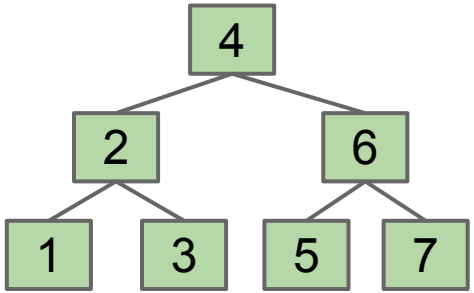
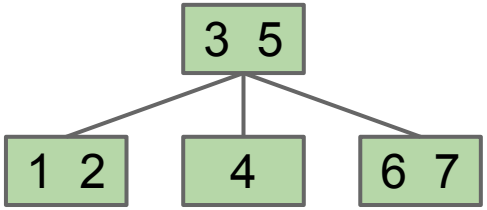




Lecture 17 (Data Structures 3)

B-Trees (and 2-3 and 2-3-4 Trees)

CS61B, Fall 2024 @ UC Berkeley

Slides credit: Josh Hug

BST Height, Big O vs. Worst Case Big Theta

Lecture 17, CS61B, Fall 2024

Binary Search Trees

- **BST Height, Big O vs. Worst Case Big Theta**
- Worst Case Performance

B-Trees

- Splitting Juicy Nodes
- Chain Reaction Splitting
- B-Tree Terminology
- Invariants
- Worst Case Performance

Big Theta vs. Big O

A student a couple weeks ago asked: If we have big theta, why do we also have big O?

The answer is the same as: If we have $=$, why do we also have $\leq$.

Let's dig on this point, and also get a better feel for BSTs as we go.

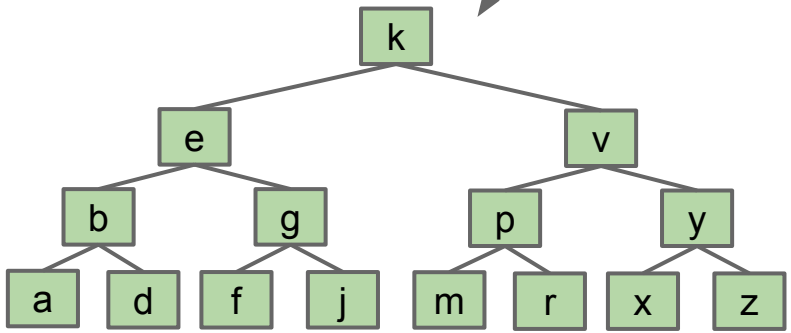
BST Tree Height

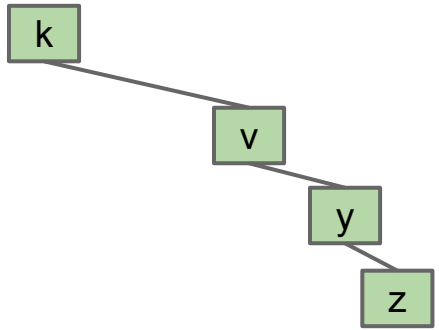
Let's start today by carefully discussing the height of binary search trees.

Trees range from best-case "bushy" to worst-case "spindly".

- Difference is dramatic!

H=3



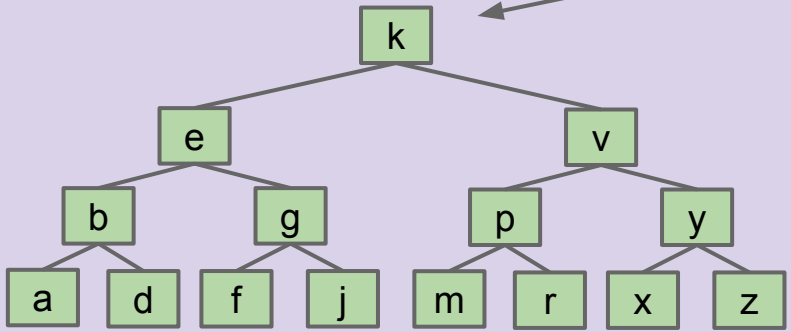


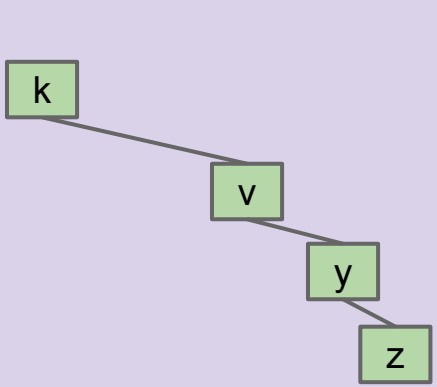
H=3

Tree Height: http://yellkey.com/major

Height varies dramatically between “bushy” and “spindly” trees.

H=3





H=3

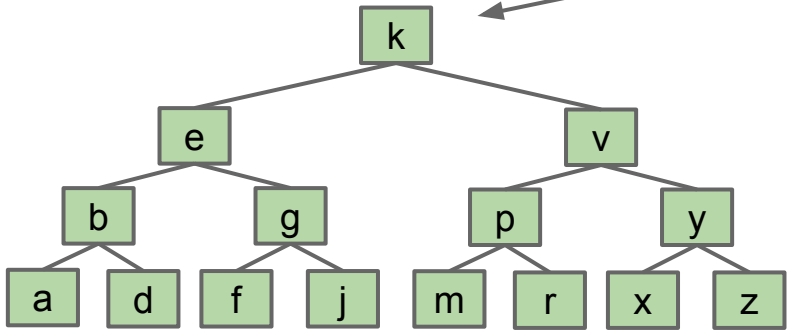
Let H(N) be the height of a tree with N nodes. Give H(N) in Big-Theta notation for “bushy” and “spindly” trees, respectively:

A. Θ(log(N)), Θ(log(N))
B. Θ(log(N)), Θ(N)
C. Θ(N), Θ(log(N))
D. Θ(N), Θ(N)

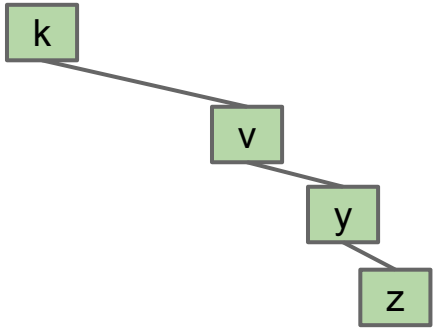
Tree Height

Height varies dramatically between “bushy” and “spindly” trees.

H=3



H = Θ(log(N))



H=3

H = Θ(N)

Performance of operations on spindly trees can be just as bad as a linked list!

- Example: contains(“z”) would take linear time.

Statements about Tree Height: http://yellkey.com/run

Which of these statements are true?

A. Worst case BST height is $\Theta(N)$.

B. BST height is $O(N)$.

C. BST height is $O(N^2)$.

Statements about Tree Height

Which of these statements are true?

A. Worst case BST height is Θ(N). "Is equal to linear."

B. BST height is O(N). "Is less than or equal to linear."

C. BST height is $O(N^2)$. "Is less than or equal to quadratic."

All are **true**!

- A worst case (spindly tree) has a height that grows exactly linearly - Θ(N).
- All BSTs have a height that grows linearly or better - O(N).
- All BSTs have a height that grows quadratically or better - $O(N^2)$.

Statements about Tree Height: http://yellkey.com/economy

Which of these statements is more informative?

A. Worst case BST height is Θ(N).

B. BST height is O(N).

C. They are equally informative.

Statements about Tree Height

Which of these statements is more informative?

A. Worst case BST height is Θ(N).

B. BST height is O(N).

C. They are equally informative.

Saying that the worst case has order of growth N is more informative than saying the height is O(N).

Let's see an analogy.

Question: http://yellkey.com/keep

Which statement gives you more information about a hotel?

A. The most expensive room in the hotel is $639 per night.

B. Every room in the hotel is less than or equal to $639 per night.

Question

Which statement gives you more information about a hotel?

A. The most expensive room in the hotel is $639 per night.

B. Every room in the hotel is less than or equal to $639 per night.

Most expensive room: $639/nt

THE RITZ-CARLTON
LAKE TAHOE

All rooms <= $639/nt

THE RITZ-CARLTON
LAKE TAHOE

(A nice place to stay!)

BST Height

BST height is all four of these:

- O(N).
- Θ(log N) in the best case (“bushy”).
- Θ(N) in the worst case (“spindly”).
- $O(N^2)$.

The middle two statements are more informative.

- Big O is NOT mathematically the same thing as “worst case”.
  - e.g. BST heights are $O(N^2)$, but are not quadratic in the worst case.
  - … but Big O often used as shorthand for “worst case”.

The Usefulness of Big O

Big O is still a useful idea:

- Allows us to make simple blanket statements, e.g. can just say “binary search is O(log N)” instead of “binary search is Θ(log N) in the worst case”.
- Sometimes don’t know the exact runtime, so use O to give an upper bound.
  - Example: Runtime for finding shortest route that goes to all world cities is $O(2^N)^*$. There might be a faster way, but nobody knows one yet.
- Easier to write proofs for Big O than Big Theta, e.g. finding runtime of mergesort, you can round up the number of items to the next power of 2 (see A level study guide problems for Asymptotics2 lecture). A little beyond the scope of our course.

