Search Trees:
BSTs and B-Trees

David Kauchak
cs302
Spring 2013

Administrative

Proof by contradiction

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

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

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

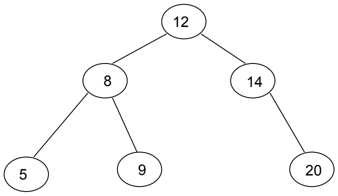
the left and right children are also binary trees

Why not?

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

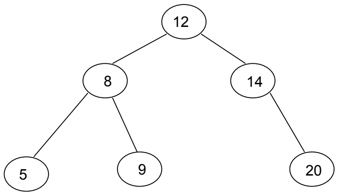
Can be implemented with with pointers or an array

Example



What else can we say?

$$left(i) < i \le right(i)$$



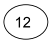
All elements to the left of a node are less than the node

All elements to the right of a node are greater than or equal to the node

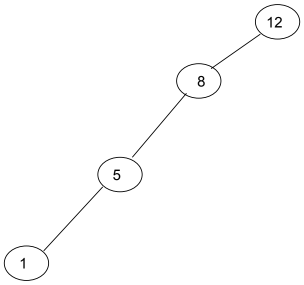
The smallest element is the left-most element

The largest element is the right-most element

Another example: the loner



Another example: the twig



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

Search

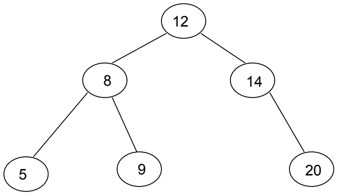
How do we find an element?

```
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)
```

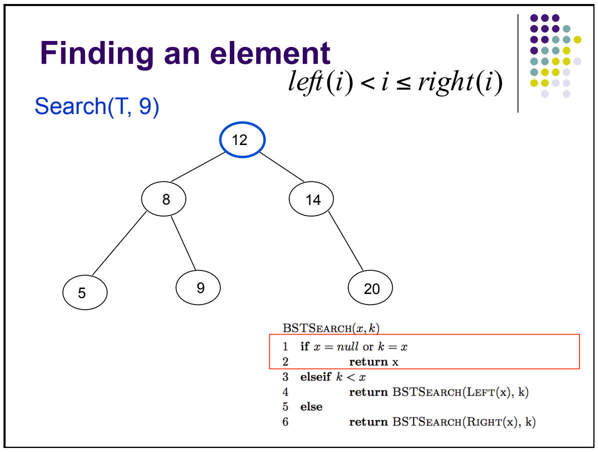
Finding an element

$left(i) < i \le right(i)$

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)
```

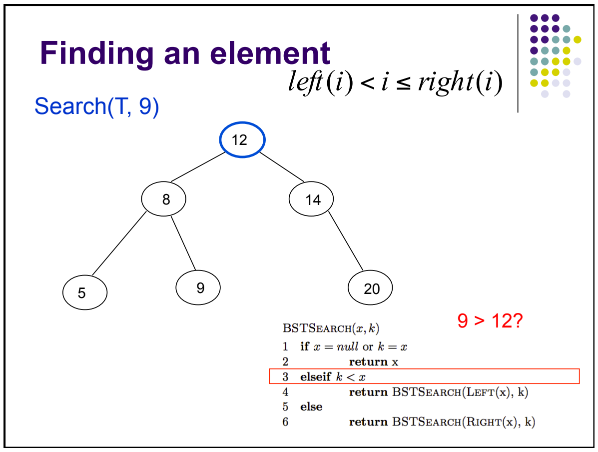


Finding an element

$left(i) < i \le right(i)$

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)
```



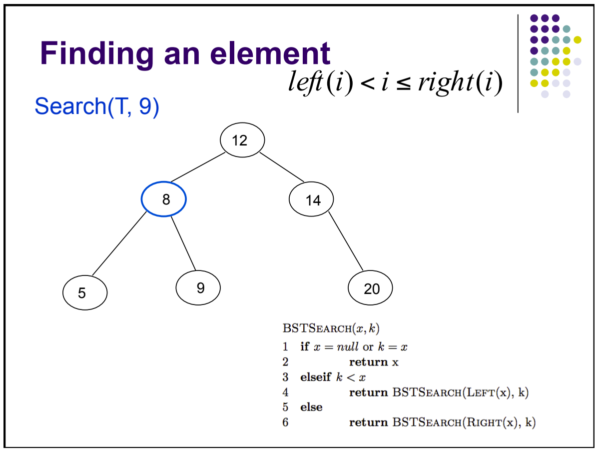
Finding an element

$left(i) < i \le right(i)$

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)
```

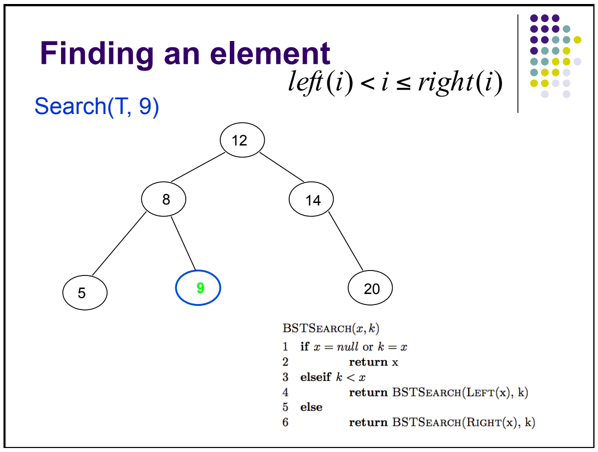


Finding an element

$left(i) < i \le right(i)$

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)
```

