Points off	1	2	3	4	5		Total off	Net Score

CS 307 – Final – Spring 2008

Name_____

UTEID login name _____

Instructions:

1. Please turn off your cell phones.
2. There are 5 questions on this test.
3. You have 3 hours to complete the test.
4. You may not use a calculator on the test.
5. When code is required, write Java code.
6. You may add helper methods if you wish when answering coding questions.
7. When answering coding questions assume the preconditions of the methods are met.
8. There is a quick reference sheet with some of the classes from the Java standard library attached to the test.

1. (1.5 point each, 30 points total) Short answer. Place you answers on the attached answer sheet.

For questions that ask what is the output:

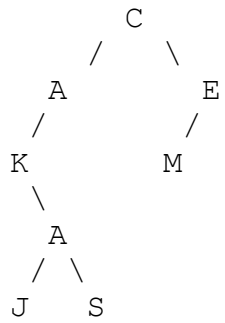
- If the code contains a syntax error or other compile error, answer "Compiler error".
- If the code would result in a runtime error or exception answer "Runtime error".
- If the code results in an infinite loop answer "Infinite loop".

On questions that ask for the Big O of a method or algorithm, recall that when asked for Big O your answer should be the most restrictive Big O function. For example Selection Sort has an expected case Big O of O(N^2), but per the formal definition of Big O it is correct to say Selection Sort also has a Big O of O(N^3). Give the most restrictive, correct Big O function. (Closest without going under.)

A. The following values are inserted one at a time in the order shown into an initially empty binary search tree using the traditional insertion algorithm. Draw the resulting tree.

```
7    0    -5    5    3
```

Consider the following binary tree:



B. What is the result of a pre-order traversal of the binary tree?

C. What is the result of a in-order traversal of the binary tree?

D. What is the result of a post-order traversal of the binary tree?

E. Is the tree shown above a binary search tree?

F. What is the height of the tree shown above?

G. What is the average case Big O for adding an element to a Red Black Tree that already contains N elements?

H. The following Java code will result in a syntax error. Explain the cause of the syntax error

```
ArrayList list = new ArrayList();
list.add("Texas");
list.add("Nevada");
System.out.println( list.get(1).charAt( 2 ) );
```

I. The following code attempts to determine if an element is present in a list but has a logic error. Explain what the logic error is.

```
public boolean isPresent(ArrayList<String> list, String tgt){
     boolean found = false;
     Iterator<String> it = list.iterator();
     while( it.hasNext() )
          found = found || it.next().equals( tgt );
     return found;
}
```

J. What is the output of the following code?

```
String data = "HAT";
Queue<Character> q = new Queue<Character>();
Stack<Character> st = new Stack<Character>();
for(int i = 0; i < data.length; i++){
     q.enqueue( data.charAt(i) );
     st.push( data.charAt(i) );
}
while( !q.isEmpty() ){
     System.out.print( st.pop() );
     System.out.print( q.dequeue() );
}
```

K. An `UnsortedSet` uses an `ArrayList` as its internal storage container. The `UnsortedSet` contains N `String`s. The `UnsortedSet` includes a method to find the minimum value in the UnsortedSet. What is the expected Big O of the method?

L. An `SortedSet` uses a Java `TreeSet` as its internal storage container. The `SortedSet` contains N `String`s. The `SortedSet` includes a method to find the maximum value in the SortedSet. What is the expected Big O of the method?

M. What is the Big O of the following method? The `LinkedList` initially contains N items.

```
public void addSome(LinkedList<Integer> list){
     for(int i = 0; i < 5; i++)
          list.add( list.size() / 2, i );
}
```

N. What is the Big O of the following method? The `ArrayList` initially contains N items.

```
public void removeAllVersion1(ArrayList<String> list){
     int limit = list.size();
     for(int i = 0; i < limit; i++)
          list.remove( list.size() - 1 );
}
```

O. What is the Big O of the following method? The `ArrayList` initially contains N items.

```
public void removeAllVersion2(ArrayList<String> list){
     while( !list.isEmpty() )
          list.remove( 0 );
}
```

P. What is the Big O of the following method? The `BST` class is a binary search tree that uses the traditional insertion algorithm.

```
public BST<Integer> createTree(int n){
     BST<Integer> result = new BST<Integer>();
     for(int i = 0; i < n * 3; i++)
          result.add( i );
     return result;
}
```

Q. You are using a sorting method that uses the insertion sort.
It takes the method 1 second to sort 1000 items that are initially in random order.
How long do you expect the method to take to sort 4000 items that are initially in random order?

