Dictionaries are the **more general** structure, but, in terms of implementation, they're nearly identical.

$k_1, k_2, k_3 \ldots$

$(k_1, v_1), (k_2, v_2)$

Dictionary Implementations, Take # 1

For each of the following potential implementations, what is the worst case runtime for `insert`, `find`, `delete`?

	insert	find	delete
Unsorted Array	$O(n)$	$O(n)$	$O(n)$
Unsorted Linked List (back)	$O(n)$	$O(n)$	$O(n)$
Sorted Linked List	$O(n)$	$O(n)$	$O(n)$
Sorted Array List	$O(n)$	$O(\lg n)$	$O(n)$

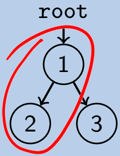
It turns out there are **many** different ways to do much better.

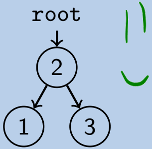
But they all have their own trade-offs!

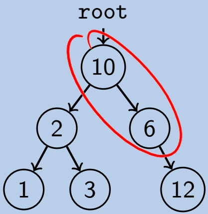
So, we'll study many of them:

- "Vanilla BSTs" – today (vanilla because they're "plain")
- "Balanced BSTs" – there are many types: we'll study **AVL Trees**
- "B-Trees" – another strategy for **a lot of data**
- "Hashtables" – a completely different strategy (lack data ordering)

Example (Which of the following are BSTs?)



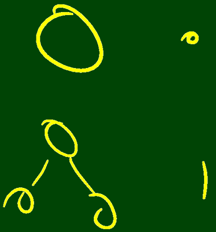




Definition (Height)

The **height** of a binary tree is the length of the longest **path** from the root to a leaf.

- Height of an empty tree?



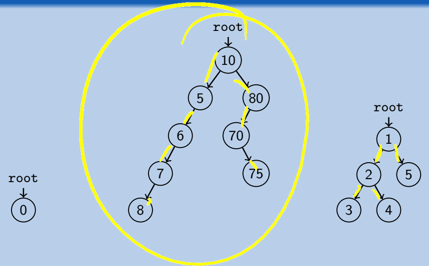
Height of a Binary Tree

Definition (Height)

The **height** of a binary tree is the length of the longest **path** from the root to a leaf.

- Height of an empty tree? **-1**
- Height of $\otimes$? **0**

`height`



4

2

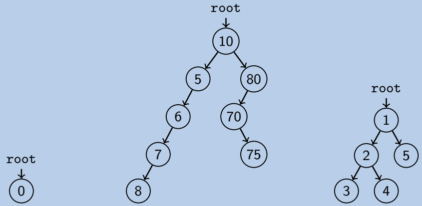
Height of a Binary Tree

Definition (Height)

The **height** of a binary tree is the length of the longest **path** from the root to a leaf.

- Height of an empty tree? **-1**
- Height of x? **0**

height



height is 0 height is 4 height is 2

```java
private int height(Node current) {
   if (current == null) { return -1; }
   return 1 + Math.max(height(current.left), height(current.right));
}
```

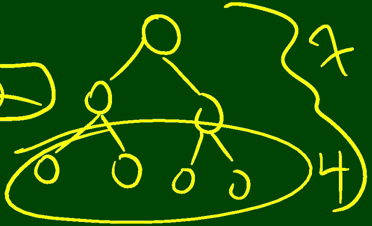
Height

```
private int height(Node current) {
    if (current == null) { return −1; }
    return 1 + Math.max(height(current.left), height(current.right));
}
```

Given that a tree has height h...

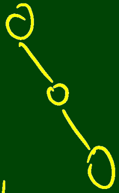
- What is the maximum number of **leaves**? 2^h
- What is the maximum number of **nodes**? $2^{h+1} - 1$
- What is the minimum number of **leaves**? 1
- What is the minimum number of **nodes**?

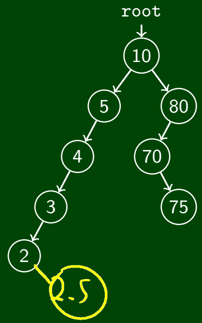
$h = \boxed{2}$



$h = \boxed{2}$

$\sum_{i=0}^{h} 2^i = 2^{h+1} - 1$





What about other `find`s?

- `findMin?`
- `findMax?`
- `deleteMin?`

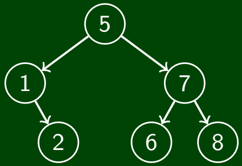
Recursive `find`

```
Data find(Key key, Node curr) {
   if (curr == null) { return null; }
   if (key < curr.key) {
      return find(key, curr.left);
   }
   if (key > curr.key) {
      return find(key, curr.right);
   }
   return curr.data;
}
```

Iterative `find`

```
Data find(Key key) {
   Node curr = root;
   while (curr != null && curr.key != key) {
      if (key < curr.key) {
         curr = curr.left;
      }
      else (key > curr.key) {
         curr = curr.right;
      }
   }
   if (curr == null) { return null; }
   return curr.data;
}
```

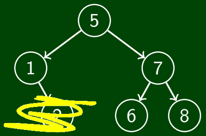
Consider the following tree:

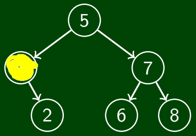


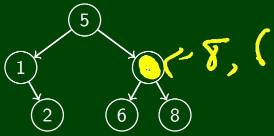
Let's try the following removals:

- `tree.delete(2)`
- `tree.delete(1)`
- `tree.delete(7)`
- `tree.delete(5)`

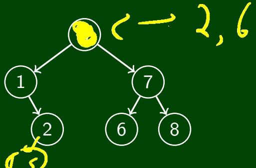
`delete` from a BST

`tree.delete(2)`  ⟶

`tree.delete(1)`  ⟶

`tree.delete(7)` 

⟶

`tree.delete(5)` 

⟶

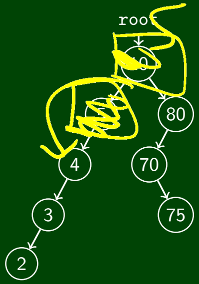
`delete` from a BST

`tree.delete(2)`

`tree.delete(1)`

`tree.delete(7)`

`tree.delete(5)`



delete(x)

- Case 1: x is a leaf
  - Just delete x
- Case 2: x has one child
  - Replace x with its child
- Case 3: x has two children
  - Replace x with the **successor** or **predecessor** of x

The tricky case is when x has two children. If we think of the BST in sorted array form, to get the successor, we `findMin`(right subtree) (or predecessor is `findMax`(left subtree))