Points off	1	2	3	4A	4B		Total off	Net Score

CS 314 – Midterm 2 – Spring 2014

Your Name____________________________________

Your UTEID __________________________________

Instructions:

1. There are **4** questions on this test. 80 points available. Scores will be scaled to 200 points.
2. You have 2 hours to complete the test.
3. Place you answers on this test. Not the scratch paper.
4. You may not use a calculator or any other electronic devices while taking the test.
5. When writing a method, assume the preconditions of the method are met. Do not write code to check the preconditions.
6. On coding questions you may add helper methods.
7. When answering coding questions, ensure you follow the restrictions of the question.
8. 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.
9. 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 the order (Big O) of `methodA`? N = `num`. All methods from `Random` are O(1).

```
public LinkedList<Double> methodA(int num) {          _______________________
     LinkedList<Double> result = new LinkedList<Double>();
     Random r = new Random();
     for(int i = 0; i < num; i++)
          if(i % 2 == 0)
               result.add(r.nextDouble());
          else
               result.addFirst(r.nextDouble());
     return result;
}
```

B. What is the order (Big O) of `methodB`? N = `data.length` _______________________

```
public ArrayList<Integer> methodB(int[] data) {
     ArrayList<Integer> result = new ArrayList<Integer>();
     for(int i = 0; i < data.length; i++)
          result.add(data[data.length - i - 1]);

     for(int i = 0; i < data.length; i++)
          result.add(data[i]);

     return result;
}
```

C. What is the worst case order (Big O) of `methodC`? N = `list.size()`

____

```
// data != null
public double methodC(LinkedList<Double> list, double floor) {
     double result = 0;
     final int LIMIT = list.size();
     for(int i = 0; i < LIMIT; i += 2) {
          if(list.get(i) > floor) {
               result += list.get(i);
          }
     }
     return result;
}
```

D. What is the best case order (Big O) of `methodD`? N = `list.size()` ___________________

```
public int methodD(ArrayList<String> list) {
     int total = 0;
     int start = list.size() * 3 / 4;
     int stop = list.size() / 4;
     for(int i = start; i >= stop; i--) {
          String temp = list.remove(i);
          if(temp.length() > 10)
               total += temp.length();
     }
     return total;
}
```

E. What is returned by the method call `methodE(20)`? ______________________

```
public int methodE(int x) {
     if(x < 3)
          return 1;
     else
          return x + methodE(x / 3);
}
```

F. What is returned by the method call `methodF(4)`? ______________________

```
public int methodF(int x) {
     if(x <= 1)
          return 2;
     else
          return 2 + methodF(x - 1) + methodF(x - 2);
}
```

G. What is output by the method call `methodG(7, -1)`?

____

```
public void methodG(int x, int y) {
     if(x <= y)
          System.out.print("!");
     else {
          System.out.print(x);
          methodG(x - 1, y + 2);
          System.out.print(y);
     }
}
```

H. What is returned by the method call `methodH(new int[] {2, 1, 2, 1, 0, 3}, 3)`?
Note, the argument for `index` is `3` not `0`.

____

```
public int methodH(int[] arr, int index) {
     if(index == arr.length)
          return arr.length;
     else
          return arr[index] + methodH(arr, index + 1);
}
```

I. A method uses the selection sort algorithm to sort an array of `double`s. The method takes 5 seconds to complete given an array with 10,000 distinct elements in random order. What is the expected time for the method to complete given an array with 30,000 distinct elements in random order?

____

J. A method uses the quicksort algorithm to sort an array of `int`s. The method picks the middle element of an array (whole or sub array) as the pivot. The method takes 20 seconds to sort an array with 1,000,000 distinct elements in random order. What is the expected time for the method to complete given an array with 2,000,000 distinct elements in random order?

____

K. A method uses the quicksort algorithm to sort an array of `int`s. The method picks the first element of an array (whole or sub array) as the pivot. The method takes 10 seconds to sort an array of 10,000 distinct elements already sorted in ascending order. What is the expected time for the method to complete given an array with 20,000 distinct elements already sorted in ascending order?

____

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

```
3  15  5  6  +  -  *
```

M. What is the average case order (Big O) of the following method? _________________________

```
public int binarySearchM(LinkedList<Double> list, Double tgt) {
     int low = 0;
     int high = list.size() - 1;
     int result = -1;
     while(result == -1 && low <= high) {
          int mid = (low + high) / 2;
          int dir = tgt.compareTo(list.get(mid));
          if(dir == 0)
               result = mid;
          else if(dir < 0)
               high = mid - 1;
          else
               low = mid + 1;
     }
     return result;
}
```

N. What is output by the following code? Assume the `Queue314` class is a traditional queue class like the one we implemented in lecture.

