Name______________________________________

CS 210
Practice Final Exam

Instructions: closed books, closed notes, open minds,
2 hour time limit.
There are 3 sections.

Part I: Concepts
Part II: Comparison and Analysis of Common Data Structures
Part III: Using Data Structures

Part I: Definitions and Explanations   (about 35 points total)

1. Define each of the terms below.   (10 pts)

   interface, abstract class, class
   encapsulation, data hiding, polymorphism, dynamic binding
   maps, dictionaries, keys and values
   hash table, hash code, hash function, collisions

2. What is meant by FIFO and LIFO? Which describes a stack and which describes a queue?
   (2 pts)

3. List two data structures that would be appropriate choices for implementing a *stack*.   List two that would be inappropriate choices.   (2 pts)
   Appropriate:
   Inappropriate:

4. List two data structures that would be appropriate choices for implementing a *queue*. List two that would be inappropriate choices.   (2 pts)
   Appropriate:
   Inappropriate:

5. Do the *keys* in a Map abstract data type have to be unique? If so, why? Do the *values* have to be unique? If so, why? (3 pts)

6. A Map abstract data type can be implemented as a hash table (for example, the Java `HashMap` class). How is the relationship between the map *key* and *value* represented in a hash table? In other words, how do you determine where to add or retrieve a value in a hash table based on its key? (3 pts)

7. In hashing, what is a *collision*? Describe the two methods for dealing with collisions: *chaining* and *linear probing*. (3 pts)

8. What is the primary advantage of using a hash table? What are the constraints or disadvantages; why don't we just always use hash tables? (4 pts)

9. Given a preorder or postorder traversal and an inorder traversal of a binary tree as input, it is possible to determine if an actual tree can be constructed from the given data. If a tree cannot be constructed, there must be some inconsistency between the two traversals. (3 pts)

   a) Preorder: 3 2 1 5 4 8 7 6 9 10 13 12 11 14
   Inorder: 1 2 4 5 3 6 7 9 8 10 11 12 14 13
   Postorder: 1 4 5 2 6 9 7 11 14 12 13 10 8 3
   Draw the tree, using preorder and inorder traversals to guide you. (Hint: you know from the preorder traversal that 3 would have to be the root – why? Using the inorder traversal, determine what numbers are in the left and right subtrees of 3? Can you apply this process recursively?) Double-check your results using the postorder traversal. (3 pts)

   b) Preorder: 10 6 22 1 8 15 35 3 31 36
   Inorder: 1 3 6 8 10 15 22 31 35 36
   Explain why a tree cannot be constructed. What goes wrong? (3 pts)

10. Draw the BST of `Song` objects that would be generated for the following list of songs if entered in the order given. Assume that items in the BST are ordered based on the song title. (3 pts)

Data entered:

Song Title	**Artist Name**	**Album Name**	**Year Released**
Jump	Van Halen	1984	1984
Circle of Life	Billy Joel	Love Songs	1996
Time After Time	Cyndi Lauper	She's So Unusual	1984
Mr. Jones	Counting Crows	August and Everything After	1993
Nothing Man	Bruce Springsteen	The Rising	2002
Welcome to the Jungle	Guns N' Roses	Appetite for Destruction	1987
Billie Jean	Michael Jackson	Thriller	1983

Binary Search Tree: (for each node, just show the song title & year, separated by a comma)

11. The following `removeLeftmost` method in the `BinarySearchTree` class is NOT correct. Identify the test case that fails. (2 pts)

```
private Comparable removeLeftmost()
{
      if ( left.isEmpty() )
      {
            Comparable dataToReturn = (Comprable) data;
            data = right.getData();
            left = right.leftTree();
            right = right.rightTree();
            return dataToReturn;
      }
      else
            return left.removeLeftmost();
}
```

Part II: Comparison and Analysis of Common Data Structures (18 points total)

12. For each data structure below, discuss the time complexity to add, remove, and search for elements in the data structure. (9 pts)

Ordered ArrayList	Binary Tree (not a BST)
Unordered ArrayList	BST
Ordered LinkedList	HashMap
Unordered LinkedList	TreeMap

13. Design question: Consider a variation of the queue, called a priority queue. The basic idea of this structure is similar to that of a queue, in that we have elements waiting in line for a service. It differs from a queue in that selection is not on a strictly first-come-first-serve basis; it also takes into account some type of priority. For example, patients in an emergency waiting room are treated based on the severity of their injuries, not on when they arrived. In air traffic control, there is a queue of planes waiting to land, but the controller can move a plane to the front of the line if the plane is low on fuel, or there is a sick passenger on board.

