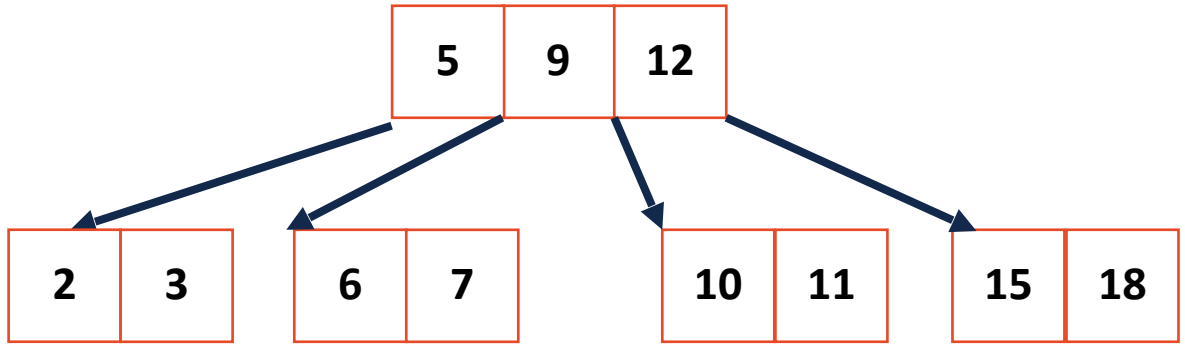
$M \geq 5$




```
1  struct BTreeNode {
2    std::vector<DataPair> elements;
3    std::vector<BTreeNode*> children;
4  };
```

BTree Find

Find(12)



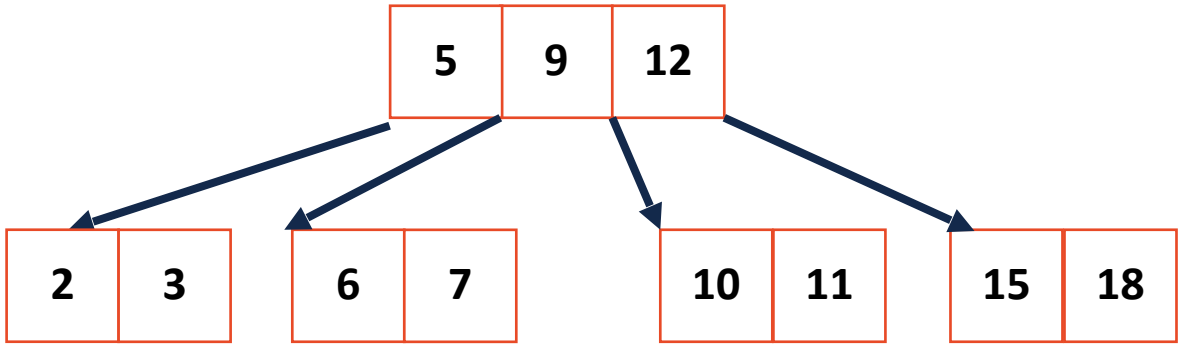
BTree Find

`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!

