CSCI 2270 Exam 1
Spring 2013

Pointers, Recursion, Computational Complexity, Linked Lists, Binary Search Trees, Sorting Algorithms

Write Clearly!

Name: ______________________________

9-Digit Student ID: ______________________________

TA (circle one): Xin Shang Yogesh Don't Know

The following two C++ struct definitions will be used in the exam:

```
struct node {          struct bt_node {
  node* next;            int value;
  int value;             bt_node* left;
};                       bt_node* right;
                       };
```

1. Design Question (6 points)

Design a new function called `to_list` for binary search trees that creates and returns a linked list that is arranged in the tree's inorder traversal. You may define a helper function if you like. Describe what functions accept as input, what they return, and what modifications (if any) they will make to the binary tree. Consider the stopping condition(s). You may write pseudo-code. Your design will be graded on how coherent it is. It should be clear enough that a randomly chosen programmer would be able to implement it.

function named to_list.
takes a bt_node* that points to root of binary search tree.
-- creates a blank linked list.
-- does an in order tree traversal, visit appends to linked list
returns reference to node* representing start of the list

+1 for input; +2 for return info, +2 for stopping cond; +1 for it working correctly

2. Debugging Question (4 points)

The following is a recursive size function for binary trees. Identify a bug that causes it to return the wrong number. Then fix the bug. (Hint: draw a three-node tree and see what happens when it is used with this buggy size function.)

```
int size(bt_node* top) {
  if (top == NULL) {
    return 1; <-- that's the bug. should be return 0;
  } else {
    return 1 + size(top->left) + size(top->right);
  }
}
```

_____ / 10

3. Pointer Question (4 points)

```
 1 void pointer_silliness() {
 2   node* ape = new node;
 3   ape->value = 42;
 4   node** cat = &ape;
 5   cout << "The 'value' member of 'ape' is " << ape->value << endl;
 6   cout << "The address of 'ape' is " << &ape << endl;
 7   cout << "The value of 'cat' is " << cat << endl;
 8   node*** tricky = &cat;
 9   cout << "The tricky thing is " << *tricky << endl;
10 }
```

(a) (2 points) This silly function creates some variables and reports information about them. Two things: What is the meaning of the output of lines 6, 7, and 9? Are they different or the same?

**They report an address in memory. They are all the same.
This is the address given to ape when we created it on line 2.**

(b) (2 points) What is the data type of `*tricky` ?

node**

4. Recursion Question (6 points)

Here is a recursive function that counts the number of times you can divide a positive integer until it is **less than** two. In C++ (like many languages), when you divide `a / b` and it does not divide evenly, the result is rounded down. So using this integer math, `5/2` is 2; `4/2` is 2, and `3/2` is 1.

```
1 int num_halves(int val) {
2   if (val == 2 || val == 1) {
3     return 0;
4   } else {
5     return 1 + num_halves(val / 2);
6   }
7 }
```

(a) (1 point) Stare at the stopping condition on line 2 and convince yourself that it is not sufficient to prevent an infinite loop (there is also an off-by-one bug on that line as well). Specify an input value that causes an infinite loop.

0 or anything negative.

(b) (1 point) Correct line 2 so it will not allow an infinite loop but still allows the function to return correct values.

"if val < 2". avoids bad input and stops when input is less than 2 (that's the stopping condition).

(c) (2 points) What does your corrected version return if we call `num_halves(18)` ?

4. If your bug fix from step b changes what is returned, we counted this as correct if your answer is consistent.

(d) (2 points) How about num_halves(-18) ?

0. Same thing as (c).

_____ / 10

5. Linked List Question (6 points)

Re-write the linked list `contains` function to work using recursion. It might be useful to recall the helpful advice from the design question. *Please write proper C++ code.* The function signature is provided here to get you started. Note: this should be really short, like under 8 lines. If it is long, you're doing it wrong.

```
bool contains(node* top, int value) {
   // your code here...
   if (top == NULL) return false;
   if (top -> value == value) return true;
   return contains(top->next, value);
}
```

+2 for correct syntax
+2 for being recursive
+2 for being correct

6. Binary Search Tree Question (6 points)

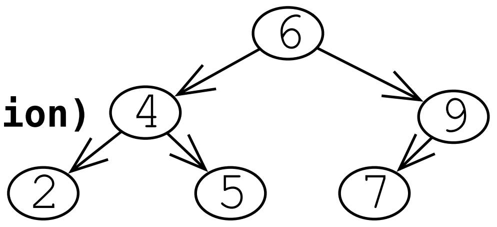
Write pseudocode (or C++ if you like) for a function called `is_bst` that returns true or false depending on if an input binary tree satisfies the invariant of a binary *search* tree. For example, one of the following trees is a binary search tree, the other is not. An empty tree is considered a binary search tree.

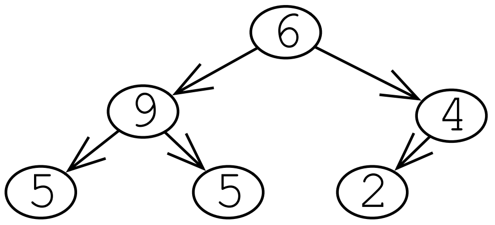
```
is_bst(node):
  if node is null, return true (by definition)

  otherwise,

  if left value is >= than node value,
  or if right value is < node value
     then return false

  otherwise,
    return is_bst(left) && is_bst(right)
```





+2 for checking null/empty input
+2 for checking invariant correctly
+2 for returning result of recursing left and right (and combining the results correctly)

_____ / 12

7. Sorting Algorithm Question (4 points)

Consider this input array: [5, 9, 14, 2, 7].

(a) (2 points) Write the contents of the array after the bubble sort algorithm has run *one time* through the data. Pseudocode for this routine is below.

5, 9, 2, 7, 14

(b) (2 points) Describe in a sentence or two how we could use our existing binary search tree implementation to create a sorted list of the data above. Or, write C++ or pseudocode.

add each number to a BST, then either use our to_array function, or simply do an inorder traversal, since the BST will naturally sort things.

```
bubble(data) {
    num_rounds = 0
    swapped = true
    while (swapped) {
        swapped = false
        num_rounds++
        for i=0, i < data.size-1, i++ {
            if data[i] > data[i+1] {
                swap cells i and i+1 in data
                swapped = true
            }
        }
    }
}
```

8. Complexity Question (4 points)

Say we implement two new functions for binary search trees that report on the tree's depth. The first function, `min_height`, gives us the smallest possible path length from the root to a leaf node; the other, `max_height`, gives the longest possible such path.

What is the *worst case computational complexity* for:

(a) (1 point) `min_height`: **O(n) -- visit all nodes to know for sure**

(b) (1 point) `max_height`: **O(n) -- same reason**

(c) (1 point) Your recursive `contains` function from question 5: **O(n) -- same reason :)**

(d) (1 point) Your `is_bst` function from question 6: **O(n) you guessed it, same reason.**

_____ / 8