Points off	1	2	3	4	5	6	Total off	Net Score

CS 307 – Final – Fall 2007

Name_____

UTEID login name _____

Instructions:

1. Please turn off your cell phones.
2. There are 6 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. Ensure your answers are legible.
7. You may add helper methods if you wish when answering coding questions.
8. When answering coding questions assume the preconditions of the methods are met.
9. 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:

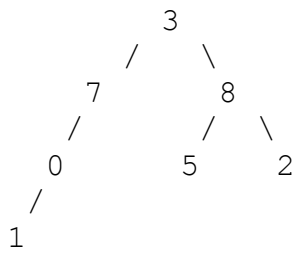
- If the code contains a syntax error or other compile error, answer "Compiler error".
- If the code would result in a runtime error or exception answer "Runtime error".
- If the code results in an infinite loop answer "Infinite loop".

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 1 at a time in the order shown into an initially empty binary search tree using the traditional algorithm. Draw the resulting tree.

```
16  7  -5  0  25
```

Consider the following binary tree:

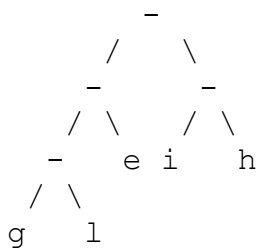


B. What is the result of a pre-order traversal of the binary tree?

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?

Consider the following Huffman code tree. Characters only exist in nodes that are leaves. Nodes with characters are represented with a dash. '-'



E. Given the above Huffman code tree, what is the binary code for the word "`lie`"?

F. What is output by the following code?

```
Stack<Character> s1 = new Stack<Character>();
String name = "isabelle";
for(int i = 0; i < name.length(); i++)
     if( name.charAt(i) > 'c' )
          s1.push( name.charAt(i) );

while( !s1.isEmpty() )
     System.out.print( s1.pop() );
```

G. Consider a linked list that uses singly linked nodes and maintains a reference to the first node and last node in the list. What is the Big O of removing the last element of the list? The list has N elements in it.

H. What is the worst case height of a binary search tree that contains N elements and was created using the naïve insertion algorithm.

I. What is the worst case height of a binary search tree that contains N elements and was created using the red-black tree insertion algorithm.

J. What is output by the following code?

```
int[] data = {5, 8, 2, 5};
Queue<Integer> q1 = new Queue<Integer>();

for(int i = 0; i < data.length; i++){
     if( data[i] % 2 != 0 )
          q1.enqueue( data[i] );
     q1.enqueue( data[i] );
}

while( !q1.isEmpty() )
     System.out.print( q1.dequeue() );
```

K. You are developing a program and you need to represent 60 distinct elements of a new data type. What is the minimum number of bits necessary to represent 60 distinct items?

L. The following code results in a syntax error. Briefly explain why.

```
ArrayList<String> ns = new ArrayList<String>();
ns.add( "A" );
ns.add( "B" );
ns.add( 12 );
ns.add( 0, "C" );
```

M. What is output by the following code?

```
ArrayList<Character> nums = new ArrayList<Character>();
nums.add( 'a' );
nums.add( 'c' );
nums.add( 0, 'e' ); // first parameter is position to add at.
nums.add( 'a' );
nums.add( 2, 'b' );
nums.remove( 3 );
for( Character c : nums )
     System.out.print( c );
```

For questions N and O consider an array based List class and the add method for that class.

```
public class List{

     private Object[] myCon;
     private int mySize;

     public void add(Object x){
          if( mySize == myCon.length )
               resize();
          myCon[mySize] = x;
          mySize++;
     }

     //other methods not shown
}
```

The Big O of the `System.arraycopy` method is equal to the last parameter in the method call.