R. You have 1,000,000 integers in an array that you need to search.
The integers are currently unsorted.
You expect to have to perform 10,000 searches before the data is changed.
Should you sort the data or not before doing the searches?
Justify your answer with calculations.

S. You need to encode the integers between 10,000 and 12,000. What is the minimum number of bits needed per integer to do this?

T. What does the following postfix expression evaluate to?

```
5 10 + 8 2 + *
```

2. (Binary Trees, 20 points) This question has two parts.

Part A. (6 points) Complete a constructor for a `HuffmanTree` class. The `HuffmanTree` class contains a Huffman code tree. It has a single instance variable, a `HuffNode` object, which is the root of the Huffman code tree.

The constructor has one parameter a `Map`. A *Map* is a data structure that stores *key-value pairs*. Each key is associated with a value. The methods for the `Map` class are listed on the quick reference sheet included with the exam.

The `Map` sent to the `HuffmanTree` constructor has keys of type `Character`, the wrapper class for primitive `chars`. The `char` value of a `Character` object may be accessed by the method `charValue` or via auto un-boxing. The values of type `String`. Each `String` will have a length greater than or equal to 1 and consist only of the `chars` `1` and `0`. The value for a given key is the Huffman code for that character.

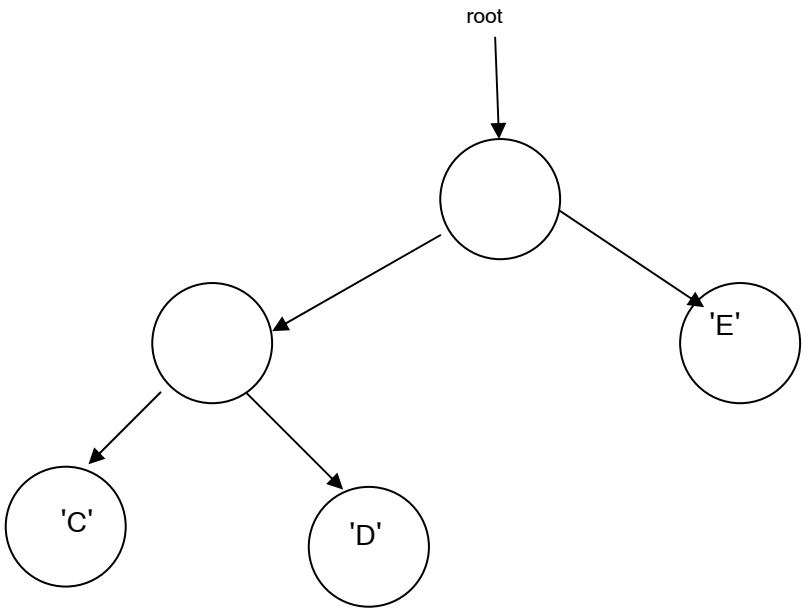
In the constructor call the private method `addCharacter` for each `Character` and `String` in the parameter `Map`. Do not write the method `addCharacter` in part A.

```
public class HuffmanTree{
     private HuffNode root;

     /*   pre: code != null
          All values in code have a length greater than or equal
          to 1.
          Values only contain the chars '1' and '0'.
     */
     public HuffmanTree( Map<Character, String> code ){
          // Assume assertions checked here.
          // Complete the method below.
```

Part B. (14 points) Complete the private instance method `addCharacter` for the `HuffmanTree` class. This method adds a given character to the tree based on its code.

Recall that each charcater in a Huffman code tree is placed in the binary tree based on its code. Starting at the root a 0 indicates a move to the left child and a 1 indictaes a move to the right child. By starting at the root of the tree and following the directions of the code you will reach the leaf containing the associated character.

For example given the following small,but complete Huffman code tree the code for `'C'` is 00., the code for `'D'` is 01, and the code for `'E'` is 1.



You are completeting the method to build the tree by adding a new character and the needed nodes to the tree. Before you add the first character the instance variable root will equal `null`. The number of nodes you have to add to the tree will vary for each character added.