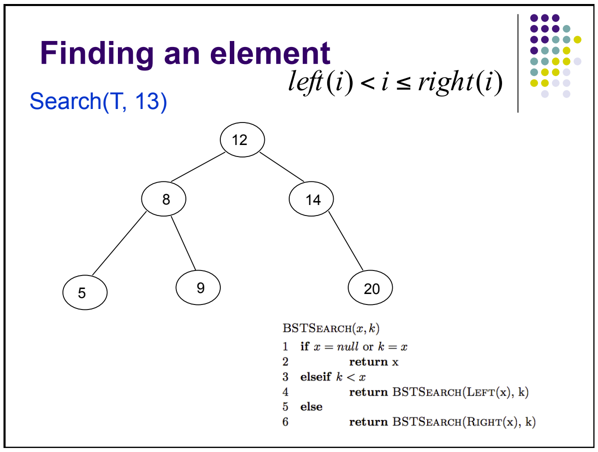


Finding an element

$left(i) < i \le right(i)$

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)
```

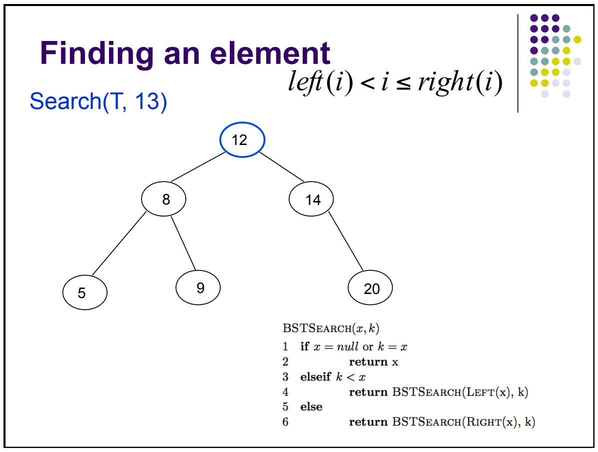


Finding an element

$$left(i) < i \leq right(i)$$

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)
```

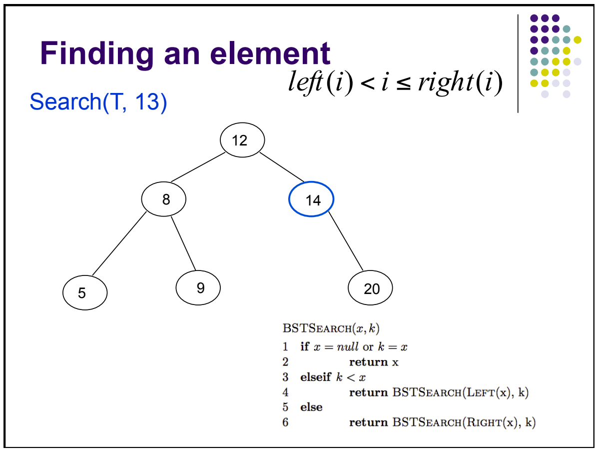


Finding an element

$$left(i) < i \leq right(i)$$

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)
```

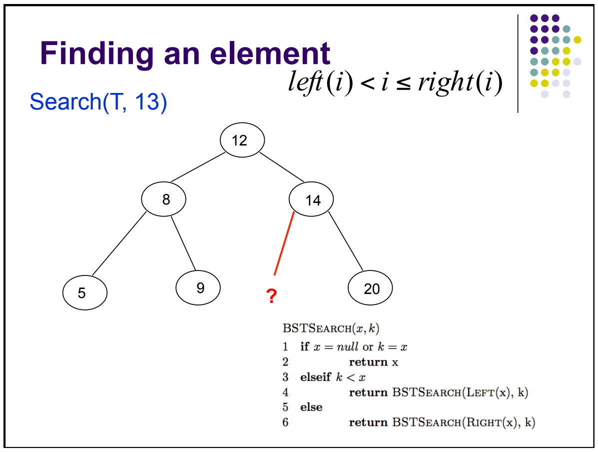


Finding an element

$$left(i) < i \leq right(i)$$

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)
```

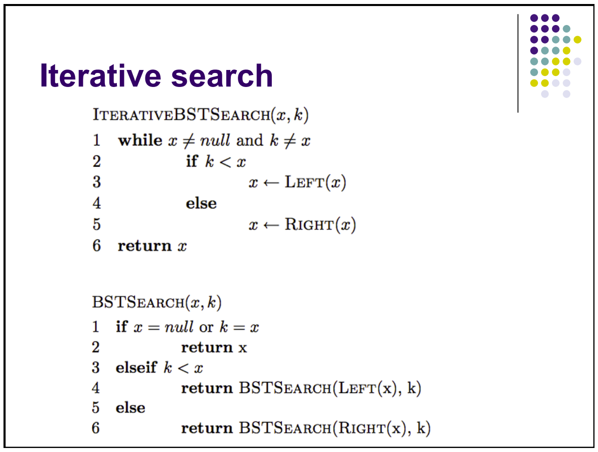


Finding an element

$$left(i) < i \leq right(i)$$

Search(T, 13)

