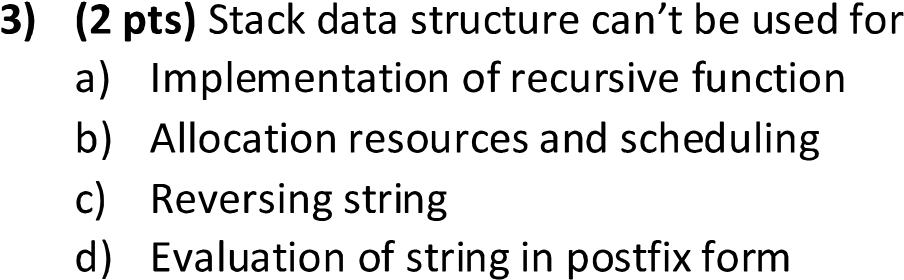
 Darek Konopka

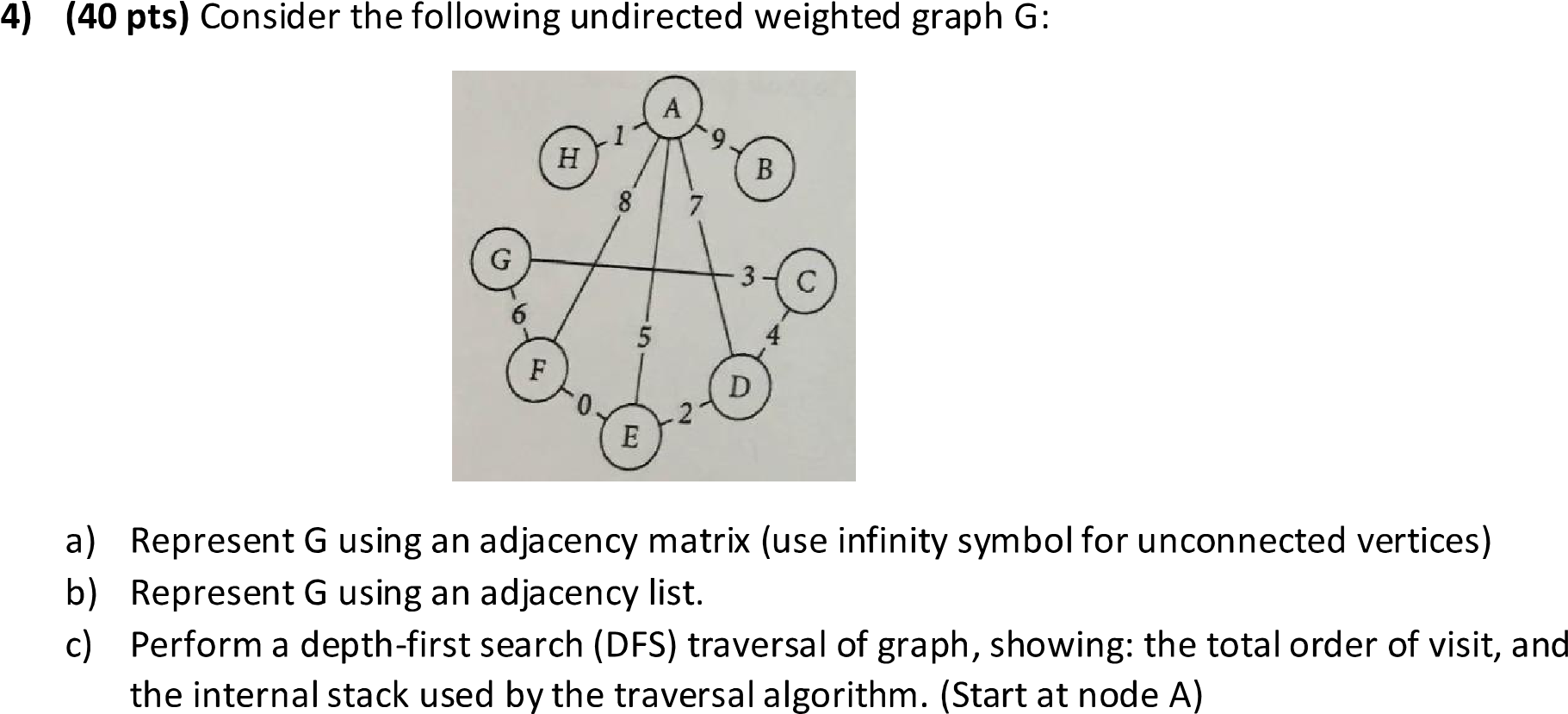
Darek Konopka

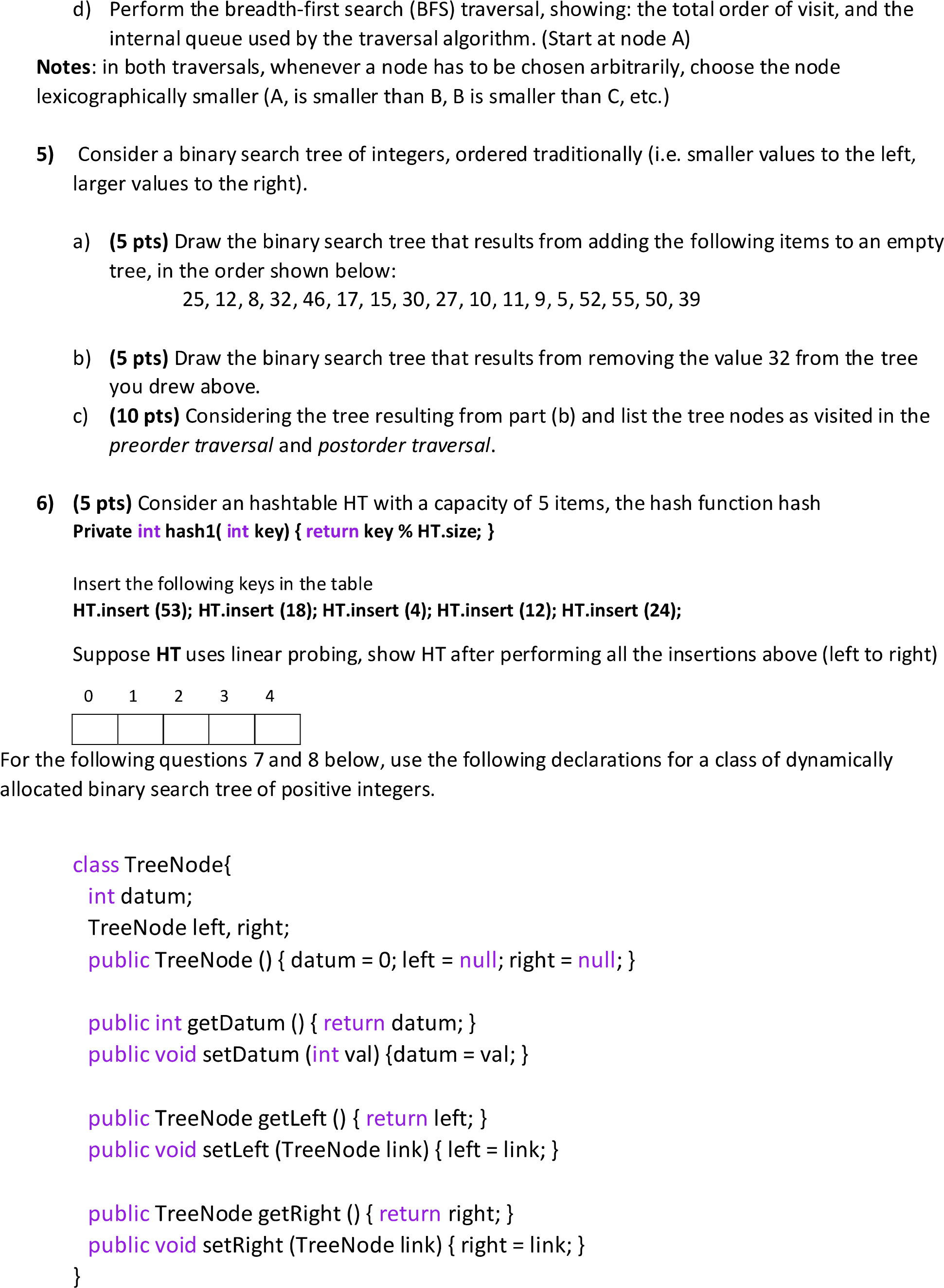


B.) ABC\*+

A.) True

B.) Allocation resources and scheduling





A.)

0, 1, 0, 1, 1, 1, 0, 1,

1, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 1, 0, 0, 1, 0,

1, 0, 1, 0, 1, 0, 0, 0,

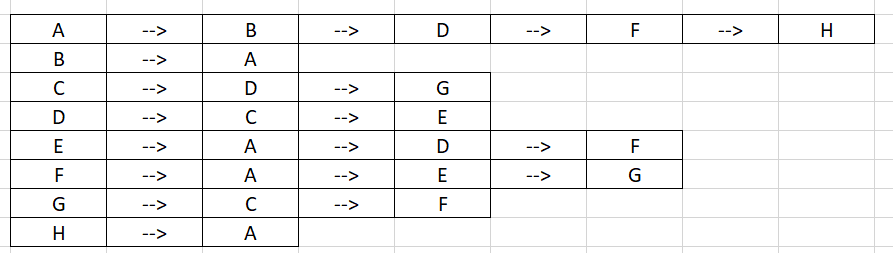
1, 0, 0, 1, 0, 1, 0, 0,

1, 0, 0, 0, 1, 0, 1, 0,

0, 0, 1, 0, 0, 1, 0, 0,

1, 0, 0, 0, 0, 0, 0, 0,

B.)



C.)