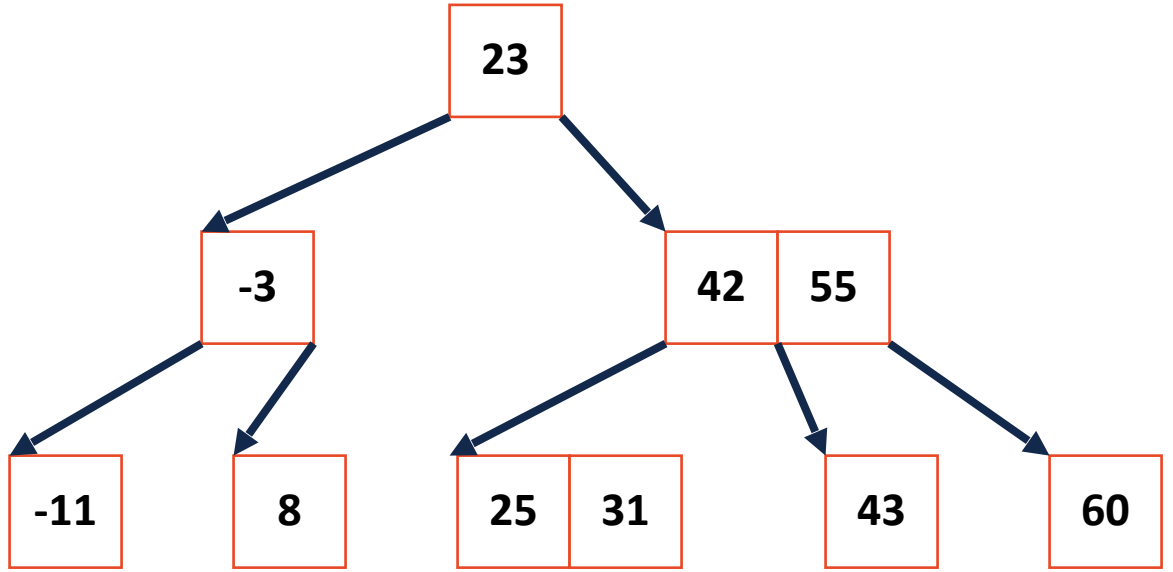
BTree Find

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

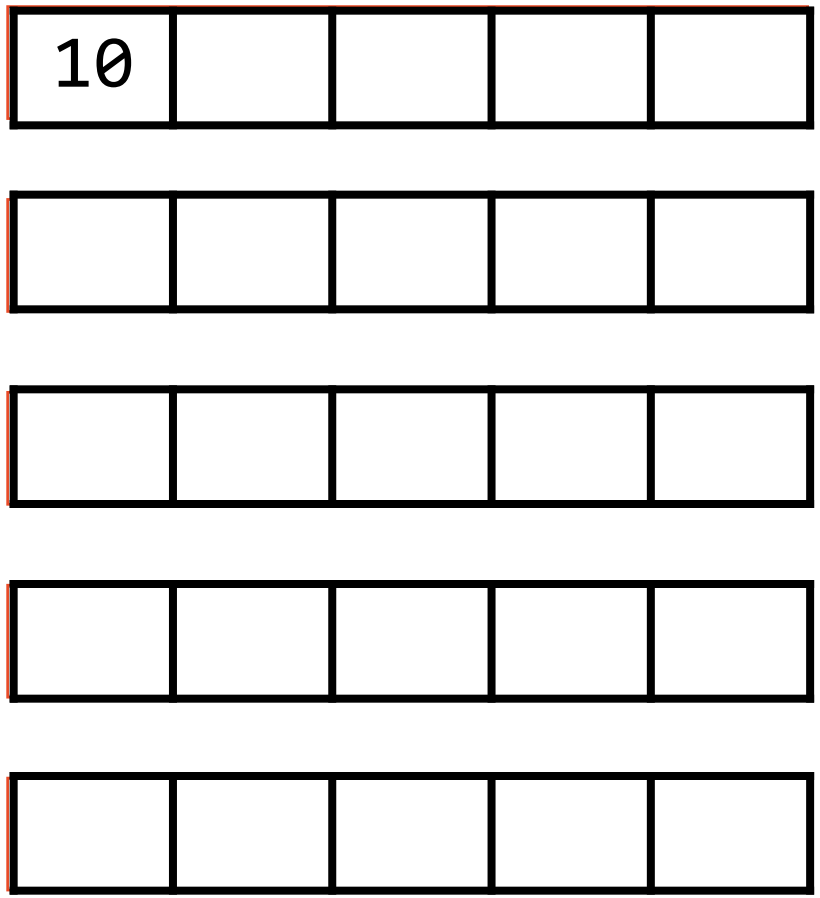
Given an empty BTree, we make a new root node which has…

`Insert(10)`

BTree Insertion

M = 5

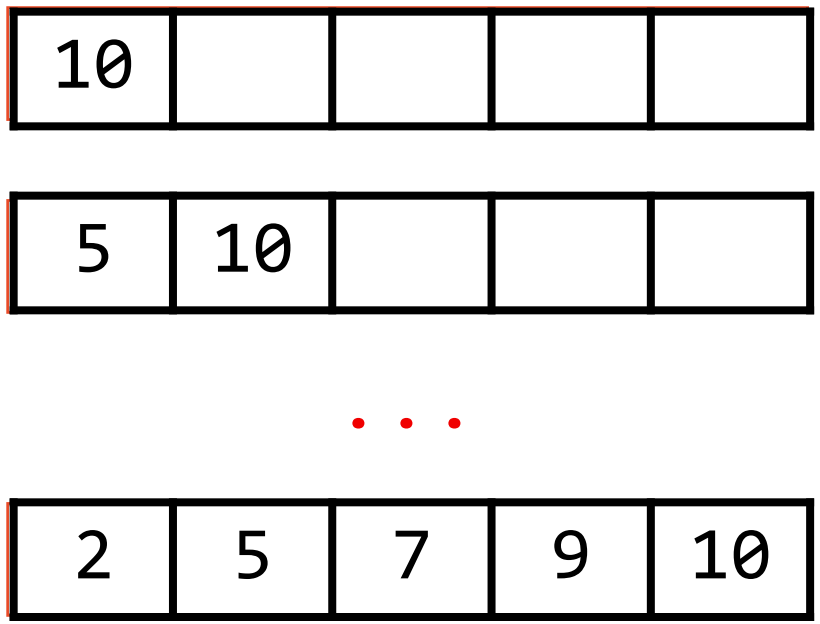
Chain insertions fill our array in sorted order



BTree Insertion

M = 5

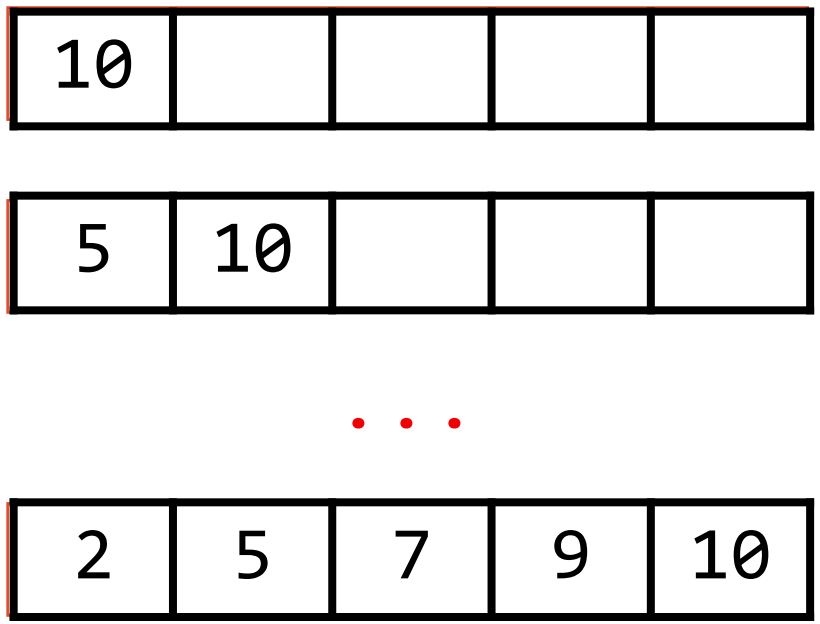
Problem 1: Sorted array insert is slow!