Here is the `HuffNode` class:

```
public class HuffNode
{   public HuffNode()
    public HuffNode(char ch) //makes left and right children null
    public HuffNode(char ch, HuffNode left, HuffNode right)

    public char getChar()
    public HuffNode getLeft()
    public HuffNode getRight()

    public void setLeft(HuffNode left)
    public void setRight(HuffNode right)
}
```

Complete the following method:

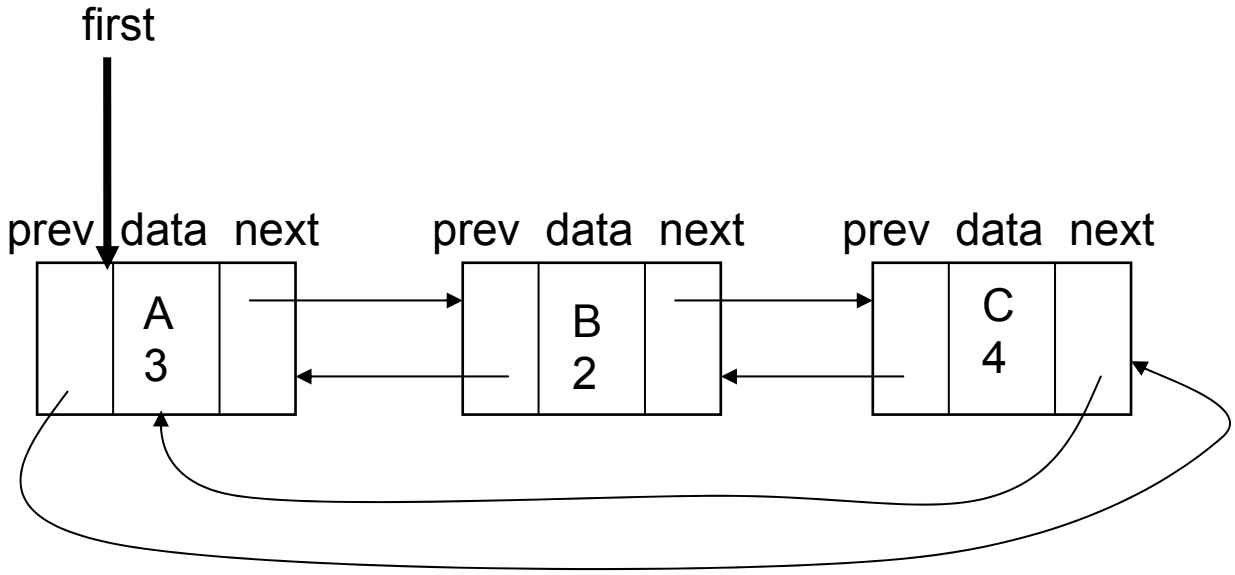
```
/* pre: code != null, code only contains the chars '1' and '0',
   code.length() >= 1
   post: ch added to this HuffmanTree.
*/
private void addCharacter(char ch, String chCode){
     // Assume preconditions checked here.
```

3. (Linked List, 20 points) Complete a method to remove an element from a multi-set. A *multi-set* is like a set except that duplicate items may appear. So for example (A, C, B, A, A, C, C, C, B) is a multi-set.

In this question the `MultiSet` class uses a doubly linked list as its internal storage container. The `MultiSet` class is generic based on the data type `Object`.

This `MultiSet` class uses doubly linked nodes for its linked list. This `MultiSet` class only contains a reference to the first node in the list, but the list is *circular*. The `MultiSet` class does not use a separate LinkedList class. It maintains the linked list itself.

A circular link list is implemented so that the first node in list has its `previous` reference set to refer to the last node in the list, instead of `null`. The last node in the list has its `next` reference set to the first node in the list, instead of `null`.



The elements of a multi-set do not have a definite order. Thus it is not necessary to store elements that appear more than once multiple times. Instead each node has two pieces of data: the data being stored, such as A, B, or C as in the above example, and the number of times it appears in the multi-set. So the multi-set (A, C, B, A, A, C, C, C, B) would be represented by the above linked list.