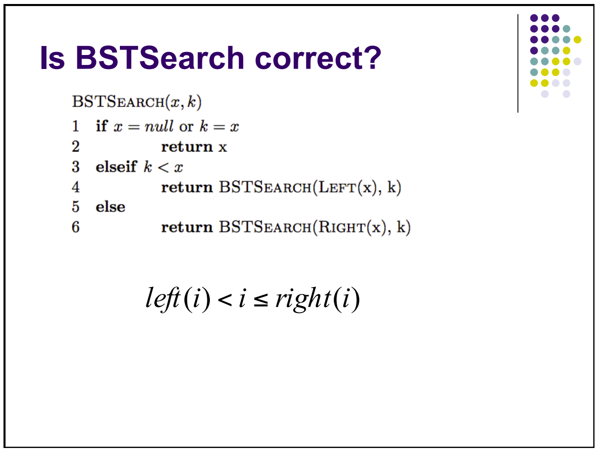
```
BSTSearch($x, k$)
if $x = null$ or $k = x$
    return x
elseif $k < x$
    return BSTSearch(Left(x), k)
else
    return BSTSearch(Right(x), k)
```



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 $x$
```

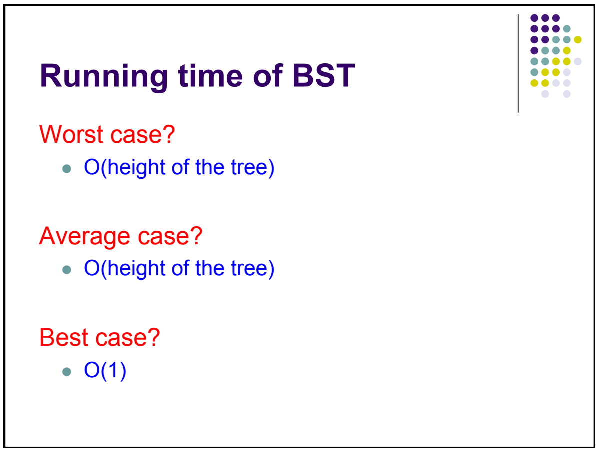
```
BSTSEARCH($x, k$)
if $x = null$ or $k = x$
        return x
elseif $k < x$
        return BSTSEARCH(LEFT(x), k)
else
        return BSTSEARCH(RIGHT(x), k)
```



Is BSTSearch correct?

```
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)
```

$$left(i) < i \leq right(i)$$



Running time of BST

Worst case?

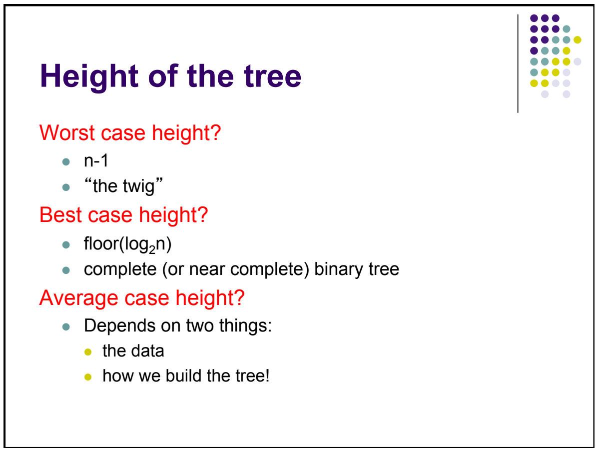
- O(height of the tree)

Average case?

- O(height of the tree)

Best case?

- O(1)



Height of the tree

Worst case height?

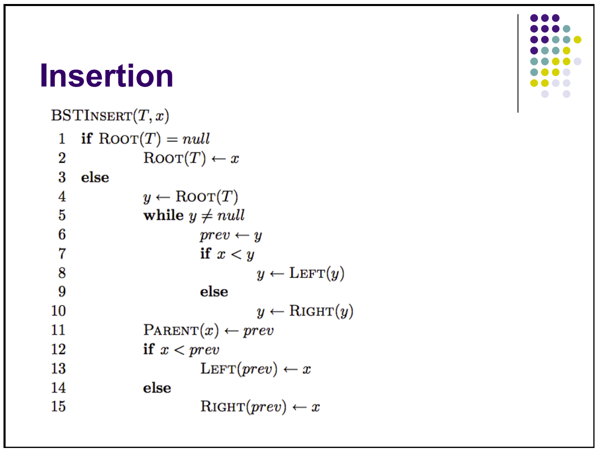
- n-1
- "the twig"

Best case height?

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

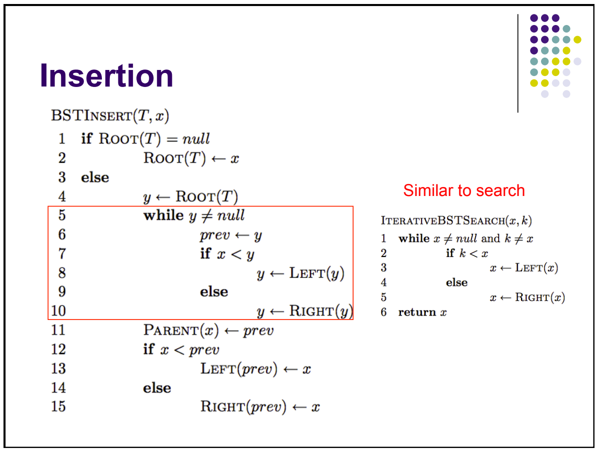
Average case height?

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



Insertion

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

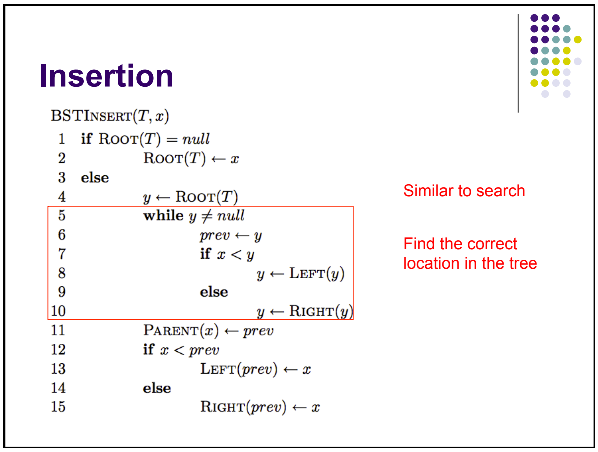


Insertion

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

Similar to search

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

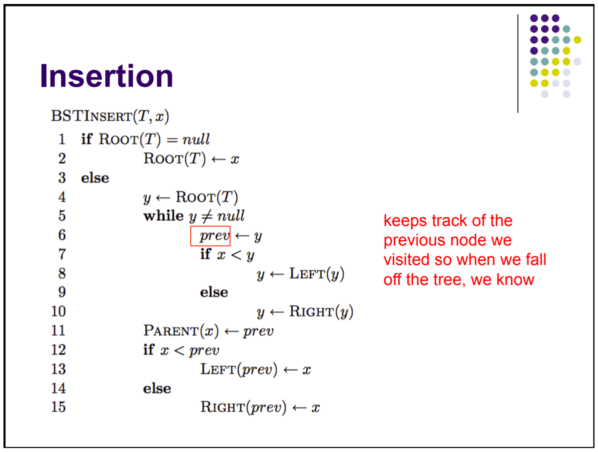


Insertion

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

Similar to search

Find the correct location in the tree



Insertion

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

keeps track of the previous node we visited so when we fall off the tree, we know

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($prev$) $\leftarrow x$
```

