```
class Node<K, V> {
  K key;
  V value;
  Node<K, V> left, right;
}

class BSTMap<K,V> implements OrderedDefaultMap<K,V>{

Node<K, V> root;
int size;
Comparator<K> comparator;

...

Node<K, V> set(Node<K, V> node, K key, V value) {
  if (node == null) {
    this.size += 1;
    return new Node<K, V>(key, value, null, null);
  }
  int comp = this.comparator.compare(node.key, key);
  if (comp < 0) {
    node.right = this.set(node.right, key, value);
    return node;
  } else if (comp > 0) {
    node.left = this.set(node.left, key, value);
    return node;
  } else {
    node.value = value;
    return node;
  }
}

@Override
public void set(K key, V value) {
  if (key == null) {
    throw new IllegalArgumentException();
  }
  this.root = this.set(this.root, key, value);
}

}
```

this

@A.set(null, "orange", 5)	
node	null
return @F	

@A.set(@D, "orange", 5)	
node	@D
comp	-1
node.right = ... return node;	

@A.set(@C, "orange", 5)	
node	@C
comp	1
node.left = ... return node;	

@A.set(@B, "orange", 5)	
node	@B
comp	-1
node.right = (...) return node;	

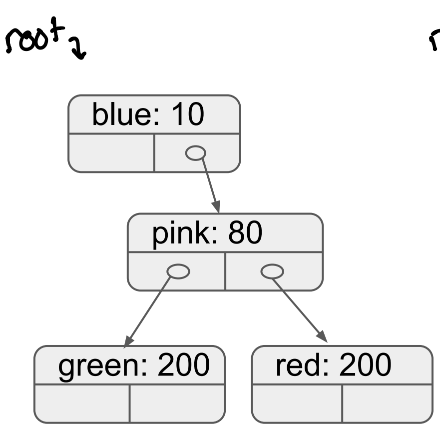
this

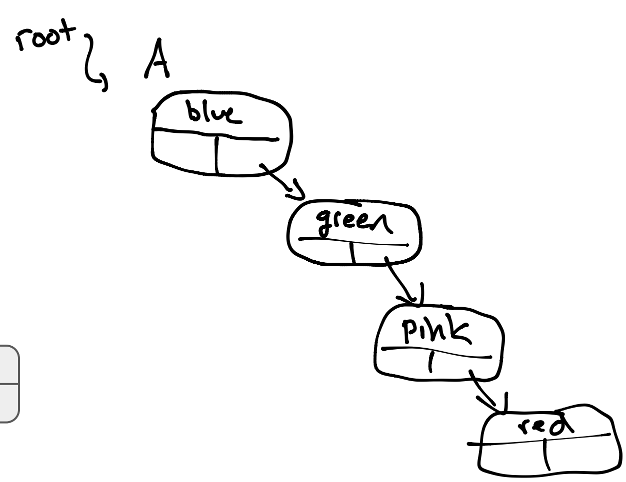
@A.set("orange", 5)	
key	"orange"
value	5
this.root = (...)	

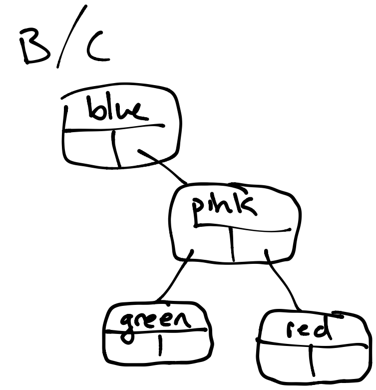
@A	BSTMap root = ~~@B~~ @B size = ~~4~~ 5 comparator = String::compare
@B	Node key = "blue" value = 10 left = null right = ~~@C~~ @C
@C	Node key = "pink" value = 80 left = ~~@E~~ @D right = @E
@D	Node key = "green" value = 200 left = null right = ~~null~~ @F
@E	Node key = "red" value = 200 left = null right = null
@F	Node Key = "orange" Value = 5 l = null r = null

Based on set() above, what order should we add elements to an empty tree to get the below?

A: blue, green, pink, red
B: blue, pink, green, red
C: blue, pink, red, green
D: red, pink, green, blue
E: More than one of these works







Definition: A **binary search tree (BST)** is a tree where at **every** node, all keys to the **left** of that node are **smaller** than that key, and all keys to the **right** are larger.

```
class Node<K, V> {
  K key;
  V value;
  Node<K, V> left, right;
}

class BSTMap<K,V> implements OrderedDefaultMap<K,V>{
```

Worst case:
$h = n$

```
int height() {

}
```

Best case:
$h = \lg(n) + 1$

CSE 100:
balanced trees

set is $O(h)$

```
void pAE(Node<K,V> n) {
  if (node == null) { return; }
  s.o.p(n.key);
  pAE(n.left);
  pAE(n.right);
}
```

```
void printAllElements() {

}

}
```

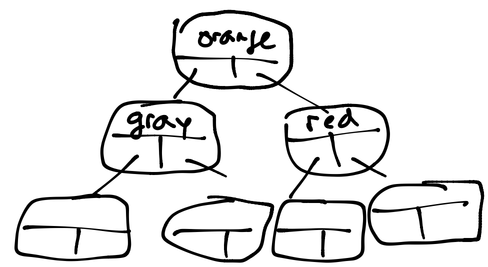
Definition: the **height** of a tree is the number of nodes on the **longest** path from the root to the bottom (or to a **leaf**).

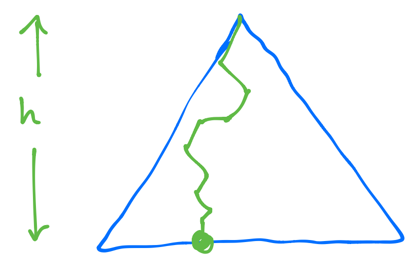
The example on the front has height **3**. After we add "orange" it has height **4**.

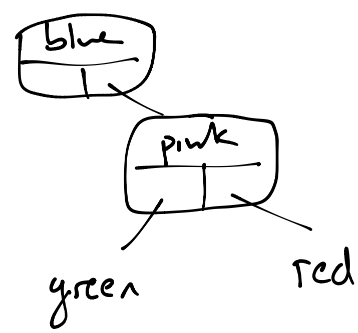
Consider adding "blue", "pink", "orange", "red", "green", "gray", and "yellow" to an empty tree. What is the smallest and largest height possible? [Which order gives these results?]

A: smallest: 4, largest: 6
B: smallest: 3, largest: 7
C: smallest: 4, largest: 7
D: smallest: 2, largest: 7
E: smallest: 3, largest: 6

sorted order gives $h = 7$







blue, pink, green, red

- Exam in-class Wed
- PA7 due 1 week from today