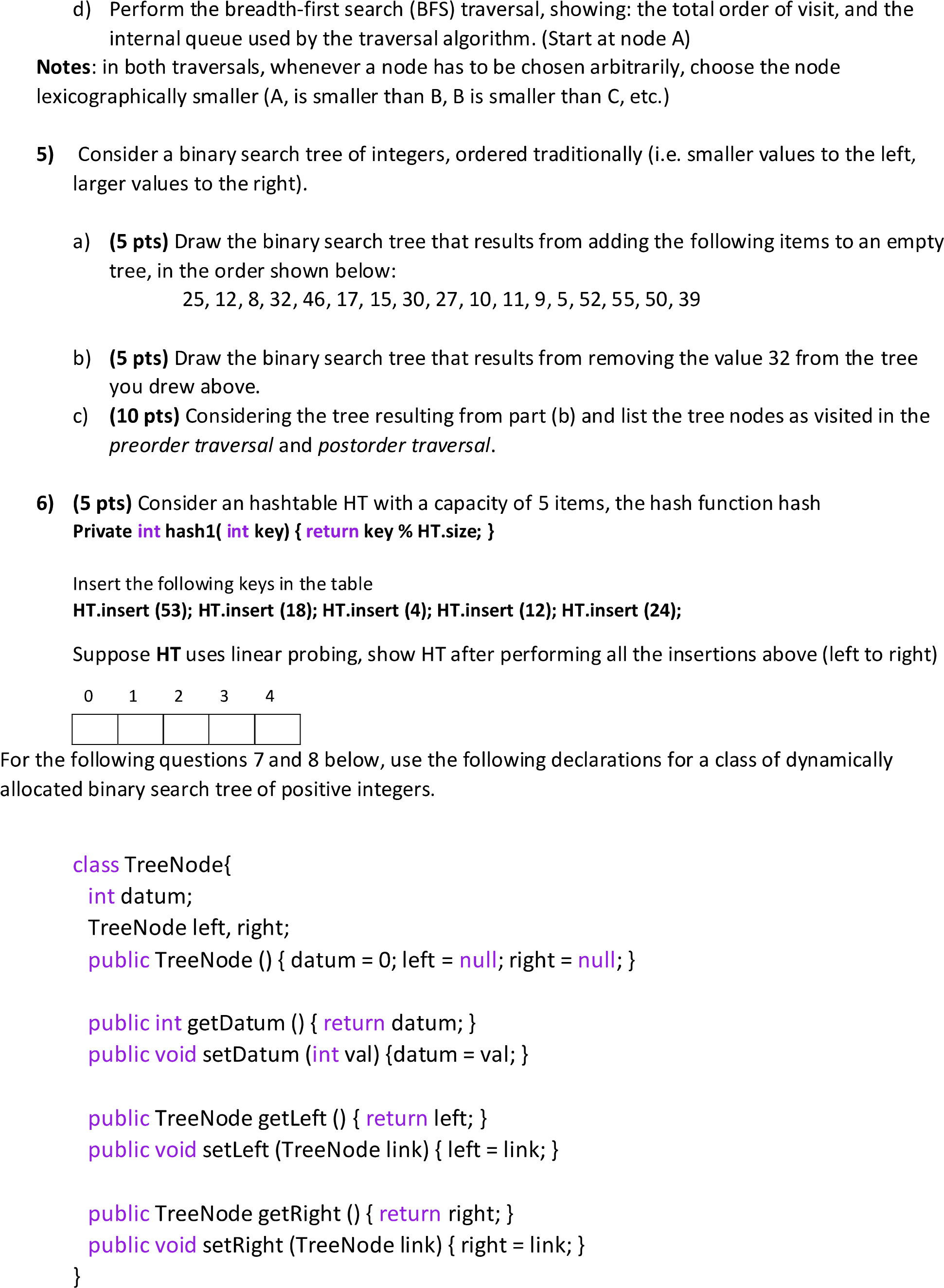
The Depth First Traversal is:

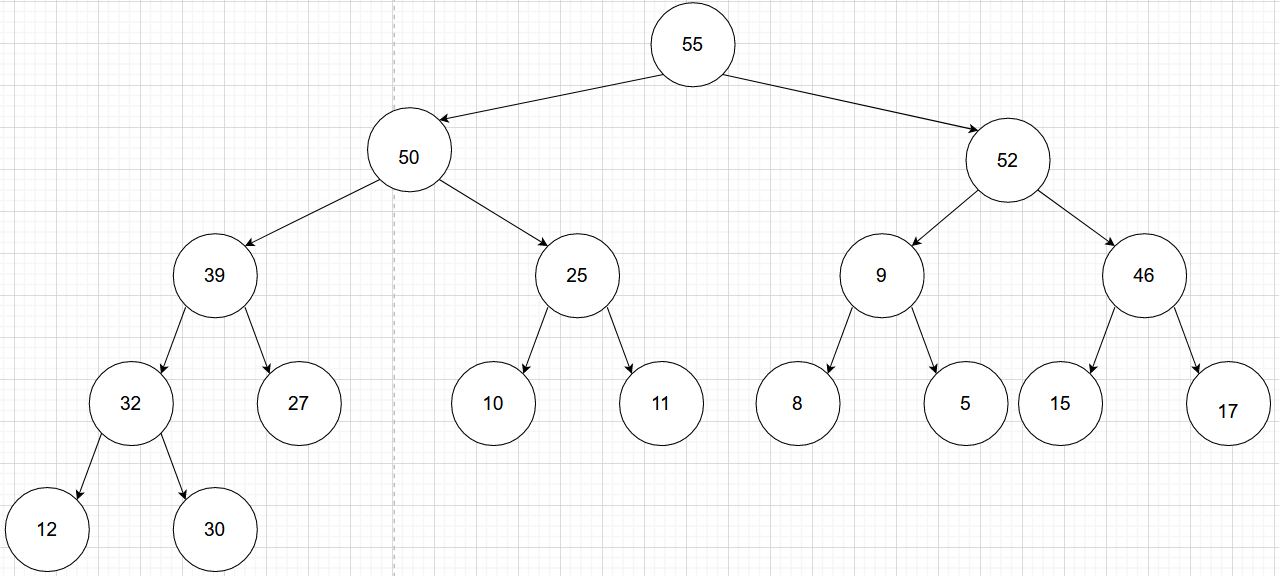
0 1 3 4 5 6 7 2 => A B D E F G H B

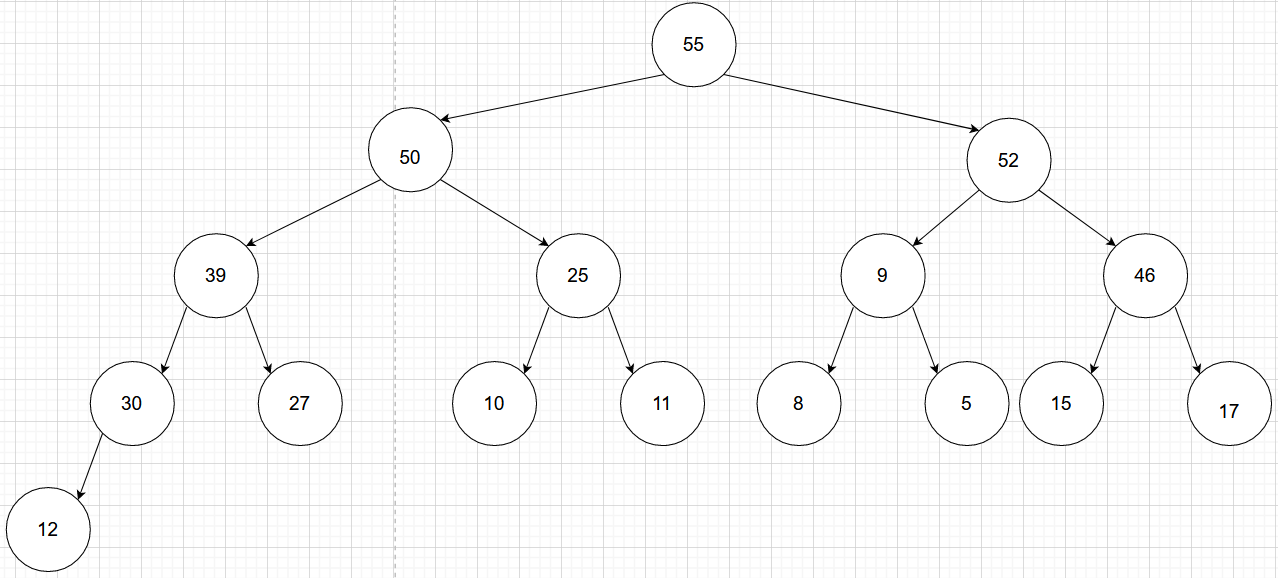
D.)

The Breadth First Traversal starting at Node A is:

0 1 3 5 7 4 6 => A B D F H E G



A.)

B.)