BTree Insertion

M = 5

Problem 1: Sorted array insert is slow!



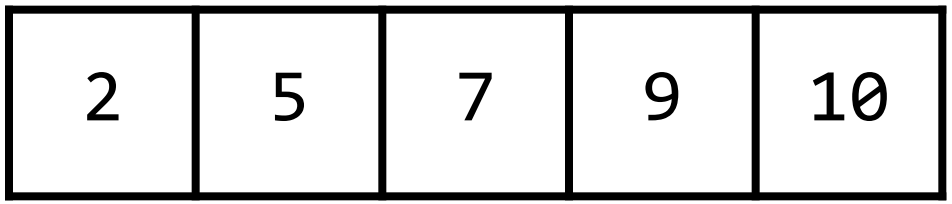
Solution: M is a constant! (So no its not)

BTree Insertion

M = 5

Problem 2: A BTree of order **M** can only store **M-1** keys!

When a BTree node reaches **M** keys, what do we do?

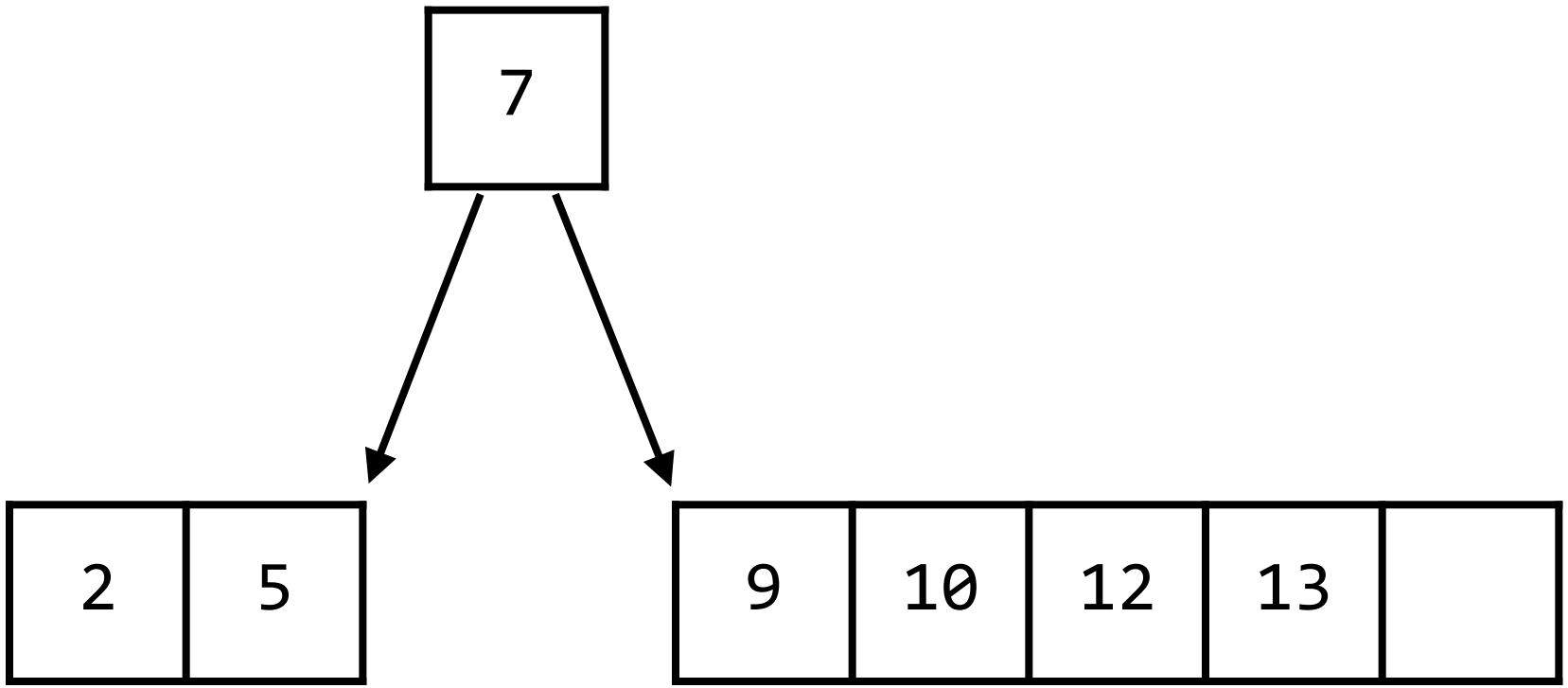


BTree Insertion

M = 5

Problem 2: A BTree of order **M** can only store **M-1** keys!

