B-Trees

CS 225 - Fall 2025

Mattox Beckman / Based on slides from Brad Solomon

Objectives

Objectives

- Explain how engineering reality can affect trees
- Explain how B Tree find and insert work

Announcements

- There were a LOT of cases for MP Lists
- Informal Early Feedback in the Discord

Balanced BST Summary

Pros:

Balanced BST Summary

Pros:

$\mathcal{O}(\log\ n)$ insert

Optimal 1D range queries

Balanced BST Summary

Pros:

$\mathcal{O}(\log\ n)$ insert

Optimal 1D range queries

Cons:

Balanced BST Summary

Pros:

- $\mathcal{O}(\log\ n)$ insert
- Optimal 1D range queries

Cons:

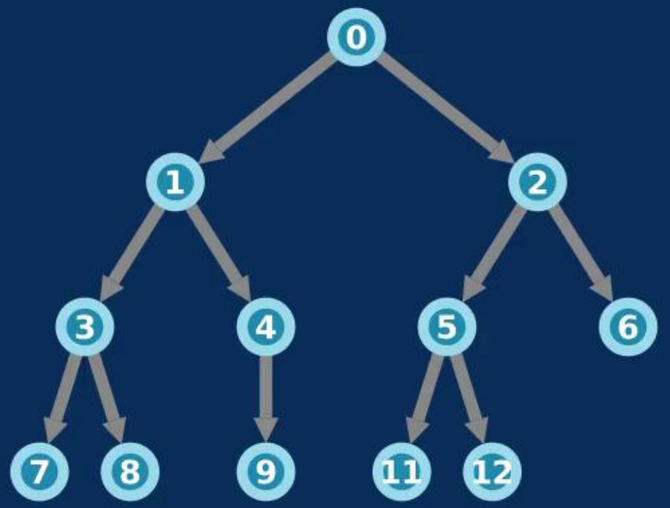
- $\mathcal{O}(\log\ n)$ is actually slow
- Needs a **LOT** of memory

Engineering vs Theory

Big-O assumes all operations take the same time

This is not always true!

Can you think of examples?



Hardware Limitations

Can we always fit our data in main memory?

Where else can we keep our data?

Does this match our assumption that lookups are $\mathcal{O}(n)$?

Hardware Limitations

Can we always fit our data in main memory?

- No! Just check your cell phone camera....

Where else can we keep our data?

Does this match our assumption that lookups are $\mathcal{O}(n)$?

Hardware Limitations

Can we always fit our data in main memory?

- No! Just check your cell phone camera....

Where else can we keep our data?

- Hard Drives, the Cloud, Mass Storage....

Does this match our assumption that lookups are $\mathcal{O}(n)$?

Hardware Limitations

Can we always fit our data in main memory?

- No! Just check your cell phone camera....

Where else can we keep our data?

- Hard Drives, the Cloud, Mass Storage....

Does this match our assumption that lookups are $\mathcal{O}(n)$?

