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

CS 314 – Final Exam – Fall 2016

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 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 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. The following method takes 3 seconds to complete when `d.length` = 10,000. What is the expected time for the method to complete when `d.length` = 20,000?

```
public int a(int[] d) {
     int result = 0;
     for (int i = 0; i < d.length; i++)
          for (int j = d.length - 1; j >= i; j--)
               for (int k = 0; k < 10; k++)  {
                    int v = j - k;
                    if (v > 0 && d[i] == d[j] && d[i] != d[v])
                         result++;
               }
     return result;
}
```

____

B. What is returned by the method call `b(7)`? _________________________

```
public int b(int x) {
     if (x <= 0)
          return x * 3;
     else
          return 1 + b(4 + x);
}
```

C. What is returned by the method call `c(8)`? _________________________

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

D. It takes 1 second for the following code to complete when `list1.size()` = 25,000. What is the expected time for the method to complete when `list1.size()` = 50,000?
Assume `list1.size() == list2.size()`.

____

```
public int d(LinkedList<Integer> list1, LinkedList<Integer> list2) {
     int result = 0;
     int n = list1.size();
     for (int i = n - 1; i >= n / 2; i--) {
          for (int j = 0; j < n; j++) {
               if (list1.get(i) % list2.get(j) > 1000) {
                    result += list1.get(i) + list1.get(j);
               }
          }
     }
     return result;
}
```

E. Consider the following client code of the `IntList` class and the `resize` method from the `IntList` class. It takes 5 seconds for the client code to complete when N = 500,000. What is the expected time for the client code to complete when N = 2,000,000?

```
// Client code. The value N has already been initialized.
IntList list = new IntList(); // initial capacity 10
for (int i = 0; i < N; i++) {
     list.add( (int) (Math.random() * 1000000) );
}

public class IntList {
     private int[] con;
     private int size;

     private int resize() {
          int[] temp = new int[size * 2 + 1]; // A
          for (int i = 0; i < size; i++) {
               temp[i] = con[i];
          }
          con = temp;
     }
}
```

F. Assume the line marked **//A** in the `resize` method from question 1.E is altered to the following:

```
int[] temp = new int[size + 500]; // A
```

The same client code now takes 20 seconds to run when N = 100,000. What is the expected time for the client code to complete when N = 400,000?

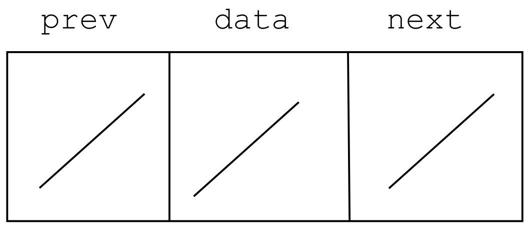
____

G. The following method takes 10 seconds to complete when N = 1,000,000. What is the expected time for the method to complete when N = 4,000,000? `TreeSet` is the `java.util.TreeSet` class.

```
public TreeSet<Integer> getNTree(int N) {
     TreeSet<Integer> result = new TreeSet<>();
     for (int i = 0; i < N; i++) {
          result.add(i * 2);
     }
     return result;
}
```

____

H. 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 all 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 doubly linked node from the linked list assignment and that the fields of the class are all `public`.



```
DoubleListNode<Object> n1 =
      new DoubleListNode <Object>(null, "**", null);
                 // parameters are prev, data, next

DoubleListNode<Object> n2 =
      new DoubleListNode <Object>(null, new int[]{1, 5}, n1);

n1.next = n2.next;
n1.prev = n2;
```

I. The following values are inserted into an initially empty binary search tree in the order shown. The binary search tree using the naïve insertion algorithm demonstrated in lecture. What is the result of a post order traversal of the resulting tree?

