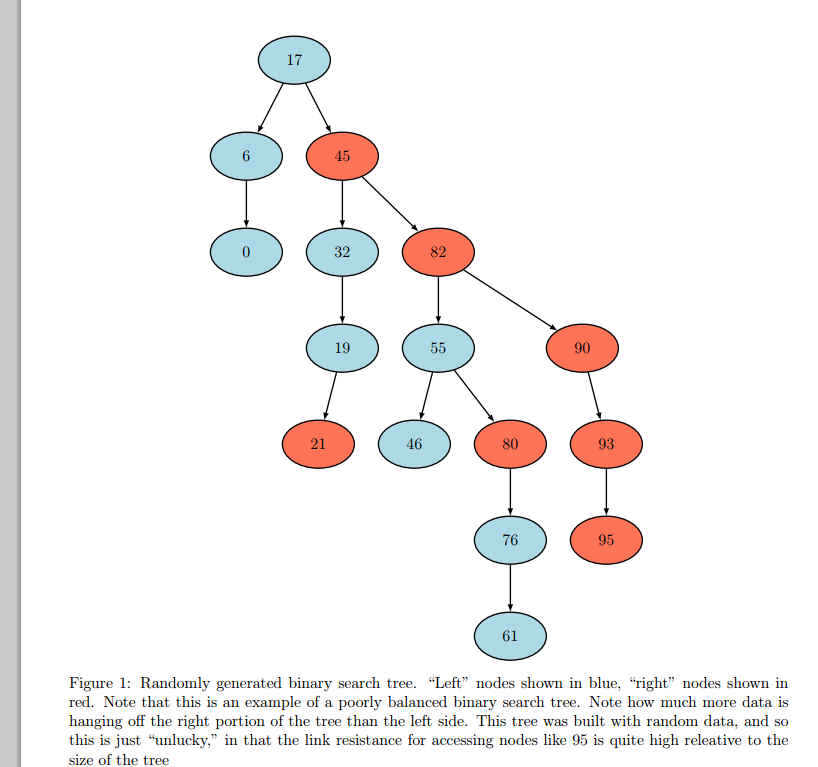
**Midterm 2 review**

**CSCI 2270**

**1. How does a binary search tree’s shape depend on the order of the numbers inserted into it?**

· If you have a root with descendants of smaller numbers than the root number, the tree shape will be lopsided

A. A binary search tree’s shape is entirely dependant on the order of insertions. If you insert into a binary tree in monotonical increasing or decreasing order, you will just get a (worse) linked list (think about it!)

(see fig 1)

**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?**

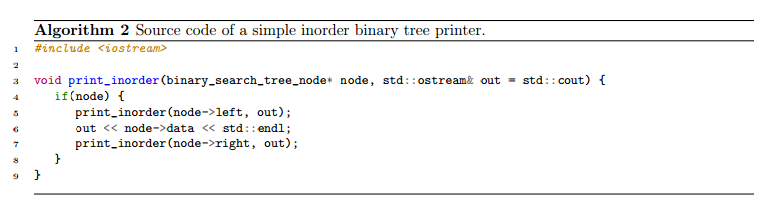
· The time it takes to search in both processes is (Olog(n)) time

· Binary search trees have a hierarchy. If it is a lopsided tree, it could take longer to search. The array is linear, so it does not depend on a hierarchy and only has one way to go in searching

· The order of the numbers depends on luck

A. Binary search of a sorted array is similar to searching a binary tree in that (for a balanced binary search tree) the problem size is shrinking by a factor of two at every step. Note that this is the best case for a binary search tree, as it is possible for the tree to be wildly out of balance. In the worst case, you end up with a linked list, which just gives linear search time. If the tree is being populated with random data, it is largely luck dependant how well balanced the tree is. See figure 1 for an example