____

```
Queue314<Integer> q = new Queue314<Integer>();
final int MAX = 4;
int count = 0;
for(int i = 0; i <= MAX; i++) {
     for(int j = 0; j <= i; j++) {
          q.enqueue(j);
          count++;
     }
}

final int LIMIT = count / 2;
count = 0;

while(count < LIMIT) {
     System.out.print(q.dequeue() + " ");
     count++;
}
```

O. You have an array with 1,000,000 distinct elements in random order.

You have to search the array 10 times to determine if a given element is present or not.

What will result in less work? Sorting the array with quicksort and then doing the searches using binary search OR just doing the searches with linear search. (without sorting)

Justify your answer with calculations.

P. What is output by the following code? _______________________________

```
ArrayList<Integer> list = new ArrayList<Integer>(5);
Iterator<Integer> it = new Iterator(list);
System.out.print(it.hasNext() + " " + it.next());
```

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

```
       X
     /   \
    Q     H
   / \     \
  M   V     Q
 /     \   /
G       A K
```

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

____

R. What is the result of an 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. What is output by the following code if `list1` is an `ArrayList<Integer>` that contains the following values: `[5, -3, -1, 2, 4, 7, 6]`

```
Iterator<Integer> it1 = list1.iterator();
int total = 0;
for(int i = 0; i < 3; i++) {
     total += it1.next();
     total += it1.next();
     it1.remove();
}
System.out.println(total + " " + list1);
```

____

2. stacks - 15 points. Write a client method that removes all elements from a stack that are less than a given value.

Assume the `Stack` class in this question has the `push`, `pop`, `top`, and `isEmpty` methods and a zero argument constructor that creates an empty `Stack`.

The method removes all elements in a `Stack` "less than" a given cutoff value, **but the relative order of the rest of the elements in the `Stack` is unchanged.** "Less than" is determined by the `compareTo` method.

Example: If the initial `Stack` is the one on the left and the cutoff value is 4 the resulting `Stack` is on the right.

```
Initial Stack                      Resulting Stack
top ->     4                       top ->     4
           2                                  7
           7                                  9
           2
           9
           3
```

The only class and methods you may use are the `Stack` class, including temporary `Stack`s, and the `compareTo` method from the `Comparable` interface.

Your method shall be as efficient as possible in terms of time and space given the constraints of the question.

```
// pre: st != null, cutoff != null
public <E extends Comparable<E>> void remove(Stack<E> st, E cutoff) {
```

3. linked lists - 15 points. Complete the `removeBetween` instance method for the `LinkedList314` class. The method removes all elements of the list between the first and second occurrence of a target value.

- 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 and last nodes in the chained structure of nodes.
- When the list is empty, `first` and `last` are set to `null`.
- None of the data in the list equals `null`.
- If the list is not empty the last node in the list has its next reference set to `null`.
- You may use the `Node` class and the `Object equals` method. You may not use other methods from `LinkedList314` unless you implement them as part of your answer.
- **You may not use any other Java classes or native arrays.**
- **Your method shall be as efficient as possible given the constraints of the question. Your method shall be O(1) space, meaning no matter how many elements are in the list your solution always uses the same amount of space.**

```
public class LinkedList314<E> {

     private Node<E> first; // refers to first node in the chain of nodes
     private Node<E> last; // refers to last node in the chain of nodes
}
```

The `Node` class.

```
public class Node<E> {
     public Node(E item, Node<E> next)
     public E getData()
     public Node<E> getNext(
     public void setData(E item)
     public void setNext(Node<E> next)
}
```

Examples.

```
[].removeBetween(A) ->  resulting list []

[A].removeBetween(A)  ->  resulting list [A]

[A, A].removeBetween(A)  ->  resulting list [A, A]

[A, A, A].removeBetween(A)  ->  resulting list[A, A, A]

[A, C, B, A].removeBetween(A)  ->  resulting list [A, A]

[C, C, C].removeBetween(A)  ->  resulting list [C, C, C]

[B, A, B, C, G, B, A, H, A, G].removeBetween(A)
     ->  resulting list [B, A, A, H, A, G]
```

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

```
/* pre: tgt != null
   post: Remove all elements from this list between the first and
   second occurrence of tgt.

   If there are not two occurrences of tgt in this LinkedList314
   or there are no elements between the first two occurrences of tgt,
   then this LinkedList314 is unchanged. */
public void removeBetween(E tgt) {
```

4. maps and recursion - 30 points. This question has two parts. The goal is find a ***valid coloring*** of a map (as in "a diagrammatic representation of an area of land or sea showing physical features, cities, roads, etc.", not the map data structure) with a given number of colors or show that no valid coloring exists.
A valid coloring is one in which no areas that border each other on the map are the same color.

Illinois
Colorado
Kansas
Missouri
Oklahoma
Arkansas
New Mexico
Mississippi
Louisiana
Texas
Gulf of Mexi

Consider this small example with a portion of the United States including only the states of Colorado, New Mexico, Kansas, Oklahoma, Texas, Missouri, Arkansas, Louisiana, Illinois, and Mississippi.