- Nope. See https://gist.github.com/hellerbarde/2843375

B Tree Design Motivations

Keep the number of seeks low

When possible store data locally

Make sure data is relevant

B Tree Design Motivations

Keep the number of seeks low

- Make a tree that is wide and short

When possible store data locally

Make sure data is relevant

B Tree Design Motivations

Keep the number of seeks low

- Make a tree that is wide and short

When possible store data locally

- Store more than one key per node

Make sure data is relevant

B Tree Design Motivations

Keep the number of seeks low

- Make a tree that is wide and short

When possible store data locally

- Store more than one key per node

Make sure data is relevant

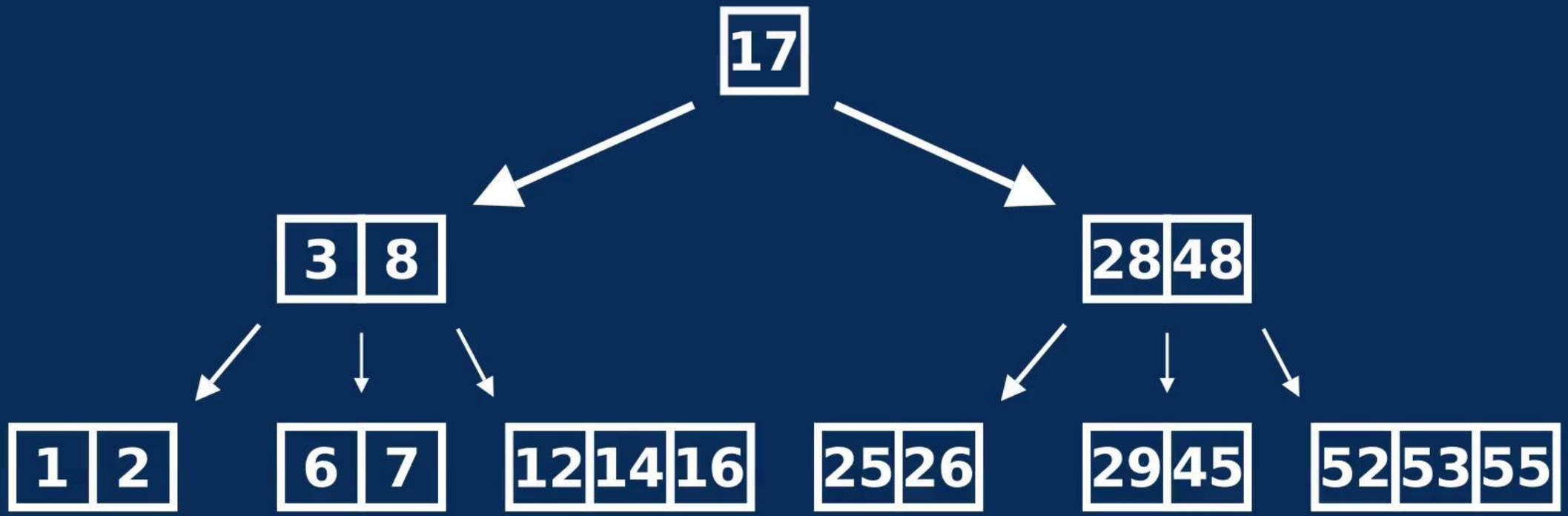
- Make sure the tree is ordered

Definitions

A B Tree of order m is an m-ary tree

