3.2 BINARY SEARCH TREES

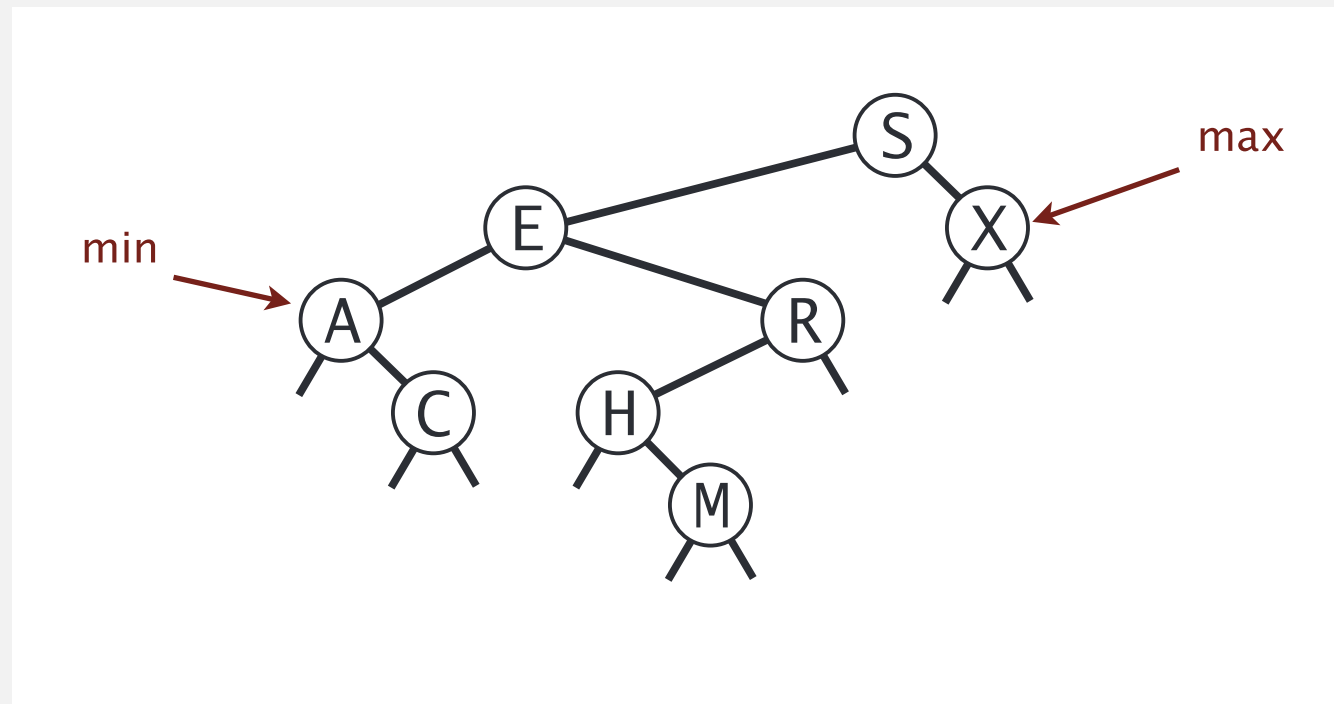▸ *BSTs*

▸ **ordered operations**

▸ *deletion*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

http://algs4.cs.princeton.edu

Minimum and maximum

Minimum. Smallest key in table.

Maximum. Largest key in table.