N. Given the following `resize` method for the `List` class, what is the average case Big O of the `add` method for a list that contains N items?

```
private void resize(){
     Object[] temp = new Object[ (mySize * 3) / 2 ];
     System.arraycopy(myCon, 0, temp, 0, myCon.length);
     //(source, source-pos, destination, dest-pos, num elements)
     myCon = temp;
}
```

O. Given the following `resize` method for the `List` class, what is the average case Big O of the `add` method for a list that contains N items?

```
private void resize(){
     Object[] temp = new Object[ mySize + 50 ];
     System.arraycopy(myCon, 0, temp, 0, myCon.length);
     //(source, source-pos, destination, dest-pos, num elements)
     myCon = temp;
}
```

P. What is the T(N) of method `hen` ? Recall T(N) is a function that describes the actual number of executable statements in terms of the amount of data. Assume N = `list.length`.

```
public void hen(int[] list){
     for(int i = 0; i < list.length; i += 4)
          list[i] = list[i] * 2;

     for(int i = 0; i < list.length; i += 3)
          list[i] = list[i] / 2;
}
```

Q. What is output by the following code?

```
Set<Integer> st = new TreeSet<Integer>();
st.add( 3 );
st.add( 2 );
st.add( 5 );
st.add( 2 );
st.add( 6 );

for(int i : st )
     System.out.print( i );
```

R. Name a data structures that is typically used as the internal storage container when implementing a Stack.

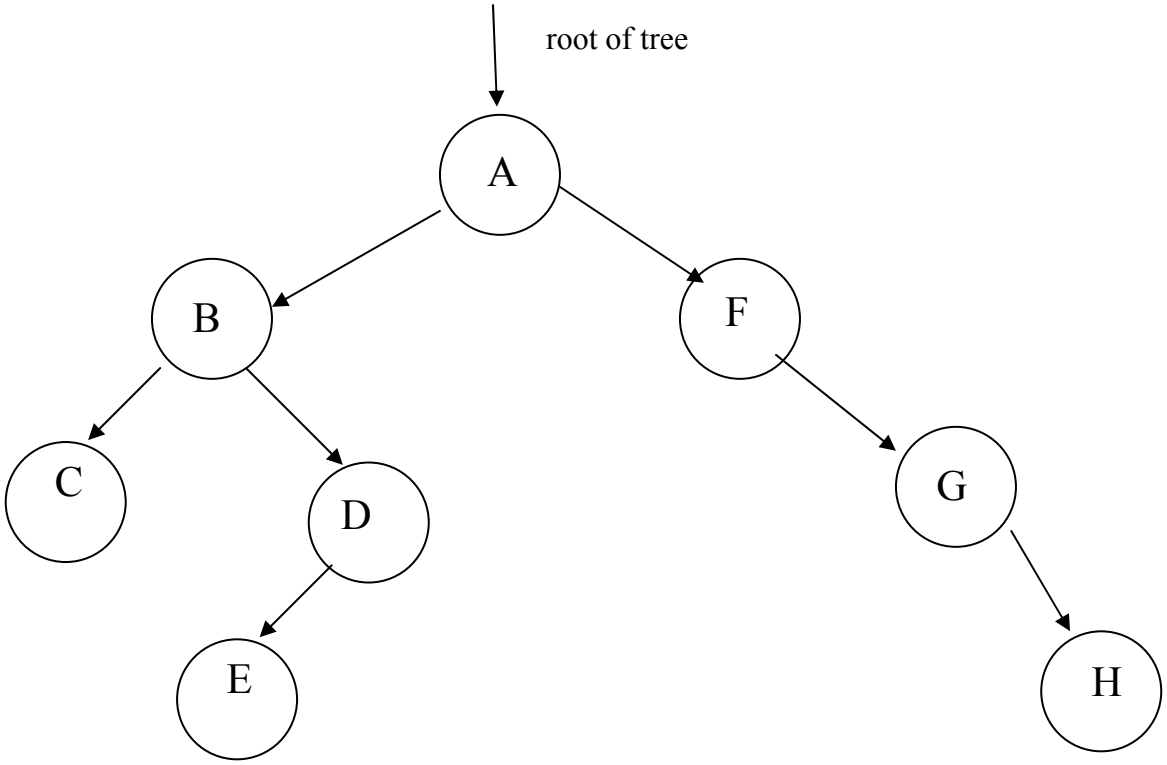
S. What is the average case Big O for finding the maximum value in a binary search tree created using the naïve insertion algorithm?