Nodes are **ordered**, have up to $m - 1$ keys and $keys + 1$ children

All leaves are on the same level



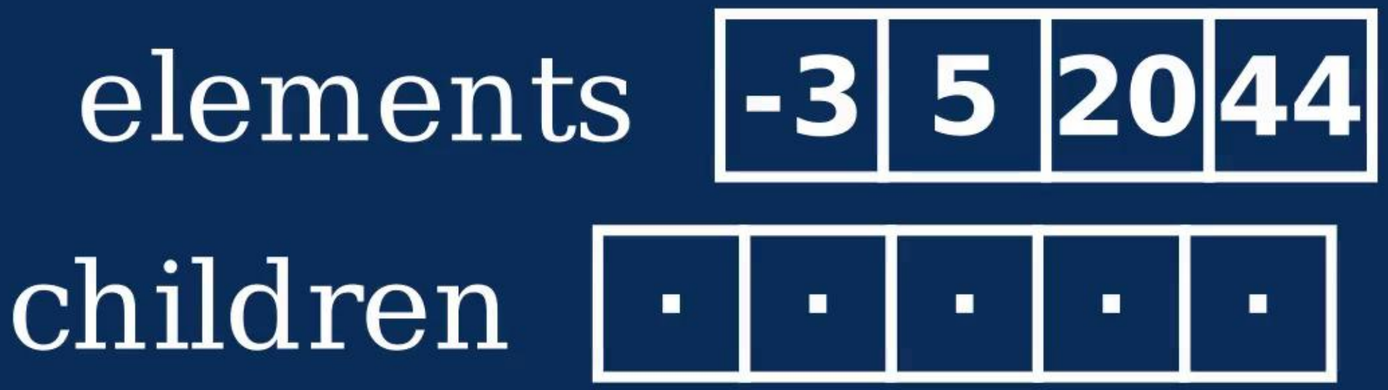
ADT

- Constructor
- Insert
- Find
- Delete

- Constructor
- Insert
- Find
- Delete

B Tree Node of Order m

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



Find

Base Case

- root empty $\Rightarrow$ return
- leaf $\Rightarrow$ do array find
-

Recursive Case

- Array find for first $\geq$ match.
- Recurse to appropriate child

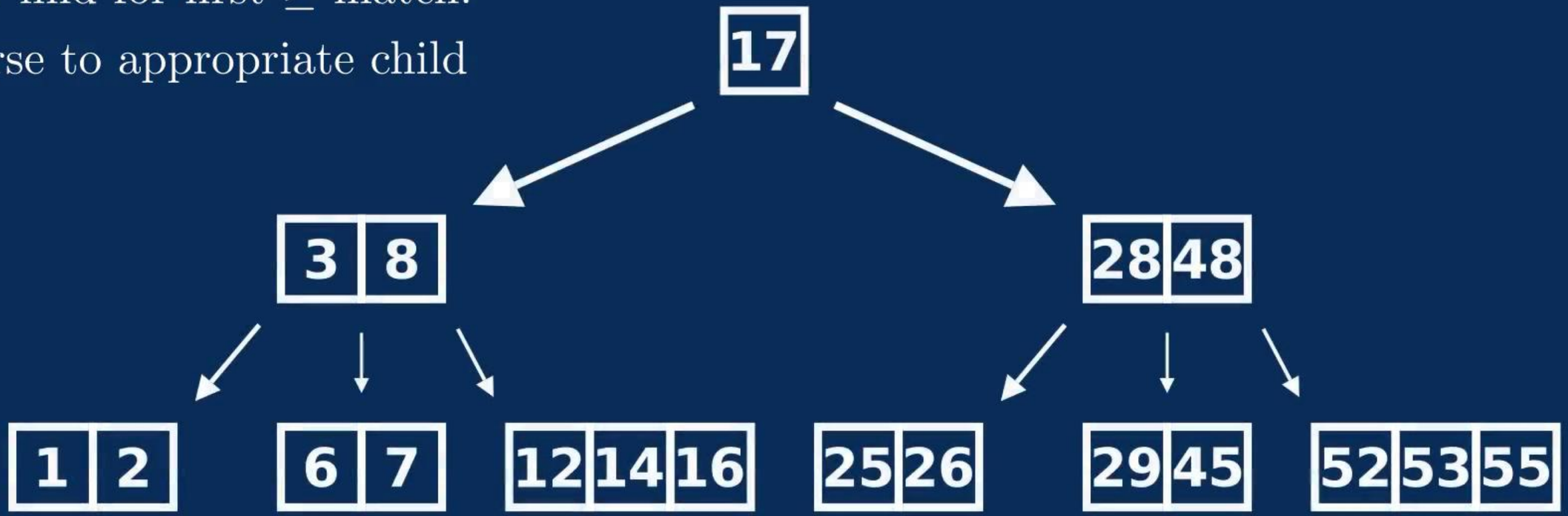
Find

Base Case

- root empty ⇒ return
- leaf ⇒ do array find
-

Recursive Case

- Array find for first ≥ match.
- Recurse to appropriate child



Find

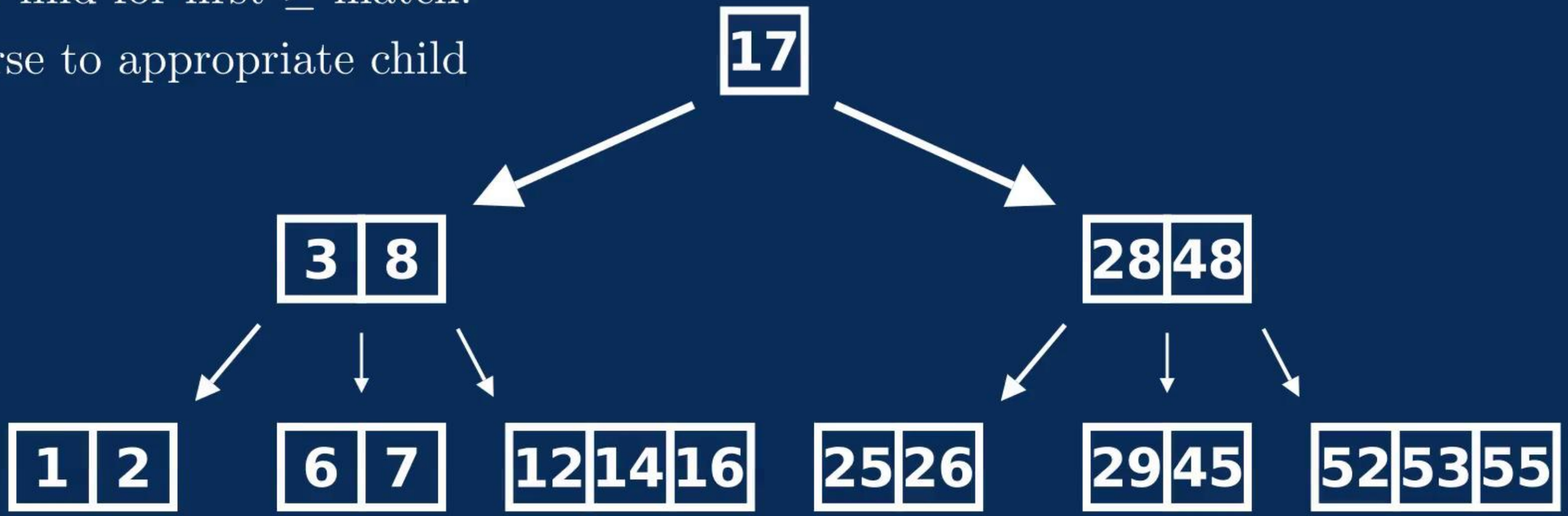
Base Case

- root empty ⇒ return
- leaf ⇒ do array find

Recursive Case

- Array find for first ≥ match.
- Recurse to appropriate child

find(6)