*: Under certain assumptions and constraints not listed.

BST Height

BST height is both of these:

- Θ(log N) in the best case (“bushy”).
- Θ(N) in the worst case (“spindly”).

Let’s now turn to understanding the performance of BST operations.

Worst Case Performance

Lecture 17, CS61B, Fall 2024

Binary Search Trees

- BST Height, Big O vs. Worst Case Big Theta
- **Worst Case Performance**

B-Trees

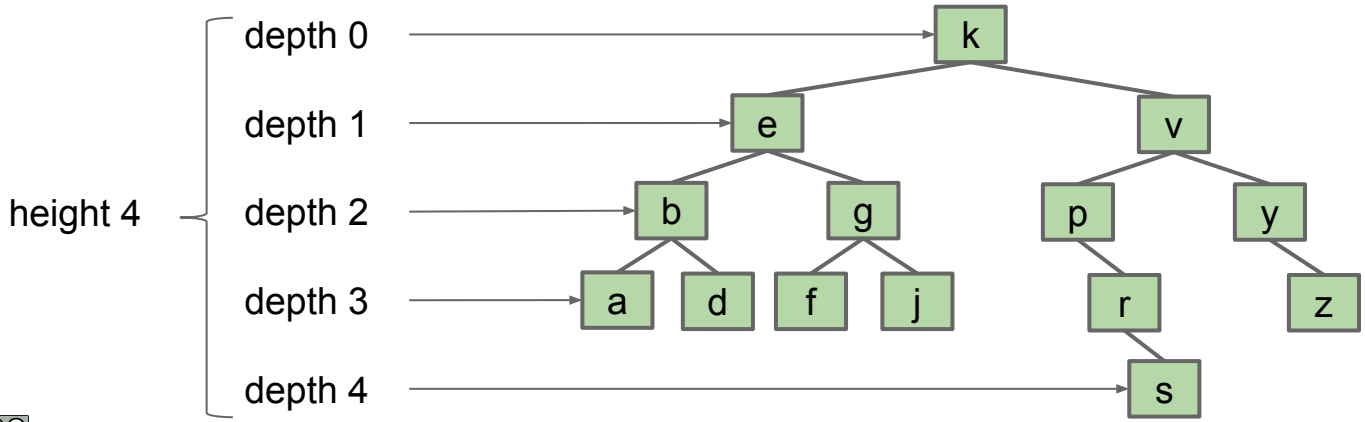
- Splitting Juicy Nodes
- Chain Reaction Splitting
- B-Tree Terminology
- Invariants
- Worst Case Performance

Height and Depth

Height and average depth are important properties of BSTs.

- The **"depth" of a node** is how far it is from the root, e.g. depth(g) = 2.

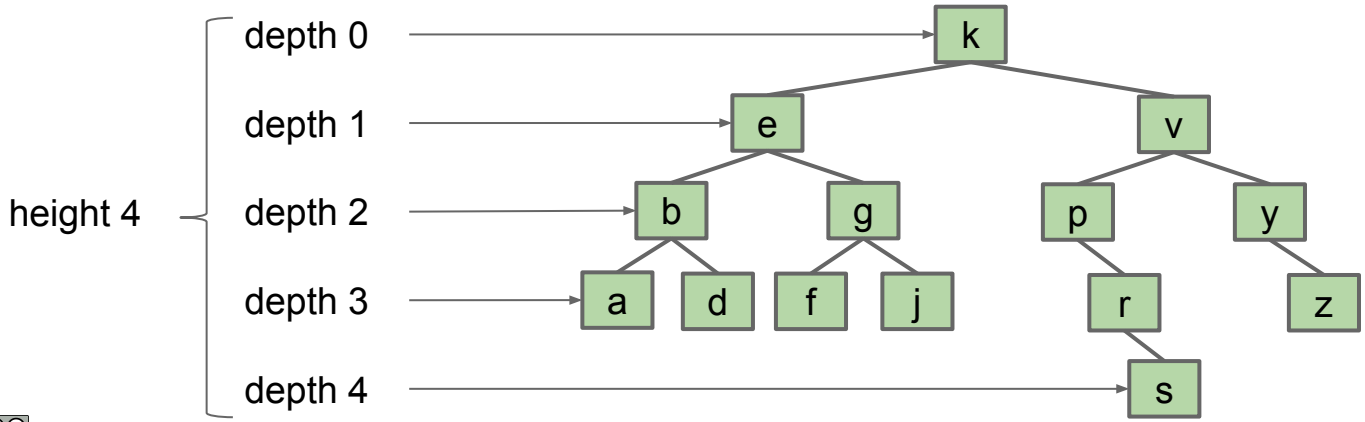
- The **"height" of a tree** is the depth of its deepest leaf, e.g. height(T) = 4.
- The **"average depth"** of a tree is the average depth of a tree's nodes.
  - (**0**x1 + **1**x2 + **2**x4 + **3**x6 + **4**x1)/(1+2+4+6+1) = 2.35



Height, Depth and Runtime

Height and average depth determine runtimes for BST operations.

- The **“height” of a tree** determines the worst case runtime to find a node.
  - Example: Worst case is contains(s), requires 5 comparisons (height + 1).
- The **“average depth”** determines the average case runtime to find a node.
  - Example: Average case is 3.35 comparisons (average depth + 1).

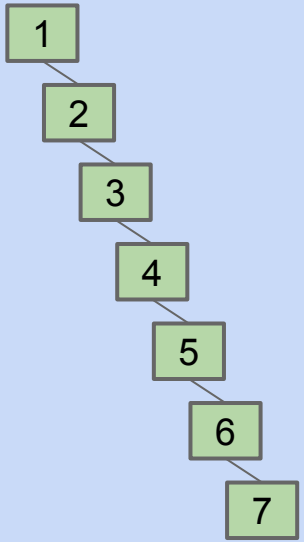


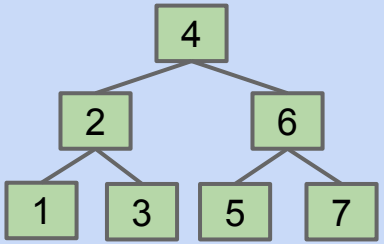
BSTs in Practice

Suppose we want to build a BST out of the numbers 1, 2, 3, 4, 5, 6, 7.

Give an example of a sequence of add operations that results in:

- A spindly tree.
- A bushy tree.

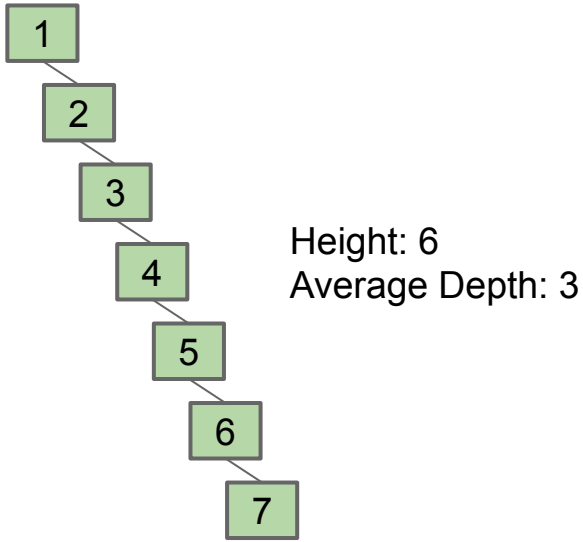




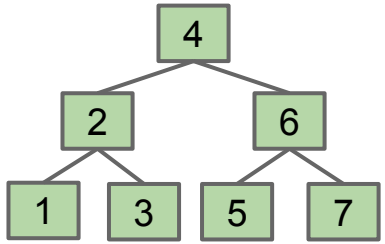
BSTs in Practice

Give an example of a sequence of add operations that results in:

- A spindly tree.
  - add(1), add(2), add(3), add(4), add(5), add(6), add(7)
- A bushy tree.
  - add(4), add(2), add(1), add(3), add(6), add(5), add(7)



Height: 6
Average Depth: 3



Height: 2
Average Depth: 1.43

Important Question: What about Real World BSTs?

