1. (10 points) Warmup Fun

a. (4 Points) Tree Traversal
This was a poorly worded question. You could either answer the questions by assuming it was a binary tree and assign the left and right child based on where they are graphically (in red), or you could have assigned whether something was a left or right child based on what was alphabetically first (in blue). The only difference it made was the swapping of the D and E.

i. Write out the order in which you would print the nodes if you printed them in **breadth first order** (break ties alphabetically).

A B F G C H D
A B F G C H D E

ii. Write out the order in which you will print the nodes if you printed them **depth first preorder** (break ties alphabetically).

A B G E D F C H
A B G D E F C H

iii. Write out the order in which you will print the nodes if you printed them **depth first inorder** (break ties alphabetically).

B E G D A C F H
B D G E A C F H

iv. Write out the order in which you will print the nodes if you printed them **depth first postorder** (break ties alphabetically).

E D G B C H F A
D E G B C H F A

b. (2 Points) Balanced Trees
Recall the definition for Completely Balanced:

"An empty tree and a tree containing just one node are both completely balanced; any other tree is completely balanced if and only if the height of its left child is equal to the height of its right child, and its left and right children are also completely balanced."

---

**Diagram:**

```
      A
     / \   \       \   \   \       \   F
    B    C   D       G   E   H
   /     / \   \     /   / \   /
  E     D   C
```

---
i. How many nodes does a **completely balanced** tree of depth n contain (a tree consisting of a single node would be depth 1)?

A completely balanced tree will be full on every level.

<table>
<thead>
<tr>
<th>Tree Depth</th>
<th>Tree Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>15</td>
</tr>
</tbody>
</table>

The pattern is $1 + 2 + 4 + 8 \ldots$ So the answer can be $\text{SUM from } i = 0 \rightarrow \text{depth} - 1 \text{ of } n^i$. This can be simplified to $(2^\text{depth}) - 1$.

ii. Also recall the definition for Maximally Balanced:

"The isMaximallyBalanced method returns true either when the tree is completely balanced, or when all the leaves are on level k and level k+1 for some k, level k is full, and the leaves at level k+1 are as far left as possible, and returns false otherwise. (The level of a leaf is its distance from the root.)"

What's the fewest number of nodes a **maximally balanced** tree of depth n can have?

The smallest maximally balanced tree of depth n is a completely balanced tree of depth n-1 plus one more node. So using the formula from earlier we get $2^{\text{depth}-1}$.

c. (2P) Hashing

i. In 3 sentences or less what major effects would this hash code method have on a hash map with keys of type Something.

```java
public class Something{
    // Stuff
    ...
    public int hashCode(){
        return 3;
    }
}
```

All instances of Something will hash to the same value. This means that they will all end up in the same bucket and (assuming they are using linked lists for collisions within a bucket) the hash map will essentially be a linked list representation of a map.

ii. In 3 sentences or less what major effects would this hash code method have on a hash map with keys of type SomethingElse
public class SomethingElse{
    // Stuff
    ....
    public int hashCode(){
        // Return a random number under 10
        return Math.random(10);
    }
}

This is not a valid hash function since it is nondeterministic, meaning it can return different values when called on the same object multiple times. This can be a problem because you may put it in one bucket and search for it in another. This means the hash map won’t work at all.

d. (2P) Exceptions

public class Stuff{
    public static void main(String [ ] args){
        try {
            int temp = 5;
            throw new IllegalStateException("error");
            temp = 6;
        } catch (Exception e){
            System.out.println(temp);
        }
        System.out.println(temp + 1);
    }
}

The code will not compile because "int temp" is inside of the try and therefore cannot be seen outside of the try. This means when the compiler reaches System.out.println(temp + 1) it will not know what temp is and will give an error.
2. (12 points) Linked Lists

In this question, we will be adding methods to the `NumSet` class. It contains a single instance variable, `NSNode myHead`, which is the head of a sorted linked list of `int`s representing the set. Since it is a set, each `int` is unique, and since it is sorted, a number will come before all numbers greater than it.

```java
public class NumSet{
    private NSNode myHead;
}
```

