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");
   }  

   \[ f(n) = O(n) \]

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

   \[ f(n) = O(n) \]

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

   \[ f(n) = O(n \log n) \]

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

   a. \[ 6n^3 + n + n \log(n) \]  

   \[ f(n) = O(n^3) \]

   b. \[ n + n^2 + \log(n) \]  

   \[ f(n) = O(n^2) \]

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

   \[ O(1) \quad O(\log(n)) \quad O(n \log(n)) \quad O(n^2) \quad O(2^n) \]
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() {
       while (true) {
           try {
               int val = Integer.parseInt(JOptionPane.showInputDialog("Enter an integer");
               return val;
           } catch (NumberFormatException e) {
               JOptionPane.showMessageDialog(null, "That wasn't an integer");
           }
       }
   }
   ```

   or

   ```java
   static int getIntFromDialog() {
       try {
           int val = Integer.parseInt(JOptionPane.showInputDialog("Enter an integer");
           return val;
       } catch (NumberFormatException e) {
           JOptionPane.showMessageDialog(null, "That wasn't an integer");
           return getIntFromDialog();
       }
   }
   ```

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

   In a try-finally statement, it is executed when execution leaves the try block (no matter how execution levels the try statement). In a try-catch-finally, when execution leaves the try/catch construct (either after the try block if execution terminates normally, or after the catch block if the try block throws an exception which is caught in a catch clause.

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

   That all of the statements/branches are executed by the test cases. A necessarily but not sufficient condition for having a complete set of test cases.
Problem 3 (6 pts) Hashing

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

   Any two of:
   * Two unequal objects are likely to have unequal hash codes
   * Hash code values are uniformly distributed over the range of 32 bit integers
   * Satisfies the hashCode contract

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

   If two objects compare as equal, their hashCodes must be equal

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

   When two objects are mapped to the same hash table bucket.

4. (1 pt) What is usually returned by the default implementation of the hashCode() method?

   A value derived from the memory location of the object. Using the default implementation, two distinct objects are likely to have distinct hashCodes.
Problem 4 (22 pts) Java Language Features

1. (2 pts) F → An inner class can only access public variables and methods of the enclosing class.
2. (2 pts) F → Java enum values are represented as integers.
3. (2 pts) F → Java break statements should be avoided in code written for CMSC 132H
4. (2 pts) F → An abstract class cannot have any constructors.
5. (2 pts) 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 → You can pass a ArrayList<String> to a method that expects a List<String>
7. (2 pts) 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></th>
<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>true</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>lst.add(“foo”);</td>
<td>false</td>
<td>true</td>
<td>true</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 move() will print (using System.out.println) the message “Game Move”.

```java
Game y = new Game() {
    public void move() {
        System.out.println(“Game move”);
    }
}
```
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>4</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>yes</td>
<td>1,4</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.
a) static boolean nim(int a, int b) {
    if (a <= 0 && b <= 0)
        return true;
    for (int i = 1; i <= Math.min(3, a); i++)
        if (!nim(a - i, b))
            return true;
    for (int i = 1; i <= Math.min(3, b); i++)
        if (!nim(a, b - i))
            return true;
    return false;
}

b) a == 0 and b == 0

c) static boolean nimIterative(int a, int b) {
    boolean[][] win = new boolean[a + 1][b + 1];
    win[0][0] = true;
    for (int i = 0; i <= a; i++)
        for (int j = 0; j <= b; j++) {
            for (int iTake = 1; iTake <= 3 && i - iTake >= 0; iTake++)
                if (!win[i - iTake][j])
                    win[i][j] = true;
            for (int jTake = 1; jTake <= 3 && j - jTake >= 0; jTake++)
                if (!win[i][j - jTake])
                    win[i][j] = true;
        }
    return win[a][b];
}
Problem 6 (20 pts) Sets and Maps

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

```java
public 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.

```java
public class StateRoads {
    Map<Integer, Set<String>> map;

    public StateRoads() {
        map = new HashMap<Integer, Set<String>>();
    }

    public void associateStateWithRoadNumber(int roadNumber, String state) {
        Set<String> states = map.get(roadNumber);
        if (states == null) {
            states = new TreeSet<String>();
            map.put(roadNumber, states);
        }
        states.add(state);
    }

    public Set<String> getStatesWithRoad(int roadNumber) {
        Set<String> states = map.get(roadNumber);
        if (states == null)
            return Collections.emptySet();
        return states;
    }
}
```
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
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() {
        /** nothing needed */
    }
    public T getElementAtIndex(int index) {
        Node n = head;
        while (index-- > 0 && n != null) {
            n = n.next;
            if (n == null)
                return null;
        }
        return n.data;
    }
    public boolean delete(T targetElement) {
        if (head != null && head.data.equals(targetElement)) {
            head = head.next;
            return true;
        }
        Node n = head;
        while (n.next != null && !n.next.data.equals(targetElement)) {
            n = n.next;
            if (n.next == null)
                return false;
        }
        n.next = n.next.next;
        return true;
    }
}
```

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.