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

CS 314 – Final Exam – Spring 2017

Your Name_____

Your UTEID _____

Instructions:

1. There are **6** questions on this test. 100 points available. Scores will be scaled to 300 points.
2. You have 3 hours to complete the test.
3. Place your 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. Short answer - 1 point each, 20 points total. 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 shall 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. Consider the following `hashCode` method for a linked list class.
What is the most significant issue with the **results** of the method?

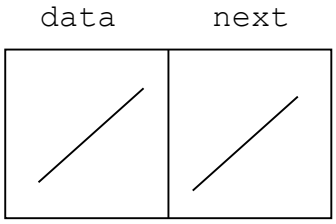
_____

```
public int hashCode() {
     // Instance method in a linked list class.
     int result = 0;
     for (int i = 0; i < size(); i++) {
          int temp = get(i).hashCode();
          result = temp + result;
     }
     return result;
}
```

B. Given a `LinkedList` that contains N elements what is the order, Big O, of the `hashCode` method in part A?

_____

C. Draw the variables, references, and objects that exist after the following code executes. Draw node objects as shown below and boxes for variables. (The example has both instance variables set to `null`. The example does not show any of the variables that actually refer to the Node object. You must show all variables and their references in your drawing.) Use arrows to show references and a forward slash to indicate variables that store `null`. Assume the Node class is the one from our singly linked list examples in lecture.




```
Node n1 = new Node(null, null); // data, next
Node n2 = new Node(new Node(n1, n1), n1);
n1.setData(new int[] {2, 5});
n1 = (Node) n2.getData();
n2.getNext().setNext(n2);
```

D. The following method takes 0.20 seconds to complete when N = 1,000,000. What is the expected time for the method to complete when N = 4,000,000? The question uses the Java `TreeSet` class.

_____

```
public static TreeSet<Integer> d(int N) {
     TreeSet<Integer> result = new TreeSet<>();
     for (int i = N; i >= 10; i -= 3) {
          result.add(i);
     }
     return result;
}
```

E. The following method takes 0.03 seconds to complete when N = 500,000. What is the expected time for the method to complete when N = 1,000,000? The question uses the `BinarySearchTree` class from assignment 9.

_____

```
public static BinarySearchTree<Integer> e(int N) {
     BinarySearchTree<Integer> result;
     result = new BinarySearchTree<>();
     Random r = new Random(); // O(1)
     for (int i = 0; i < N; i++) {
          int val = r.nextInt(50); // O(1)
          result.add(val);
     }
     return result;
}
```

F. The following values are added, one at a time, in the order shown, to an initially empty binary search tree. The tree uses the naive insertion algorithm presented in class.
What is the height of the resulting tree?

_____

```
5, -5, 0, 7, 0, 9, -5, 3, -2, -1
```

G. What is the result of a post order traversal of the resulting tree from part F?

_____

H. What is output by the following method if **`list`** initially contains these values in the order shown? **`[6, 3, 8, 8, 9, 9, 7, 9]`**

_____

```
public static void h(List<Integer> list) {
     if (list.size() > 0) {
          Iterator<Integer> it = list.iterator();
          int prev = it.next();
          while (it.hasNext()) {
               int curr = it.next();
               if (curr == prev) {
                    it.remove();
                    it.remove();
               }
               prev = curr;
          }
     }
     System.out.println(list);
}
```

I. The following values are inserted one at a time into an initially empty **max** heap. Draw the resulting **max** heap.

9, 4, 9, 2, 4, 10, 4

J. When discussing the map coloring problem, we represented the countries of South America using a graph. What were the vertices in the graph and when did an edge exist between two vertices?

_____

K. The *outdegree* of a vertex in a directed graph is the number of edges that originate at the vertex and lead to another vertex. Recall our graph class on assignment 11 used an adjacency list of edges. The other alternative presented for representing a graph was an adjacency matrix. Which representation makes it easier to determine the *outdegree* of a vertex and why?