The `NSNode` contains three methods you have access to. `getItem()` returns the `int` this node contains. `getNext()` returns the next `NSNode` in the list (or null if there is none). `setNext()` sets the value of the next node in the list.

```java
public class NSNode{
    private int myItem;
    private NSNode myNext;

    // Constructors and methods not available to you
    ...

    // Returns the item
    public int getItem(){
        return myItem;
    }

    // Returns the next
    public NSNode getNext(){
        return myNext;
    }

    // Returns
    public void setNext(NSNode next){
        myNext = next;
    }
}
```

Using these, we can create the `NumSets` shown below where `set1` represents the set `{1, 3, 4, 6, 13}` and `set2` represents the set `{3, 5, 6, 7}`:

```
set1

set2
```
a. (5P) Union
The union of two sets is a set that contains all of the elements in either of the two sets. For example the union of set1 {1,3,4,6,13} and set2 {3,5,6,7} is set3 {1,3,4,5,6,7,13}.

i. Inside of the NumSet class, write a method union() that takes a second NumSet as an argument and modifies this NumSet to represent the union of the two. Your solution must be destructive (you can never say new NSNNode()) meaning the second NumSet will no longer be a valid representations of a set. It also must be recursive (you cannot use any looping constructs and the helper must recursively call itself).

```java
void union(NumSet other){
    if (other == null || other.myHead == null){
        return;
    }
    if (myHead == null){
        myHead = other.myHead
        return;
    }
    if (myHead.getItem() < other.myHead.getItem()){;
        unionHelper(myHead, other.myHead);
    } else {
        myHead = unionHelper(other.myHead, myHead);
    }
}

static NSNNode unionHelper(NSNode ln1, NSNode ln2){
    if (ln2 == null){
        return ln1;
    }
    if (ln1.getNext() == null){
        ln1.setNext(ln2);
        return ln1;
    }
    if (ln1 getNext()).getItem() == ln2 getNext().getItem(){
        ln1.setNext(unionHelper(ln1 getNext(),ln2 getNext()));
        return ln1;
    }
    if (ln1 getNext()).getItem() < ln2 getNext().getItem(){
ln1.setNext(unionHelper(ln1.getNext(), ln2));
return ln1;
}

ln1.setNext(unionHelper(ln2, ln1.getNext());
return ln1;

ii. Use the following variables:
   s - the number of elements contained in both set1 and set2
   s1 - the number of elements in set 1
   s2 - the number of elements in set 2

   How many comparisons does your program make? S1 + S2
   (by comparisons I meant calls to <, which in this case for each
   comparison one node is discarded, so s1 + s2)

   How many calls to setNext() are made? The number of nodes in the
   final answer (s1 + s2 - s)

b. (5P) Intersection
   The intersection of two sets is the set that is contained within both sets. For
   example the intersection of set1 {1,3,4,6,13} and set2 {3,5,6,7} is set4{3,6}.

   set4

   i. Inside of the NumSet class write a method intersection() that takes in a
      second NumSet as an argument and changes the first NumSet to represent
      the intersection of those two sets. Once again your solution must be
      destructive (you can never say new) but this time set2 must not be
      changed. This time your method must be iterative (use some sort of
      looping construct) and CANNOT use recursion. Also, your worst case run
      time must be proportional to the sums of the lengths of the two lists.

void intersect(NumSet other){
   NSNode a = myHead;
   NSNode b = other.myHead;

   // Make sure that myHead is a node that we will keep
   while(a != null && b!= null){
      if (a.getItem() == b.getItem()){
         break;
      }else if (a.getItem() > b.getItem()){

c. (2P) More Fun
Here is the code for a method that should remove all elements present in the
second set from the first set (set1 == set1.unique(set2).union(set1.intersect(set2)) :
the union of the intersection of the two sets and what this function returns is the
original set). There is a bug in it. Give two example lists in which the bug is
revealed, and explain what the bug is.

