Points off	1	2	3	4	5		Total off

CS 314 – Exam 2 – Spring 2017

Your Name_____

Your UTEID _____

Instructions:

1. There are **5** 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 returned by the method call `a(13)`? _____

```
public static int a(int x) {
     if (x == 0) {
          return 0;
     } else {
          return x + a(x / 2);
     }
}
```

B. What is returned by the method call `a(-7)`? _____

C Given a positive `int` N, what is the order of method `a`? _____

D. What is returned by the method call `d(9)`? _____

```
public static int d(int x) {
     if (x <= 4) {
          return x;
     } else {
          int total = 1;
          total += d(x - 2);
          total += d(x - 4);
          return total;
     }
}
```

E. What is the worst case order (Big O) of method `e` if the parameter `list` refers to a Java `LinkedList`? N = `list.size()`

_____

```
public static void e(List<Integer> list, int t) {
     Iterator<Integer> it = list.iterator();
     for (int i = 0; i < list.size() / 2; i++) {
          it.next();
     }
     while (it.hasNext()) {
          if (it.next() == t) {
               it.remove();
          }
     }
}
```

F. What is the worst case order (Big O) of method `e` if the parameter `list` refers to a Java `ArrayList`? N = `list.size()`

_____

G. What is the worst case order (Big O) of method `g`?
N = `list1.size(), list1.size() == list2.size()`

_____

```
// pre: no elements of lists are null
public static <E> int g(LinkedList<E> list1, LinkedList<E> list2) {
     int r = 0;
     for (int i = 0; i < list1.size(); i++) {
          E x = list1.get(i);
          for (int j = i + 1; j < list2.size(); j++)
               if (x.equals(list2.get(j)))
                    r++;
     }
     return r;
}
```

H. A sorting methods uses the traditional mergesort algorithm presented in class. It takes the method 40 seconds to sort an array of 1,000,000 `int`s that are already in ascending order. What is the expected time for the method to sort an array of 2,000,000 `int`s that are already in ascending order?

_____

I. A sorting methods uses the traditional quicksort algorithm presented in class. It takes the method 10 seconds to sort an array of 10,000 `int`s, all equal to the same value. (For example, an array with 10,000 `0`s in it.). What is the expected time for the method to sort an an array of 30,000 `int`s, all equal to the same value.

_____

J. A sorting methods uses the traditional insertion sort algorithm presented in class. It takes the method .002 seconds to sort an array of 1,000,000 `int`s that are already in ascending order. What is the expected time for the method to sort an array of 2,000,000 `int`s that are already in ascending order?

_____

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

_____

```
Stack<Integer> st = new Stack<Integer>();
for(int i = 1; i < 15; i *= 3) {
     st.push(i + 1);
     st.push(i);
}

while (!st.isEmpty()) {
     System.out.print(st.pop() + " ");
}
```

L. Given the following postfix expression what is the equivalent infix expression?
(x, y, and z are vaiables)

```
x  y  z  -  /
```

_____

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

```
10  3  8  12  -  *  +
```

N. The binary search tree class in the code below uses the simple, naïve add algorithm demonstrated in class. It takes .1 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? N = `n`. Assume the `Random.nextInt()` method is O(1).

_____

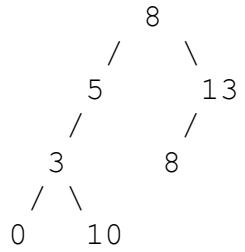
```
public static BST<Integer> makeTree(int n) {
      BST<Integer> result = new BST<Integer>();
      Random r = new Random();
      for (int i = 0; i < n; i++) {
            result.add(r.nextInt());
      }
      return result;
}
```

O. Why is it problematic to use a Java `ArrayList` as the internal storage container for a `Queue` class?

_____

P. What is the minimum height of a full binary tree with 13 nodes? _____

Consider the following binary tree. `8` is the root of the tree.



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

_____

R. What is the result of a pre-order traversal of the binary tree shown on the previous page?

_____

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

_____

T. The binary tree on the previous page is not a binary search tree. What, specifically in the tree shown, violates the requirements for a binary search tree?