Find

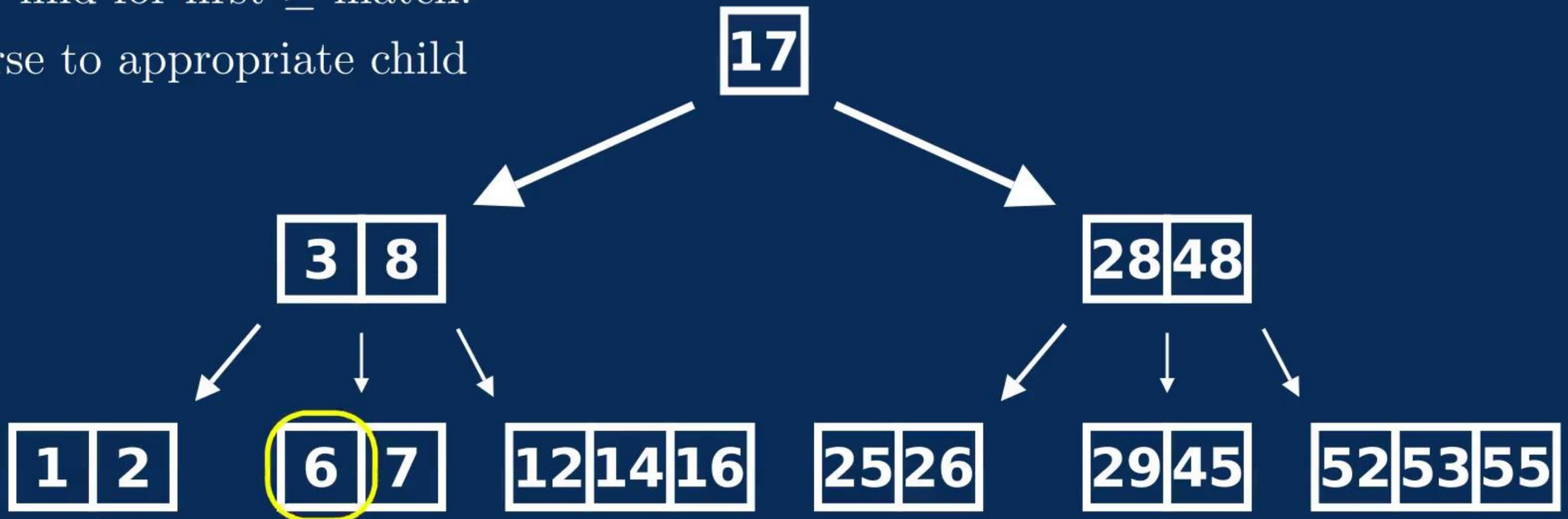
Base Case

- root empty ⇒ return
- leaf ⇒ do array find

Recursive Case

- Array find for first ≥ match.
- Recurse to appropriate child

find(6)



Find

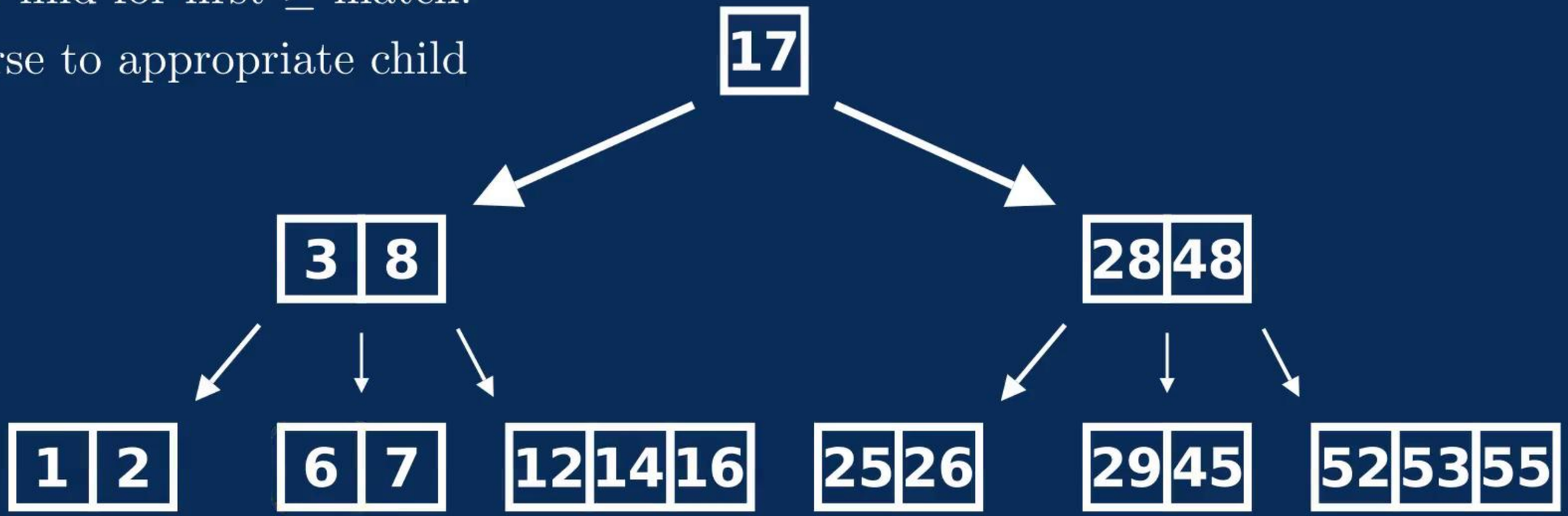
Base Case

- root empty ⇒ return
- leaf ⇒ do array find

Recursive Case

- Array find for first ≥ match.
- Recurse to appropriate child

find(16)