In your remove method if an element appears more than once its count will be decremented by 1. If an item only appears once that node will be removed from the linked list. It is possible that the element sent to the remove method does not appear in the `MultiSet`'s linked list in which case your method makes no changes. You must also make the appropriate change to the `MultiSet`'s `size` instance variable.

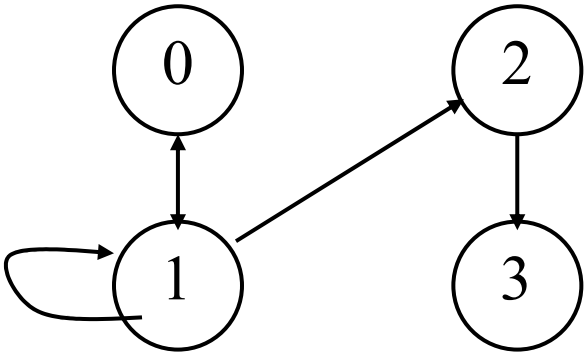
Your method must be O(1) space. You cannot use any other methods in the `MultiSet` class. You may write your own helper methods if you wish.

Here is the doubly linked node class the `MultiSet` class uses.

```
public class Node
{     // create a node with a count of 1
      public Node(Object item, Node next, Node previous)

      public Object getData()
      public int getCount()
      public Node getNext()
      public Node getPrevious()

      public void setData(Object item)
      public void setNext(Node next)
      public void setPrevious(Node previous)
      public void decrementCount() // decrease this node's count by 1
      public void incremenetCount() // increment this node's count by 1
}

public class MultiSet
{
      // points to the first node in this MultiSet's linked list
      private Node first;
      private int size; // size of list
      // when size == 0, first == null

      // pre: obj != null
      // post: remove one occurrence of obj from this multiSet if obj is
      // present. If obj is not present do nothing
      // post: return true if this MultiSet changed as a result of this
      // method call, false otherwise.
      public boolean remove(Object obj){
            assert obj != null;
```

// more room on next page

4. (Implementing and using data structures, 20 points) Complete a method to determine if it is possible to move from one node in a graph to another node in the graph within a given number of steps. A *graph* is a data structure similar to a tree. The graph consists of nodes and links between nodes. (On the word ladder assignment the words could be considered the nodes in a graph and a link between nodes existed if the words were one letter different.) The links may either be *directed*, indicating an allowed direction of travel between two nodes or *undirected*, indicating you can freely move back and forth between two nodes that have a link between them. In this question you will be working with a directed graph.



Simple directed graph with 4 nodes.

One way of representing a graph is with adjacency matrix. The number of rows and columns in the matrix is equal to the number of nodes in the graph. An element in the matrix is set to 1 if a link exists from the node specified by the row to the node specified by the column.

	0	1	2	3
0	0	1	0	0
1	1	1	1	0
2	0	0	0	1
3	0	0	0	0

In this example it is not possible to travel from a node to itself unless the link explicitly exists, thus the 0s along the diagonal except for node 1.

In the `Graph` class you are working with the nodes are designated by numbers 0 to N-1. These correspond to the rows and columns of the adjacency matrix. The adjacency matrix stores `booleans` not `ints`.

Write a method to determine if it is possible to move from one node in a graph to another node within a given number of steps.

You may use whatever classes you wish. There are no time or space restrictions.

Do not assume any other methods exist in the Graph class besides those shown.

You will complete the following method:

```
public boolean pathExists(int startNode, int tgtNode, int steps)
```

Examples given the above graph

```
pathExists(0, 0, 1) -> false     pathExists(2, 3, 1) -> true
pathExists(5, 0, 1) -> false     pathExists(1, 1, 0) -> false
pathExists(0, 1, 2) -> true      pathExists(1, 1, 1) -> true
pathExists(0, 1, 1) -> true      pathExists(1, 1, 2) -> true
pathExists(0, 1, 0) -> false     pathExists(10, 5, 2) -> false
pathExists(3, 2, 1) -> false     pathExists(0, 3, 3) -> true
```

Here is the `Graph` class that contains your method.

```
public class Graph{

     private boolean[][] adjacencyMatrix;
     // adjacencyMatrix will be a N x N matrix where N is the
     // number of nodes in this Graph. Elements are set to
     // true if a link exists from the node specified by the
     // row to the node specified by the column, false otherwise.

// Complete the following method
/* pre: none
   post: return true if it is possible to move from startNode to
     tgtNode by following no more than steps links. Return false
     otherwise.
*/
public boolean pathExists(int startNode, int tgtNode, int steps){
     // The instance variable adjacencyMatrix has already been
     // initialized.
```

```
// more space on next page if necessary
```

5. (using data structures 10 points) Write a method to determine if the top and bottom elements of a given Stack are equal to each other.