Solution: When we hit **M** items, split and make a new **parent node!**

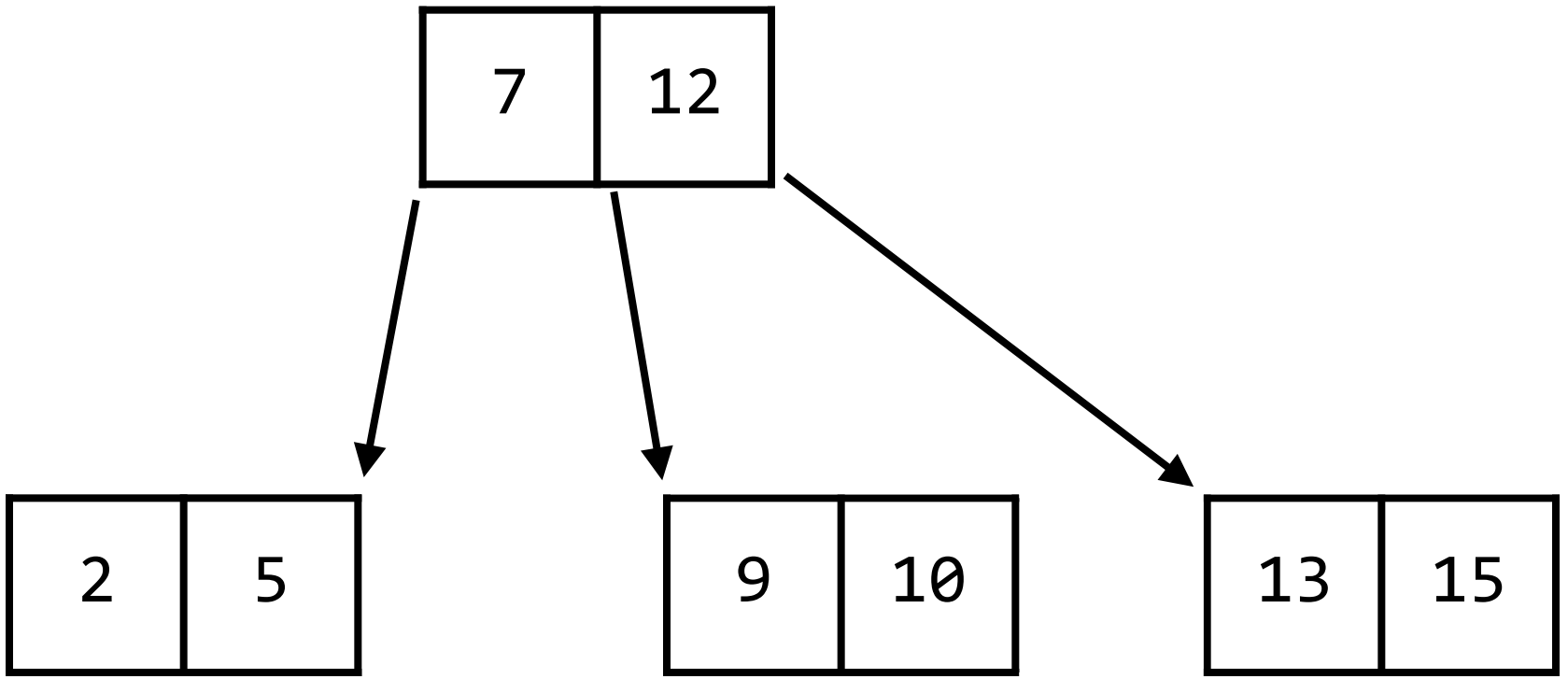


BTree Insertion

M = 5

Problem 2: A BTree of order **M** can only store **M-1** keys!

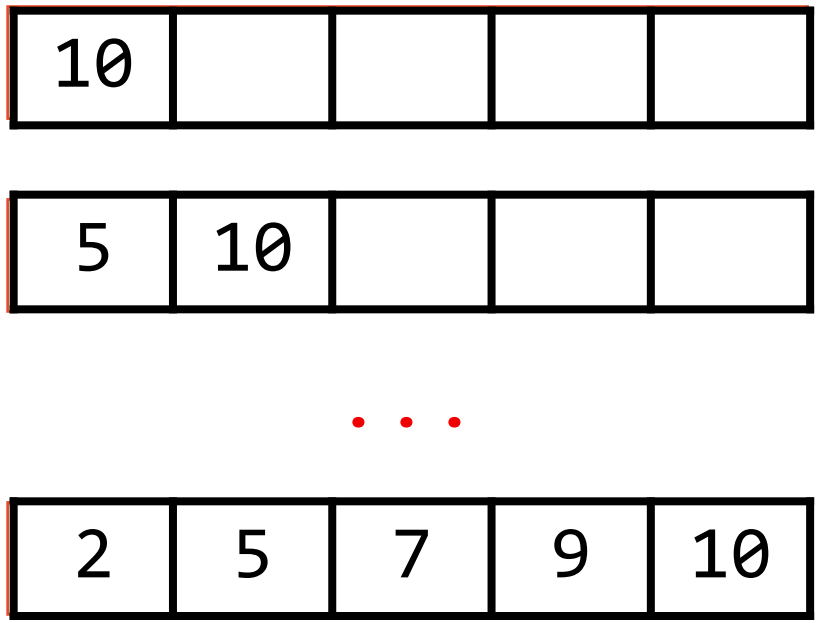
Solution: When we hit **M** items, split and make a new **parent node!**