Find

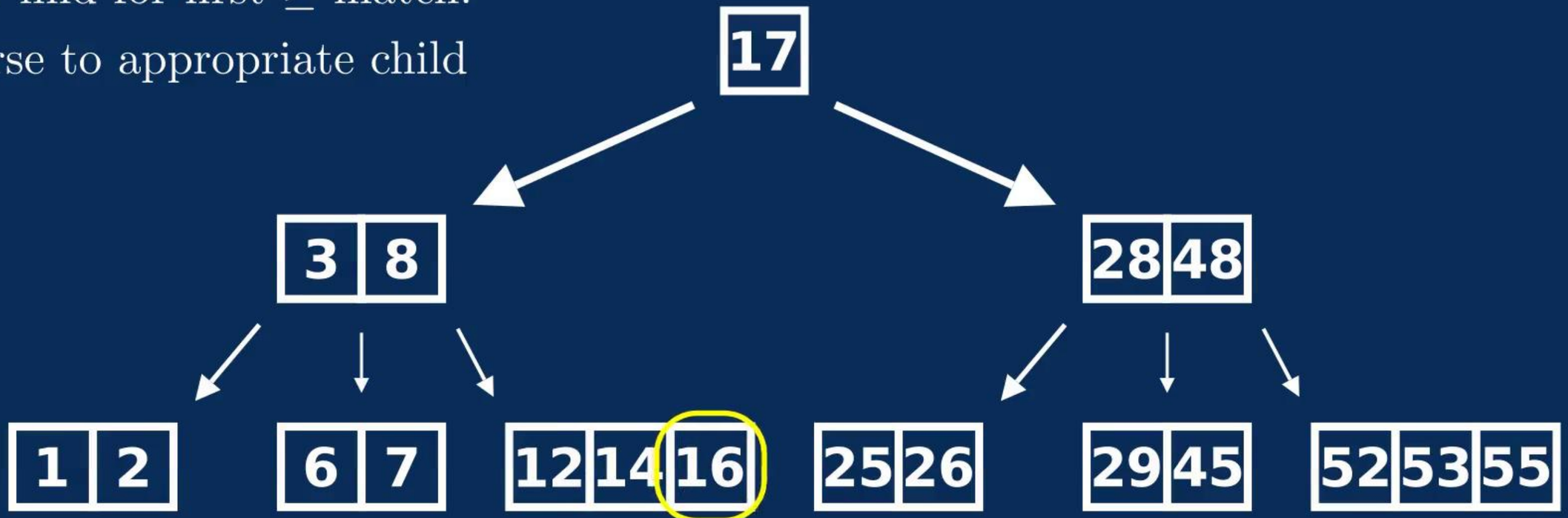
Base Case

- root empty ⇒ return
- leaf ⇒ do array find

Recursive Case

- Array find for first ≥ match.
- Recurse to appropriate child

find(16)



Find

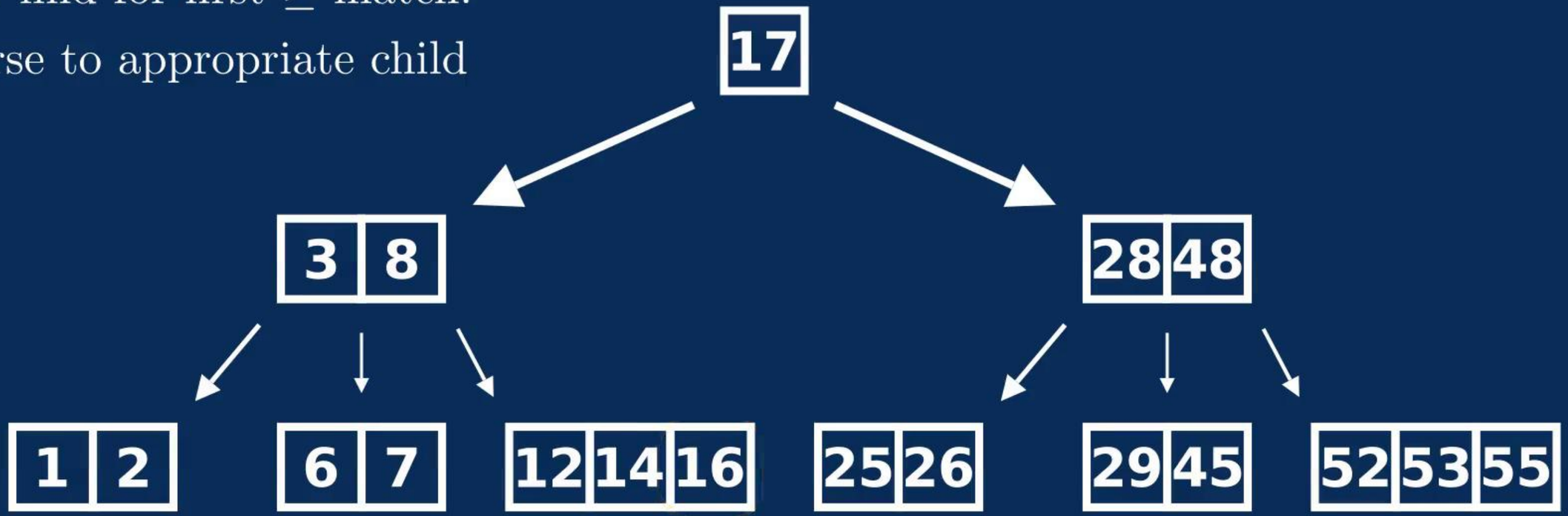
Base Case

- root empty ⇒ return
- leaf ⇒ do array find

Recursive Case

- Array find for first ≥ match.
- Recurse to appropriate child

find(45)



