**Midterm 2 review CSCI 2270**

1. **How does a binary search tree’s shape depend on the order of the numbers inserted into it?**A Binary Search Tree’s shape depends on the order in which the numbers are inserted into it. If the #’s are inserted in sorted order, the resulting binary search tree is really just a linked list.
2. **What parts are similar in the two processes of searching a binary tree and searching a sorted array by binary search? What parts are different? What depends on luck?**Binary Search trees are similar to the binary search of a sorted array in that they decrease the problem size by half at each step. A binary Search tree has abes, a complexity of nlog(n). This performance is largely based on luck, as the order in which the numbers are input into the BST can greatly impact it’s efficiency. Binary search of a sorted array is at worst nlog(n) efficiency.
3. **Given an arbitrary binary tree, print it out in preorder, inorder, and postorder.**Preorder -> self, left, right  
   Inorder -> left, self, right  
   Postorder -> left, right, self
4. **Given a bunch of numbers, in some order, insert them into a binary search tree.**

**5a. Given the binary search code and a particular array of sorted numbers, tell me the first array slot the search code will check to find 3 in the array 1 3 5 6 8 9 11 14. What’s the last array slot a search for the 3 will check?**The first slot to be checked would be 6. The second slot would be the 3.

**5b. Repeat the question but look for a number that’s not in the array, like 10. What will be the last slot checked?**

1. **To get the 6 big\_number comparison functions ==, !=, <, >, <=, and >=, how many must you write, and why, and what can you do for the other ones instead of writing them all from scratch?**
2. **What time penalty comes from using the add\_node function when copying a list?**Using the add\_node function when copying a list takes the copy function from O(n) complexity to O(n2) complexity.
3. **When is a binary search tree most efficient? Least efficient? Why?**A Binary Search Tree is most efficient when the tree is perfectly balanced. When a tree is perfectly balanced, its depth, and therefore search time is minimized to log(n) time. The worst case for a BST is that it degenerates to a linked list.
4. **Given the code in bintree.cpp, can you make a function that multiplies every number in a binary tree by 7?**  
   multiply( node, factor){  
    if(node){  
    node->data \*= factor;  
    multiply(node->left, factor);  
    multiply(node->right, factor);  
    }  
   }
5. **Given the code in bintree.cpp, can you make a function that reverses (mirror images a binary search tree)?**  
     
   mirror(BST\_Node\* from, BinTree\_Node\* to){  
    if(from){  
    if(!to){  
    to = new BinTree\_Node();  
    }  
    to->data = from->data;  
    mirror(from->left, to->right);  
    mirror(from->right, to -> left);  
    }  
   }
6. **If you had a mirror imaged binary search tree, what would you need to do when inserting data into it?**The opposite of what you would do when inserting data into a normal binary search tree!
7. **Why is self assignment a problem for operator =?**  
   Self-assignment is a problem for the = operator because the first step of the = operator clears and de-allocates the memory for (\*this) before assigning it to the object on the right of the operator. Self-assignment would effectively destroy the object.
8. **Why is self assignment not a problem for the copy constructor?**  
   Self-assignment is not a problem for the copy constructor because the c++ compiler will not allow an object to be instantiated from an object that does not exist yet, i.e. itself.
9. **What is the difference between an assignment operator and a copy constructor?**  
   The operator= clears a given object and copies the data from the other object into (\*this). The copy constructor is similar, however it copies the data from an object that exists into one that is being created.
10. **Use pointer arithmetic to write a function to reverse an array.**
11. **Why can’t we do binary search on a linked list?**  
    We cannot do a binary search on a linked list because it is not indexed. The only way to search through a linked list is linearly, walking through every node, as each node only knows its ‘next’ neighbor.
12. **Why is contains for a binary search tree faster than O(n)? Can binary tree contains be this fast?**   
      
    Contains for a binary search tree can take O(n) time to complete in the worst case that the binary tree is completely degenerate and is really just a linked list. This case is extremely rare however. Most of the time, a binary search tree is faily well balanced, which means that twith each step in the search process, you are throwing away – at best – half of the remaining dataset. This means that many nodes go untouched, resulting in a computation time less than O(n).
13. **Suppose I am adding 2 big\_numbers as follows:**

**big\_number alice(98); big\_number bobo(87); alice+=bobo;**

**In the code for operator +=, big\_number& big\_number::operator+= (const big\_number& b) which number, alice or bobo corresponds to b? Which number corresponds to \*this?**

1. **Tell me how a stack can be used to tell if a program has balanced {}.**
2. **Trace out the tree\_copy function for a particular binary tree. Which node is copied first? Last?**
3. **Trace out the tree\_clear function for a particular binary tree. Which node is cleared first? Last?**
4. **Be nauseatingly familiar with the copy command.**Copy( index to start at , index to stop at [exclusive], index to start copying to)
5. **Where is the smallest number in a binary search tree? How would you find it?**The furthest bottom-left node!
6. **When I compare 2 big\_numbers, which digits should I compare first and why?**