BTree Insertion

M = 5

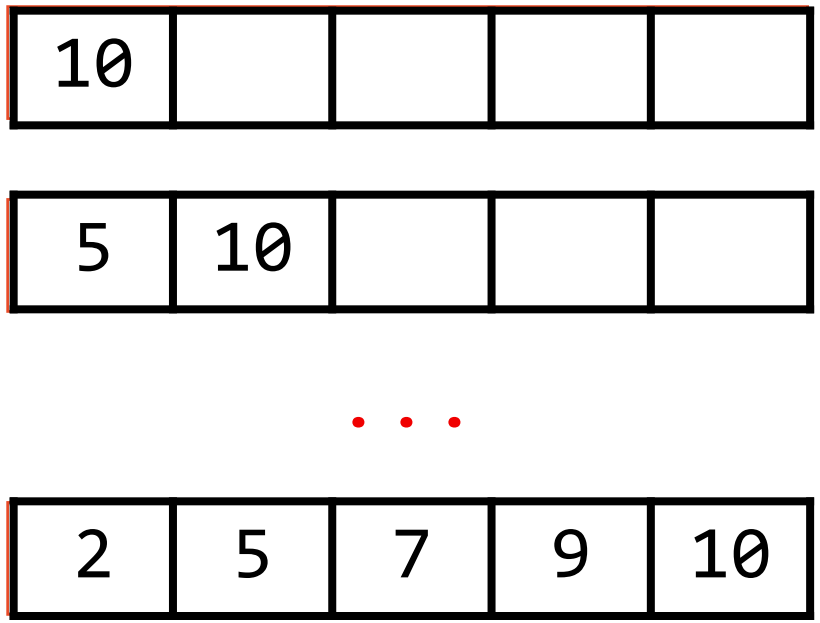
Problem 3: I need to find median value AFTER inserting the **M**th value



BTree Insertion

M = 5

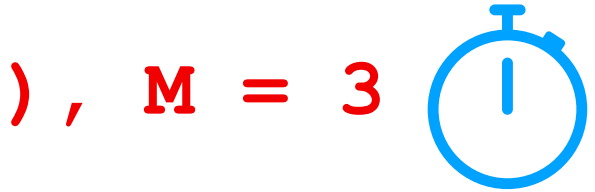
Problem 3: I need to find median value AFTER inserting the **M**th value



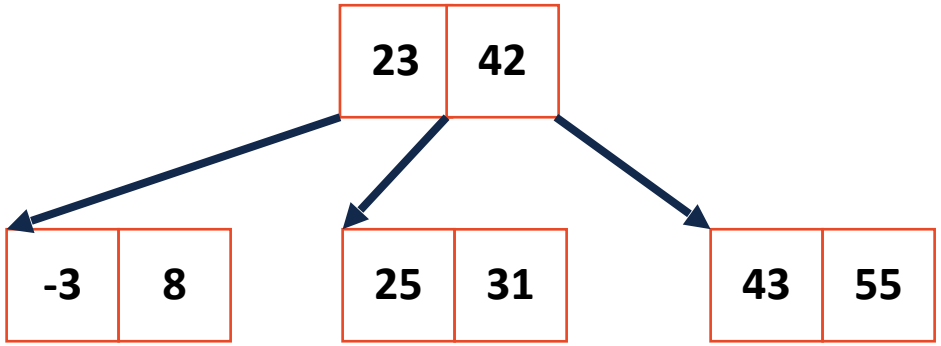
Non-Optimal Solution: Pre-allocate **M** size arrays for every node!

BTree Recursive Insert

`Insert(56), M = 3`



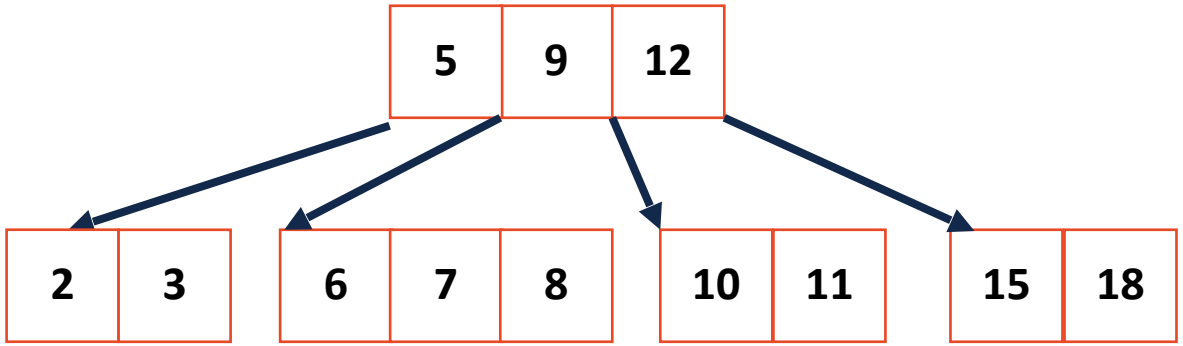
Insert always starts at a leaf but can propagate up repeatedly.



BTree Remove

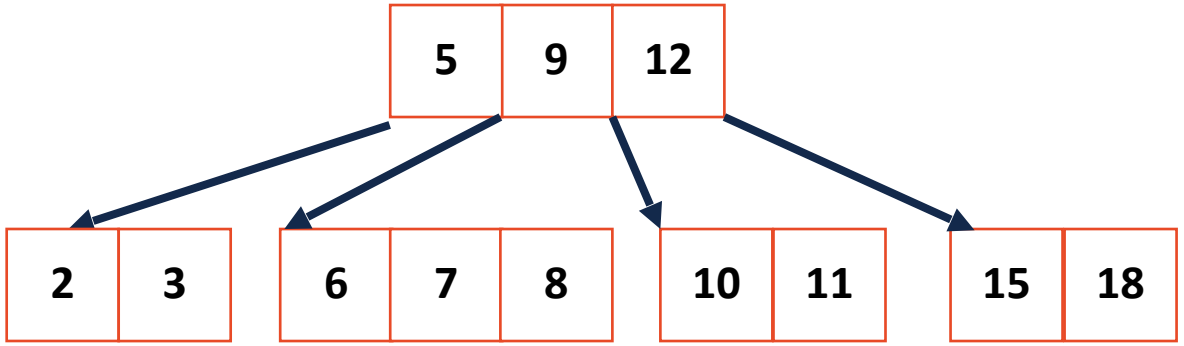
BTree removal is complicated! **It won't be part of the lab.**

