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

CS 314 – Exam 2 – Fall 2016

Your Name____________________________________

Your UTEID __________________________________

Instructions:

1. There are **6** questions on this test. 100 points available. Scores will be scaled to 200 points.
2. You have 2 hours to complete the test.
3. Place you final answers on this test. Not on scratch paper. Answer in pencil.
4. You may not use a calculator or any other electronic devices while taking the test.
5. When answering coding questions, ensure you follow the restrictions of the question.
6. Do not write code to check the preconditions.
7. On coding questions you may implement your own helper methods.
8. On coding questions make your solutions as efficient as possible given the restrictions of the question.
9. Test proctors will not answer any questions regarding the content of the exam. If you think a question is ambiguous or has an error, state your assumptions and answer based on those assumptions.
10. When you complete the test show the proctor your UTID, give them the test and all the scratch paper, used or not, and leave the room quietly.

1. (1 point each, 20 points total) Short answer. Place your answer on the line next to or under the question. Assume all necessary imports have been made.

   a. If a question contains a syntax error or other compile error, answer **compile error**.
   b. If a question would result in a runtime error or exception, answer **runtime error**.
   c. If a question results in an infinite loop, answer **infinite loop**.
   d. Recall when asked for Big O your answer should be the most restrictive correct Big O function. For example Selection Sort has an average 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) or O(N^4). I want the most restrictive, correct Big O function. (Closest without going under.)

A. What is output by the following code?

____

```
String sta = a("DC");
System.out.println(sta.length());

public static String a(String s) {
     if (s.length() > 30)
          return s + "DONE";
     else if (s.length() % 2 == 0)
          return a(s + "AGAIN" + s);
     else
          return a(s + "ONE");
}
```

B. What is output by the method call `b(2, 6)`? ______________________________

```
public static void b(int x, int y) {
     if (x > 4)
          System.out.print("*");
     else if (x == y)
          System.out.print("=" + y + "=");
     else {
          System.out.print(y + " ");
          b(x + 1, y - 1);
          System.out.print(" " + x);
     }
}
```

C. What is returned by the method call `c(20)`? ______________________

```
public static int c(int n) {
     if (n <= 0)
          return 2;
     else {
          int t1 = c(n - 10);
          int t2 = c(n - 5);
          return 1 + t1 + t2;
     }
}
```

D. What is returned output by the method call `d1(5)`? ______________________

```
public static void d1(int x) {
     System.out.print("-");
     if (x <= 1)
          System.out.print(x);
     else {
          d2(x);
     }
}

public static void d2(int x) {
     System.out.print(x);
     if (x % 2 == 0)
          d1(x / 2);
     else
          d2(3 * x + 1);
}
```

E. What is the worst case order (Big O) of `e`? The method uses the Java LinkedList class. N = `list.size()`

____

```
public static LinkedList<Integer> e(LinkedList<Integer> list, int t) {
     LinkedList<Integer> result = new LinkedList<Integer>();
     int i = 0;
     while (i < list.size()) {
          int x = list.get(i);
          if (x < t) {
               list.remove(i);
               result.add(0, x); // insert at position 0
          } else {
               i++;
          }
     }
     return result;
}
```

F. What is the worst case order (Big O) of `f`? N = `list.size()` _________________

```
public static ArrayList<String> f(ArrayList<String> list) {
     ArrayList<String> result = new ArrayList<String>();
     Iterator<String> it = list.iterator();
     while (it.hasNext()) {
          String temp = it.next();
          if (temp.length() > 10) {
               it.remove();
               result.add(temp);
          }
     }
     return result;
}
```

G. What is output by the following code? _________________

```
int[] data = {12, 15, 3, 10, 10, 3, 6};
ArrayList<Integer> list = new ArrayList<Integer>();
for (int x : data) {
     list.add(x);
}
Iterator<Integer> it = list.iterator();
while (it.hasNext() ) {
     if (it.next() % 3 == 0) {
          System.out.println(it.next());
     }
}
```

H. What is the result of the following postfix expression? (single integer for answer) _____________

```
20 3 2 + 2 * / 4 +
```

I. A method uses the insertion sort algorithm. Given an array of 100,000 ints in random order it takes the method 10 minutes to complete. What is the expected runtime when given an array of 200,000 elements in random order?