`7, 12, 5, 10, 12, 7, -5, 12, 8`

____

J. The following method takes 10 seconds to complete when N = 50,000. What is the expected time for the method to complete when N = 150,000? The `BST_314` class is the binary search tree class we developed in lecture. The binary search tree using the naïve insertion algorithm and implements add iteratively.

```
public BST_314<Integer> getNTree(int N) {
     BST_314<Integer> result = new BST_314<>();
     for (int i = 0; i < N; i += 2) {
          result.add(i / 10);
     }
     return result;
}
```

____

K. Recall on the binary search tree assignment you completed an experiment in which you inserted N random values to a binary search tree and then determined the height of the tree. A student has turned in the following data for their experiment:

N	AVERAGE HEIGHT OF TREE
1,000	11.1
2,000	11.9
4,000	13.0
8,000	14.1
16,000	15.2

I am skeptical the student actually ran the experiment. Explain why I am skeptical.
Limit your answer to one, concise sentence.

____

L. The following code takes 10 seconds to complete when N = 1,000,000. What is the expected time for the method to complete when N = 2,000,000? Assume `Math.random` is O(1).

```
public HashSet<Double> getRandomSet(int N) {
     HashSet<Double> set = new HashSet<>();
     for (int i = 0; i < N; i++) {
          set.add(Math.random());
     }
     return set;
}
```

____

M. What is the worst case order (Big O) of the following method. The hastSet already contains N elements. N = `hashSet.size()`.

```
public void addVals(int[] data, HashSet<Integer> hashSet) {
     int i = 0;
     while (i < data.length && i < 5) {
          hashSet.add(data[i]);
          i++;
     }
}
```

____

N. The following values are inserted 1 at a time in the order shown into a min heap using the algorithm demonstrated in lecture. Draw the resulting min heap as a tree.

9, 5, 3, 9, 0, 5

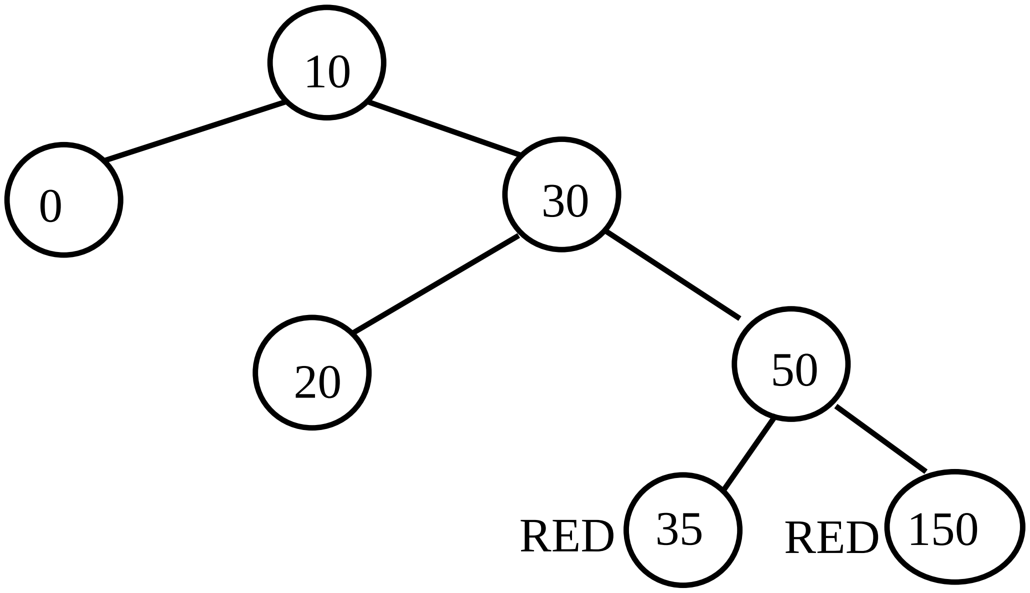
O. Consider the following code. The `PrintStream` object is connected to an initially empty file. When the file is processed via a SimpleHuffProcessor, the resulting file is **larger** than the original. Explain why in one concise sentence.

```
public void createFile(PrintStream out) {
     Random r = new Random();
     final int LIMIT = 256;
     for (int i = 0; i < 2_000_000; i++) {
          out.print( (char) r.nextInt(LIMIT) );
     }
}
```

____

P. Is the following tree a Red-Black tree? If not, explain why not.
Nodes not labeled RED are BLACK.

____



Q. What is the cost of the shortest (cheapest) path from Vertex A to Vertex G in the example Graph on coding question 5?

____

R. In general why would you implement an algorithm recursively rather than iteratively?

____

S. On the graph assignment for the year 2008 there were approximately 200 distinct teams and approximately 800 edges in the graph. Is this a sparse graph or a dense graph?

