CS 307 – Final Exam – Fall 2001

Name____________________________________
Last 4 digits of SSN / Student ID ____________
Class Unique ID ___________________________

Instructions:
1. There are 4 questions on this test. Question 1 is worth 60 points, all others are worth 20 points.
2. You will have 3 hours to complete the test.
3. You may not use a calculator.
4. When code is required, write Java code.
5. You may not use any classes or methods from the Java Standard Library other than the ones specifically mentioned on each question.
6. The style guide is not in effect.
7. Please make your answers legible.

1. Java mechanics and short answer questions. (2 points each) Write the answer to each question in the space provided. If code results in an error indicate if it is a compile error or runtime error.

A. The following numbers are inserted in the order shown into a binary search tree with no checks to ensure or maintain balance. The tree is initially empty. Draw the resulting tree.

15 86 90 12 45 2 8
For parts B, C, and D consider the following binary tree. For each question assume when a node is processed the value in the node is printed out by the statement:

```java
System.out.print( currentNode.data + " ");
```

B. What is the output of a preorder traversal of the tree?

C. What is the output of an inorder traversal of the tree?

D. What is the output of a postorder traversal of the tree?
E. Assume we are implementing a hash table with an array of linked lists as the storage container. Assume chaining is used to resolve collisions. Objects that hash to the same value are simply placed at the end of the linked list at the index the key hashes to. The table is rehashed whenever any chain reaches a length of 10. What is the average case and worst case Big O for inserting an element into the table? (Assume keys are unique, and the hash function does a good job of distributing the keys over the indexes of the array used as storage.)

Average _______________________  Worst _______________________

F. If the same hash table described in part E is used what is the Big O of accessing an element in the hash table?

____________________

G. Assume a hash table uses strings consisting of only lower case letters as the keys for inserting items into the table. The hash function assigns an integer to each character in the string, 1 for 'a', 2 for 'b' and so forth and then adds these values together. What would the hash value of "dog" be?

____________________

H. This question uses the same hash function as part G. Assume the hash table currently uses an array of size 100. What must be done to the hash values generated via the hash function from part G to be able to insert objects into the array?

________________________________________________________

I. Assume social security numbers are used as the keys for inserting Person Objects into a hash table. The hash function adds the last 4 digits of the social security number to obtain the hash value. The array to hold objects has a length of 10. The table is not resized during the following insertions. Linear probing is used to resolve collisions. Show the hash table after inserting the following Person objects in the order shown.

Mike:  723-45-5631
Kelly  565-27-0001
Olivia: 627-98-0125
Belle: 527-41-4100

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
J. Consider a Stack class designed only to hold integers. It has the following methods:

```java
public Stack() //create an empty stack
public void push(int x)
public void pop() //remove top element
public int top() //access top element without removing it
public void makeEmpty()
public boolean isEmpty()
public int size()
```

What is the output of the following code?

```java
Stack s = new Stack();
for(int i = 2; i < 20; i+= 2)
    s.push(i);
for(int j = 0; j < s.size(); j++)
{  System.out.print( s.top() + " " );
    s.pop();
}
```

K. Consider a Queue class designed only to hold integers. It has the following methods:

```java
public Queue() //create an empty queue
public void enqueue(int x)
public void dequeue() //remove front element
public int front() //access front element without removing it
public void makeEmpty()
public boolean isEmpty()
public int size()
```

What is the output of the following code?

```java
Queue q = new Queue();
q.enqueue(10);
q.enqueue(12);
q.enqueue(7);
q.dequeue();
q.enqueue( q.front() );
q.enqueue( q.front() );
q.dequeue();
q.enqueue( q.front() * 2 );
q.enqueue( 13 );
q.enqueue( q.size() );
q.dequeue();
q.dequeue();
System.out.println( q.size() + " " + q.front() );
```
L. For a binary search tree with no special algorithms to ensure balance is maintained what is the average case Big O to insert N item into an initially empty tree?

M. For a binary search tree with no special algorithms to ensure balance is maintained what is the worst case Big O to insert N items into an initially empty tree?