Find

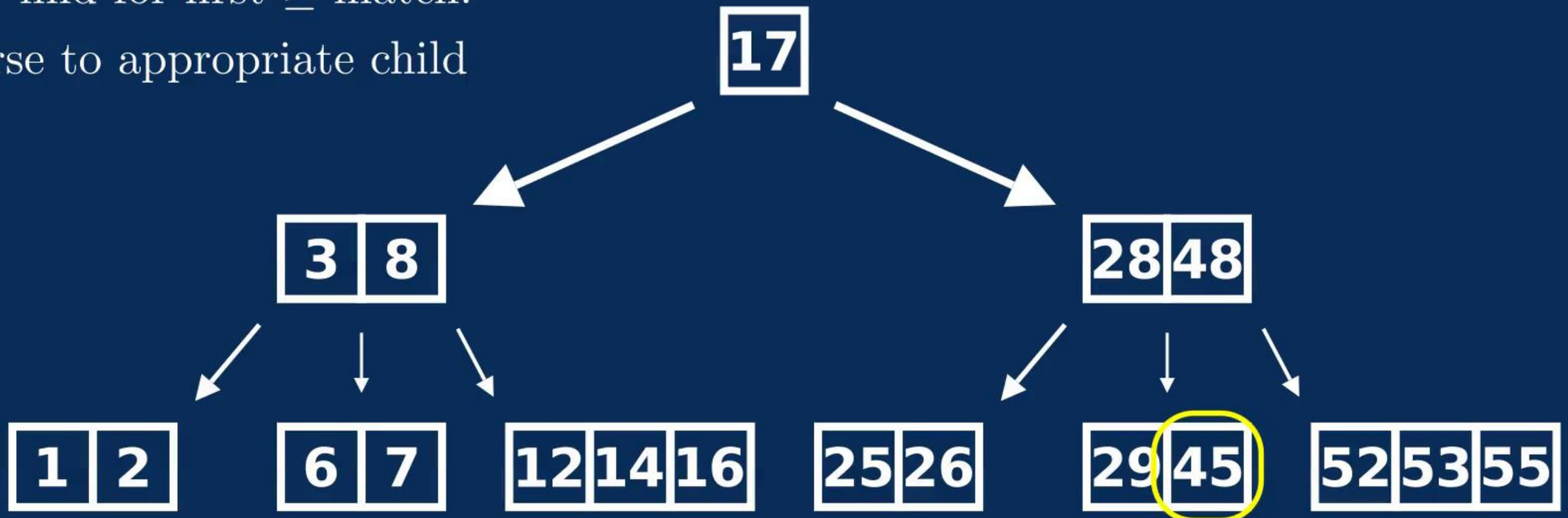
Base Case

- root empty ⇒ return
- leaf ⇒ do array find

Recursive Case

- Array find for first ≥ match.
- Recurse to appropriate child

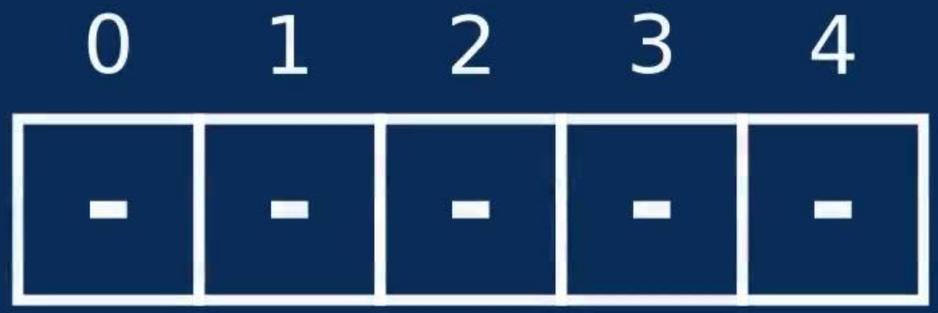
find(45)



Insert

Node insert is simply ordered array insert.

Insert 5, 3, 8, 2, 4



Insert

Node insert is simply ordered array insert.

Insert 5, 3, 8, 2, 4

Insert

Node insert is simply ordered array insert.

Insert 5, 3, 8, 2, 4

Insert

Node insert is simply ordered array insert.