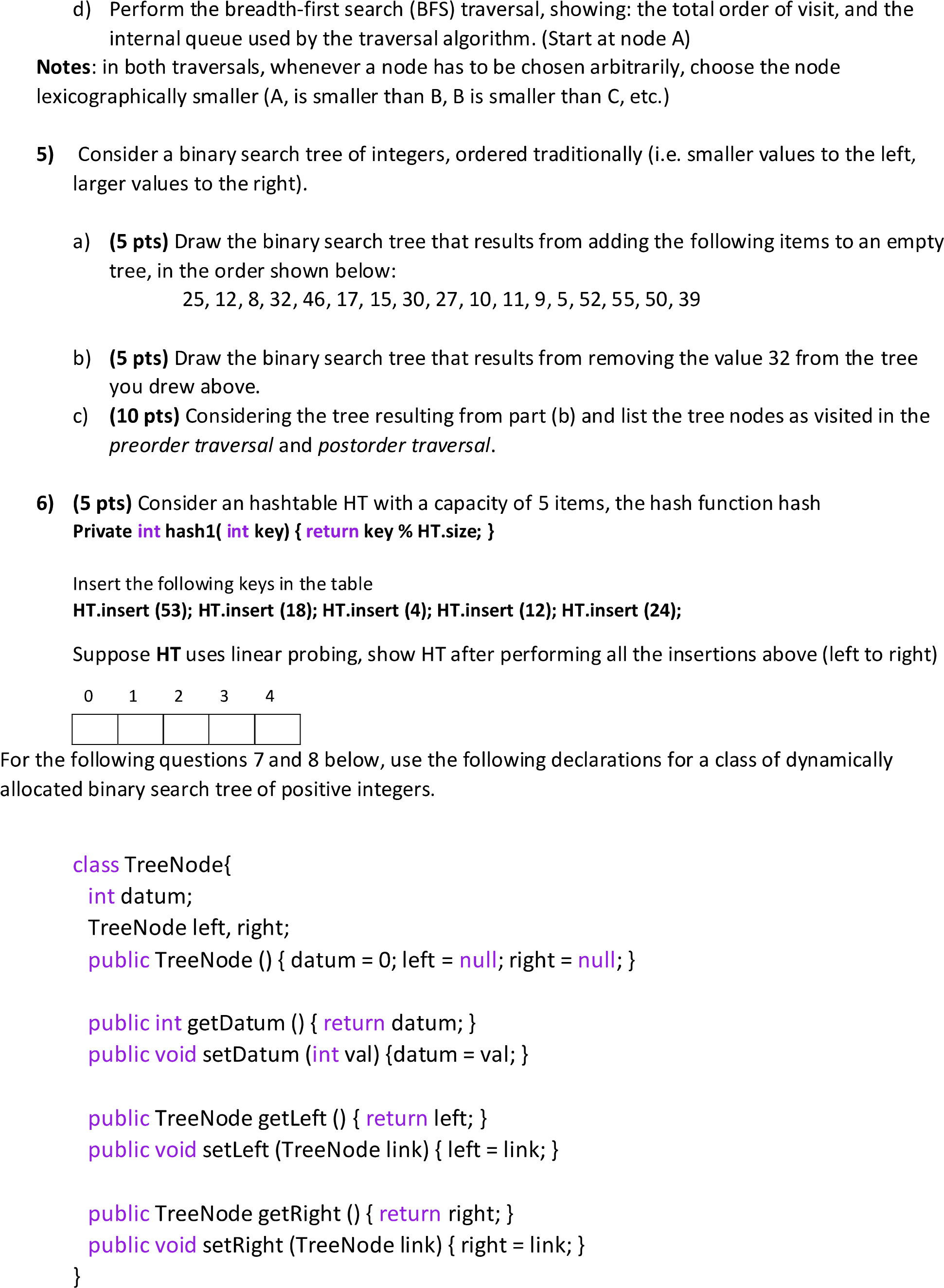
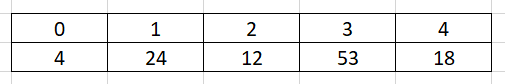