N. Is it possible to implement a queue using a SINGLY linked list and ensure that each of its operations (enqueue, dequeue, front, isEmpty, getSize, and makeEmpty) is O(1), i.e. constant time?

O. What is the Big O of this code? method1 always has a Big O of O(1).

```java
for (int i = 0; i < N; i++)
    for (int j = 0; j < N; j++)
        for(int k = 1; k < N; k *=2)
            method1();
```

P. What is the Big O of this code? method2 always has a Big O of O(N) where N is the equal to the argument sent to the method.

```java
for (int i = 0; i < N; i++)
    method2(i);
for (int j = 0; j < 3 * N / 4; j++)
    method2(j);
```
Q. A strictly binary tree is a binary tree in which every node has either 0 or 2 children. No node has exactly 1 child. How many nodes are in a strictly binary tree that has 8 leaves? If there is more than 1 answer list all answers.

_______________________________________

R. Consider the following ListNode class for Linked Lists:

```java
public class ListNode
{
    public Object data;
    public ListNode next;
}
```

Draw the data structure that exists after the following code is executed. Use a "/" to indicate any object references that are null.

```java
ListNode N1 = new ListNode();
ListNode N2 = new ListNode();
N1.next = N2;
N1.next.next = N2;
```

S. Consider the following method:

```java
public int Sword( int n )
{
    if ( n <= 1 )
        return 0;
    else
        return n + Sword( n - 1 ) + Sword( n - 2 );
}
```

What is the output of the statement `System.out.println( Sword(4) );`? ______________
T. What is the Big O of method Sword in part S?

__________________________________________

U. Consider using the quicksort algorithm to sort an array of integers into ascending order. Recall that quicksort first partitions the array about an element (the pivot) and then recursively sorts the two partitions. (or parts) One way of choosing the partition is to always use the left-most (first) element in the subarray being partitioned.

What is the Big O of quicksort on the following lists?

A list whose values are in descending order: _________________

A list whose values are in random order: _________________

V. In this question you are asked to consider using either a binary search tree or a hash table using chaining to resolve collisions as the storage container for data. Which of the following reasons best support choosing the binary search tree over the hash table:

I. The Big O average time required to insert an item into a binary search tree is less that the Big O average time required to insert an item into a hash table.

II. The Big O average time required to determine whether a given value is in a binary tree is less than the Big O average time required to determine whether a given value is in a hash table.

III. The Big O average time required to print all items stored in a binary search tree in sorted order is less than the Big O average time required to print all items stored in a hash table in sorted order.

List all reasons that apply:

__________________________________________
W. Assume the following declaration for a binary tree node

```java
public class BTNode {
    public Object data;
    public BTNode left;
    public BTNode right;
}
```

Now consider the following method:

```java
public int Shield(BTNode n) {
    if (n == null)
        return 0;
    else if (n.right != null && n.left != null)
        return 1 + Shield(n.right) + Shield(n.left);
    else
        return Shield(n.right) + Shield(n.left);
}
```

Summarize in one sentence what method Shield accomplishes:

_____________________________________________________________________________

X. In the word ladder assignment an alternative solution would be to determine before hand all the words one different from every other word. Then for each requested ladder a search would be done using this pregenerated list of words. This would save computation time if the program is asked to do lots of ladders because the program isn't searching for all the words one different from a given word more than once. One way of generating the list of words is to take each word and then compare it to every other word in the list. What is the Big O of generating the list of words one letter different from each word using this method?

_____________________________________________________________________________

Y. Another option for generating the list of words one letter different from each word is to take each word in the list generate every word that is one letter different from that word and then search the word list to see if that word is present. For example if the word is "angel" we would start with "bngel" and see if it is in the list, then "cngel" and see if that is in the list and so forth. Assume the list of words is in sorted order. What is the Big O of this method?