____

J. The following method takes 20 seconds to complete when `data.length` is 1,000,000 and `targets.length` is 1,000,000. What is the expected time for the method to complete when `data.length` is 2,000,000 and `targets.length` is 2,000,000

```
public static int j(int[] data, int[] targets) {
     Arrays.sort(data); // quicksort
     int result = 0;
     for (int i = 0; i < targets.length; i++) {
          if (Arrays.binarySearch(data, targets[i]) >= 0) {
               result++;
          }
     }
     return result;
}
```

____

K. What is output by the following code? The code uses the Java `Stack` class.

```
Stack<Integer> st = new Stack<Integer>();
int sub = 0;
for(int i = 0; i < 30; i = 1 + i + i) {
     st.push(i);
}

for(int i = 0; i < st.size(); i++) {
     System.out.print(st.pop() + " ");
}
```

L. The method `odd` takes .2 seconds to complete when `data.length` is 1,000. What is the expected time to complete when `data.length` is 2,000? All elements in `data` are distinct, no repeats.

____

```
public static Thing odd(int[] data) {
     Thing result = new Thing();
     for (int i = 0; i < data.length; i++) {
          result.add(data[i]);
     }
     return result;
}

public class Thing {
     private ArrayList<Integer> con;

     public Thing() { con = new ArrayList<Integer>(); }

     public void add(int obj) {
          con.add(obj);
          Collections.sort(con); // mergesort
     }
}
```

M. Explain two times a class must be declared `abstract`. Be specific and concise.

1.____

2.____

N. We are going to implement a Stack and use a singly linked list as our internal storage container. The linked list has references to the first and last nodes in the structure. Which end of the list should be the top of the stack for efficient stack operations? beginning of list, end of list, or either end

____

O. The binary search tree class in the code below uses the simple, naïve add algorithm demonstrated in class. It takes .005 seconds for the method to complete when n is 1,000,000. What is the expected time for the method to complete when n is 2,000,000?

____

```
public static BST<Integer> makeTree(int n) {
     BST<Integer> result = new BST<Integer>();
     int[] data = new int[n];
     for(int i = 0; i < data.length; i++) {
          result.add(data[i]);
     }
     return result;
}
```

P. What is the height of a complete binary tree that contains 10 nodes? _________________

Consider the following binary tree. `5` is the root of the tree.

```
      5
    /   \
   0     9
    \     \
     7     2
    / \
   4   8
```

Q. What is the result of a pre-order traversal of the binary tree shown above?

____

R. What is the result of a in-order traversal of the binary tree shown above?

____

S. What is the result of a post-order traversal of the binary tree shown above?

____

T. The following values are inserted one at a time into a binary search tree using the simple, naïve algorithm demonstrated in class. Draw the result tree.

```
24   0    10   15   30   10   19   13   29
```

2. Linked Lists I (16 points) - Complete the `replaceSmaller` instance method for the `LinkedList314` class. The method replaces values in the calling linked list with values from another linked list sent as a parameter. An element in the calling linked list is replaced if the element from the parameter, at the same position, is larger than it based on the `compareTo` method.

- **You may not use any other methods in the `LinkedList314` class unless you implement them yourself as a part of your solution.**
- The `LinkedList314` class uses singly linked nodes.
- The list has references to the first node in the linked structure of nodes.
- When the list is empty, `first` is set to `null`.
- None of the elements of either linked list equals `null`.
- If the list is not empty the last node in the list has its next reference set to `null`.
- These linked lists store elements that are `Comparable`.
- You may use the nested `Node` class and the `Comparable` `compareTo` method.
- **You may not use any other Java classes or native arrays.**

```
public class LinkedList314<E extends Comparable<? super E>> {

     // refers to first node in the chain of nodes.
     private Node<E> first;
     // No other instance variables

     // The nested Node class.
     private static class Node<E extends Comparable<? super E>> {
          private E data;
          private Node<E> next;
     }
}
```

