BINARY SEARCH TREES

David Kauchak
CS 140 – Spring 2023

1

Binary Search Trees

2

Binary Search Trees

BST – A binary tree where a parent's value is greater than all values in the left subtree and less than or equal to all the values in the right subtree

$$leftTree(i) < i \leq rightTree(i)$$

and the left and right children are also binary search trees

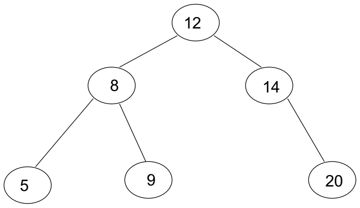
Why not?

$$leftTree(i) \leq i \leq rightTree(i)$$

Ambiguous about where elements that are equal would reside

3

Example

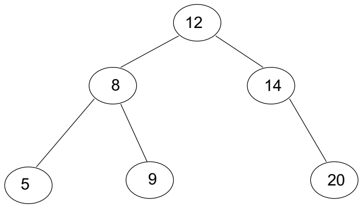


Can be implemented with with references or an array

4

What else can we conclude?

$$leftTree(i) < i \le rightTree(i)$$

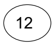


The smallest element is the left-most element

The largest element is the right-most element

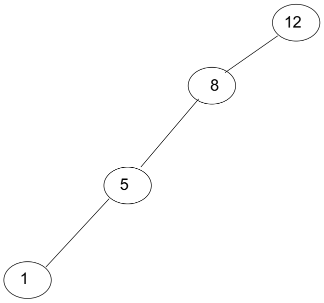
5

Another example: the solo tree



6

Another example: the twig



7

Operations

Search(T,k) – Does value k exist in tree T

Insert(T,k) – Insert value k into tree T

Delete(T,x) – Delete node x from tree T

Minimum(T) – What is the smallest value in the tree?

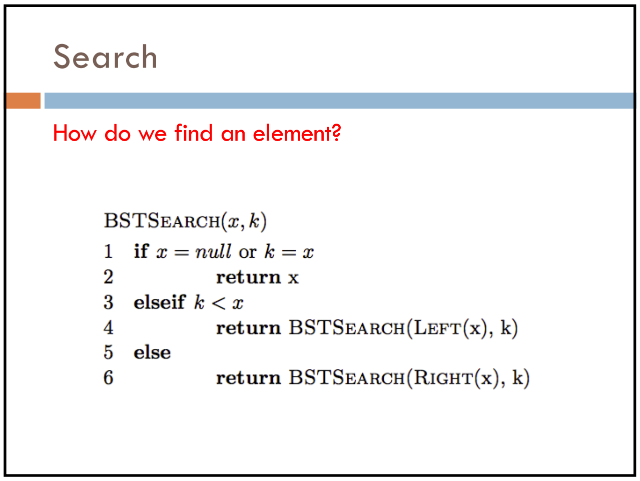
Maximum(T) – What is the largest value in the tree?

Successor(T,x) – What is the next element in sorted order after x

Predecessor(T,x) – What is the previous element in sorted order of x

Median(T) – return the median of the values in tree T

8

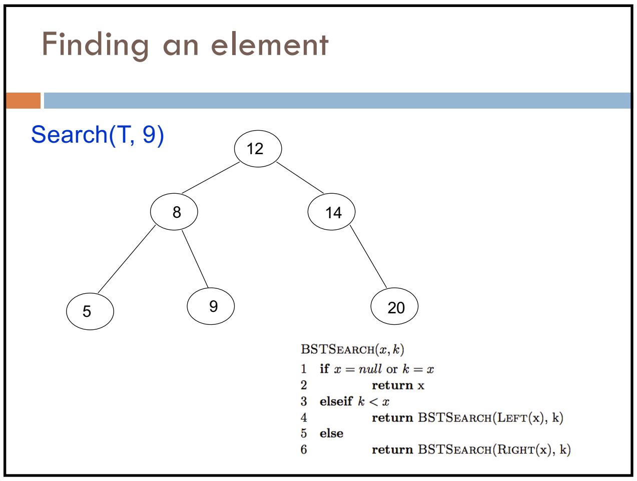


Search

How do we find an element?

```
BSTSEARCH($x, k$)
if $x = null$ or $k = x$
        return x
elseif $k < x$
        return BSTSEARCH(LEFT(x), k)
else
        return BSTSEARCH(RIGHT(x), k)
```

9

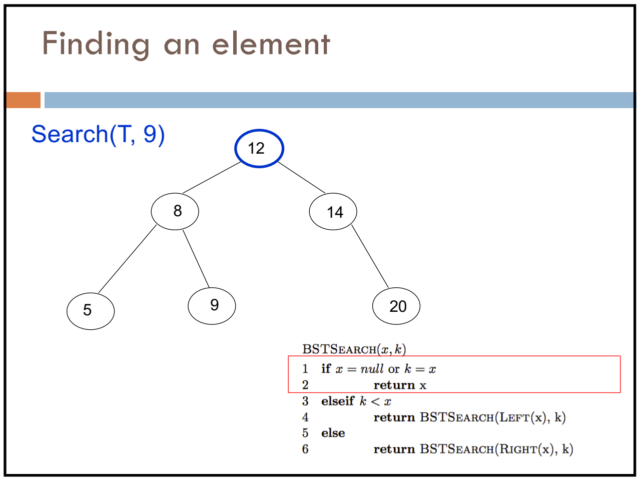


Finding an element

Search(T, 9)

```
BSTSEARCH($x, k$)
if $x = null$ or $k = x$
        return x
elseif $k < x$
        return BSTSEARCH(LEFT(x), k)
else
        return BSTSEARCH(RIGHT(x), k)
```

10

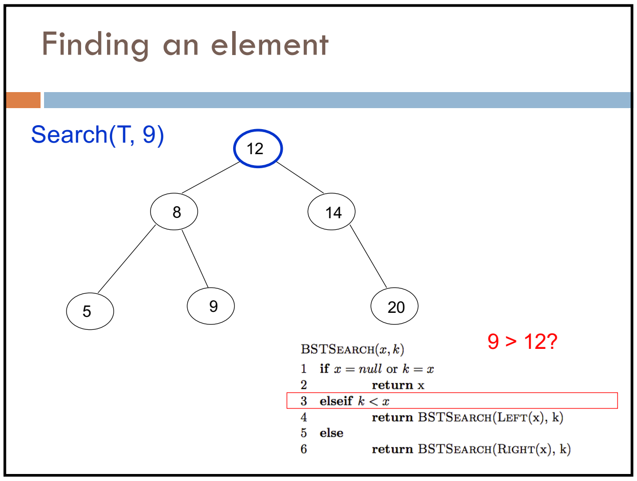


Finding an element

Search(T, 9)

```
BSTSEARCH($x, k$)
if $x = null$ or $k = x$
        return x
elseif $k < x$
        return BSTSEARCH(LEFT(x), k)
else
        return BSTSEARCH(RIGHT(x), k)
```

11



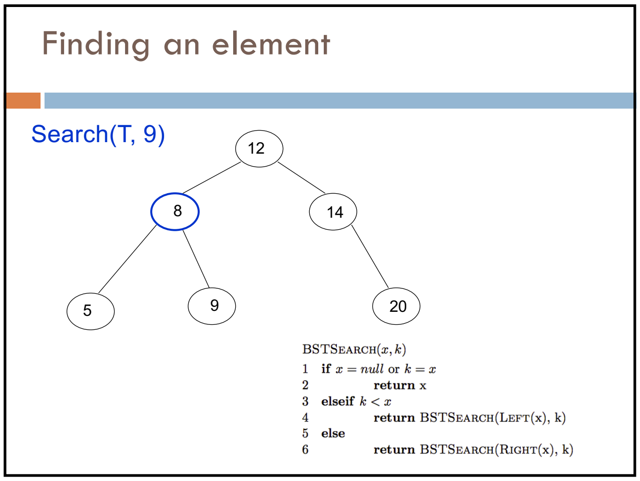
Finding an element

Search(T, 9)

9 > 12?

```
BSTSEARCH($x, k$)
if $x = null$ or $k = x$
        return x
elseif $k < x$
        return BSTSEARCH(LEFT(x), k)
else
        return BSTSEARCH(RIGHT(x), k)
```

12

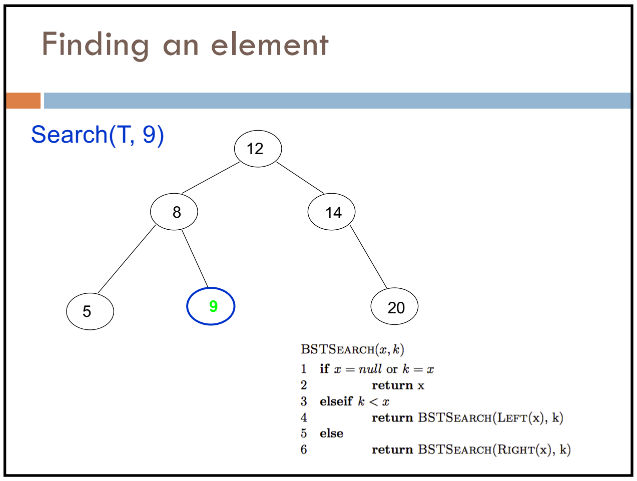


Finding an element

Search(T, 9)

```
BSTSEARCH($x, k$)
if $x = null$ or $k = x$
        return x
elseif $k < x$
        return BSTSEARCH(LEFT(x), k)
else
        return BSTSEARCH(RIGHT(x), k)
```

13

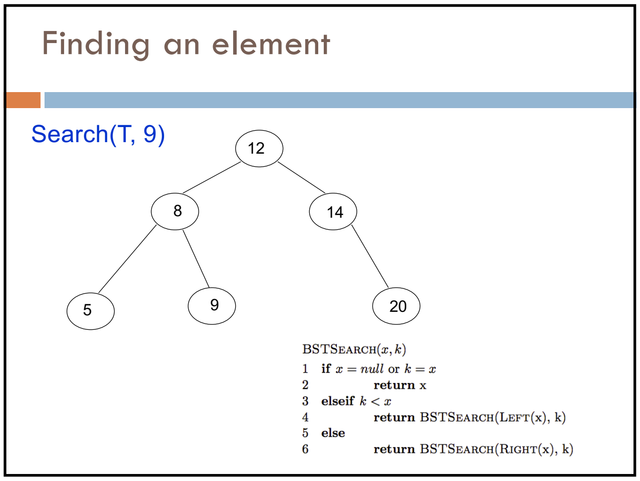


Finding an element

Search(T, 9)

```
BSTSEARCH($x, k$)
if $x = null$ or $k = x$
        return x
elseif $k < x$
        return BSTSEARCH(LEFT(x), k)
else
        return BSTSEARCH(RIGHT(x), k)
```

