First Name (PRINT): ____________________________________________________  
Last Name (PRINT): ____________________________________________________  
University ID: _________________________________________________________  

I pledge on my honor that I have not given or received any unauthorized assistance on this examination.  
Your signature: _________________________________________________________  

Instructions

- This exam is a closed-book and closed-notes exam.  
- Total point value is 100 points, 50 minutes exam.  
- Please use a pencil to complete the exam.  
- PUNT RULE: For any question, you may write PUNT, and you will get \( \frac{1}{4} \) of the points for the question (rounded down). If you feel totally lost on a question, you are encouraged to punt rather than write down an incorrect answer in hopes of getting some partial credit.  
- WRITE NEATLY. If we cannot understand your answer, we will not grade it (i.e., 0 credit).  

Grader Use Only

<table>
<thead>
<tr>
<th>#</th>
<th>Section</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>Algorithmic Complexity</td>
<td>(12)</td>
</tr>
<tr>
<td>#2</td>
<td>Program Correctness</td>
<td>(10)</td>
</tr>
<tr>
<td>#3</td>
<td>Hashing</td>
<td>(6)</td>
</tr>
<tr>
<td>#4</td>
<td>Language Features</td>
<td>(22)</td>
</tr>
<tr>
<td>#5</td>
<td>Recursion</td>
<td>(20)</td>
</tr>
<tr>
<td>#6</td>
<td>Sets and Maps</td>
<td>(20)</td>
</tr>
<tr>
<td>#7</td>
<td>Linear Data Structures</td>
<td>(20)</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>Total (110)</td>
<td>(110)</td>
</tr>
</tbody>
</table>
Problem 1 (12 pts) Algorithmic Complexity

1. (6 pts) Calculate the asymptotic complexity of the code snippets below (using big-O notation) with respect to the problem size $n$.

   a. for (int $i=1; i<=n/2; i++$) {
       System.out.println("Hello");
   }

   b. for (i =1; i<=10; i++) {
       for (t =1; t<=$n; t++$) {
           System.out.println("Hello");
       }
   }

   c. for (int $i=1; i<=n; i=i*2$) {
       for (int $k=1; k<=$n; k++$) {
           System.out.println("Hello");
       }
   }

2. (4 points) Give the asymptotic bound of the following functions:

   a. $6n^3 + n + n\log(n)$

   b. $n + n^2 + \log(n)$

3. (2 pts) List the following big-O expressions in order of asymptotic complexity (lowest complexity first)

   $O(n\log(n))$  $O(n^2)$  $O(2^n)$  $O(\log(n))$  $O(1)$
Problem 2 (10 pts) Program Correctness and Exceptions

1. (6 pts) The following code fragment throws a NumberFormatException when the user enters a non-integer value. Modify the following code fragment if the exception is thrown, the code calls `JOptionPane.showMessageDialog(null, "That wasn't an integer");` and then displays the input dialog again, until it successfully gets, parses and returns an int value.

   ```java
   static int getIntFromDialog() {
       int val = Integer.parseInt(
           JOptionPane.showInputDialog("Enter an integer"));

       return val;
   }
   }
   ```

2. (2 pts) When is a finally block executed?

3. (2 pts) What does it mean for a piece of code to have 100% code coverage?
Problem 3 (6 pts) Hashing

1. (2 pts) Name two properties of a good hash function.

2. (2 pts) Describe the Java hashCode Contract (i.e., what is required of a hashCode() implementation)?

3. (1 pt) What is a collision in a hash table?

4. (1 pt) What is usually returned by the default implementation of the hashCode() method?
Problem 4 (22 pts) Java Language Features

1. (2 pts) **T or F** → An inner class can only access public variables and methods of the enclosing class.
2. (2 pts) **T or F** → Java `enum` values are represented as integers.
3. (2 pts) **T or F** → Java `break` statements should be avoided in code written for CMSC 132H
4. (2 pts) **T or F** → An abstract class cannot have any constructors.
5. (2 pts) **T or F** → A class extending an abstract class will become abstract if abstract method(s) from the super class are not defined in the subclass.
6. (2 pts) **T or F** → You can pass a `ArrayList<String>` to a method that expects a `List<String>`
7. (2 pts) **T or F** → You can pass a `List<String>` to a method that expects a `List<Object>`
8. (6 pts) For each of the 6 table cells, state true or false as to whether the operation described can be performed on a parameter of the specified type:

<table>
<thead>
<tr>
<th>List&lt;? extends String&gt; lst</th>
<th>List&lt;? super String&gt; lst</th>
<th>List&lt;String&gt; lst</th>
</tr>
</thead>
<tbody>
<tr>
<td>String a = lst.remove(0);</td>
<td>lst.add(“foo”);</td>
<td></td>
</tr>
</tbody>
</table>

9. (2 pts) The **Game** interface defines a single method with the following signature: `public void move();`
   Complete the following assignment where `y` is assigned an object that implements the **Game** interface and the method `play()` will print (using `System.out.println`) the message “Game Move”.

