Activity #8: Binary Search Trees
Recorder's Report

Manager:	Reader:
Recorder:	Driver:
Date:	Score: Satisfactory / Not Satisfactory

Record your team's answers to the key questions (marked with key) below.

(a) Model 2, Question #7, 8

(b) Model 2, Question #10

(c) Model 2, Question #13

(d) Model 3, Question #17

Activity #8: Binary Search Trees

Binary search trees allow binary search for fast lookup, addition, and removal of data items, and can be used to implement dynamic sets and lookup tables. Since the nodes in a BST are laid out in such a way that each comparison skips about half of the remaining tree, the lookup performance is proportional to that of binary logarithm.

Content Learning Objectives

After completing this activity, students should be able to:

- Explain a binary search tree (BST)
- Knowledge of the rules for a BST

Process Skill Goals

During the activity, students should make progress toward:

- Write code that adds and accesses a BST

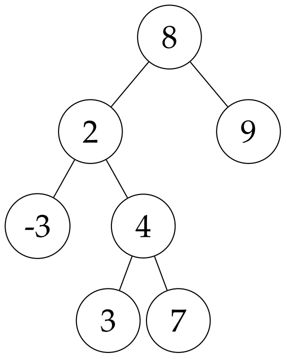
Model 1 BST Basics

Questions (20 min) Start time:

A **BINARY SEARCH** tree is a binary tree in which the data (keys) are stored in order such that all nodes to the right of node N have keys bigger than N and all nodes to the left of node N have keys smaller than N.

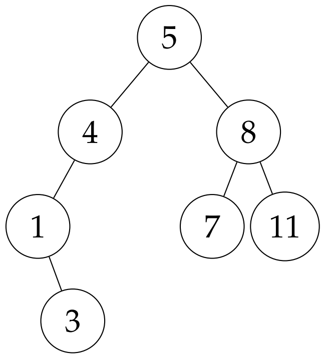
BSTs give us a good data structure to implement the dictionary ADT (insert, find, delete, create, print).

Here is a simple BST where the keys are ints:

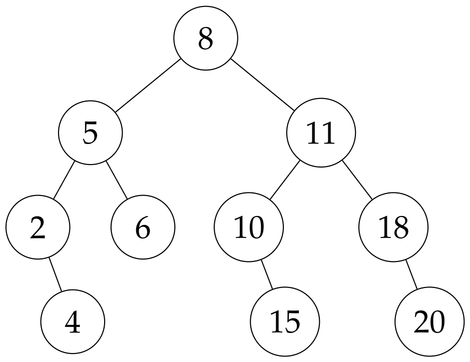


- Choose any node in the tree.
- Its left subtree descendants are less than the node's value.
- Its right subtree descendants are greater than the node's value.

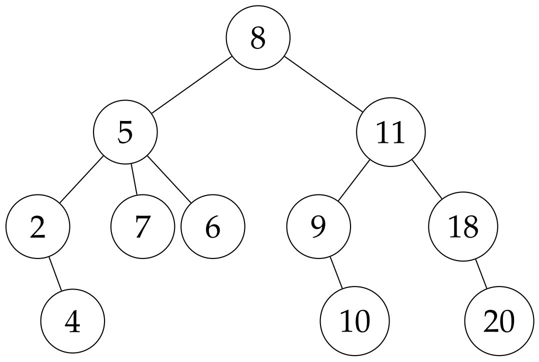
1. Is this a valid binary search tree? yes no If no, explain why.



2. Is this a valid binary search tree? yes no If no, explain why.

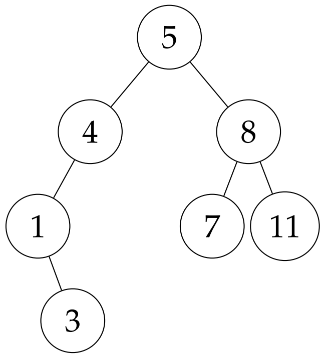


3. Is this a valid binary search tree? yes no If no, explain why.



Inserting new nodes

How do we insert new nodes into a BST?