T. Can a native array be used to efficiently (all operations O(1)) implement a Queue?

2. (Binary Trees, 15 points) Complete a public instance method, `nodesAtDepth`, for a `BinaryTree` class. The `nodesAtDepth` method has one parameter, an `int`, that is the target depth. The method returns the number of nodes in the tree that are at the target depth. Recall that the depth of a node is the number of links from the root of the tree to the node.

Important. You method shall be no worse than O(h) space where h is the height of the tree. You may not assume or use any other methods in the `BinaryTree` class although you may write your own helper methods if you wish.

Here is an example. Consider the following binary tree.



- There is 1 node with a depth of 0 (A)
- There are 2 nodes with a depth of 1 (B, F)
- There are 3 nodes with a depth of 2 (C, D, G)
- There are 2 nodes with a depth of 3 (E, H)
- There are 0 nodes with a depth of 4

Here is the `BinaryTree` class:

```
public class BinaryTree
{     private TreeNode root; // if size == 0, root == null
      private int size;

      // returns the number of nodes in this binary tree
      public int getSize()

      // to be completed by the student
      /* return the number of nodes in this BinaryTree at the given depth
         pre: targetDepth > -1
      */
      public int nodesAtDepth(int targetDepth)

      // other methods not shown
}
```

Here is the `TreeNode` class:

```
public class TreeNode
{    public TreeNode()
     public TreeNode(Object initValue)
     public TreeNode(Object initValue, TreeNode initLeft,
                     TreeNode initRight)

     public Object getValue()
     public TreeNode getLeft()
     public TreeNode getRight()

     public void setValue(Object theNewValue)
     public void setLeft(TreeNode theNewLeft)
     public void setRight(TreeNode theNewRight)
}
```

```
/* return the number of nodes in this BinaryTree at the given depth
   pre: targetDepth > -1
*/
public int nodesAtDepth(int targetDepth){
     assert targetDepth > -1;
```

Scratch paper

3. (Linked List, 15 points) Write an instance method for a `LinkedList` class named `removeLastOccurrence`. The method removes the last occurrence of a value from the `LinkedList`. The method returns true if the list changed as a result of the method call, false otherwise.

This `LinkedList` class uses doubly linked nodes. This `LinkedList` class only contains a reference to the first node in the list, but the list is *circular*.

A circular link list is implemented so that the first node in list, the node that `head` refers to, 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`.

Examples:

```
[A, A, B, C, A, B].removeLastOccurrence(A) -> [A, A, B, C, B]
[A, A, B, C, A, B].removeLastOccurrence(B) -> [A, A, B, C, A]
[A, A, B, C, A, B].removeLastOccurrence(C) -> [A, A, B, A, B]
[D, A, B].removeLastOccurrence(D) -> [A, B]
[D, A, B].removeLastOccurrence(C) -> [D, A, B]
[].removeLastOccurrence(A) -> []
```

You may not use any other methods in the `LinkedList`. You may create your own helper methods if you wish.