```java
void unique(NumSet other){
    NSNode presentNode;
    // Move the head to the first remaining element
    for (presentNode = myHead;
         presentNode != null && other.contains(presentNode.getItem());
         presentNode = presentNode.getNext()){
        myHead = presentNode.getNext();
    }
    // Present node is either null, or a legal node
    for (;presentNode != null && presentNode.getNext() != null;
         presentNode = presentNode.getNext()){
        if (other.contains(presentNode.next().getItem())){
            presentNode.setNext(presentNode.next().getNext());
        }
    }
}
```

set5

// Checks whether an element is in the list, functions correctly
boolean contains(int other){
    ...
}

// Checks whether an element is in the list, functions correctly
boolean contains(int other){
    ...
}

// Checks whether an element is in the list, functions correctly
boolean contains(int other){
    ...
}
TestList 1 (use as many spaces as necessary):  1 3 4 5
TestList 2 (use as many spaces as necessary):  3 4

Generalize when the bug will occur:
The bug is that if two consecutive elements in the first list are present in the second list only one of them will be removed. This is because even if we remove a node we still increment the present node pointer.
3. (10 points) Special Nodes
   public class SpecialNode{
       private int myItem;
       private SpecialNode a;
       private SpecialNode b;
   }
   
a. (5P) Binary Tree to Sorted Doubly Linked List
   Imagine we are using SpecialNode to represent a Binary Search Tree, with
   SpecialNode one representing the left child and SpecialNode two representing
   the right child.

   Write a method that destructively (no calls to new) converts this to a sorted
doubly linked list where SpecialNode a represents the node before you and
SpecialNode b is the node after you (feel free to use a helper). The above tree
would become:

   There are a couple ways to do this question. I am going to do it using recursion,
another way would be to make an inorder iterator.

   public void convertToDoublyLinkedList(){
       if (a != null){
           // Recursively call convert on the left node
           a.convertToDoublyLinkedList();
           // Now go to the end of a's linked list and connect it to us
           SpecialNode temp = a;
           while(temp.b != null){
               temp = temp.b;
           }
           a = temp;
   }
temp.b = this;
}

if (b != null){
    // Recursively call convert on the right node
    b.convertToDoublyLinkedList();
    // Now go to the beginning of b's linked list and connect it to us
    temp = b;
    while (temp.a != null){
        temp = temp.a;
    }
    b = temp;
    temp.a = this;
}

b. (5P) Sorted Doubly Linked List to Binary Tree
Write a method that assumes this the head of a doubly linked list and creates a
binary search tree with all the same elements. The method should be
destructive. Also the tree should abide by the following rules, for every node in
the tree, the size of its left child must be at most 1 different than the size of its right
child.

public SpecialNode convertToBinarySearchTree(){
    // Figure out the size
    int size = 0;
    SpecialNode temp = this;
    while (temp != null){
        size++;
        temp = temp.b;
    }

    // Go to the middle
    temp = this;
    for (int i = 0; i < ((size + 1)/2); i++){
        temp = temp.b;
    }

    // Disconnect the left node, and reconnect it
    temp.a.b = null;
    temp.a = convertToBinarySearchTree();

    // Disconnect the right node, and reconnect it
    temp.b.a = null;
    temp.b = temp.b.convertToBinarySearchTree();
return this;

}
4. (4 points) Warm Down Fun

a. (1P) Stack and Queue
   i. q is Queue, what does this print
      q.enqueue("a");
      q.enqueue("b");
      q.enqueue("c");
      System.out.println(q.dequeue() + q.dequeue() + q.dequeue());
      abc (queues are first in first out)

   ii. s is a Stack, what does it print
      s.push("a");
      s.push("b");
      s.push("c");
      System.out.println(s.pop() + s.pop() + s.pop());
      cba (stacks are last in first out)

b. (2P) Run Time
   In terms of i and j, how many times does each snippet print hello:

   i. i * j
      for (;i > 0;i--){
         int j = something; // reset the value of i
         for (; j > 0; j--){
             System.out.println("hello");
         }
      }

   ii. i + j
      for (; i > 0; i--){
          System.out.println("hello");
      }
      for(; j > 0; j--){
          System.out.println("hello");
      }

   iii. log BASE 3 (i)
      for (; i > 1; i /= 3){
          System.out.println("hello");
      }

   c. (1P) Abstract Class or Interface
   Give one reason why you would use an abstract class over an interface.
Abstract classes can have concrete methods inside of them along with signatures.