14

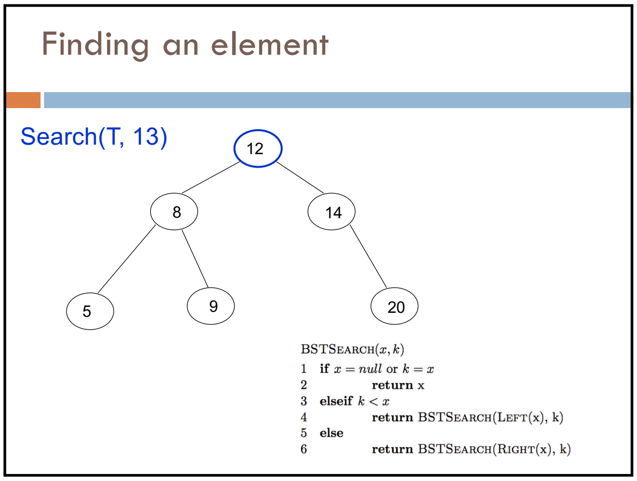


Finding an element

Search(T, 9)

```
BSTSEARCH($x, k$)
if $x = null$ or $k = x$
        return x
elseif $k < x$
        return BSTSEARCH(LEFT(x), k)
else
        return BSTSEARCH(RIGHT(x), k)
```

15

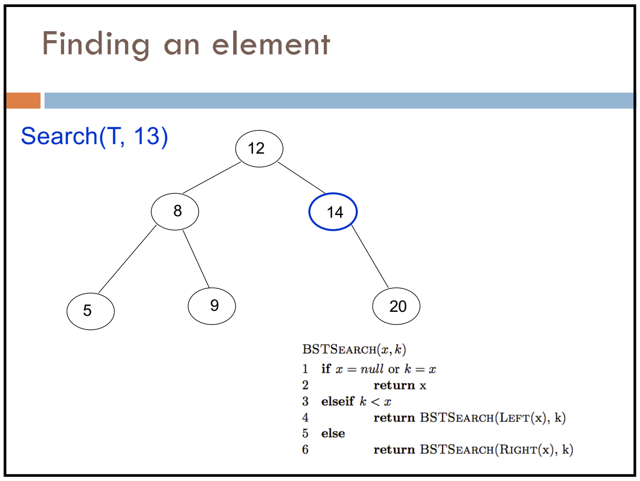


Finding an element

Search(T, 13)

```
BSTSEARCH($x, k$)
if $x = null$ or $k = x$
        return x
elseif $k < x$
        return BSTSEARCH(LEFT(x), k)
else
        return BSTSEARCH(RIGHT(x), k)
```

16

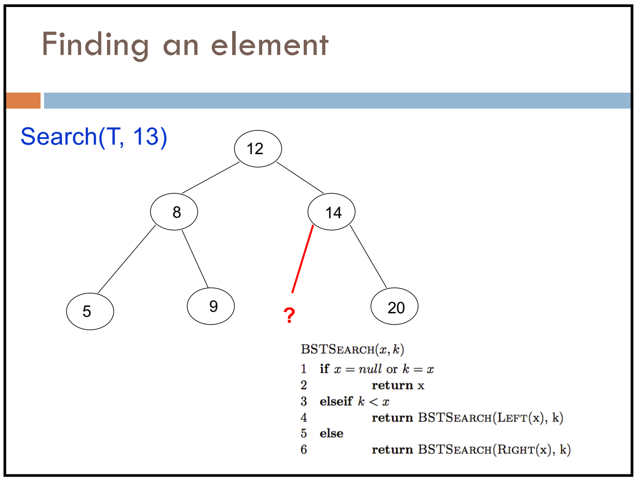


Finding an element

Search(T, 13)

```
BSTSearch($x, k$)
if $x = null$ or $k = x$
        return x
elseif $k < x$
        return BSTSearch(Left(x), k)
else
        return BSTSearch(Right(x), k)
```

17

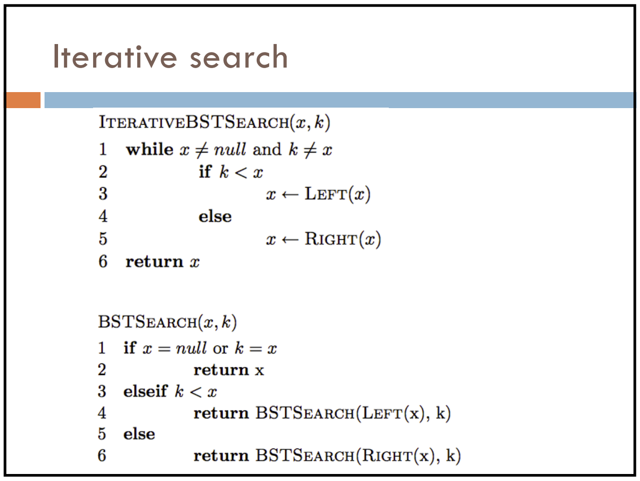


Finding an element

Search(T, 13)

```
BSTSearch($x, k$)
if $x = null$ or $k = x$
        return x
elseif $k < x$
        return BSTSearch(Left(x), k)
else
        return BSTSearch(Right(x), k)
```

18

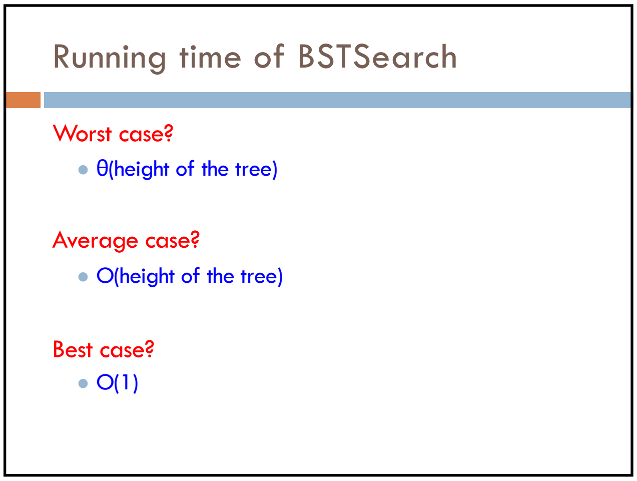


Iterative search

```
IterativeBSTSearch($x, k$)
while $x \neq null$ and $k \neq x$
        if $k < x$
                $x \leftarrow$ Left($x$)
        else
                $x \leftarrow$ Right($x$)
return $x$
```

```
BSTSearch($x, k$)
if $x = null$ or $k = x$
        return x
elseif $k < x$
        return BSTSearch(Left(x), k)
else
        return BSTSearch(Right(x), k)
```

19



Running time of BSTSearch

Worst case?

- θ(height of the tree)

Average case?

- O(height of the tree)

Best case?

- O(1)

21

Height of the tree

Worst case height?

- n-1
- “the twig”

Best case height?

- $\lfloor \log_2 n \rfloor$
- complete (or near complete) binary tree

Average case height?

- Depends on two things:
  - the data
  - how we build the tree!

Insertion

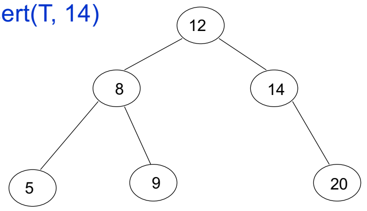
Search and then insert when you find a “null” spot in the tree

Insertion

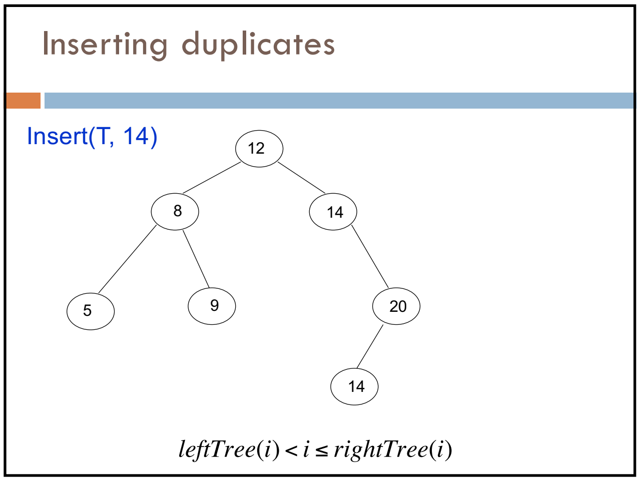
```
BSTInsert($T, x$)
if Root($T$) = $null$
        Root($T$) $\leftarrow x$
else
        $y \leftarrow$ Root($T$)
        while $y \neq null$
                $prev \leftarrow y$
                if $x < y$
                        $y \leftarrow$ Left($y$)
                else
                        $y \leftarrow$ Right($y$)
        Parent($x$) $\leftarrow prev$
        if $x < prev$
                Left($prev$) $\leftarrow x$
        else
                Right($prev$) $\leftarrow x$
```