4. Suppose we want to insert the value 6. Where does it go?

5. Now, we want to insert 0. Where does it go?

6. Now, we want to insert 9. Where does it go?

Inserting into a BST is quite simple. Insertions happen at the leaves. Here is an iterative version:

```c
/* insert
 * inserts data item d into tree; note that this is a BST so it is ordered
 */
void insert(TreeData d, TreeNode **tptr) {
    // create new node for data
    TreeNode *toInsert = newTreeNode(d);
    TreeNode *curr = *tptr;
    if (curr == NULL) {
        *tptr = toInsert; // make this the tree
        return;
    }
    // check value of t to see if new node should be to the right or left of curr
    while (curr != NULL) {
        if (d < curr->value) { // goes to left
            if (curr->left == NULL) {
                curr->left = toInsert;
                return;
            }
            // keep going left
            curr = curr->left;
        } else { // goes to right
            if (curr->right == NULL) {
                curr->right = toInsert;
                return;
            }
            // keep going right
            curr = curr->right;
        }
    }
}
/* newTreeNode
 * helper function, creates a new tree node with value d
 * returns the address of the new node
 */
TreeNode *newTreeNode(TreeData d) {
    TreeNode *toReturn = (TreeNode *)malloc(sizeof(TreeNode));
    toReturn->value = d;
    toReturn->left = NULL;
    toReturn->right = NULL;
    return toReturn;
}

```

Here is a recursive version to insert an item:

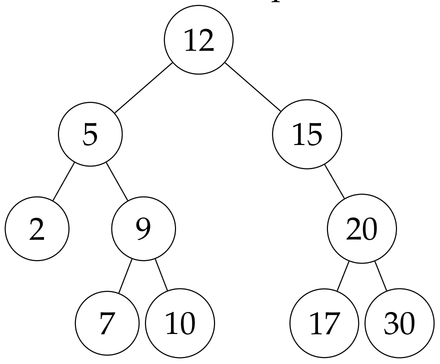
```c
/* insertR (this function is written recursively)
 * inserts data item d into tree; note that this is a BST so it is ordered
 */
void insertR(TreeData d, TreeNode **tptr) {
    if (*tptr == NULL) {
        *tptr = newTreeNode(d);
    } else if (d < (*tptr)->value) {
        insertR(d, &(*tptr)->left);
    } else {
        insertR(d, &(*tptr)->right);
    }
}

```

Model 2 Delete Node

Questions (10 min) Start time:

The final dictionary operation that we need to examine is delete. Given the tree below, how would you delete each of the nodes (assume the deletions are independent, so you are starting with the same tree prior to each deletion).



7. How would you delete 7?

8. How would you delete 15?

9. How would you delete 5?

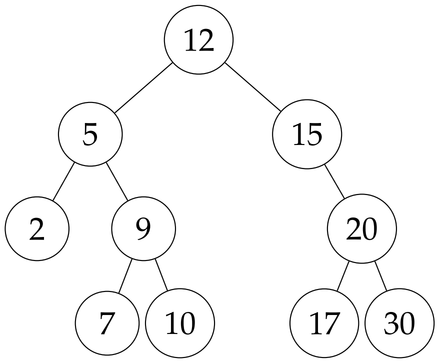
In general, here is the strategy for deletion:

Delete(D, T):
If Find(D, T) is false, do nothing.
If T is a leaf node, delete it and update its parent to point to null instead of T.

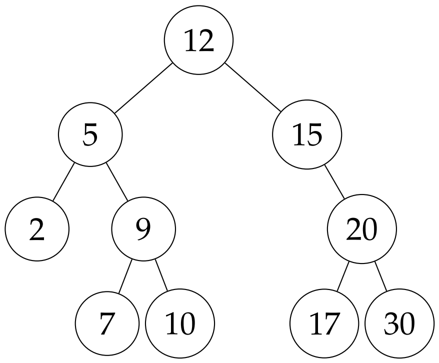
If T is an interior node and T has just a right child, delete T and update its parent to point to T's right child.
If T is an interior node and T has just a left child, delete T and update its parent to point to T's left child.
Else (T is interior with 2 children):

Find the next successor of T by traversing to T's right child and then going all the way to the leftmost leaf. This leftmost leaf is the next largest item in the tree. Copy the value of this leftmost leaf to T. If leftmost leaf does not have a right subtree, delete leftmost leaf with same procedure as leaf node above. If leftmost leaf has a right subtree, then delete with the same procedure as interior node with just a right child.