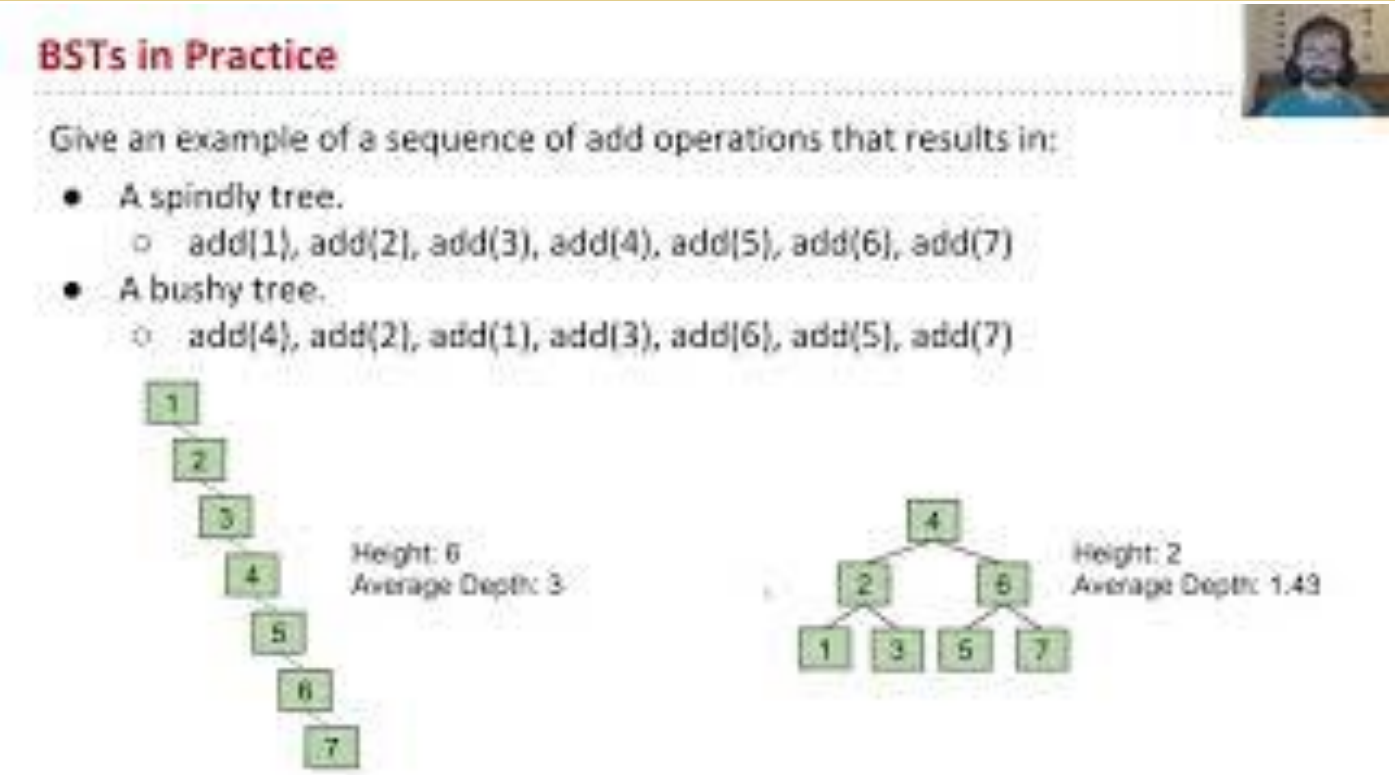
BSTs have:

- Worst case Θ(N) height.
- Best case Θ(log N) height.

… but what about trees that you'd build during real world applications?

One way to approximate this is to consider randomized BSTs.

Simulation: Trees Built from Random Inserts



Nice Property. Random trees have Θ(log N) average depth and height.

- In other words: Random trees are bushy, not spindly.

Video courtesy of Kevin Wayne (Princeton University)

Randomized Trees: Mathematical Analysis

Average Depth. If N distinct keys are inserted into a BST, the expected average depth is ~ 2 ln N = Θ(log N).

- Thus, average runtime for contains operation is Θ(log N) on a tree built with random inserts.
- Will discuss this proof briefly closer to the end of this course.

Tree Height. If N distinct keys are inserted in random order, expected tree height is ~ 4.311 ln N (see Reed, 2003).

- Thus, worst case runtime for contains operation is Θ(log N) on a tree built with random inserts.
- Proof is well beyond the scope of the course (and is 27 pages long!).

Note: ~ is the same thing as Big Theta, but you don't drop the multiplicative constants.

Important Question: What about Real World BSTs?

BSTs have:

- Worst case Θ(N) height.
- Best case Θ(log N) height.
- Θ(log N) height if constructed via random inserts.

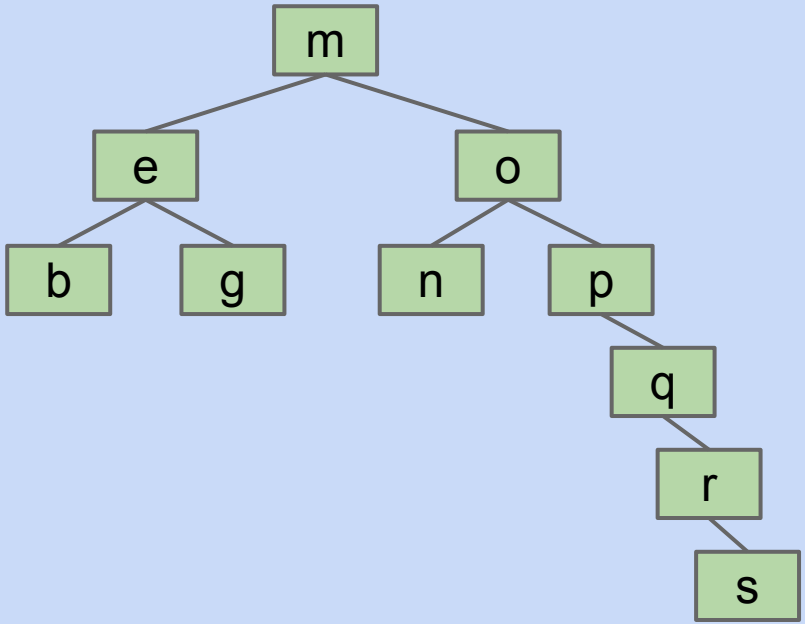
In real world applications we expect both insertion and deletion.

- See (IMO really interesting!) extra slides for more on simulations of trees including deletion.
- Can show that random trees including deletion are still Θ(log N) height (if you randomly pick between predecessor and successor when deleting).

Good News and Bad News

Good news: BSTs have great performance if we insert items randomly.
Performance is Θ(log N) per operation.

Bad News: We can't always insert our items in a random order. Why?

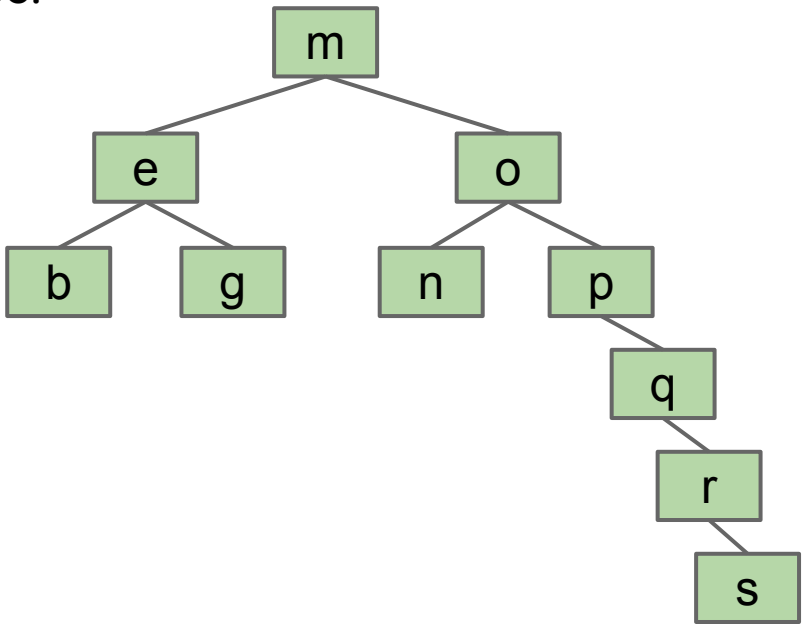


Good News and Bad News

Good news: BSTs have great performance if we insert items randomly.
Performance is Θ(log N) per operation.

Bad News: We can't always insert our items in a random order. Why?

- Data comes in over time, don't have all at once.
  - Example: Storing dates of events.
    - add("01-Jan-2019, 10:31:00")
    - add("01-Jan-2019, 18:51:00")
    - add("02-Jan-2019, 00:05:00")
    - add("02-Jan-2019, 23:10:00")



In this lecture, we'll do something totally different.

Splitting Juicy Nodes

Lecture 17, CS61B, Fall 2024

Binary Search Trees

- BST Height, Big O vs. Worst Case Big Theta
- Worst Case Performance

B-Trees

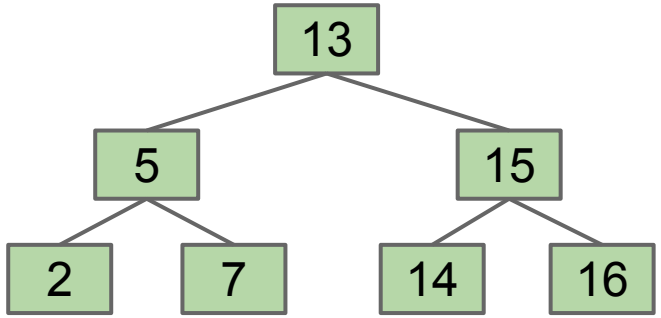
- **Splitting Juicy Nodes**
- Chain Reaction Splitting
- B-Tree Terminology
- Invariants
- Worst Case Performance

Avoiding Imbalance

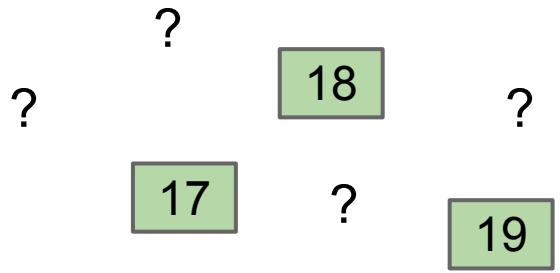
The problem is adding new leaves at the bottom.