____

T. What is the advantage of creating a dynamic programming solution to solution rather than creating a solution that uses recursion?

____

2. Linked Lists - 16 points. Complete the `removeEveryNth` instance method for the `LinkedList314` class. The method removes the first element in the linked list and every Nth element after that. The method returns the number of elements removed from the list.

- **<u>You may not use any other methods in the `LinkedList314` class unless you implement them yourself as a part of your solution.</u>**
- The `LinkedList314` class uses singly linked nodes.
- The list has a reference 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 `removeEveryNth(int n)`. In this example the lists contain `Integer` objects. The elements that are removed are bolded. The resulting list is shown on the right.

```
[].removeEveryNth(3) ->  [], returns 0

[0].removeEveryNth(2) ->  [], returns 1

[0, 1].removeEveryNth(2) ->  [1], returns 1

[0, 1, 2].removeEveryNth(2) ->  [1], returns 2

[0, 1, 2, 3].removeEveryNth(2) ->  [1, 3], returns 2

[0, 1, 2, 3].removeEveryNth(1) ->  [], returns 4

[0, 1, 2, 3].removeEveryNth(3) ->  [1, 2], returns 2

[0].removeEveryNth(3) ->  [],returns 1

[0, 1].removeEveryNth(3) ->  [1], returns 1

[0, 1, 2, 3, 4, 5, 6, 7].removeEveryNth(3) ->  [1, 2, 4, 5, 7],
                                                returns 3
```

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

```
/* pre: n >= 1
   post: per the question description */
public int removeEveryNth(int n){
```

3. Binary Trees - 16 points. Complete the `getCombinedTrees` method for a binary tree of `int`s class. The binary tree class for this question stores `int`s, but it is **not** a binary search tree.

The method creates and returns a new `IntTree` that is the combination of the nodes of the calling `IntTree` and another `IntTree` sent as a parameter.

You may use the nested `BNode` class, but no other Java classes or methods.

Consider the following trees and the expected result.

```
      this             other                    result

       8                18                         3
     /   \            /    \                     /   \
   9      4         12      19                 3       3
  /      / \          \       \               / \     / \
 5      6   0         13       17            1   2   1   3
       /                         \                  /     \
      7                           15               1       2
                                 /                        /
                               14                        2
```

A node exists in the result if it exists in the same position in either the calling tree, the other tree or both. Note, the result is based on the **structure** of the two original trees, not the values in the original trees.

All values stored in nodes in the resulting tree will be 1, 2, or 3. A node stores a 1 if that node existed only in the calling tree (this), a 2 if that node existed only in the other tree, and a 3 if it existed in both trees.

For example the node in the calling tree that contains a 5 (left - left from root) only exists in that tree. So the node in the result (left - left from root) contains a 1.

The node in the other tree that contains a 13 (left - right from root) only exists in that tree. So the node the result (left - right from root) contains a 2.

Both trees have a root node, so the root node in the result contains a 3.

```java
public class IntTree {
     private BNode root; // root == null iff tree is empty
     public IntTree() { root = null; }

     private static class BNode {
          private int value;
          private BNode left;
          private BNode right;
     }
}
```

Complete the following instance method for the `IntTree` class. You may use the `IntTree` constructor and the `BNode` class. Do not use any other methods from the `IntTree` class unless you implement them as part of your answer. **Do not create any data structures besides the resulting `IntTree`.**

