

Exam 1

Due: TBA

This is a practice exam for CS16. On the real exam, you'll be doing five problems that are somewhat similar to these. Topics for the real exam include everything we've discussed in class up to (but not including) the convex hull algorithm(s) (i.e., pretty much everything on which you've had homework problems).

Problem 1

You're given an array of distinct integers $a[0] \dots A[n-1]$ (some of which may be negative), sorted in increasing order. You want to find out whether there's an index i with $A[i] = i$. Give an algorithm for doing so that runs in time $O(\log n)$.

Problem 2

A nonempty binary tree has leaves decorated with tones (white/black). Give pseudocode that decorates the interior nodes with a tone, using this rule:

- If a node has only one child, the node is colored white.
- If the node's two children have the same tone, the node is colored white.
- If the node's two children have different tones, the node is colored black.

Problem 3

You're given as input a list L of 0s and 1s. You are to compute, as output, a list D of numbers indicating, for each position of L , how many ones appear *after* that node (hence for the rightmost "one" in L , the corresponding number in D is zero).

(a) Fill in the table:

L	0	1	0	0	1	0	0	0	1	0
D			2	2				1	0	

(b) Write an algorithm to produce the list D . Full credit for $O(n)$, where n is the number of elements in L .

Problem 4

We've talked about a data structure for binary trees based on references (i.e., a `Node` contains, as instance variables, references to two other `Nodes`, its `leftChild` and its `rightChild`. Assuming each node stores an integer value, we can make a binary *search* tree by insisting that the values of nodes that are left-descendants of node n must be less than that of n , and those for right-descendants are greater-than-or-equal-to the value at node n . Here's an implementation of this idea with a limited selection of methods:

```
// Binary search tree class, where each node stores an integer
public class BST {
    private class Node {
        private Node leftChild = null;
        private Node rightChild = null;
        private int val;

        protected Node(int d) {
            val = d; // left and right kids are null.
        }
        protected insert(int k){
```

```
        if (k < val) {
            if (leftChild == null) {
                leftChild = new Node(k);
            }
            else {
                leftChild.insert(k);
            }
        }
        else if (rightChild == null) {
            rightChild = new Node(k);
        }
        else{
            rightChild.insert(k);
        }
    }

    protected boolean contains(int k){
        if (k == val) return true;
        if (k < val){
            if (leftChild == null)) return false;
            return leftChild.contains(k);
        }
        else { // k > val
            if (rightChild == null)) return false;
            return rightttChild.contains(k);
        }
    }
}

private Node root;

public BST(int n) // create a Bin. Search Tree with the given number at the root
{
    root = new Node(n);
}

public insert(int k) // add a new node containing the number k
{
    root.insert(k);
}

public boolean contains(int k){
    return root.contains(k);
}
}
```

Modify this code so that the BST has a new method, `int balance()` that returns the number of children in the right subtree minus the number of children in the left subtree.

The new method should run in $O(1)$ time.

Problem 5

Below is some code that does something to a list of n elements.

(a) Hand-simulate the algorithm to determine the output when the input is the list 1, 4, 2, 6, 3, 3, 3, 1.

(b) What's the big-O running time in terms of n , the length of the input list? Explain your answer briefly, perhaps even writing a recurrence for the running time.

(c) Write a program that achieves the same result in much better big-O time.

```
// input: a list numbers
// output: ???

Q = new empty queue
Lp = new empty list

while (!L.isEmpty())
    Q.enqueue(L.extractFirst());

while (!Q.empty())
    Lp.addLast(Q.dequeue());
    reverse(Q);

return Lp;

// reverse the elements in the queue
function reverse(Queue Q){
    S = new Stack();
    while !Q.empty()
        S.push(Q.dequeue());
    while !S.empty();
        Q.enqueue(S.pop());
}
```