Crazy idea: Never add new leaves at the bottom.

- Tree can never get imbalanced.



Q: What do we do with incoming keys?

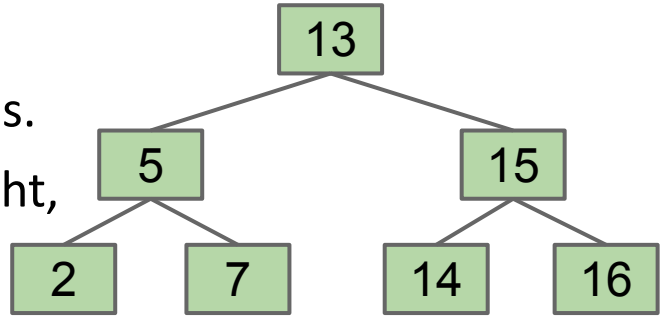


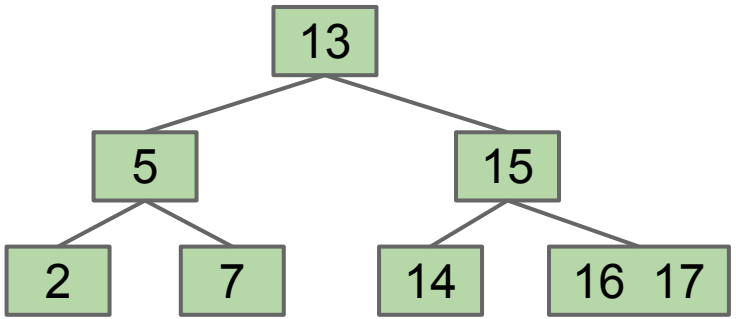
Avoiding Imbalance through Overstuffing

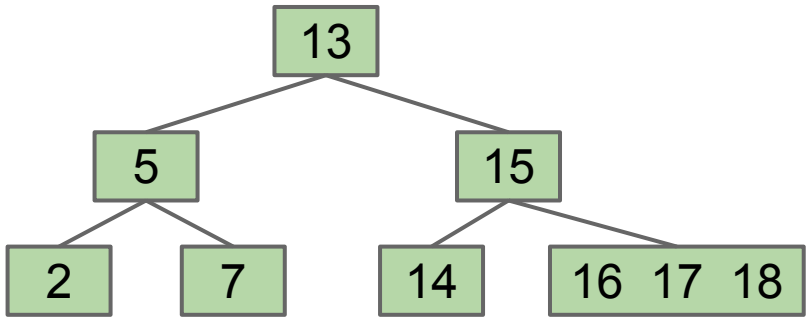
The problem is adding new leaves at the bottom.

Avoid new leaves by “overstuffing” the leaf nodes.

- “Overstuffed tree” always has balanced height, because leaf depths never change.
  - Height is just max(depth).





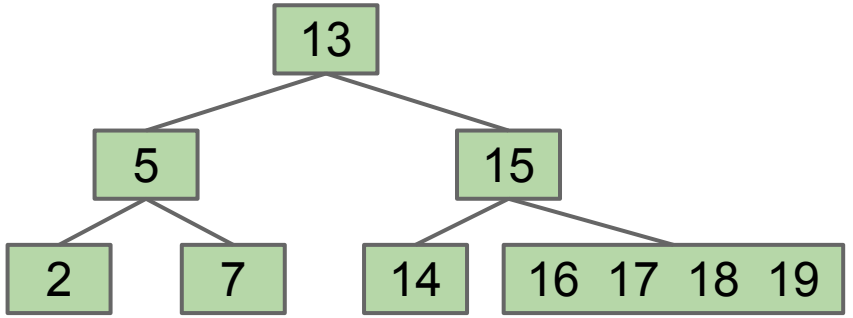


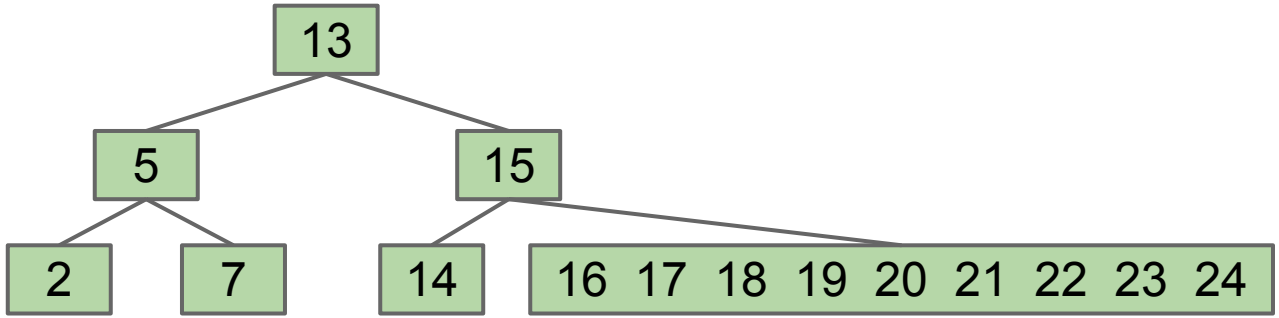
Avoiding Imbalance through Overstuffing

Overstuffed trees are a logically consistent but very weird data structure.

- contains(18):
  - Is 18 > 13? Yes, go right.
  - Is 18 > 15? Yes, go right.
  - Is 16 = 18? No.
  - Is 17 = 18? No.
  - Is 18 = 18? Yes! Found it.

Q: What is the problem with this idea?

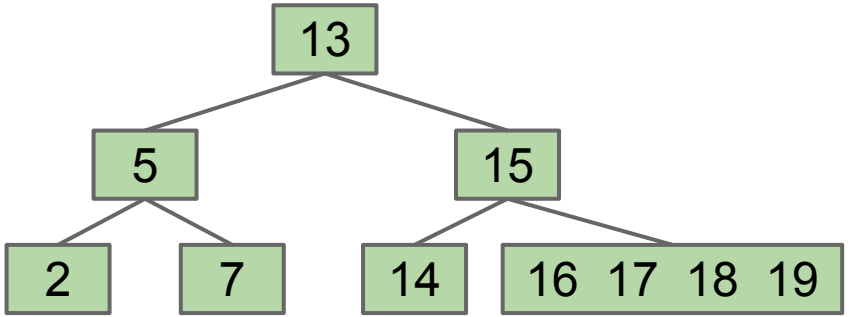




Revising Our Overstuffed Tree Approach

Height is balanced, but we have a new problem:

- Leaf nodes can get too juicy.

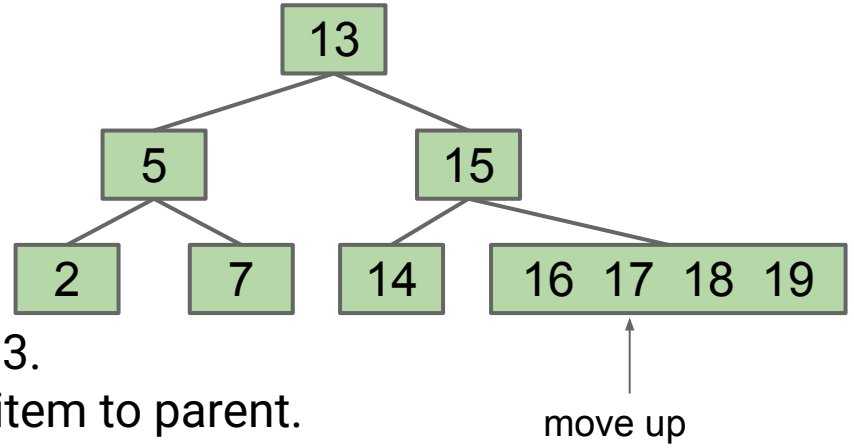


Solution?

Revising Our Overstuffed Tree Approach: Moving Items Up

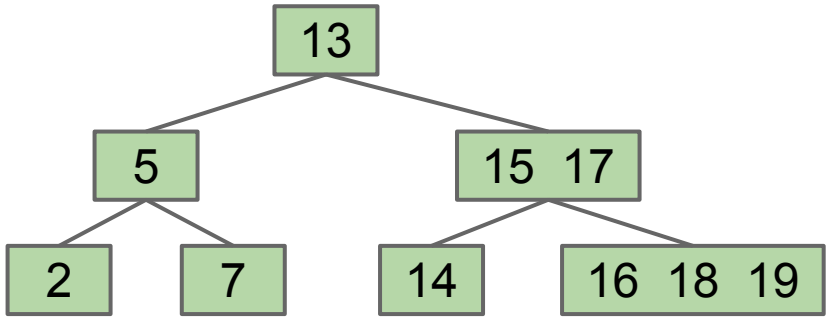
Height is balanced, but we have a new problem:

- Leaf nodes can get too juicy.



Solution?

- Set a limit L on the number of items, say L=3.
- If any node has more than L items, give an item to parent.
  - Which one? Let's say (arbitrarily) the left-middle.