_____

L. What is the order, Big O, of the following method? N = `data.length`.

_____

```
public static int el(int[] data, int tgt) {
     int result = 0;
     for (int i = 0; i < data.length; i++) {
          int tempTgt = tgt * data[i];
          for (int j = 1; j <= data.length; j *= 2) {
               int tot = 0;
               for (int k = 0; k < j; k++) {
                    tot += data[k];
               }
               if (tot > tempTgt) {
                    result++;
               }
          }
     }
     return result;
}
```

M. What can we be sure the following code will output?

```
String s1 = "";
String s2 = "";
Random r = new Random();
for (int i = 0; i < 4; i++) {
     s1 += (char) (r.nextInt(5) + 'M');
     s2 += (char) (r.nextInt(5) + 'A');
}
System.out.println(s1.compareTo(s2));
```

N. Given the following Huffman codes for the given values, draw the resulting Huffman code tree.

Value	Huffman Code
48	11
53	101
32	0
65	100

O. The following method takes 1 second to complete when `data.length = 500,000` and `set.size() = 1,000,000`. What is the expected time for the method to complete when `data.length = 2,000,000` and `set.size() = 4,000,000`. All elements in `data` are distinct and all elements in `data` are present in `set`.

```
public static int methodO(int[] data, HashSet<Integer> set) {
     int result = 0;
     for (int x : data)
          if (set.remove(x))
               result++;

     return result;
}
```

P. Recall the standard count format from the Huffman coding assignment. The frequency of every possible value in the original file was listed as a 32 bit integer. Here is an alternative header format that explicitly lists all the values in the original file, the length of their new Huffman code, and the code itself.

```
<Original Value><Length of Huffman Code><Huffman Code>
```

If `<Original Value>` and `<Length of Huffman Code>` are both 8 bit integers, when will this result in a smaller header than the standard count format?

_____

Q. Draw a Red Black Tree with the maximum possible height that contains the values 1, 2, 3, 4, 5, 6, 7. Label red nodes with the word RED. Do not label black nodes. Show the values in each node.

R. What is the order, Big O, of the following method? The method uses the Java `PriorityQueue` class which in turn uses a min heap as its internal storage container. N = `data.length`.

_____

```
public static Collection<Integer> r(int[] data) {
     PriorityQueue<Integer> result = new PriorityQueue<>();
     for (int x : data) {
          result.add(x);
     }
     return result;
}
```

S. What is output by the following code?

```
int[] vals = {6, 4, 3, 2, 3, 0, 2};
TreeMap<Integer, Integer> ts = new TreeMap<>();
for (int x : vals) {
     ts.put(vals[x], x);
}

System.out.print(ts);
// The Map toString takes the form:
// {key1=value1, key2=value2, ..., keyN=valueN}
```

_____

T. What is output by the method call **`t(14)`**?

```
public static void t(int x) {
     if (x <= 2) {
          System.out.print("t");
     } else {
          int y = x / 2;
          System.out.print(y);
          t(y);
          System.out.print(y + x);
     }
}
```

_____

2. Linked Lists - 16 points. Complete the `combine` instance method for the `LinkedList314` class. The method combines the calling object and the parameter into a new linked list object. For this question the linked lists always store `int`s. The nodes in the resulting list store the sum of the two corresponding elements (same position in list) from the original lists. The size of the resulting list is equal to the size of the smaller of the two original lists.

- You may use the provided zero argument constructor.
- **You may not use any other methods in the `LinkedList314` class unless you implement them yourself as a part of your solution.**
- **Do not use recursion.**
- 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` stores `null`.
- If the list is not empty, the last node in the chain of nodes next reference stores `null`.
- You may use the nested `Node` class.
- **You may not use any other Java classes or native arrays.**

```
public class LinkedList314 {

     // Refers to first node in the chain of nodes.
     private Node first;
     // No other instance variables

     public LinkedList314() {  first = null;   }

     // The nested Node class.
     private static class Node {
          private int data;
          private Node next;

          public Node(int d) { data = d; }
     }
}
```

Examples of calls to `combine(LinkedList314 other)`:

```
calling: [] -> returns [], empty list
other:   []

calling: [5] -> returns [], empty list
other:   []

calling: [5, 20, 12, 20] -> returns [15]
other:   [10]

calling: [5, 7,  7] -> returns [9, 14, 19]
other:   [4, 7, 12, 20]
```

Complete the `combine` method for the `LinkedList314` class on the next page

```
/* pre: other != null
   post: per the question description. Neither list is altered. */
public LinkedList314 combine(LinkedList314 other) {
```

3. Trees - 16 points. Complete a private instance method for a binary search tree class that checks the **Red Rule** for Red - Black trees. The binary search tree class uses a Red - Black tree as its internal storage container.