_____________________________________________________________________________
Z. For this question you are to consider the two algorithms from parts X and Y to generate the list of words one letter different from each word. For what number of words is the method in part Y faster than the method in part X? Assume checking to see if a word is the same as another word takes 1 operation. For example to see if "angel" is one letter different from "cakes" is considered a single operation. Also assume generating a word one letter different from a word is 1 operation. For example given "angel", generating "bngel" is one operation, generating "cngel" is one operation, and so forth. You may ignore the time it takes to add each word that is one letter different to the list of off by one words because this will be the same for both algorithms. Finally assume the list of words is already in sorted order. Again, for what size word lists is the method in part Y faster than the method in part X? Your answer can either be an integer or a formula in terms of N, words in the list. Note this question concerns actual execution time not Big O time.

AA. Consider the following ListNode class for a Linked List of integers:

```java
public class ListNode {
    public int data;
    public ListNode next;
}
```

Now consider the following method

```java
public void Ring(ListNode n)
{
    if( n != null )
    {
        Ring( n.next );
        if( ( n.data % 2 ) == 0 )
        {
            System.out.println( n.data );
        }
    }
}
```

Summarize in one sentence what method Ring accomplishes:
AB. A **heap** is a data structure with the following properties: it is a complete binary tree in which every node possesses the property that its value is less than or equal to the value in both of its child nodes. Recall a complete binary tree is one where every level except the last one has the maximum number of nodes and the nodes in the bottom level are placed from left to right. Here is an example:

![Heap Diagram]

When inserting an item the goal is to make the minimum number of changes or swaps of values in the nodes. Assume the number 9 is the next number to be inserted. Draw the heap after this insertion is complete. Remember the minimum number of changes should be made and the new tree must have the heap property, where a node's value is less than the value of both of its children, and the insertion must result in a complete tree. (Nodes added from left to right in the lowest level.)
AC. Assume the following declaration for a binary tree node

```java
public class BTNode {
    public int data;
    public BTNode left;
    public BTNode right;
}
```

Now consider the following method:

```java
public int Moria( BTNode n) {
    if ( n == null )
        return 0;
    else if( n.right == null && n.left == null)
        return n.data;
    else
        return Moria( n.right ) + Moria( n.left );
}
```

Summarize in one sentence what method Moria accomplishes:

_____________________________________________________________________________

AD. What is the value of the following postfix expression?

5 4 3 + * 2 4 * +

_________________________________
2. Write a method that recursively determines the number of different ways to roll a particular number on a given number of dice:

```java
/**
* pre: numDice > 0
* post: return the number of different ways to roll numToRoll with numDice dice. Assume each die has sides numbered from 1 to 6
*/
public int waysToRoll(int numToRoll, int numDice)
```

All dice have six sides. For example if the methods was called as follows:

```java
System.out.println( waysToRoll(4,2) );
```

The output would be 3.

<table>
<thead>
<tr>
<th>Die1 Value</th>
<th>Die2 Value</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

Thus with 2 dice there are 3 combinations that add up to 4. Note even though the first 2 both have a 3 and a 1 they are considered different combinations of rolls.

Another example:

```java
System.out.println( waysToRoll(5,3) );
```

The output would be 6:

<table>
<thead>
<tr>
<th>Die1 Value</th>
<th>Die 2 Value</th>
<th>Die 3 Value</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>5</td>
</tr>
</tbody>
</table>

Your method must be recursive. You cannot create a helper method. You must list your base case(s) and recursive step before the method implementation.
Describe your base case(s) in English: (1 point)

Describe your recursive step in English: (2 points)

/*
pre: numDice > 0
post: return the number of different ways to roll numToRoll with numDice dice. Assume each die has sides numbered from 1 to 6 */
public int waysToRoll(int numToRoll, int numDice)
{ // implement your method below
3A. (10 points) Write a method to perform a level order traversal of a binary tree.