Examples of calls to `replaceSmaller(LinkedList314<E> otherList)`. In this example the lists contain `Integer` objects. Elements that will be replaced are bolded as are the replacement values.

```
calling: [5, 3, 12] -> calling list becomes [5, 7, 12]
other:   [4, 7,  5]

calling: [] -> calling list still [], no changes
other:   [4, 7, 5]

calling: [5, 3, 12, 20] -> calling list becomes [5, 7, 12, 20]
other:   [4, 7,  5]


calling: [5, 3, 12] -> calling list becomes [5, 7, 12]
other:   [4, 7,  5, 20]

calling: [0, 3,  0, -5, 15] -> calling list becomes [4, 7, 0, -3, 15]
other:   [4, 7, -4, -3, 12]
```

Complete the following instance method of the `LinkedList314` class.

```
/* pre: otherList != null
   post: per the question description and otherList is not altered*/
public void replaceSmaller(LinkedList314<E> otherList) {
```

3. Linked Lists II (16 points) - Complete the `removeNum(int start, int number)` instance method for the `LinkedList314` class. The method removes `number` elements from the linked list starting at position `start`. Note, it may not be possible to remove the given number of elements if there are not sufficient elements in list from the given starting position. The precondition requires start to be inbounds, but only requires `number` to be >= 0. The method returns the actual number of items removed. See the example below.

- **You may not use any other methods in the `LinkedList314` class unless you implement them yourself as a part of your solution.**
- The `LinkedList314` class uses singly linked nodes.
- The list has references to the first node in the linked structure of nodes.
- When the list is empty, `first` is set to `null`.
- If the list is not empty the last node in the list has its next reference set to `null`.
- You may use the nested `Node` class.
- **You may not use any other Java classes or native arrays.**

```
public class LinkedList314<E> {

    // refers to first node in the chain of nodes.
    private Node<E> first;
    // No other instance variables

    // The nested Node class.
    private static class Node<E> {
        private E data;
        private Node<E> next;
    }
}
```

Examples of calls to `int removeNum(int start, int number)`. In this example the list contains `Integer` objects.

```
[4].removeNum(0, 1) ->  returns 1, list becomes []

[4].removeNum(0, 0) ->  returns 0, list remains [4]

[4, 3, 0, 5, 7].removeNum(0, 2) ->  returns 2, list becomes [0, 5, 7]

[4, 3, 0, 5, 7].removeNum(3, 2) ->  returns 2, list becomes [4, 3, 0]

[4, 3, 0, 5, 7].removeNum(3, 5) ->  returns 2, list becomes [4, 3, 0]

[4, 3, 0, 5, 7].removeNum(1, 6) ->  returns 4, list becomes [4]

[4, 3, 0, 5, 7].removeNum(0, 5) ->  returns 5, list becomes []

[4, 3, 0, 5, 7].removeNum(0, 10) ->  returns 5, list becomes []
```

Complete the following instance method of the `LinkedList314` class.

```
/* pre: start is a valid position in this list, number >= 0
   post: per the problem description */
public int removeNum(int start, int number) {
```

4. Stacks (16 points) - Write a method that accepts a stack of integers as a parameter and removes elements from the stack as necessary so that the remaining elements are in descending order.

Consider the following examples:

`top [12, **15**, 10, 10, **20**, 5, 0] bottom`
becomes
`top [12, 10, 10, 5, 0] bottom`

`top [12, 10] bottom`
remains
`top [12, 10] bottom`

`top [12, **15, 20, 15, 20, 15, 20**] bottom`
becomes
`top [12] bottom`

`top [12, 12] bottom`
remains
`top [12, 12] bottom`

`top [] bottom (empty stack)`
remains
`top [] bottom`

You may use a single stack or queue as an auxiliary data structure. Assume the class names are `Stack314` and `Queue314`. Each of these classes implement the four methods shown in class.

```
Stack314: isEmpty, push, pop, top
Queue314: isEmpty, enqueue, dequeue, front
```

Do not use any other Java classes or methods.

Complete the method on the next page.

```
/* pre: st != null
   post: per the problem description */
public void makeDescending(Stack314<Integer> st) {
```

5. Trees (16 points) - Implement an instance method for a binary tree class that determines the number of nodes that contain a specified value **and** have a **single left child**.