**3. Given an arbitrary binary tree, print it out in preorder, inorder, and postorder**

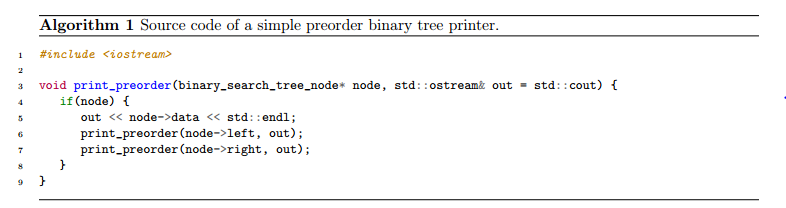


· **preorder\_print** (const binary\_tree\_node\* bintree)

if (bintree is not empty)

{print out data at root

preorder\_print(left subtree of bintree’s root)

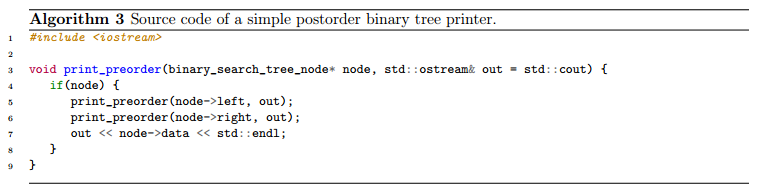
preorder\_print(right subtree of bintree’s root)}

· **inorder\_print** (const binary\_tree\_node\* bintree) :

if (bintree is not empty){

inorder\_print(left subtree of bintree’s root)

print out data at root

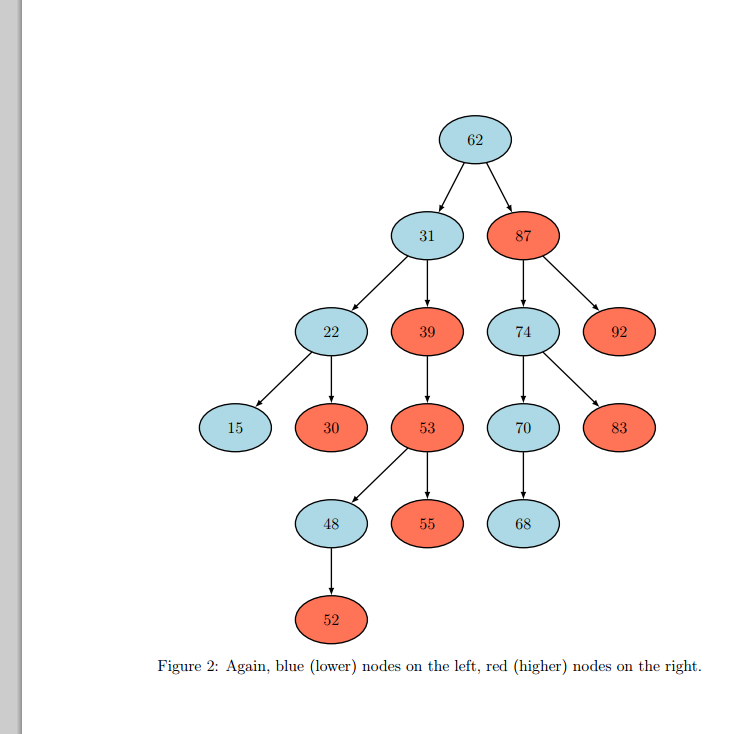
inorder\_print(right subtree of bintree’s root)}

·  **postorder\_prin**t (const binary\_tree\_node\* bintree) :

if (bintree is not empty){

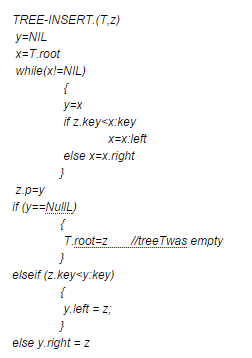
postorder\_print(left subtree of bintree’s root)

postorder\_print(right subtree of bintree’s root)

print out data at root

**4. Given a bunch of numbers, in some order, insert them into a binary search tree.**

A. The rule is simple: if whatever number we are going to insert is less than the number of the node we are at, go left and repeat. Otherwise, go right and repeat. Stop when you find an empty node,because that is where your data should be inserted.