Insert 5, 3, 8, 2, 4

Insert

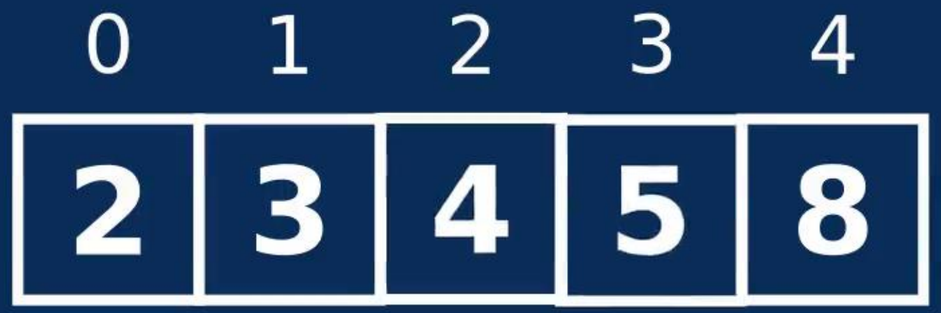
Node insert is simply ordered array insert.

Insert 5, 3, 8, 2, 4

Insert

Node insert is simply ordered array insert.

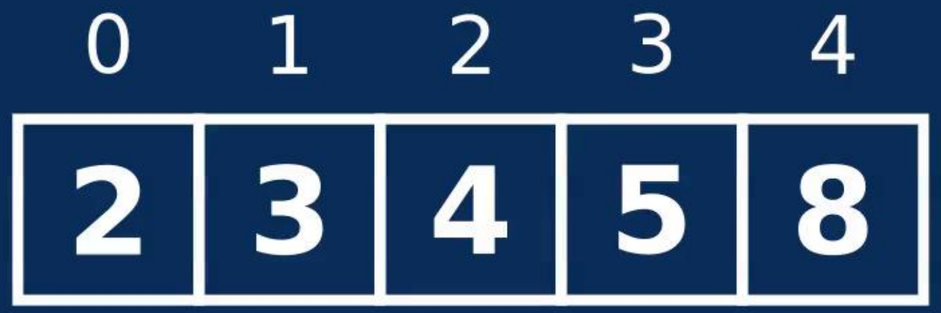
Insert 5, 3, 8, 2, 4



Insert

Node insert is simply ordered array insert.

Insert 5, 3, 8, 2, 4

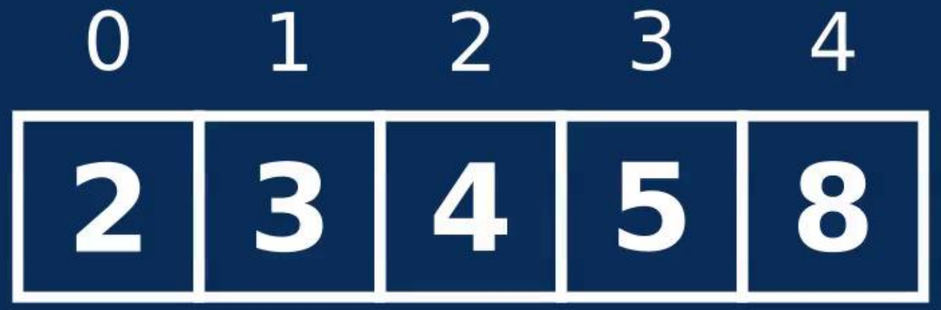


What is the time complexity of sorted array insertion?

Insert

Node insert is simply ordered array insert.

Insert 5, 3, 8, 2, 4



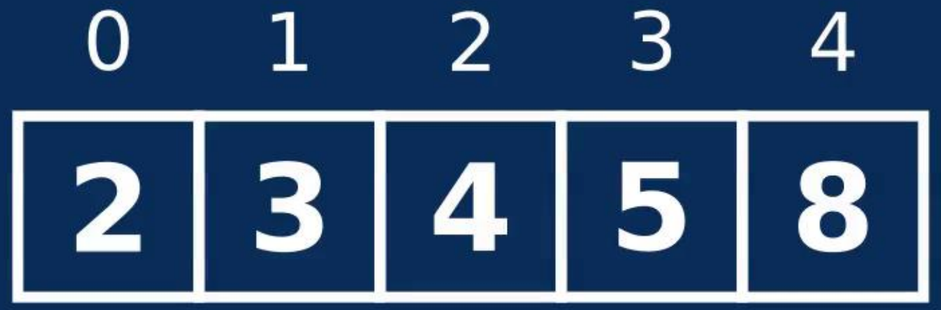
What is the time complexity of sorted array insertion?

It's $\mathcal{O}(n)$. Is this a problem?

Insert

Node insert is simply ordered array insert.

Insert 5, 3, 8, 2, 4



What is the time complexity of sorted array insertion?

It's $\mathcal{O}(n)$. Is this a problem?

No. m is constant, and memory operations are **fast**.

Splitting

What do we do if the node gets full?

Split the node and promote the median.

Let's suppose $m = 5$.

Splitting

What do we do if the node gets full?

Split the node and promote the median.