Inserting duplicates

Insert(T, 14)



$$leftTree(i) < i \leq rightTree(i)$$

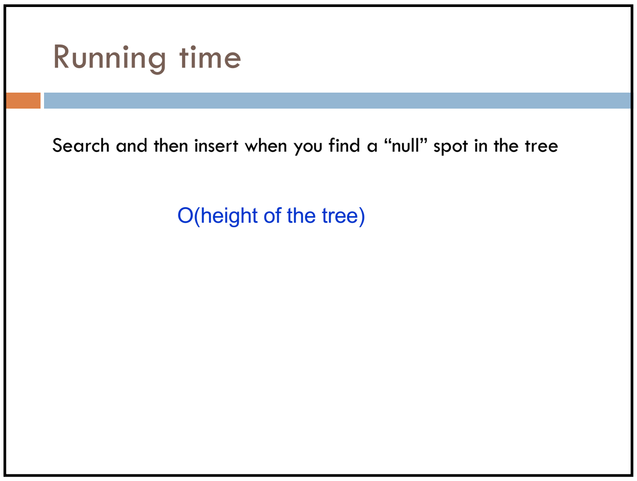


Inserting duplicates

Insert(T, 14)

$$leftTree(i) < i \le rightTree(i)$$

32

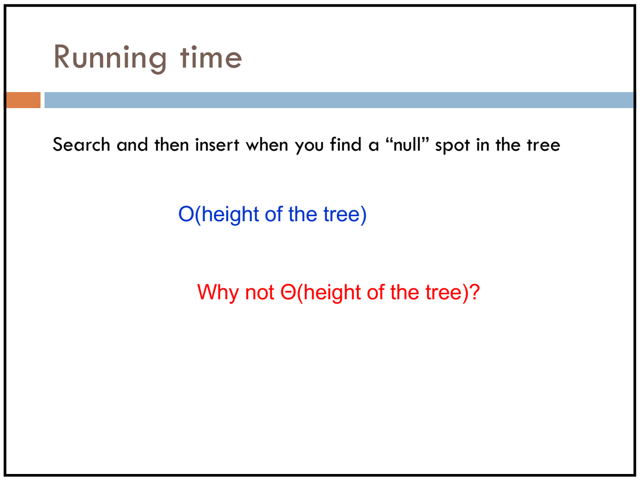


Running time

Search and then insert when you find a "null" spot in the tree

O(height of the tree)

33



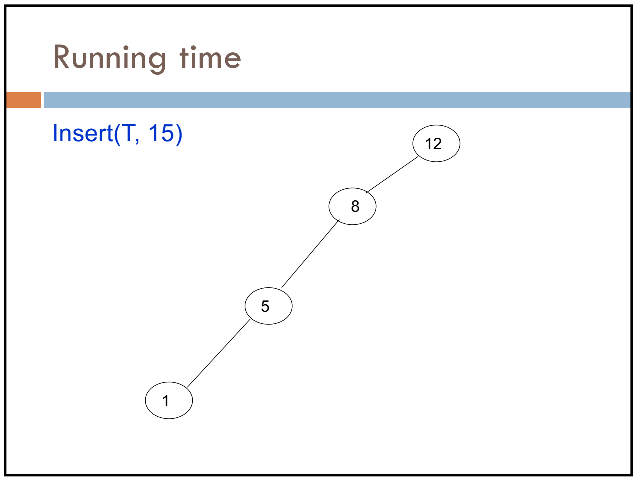
Running time

Search and then insert when you find a "null" spot in the tree

O(height of the tree)

Why not Θ(height of the tree)?

34



Running time

Insert(T, 15)

35

Height of the tree

Worst case: "the twig" – When will this happen?

Search and then insert when you find a "null" spot in the tree

36

Height of the tree

Best case: "complete" – When will this happen?

Search and then insert when you find a "null" spot in the tree

37

Height of the tree

Average case for random data?

Search and then insert when you find a "null" spot in the tree

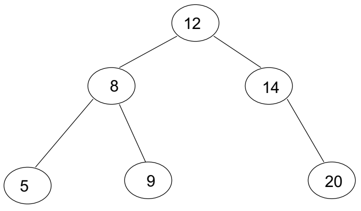
Randomly inserting data into a BST generates a tree on average that is O(log n)

38

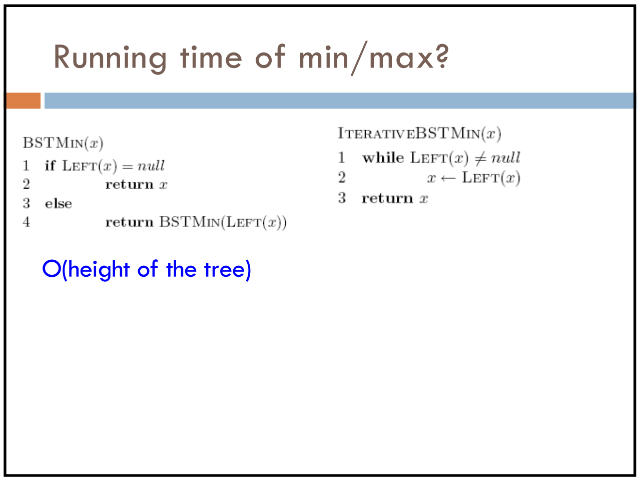
Min/Max

```
BSTMin($x$)
if Left($x$) = $null$
        return $x$
else
        return BSTMin(Left($x$))
```

```
IterativeBSTMin($x$)
while Left($x$) $\neq$ $null$
        $x \leftarrow$ Left($x$)
return $x$
```



55



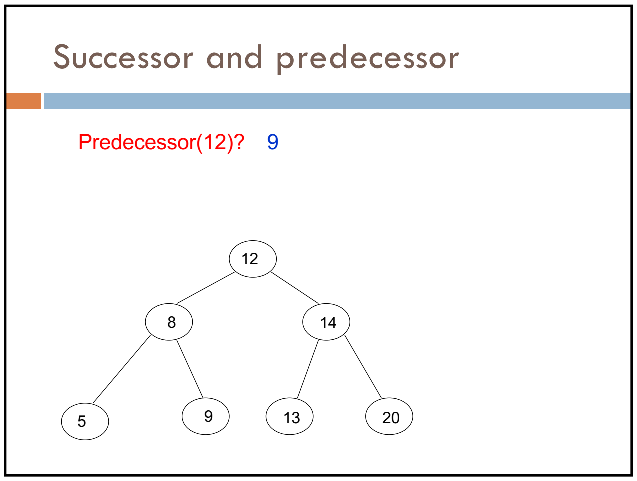
Running time of min/max?

```
BSTMin($x$)
if Left($x$) = null
        return $x$
else
        return BSTMin(Left($x$))
```

```
IterativeBSTMin($x$)
while Left($x$) $\neq$ null
        $x \leftarrow$ Left($x$)
return $x$
```

O(height of the tree)

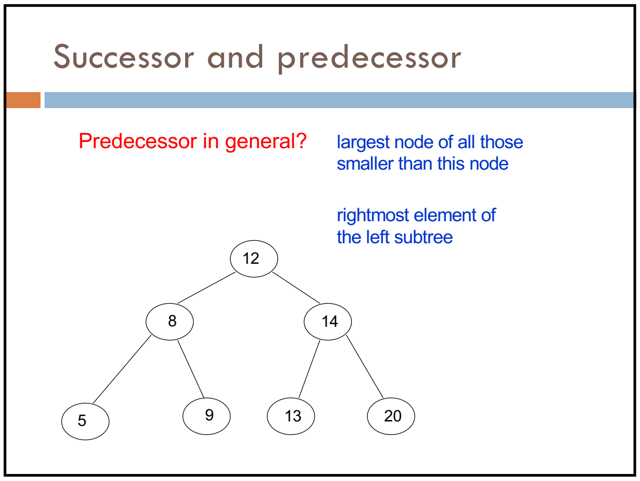
56



Successor and predecessor

Predecessor(12)? 9

57



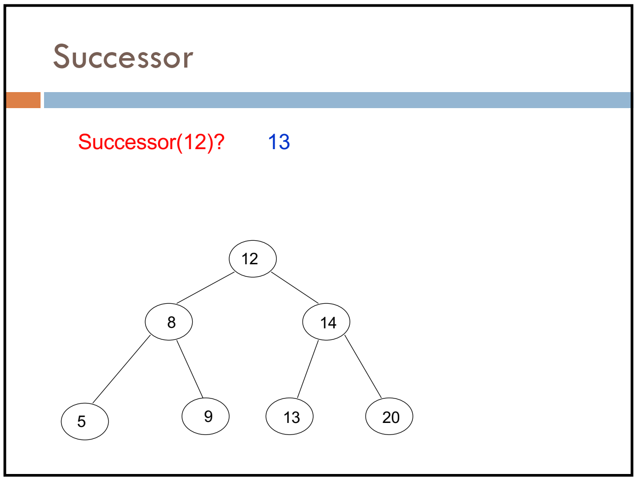
Successor and predecessor

Predecessor in general?

largest node of all those smaller than this node

rightmost element of the left subtree

58



Successor

Successor(12)? 13

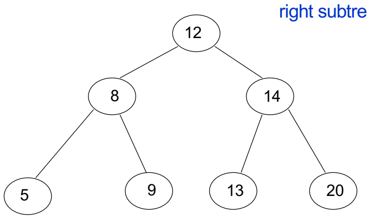
59

Successor

Successor in general?

smallest node of all those larger than this node

leftmost element of the right subtree



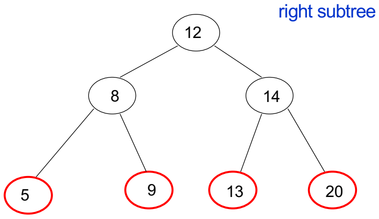
60

Successor

What if the node doesn't have a right subtree?

smallest node of all those larger than this node

leftmost element of the right subtree



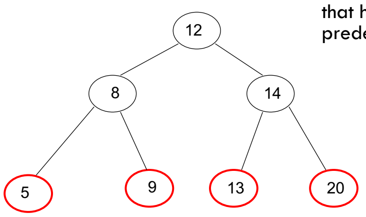
61

Successor

What if the node doesn't have a right subtree?

node is the largest

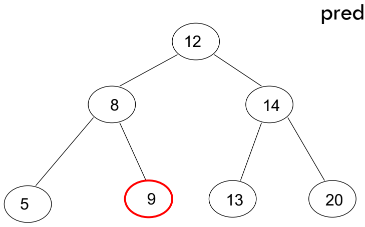
the successor is the node that has x as a predecessor