add node onto the bottom of the tree

Correctness?

```
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($prev$) $\leftarrow x$
```

maintain BST property

Correctness

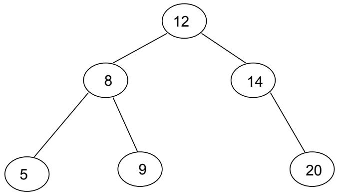
```
BSTInsert($T, x$)
if Root($T$) $= null$
        Root($T$) $\leftarrow x$
else
        $y \leftarrow$ Root($T$)
        while $y \neq null$
                $prev \leftarrow y$
                if $x < y$
                        $y \leftarrow$ Left($y$)
                else
                        $y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

What happens if it is a duplicate?

Inserting duplicate

$$left(i) < i \leq right(i)$$

Insert(T, 14)



Running time

```
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($prev$) $\leftarrow x$
```

O(height of the tree)

Running time

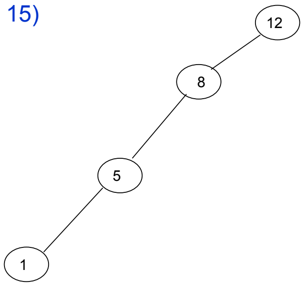
```
BSTInsert($T, x$)
if Root($T$) $= null$
    Root($T$) $\leftarrow x$
else
    $y \leftarrow$ Root($T$)
    while $y \neq null$
        $prev \leftarrow y$
        if $x < y$
            $y \leftarrow$ Left($y$)
        else
            $y \leftarrow$ Right($y$)
    Parent($x$) $\leftarrow prev$
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

O(height of the tree)

Why not Θ(height of the tree)?

Running time

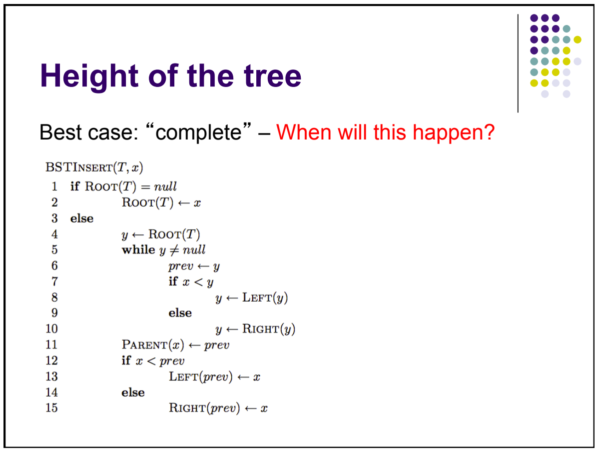
Insert(T, 15)



Height of the tree

Worst case: “the twig” – When will this happen?

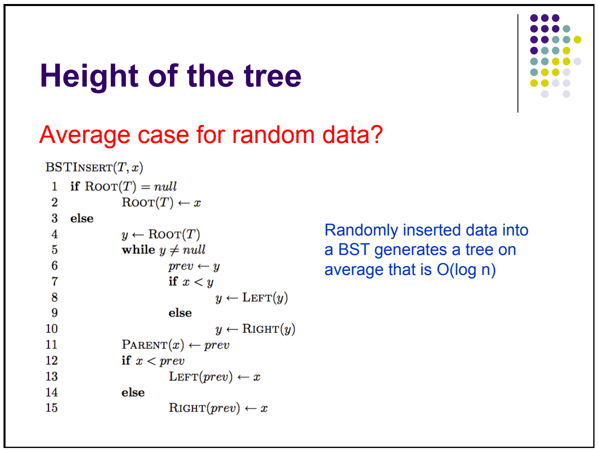
```
BSTInsert($T, x$)
if Root($T$) $= null$
    Root($T$) $\leftarrow x$
else
    $y \leftarrow$ Root($T$)
    while $y \neq null$
        $prev \leftarrow y$
        if $x < y$
            $y \leftarrow$ Left($y$)
        else
            $y \leftarrow$ Right($y$)
    Parent($x$) $\leftarrow prev$
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



Height of the tree

Best case: “complete” – When will this happen?