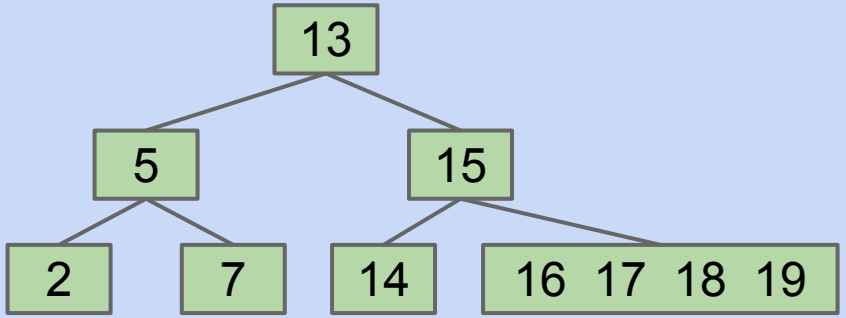
Q: What's the problem now?

- 16 is to the right of 17.

Revising Our Overstuffed Tree Approach

Height is balanced, but we have a new problem:

- Leaf nodes can get too juicy.

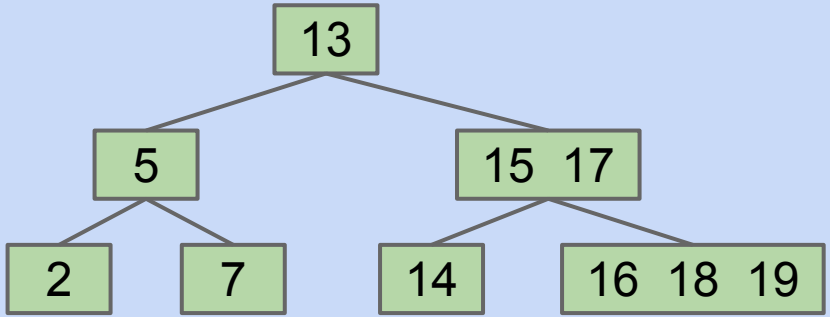


Solution?

- Set a limit L on the number of items, say L=3.
- If any node has more than L items, give an item to parent.
  - Which one? Let's say (arbitrarily) the left-middle.

Challenge for you:

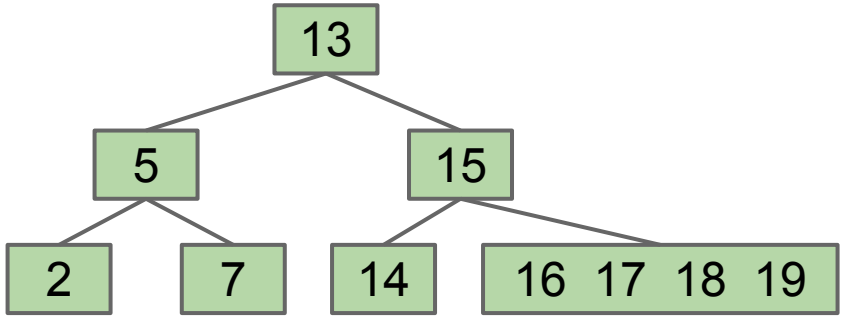
- How can we tweak this idea to make it work?



Revising Our Overstuffed Tree Approach: Node Splitting

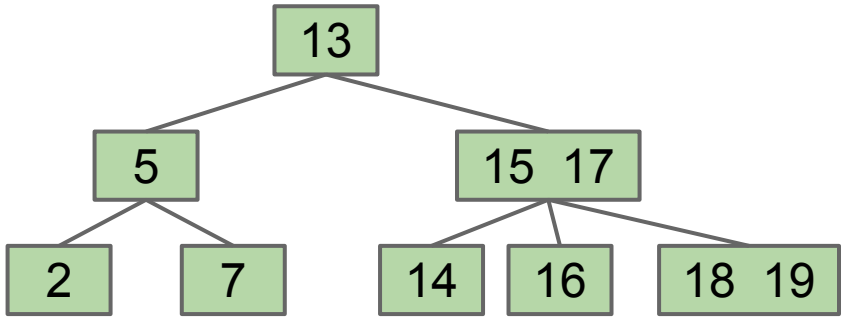
Height is balanced, but we have a new problem:

- Leaf nodes can get too juicy.



Solution?

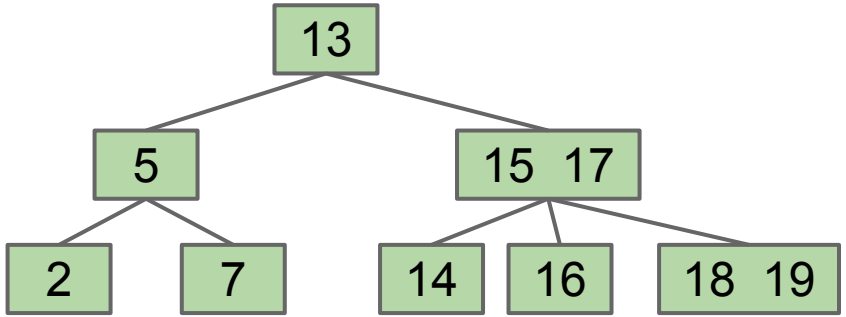
- Set a limit L on the number of items, say L=3.
- If any node has more than L items, give an item to parent.
  - Pulling item out of full node splits it into left and right.
  - Parent node now has three children!



Revising Our Overstuffed Tree Approach: Node Splitting

This is a logically consistent and not so weird data structure.

- contains(18):
  - 18 > 13, so go right
  - 18 > 15, so compare vs. 17
  - 18 > 17, so go right



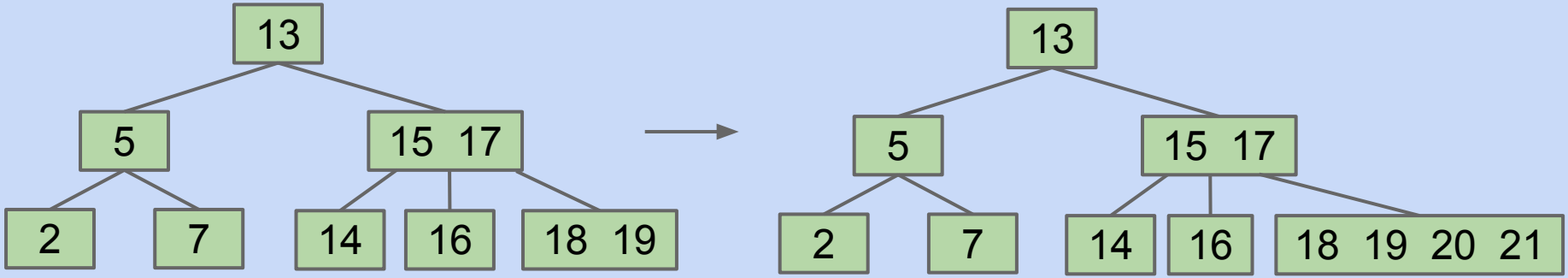
Examining a node costs us O(L) compares, but that's OK since L is constant.

What if a non-leaf node gets too full? Can we split that?

- Sure, we'll do this in a few slides, but first...

add Understanding Check

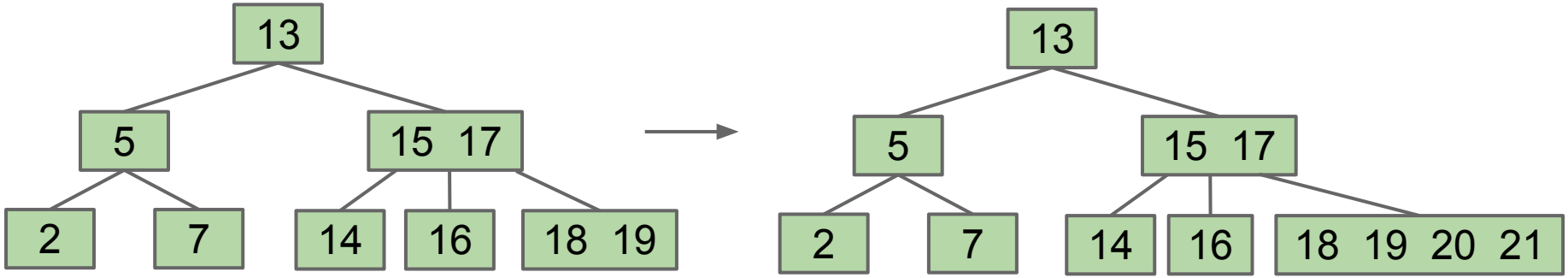
- Suppose we add 20, 21:



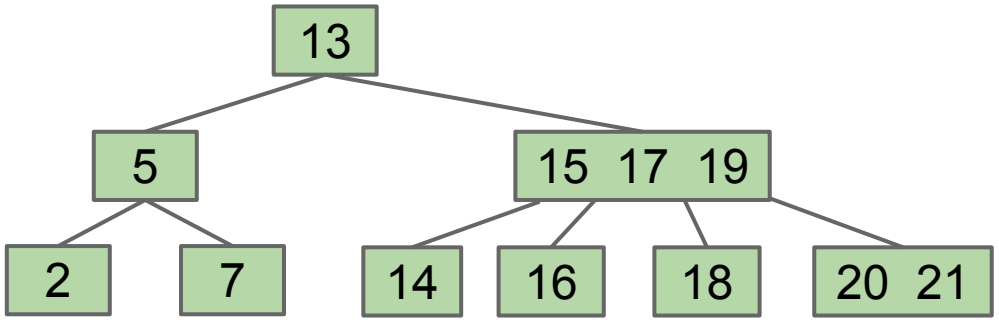
- Q: If our cap is at most L=3 items per node, draw post-split tree:

add Understanding Check

- Suppose we add 20, 21:



- Q: If our cap is at most L=3 items per node, draw post-split tree:



Chain Reaction Splitting

Lecture 17, CS61B, Fall 2024

Binary Search Trees

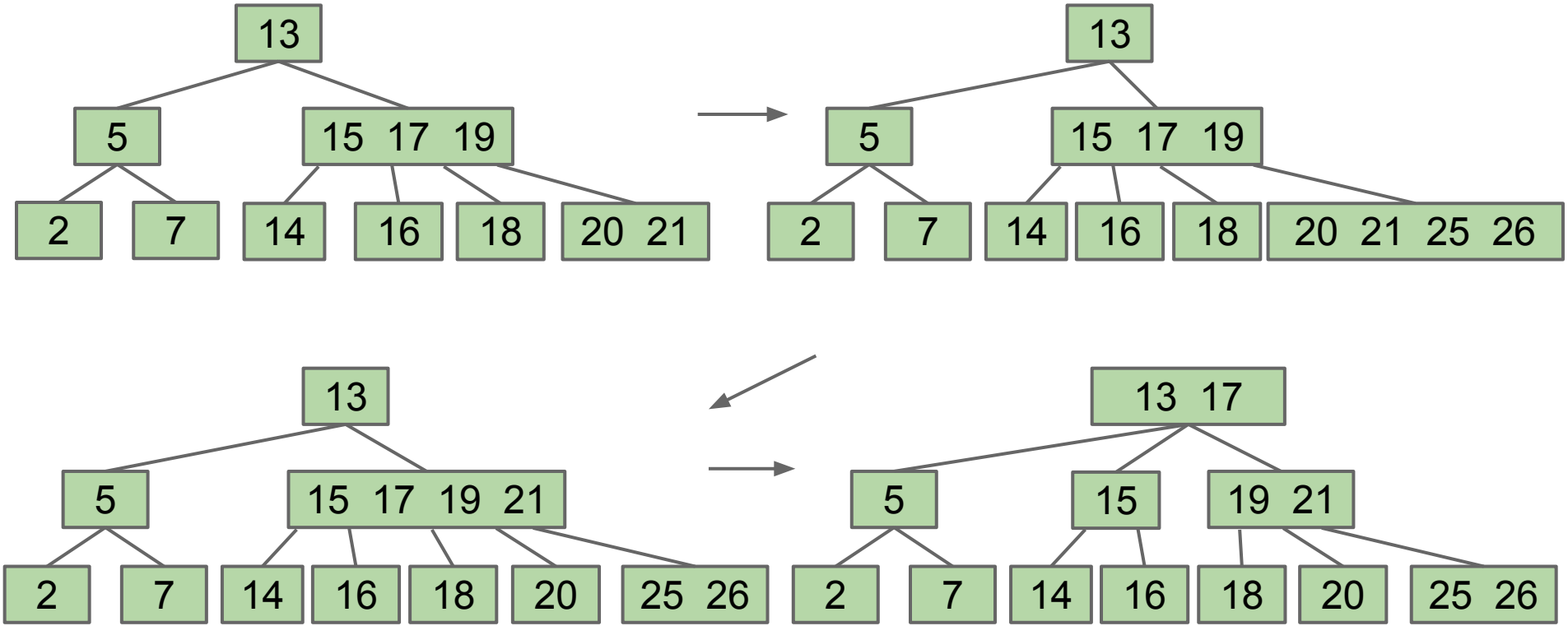
- BST Height, Big O vs. Worst Case Big Theta
- Worst Case Performance

B-Trees

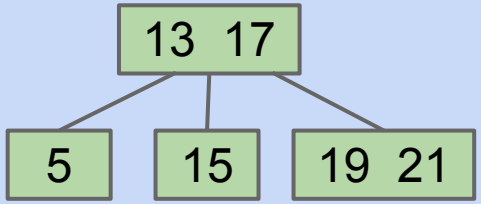
- Splitting Juicy Nodes
- **Chain Reaction Splitting**
- B-Tree Terminology
- Invariants
- Worst Case Performance

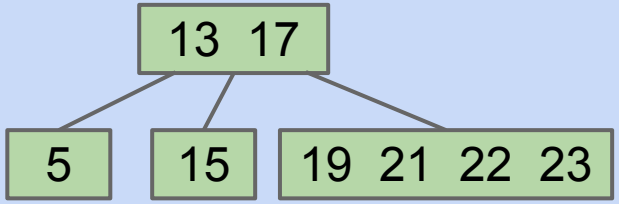
add: Chain Reaction

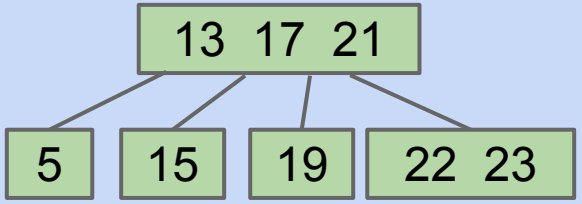
Suppose we add 25, 26:

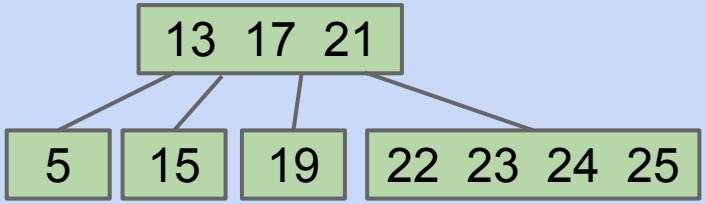


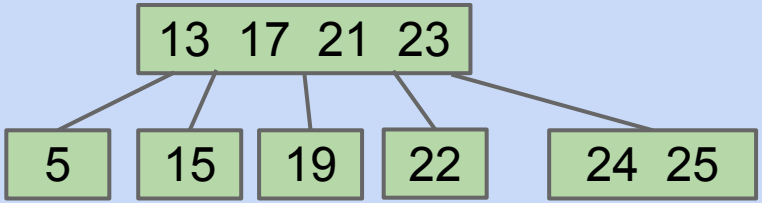
What Happens If The Root Is Too Full?





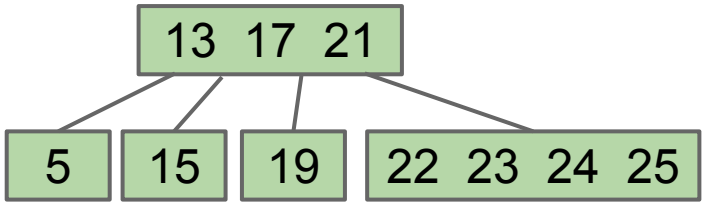


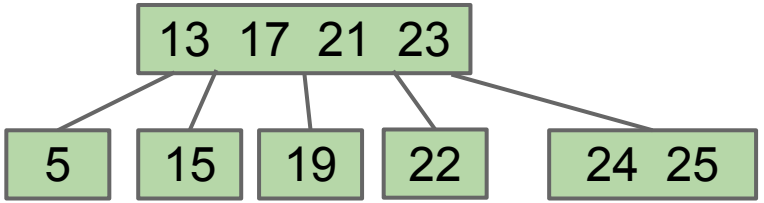




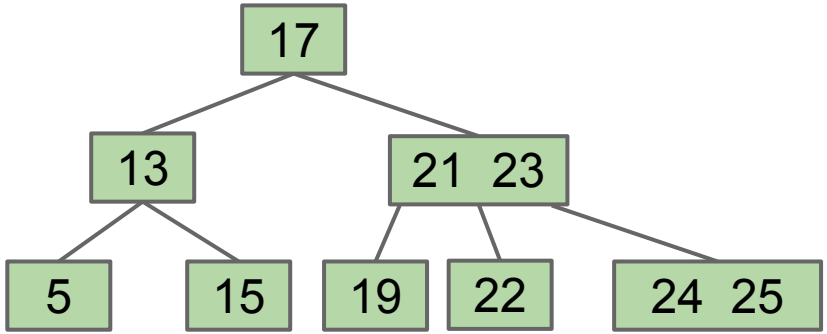
Challenge: Draw the tree after the root is split.

What Happens If The Root Is Too Full?





Challenge: Draw the tree after the root is split.



B-Tree Terminology

Lecture 17, CS61B, Fall 2024

Binary Search Trees

- BST Height, Big O vs. Worst Case Big Theta
- Worst Case Performance

B-Trees

- Splitting Juicy Nodes
- Chain Reaction Splitting
- **B-Tree Terminology**
- Invariants
- Worst Case Performance

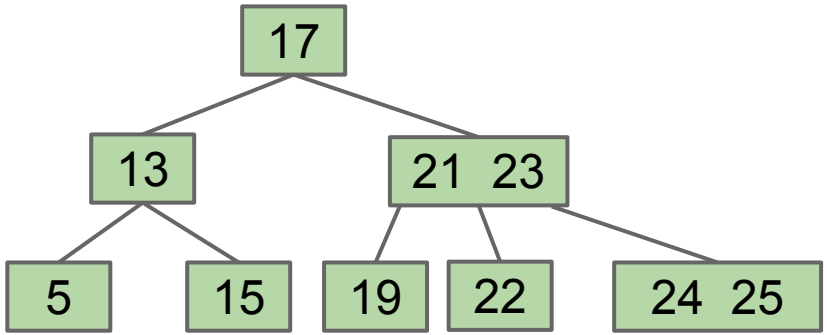
Perfect Balance

Observation: Splitting-trees have perfect balance.