_____

EXTRA CREDIT (1 point): How many current and former UT Austin Computer Science faculty members have won the Turing Award?

_____

2. Linked Lists I (20 points) - Complete the `getNumDifferences` instance method for the `LinkedList314` class. The method determines the number of elements that are different between two lists. The method compares elements at the same position. If the lists are not the same size, then **each** extra element in the larger list counts as one difference. **Do not use recursion.**

- **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 only has a reference to the first node in the chain of nodes.
- When the list is empty, `first` is set to `null`.
- **There may be node's whose data variables is set to `null`.**
- If the list is not empty, the last node in the chain of nodes, has its next reference set to `null`.
- You may use the nested `Node` class and the `equals` method for `Objects`.
- **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 `getNumDifferences (LinkedList314<E> other)`. In these examples the lists contain `Integer` objects.

```
calling: [5, 7, 12] -> returns 0, no differences
other:   [5, 7, 12]

calling: [] -> returns 0, no differences
other:   []

calling: [5, 20, 12, 20] -> return 3, 3 differences due to longer list
other:   [5]


calling: [5, 7, 12] -> returns 2, the 5:4 and the extra 20
other:   [4, 7, 12, 20]

calling: [4, null,    0, -3,  7, 5] -> returns 4
other:   [4,    7, null, -3, 12]
```

Complete the following `getNumDifferences` method for the `LinkedList314` class.

```
/* pre: otherList != null
   post: per the question description. Neither list is altered*/
public int getNumDifferences (LinkedList314<E> other) {
```

3. Stacks and Queues (20 points) - Write a method that accepts two stacks as parameters and returns `true` if the stacks have the same number of elements in the same order, false otherwise.

This method is NOT part of the Stack class, so you do not have access to the private instance variables of the Stack class.

You may create and use a single stack as an auxiliary data structure in your method.

Consider the following examples:

```
s1 top -> 4          s2 top -> 4     returns true
          3                    3

s1 top -> 4          s2 top -> 4     returns false
          3                    3
          1                    2

s1 top -> 4          s2 top -> 4     returns false
          3                    3
          1

s1 top ->            s2 top ->       returns true (both stacks empty)

s1 top ->            s2 top -> 4     returns false (s1 empty)
```

You may assume none of the elements in either stack equal `null`.

You may use these methods for the stack class.

```
Stack314: isEmpty, push, pop, top
```

You may also use the `equals` method declared in the `Object` class. Do not use any other Java classes or methods.

Note, each stack **must** be restored to its original state by the time the method completes.

Do not use recursion.

Complete the method on the next page.

```
/* pre: s1 != null, s2 != null, no elements in s1 or s2 equal null
   post: per the problem description */
public <E> boolean stacksAreEqual(Stack314<E> s1, Stack314<E> s2) {
```

4. Maps (20 points) Write a method named `removePointsHalfAverageDistance` that accepts a `Map` that has `String`s as keys and `Point` objects as values. The method also accepts a target `String`. If the target `String` is a key in the map, the method removes from the map all key value pairs, other than the target, whose `Point` is **less than half** the average distance from the `Point` associated with the target. The average is calculated based on all of the values in the map other than the value associated with the target key. **Do not use recursion.**

For example, consider the following map. (quotes not shown on `String`s). Recall the format of the map is `{key1=value1, key2=value2, ..., keyN=valueN}` and in this case the values are `Point` objects in the form `(x, y)`.

```
{Austin=(0, 0), Buda=(0, -1), Dallas=(2, 10),
Dripping Springs=(-1, -1), Houston=(10, -1),
Pflugerville=(0, 1), Round Rock=(0, 2)}
```

If the target `String` is Austin, then the average distance from the `Point` object associated with Austin to all the other `Point`s is roughly 4.3. Half of the average is 2.15. When we remove the key-value pairs with a distance less than half of the average distance (other than the target key itself) we get the following map:

```
{Austin=(0, 0), Dallas=(2, 10), Houston=(10, -1), Round Rock=(1, 2)}
```

Pflugerville and Dripping Springs have been removed.