```
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($prev$) $\leftarrow x$
```

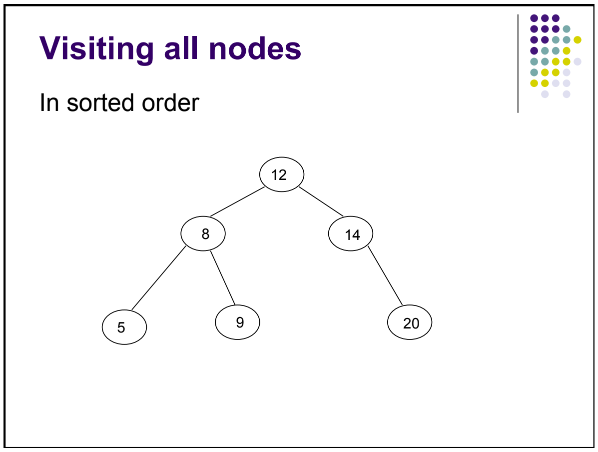


Height of the tree

Average case for random data?

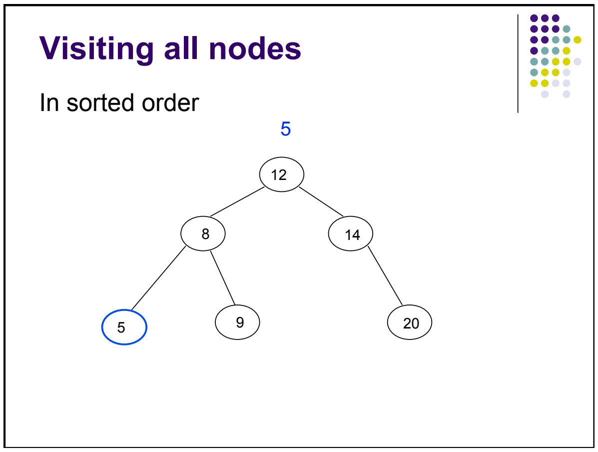
```
BSTInsert($T, x$)
if Root($T$) $= null$
        Root($T$) $\leftarrow x$
else
        $y \leftarrow$ Root($T$)
        while $y \neq null$
                $prev \leftarrow y$
                if $x < y$
                        $y \leftarrow$ Left($y$)
                else
                        $y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

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



Visiting all nodes

In sorted order



Visiting all nodes

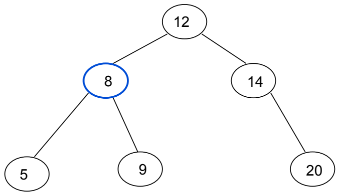
In sorted order

5

Visiting all nodes

In sorted order

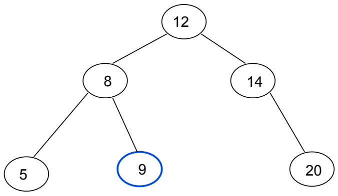
5, 8



Visiting all nodes

In sorted order

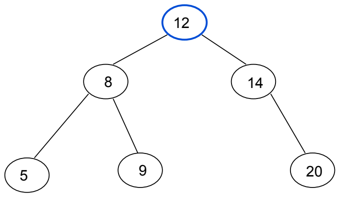
5, 8, 9



Visiting all nodes

In sorted order

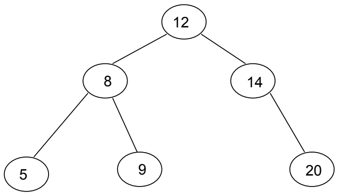
5, 8, 9, 12



Visiting all nodes

What’s happening?

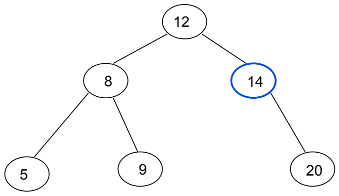
5, 8, 9, 12



Visiting all nodes

In sorted order

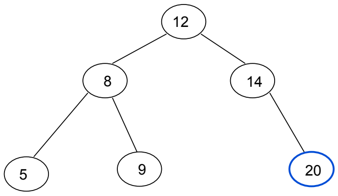
5, 8, 9, 12, 14



Visiting all nodes

In sorted order

5, 8, 9, 12, 14, 20



Visiting all nodes in order

```
InorderTreeWalk($x$)
if $x \neq null$
    InorderTreeWalk(Left($x$))
    print $x$
    InorderTreeWalk(Right($x$))
```

Visiting all nodes in order

```
InorderTreeWalk($x$)
if $x \neq null$
    InorderTreeWalk(Left($x$))
    print $x$
    InorderTreeWalk(Right($x$))
```

any operation

Is it correct?

```
InorderTreeWalk($x$)
if $x \neq null$
    InorderTreeWalk(Left($x$))
    print $x$
    InorderTreeWalk(Right($x$))
```

Does it print out all of the nodes in sorted order?

$$left(i) < i \leq right(i)$$

Running time?

```
InorderTreeWalk($x$)
if $x \neq null$
    InorderTreeWalk(Left($x$))
    print $x$
    InorderTreeWalk(Right($x$))
```

Recurrence relation:

- *j* nodes in the left subtree
- *n* – *j* – 1 in the right subtree

$$T(n) = T(j) + T(n - j - 1) + \Theta(1)$$

Or

- How much work is done for each call?
- How many calls?
- Θ(n)

What about?