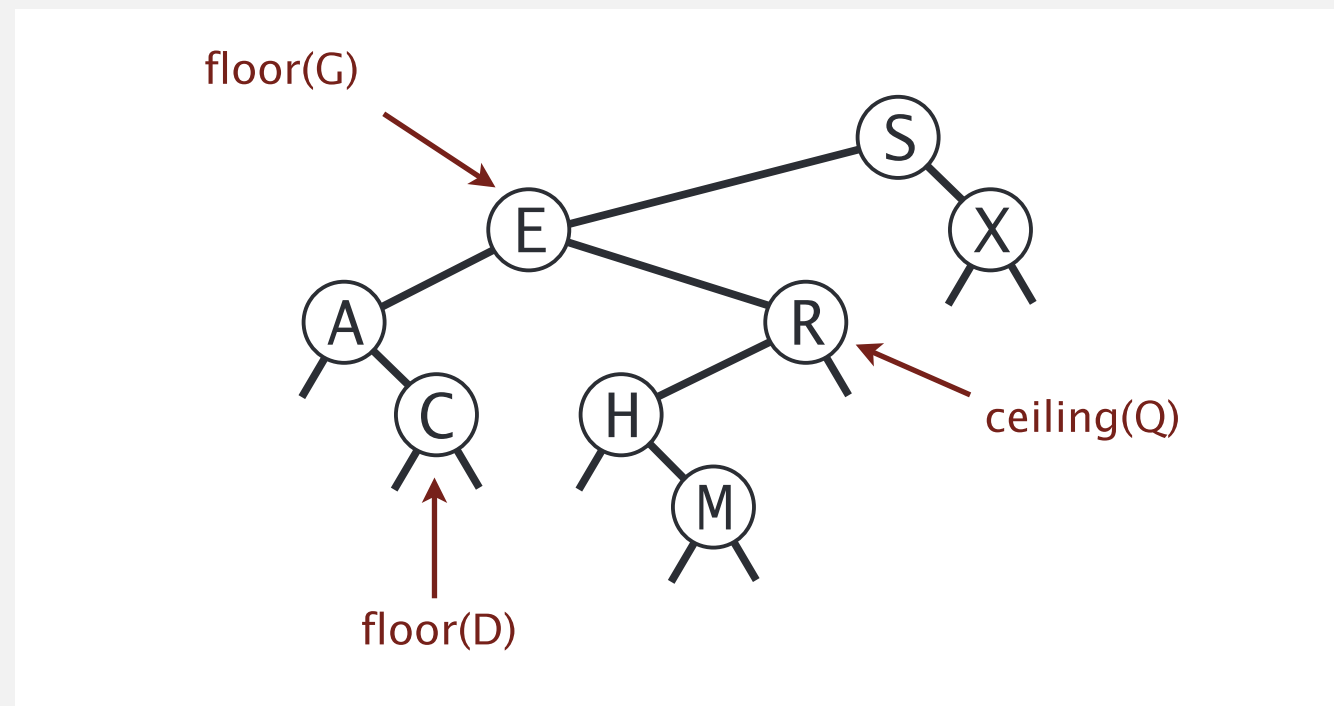
Q. How to find the min / max?

Floor and ceiling

Floor. Largest key ≤ a given key.

Ceiling. Smallest key ≥ a given key.

Q. How to find the floor / ceiling?

Computing the floor

Case 1. [k equals the key in the node]
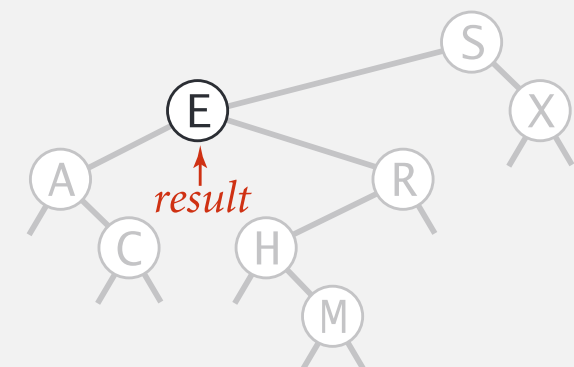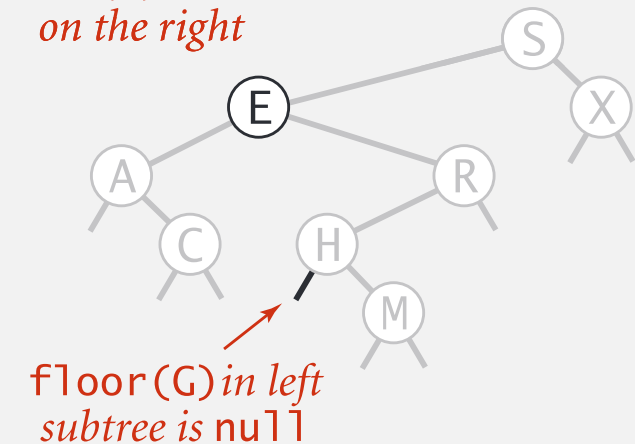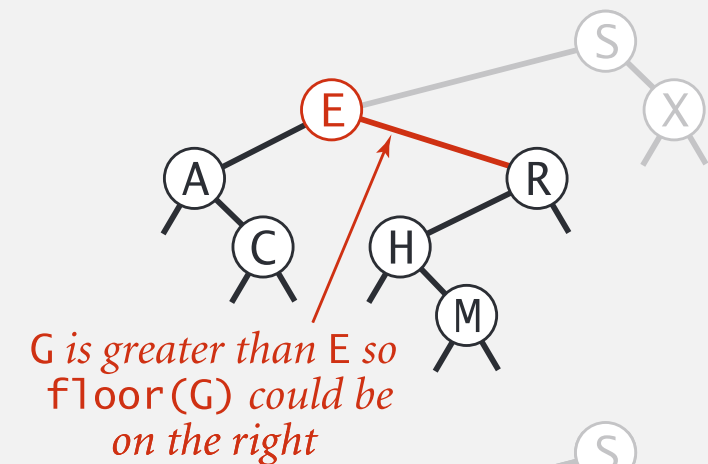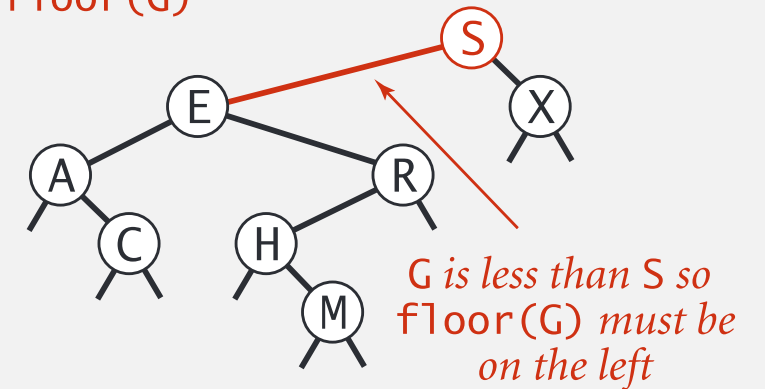
The floor of k is k.