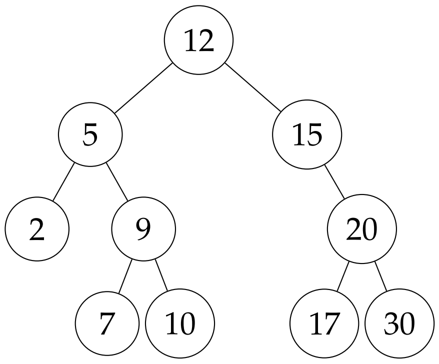
10. Delete node 5 with procedure above. Cross out nodes that are deleted and values that are updated.



11. Delete node 10 with procedure above. Cross out nodes that are deleted and values that are updated.



12. Delete node 15 with procedure above. Cross out nodes that are deleted and values that are updated.



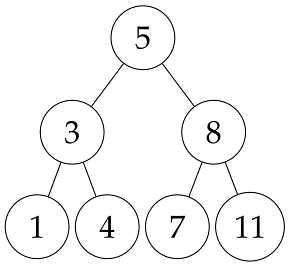
Even though we are modeling BSTs with nodes having just one value, a (key, value) pair could be stored at each node, with the keys used as the comparison values when inserting, finding, and deleting.

13. Does your group have any questions about binary search trees?

Model 3 Creating a Tree

Questions (20 min) Start time:

Finding keys
Now, how would we find an element in the tree?



Let's find 7. Start with the root. If the item is equal to 7, return true (or a pointer to this item). If the item you are looking for is > than the root, treat right subtree as root. Otherwise, treat left subtree as root. Keep applying this procedure until you hit a leaf.

14. What nodes are examined when looking for 7?

15. Now, look for 4. What nodes are examined when looking for 4?

You will implement the find function in lab.

Creating a new tree

Creating a new tree is pretty straightforward. A tree with no items is NULL.

```
    TreeNode * tree = NULL;

```

To instantiate a tree with a list of items, we could do this:

```
/* createTree
 * creates a binary search tree with data stored in array a
 */
TreeNode *createTree(TreeData a[], int size) {
    if (size <= 0) {
        return NULL;
    }
    TreeNode *toReturn = newTreeNode(a[0]); // insert first item from list
    int i;
    for (i = 1; i < size; i++) {
        insert(a[i], &toReturn);
    }
    return toReturn;
}
```

An optional dictionary operation is size. Here is an implementation of size:

```
/* size
 * returns the number of nodes in the tree
 */
int size(TreeNode *t) {
    if (t == NULL) {
        return 0;
    }
    return 1 + size(t->left) + size(t->right);
}
```

Suppose you create an empty tree and items are inserted as follows:

```
    TreeNode * tree = NULL;
    insert(5, &tree);
    insert(8, &tree);
    insert(2, &tree);
    insert(1, &tree);
    insert(10, &tree);
    insert(7, &tree);
    insert(9, &tree);
    insert(12, &tree);
```

16. What does the BST look like?

17. What nodes are examined when finding 7?

18. What nodes are examined when finding 3?

19. Now, suppose this is a new tree and insertions are done in this order:

```
    TreeNode * tree = NULL;
    insert(1, &tree);
    insert(3, &tree);
    insert(4, &tree);
    insert(6, &tree);
    insert(7, &tree);
    insert(8, &tree);
    insert(9, &tree);

```

What does this tree look like?

There are ways to balance trees, so we get the win of searches happening closer to $O(log_2 N)$ rather than $O(N)$. You can read about specific kinds of trees, such as red-black trees and AVL trees that support tree rotations.

20. Give an insertion order of the same nodes in problem 4 that results in a full (complete) BST where most interior nodes have two children. Show the tree that results from this insertion order.