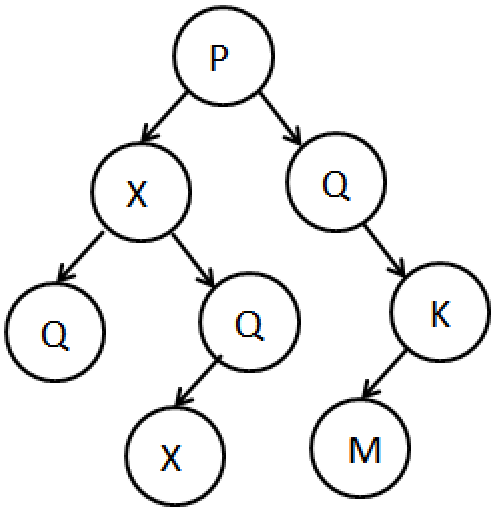
```
public class BinaryTree<E> {

     private BNode<E> root; // root == null if tree is empty

     // nested BNode class
     private static class BNode<E> {

          private E data;
          // left and right child are null if no child
          private BNode<E> left;
          private BNode<E> right;
     }
}
```

Consider the following tree, `t`. The root is the node that contains P. This is not a binary search tree. None of the data in the tree equals `null`.



```
t.numNodesWithValueAndLeftChildOnly(Q) returns 1

t.numNodesWithValueAndLeftChildOnly(K) returns 1

t.numNodesWithValueAndLeftChildOnly(P) returns 0

t.numNodesWithValueAndLeftChildOnly(X) returns 0
```

Do not create any new nodes or other data structures. You may use the `BNode` class and the `equals` method from the `Object` class, but do not use any other methods or classes. You may implement a helper method if you wish.

```
/*   pre: value != null.
     post: per the problem description */
public int numNodesWithValueAndLeftChildOnly (E value) {
```

6. Recursive Backtracking (16 points) From the Professor Layton series of games. Write a method to determine if it possible to solve a marble jumping puzzle or not. The puzzle uses a board such as this:

The board is represented as rectangular 2d array of chars. Open spaces will indicated with an `o`. Marbles will be indicated with an `m`. Cells that cannot be moved into will be represented with an `x`. So, for example the above board will be represented like this:

0	1	2	3	4	5	6
x	x	o	o	o	x	x
x	x	o	o	o	x	x
o	o	***o***	o	o	o	o
o	o	**m**	o	m	o	o
o	o	**m**	m	m	o	o
x	x	m	m	m	x	x
x	x	m	m	m	x	x

Note this is simply one example. The board size, arrangement, and initial marble position will vary.

In the marble jumping puzzle, marbles can jump over adjacent marbles that are located up, down, left, or right of the marble as long as there is an open spot on the other side of the marble being jumped over. When a marble is jumped it is removed from the board. So in the example above one of the legal moves would be to take bolded marble and move it over the marble above it leading to this configuration where the marble that was jumped over has been removed.

x	x	o	o	o	x	x
x	x	o	o	o	x	x
o	o	***m***	o	o	o	o
o	o	**o**	o	m	o	o
o	o	**o**	m	m	o	o
x	x	m	m	m	x	x
x	x	m	m	m	x	x

So a move consists of three changes:

1. Picking up the source marble from its spot.
2. Moving the source marble to its destination.
3. Removing the marble the source marble passed over from the board.

Classes to represent the board and moves have already been created.

Here is the `Move` class:

```
public class Move {

      public int sourceRow() // returns row of marble to move
      public int sourceCol() // returns column of marble to move

      public int destRow() // returns row to move marble to
      public int destCol() // returns column to move marble to

      // returns row of marble removed if this move is made
      public int removedRow()
      // returns column of marble removed if this move is made
      public int removedCol()
}
```

And here is the `Board` class:

```
public class Board {

      public int numMarblesOnBoard() // returns number of marbles on board

      // Returns the current legal moves on this Board.
      public Move[] getMoves()

      // Remove a marble at the given location if one is present.
      public void removeMarble(int row, int col)

      // place a marble at the given location.
      public void placeMarble(int row, int col)

}
```

Complete the following method:

```
/*    pre: board != null
      post: return true if the puzzle represented by board can be solved.
      In other words, a series of legal moves can be made so that
      only one marble remains.  */
public boolean canBeSolved(Board board){
```

Complete this method on the next page.

```
/*   pre: board != null
     post: return true if the puzzle represented by board can be solved.
     In other words, a series of legal moves can be made so that
     only one marble remains.  */
public boolean canBeSolved(Board board){
```