Recall the nodes in a Red - Black tree are colored red or black. The **Red Rule** states a node that is red cannot have any children that are also colored red.

Complete the private instance method for the `BST` class. You may use the nested `RBNode` class.

You are only checking if the **Red Rule** is met. Do not check if the Path Rule is met.

Do not use any other Java classes or methods.

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

    private int size; // number of elements in this BST
    private RBNode<E> root; //root of tree. root == null iff size == 0

     private static class RBNode<E extends Comparable<? super E>> {

          private E data;
          private RBNode<E> left; // left child, null if no left child
          private RBNode<E> right; // right child, null no rt. child

          private boolean isBlack; // true if node is colored black
                                   // false if node is colored red
```

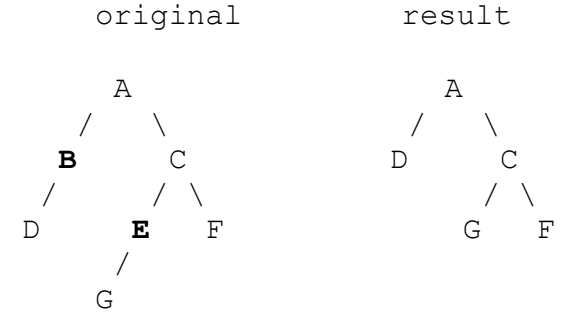
Complete the method on the next page.

```
/* pre: none
   post: Return true if Red Rule met in this tree, false otherwise. */
private boolean redRuleMet() {
```

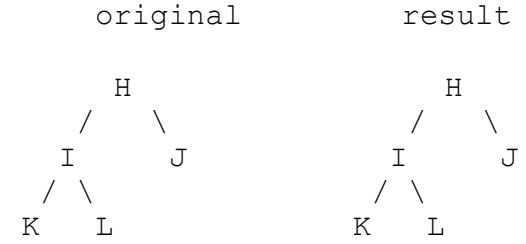
4. Binary Trees - 16 points. Complete a helper method for a binary tree class that alters the binary tree so that it is a full binary tree. Recall a full binary tree is one in which every node is a leaf or has 2 children. A full binary tree does not have any internal nodes with a single child. The binary tree class for this question stores `char`s, but it is **not** a binary search tree.

The helper method changes the binary tree to a full binary tree by removing any and all internal nodes that have a single child.

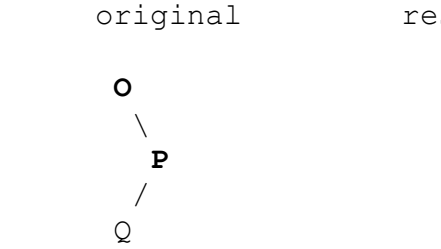
Consider the following examples:



In this example notice how the internal nodes that stored B and E have been removed.



No change, already a full binary tree,



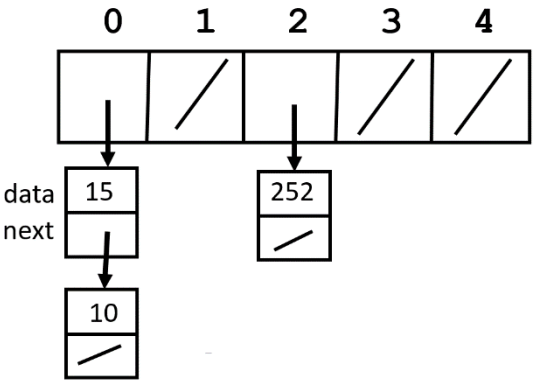
Nodes storing O and P removed. Node that contains Q becomes root of tree

```
public class BinaryTree {
     private BNode root; // root == null iff tree is empty

     private static class BNode {
          private char data;
          private BNode left;
          private BNode right;
     }
}
```

Complete the helper method for the `BinaryTree` class. You may use the `BNode` class. Do not use any other methods from the `BinaryTree` class unless you implement them as part of your answer. **Do not create any data structures or use any other Java classes or methods.**