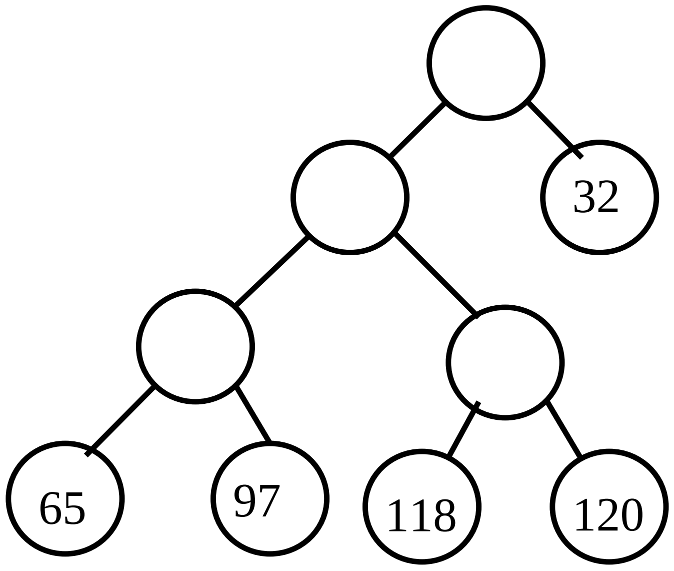
```
/* pre: other != null
   post: Return a new IntTree per the problem description.
     Neither this or other are altered as a result of this method. */
public IntTree getCombinedTrees(IntTree other) {
```

4. Huffman Code Trees (16 points) - On the Huffman Coding assignment there were two ways of building the tree from information in the header of the compressed file. The first was based on the frequency of each value. The second was based on a binary representation of the tree itself. This question deals with a third way of building the tree, based on the codes for the values themselves.

Consider the following example of chunks and codes.

Original Value	Code
65	"000"
32	"1"
97	"001"
118	"010"
120	"011"

Given the above chunks and codes the resulting Huffman code tree is:



Complete a constructor for the `HuffmanTree` class that creates a new Huffman code tree based on a `Map` of `Integer` values and `String` codes. Here is the `HuffmanTree` class the method is a part of.

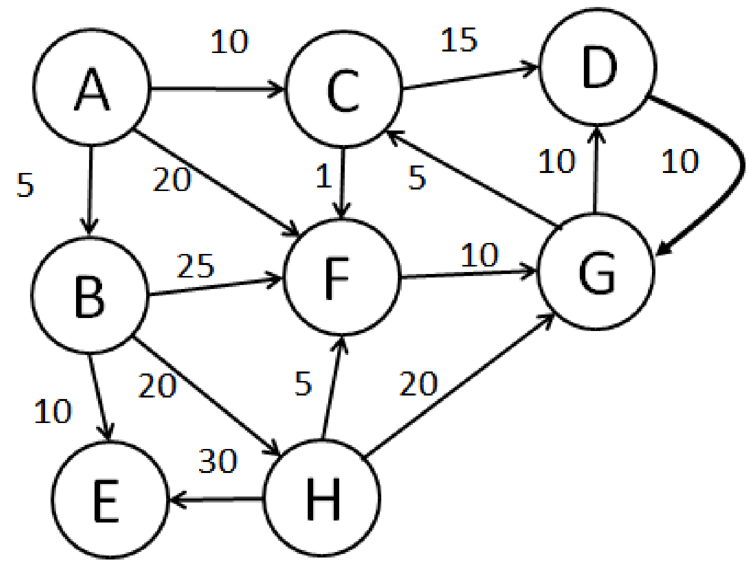
```
public class HuffmanTree {

      private TreeNode root;

      private static class TreeNode {

            private int value;
            private int freq;
            private TreeNode left;
            private TreeNode right;
      }
```

Complete the following method. You may use methods from the Map interface, iterators, and Strings.
DO NOT USE RECURSION IN YOUR ANSWER.

```
/* pre: codes != null
        all chars in code's values are equal to '0' or '1'
   post: the HuffmanTree has been created based on codes
*/
public HuffmanTree(Map<Integer, String> codes) {
```

5. Graphs - 16 points. Write a method that uses recursive backtracking to find the *longest path* from one vertex to another in a directed, weighted graph. For a weighted graph the longest path is the costliest path from the start vertex to the goal vertex.

Restrict the longest path to a *simple path*. A simple path is one where no vertex is visited more than once. In a graph with cycles such as the one below we could create an infinite cost in some cases by running through the cycle of C to F to G and back to C.



For example if the start vertex is A and the goal vertex is G the costliest path is A -> B -> H -> G with a total cost of 45.

Complete a helper method that uses recursive back tracking to find the longest path (highest cost) from the given start vertex to the given end vertex.

You may assume there is at least one path from the start vertex to the end vertex.

