Unless you qualify for extended time on exams, you have 2 hours to complete this exam once you start it. When you have finished the exam please write and sign the Honor Pledge on this page. Also give your starting and stopping times for the exam.

If you forget the name of something (oh, what does Java call the length of a String??) just write a note saying “I’m going to call this X” and then use that.

The 8 numbered questions are equally weighted.
1. Here is a list of data: 4 2 12 1 3 9 0 6. For each of the following structures I will walk through the list in order, add each item to the structure and then go into a loop in which I remove elements one at a time from the structure and print them as I remove them. **In what order do I print the items for**

A) **A stack.** Adding to the structure: push; removing from the structure: pop

B) **A queue.** Adding to the structure: offer; removing from the structure: poll

C) **A priority queue.** Adding to the structure: offer; removing from the structure: poll

D) In the add stage, I insert the values into a **BinarySearchTree** that starts off empty. So 4 becomes the root and we insert the other values around it. Skip the remove stage and just give a level-by-level listing of the final tree, as we did on Exam 2. If you don’t remember Exam 2, list the root on the first line, the root’s two children on the third line, and so forth.

E) **In the add stage I form a hash table of size 8** (the data fits; you don’t need to resize the table) with linear open addressing, using each data value as its own hash code (so the **hash value is the remainder when we divide the value by 8**). In the remove stage I remove and print the data at index 0, then the data at index 1, then index 2, etc.
2. In each part give a Big-Oh estimate of the worst-case time it takes to find an element in the given structure
   a) An AVL tree with n nodes
   b) A Binary Search tree with n nodes
   c) A sorted ArrayList with n nodes
   d) A sorted LinkedList with n nodes
   e) A directed graph with n nodes and m edges, using the graph structure from Lab 9, where the data is stored in the vertices so you need to search through the vertices.
3. Here is a wild recursive function.

```c
int collatz( int n ) {
    if (n == 1)
        return 1;
    else if (n%2 == 0)
        return 1+collatz(n/2);
    else
        return 1+collatz(3*n+1);
}
```

Explain in English how you would create a Dynamic Programming version of this function. Note that the arguments to collatz can become very large – larger than any table size. You can handle that any way you like (e.g., expanding the table or only using Dynamic Programming for some values of n), but you should handle it some way. What would you initialize your table entries to?
4. Here is an AVL tree:

In case you have forgotten what level-by-level listings look like, here is one for this tree:
100
50  200
30 70 150 300
20 130 250 400
220

Give either the AVL tree or a level-by-level listing of the AVL tree that results from inserting value 230 into this tree.
5. Here is a picture of a binary Heap represented as a tree:

If you prefer this could be represented as an array:

| 5 | 30 | 10 | 40 | 35 | 15 | 20 | 50 | 60 |

**Give an algorithm (in English, not in code) for deleting a node from this heap.** For example, we might want to delete the value 30. You can assume that we are starting at the node that needs to be deleted; you don’t have to find it. Note: we didn’t talk about this in class; I’m asking you to invent an algorithm.
6. Here is most of a class that implements queues with integer values:

```java
class MyQueue{

class Node {
    int data;
    Node next;
    Node(int d ) {  //NODE CONSTRUCTOR
        data = d;
        next = null;
    }
}

Node head, tail;

public MyQueue() {  //QUEUE CONSTRUCTOR
    head = null;
    tail = null;
}

public void offer(int x){ //INSERT INTO QUEUE
    if (head == null) {
        tail = new Node(x);
        head = tail;
    } else {
        tail.next = new Node(x);
        tail = tail.next;
    }
}
}
```

Give code for the following method which removes and returns the head of the queue:

```java
int poll() throws NoSuchElementException
```
7. We can represent binary trees with the following Node class:

```java
class Node {
    String name;
    Node left, right;
}
```

In this representation an empty tree is null; a leaf is a Node where left and right are both null.

a) **Write a function int height(Node t) that returns the length of the longest path from node t to a leaf.** The height of an empty tree is -1; the height of a leaf is 0.

b) The diameter of a tree is the length of the longest path from one leaf to another. This longest path may or may not go through the root. In the following pictures the tree on the left has diameter 3, and the tree on the right has diameter 6; the dark edges show longest paths.

```
The diameter of a leaf is 0, the diameter of an empty tree is -1.
Write int diameter(Node t) that returns the diameter of the tree whose root is t.
```
8. Finding shortest paths in a rectangular grid is easy -- the length of the shortest path from one node to another is the difference in their columns plus the difference in their rows. But this goes out the window if the grid is missing some of its `nodes. Consider the following grid where the black circles are existing nodes and the white circles are missing nodes:

Node A is at \([3,2]\) (row 3, column 2). Node B is at \([1,2]\). The shortest path from A to B has 6 steps: From A to \([3,1]\) to \([3,0]\) to \([2,0]\) to \([1,0]\) to \([1,1]\) to B. If the node at \([2,4]\) was live there would be a path of length 4 from A to B.

Suppose we have a grid with R rows and C columns, and a boolean function \(\text{Live}(r, c)\) that returns true if there is a live node at row \(r\), column \(c\) and false if there isn’t. You can also assume that \(\text{Live}(r, c)\) returns false if \(r<0\) or \(r \geq R\) or \(c<0\) or \(c \geq C\). Finally, you can assume that, like the Square class in Lab 3, each node knows its row and column location.

I am going to designate one existing node on the grid as BOB. **Give an algorithm that finds the length of the shortest path, if there is one, from BOB to every other existing node. Note that I am only asking for the lengths, not the actual paths.** If it helps you can modify the nodes to store any data you want.