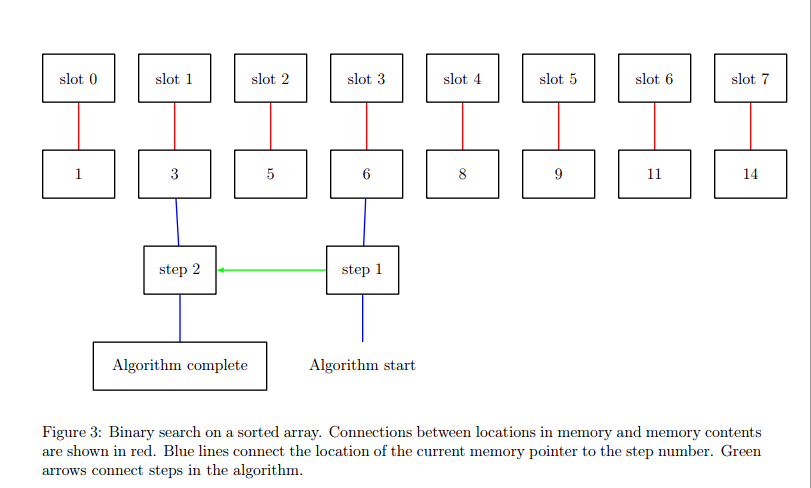


**5a. Given the binary search code and a particular array of sorted numbers, tell me the first array slot the search code will check to find 3 in the array 1 3 5 6 8 9 11 14. What’s the last array slot a search for the 3 will check?**

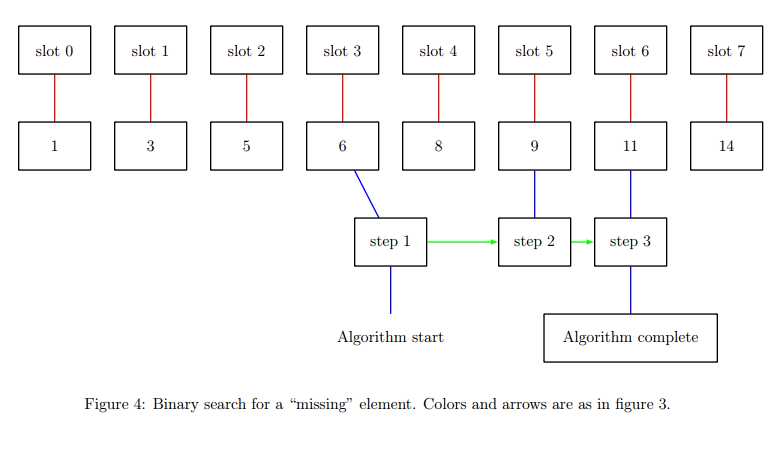
· It will check 6 first(index 3)

o First it will count the numbers of elements in the array then divide by two. We will go to that index and compare

· It will find 3 last(index 1)



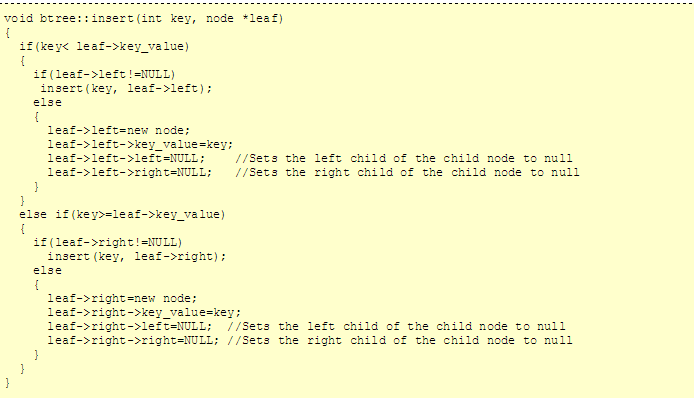
**5b. Repeat the question but look for a number that’s not in the array, like 10. What will be the last slot checked?**