Case 2. [k is less than the key in the node]

The floor of k is in the left subtree.

Case 3. [k is greater than the key in the node]

The floor of k is in the right subtree

(if there is any key $\leq k$ in right subtree);

otherwise it is the key in the node.

finding `floor(G)`

G is less than S so `floor(G)` *must be on the left*

G is greater than E so `floor(G)` *could be on the right*

`floor(G)` *in left subtree is* `null`

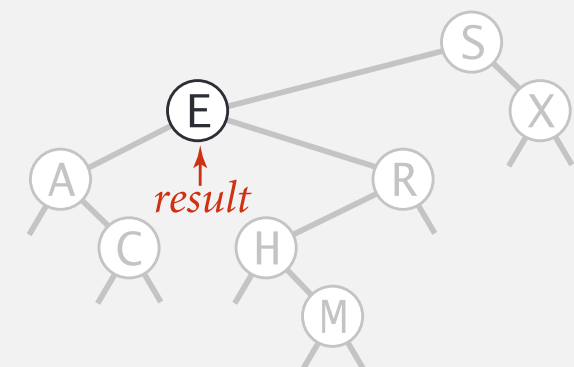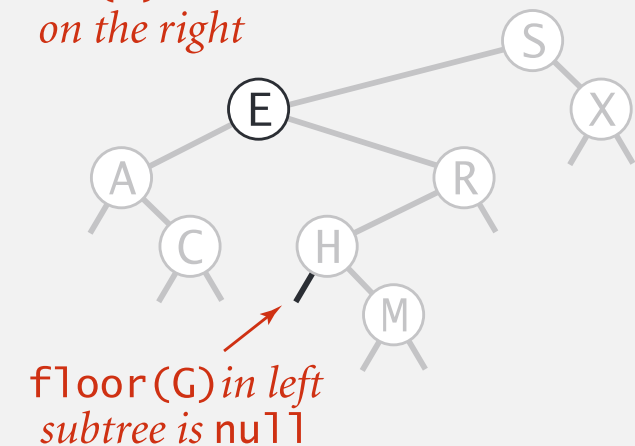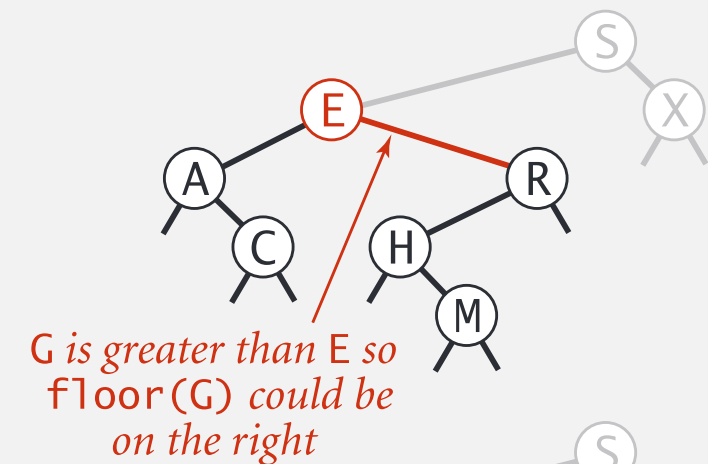*result*