Use the following class for Binary Tree Nodes:

```java
class BTNode {
    public Object data;
    public BTNode left;
    public BTNode right;
}
```

A level order traversal of the above tree would produce the following output:

7  11  13  39  12  2001  571

For this question you may use a Queue class with the following methods:

```java
public Queue() // create an empty Queue
public void enqueue(Object val) // put val at the end of queue
public void dequeue() // discard the front element
public Object front() // get the front element. (still front
                      // element after this method is called
public void makeEmpty()
public boolean isEmpty()
public int size() // number of elements in queue
```

Write out your algorithm for this task in English below: (2 points)
/*
   pre: node != null
   post: prints out the data stored in the tree for which node
   is the root via a level order traversal to standard output
   (via System.out)
*/

public void levelOrderTraversal(BTNode node)
{
   // complete the method below
3B. (10 points) Write a method to determine the depth of a binary tree. Depth is the number of edges (links) from the root of the tree to the deepest or lowest leaf of the tree. The tree shown on page 2 has a depth of 3. The tree shown on page 14 has a depth of 2. The same BTNode class from part A is used.

    /*
     * pre: none
     * post: return the depth of the tree
     */
    public int getDepth( BTNode node )
    {   // implement your method below
4. Implement the insert method for a HashTable class. The class uses an array of LinkedLists as its storage container. Collisions are resolved via chaining, with Objects placed at the end of the LinkedList for the index the object's key hashes to. If an insertion results in a LinkedList reaching a size greater than MAX_CHAIN_SIZE then the hash table is increased in size to the next prime number after

\[
\text{MAX\_CHAIN\_SIZE} \times 2 \times \text{myContainer.length}
\]

(myContainer is the name of the array of LinkedLists instance variable).

The HashTable class is designed to hold objects of type Hashable. The Hashable class contains two Objects. One is the key that is used to create the hash value and the other is the actual data to be stored. Here is the public interface of the Hashable class

```java
public class Hashable {
    public Hashable(Object key, Object data)
    public Object getKey()
    public Object getData()
}
```

The following parts of the HashTable class are available for your use

```java
public class HashTable {
    private static final int MAX_CHAIN_SIZE = 5;
    private LinkedList[] myContainer; // container for objects
    private int iMySize; // number of objects currently in table

    /* pre: none
     * post: returns number of objects currently in HashTable
     */
    public int getSize() {
    }

    /* pre: none
     * post: return the first prime number >= val
     */
    protected int nextPrime( int val ) {
    }

    /* pre: val != null
     * post: return an int which is the hash value for val. The
     * int returned is >= 0. this method always returns the
     * same value for a given key regardless of the size of
     * myContainer.
     */
    protected int getHashValue( Object key ) {
    }
}
The following is the LinkedList class used

```java
public class LinkedList
{
    // only the public portions of this class are shown

    /*
     * pre: none
     * post: getSize() = 0
     */
    public LinkedList()

    /*
     * pre: val != null
     * post: getSize() = 1,
     *       getFirst() and getLast() both return val
     */
    public LinkedList(Object val)

    /*
     * pre: val != null
     * post: getSize() = old getSize() + 1,
     *       getFirst() returns val
     */
    public void addFirst(Object val)

    /*
     * pre: val != null
     * post: getSize() = old getSize() + 1, getFirst returns val
     */
    public void addLast(Object val)

    /*
     * pre: none
     * post: returns number of items in list
     */
    public int getSize()

    /*
     * pre: none
     * post: returns an Iterator Object ready to return the
     *       first element in the list
     */
    public Iterator iterator()
}
```
The following methods are available for the Iterator object returned by the LinkedList class iterator method

```java
public interface Iterator
{
    /* pre: none
    post: returns true if there is one or more items left to be retrieved
    */
    public boolean hasNext();

    /* pre: hasNext = true
    post: returns a reference to the current Object and advances the iterator to the next object
    */
    public Object next();
}
```

complete the insert method for the HashTable class. Use the next page is more space is needed

```java
/* pre: val != null
    post: val inserted into HashTable, getSize = old getSize() + 1
    */
public void insert(Hashable val)
{
    /* note to implementer. If the insertion of val to the hash table results in any LinkedList's size being greater than MAX_CHAIN_SIZE the storage container is to be resized to the next prime number after MAX_CHAIN_SIZE * 2 * myContainer.length.

    All items in the table are to be rehashed and inserted into their proper positions. You may assume this rehashing will not result in any of the new LinkedList's size being greater than MAX_CHAIN_SIZE */
```
// more space for insert method for HashTable class