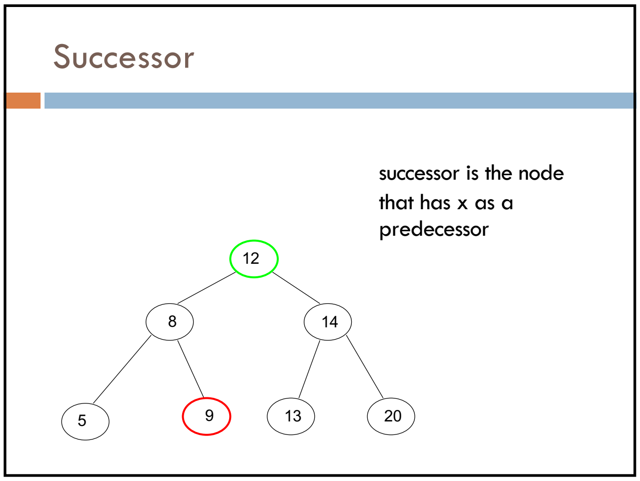
62

Successor

successor is the node that has x as a predecessor



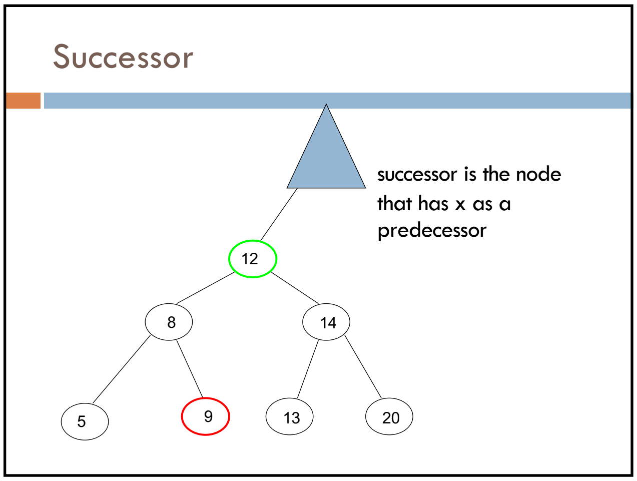
63



Successor

successor is the node that has x as a predecessor

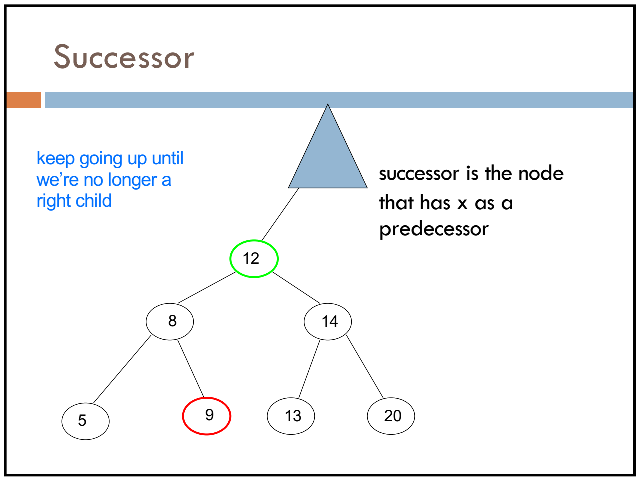
64



Successor

successor is the node that has x as a predecessor

65

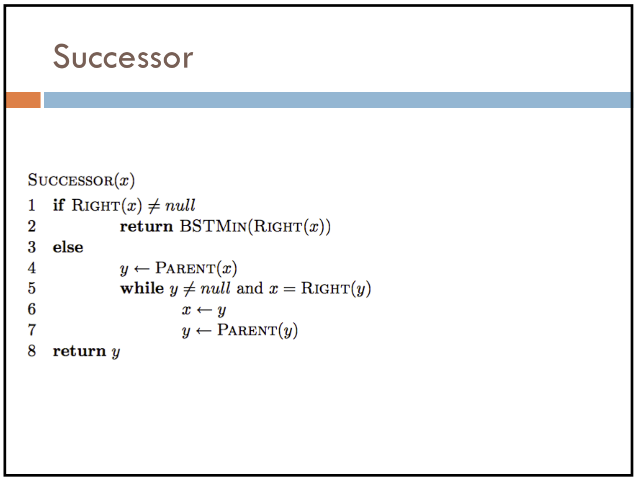


Successor

keep going up until we're no longer a right child

successor is the node that has x as a predecessor

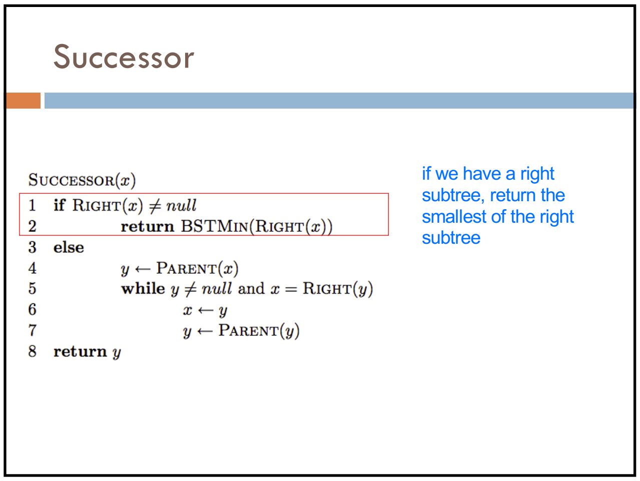
66



Successor

```
SUCCESSOR($x$)
if RIGHT($x$) $\neq$ null
        return BSTMIN(RIGHT($x$))
else
        $y \leftarrow$ PARENT($x$)
        while $y \neq$ null and $x =$ RIGHT($y$)
                $x \leftarrow y$
                $y \leftarrow$ PARENT($y$)
return $y$
```

67

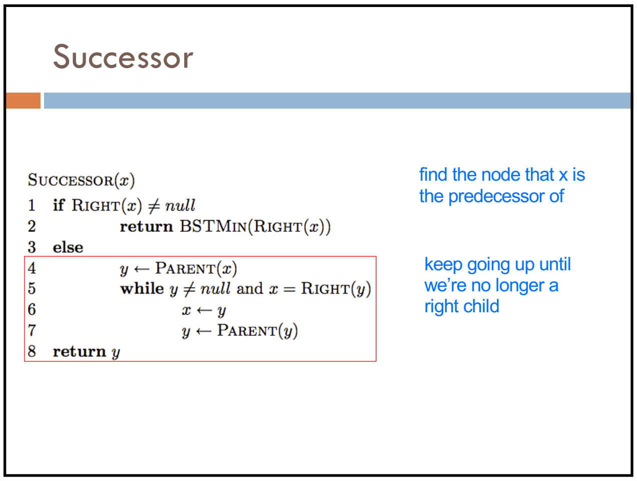


Successor

```
SUCCESSOR($x$)
if RIGHT($x$) $\neq$ null
        return BSTMIN(RIGHT($x$))
else
        $y \leftarrow$ PARENT($x$)
        while $y \neq$ null and $x =$ RIGHT($y$)
                $x \leftarrow y$
                $y \leftarrow$ PARENT($y$)
return $y$
```

if we have a right subtree, return the smallest of the right subtree

68



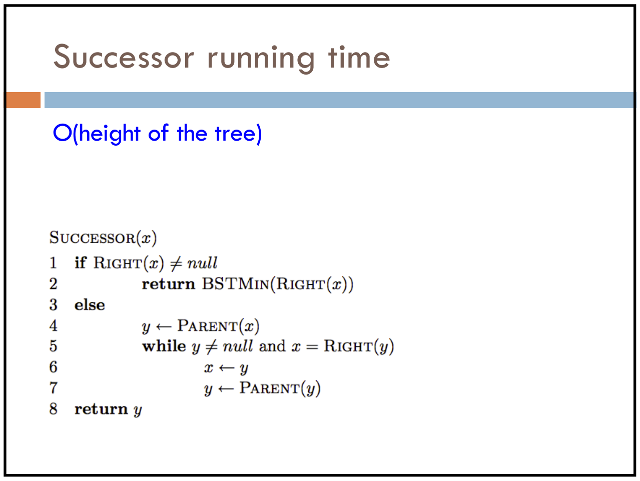
Successor

```
SUCCESSOR($x$)
if RIGHT($x$) $\neq$ null
        return BSTMIN(RIGHT($x$))
else
        $y \leftarrow$ PARENT($x$)
        while $y \neq$ null and $x =$ RIGHT($y$)
                $x \leftarrow y$
                $y \leftarrow$ PARENT($y$)
return $y$
```

find the node that x is the predecessor of

keep going up until we're no longer a right child

69

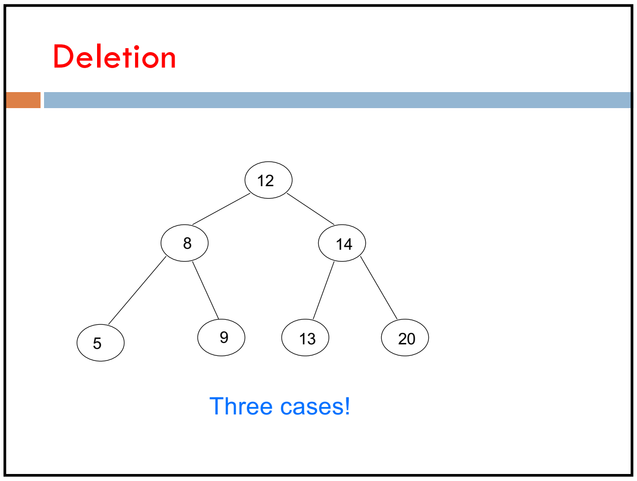


Successor running time

O(height of the tree)

```
SUCCESSOR($x$)
if RIGHT($x$) $\neq$ null
        return BSTMIN(RIGHT($x$))
else
        $y \leftarrow$ PARENT($x$)
        while $y \neq$ null and $x =$ RIGHT($y$)
                $x \leftarrow y$
                $y \leftarrow$ PARENT($y$)
return $y$
```

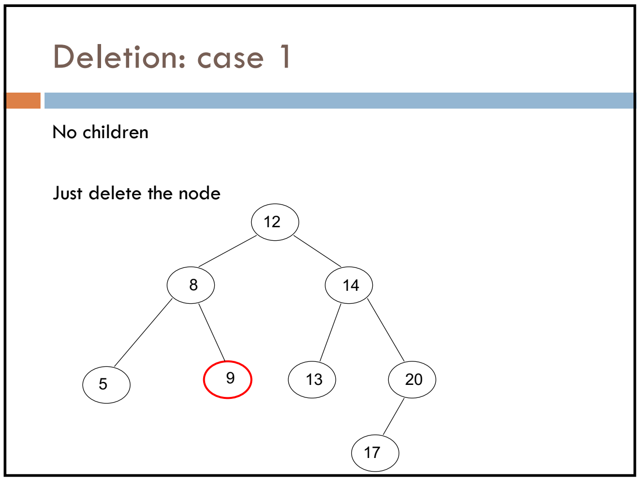
70



Deletion

Three cases!