However lets consider how we would handle the following cases…



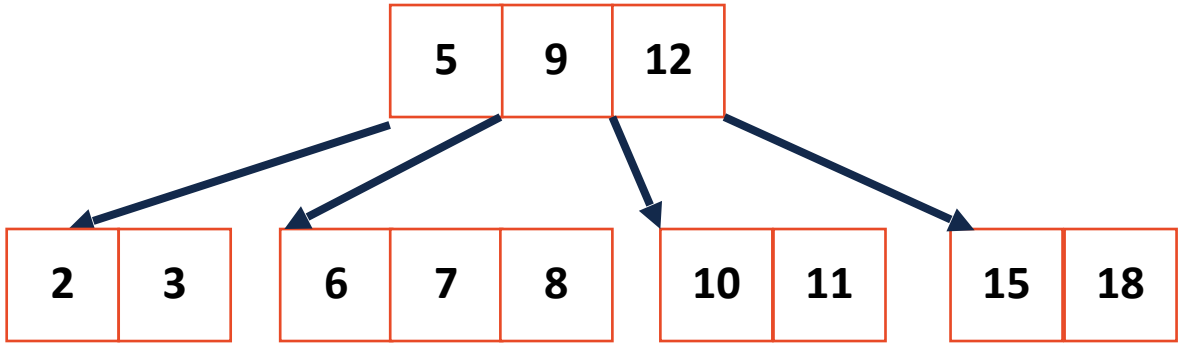
BTree Remove

Remove(8)



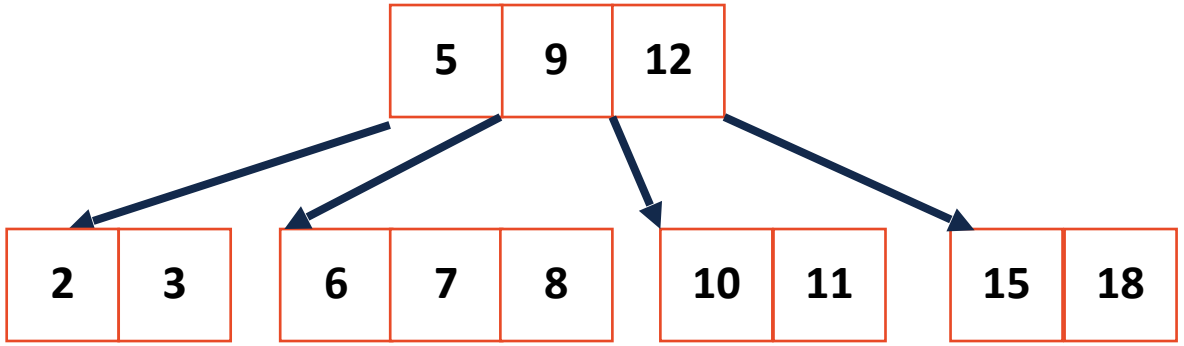
BTree Remove

Remove(2)



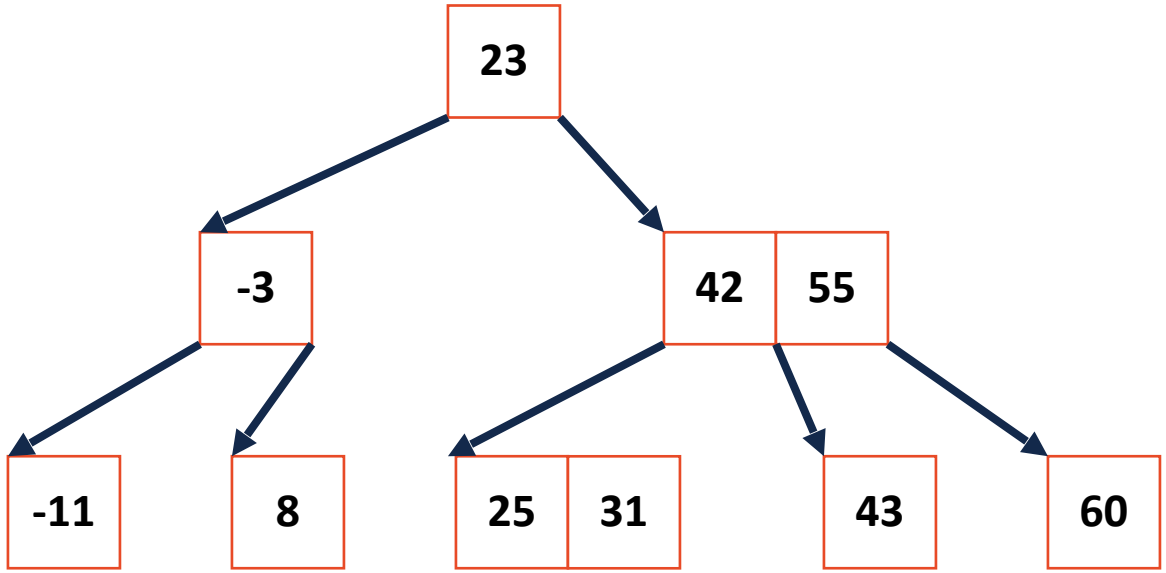
BTree Remove

Remove(15)