    Describe a data structure to implement a PriorityQueue based on one or more of the data structures we have studied this quarter (you may draw a picture, if it helps). How would you add and remove items from your data structure so that it preserves its priority queue nature?

    (9 pts)

Part III: Using Data Structures

14. Write a method, `merge`, that takes two ordered lists as parameters and returns a new linked list that contains the elements of the two lists in order. (The order is determined by the `compareTo` method for the element class.) For example, if one list contained the elements B E N T and the other list contained the elements F I R S, the resulting list would be B E F I N R S T. If the input lists have different sizes, the new list should alternate for the number of elements in the smaller list, and then the rest of the elements of the longer list should be added to the new list. For example, if one list contained the elements B E N T, and the other list contained I S, the new list would be B E I N S T.

```
// merges two ordered lists, list1 and list2, into one ordered list
// precondition: T satisfies the Comparable interface
public List<T> merge (List<T> list1, List<T> list2)
{
```

Questions 15 - 16 refer to the `Song` and `PlayListTree` classes on the supplemental sheet.

15. Implement the `inventory` method in the `PlayListTree` class provided on the supplemental sheet. Method `inventory` should print the song title, artist name, album name, and year released for each song in the playlist, **in alphabetical order by song title**.

```
// prints the elements of the tree in order
public void inventory ()
{







}
```

16. Implement the `getArtist` method in the `PlayListTree` class provided on the supplemental sheet. Method `getArtist` should find the `Song` object in the tree with the specified title and return the name of the artist for this song. If the playlist does not contain the specified song, `getArtist` should return `null`.

```
// Determines the artist name of a particular song in the tree
public String getArtist(String songTitle)
{







}
```

17. Suppose we have a `HashMap` containing inventory records for a grocery store. The key is the UPC # of each item, and the value is a `GroceryItem` object (see below) that contains the name, UPC #, and quantity. We know that the `HashMap` class has no iterators. In order to traverse through the structure, we have to use one of the `keySet()`, `values()`, or `entrySet()` methods, and the iterator from the collection that gets returned. Write the code to print out the name and UPC # of each item in the `HashMap` three times, once for each method.

```
HashMap<BigInteger,GroceryItem> inventoryMap;
inventoryMap = new HashMap<BigInteger, GroceryItem>();         // Map is created
GroceryItem item1 = new GroceryItem("Special K Protein Bars",new BigInteger("3800029172"),5);
GroceryItem item2 = new GroceryItem("Jiff Peanut Butter",new BigInteger("5150024177"),7);
GroceryItem item3 = new GroceryItem("O.R. Gourmet Popcorn",new BigInteger("2700048770"),10);
GroceryItem item4 = new GroceryItem("Aunt Millie Bread",new BigInteger("7131400263"),3);
GroceryItem item5 = new GroceryItem("Campbell's chicken soup",new BigInteger"5100001251"),8);

inventoryMap.put(item1.getUPC(),item1);
inventoryMap.put(item2.getUPC(),item2);
inventoryMap.put(item3.getUPC(),item3);
inventoryMap.put(item4.getUPC(),item4);
inventoryMap.put(item5.getUPC(),item5);

// your code to print out name and UPC #
```

```
public class GroceryItem
{
       private String name;
       private BigInteger UPC;
       private int quantity;

// accessor methods
       public String name() { return name; }
       public BigInteger getUPC() {  return UPC; }
       public int quantity() {  return quantity;  }
}
```

18. Suppose the `GroceryItem` objects from the previous problem were put into a `TreeMap` instead of a `HashMap`. Would the code you wrote to print the names and UPC's be different? If so, how? In what order would the objects be printed? Is this different from the order given by the `HashMap`? Which would be a better structure to use, given the nature of the data? Why?