```
/* pre: none
   post: This tree is a full binary tree. Internal nodes that had a
     single child have been removed from the tree per the problem
     description. */
public void makeFull() {
     root = help(root);
}

// complete the helper method including the method signature
```

5. Hash tables - 16 points. Complete the `remove` method for a hash table that uses closed addressing and buckets of singly linked nodes to store the elements of the table. Consider the following model of the internal storage container of a hash table that stores three elements. Forward slashes represent variables that store `null`. The hash table does not allow duplicates.



If the parameter sent to the `remove` method is not present, the hash table is not altered.

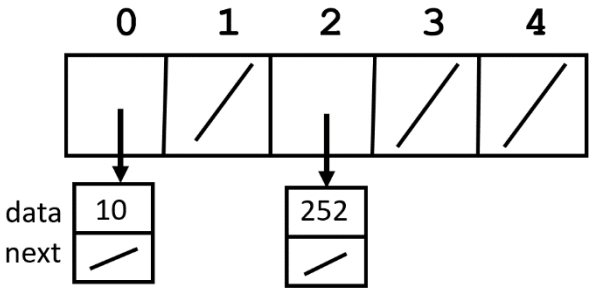
If the item to be removed is the only element present in a chain at a given index, the array element at that index shall be set to `null`.

You may use the provided `Node` class which is nested in the `Hashtable` class. You may use the `hashCode` method (which returns an `int`), the `equals` method from the `Object` class, and the absolute value (`abs`) method from the `Math` class.

Do not use any other Java classes or methods.

The `Hashtable` class and nested `Node` class for this question:

```
public class Hashtable<E> {

     private int size; // number of elements in Hashtable
     private Node<E>[] con; // container for chains of elements

     public boolean remove(E val) // to be completed

     private static class Node<E> {

          private E data;
          private Node<E> next; // set to null if last node in chain
```

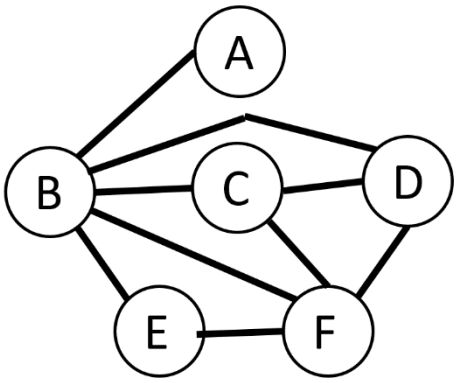


Hash table after removing 15.

Complete the method on the next page.

```
/* pre: val != null
   post: val is no longer present in this Hashtable. size is updated
     appropriately. Returns true if this Hashtable was altered as a
     result of this method call, false otherwise. */
public boolean remove(E val){
```

6. Graphs - 16 points. A *clique* is a subset of vertices in an undirected graph such that every pair of distinct vertices in the subset are *adjacent*. In other words, every pair of distinct vertices in the subset have an edge connecting them. Consider the following undirected graph.



Trivially, any pair of vertices that share an edge form a clique. So for example, (A, B) form a clique.

(B, C, F) form a clique. Every vertex has an edge to every other vertex in the subset. Note, (B, C, F) are part of a larger clique, (B, C, D, F) but are still considered a clique themselves.

(B, C, E, F) **do not** form a clique because there is no edge connecting C and E.

Write an instance method for the `Graph` class that determines if a given set of vertices form a clique or not. Assume all elements in the given set are present in the `Graph`.

Recall the following classes:

```
public class Graph {
    // The vertices in the graph.
    private Map<String, Vertex> vertices;

    private static class Vertex {
        private String name;
        private List<Edge> adjacent;
        private int scratch;
    }

    private static class Edge {
        private Vertex dest;
        private double cost;
        // equals NOT overridden
    }
}
```

You may use the `Map`, `Set`, `Vertex`, `Edge`, `List`, and `Iterator`, classes and the `equals` method.

Do not create ANY additional data structures besides `Iterators`. Do not use recursion.

```
/* pre: names != null, all elements of names, represent vertices in
          this Graph.
   post: Returns true if the vertices specified by the elements of
     names form a clique in this Graph, false otherwise. */

public boolean formsAClique(Set<String> names) {
```