**Problem Set 7, Part I**

**Problem 1: Working with stacks and queues**

**1-1)**

**public static void doubleAllStack(Stack<Object> stack, Object item) {**

**if(item == null){**

**throw new NullPointerException();**

**}else{**

**Stack<Object> temp = new Stack<Object>();**

**while(stack.isEmpty() != true){**

**temp.push(stack.pop());**

**}**

**while(temp.isEmpty() != true){**

**if(temp.peek().equals(item)){**

**stack.push(item);**

**}**

**stack.push(item);**

**temp.pop();**

**}**

**}**

}

**1-2)**

**public static void doubleAllQueue(Queue<Object> stack, Object item) {**

**if(item == null){**

**throw new NullPointerException();**

**}else{**

**Queue<Object> temp = new Queue();**

**while(stack.isEmpty() != true){**

**temp.insert(stack.remove());**

**}**

**while(temp.isEmpty() != true){**

**if(temp.peek().equals(item)){**

**stack.insert(item);**

**}**

**stack.insert(item);**

**temp.remove();**

**}**

**}**

}

**Problem 2: Using a queue to search a stack**

**boolean isFound = false;**

**while(S.isEmpty() != true){**

**if(S.peek().equals(I)){**

**IsFound = true;**

**}**

**Q.insert(S.pop());**

**}**

**for(int x = 0; x < 2; x++){**

**while(Q.isEmpty() != true){**

**S.push(Q.remove());**

**}**

**while(S.isEmpty() != true){**

**Q.insert(S.pop());**

**}**

**}**

**if(isFound){**

**return true;**

**}else{**

**return false;**

**}**

**Problem 3: Binary tree basics**

**3-1) 3**

**3-2) 4 leaf nodes 4 internal nodes**

**3-3)** 44,35,23,28,53,48,62,57,80

**3-4) 28,23,35,48,57,80,62,53,44**

**3-5) 44,35,53,23,48,62,28,57,80**

**3-6) Yes because the left subtree of each root is less than the root and the right subtree of each root is greater than the root**

**3-7) No because with 23 and 28 the right subtree difference between the left is 1 when it should be the left. There is also a difference in height on the right between the left by 1 on node 53**

**Problem 4: Tree traversal puzzles**

**4-1)**



**4-2)**



**Problem 5: Binary search trees**

**5-1)**



**5-2)**



**Problem 6: Counting keys below a threshold**

**6-1) The best case would be if the tree is just one node thus the two if statements would only iterate once leaving the big 0 notation to be O(1). The worst case is if the tree has a left and right node, making the if statements reiterate the whole method, leaving the big o notation to be O(n).**

**6-2)**

private static int numSmallerInTree(Node root, int t) {

int count = 0;

if(root.key < t){

if(root.left == null){

return count + 1;

}else{

return count + 1 + numSmallerInTree(root.left, t);

}

}else{

if(root.right == null){

return count + 1;

}else{

return count + 1 + numSmallerInTree(root.right, t);

}

}

}

**6-3)The best case would be if the tree is just one node thus the if statement would only iterate once leaving the big 0 notation to be O(1). The worst case if the tree has a right node that is less than t throughout or at the leaf nodes and has a left node, making the if statements reiterate the whole method, leaving the big o notation to be O(n^2).**

**Problem 7: Balanced search trees**