Recall the distance formula for two points: `distance = sqrt((x1 - x2)^2 + (y1 - y2)^2)`

You may use the following methods from the `Map` interface.

put(key, value)	adds a mapping from the given key to the given value
get(key)	returns the value mapped to the given key (null if none)
remove(key)	removes any existing mapping for the given key
size()	returns the number of key/value pairs in the map
keySet()	returns a set of the keys for this map.

You may call the `iterator` method on the key set and use all of the methods from the `Iterator` interface. You may use the `equals` method on any objects. You may use the `pow` and `sqrt` methods from the `Math` class.

You may use the following `Point` class.

```
public class Point {
     public Point(int x, int y)
     public int getX()
     public int getY()
}
```

Do not create or use any auxiliary data structures. **Complete the method on the next page.**

```
/* pre: m != null, target != null, if target is a key in m then
     m.size() >= 2. post: per the problem description */
public void removePointsHalfAverageDistance(Map<String, Point> m,
                                            String target) {
```

5. Recursive Backtracking (20 points) Remember the good old days of CS312? Remember Connect Four? When we studied recursion you probably said to yourself, "I bet I could add a computer player that crushes most humans, just like in Evil Hangman." Well, here is your chance.

Write a method that determines if it is possible for a given player, red or black, to win a game of connect four. from a given board state (set up).

Recall, Connect Four is played on a board with 6 rows and 7 columns. Players place a checker at the top of a column and the checker falls down to the lowest, open row in that column. A checker may not be placed in a column that is full. The players use red and black checkers. Red goes first. Players alternate turns until one player has four checkers in a row horizontally, vertically, or diagonally. It is possible for the game to end in a draw if no player can achieve four in a row.

Here is a sample board after 6 moves:

```
0 1 2 3 4 5 6 column numbers
. . . . . . .
. . . . . . .
. . . . . . .
. . . . . . .
r . . . . . .
r . b b b . r
```

In the sample board, black can win the game in two moves. It is red's turn and if they add a checker to column 1, black wins by adding a checker to column 5. Likewise, if red adds a checker to column 5, black wins by adding a checker to column 1. If red moves to any columns besides 1 or 5, black can win in 1 move.

Write a recursive backtracking method that determines if a given player can win the game from the current board state. For this question use the given `ConnectFourBoard` class that handles many of the low level details of the board. We use the `char`'s `'r'` and `'b'` to represent the two different players' checkers. `'.'` is used to represent open spaces.

Here is the public interface of the `ConnectFourBoard` class you may use on this question:

```
public class ConnectFourBoard {
      public static int NUM_COL = 7;

      // 0 <= col < NUM_COL
      public boolean columnIsOpen(int col) // can we drop a checker in column

      public boolean gameOver() // returns true if game is over

      // pre: gameOver()
      public char winner(); // returns char of winner. 'r' for red, 'b' for
                            // black, or 'd' for a draw with no winner

      // pre: 0 <= col < NUM_COL, checkerColor == 'r' or 'b'
      public void dropPiece(int col, char checkerColor)

      // pre: 0 <= col < NUM_COL
      public void pickUpTopChecker(int col) // remove top checker in col
}
```

Complete the method on the next page using recursive backtracking to determine if the player represented by **`goal`** can win the game from the given board state.

You may not use any Java classes or methods besides the `ConnectFourBoard` class.

Hints: First, you don't need a helper method. Second, don't alter the value of `goal`. Third, what makes this method so ***interesting*** is we have slightly different tasks if the current player is the same as goal or not. In the recursive case, If the current player is the same as goal, we can return `true` as soon as we find a single winning move. If the current player is **not** the same as goal, we return `false` as soon as we find one move that results in goal **not** winning.

```
// pre: board != null, goal and cur == 'r' or 'b', numMoves >= 0
// cur is the current player.
public boolean canWin(ConnectFourBoard b, char goal, char cur)
```