C.)

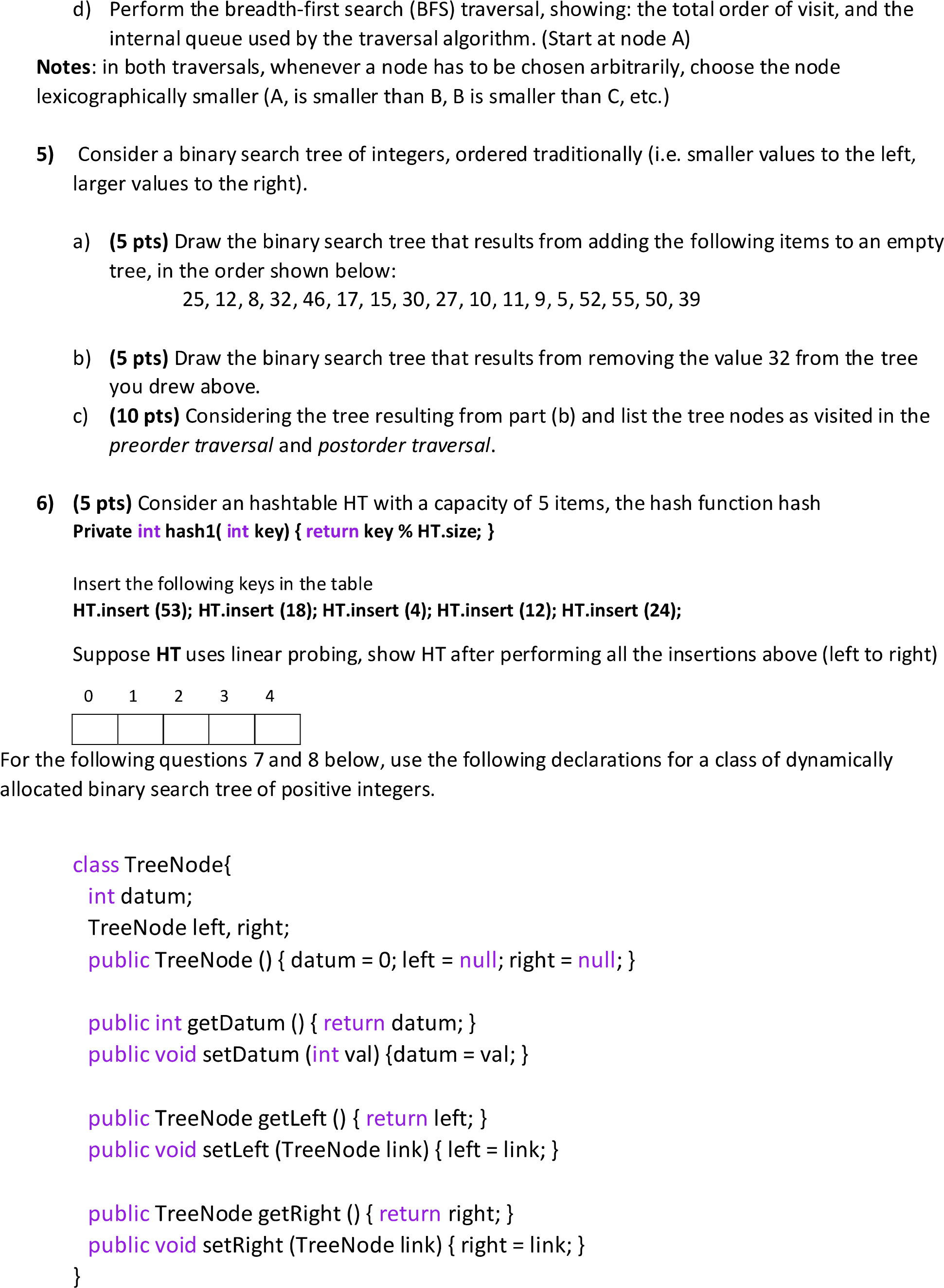
Preorder:

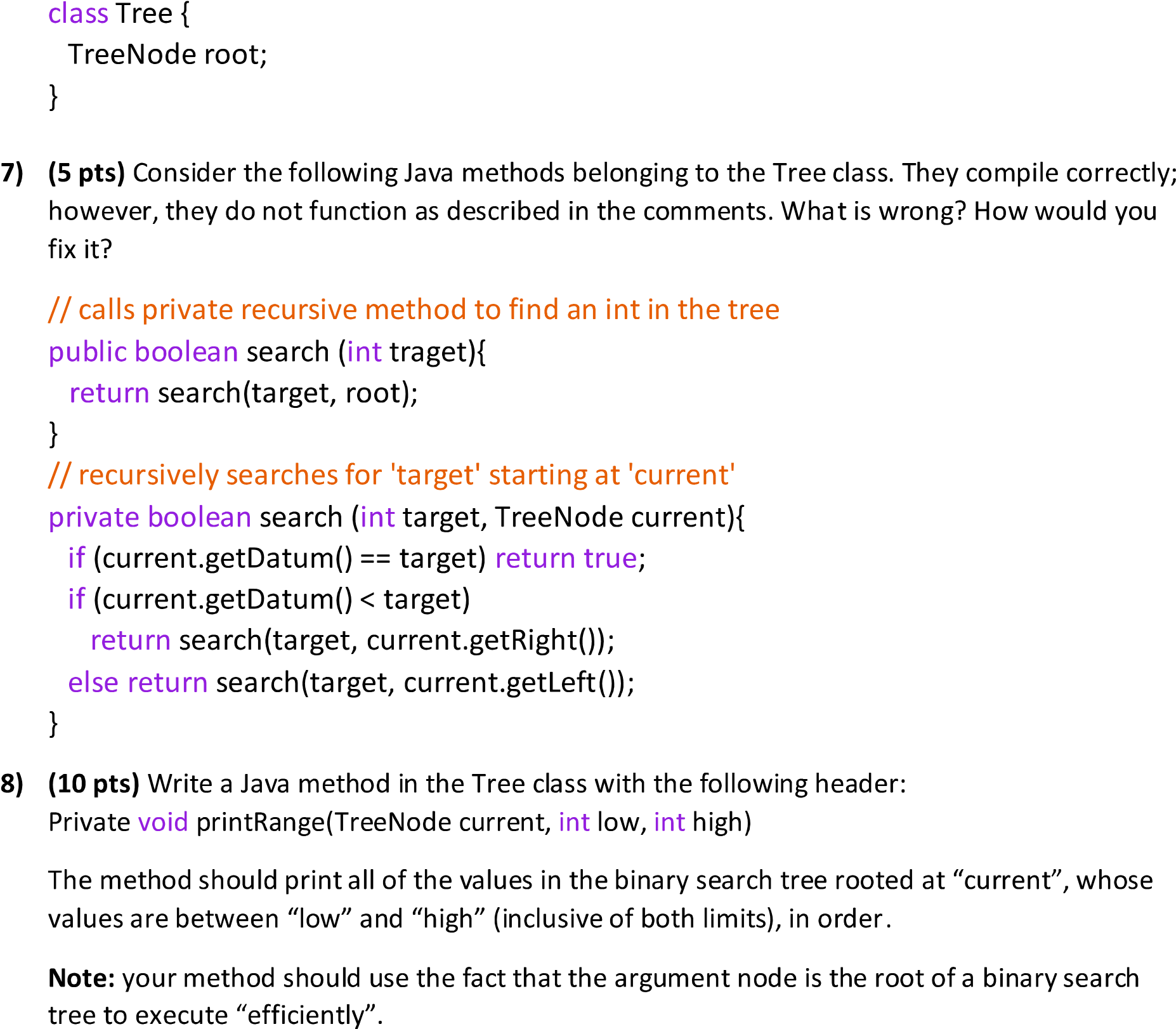
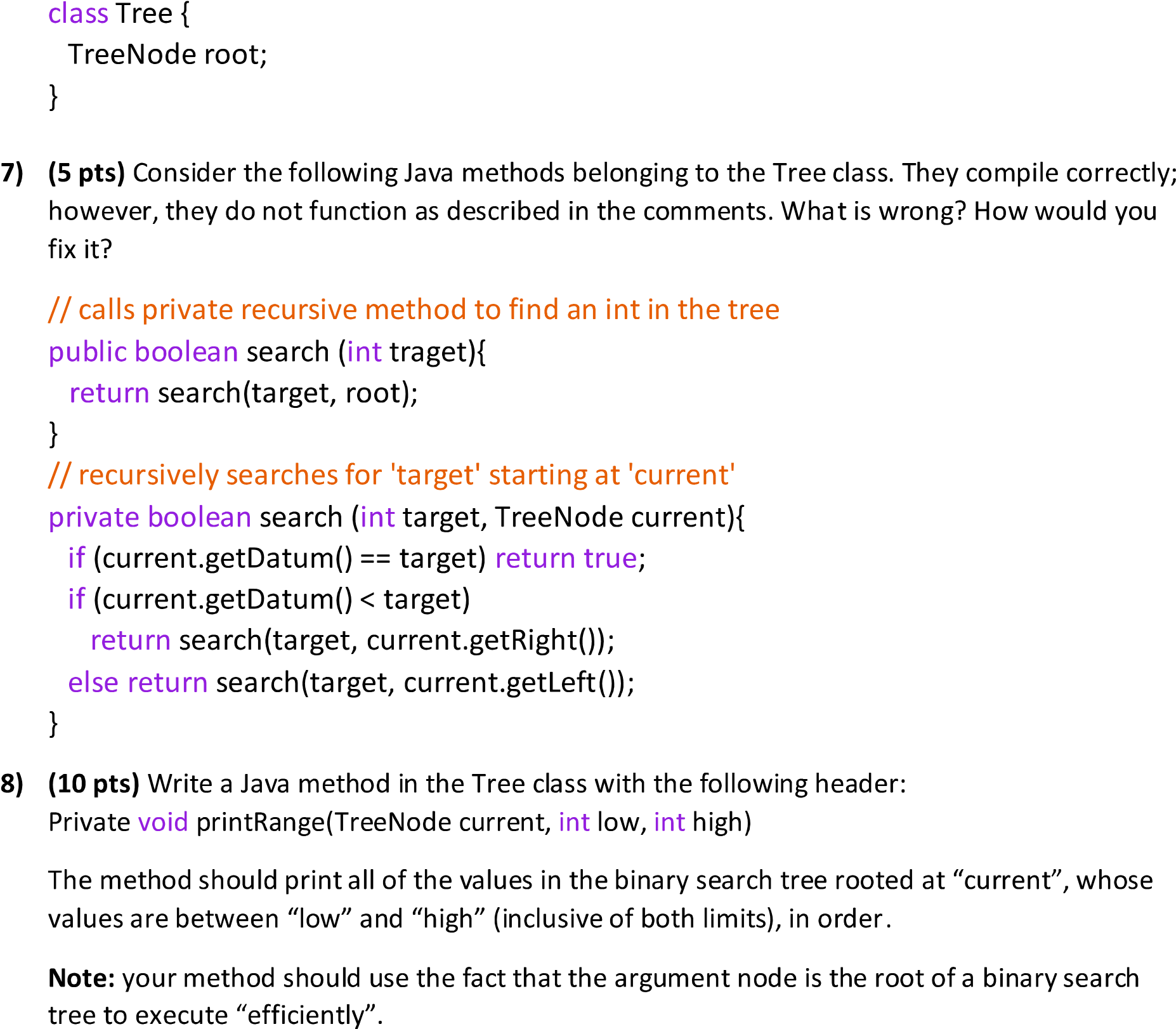
55, 50, 39, 30, 12, 27, 25, 10, 11, 52, 9, 8, 5, 46, 15, 17