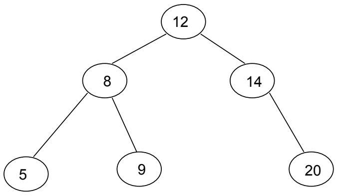
```
TreeWalk(x)
if $x \neq null$
    print $x$
    TreeWalk(Left($x$))
    TreeWalk(Right($x$))
```

Preorder traversal

```
TreeWalk(x)
if $x \neq null$
    print $x$
    TreeWalk(Left($x$))
    TreeWalk(Right($x$))
```

12, 8, 5, 9, 14, 20

How is this useful?



Tree copying: insert in to new tree in preorder

prefix notation: (2+3)*4 -> * + 2 3 4

What about?

```
TreeWalk(x)
if $x \neq null$
    TreeWalk(Left($x$))
    TreeWalk(Right($x$))
    print $x$
```

Postorder traversal

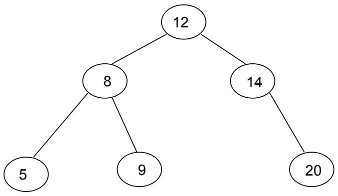
```
TreeWalk(x)
if $x \neq null$
    TreeWalk(Left($x$))
    TreeWalk(Right($x$))
    print $x$
```

5, 9, 8, 20, 14, 12

How is this useful?

postfix notation:
(2+3)*4 -> 4 3 2 + *

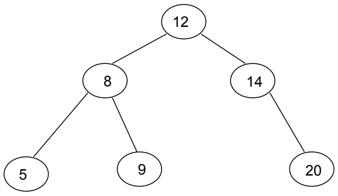
?