· 11 will be the last number checked look at picture to better understand why →

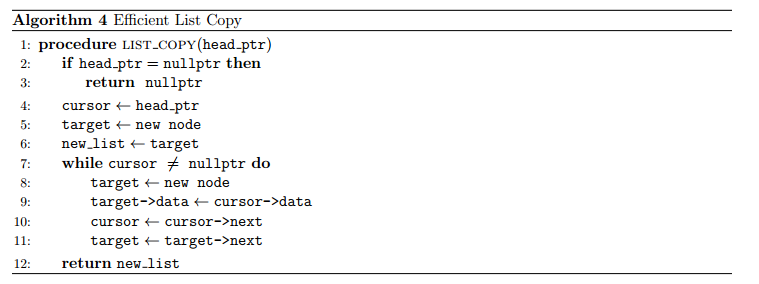
**6. To get the 6 big\_number comparison functions ==, !=, <, >, <=, and >=, how many must you write, and why, and what can you do for the other ones instead of writing them all from scratch?**

· equivalence, less than, greater// you can create everything else from stemming from these functions

Insert function for a binary tree

**7. What time penalty comes from using the add\_node function when copying a list?**

· *walk through the length of the list every single time that add a node and that o-n time*

*A. It moves the operation from O(n) to O(n2). The add node function only has access to the head ptr node, and so it is forced to “walk” to the end of the list again and again as we repeatedly call add node. In contrast, an implementation written specifically for the purpose of copying a list (or even std::copy, provided you are smart enough to write your own iterator), would be much faster. See algorithm 4 for a simple pseudocode* example

**8. When is a binary search tree most efficient? Least efficient? Why?**

. A binary search tree is most efficient when it is perfectly balanced. It is least efficient when

its depth is equal to the number of nodes it contains (i.e., it has degenerated into a linked list). The point of a binary search tree is to provide for fast lookup and insert. It does this by putting its data in a “sorted” order. This order is such that, at every step of the insert and search algorithms, the problem size is cut in half, providing O(log n) execution time. This is best case for a binary search tree. In the worst case, the tree has degenerated into a line (linked list), and you get linear (O(n)) execution time.

*Advantages***:**

· Binary Search Tree is fast in insertion and deletion etc. when balanced.

· Very efficient and its code is easier than other data structures.

· Stores keys in the nodes in a way that searching, insertion and deletion can be done efficiently.

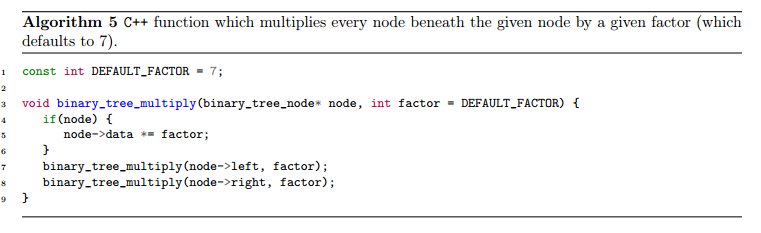
· Implementation is very simple in Binary Search Trees.

· Nodes in tree are dynamic in nature.

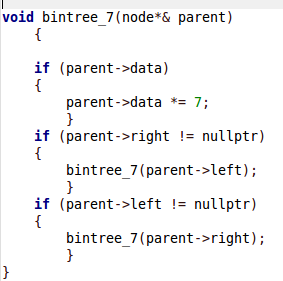
*Disadvantages***:**

· The shape of the binary search tree totally depends on the order of insertions, and it can be degenerated.

· When inserting or searching for an element in binary search tree, the key of each visited node has to be compared with the key of the element to be inserted or found, i.e., it takes a long time to search an element in a binary search tree.

· The keys in the binary search tree may be long and the run time may increase.