- You not use any other classes expect `Object`, `Queue`, and `Stack`.
- You may use temporary `Stack`s and `Queue`s if you wish.
- The `Stack` is restored to its original state when the method is completed.
- The method is not part of the `Stack` class.
- The `Stack` class only has the traditional stack methods. If you use a `Queue` assume it only has the traditional `Queue` methods. Assume the `Stack` and `Queue` classes are made generic using `Object` and not the type parameter syntax.

```
/* pre: st != null
   post: return true if the top and bottom elements of this Stack
     are equal to each other, false otherwise.
     If the Stack is empty return false.
     If the Stack has one item return true.
     The Stack st is restored to its original state.
*/
public boolean topAndBottomEqual(Stack st){
     assert st != null;
```

Name:_____

Answer sheet for question 1, short answer questions. Put answers next to the letter.

A.

B.

C.

D.

E.

F.

G.

H.

I.

J.

K.

L.

M.

N.

O.

P.

Q.

R.

S.

T.

Java class reference sheet. Throughout this test, assume that the following classes and methods are available.

class Object
// all classes inherit and may override these methods
• boolean equals(Object other)
• String toString()
• int hashCode()

interface Comparable
• int compareTo(Object other)
// return value < 0 if this is less than other
// return value = 0 if this is equal to other
// return value > 0 if this is greater than other

class Integer implements Comparable
• Integer(int value) // constructor
• int intValue()

class Double implements Comparable
• Double(double value) // constructor
• double doubleValue()

class String implements Comparable
• int length()
• String substring(int from, int to)
// returns the substring beginning at from
// and ending at to-1

• String substring(int from)
// returns substring(from, length())

• int indexOf(String s)
// returns the index of the first occurrence of s;
// returns -1 if not found

class Math
• static int abs(int x)
• static double abs(double x)
• static double pow(double base, double exponent)
• static double sqrt(double x)

class Random
• int nextInt(int n)
// returns an integer in the range from 0 to n-1 inclusive
• double nextDouble()

// returns a double in the range [0.0, 1.0)

interface List<AnyType>
• int size()
• boolean add(AnyType x)
// appends x to end of list; returns true

• AnyType get(int index)
• boolean contains(AnyType elem)
// returns true if elem is present in this List

• AnyType set(int index, AnyType x)
// replaces the element at index with x
// returns the element formerly at the specified position

• Iterator iterator()

class ArrayList<AnyType> implements List
Methods in addition to the List methods:
• void add(int index, AnyType x)
// inserts x at position index, sliding elements
// at position index and higher to the right
// (adds 1 to their indices) and adjusts size

• AnyType remove(int index)
// removes element from position index, sliding elements
// at position index + 1 and higher to the left
// (subtracts 1 from their indices) and adjusts size
// returns the element formerly at the specified position

class LinkedList<AnyType> implements List
Methods in addition to the List methods:
• void addFirst(AnyType x)
• void addLast(Object x)
• Object getFirst()
• Object getLast()
• Object removeFirst()
• Object removeLast()

interface Set<AnyType>
• boolean add(AnyType x)
• boolean contains(AnyType x)
• boolean remove(AnyType x)
• int size()
• Iterator iterator()

class HashSet<AnyType> implements Set
class TreeSet<AnyType> implements Set

interface Map<KeyType, ValType>
• Object put(KeyType key, ValType value)
// associates key with value
// returns the value formerly associated with key
// or null if key was not in the map

• ValType get(KeyType key)
• ValType remove(KeyType key)
• boolean containsKey(KeyType key)
• int size()
• Set<KeyType> keySet() //get a Set of all keys in this map

class HashMap<KeyType, ValType> implements Map
class TreeMap<KeyType, ValType> implements Map

interface Iterator<AnyType>
• boolean hasNext()
• AnyType next()
• void remove()