Computing the floor

```
public Key floor(Key key)
{
    Node x = floor(root, key);
    if (x == null) return null;
    return x.key;
}
private Node floor(Node x, Key key)
{

    if (x == null) return null;
    int cmp = key.compareTo(x.key);

    if (cmp == 0) return x;


    if (cmp < 0)  return floor(x.left, key);


    Node t = floor(x.right, key);
    if (t != null) return t;
    else           return x;


}
```

finding `floor(G)`

G is less than S so `floor(G)` *must be on the left*

G is greater than E so `floor(G)` *could be on the right*

`floor(G)` *in left subtree is* `null`

result
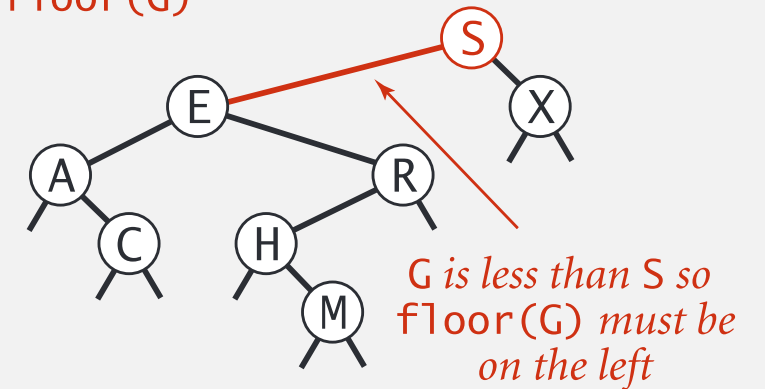
Rank and select

Q. How to implement `rank()` and `select()` efficiently?

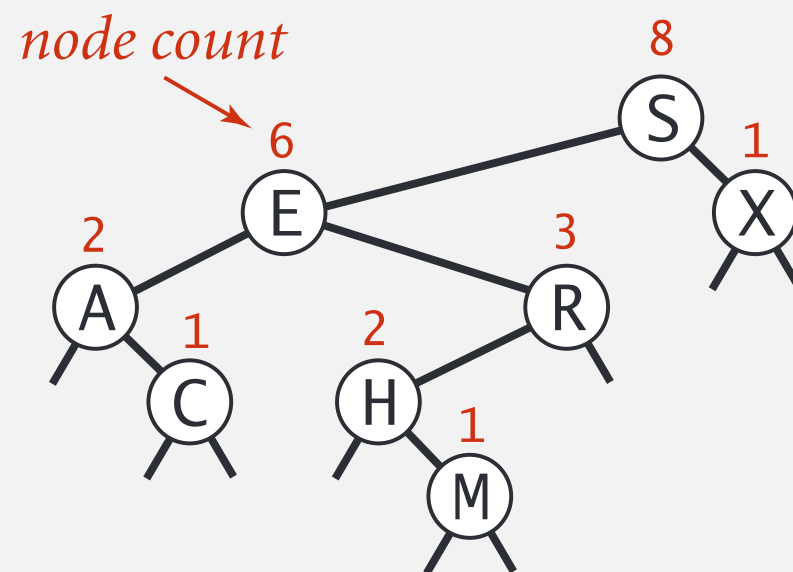A. In each node, we store the number of nodes in the subtree rooted at that node; to implement `size()`, return the count at the root.