Postorder:

12, 30, 27, 39, 10, 11, 25, 50, 8, 5, 9, 15, 17, 46, 52, 55





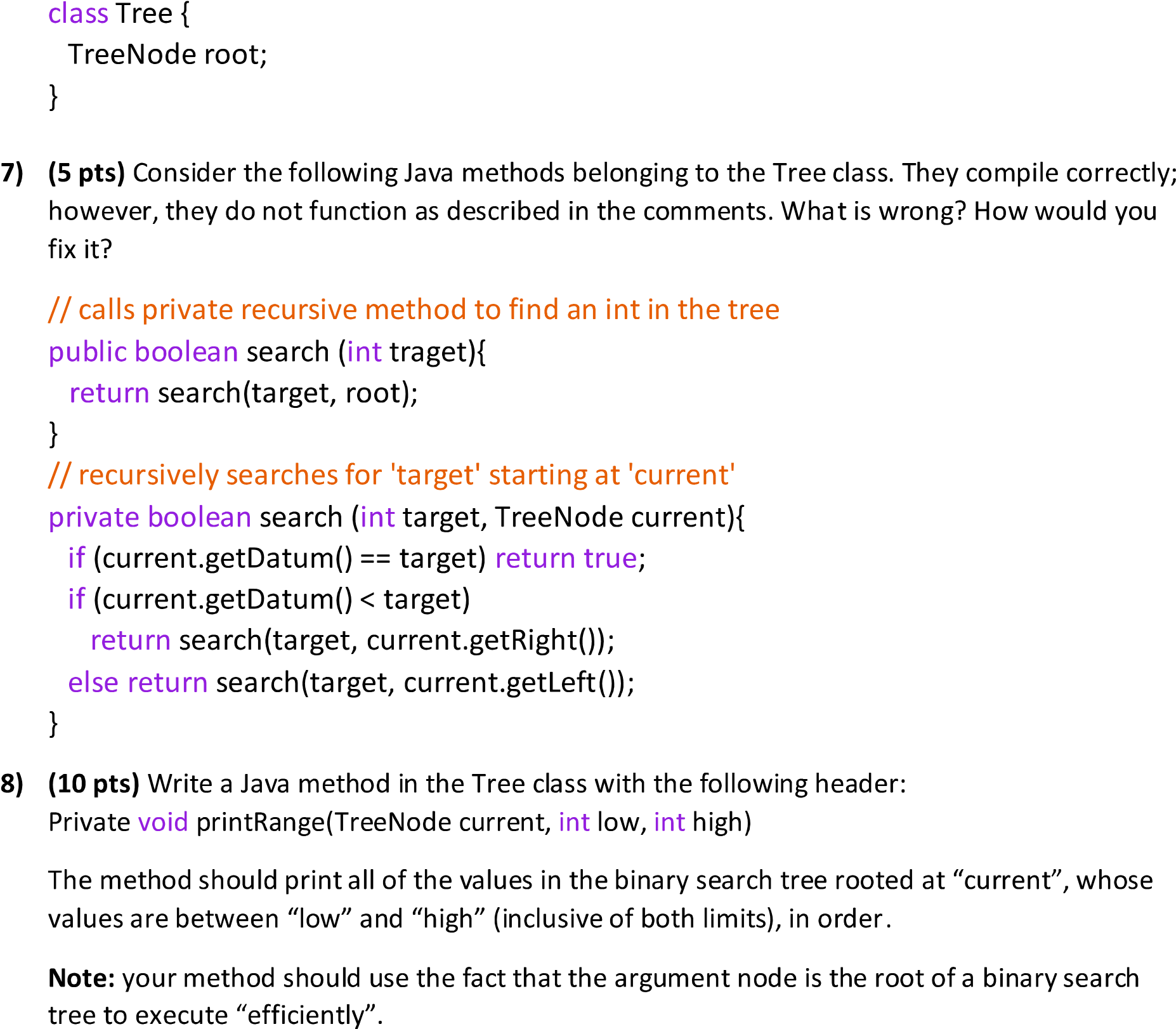


First, I would want to ask, why are we returning a Boolean in the recursive method? Also, the fact the recursive method and the normal method have the same name makes me question if this compiled since you have two methods with the same name. That first fact of this method makes me question it, and makes me want to rewrite it. Here is how I would write a code to recursively search for ‘target’ starting at ‘current’:

// calls private recursive method to find an int in the tree

boolean search(int target) {   
 root = searchRec(target, current);   
 if (root!= null)  
 return true;  
 else  
 return false;  
 }

//recursive search function  
 Private TreeNode searchRec(int target, TreeNode current) {   
 // base case: current is null or target presents at current  
 if (current ==null || current.getDatum()==key)   
 return root;   
 //value is less than current's target  
 if (current.getDatum() > target)   
 return searchRec(target, current.getLeft());   
 // value is greater than current' target  
 return searchRec(target, current.getRight());   
 }



void printRange(TreeNode current, int low, int high) {  
   
 // base case   
 if (current == null) {  
 return;  
 }  
   
 // Since the tree is sorted, recurse from the left subtree first  
 // if current is greater than low   
 if (low < current.getDatum()) {  
 printRange(current.getLeft(), low, high);  
 }  
   
 // if current lies in range, then prints root's data   
 if (low <= current.getDatum() && high >= current.getDatum()) {  
 System.out.print(current.getDatum() + " ");  
 }  
   
 // If current is smaller than high, then we only  
 // can get the elements in right subtree   
 if (high > current.getDatum()) {  
 printRange(current.getRight(), low, high);  
 }  
 }