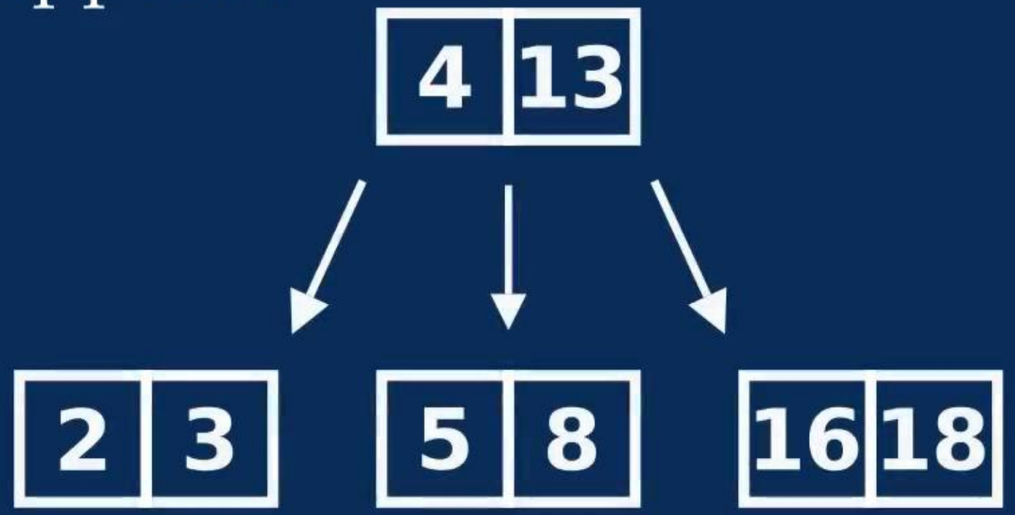
Let's suppose $m = 5$.

Splitting

Splitting can happen recursively.

Let $m = 3$. Insert 25.

Can you predict what happens?

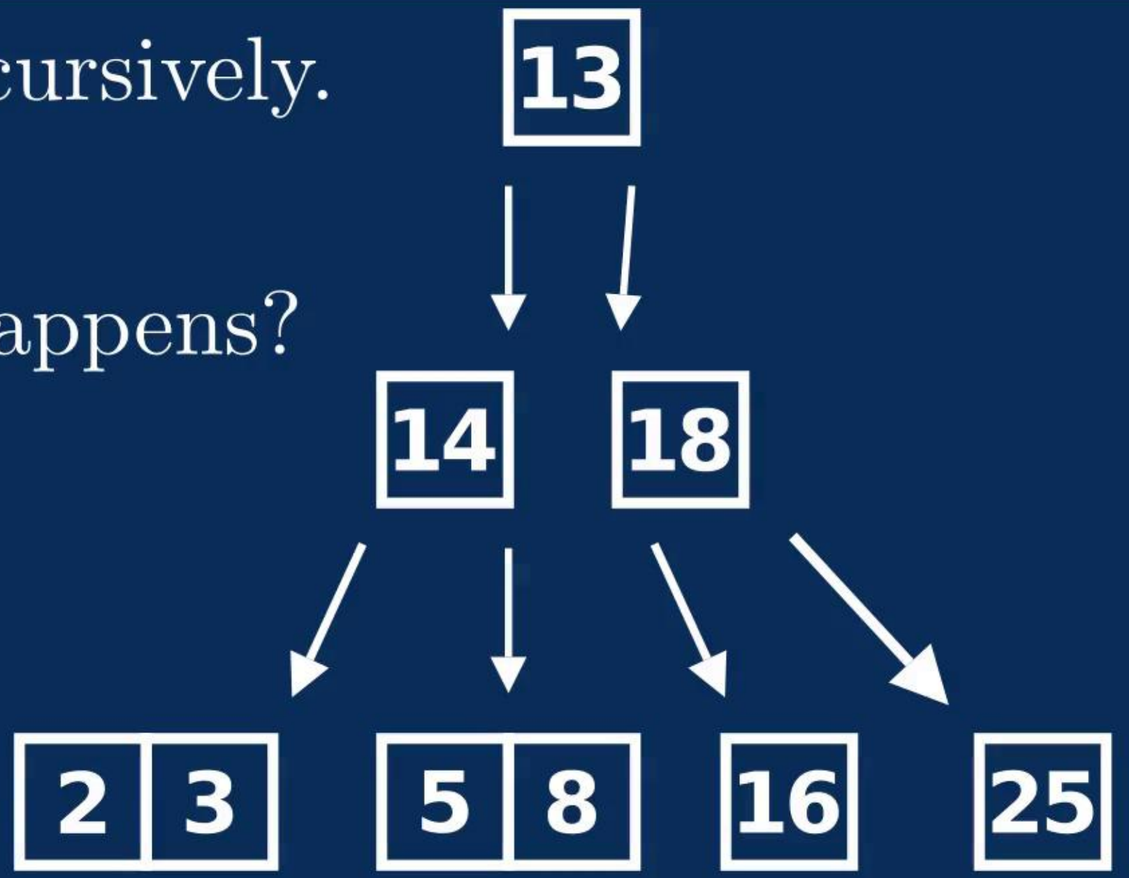


Splitting

Splitting can happen recursively.

Let $m = 3$. Insert 25.

Can you predict what happens?



That's All!

Next time: B Tree delete and analysis