node count

BST implementation: subtree counts

```
private class Node
{
    private Key key;
    private Value val;
    private Node left;
    private Node right;
    private int count;
}
```

number of nodes in subtree

```
public int size()
{   return size(root);   }
```

```
private int size(Node x)
{
    if (x == null) return 0;
    return x.count;
}
```

ok to call when x is null

```
private Node put(Node x, Key key, Value val)
{
    if (x == null) return new Node(key, val, 1);
    int cmp = key.compareTo(x.key);
    if      (cmp  < 0) x.left  = put(x.left,  key, val);
    else if (cmp  > 0) x.right = put(x.right, key, val);
    else if (cmp == 0) x.val = val;
    x.count = 1 + size(x.left) + size(x.right);
    return x;
}
```
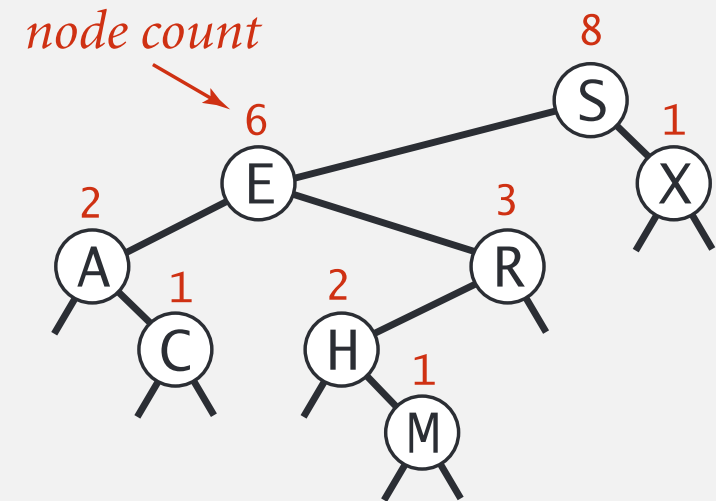
initialize subtree count to 1

Rank

Rank. How many keys $< k$?

Easy recursive algorithm (3 cases!)

node count

8

6

S

1

2

E

3

X

A

1

2

R

C

H

1

M

```
public int rank(Key key)
{   return rank(key, root);   }


private int rank(Key key, Node x)
{
    if (x == null) return 0;
    int cmp = key.compareTo(x.key);
    if       (cmp  < 0) return rank(key, x.left);
    else if (cmp  > 0) return 1 + size(x.left) + rank(key, x.right);
    else if (cmp == 0) return size(x.left);
}
```
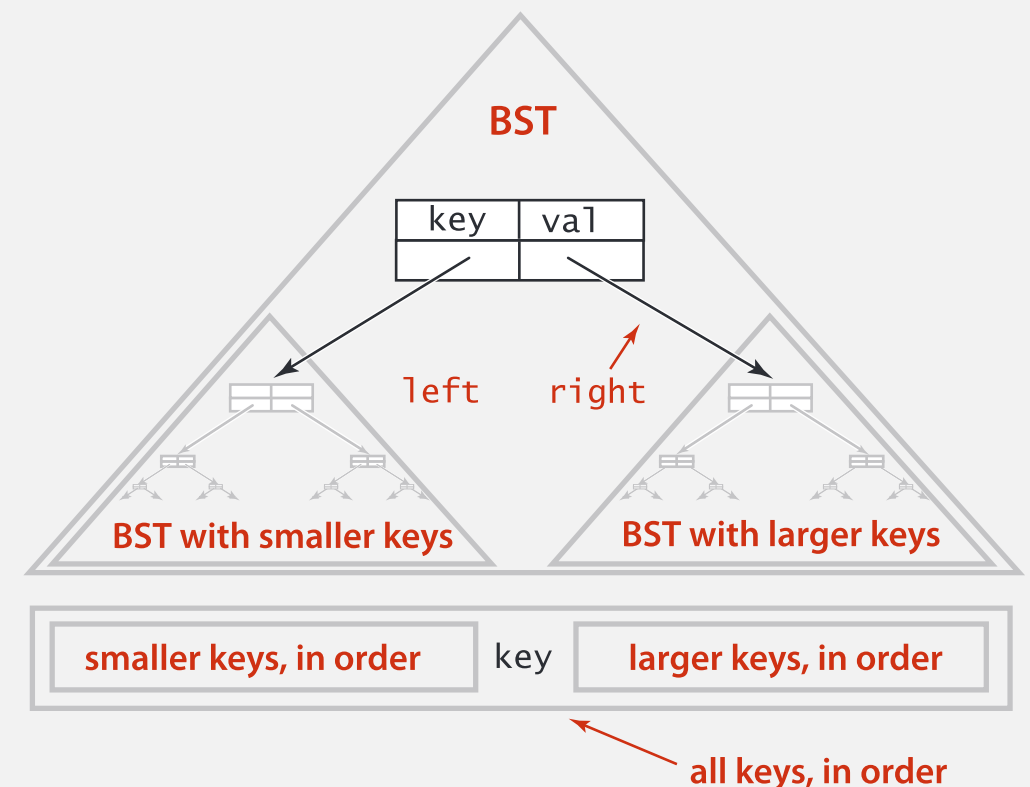
Inorder traversal

