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

CS 314 – Final Exam – Fall 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 any 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. The root of a complete binary tree has a height of 4. What is the minimum number of nodes in the tree that have a depth of 4?

____

B. The root of a complete binary tree has a height of 5. What is the maximum number of nodes in the tree that have a depth of 5?

____

C. The *indegree* of a vertex V in a directed graph is the number of edges that originate at other vertices and lead directly to V. Recall our graph class on assignment 11 used an adjacency list of edges. Recall one alternative for representing a graph is an adjacency matrix. Which representation is faster when determining the *indegree* of a vertex and why?

____

D. Most implementations of graphs do not use an adjacency matrix to represent the graph. Why not?

____

E. Consider the following array which is the internal storage container for a min heap of size 9. Draw the min heap as a tree. Element indices are shown in the top row and the element is shown in the bottom row.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	3	5	10	12	8	37	19	25	13	21	42	53	60	65	62

F. The value 3 is added to the heap in part E. Draw the new heap as a tree.

G. What is the worst case order for the **`remove`** method of a hash table that uses open addressing and linear probing to resolve collisions and currently has N elements? The hash table uses a load limit of 0.75. When does the worst case occur?

____

H. What is the *path rule* for Red Black Trees?

____

I. What is the minimum number of bits needed to encode 20 distinct items?

____

J. The following method takes 0.0001 seconds to complete when n = `1000`. What is the expected time for the method to complete when n = `4000`?

____

```
public static double[] j(int n) {
     double[] result = new double[n];
     sort(result); // uses mergesort as shown in class
     return result;
}
```

K. What is returned by the method call **`k(7)`**? _______________________

```
public static int k(int x) {
     if (x <= 3)
          return x * 2;
     return k(x - 3) + x + k(x - 1);
}
```

____

L. The following method takes 40 seconds to complete when `data.length` = 500,000 and the maximum value in the array has 9 digits. What is the expected time for the method to complete when `data.length` = 1,000,000 and the maximum value in the array has 9 digits?

____

```
public static void el(int[] data) {
     int max = findMax(data); // linear search
     sort(data, max); // radix sort
}
```

M. The following method takes 1 second to complete when `n = 1_000_000`. How long do you expect the method to take when `n = 2_000_000`? The `PriorityQueue` class uses a min heap as its internal storage container.

____

```
public static PriortyQueue<Integer> m(int n) {
     PriortyQueue<Integer> result = new PriorityQueue<Integer>();
     for (int i = 0; i < n; i++) {
          result.enqueue(n);
     }
     return result;
}
```

N. Consider the following `SortedSet` class implementation.

```
private ArrayList<E> con;

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

public boolean add(E val) {
     if (!this.contains(val)) { // uses binary search
          con.add(val);
          sort(con); // mergesort
          return true;
     }
     return false;
}
```

What is the order (Big O) of the following method? Assume `Math.random` is O(1).

____

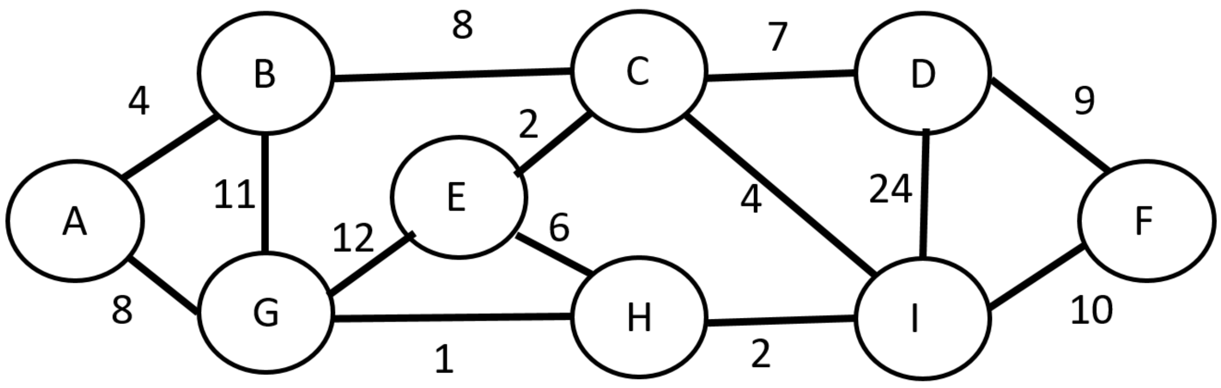
```
public static SortedSet<Double> getSet(int n) {
     SortedSet<Double> result = new SortedSet<>();
     for (int i = 0; i < n; i++) {
          result.add(Math.random());
     }
     return result;
}
```