- If we split the root, every node gets pushed down by exactly one level.
- If we split a leaf node or internal node, the height doesn't change.

We will soon prove: All operations have guaranteed O(log N) time.

- More details soon.



The Real Name for Splitting Trees is “B Trees”

Splitting tree is a better name, but I didn’t invent them, so we’re stuck with their real name: **B-trees**.

- B-trees of order L=3 (like we used today) are also called a 2-3-4 tree or a 2-4 tree.
  - “2-3-4” refers to the number of children that a node can have, e.g. a 2-3-4 tree node may have 2, 3, or 4 children.
- B-trees of order L=2 are also called a 2-3 tree.

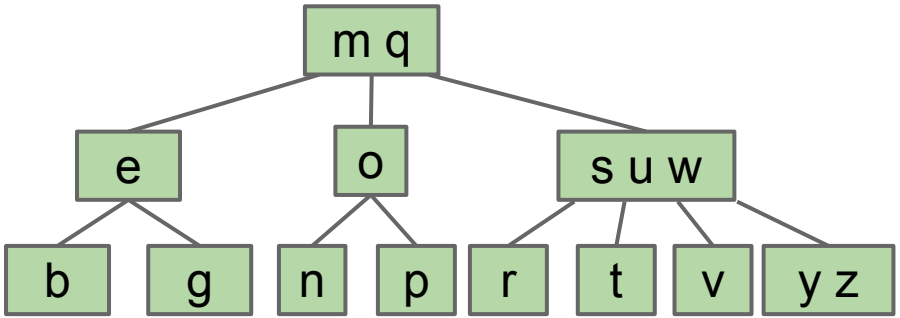
The origin of "B-tree" has never been explained by the authors. As we shall see, "balanced," "broad," or "bushy" might apply. Others suggest that the "B" stands for Boeing. Because of his contributions, however, it seems appropriate to think of B-trees as "Bayer"-trees.

- Douglas Corner (The Ubiquitous B-Tree)

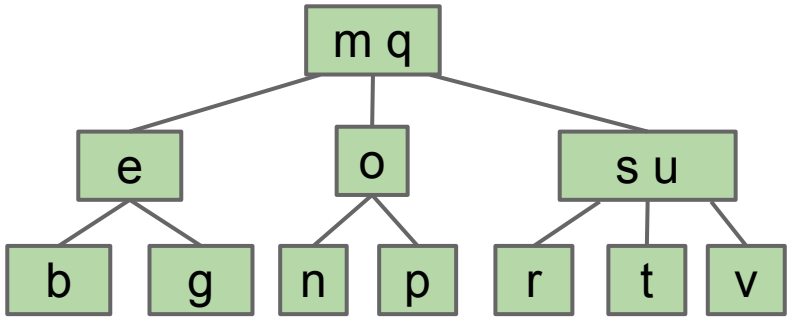
A note on Terminology

B-Trees are most popular in two specific contexts:

- Small L (L=2 or L=3):
  - Used as a conceptually simple balanced search tree (as today).
- L is very large (say thousands).
  - Used in practice for databases and filesystems (i.e. systems with very large records).



2-3-4 a.k.a. 2-4 Tree (L=3):
Max 3 items per node.
Max 4 non-null children per node.



2-3 Tree (L=2):
Max 2 items per node.
Max 3 non-null children per node.

Invariants

Lecture 17, CS61B, Fall 2024

Binary Search Trees

- BST Height, Big O vs. Worst Case Big Theta
- Worst Case Performance

B-Trees

- Splitting Juicy Nodes
- Chain Reaction Splitting
- B-Tree Terminology
- **Invariants**
- Worst Case Performance

Exercise: Recorded Video Viewers Only

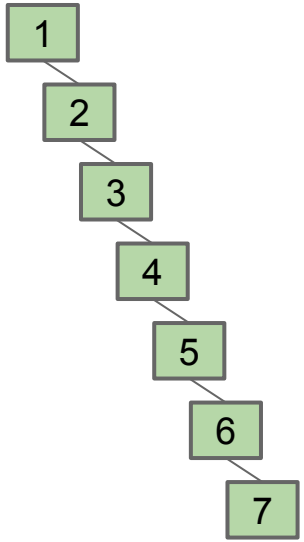
Add the numbers 1, 2, 3, 4, 5, 6, then 7 (in that order) into a regular BST.

Then try adding 1, 2, 3, 4, 5, 6, then 7 (in that order) into a 2-3 tree (L=2).

- For L=2, pass the middle item up.

Exercise

Add the numbers 1, 2, 3, 4, 5, 6, then 7 (in that order) into a regular BST.



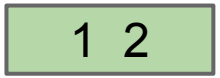
Resulting BST is spindly.

Exercise

Add 1, 2, 3, 4, 5, 6, 7 (in that order) into a 2-3 tree. For L=2, pass the middle item up.

Adding 1.

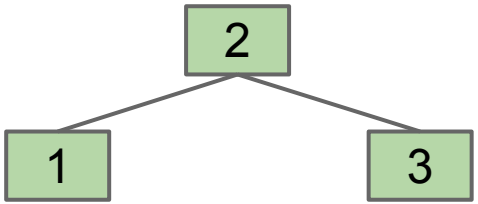
Adding 2.



Adding 3. Node is too full, so we need to split.

Before split

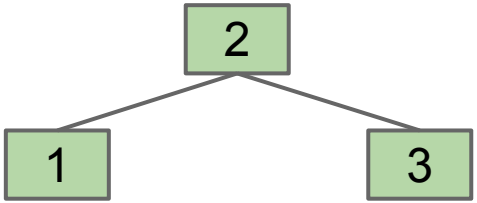
After split



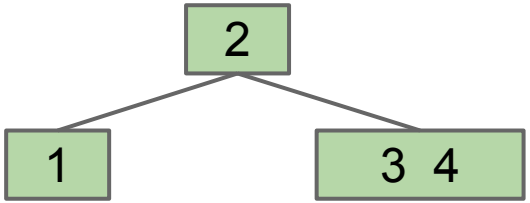
Exercise

Add 1, 2, 3, 4, 5, 6, 7 (in that order) into a 2-3 tree. For L=2, pass the middle item up.

After adding 3.

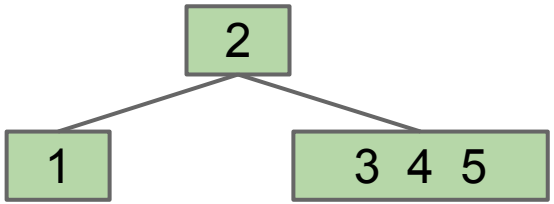


Adding 4.

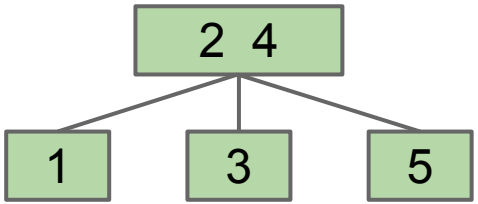


Adding 5. Node is too full, so we need to split.

Before split



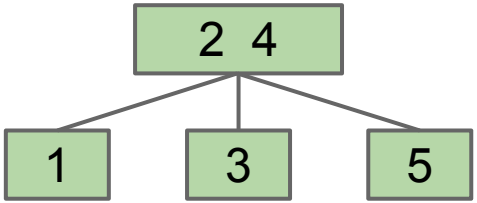
After split



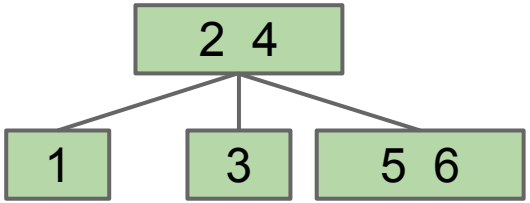
Exercise

Add 1, 2, 3, 4, 5, 6, 7 (in that order) into a 2-3 tree. For L=2, pass the middle item up.

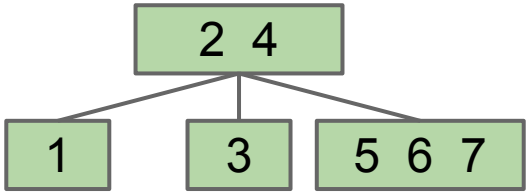
After adding 5.



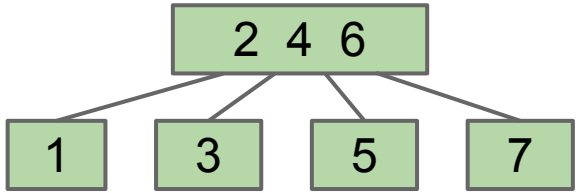
Adding 6.



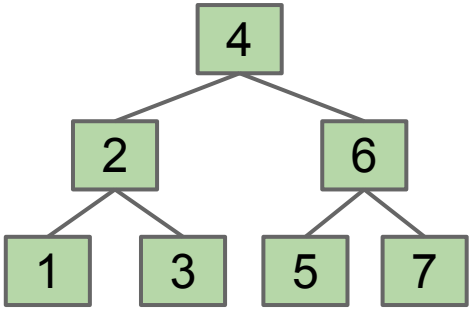
Adding 7. Node is too full, so we need to split.