- Traverse left subtree.
- Enqueue key.
- Traverse right subtree.

```
public Iterable<Key> keys()
{
    Queue<Key> q = new Queue<Key>();
    inorder(root, q);
    return q;
}

private void inorder(Node x, Queue<Key> q)
{
    if (x == null) return;
    inorder(x.left, q);
    q.enqueue(x.key);
    inorder(x.right, q);
}
```
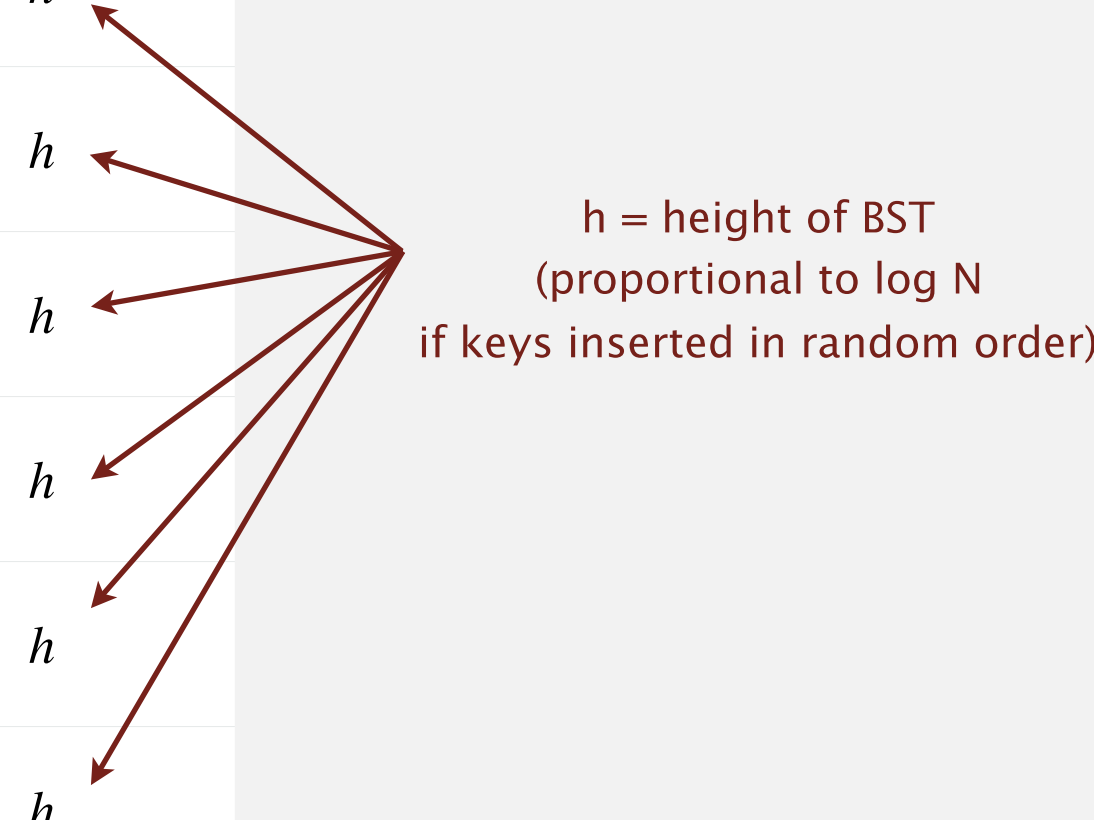

BST

key | val

left   right

BST with smaller keys   BST with larger keys

smaller keys, in order   key   larger keys, in order

all keys, in order

Property. Inorder traversal of a BST yields keys in ascending order.

BST: ordered symbol table operations summary

	sequential search	binary search	BST
search	N	$\lg N$	h
insert	N	N	h
min / max	N	1	h
floor / ceiling	N	$\lg N$	h
rank	N	$\lg N$	h
select	N	1	h
ordered iteration	$N \log N$	N	N

h = height of BST
(proportional to log N
if keys inserted in random order)

order of growth of running time of ordered symbol table operations