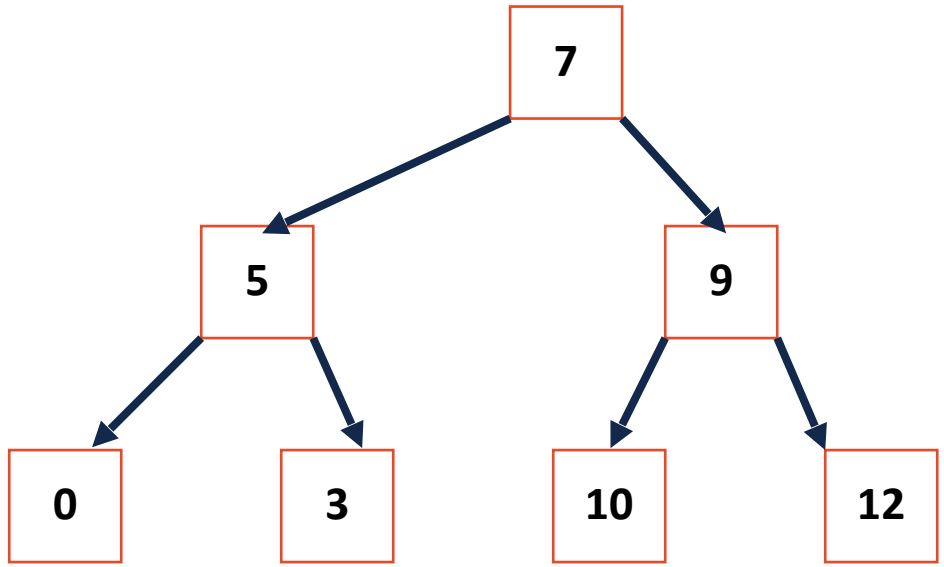
BTree Remove

Remove(42)



BTree Remove

Remove(5)



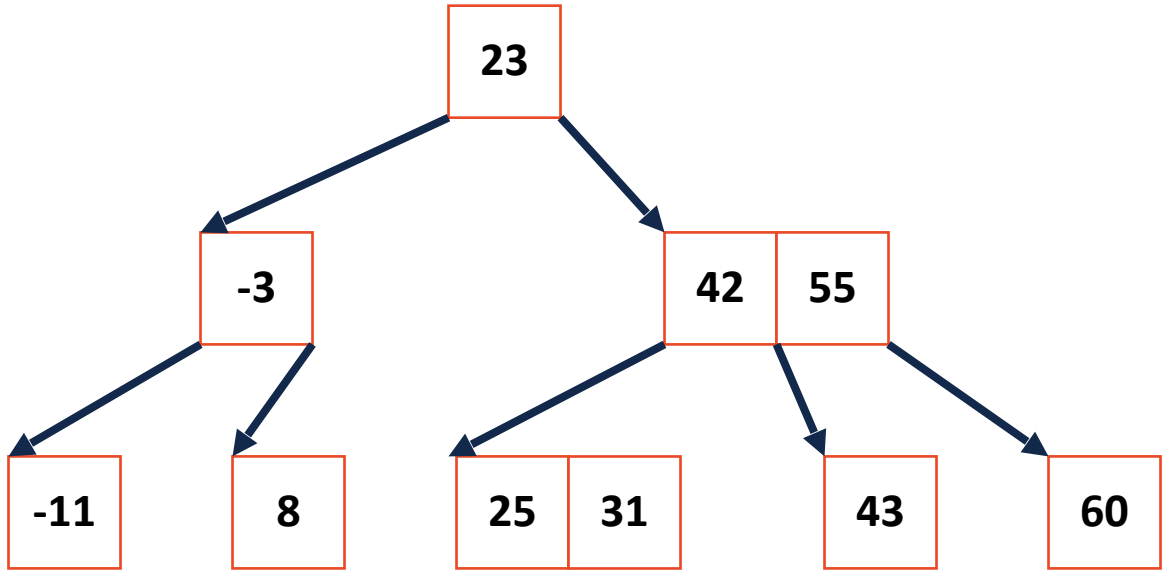
BTree Visualization/Tool

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

For next time: BTree Analysis

We've seen the ADT

What is the runtime for our BTree operations?