   ```java
   Game y =
   ```
Problem 5 (20 pts) Recursion

**Nim** is a two-player mathematical game of strategy in which players take turns removing objects from distinct heaps. On each turn, a player must remove at least one object, and may remove any number of objects provided they all come from the same heap. In some games, there is a limit on the number of objects that can be taken in one turn. Nim is usually played as a *misère* game, in which the player to take the last object loses.

We consider a *misère* nim game with two heaps where in a turn you must take 1, 2 or 3 objects. The table to the right shows some winning/loosing positions (by the way, nim heaps have nothing to do with the heaps used for priority queues)

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>win</th>
<th>move</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>yes</td>
<td>0,1</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>yes</td>
<td>0,1</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>yes</td>
<td>1,0</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>yes</td>
<td>2,2</td>
</tr>
</tbody>
</table>

a) Write a recursive function for determining the winning position for a nim game with two heaps. static boolean nim(int a, int b) { … } that returns true if (a,b) is a winning position. Do not use any looping constructs. You don’t need to compute the move that should be taken if the position is a winning one.

b) What is the base case for your recursive function?

c) Write a more efficient solution that using dynamic programming and/or caching.
Problem 6 (20 pts) Sets and Maps

The StateRoads class keeps track of roads that are in a state. Each road is identified by a unique number.

```java
class StateRoads {
    Map<Integer, Set<String>> map;
    public StateRoads() { // YOU MUST IMPLEMENT }
    public void associateStateWithRoadNumber(int roadNumber, String state) { // YOU MUST IMPLEMENT }
    public Set<String> getStatesWithRoad(int roadNumber) { // YOU MUST IMPLEMENT }
}
```

**What You Must Implement**

1. (4 pts) Implement a constructor for the class that creates an empty map.

2. (13 pts) Implement the associateStateWithRoadNumber method that associates the state with the specified road.

3. (3 pts) Implement the getStatesWithRoad method which returns the set of states associated with the specified road. We should be able to print the elements present in the returned set in sorted order. It should return an empty set if no states are associated with the road.

**You may find the following Map methods helpful:**

- `V get(Object key)` - Returns the value to which this map maps the specified key.
- `V put(K key, V value)` - Associates the specified value with the specified key in this map.
- `Set<K> keySet()` - Returns a set view of the keys contained in this map.
- `boolean isEmpty()` - Returns true if this map contains no key-value mappings.

**You may find the following Set methods helpful:**

- `boolean contains(Object o)` - Returns true if this set contains the specified element.
- `boolean add(E o)` - Adds the specified element to this set if it is not already present
- `V remove(Object key)` - Removes the element from the set
- `boolean isEmpty()` - Returns true if this set contains no elements.
Problem 7 (20 pts) Linear Data Structures

Implement the methods below based on the following Java class definitions. You may not add any instance variables, static variables or auxiliary methods to the LinkedList class. In addition, you may not use the Java API LinkedList class.

```java
public class LinkedList<T extends Comparable<T>> {
    private class Node {
        private T data;
        private Node next;

        public Node(T data) {
            this.data = data;
            next = null;
        }
    }

    private Node head; /* List head pointer */

    public LinkedList() { // YOU MUST IMPLEMENT THIS METHOD }
    public T getElementAtIndex(int index) { // YOU MUST IMPLEMENT THIS METHOD }
    public boolean delete(T targetElement) { // YOU MUST IMPLEMENT THIS METHOD }
}
```

1. (2 pts) Implement a constructor that defines an empty list.
2. (6 pts) Implement the method `getElementAtIndex` that returns the element at the specified index position (e.g., first element is at index 0, second element is at index 1, etc.). The method will return null if the index is larger than or equal to the number of elements in the list.
3. (12 pts) Implement the method `delete` that removes the first instance of `targetElement` from the list. The method will return true if the element is deleted; false otherwise.