A `Map` data structure is used to store the areas of the map (states) and the areas (states) that border it. The keys of the `Map` are `String`s representing the names of areas (states) and the associated value is a `List` of `String`s of the areas (states) that border the key. Given the small example to the left, the `Map` would contain these keys and values:

Key (State)	Value (States that border the key)
Colorado	[Kansas, New Mexico, Oklahoma]
New Mexico	[Colorado, Oklahoma, Texas]
Texas	[New Mexico, Oklahoma, Arkansas, Louisiana]
Oklahoma	[Texas, New Mexico, Colorado, Kansas, Missouri, Arkansas]
Kansas	[Colorado, Oklahoma, Missouri]
Missouri	[Kanas, Oklahoma, Arkansas, Illinois]
Arkansas	[Texas, Oklahoma, Missouri, Mississippi, Louisiana]
Louisiana	[Texas, Arkansas, Mississippi]
Mississippi	[Louisiana, Arkansas]
Illinois	[Missouri]

Use another `Map` to store the color assigned to each area. Colors are represented by the Java `Color` class.

Part A. using maps - 15 points. Write a non-recursive helper method that determines if the current coloring is acceptable. A coloring is acceptable if there are no areas that border each other that have been assigned the same color. So for example if Texas has been colored `RED` and New Mexico has been colored `RED` the method returns false. If Texas has been assigned `RED` and New Mexico has been assigned `BLUE` but no other states have been colored, the method returns `true`. (The coloring is acceptable so far.)

Recall the following methods from the `Map` interface.

- `Set<K>` **`keySet`**`()` - Returns a Set view of the keys contained in this map.
- `V` **`get`**`(Object key)` - Returns the value to which the specified key is mapped, or null if this map contains no mapping for the key.
- `V` **`put`**`(K key, V value)` - Associates the specified value with the specified key in this map
- `V` **`remove`**`(K key)` - Removes the mapping for a key from this map if it is present.
- `boolean containsKey(K key)` - Returns true if key is in this mapping.

You may obtain iterators for `Collection`s and use the methods from the `Iterator` interface. Recall `Map`s do not have an `Iterator iterator()` method. You may use the `Color` class's `equals` method.

Do not construct any new data structures in your answer other than obtaining `Iterator` objects.

You method shall be as efficient as possible in terms of time and space given the constraints of the question.

```
/*   pre: borders != null, coloredAreas != null
     All keys in coloredAreas are keys in borders.
     All elements in the values of borders are also keys in borders.
     The number of keys in coloredAreas may be less than the number of
     keys in borders because some states have not been assigned a
     color.

     Return true if there are no color conflicts, false otherwise.
     Neither borders nor coloredAreas are altered by this method.
*/
public boolean colorsOkay(Map<String, List<String>> borders,
                          Map<String, Color> coloredAreas) {
```

Part B. 15 points. Complete a recursive backtracking helper method that returns `true` if it can find a valid coloring for the given map and colors, false otherwise. The method modifies a `Map<String, Color>` so that it holds a valid color scheme if one exists.

A valid coloring is one in which every area (state) is assigned a color and no areas (states) that border each other share a color. It is not necessary to use every one of the provided colors.

For the example map and the given colors RED, GREEN, BLUE, and YELLOW one valid coloring scheme is:

Key (Area)	Value (Colors)
Colorado	RED
New Mexico	GREEN
Texas	RED
Oklahoma	BLUE
Kansas	GREEN
Missouri	RED
Arkansas	GREEN
Louisiana	BLUE
Mississippi	RED
Illinois	GREEN

The public, method is:

```
/* pre: borders != null, colors != null, All elements in the values of
   borders are also keys in borders.

   post: return a Map<String, Color> with a valid coloring scheme if one
   exists for the given map and colors or null if no valid coloring exists.
   borders is not altered as a result of this method call. */

public Map<String, Color> findValidColorScheme(
                Map<String, List<String>> borders, Color[] colors) {

     // put all the areas on the map in a List
     List<String> areas = new ArrayList<String>(borders.keySet());

     // for assigning colors to areas
     Map<String, Color> result = new HashMap<String, Color>();

     // Kick off the recursion starting with the first area and return
     // appropriate result.
     if(helper(borders, colors, areas, 0, result))
          return result;
     else
          return null;
}
```

Complete the recursive backtracking method on the next page. The choices for the current area are the colors in the array named colors.

You may use the methods from part A and the `get` and `size` methods from `ArrayList`. Also, call the `colorsOkay` method from part A as appropriate. Do not rewrite that functionality in part B.

Complete the following method:

```
private boolean helper(Map<String, List<String>> borders,
     Color[] colors, List<String> areas, int currentArea,
     Map<String, Color> result) {
```