Before split



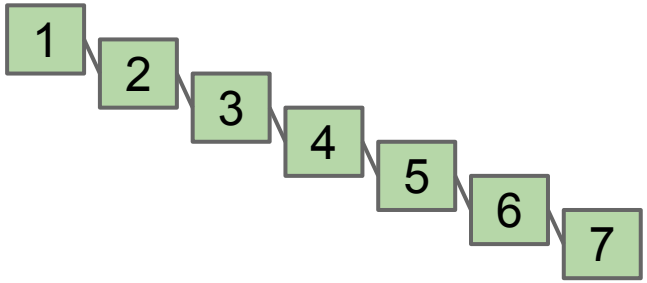
After one split



After two splits

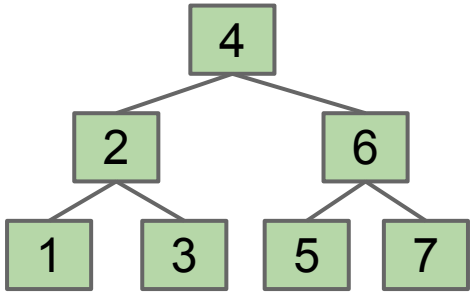
Exercise

Add the numbers 1, 2, 3, 4, 5, 6, then 7 (in that order) into a regular BST.



Then try adding 1, 2, 3, 4, 5, 6, then 7 (in that order) into a 2-3 tree.

- Interactive demo: https://tinyurl.com/balanceYD or this link.
- In this demo “max-degree” means the maximum number of children, i.e. 3.



All leaves are at depth 2.

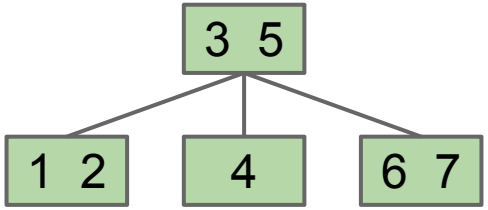
Exercise 2: Recorded Video Viewers Only

Find an order such that if you add the items 1, 2, 3, 4, 5, 6, and 7 in that order, the resulting 2-3 tree has height 1.

Exercise 2

Find an order such that if you add the items 1, 2, 3, 4, 5, 6, and 7 in that order, the resulting 2-3 tree has height 1.

- One possible answer: 2, 3, 4, 5, 6, 1, 7



All leaves are at depth 1.

Not sure why? Make sure to see https://tinyurl.com/balanceYD.

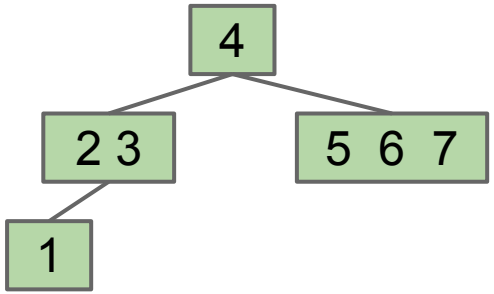
No matter the insertion order you choose, resulting B-Tree is always bushy!

- May vary in height a little bit, but overall guaranteed to be bushy.

B-Tree Invariants

Because of the way B-Trees are constructed, we get two nice invariants:

- All leaves must be the same distance from the root.
- A non-leaf node with k items must have exactly k+1 children.
- Example: The tree given below is impossible.
  - Leaves ([1] and [5 6 7]) are a different distance from the source.
  - Non-leaf node [2 3] has two items but only only one child. Should have three children.



We have not proven these invariants rigorously, but try thinking them through.

B-Tree Invariants

Because of the way B-Trees are constructed, we get two nice invariants:

- All leaves must be the same distance from the root.
- A non-leaf node with k items must have exactly k+1 children.

These invariants guarantee that our trees will be bushy.

Worst Case Performance

Lecture 17, CS61B, Fall 2024

Binary Search Trees

- BST Height, Big O vs. Worst Case Big Theta
- Worst Case Performance

B-Trees

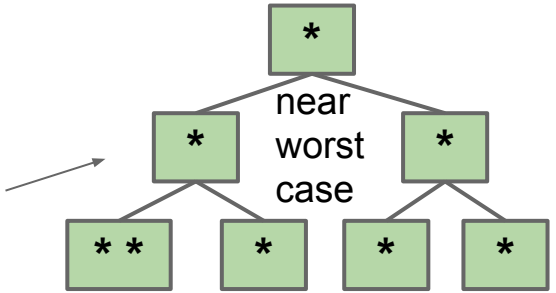
- Splitting Juicy Nodes
- Chain Reaction Splitting
- B-Tree Terminology
- Invariants
- **Worst Case Performance**

Height of a B-Tree with Limit L

L: Max number of items per node.

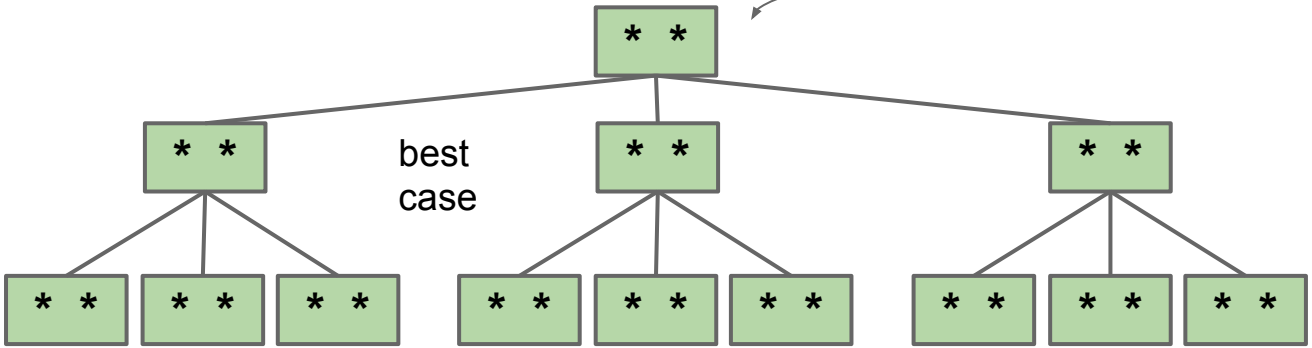
Height: Between ~$\log_{L+1}(N)$ and ~$\log_2(N)$

- Largest possible height is all non-leaf nodes have 1 item.
- Smallest possible height is all nodes have L items.
- Overall height is therefore Θ(log N).



N: 8 items
L: 2 max per node
H: 2

Height grows with $\log_2(N)$



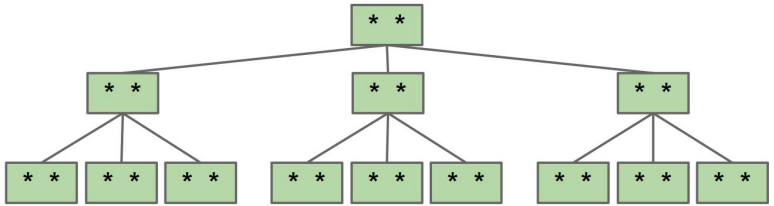
N: 26 items
L: 2 max per node
H: 2

Height grows with $\log_3(N)$

Runtime for `contains`

Runtime for contains:

- Worst case number of nodes to inspect: H + 1
- Worst case number of items to inspect per node: L
- Overall runtime: O(HL)



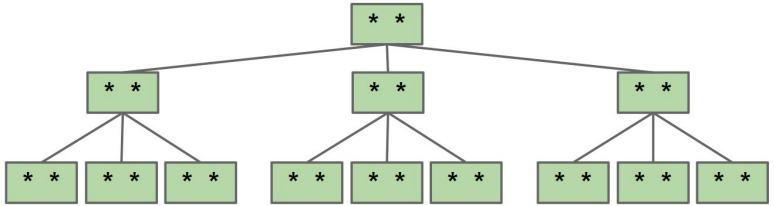
Since H = Θ(log N), overall runtime is O(L log N).

- Since L is a constant, runtime is therefore O(log N).

Runtime for add

Runtime for add:

- Worst case number of nodes to inspect: H + 1
- Worst case number of items to inspect per node: L
- Worst case number of split operations: H + 1
- Overall runtime: O(HL)



Since H = Θ(log N), overall runtime is O(L log N).

- Since L is a constant, runtime is therefore O(log N).

Bottom line: contains and add are both O(log N).

Summary

BSTs have best case height Θ(log N), and worst case height Θ(N).

- Big O is not the same thing as worst case!

B-Trees are a modification of the binary search tree that avoids Θ(N) worst case.

- Nodes may contain between 1 and L items.
- contains works almost exactly like a normal BST.
- add works by adding items to existing leaf nodes.
  - If nodes are too full, they split.
- Resulting tree has perfect balance. Runtime for operations is O(log N).
- Have not discussed deletion. See extra slides if you're curious.
- Have not discussed how splitting works if L > 3 (see some other class).
- B-trees are more complex, but they can efficiently handle ANY insertion order.

Extra slides cover deletion. Not in scope for our class.