Name: Leonard Harold, Toan Ngo, Eric Rouse, Jeff Toy

# Worksheet 29: Binary Search Trees

**On Your Own**

1. What is the primary characteristic of a binary search tree?

All operations are about O(logn), assuming a somewhat balanced tree.

2. Explain how the search for an element in a binary search tree is an example of the idea of divide and conquer.

You are always starting at a "midpoint" and determining if you are going to search for the "target value" in the "half" that is "higher" or "lower" than the current value.

3. Try inserting the values 1 to 10 in order into a BST. What is the height of the resulting tree?

The height would be 9 because the values would go to the right.

4. Why is it important that a binary search tree remain reasonably balanced? What can happen if the tree becomes unbalanced?

The more unbalanced the tree the maximum and average amount of time per operation increases from O(logn) and closer to O(n).

5. What is the maximum height of a BST that contains 100 elements? What is the minimum height?

The maximum height is 99 and the minimum is 7.

6. Explain why removing a value from a BST is more complicated than insertion.

When adding a value to a BST it can always be appended as a new leaf. However, if the if the value being removed is not a leaf then the appropriate leaf must be removed and it's value used in place of the removed value.

Worksheet 29: Binary Search Trees
Name: Leonard Harold, Toan Ngo, Eric Rouse, Jeff Toy

7. Suppose you want to test our BST algorithms. What would be some good boundary value test cases?

The boundaries are pretty few for a BST. Adding the first and removing the last elements are the only true boundaries. However, it would be good to check that a duplicate value won't be added to the tree and a value from the middle of a tree can be removed. Also it would be good to create trees that are completely unbalanced to each side (ie. like #3) and then make sure that the remove algorithm can remove a value in those cases.

8. Program a test driver for the BST algorithm and execute the operations using the test cases identified in the previous question.

```
struct BinarySearchTree *B = createBST();

initBST(B);
addBST(B, 8);
addBST(B, 4);
removeBST(B, 8);
removeBST(B, 4);
if(sizeBST(B) != 0)
        printf("Current size is wrong... %d\n", sizeBST(B));
initBST(B);
addBST(B, 8);
addBST(B, 4);
addBST(B, 12);
addBST(B, 1);
addBST(B, 6);
addBST(B, 9);
addBST(B, 2);
addBST(B, 7);
addBST(B, 15);
removeBST(B, 4);
if(sizeBST(B) != 8)
        printf("Current size is wrong... %d\n", sizeBST(B));
if(containsBST(B, 4) != 0)
        printf("BST should not contain a 4.\n");
if(containsBST(B, 2) == 0)
        printf("BST should contain a 2.\n");
if(containsBST(B, 7) == 0)
        printf("BST should contain a 7.\n");
```

The unbalanced tests would be similar though the addBSTs could be put into a loop.

Worksheet 29: Binary Search Trees
Name: Leonard Harold, Toan Ngo, Eric Rouse, Jeff Toy

9. The smallest element in a binary search tree is always found as the leftmost child of the root. Write a method getFirst to return this value, and a method removeFirst to modify the tree so as to remove this value.

```
TYPE getFirst(node *n) {
  while(n->left != 0)
     return(getFirst(n->left));
  return n->value;
}

void removeFirst(node *n){
  while(n->left != 0)
    removeFirst(n->left));
  free(n);
}
```

10. With the methods described in the previous question, it is easy to create a data structure that stores values in a BST and implements the Priority Queue interface. Show this implementation, and describe the algorithmic execution time for each of the Priority Queue operations.

```
void PQaddElement (struct BinarySearchTree *tree, TYPE newElement){
  addBST(tree, newElement);
}

TYPE PQsmallestElement (struct BinarySearchTree *tree) {
  return getFirst(tree->root);
}

void PQremoveSmallest (struct BinarySearchTree *tree){
  removeFirst(tree->root);
}
```

11. Suppose you wanted to add the equals method to our BST class, where two trees are considered to be equal if they have the same elements. What is the complexity of your operation?

$O(n^2)$