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 $x$
```

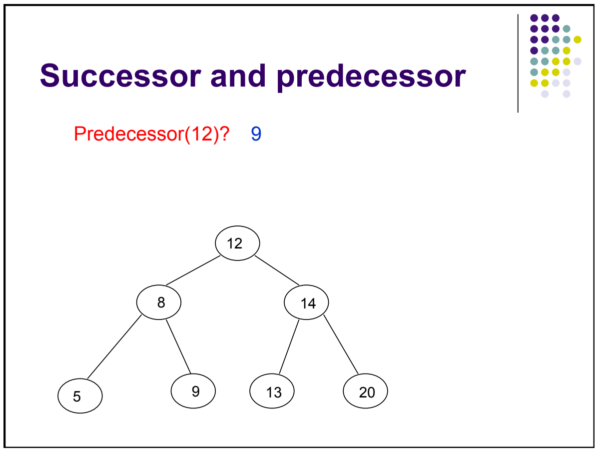


Running time of min/max?

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

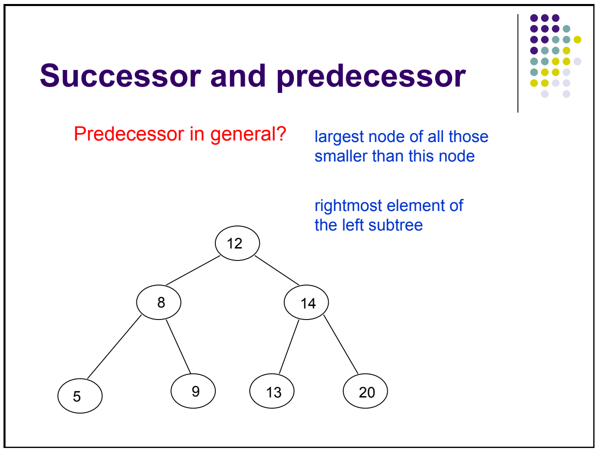
```
IterativeBSTMin($x$)
while Left($x$) $\neq null$
    $x \leftarrow$ Left($x$)
return $x$
```

O(height of the tree)



Successor and predecessor

Predecessor(12)? 9

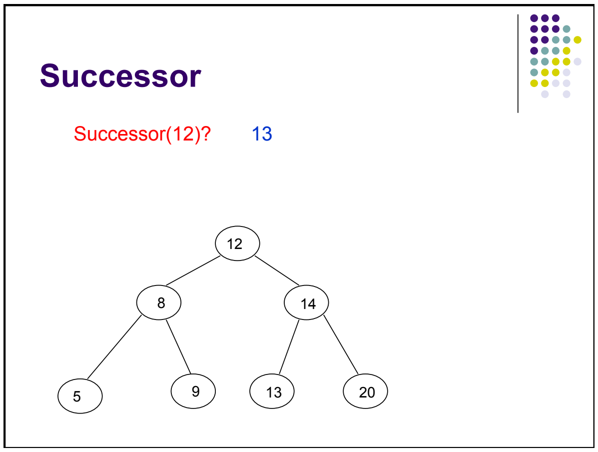


Successor and predecessor

Predecessor in general?

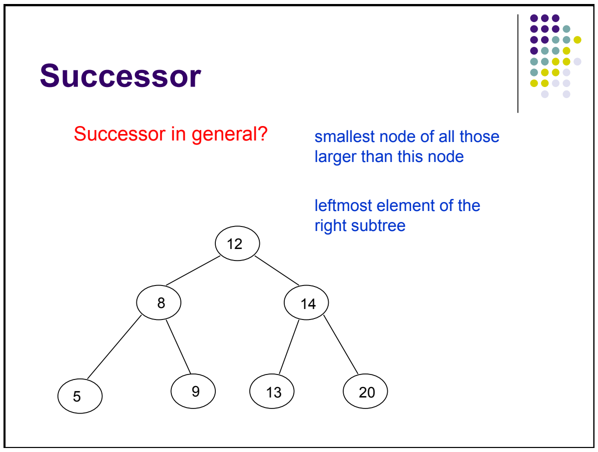
largest node of all those smaller than this node

rightmost element of the left subtree



Successor

Successor(12)? 13

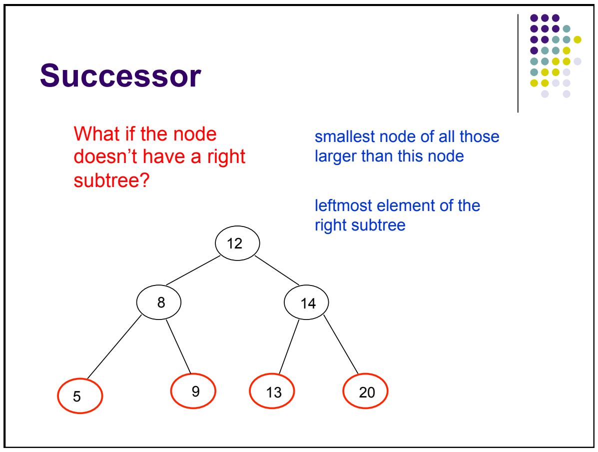


Successor

Successor in general?

smallest node of all those larger than this node

leftmost element of the right subtree

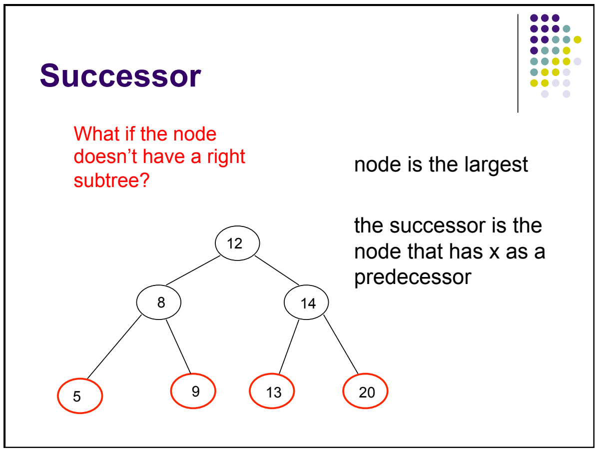


Successor

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

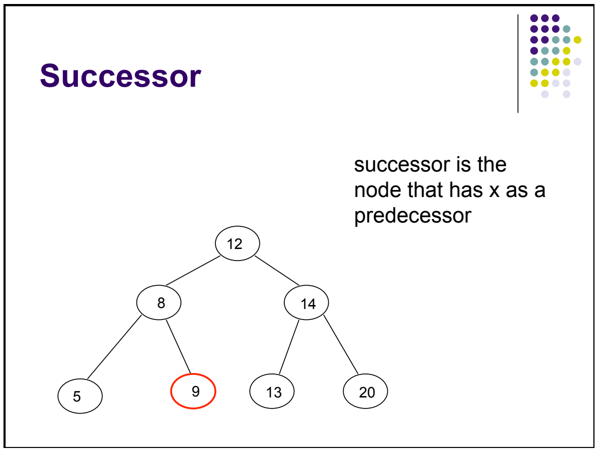


Successor

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

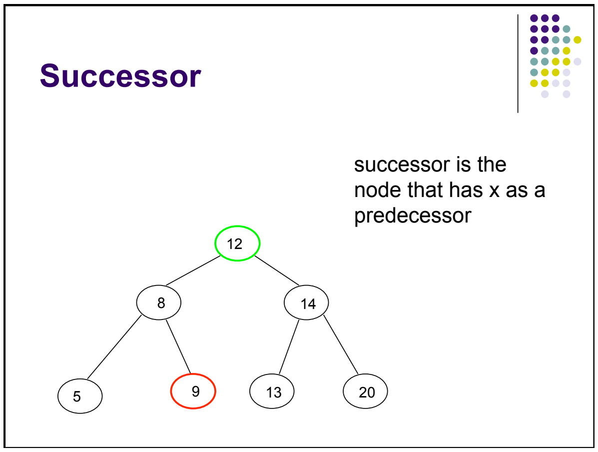
node is the largest

the successor is the node that has x as a predecessor



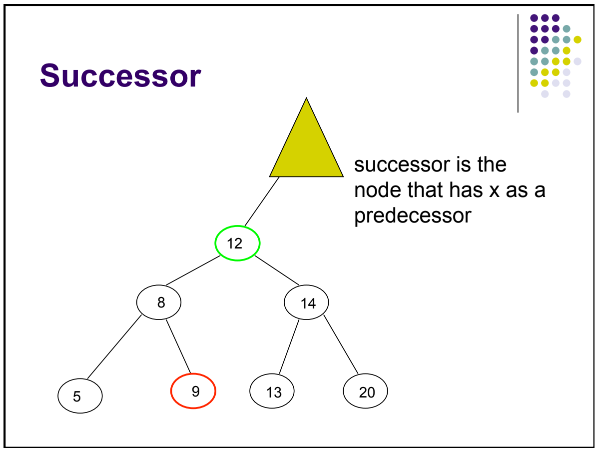
Successor

successor is the node that has x as a predecessor



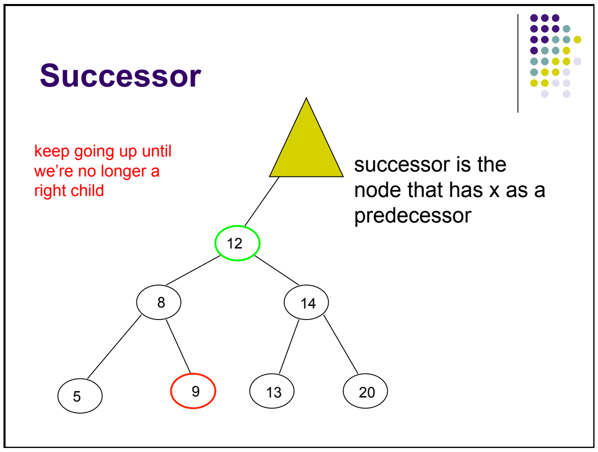
Successor

successor is the node that has x as a predecessor



Successor

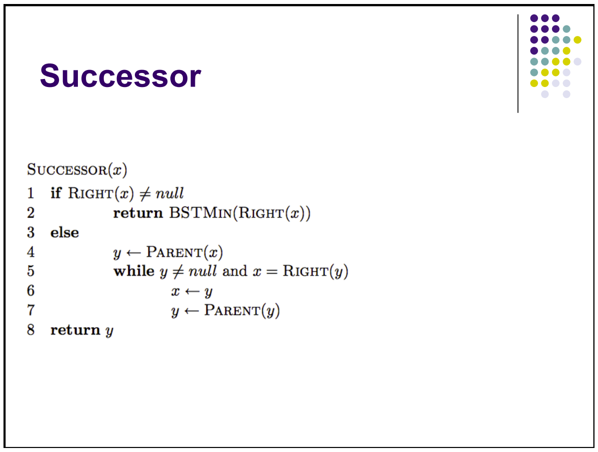
successor is the node that has x as a predecessor



Successor

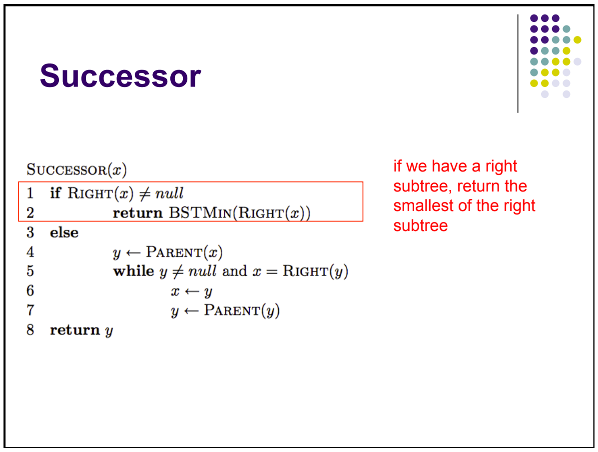
keep going up until we're no longer a right child

successor is the node that has x as a predecessor



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 $y$
```