· After a long intermixed sequence of random insertion and deletion, the expected height of the tree approaches square root of the number of keys which grows much faster than .

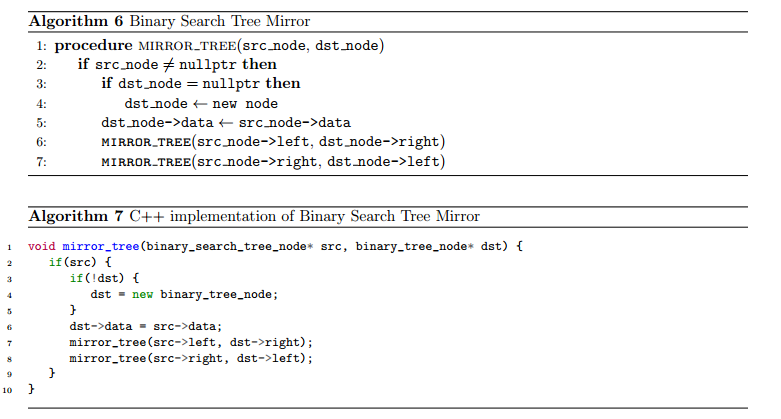


**9. Given the code in bintree.cpp, can you make a function that multiplies every number in a binary tree by 7?**

# ← Recursive mult by 7 funct

Go to the print function and where it prints out the data, simply multiply the data by 7.

**10. Given the code in bintree.cpp, can you make a function that reverses (mirror images a binary search tree)?**

· Go to the print function and replace right with left and left with right.

**11. If you had a mirror imaged binary search tree, what would you need to do when inserting data into it?**

· The smallest item is on the right this time, so when the entry is less than the node it puts it to the right and when it is greater it goes to the left

A. Follow the opposite rules as normal. If the default rule is to go left when <, and right otherwise, then you need to go left when not < (i.e., when >=) and right otherwise. See algorithm 8.

**12. Why is self-assignment a problem for operator =?**

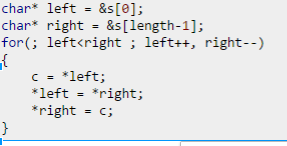
·

the self operater will clear the big number when assign and so if you self assign it will clear the number before it sets it.

**13. Why is self-assignment not a problem for the copy constructor?**

· passed in by const reference, clear the copy, then reset it but it cant change it because its passed in by const reference

**14. What is the difference between an assignment operator and a copy constructor?**

· one is passed by const refere

**15. Use pointer arithmetic to write a function to reverse an array.**

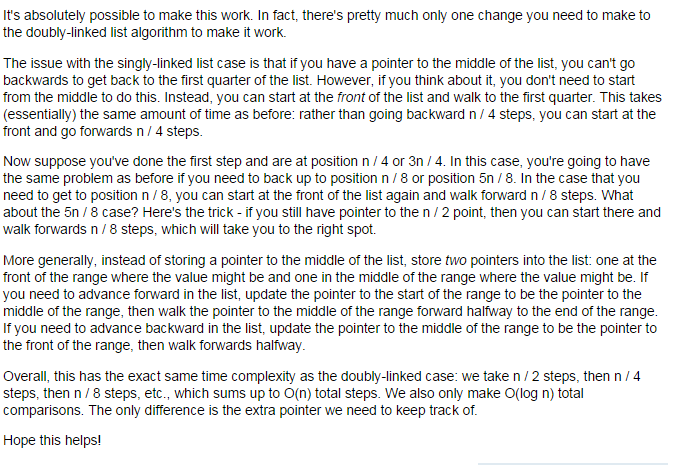
· Go to the first location in the array and switch with the very last one

o Temp = data.i;

o Data.i = data.count – i;

o Data.Count – i = temp;

**16. Why can’t we do binary search on a linked list?**

· you can only go through the list one by one and you cannot go backward. Only points forward so you cant start at the end and move backward to the ¼ point, BUT you can walk to the ¼ point from the front. in fact because of this you theoretically could do binary search on a singly linked list. I have attached an explanation:

**← For Q 16**

**17. Why is contains for a //(linked list??)binary search tree faster than O(n)? Can binary tree contains be this fast?**

· Binary search tree is sorted so it does not have to check every single item it just has to compare and go left or right depend on the output of the compare function. The binary tree is unsorted so you need to check every item

//I think that a (doubly) linked list will always be O(n) because you don’t have to check every node, whereas a binary tree can be this fast if the data was put in fairly randomly BUT if it was entered in like figure1 then you will pretty much have to check every node which is no O(n)

**18. Suppose I am adding 2 big\_numbers as follows:big\_number alice(98);**

big\_number bobo(87);

alice+=bobo;

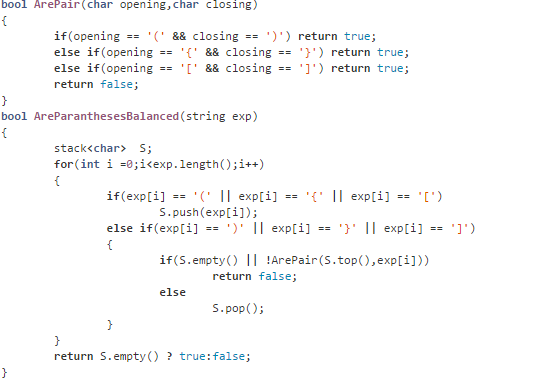
In the code for operator +=,

big\_number& big\_number::operator+= (const big\_number& b)

which number, alice or bobo corresponds to b? Which number corresponds to \*this?

· b – Bobo

· \*this - alice



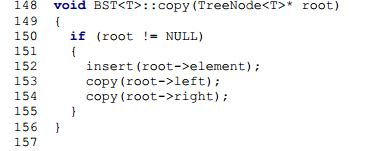
**19. Tell me how a stack can be used to tell if a program has balanced {}.**

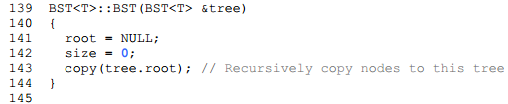
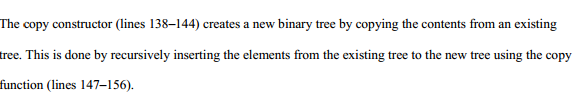
**← for Q 19**

· If stack’s empty at the end and never underflowed,

your {} are balanced

**20. Trace out the tree\_copy function for a particular binary tree. Which node is copied first? Last?**

·

o 

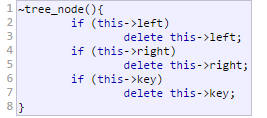
**·**

**The very left item is copied first**

**Then the right side**

**The root is copied last**

**21. Trace out the tree\_clear function for a particular binary tree. Which node is cleared first? Last?**

· Clear left, clear right, clear root

**22. Be nauseatingly familiar with the copy command.**

binary\_tree\_node\* tree\_copy(const binary\_tree\_node\* root\_ptr)

{

binary\_tree\_node\* l\_ptr; binary\_tree\_node\* r\_ptr;

if (root\_ptr == nullptr) return nullptr;

else {

l\_ptr = tree\_copy( root\_ptr->left );

r\_ptr = tree\_copy( root\_ptr->right );

binary\_tree\_node\* new\_tree = new binary\_tree\_node;

new\_tree->data = root\_ptr->data;

new\_tree->left = l\_ptr; new\_tree->right = r\_ptr;

return new\_tree;

}

}

**23. Where is the smallest number in a binary search tree? How would you find it?**

· The very left branch item. If entry is not equal to node, check to see if entry is less than the node. If it is, keep going left. If it is not, go to the right

**24. When I compare 2 big\_numbers, which digits should I compare first and why?**

· start with largest base in the number because that will the easiest way to test inequality.