71

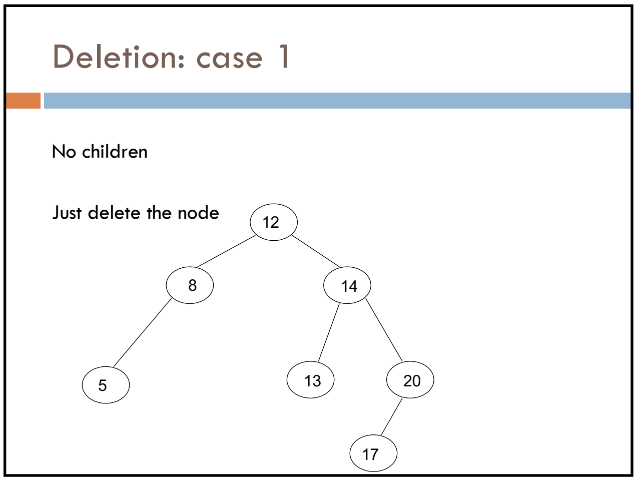


Deletion: case 1

No children

Just delete the node

72

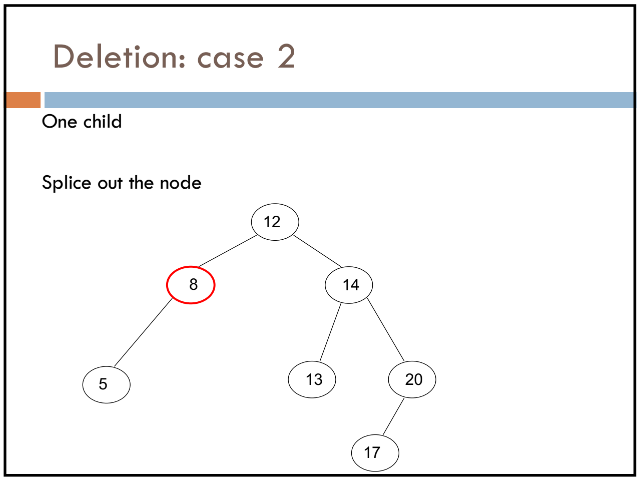


Deletion: case 1

No children

Just delete the node

73

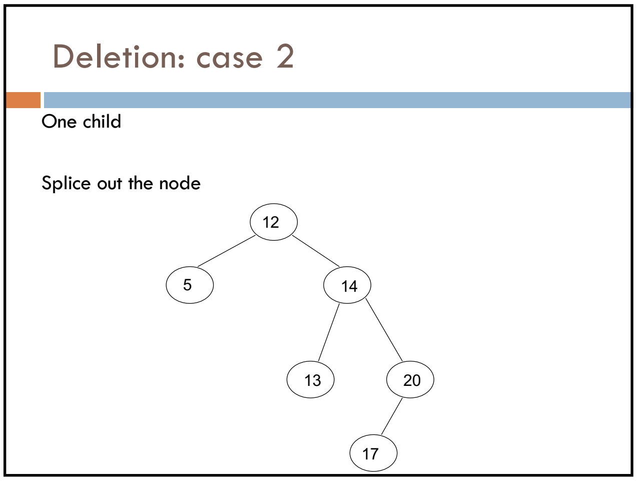


Deletion: case 2

One child

Splice out the node

74

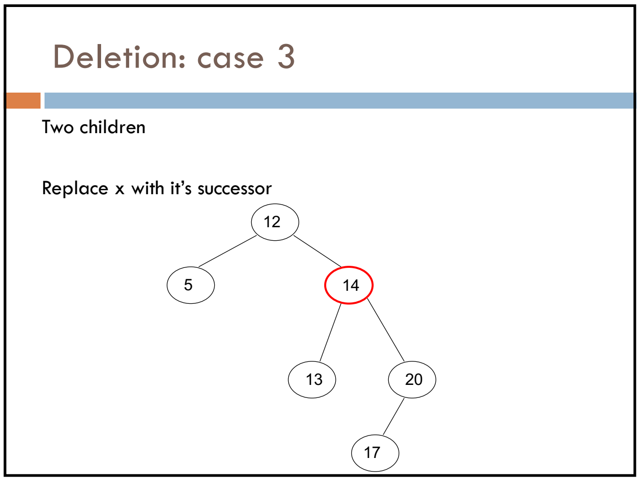


Deletion: case 2

One child

Splice out the node

75

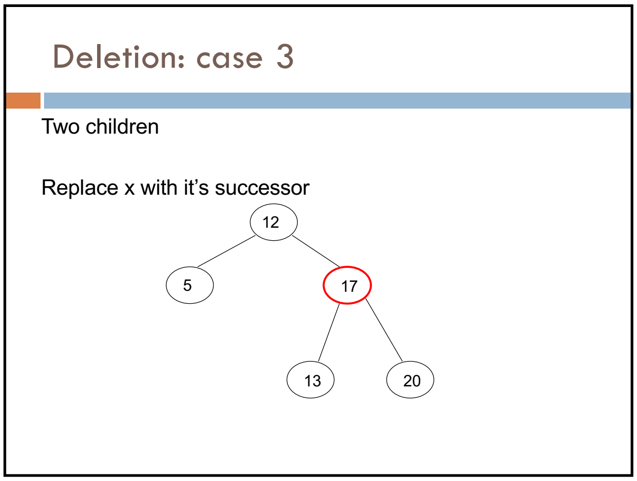


Deletion: case 3

Two children

Replace x with it's successor

76

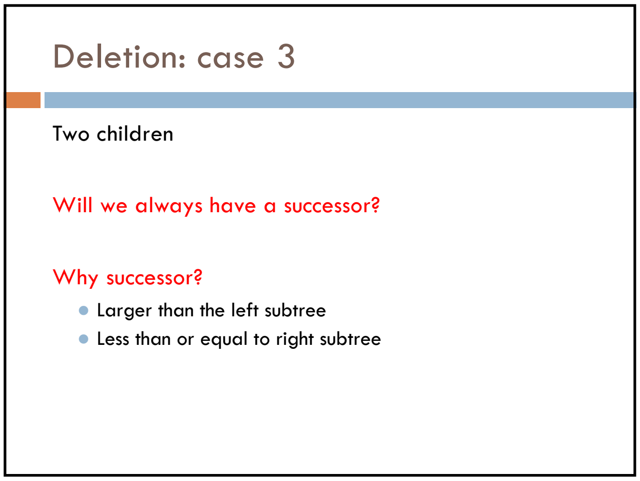


Deletion: case 3

Two children

Replace x with it's successor

77



Deletion: case 3

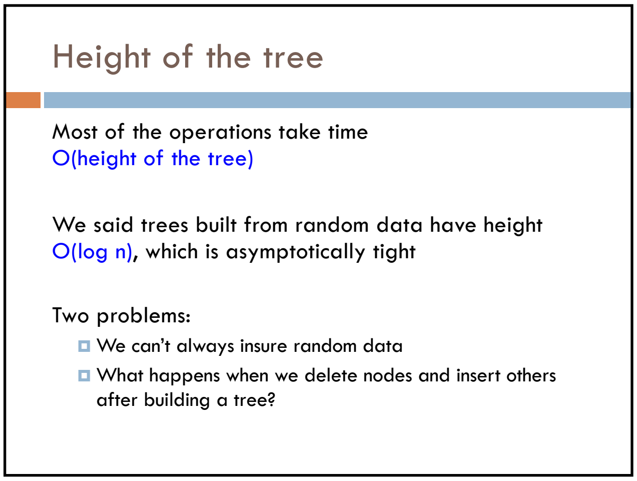
Two children

Will we always have a successor?

Why successor?

- Larger than the left subtree
- Less than or equal to right subtree

78



Height of the tree

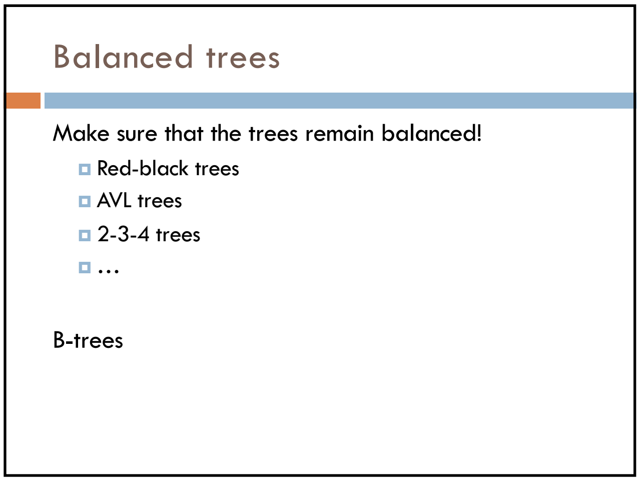
Most of the operations take time O(height of the tree)

We said trees built from random data have height O(log n), which is asymptotically tight

Two problems:

- We can't always insure random data
- What happens when we delete nodes and insert others after building a tree?

79



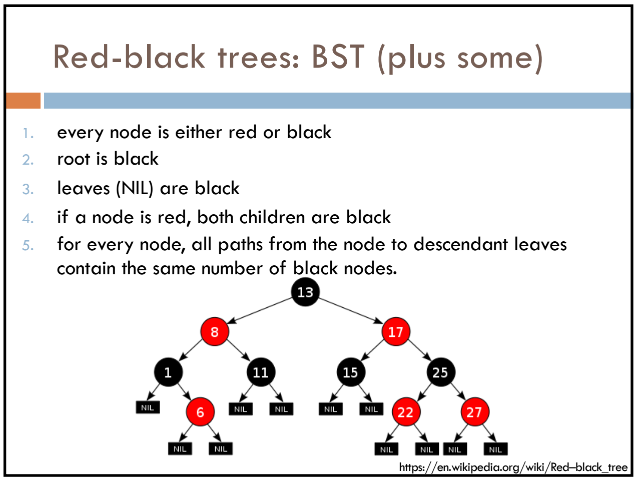
Balanced trees

Make sure that the trees remain balanced!