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

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

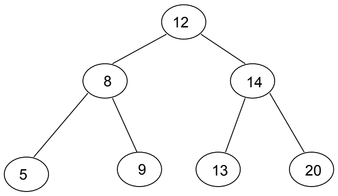
keep going up until we're no longer a right child

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 $y$
```

Deletion

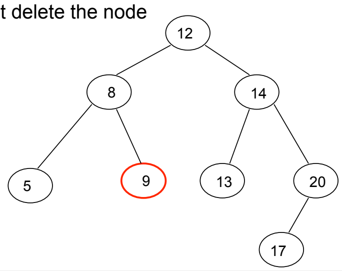


Three cases!

Deletion: case 1

No children

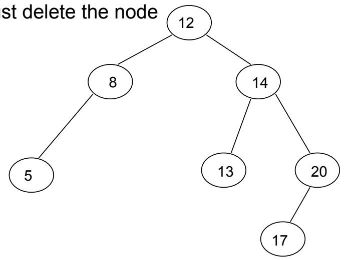
Just delete the node



Deletion: case 1

No children

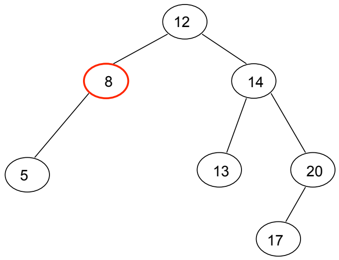
Just delete the node



Deletion: case 2

One child

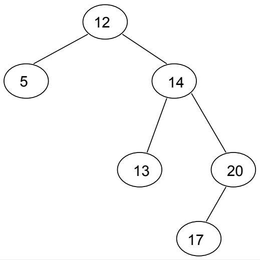
Splice out the node



Deletion: case 2

One child

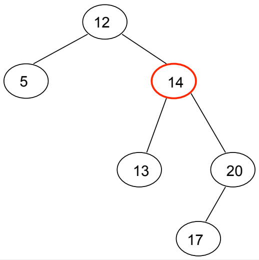
Splice out the node



Deletion: case 3

Two children

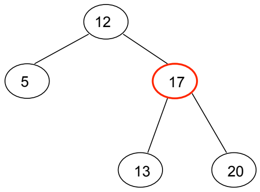
Replace x with it’s successor



Deletion: case 3

Two children

Replace x with it's successor



Deletion: case 3

Two children

Will we always have a successor?

Why successor?

- Case 1 or case 2 deletion
- Larger than the left subtree
- Less than or equal to right subtree

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?

Balanced trees

Make sure that the trees remain balanced!

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

B-trees