You may assume all edge costs are greater than or equal to 0.

```
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;
    }
}
```

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

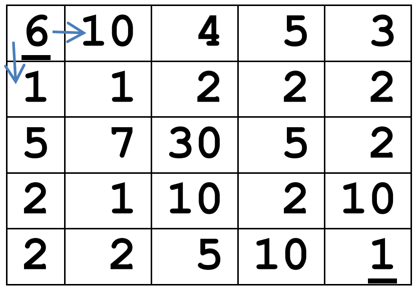
Do not create ANY additional data structures.

```
/* pre: start and end are vertices in this Graph. There is at least
     one path from start to end.
   post: Returns the longest, simple path from start to end.*/

public double longestPath(String start, String end) {
     clearAll(); // resets are Vertex scratch variables to 0
     // complete the parameters for your method
     return help(
}

private double help(
```

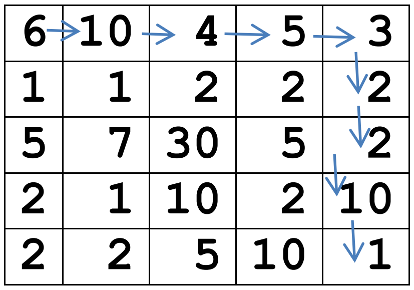
6. Dynamic Programming - 16 points. Write a dynamic programming solution for the following problem. A rectangular matrix of `ints` stores the number of coins in each cell. Each individual coin has the same value. You must start in the top left corner of the matrix and move to the bottom right corner. **When moving to the next cell you are only allowed to move down or to the right.**

We pick up the coins in each cell we pass through including the top left and bottom right cells. The goal is to find the path that results in the greatest number of coins being collected.

Recall we must start from the top left cell that contains 6 coins and we must end in the bottom right cell that contains 1 coin. From a given cell we can only move down or to the right. So from the top left cell with 6 coins we may move down to the cell with 1 coin or to the right to the cell that contains 10 coins.



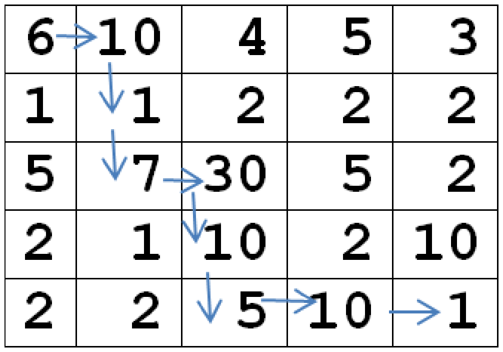
6	10	4	5	3
1	1	2	2	2
5	7	30	5	2
2	1	10	2	10
2	2	5	10	1



6	10	4	5	3
1	1	2	2	2
5	7	30	5	2
2	1	10	2	10
2	2	5	10	1

One greedy solution would simply go down or right to the cell that has the larger number of coins. Clearly that doesn't always give the optimal answer. The result of the proposed greedy solution (simply moving from the current cell to the cell with the max coins down or to the right) doesn't work with the given matrix. At cell[0][1] with 10 coins the greedy solution goes right to the cell with 4 coins instead of down to the cell with 1 coin. But this approach causes us to miss out on the cell with 30 coins.

The optimal solution is shown below. This results in 80 coins being collected.



6	10	4	5	3
1	1	2	2	2
5	7	30	5	2
2	1	10	2	10
2	2	5	10	1

We could implement a brute force, recursive back tracking solution that explores all paths and returns the optimal result. The choices at each cell would be to go right if possible and down if possible.

Instead, implement a dynamic programming solution to solve this problem.

Your solution must use an additional 2d array of ints the same size as the 2d array of ints that contains the coins.

At each cell realize the only way to get to that cell is either from the cell above or to the cell to the left.

Implement your **dynamic programming** solution on the next page. Do not use recursion!

Do not use any other data structures besides the 2d array of ints the same size as the original matrix.

Your solution will be graded for clarity and simplicity in addition to correctness.

```
// pre: coins != null, coins is a rectangular matrix
// post: Return the maximum number of coins that can be obtained when
// moving from the top left cell of coins to the bottom right cell
// while following the restriction of only moving down or right.
public static int maxCoins(int[][] coins) {
```