- Red-black trees
- AVL trees
- 2-3-4 trees
- …

B-trees

80

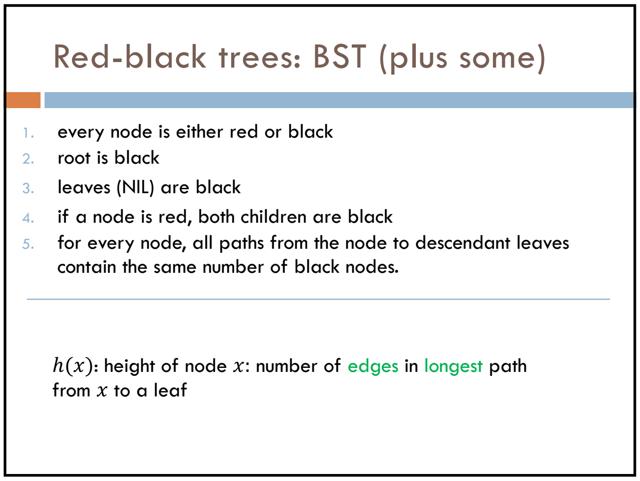


Red-black trees: BST (plus some)

1. every node is either red or black
2. root is black
3. leaves (NIL) are black
4. if a node is red, both children are black
5. for every node, all paths from the node to descendant leaves contain the same number of black nodes.

https://en.wikipedia.org/wiki/Red–black_tree

81

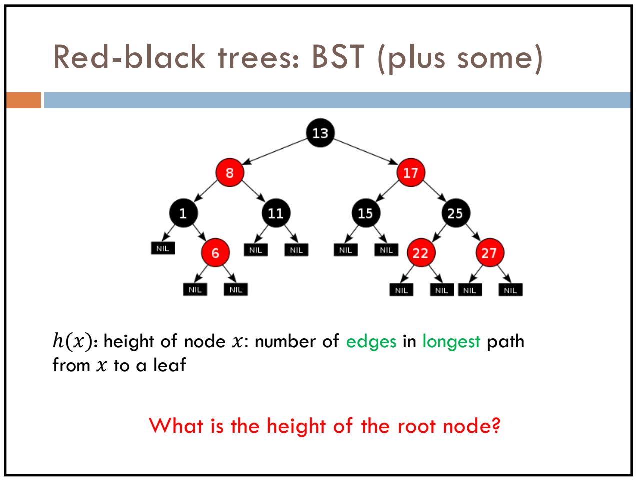


Red-black trees: BST (plus some)

1. every node is either red or black
2. root is black
3. leaves (NIL) are black
4. if a node is red, both children are black
5. for every node, all paths from the node to descendant leaves contain the same number of black nodes.

$h(x)$: height of node x: number of edges in longest path from x to a leaf

82

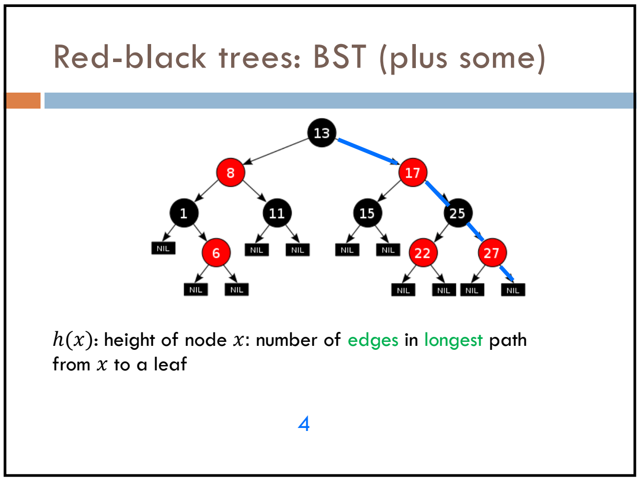


Red-black trees: BST (plus some)

$h(x)$: height of node x: number of edges in longest path from x to a leaf

What is the height of the root node?

83

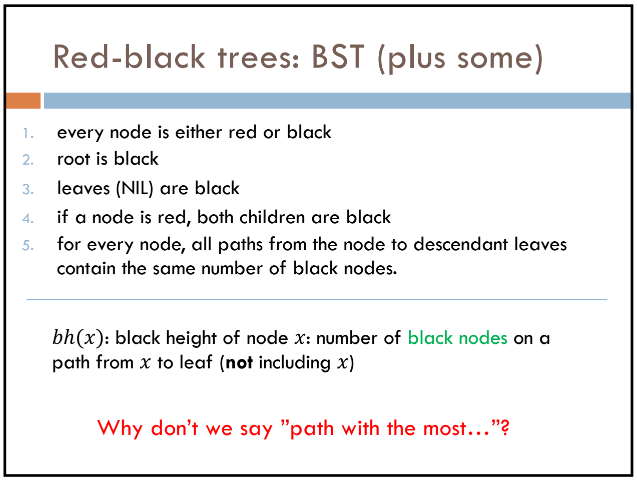


Red-black trees: BST (plus some)

$h(x)$: height of node x: number of edges in longest path from x to a leaf

4

84



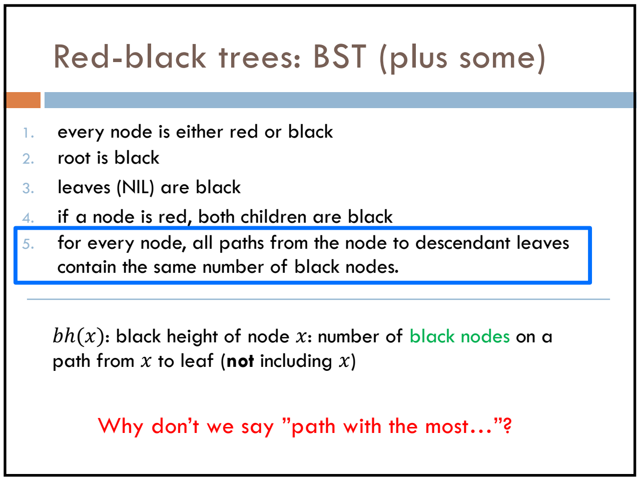
Red-black trees: BST (plus some)

1. every node is either red or black
2. root is black
3. leaves (NIL) are black
4. if a node is red, both children are black
5. for every node, all paths from the node to descendant leaves contain the same number of black nodes.

$bh(x)$: black height of node x: number of black nodes on a path from x to leaf (**not** including x)

Why don't we say "path with the most…"?

85



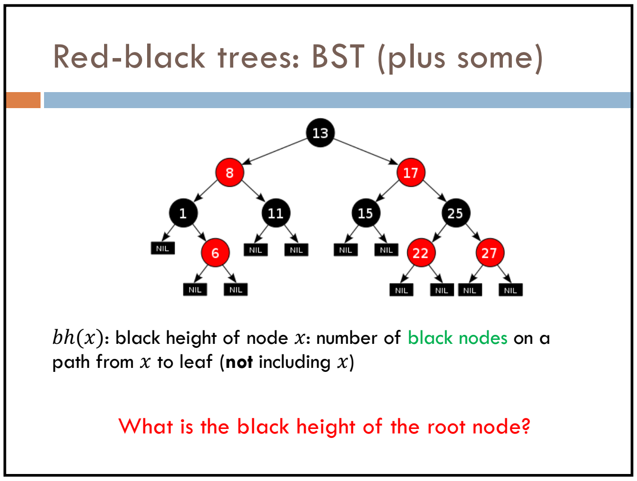
Red-black trees: BST (plus some)

1. every node is either red or black
2. root is black
3. leaves (NIL) are black
4. if a node is red, both children are black
5. for every node, all paths from the node to descendant leaves contain the same number of black nodes.

$bh(x)$: black height of node x: number of black nodes on a path from x to leaf (**not** including x)

Why don't we say "path with the most…"?

86



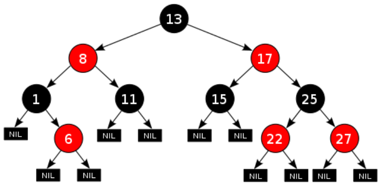
Red-black trees: BST (plus some)

$bh(x)$: black height of node x: number of black nodes on a path from x to leaf (**not** including x)

What is the black height of the root node?

87

Red-black trees: BST (plus some)



$bh(x)$: black height of node x: number of black nodes on a path from x to leaf (**not** including x)

2

88

Bounding the height

1. every node is either red or black
2. root is black
3. leaves (NIL) are black
4. if a node is red, both children are black
5. for every node, all paths from the node to descendant leaves contain the same number of black nodes.

$h(x)$: height of node x: number of edges in longest path from x to a leaf

$bh(x)$: black height of node x: number of black nodes on a path from x to leaf (**not** including x)

Claim 1: For every node x, $bh(x) \geq h(x)/2$

Proof?

89

Bounding the height

1. every node is either red or black
2. root is black
3. leaves (NIL) are black
4. if a node is red, both children are black
5. for every node, all paths from the node to descendant leaves contain the same number of black nodes.

$h(x)$: height of node x: number of edges in longest path from x to a leaf

$bh(x)$: black height of node x: number of black nodes on a path from x to leaf (**not** including x)

Claim 1: For every node x, $bh(x) \geq h(x)/2$

Worst case: nodes alternate red/black

- root is black
- leaf is black

In terms of h(x): How many black nodes are there on this path?

90

Bounding the height

1. every node is either red or black
2. root is black
3. leaves (NIL) are black
4. if a node is red, both children are black
5. for every node, all paths from the node to descendant leaves contain the same number of black nodes.

