

CS 61B Lab 11  
April 10-11, 2014

Please make sure you have a partner for this lab.

Copy the Lab 11 directory by doing the following, starting from your home directory.

```
cp -r ~cs61b/lab/lab11 .
```

All the code is in the dict package. You can compile it from your lab11 directory with

```
javac -g dict/*.java
```

Some test code is provided and can be run with

```
java dict.BinaryTree
```

Familiarize yourself with the fields and methods of the `BinaryTree` and `BinaryTreeNode` classes. `BinaryTree` is an implementation of the `Dictionary` interface (which you also used in Homework 6) as **a binary search tree**. Note that **`BinaryTree` uses `Comparable` objects as keys** (since it is an ordered dictionary), whereas in the `Dictionary` abstract class, keys are declared as plain `Objects`. Thus **you will occasionally need to cast `Objects` to `Comparables`**.

#### Part I: Finding an Element in a Binary Search Tree (2 points)

-----

Complete the implementation of the `find()` method in `dict/BinaryTree.java` by filling in the body of `findHelper()`. `find()` takes a key as its single parameter, and returns an element associated with that key, or null if there is none. (If there are several elements associated with a key, it doesn't matter which one is returned.) `findHelper()` helps by recursively finding and returning a node that contains the key (or null if no such node exists).

Take a look at `insertHelper()` for inspiration. `find()` should run in  $O(d)$  time on a tree with depth  $d$ .

#### Part II: Removing an Element with a Given Key (2 points)

-----

Fill in the body of `remove()` method in `BinaryTree.java`. `remove()` takes a key as its single parameter, and removes one item having that key if the tree contains one. (If there are several elements associated with a key, it doesn't matter which one is removed and returned.) `remove()` returns the same value that `find()` returns, but should not call `find()`. **However, `remove()` SHOULD use the `findHelper()` method you wrote for Part I.**

`remove()` should run in  $O(d)$  time if the depth of the tree is  $d$ .

#### Check-off

-----

Show your TA or Lab Assistant the code you have written, and run the test program. You'll receive points for each part that runs correctly.

2 points: If `find()` works correctly.

1 point: If `remove()` works for nodes that have no child or one child.

1 point: If `remove()` works for nodes that have two children.