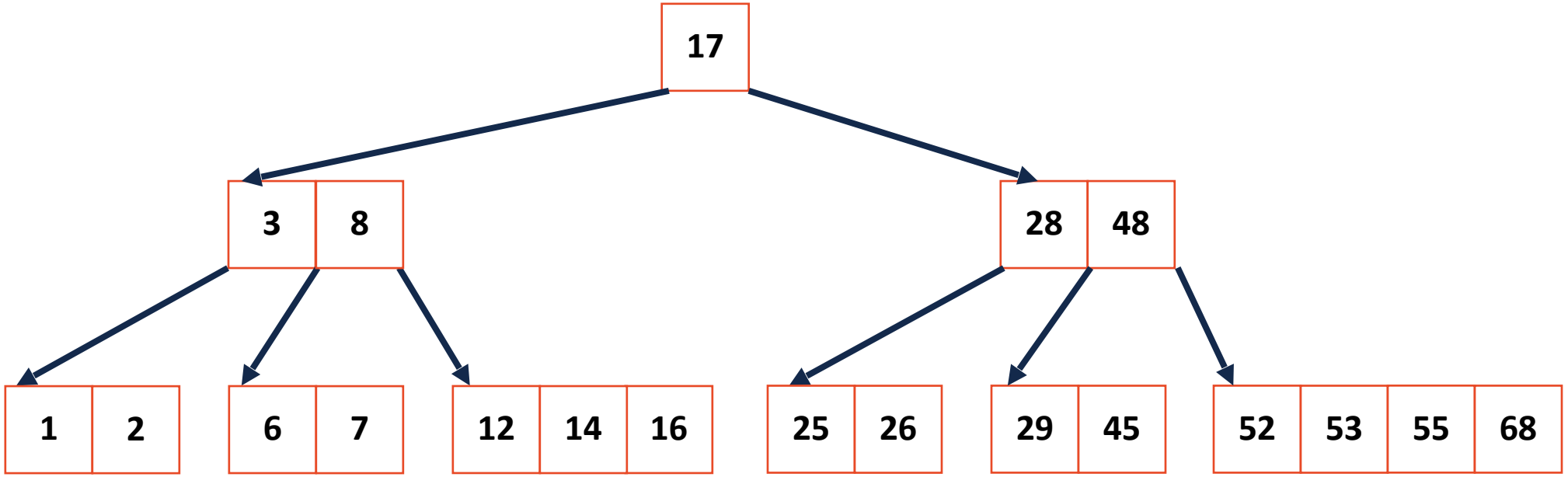


BTree Node (of order m)

Brainstorm together: What value of **m** should we be using?

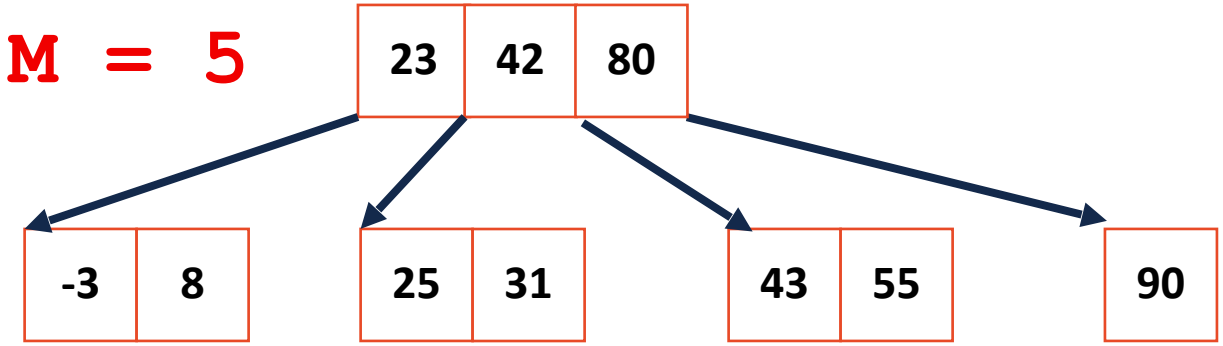
BTree

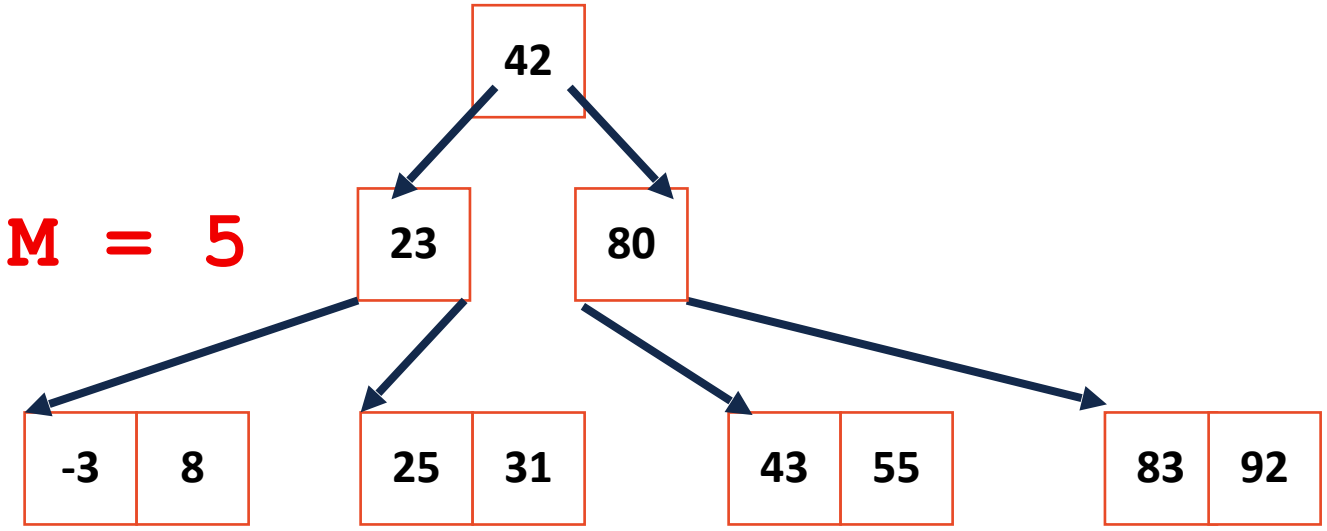
If I tell you this is a valid BTree, what is the value of m?



BTree Size Restrictions

By definition we have max, 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 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.