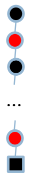
$h(x)$: height of node x: number of edges in longest path from x to a leaf

$bh(x)$: black height of node x: number of black nodes on a path from x to leaf (**not** including x)

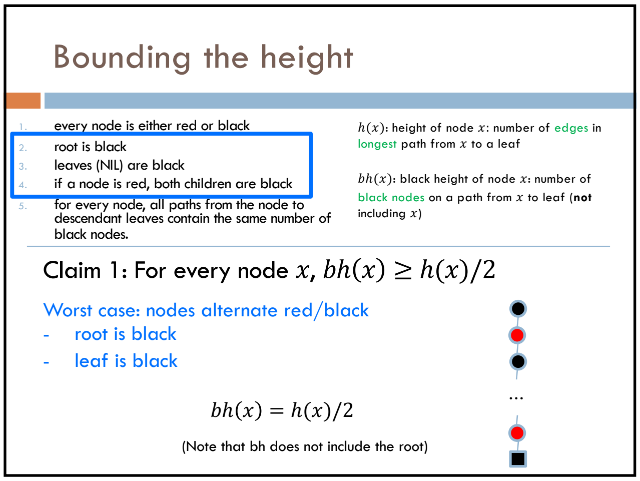
Claim 1: For every node x, $bh(x) \geq h(x)/2$

Worst case: nodes alternate red/black

- root is black
- leaf is black



91



Bounding the height

1. every node is either red or black
2. root is black
3. leaves (NIL) are black
4. if a node is red, both children are black
5. for every node, all paths from the node to descendant leaves contain the same number of black nodes.

$h(x)$: height of node x: number of edges in longest path from x to a leaf

$bh(x)$: black height of node x: number of black nodes on a path from x to leaf (**not** including x)

Claim 1: For every node x, $bh(x) \geq h(x)/2$

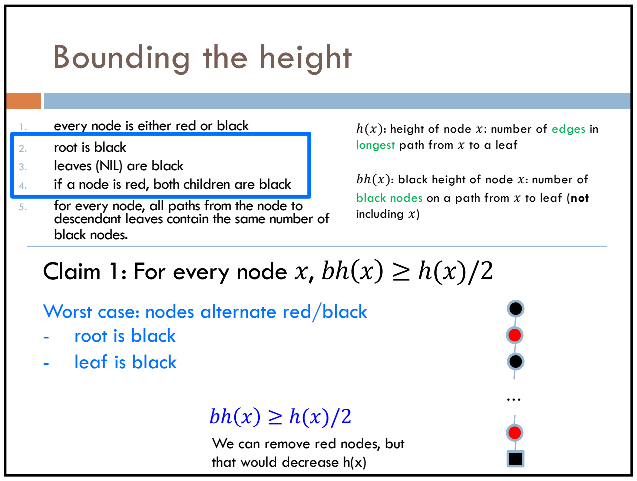
Worst case: nodes alternate red/black

- root is black
- leaf is black

$$bh(x) = h(x)/2$$

(Note that bh does not include the root)

92



Bounding the height

1. every node is either red or black
2. root is black
3. leaves (NIL) are black
4. if a node is red, both children are black
5. for every node, all paths from the node to descendant leaves contain the same number of black nodes.

$h(x)$: height of node x: number of edges in longest path from x to a leaf

$bh(x)$: black height of node x: number of black nodes on a path from x to leaf (**not** including x)

Claim 1: For every node x, $bh(x) \geq h(x)/2$

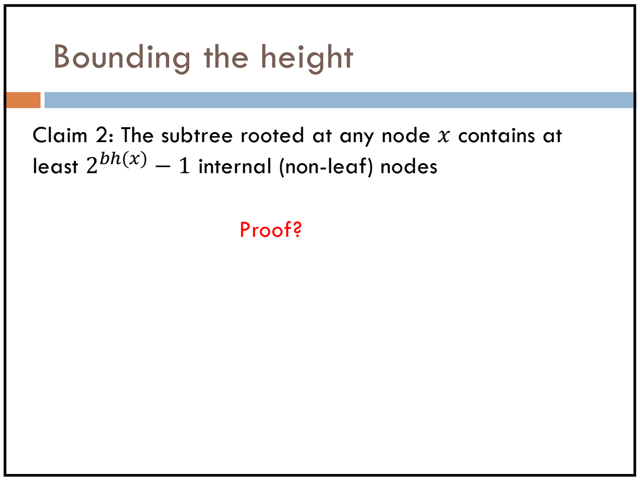
Worst case: nodes alternate red/black

- root is black
- leaf is black

$$bh(x) \geq h(x)/2$$

We can remove red nodes, but that would decrease h(x)

93

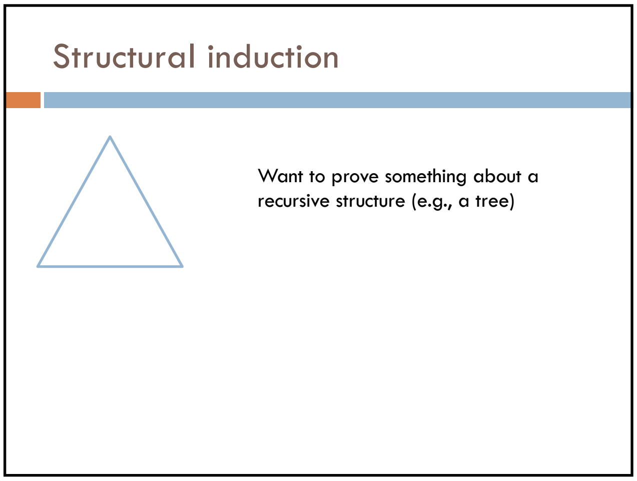


Bounding the height

Claim 2: The subtree rooted at any node x contains at least $2^{bh(x)} - 1$ internal (non-leaf) nodes

Proof?

94

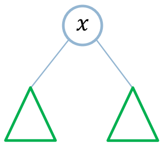


Structural induction

Want to prove something about a recursive structure (e.g., a tree)

95

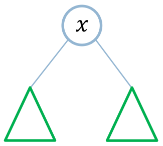
Structural induction



Proof by induction:
IH: Assume the property holds for sub-structures (i.e., subtrees)
Show that it holds for the entire tree

96

Structural induction



Base case is often the smallest structure possible (e.g., a leaf)

97

Bounding the height

Claim 2: The subtree rooted at any node x contains at least $2^{bh(x)} - 1$ internal (non-leaf) nodes

Base case:

98

Bounding the height

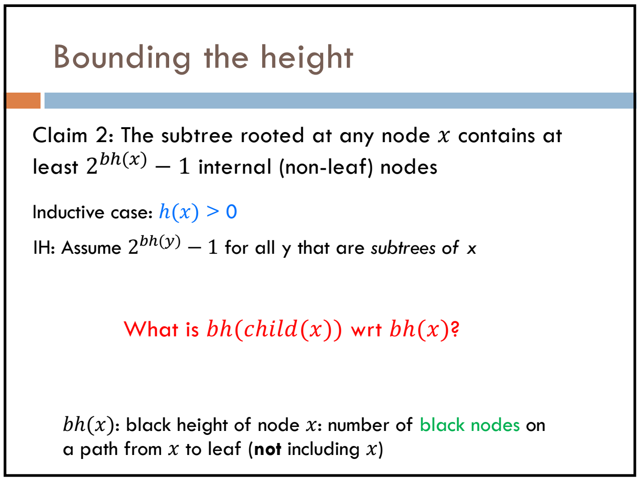
Claim 2: The subtree rooted at any node x contains at least $2^{bh(x)} - 1$ internal (non-leaf) nodes

Base case: leaf $(h(x) = 0)$

$$bh(x) = 0$$
$$2^0 - 1 = 0$$

$bh(x)$: black height of node x: number of black nodes on a path from x to leaf (**not** including x)

99



Bounding the height

Claim 2: The subtree rooted at any node x contains at least $2^{bh(x)} - 1$ internal (non-leaf) nodes

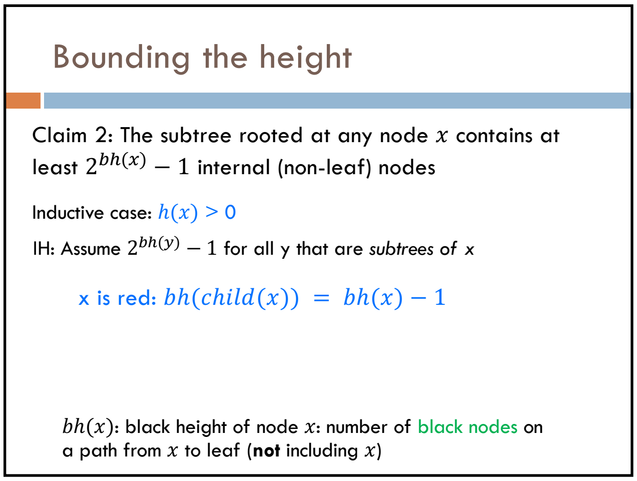
Inductive case: $h(x) > 0$

IH: Assume $2^{bh(y)} - 1$ for all y that are *subtrees* of x

What is $bh(child(x))$ wrt $bh(x)$?