You solution must be O(1) space. Your method should be as efficient time wise as possible.

```
public class Node
{     public Node(Object item, Node next, Node previous)

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

      public void setData(Object item)
      public void setNext(Node next)
      public void setPrevious(Node previous)
}
```

```
public class LinkedList
{     private Node head; // points to the first node in the list
      private int size; // size of list
      // when size == 0, head == null

      // pre: obj != null
      // post: remove the last occurrence of obj in this list.
      // return true if this LinkedList changed as a result of this
      // method call, false otherwise.
      public boolean removeLastOccurrence(Object obj){
      // complete this method on the next page
```

```
// pre: obj != null
// post: remove the last occurrence of obj in this list.
// return true if this LinkedList changed as a result of this
// method call, false otherwise.
public boolean removeLastOccurrence(Object obj){
     assert obj != null;
```

4. (Implementing data structures, 15 points) A hash table is a data structure that uses an array as its internal storage container. Items are added to the array based on the integer generated by a hash function. A hash function produces an integer based on some properties of the object. In Java hash functions are encapsulated via the `hashCode` method in the `Object` class and that most classes override.

To add an `Object` to a hash table

1. call its `hashCode` method to obtain an int, x
2. modulus x by the length of the hash table to obtain an index, i
3. go to index i in the hash tables native array
4. if that element is null or equal to the `NOTHING` object, place the `Object` in that element
5. if that element is not null a collision has occurred. Collisions can be resolved in many ways.
6. In this hash table collisions are resolved via *quadratic probing*. Check the element at index i + 1
7. If the element at i + 1 is null or equal to the `NOTHING` object, add the item at that element, if not continue to check elements until an open spot is found. The distance from the original spot goes up quadratically: 1, 2, 4, 8, 16, 32, 64, and so forth. If the end of the array is reached then wraparound.
8. A special case occurs if the hash table reaches a certain *load factor*. Load factor is equal to (number of items in hash table / length of array used as internal storage container). After adding an element, if the load factor is greater than or equal to some limit the table must be rehashed. This involves resizing the internal storage container and rehashing all elements.

Consider the following HashTable class

```
public class HashTable
{     // the NOTHING object. Values that are removed from the
      // hash table are replace with a reference to this
      // object instead of null
      private static final Object NOTHING = new Object();

      // storage container for items in this HashTable
      private Object[] con;

      // number of items in this HashTable
      private int size

      // load factor limit. When the load factor is greater than
      // or equal to this value call the rehash method
      private double loadFactorLimit;

      //called to rehash table if load factor is exceeded
      private void rehash()

      // complete the following method. Write your answer on the
      // next page
      /*    pre: item != null
            post: add this item to the hashTable. size() = old size() + 1
      */

      public void add(Object item)
}
```

You will have to use the hashcode method from the Object class:

```
public int hashCode()
      Returns a hash code value for the object.
```

```
// complete the following method from the HashTable class
/*   pre: item != null
     post:add this item to this HashTable. size() = old size() + 1
*/

public void add(Object item){
     assert Object != null;
```

5. (Implementing data structures 15 points) Using the same `HashTable` class from question 4, implement an `Iterator` class for the `HashTable` class. The `Iterator` class will be an inner class so it will have access to all of the private instance variables of the `HashTable` object that created it.

   Complete all three methods from the `Iterator` interface, `hasNext`, `next`, and `remove`. All methods should be O(1) time and space. Do not use any of the methods from the `HashTable` class. Recall the `remove` method can be called only once per call to `next`. Throw an exception if this condition is violated.

   When an object is removed from the `HashTable` it is replaced with the `NOTHING` object instead of `null`.

   Complete the class below.

```
// nested inside the HashTable class
private class HashIterator implements Iterator{
```

```
// More room on next page if needed.
```

```
// More room for HashIterator class if needed.
```

6. (Working with data structures, 10 points.) Write a method to determine how many elements are in a `Queue`. This method is not part of the `Queue` class. The only methods in the `Queue` class are the traditional `enqueue, dequeue, front, and isEmpty.` When the method is complete the Queue must be restored to the same state as before the method was called.

You may use whatever other data structures and classes you want including other stacks and queues.

```
// pre: q != null
// post: return the number of elements in q. q is unchanged by
// this method.
public int getSize(Queue q){
     assert q != 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
• boolean hasNext()
• Object next()
• void remove()