$bh(x)$: black height of node x: number of black nodes on a path from x to leaf (**not** including x)

100



Bounding the height

Claim 2: The subtree rooted at any node x contains at least $2^{bh(x)} - 1$ internal (non-leaf) nodes

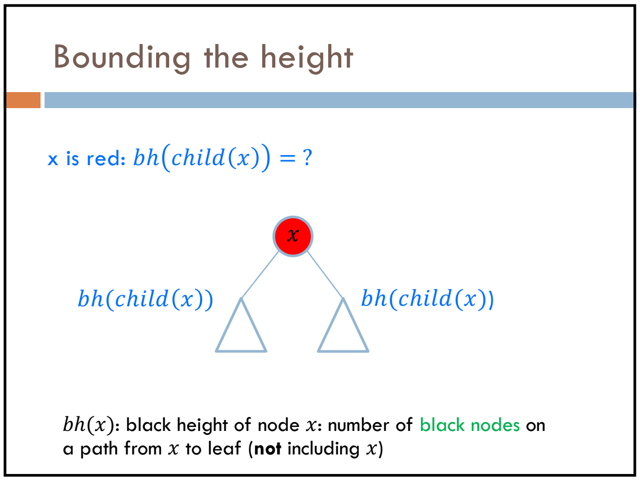
Inductive case: $h(x) > 0$

IH: Assume $2^{bh(y)} - 1$ for all y that are *subtrees* of x

x is red: $bh(child(x)) \ = \ bh(x) - 1$

$bh(x)$: black height of node x: number of black nodes on a path from x to leaf (**not** including x)

101

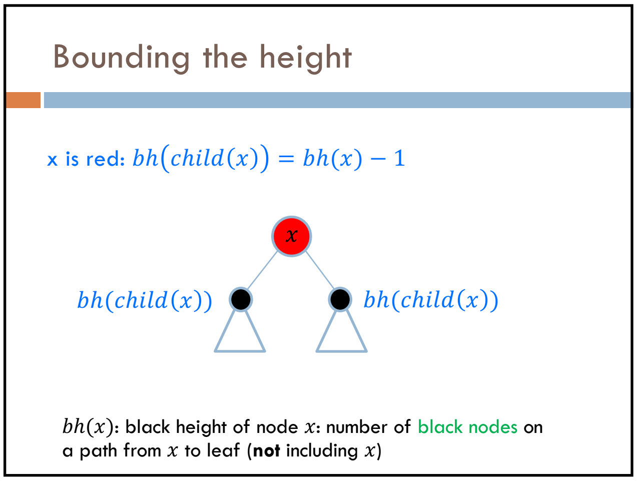


Bounding the height

x is red: $bh\big(child(x)\big) = ?$

$bh(x)$: black height of node x: number of black nodes on a path from x to leaf (**not** including x)

102



Bounding the height

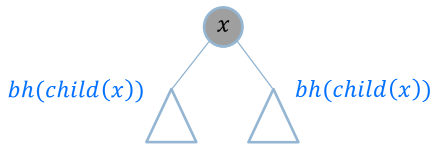
x is red: $bh\big(child(x)\big) = bh(x) - 1$

$bh(x)$: black height of node x: number of black nodes on a path from x to leaf (**not** including x)

103

Bounding the height

x is black: $bh(child(x)) = ?$

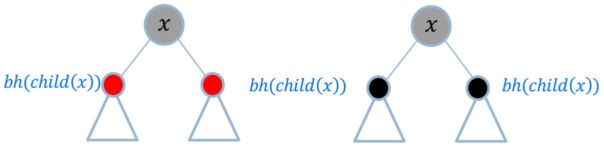


$bh(x)$: black height of node x: number of black nodes on a path from x to leaf (**not** including x)

104

Bounding the height

x is black: $bh(child(x)) = bh(x) \; or \; bh(x) - 1$



$bh(x)$: black height of node x: number of black nodes on a path from x to leaf (**not** including x)

105

Bounding the height

Claim 2: The subtree rooted at any node x contains at least $2^{bh(x)} - 1$ internal (non-leaf) nodes

Inductive case: $h(x) > 0$

IH: Assume $2^{bh(y)} - 1$ for all y that are *subtrees* of x

x is red: $bh(child(x)) \; = \; bh(x) - 1$
x is black: $bh(child(x)) \; = \; bh(x) \; or \; bh(x) - 1$

$bh(child(x)) \; \geq \; bh(x) - 1$

106

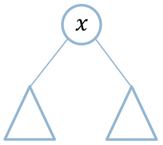
Bounding the height

Claim 2: The subtree rooted at any node x contains at least $2^{bh(x)} - 1$ internal (non-leaf) nodes

Inductive case: $h(x) > 0$

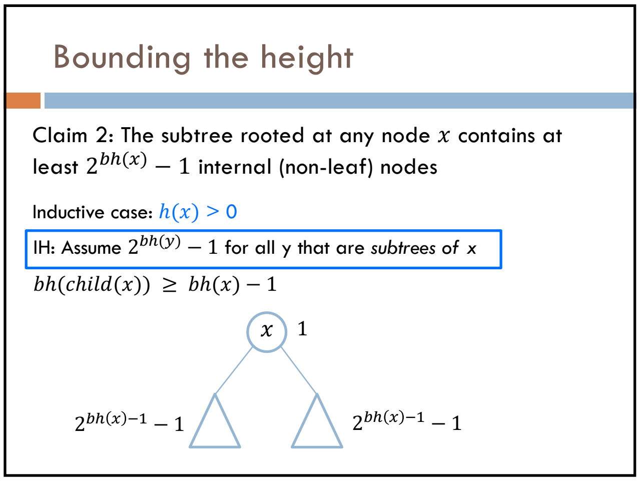
IH: Assume $2^{bh(y)} - 1$ for all y that are *subtrees* of x

$bh(child(x)) \; \geq \; bh(x) - 1$



How many (internal nodes are in this tree (at least)?

107



Bounding the height

Claim 2: The subtree rooted at any node x contains at least $2^{bh(x)}-1$ internal (non-leaf) nodes

Inductive case: $h(x) > 0$

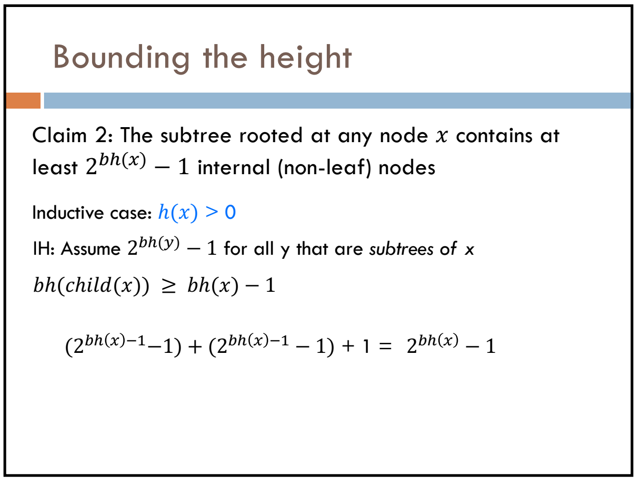
IH: Assume $2^{bh(y)}-1$ for all y that are *subtrees* of x

$bh(child(x)) \geq bh(x)-1$

x 1

$2^{bh(x)-1}-1$ $2^{bh(x)-1}-1$

108



Bounding the height

Claim 2: The subtree rooted at any node x contains at least $2^{bh(x)}-1$ internal (non-leaf) nodes

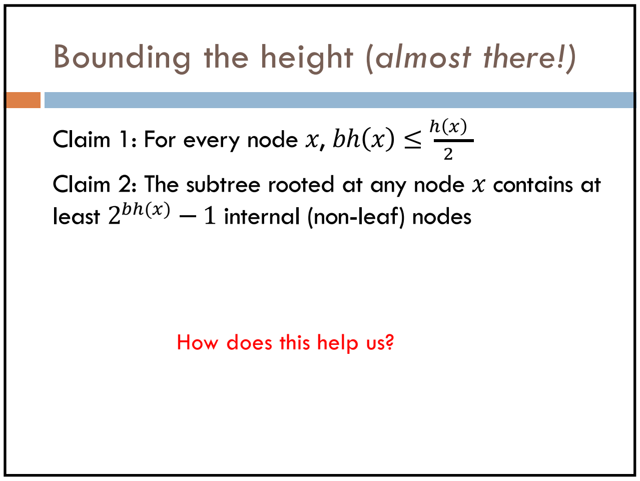
Inductive case: $h(x) > 0$

IH: Assume $2^{bh(y)}-1$ for all y that are *subtrees* of x

$bh(child(x)) \geq bh(x)-1$

$$(2^{bh(x)-1}-1)+(2^{bh(x)-1}-1)+1 = 2^{bh(x)}-1$$

109



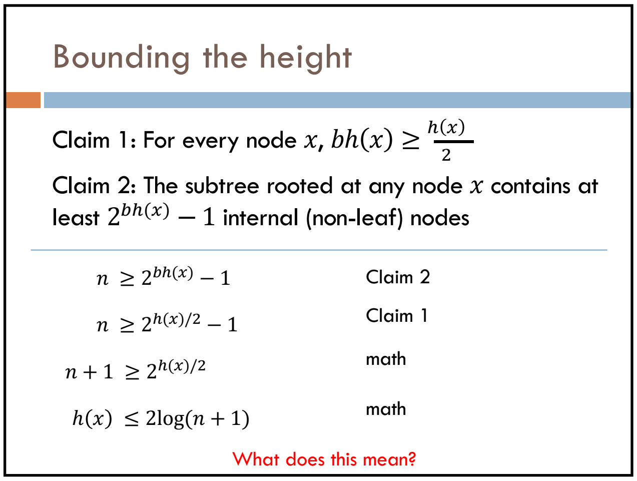
Bounding the height (*almost there!*)

Claim 1: For every node x, $bh(x) \leq \frac{h(x)}{2}$

Claim 2: The subtree rooted at any node x contains at least $2^{bh(x)}-1$ internal (non-leaf) nodes

How does this help us?

110



Bounding the height

Claim 1: For every node x, $bh(x) \geq \frac{h(x)}{2}$

Claim 2: The subtree rooted at any node x contains at least $2^{bh(x)}-1$ internal (non-leaf) nodes

$n \geq 2^{bh(x)}-1$	Claim 2
$n \geq 2^{h(x)/2}-1$	Claim 1
$n+1 \geq 2^{h(x)/2}$	math
$h(x) \leq 2\log(n+1)$	math

What does this mean?

111

Bounding the height

1. every node is either red or black
2. root is black
3. leaves (NIL) are black
4. if a node is red, both children are black
5. for every node, all paths from the node to descendant leaves contain the same number of black nodes.

If we can maintain these properties: height $O(\log n)$

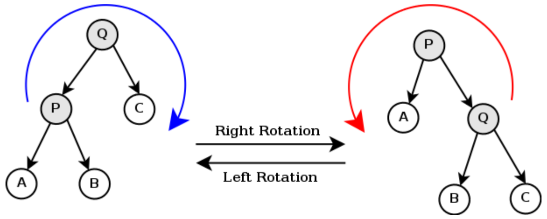
Search
Insert
Delete
Maximum

These all become $O(\log n)$

112

Can it be done?

Can we maintain the red-black tree properties without making insertion and deletion more expensive?



https://en.wikipedia.org/wiki/Tree_rotation#/media/File:Tree_rotation.png

113

A quick example

https://www.youtube.com/watch?v=vDHFF4wjWYU

114

Number guessing game

I'm thinking of a number between 1 and n

You are trying to guess the answer

For each guess, I'll tell you "correct", "higher" or "lower"

Describe an algorithm that minimizes the number of guesses

115