O. The `Collections.bianrySearch` method accepts any collection that implements the Java `List` interface. The Java `LinkedList` class implements the `List` interface. If the `Collections.binarySearch` method used the binary search algorithm presented in class, what would the average case order (Big O) of the method be when passed a `LinkedList`?

____

P. Given the undirected, weighted graph below, what is the cost of the lowest cost path from vertex D to vertex G?

____



Q. Given the graph above, what is the cost of a minimum spanning tree for the graph?

____

R. What is output by the following code? It uses the Java `Stack` class.

```
Stack<Integer> st = new Stack<>();
for (int i = 1; i < 50; i += i + 1) {
     st.push(i - 1);
}
for (int i = 0; i < st.size(); i++) {
     System.out.print(st.pop() + " ");
}
```

____

S. 500,000 distinct items that are in random order are added to an initially empty binary search tree that uses the naive insertion algorithm. What is the expected height of the resulting tree? Given an actual value, not a bounds.

____

T. The following method takes 4 seconds to complete when `n = 1_000_000`. What is the expected time to complete when `n = 2_000_000`?

____

```
public static TreeSet<Double> t(int n) {
     TreeSet<Double> result = new TreeSet<>();
     for (int i = 0 ; i < n; i++) {
          result.add(i * 1.0);
     }
     return result;
}
```

2. Linked Lists - 16 points. Complete the **`insertAndRemove`** instance method for a linked list class. The method inserts a given value after each occurrence of a target value and removes all occurrences of a second 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 **doubly 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` and the first node's `prev` reference stores `null`.
- You may use the nested `Node` class.
- **You may not use any other Java classes or native arrays.**
- **You may use the Object equals method.**

```
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 {
          private E data;
          private Node<E> next;
          private Node<E> prev;
          // No other methods. Default constructor only.
     }
}
```

Examples of calls to `insertAndRemove(E insertAfter, E insertVal, E remove)`. Values shown in the examples are **`String`**s. The method returns the number of elements in the new list

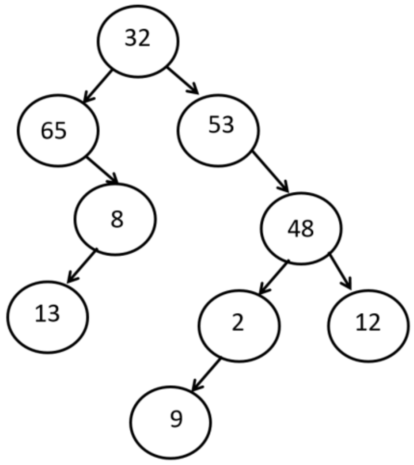
```
[A, D, A, B, A].insertAndRemove(A, M, B) -> [A, M, D, A, M, A, M]
returns 7

[A, D, A, B, A].insertAndRemove(J, M, C) -> [A, D, A, B, A] returns 5

[].insertAndRemove(A, M, B) -> [] returns 0

[B, B, B].insertAndRemove(A, M, B) -> [] returns 0

[B, J, A, J, B].insertAndRemove(A, J, B) -> [J, A, J, J] returns 4
```

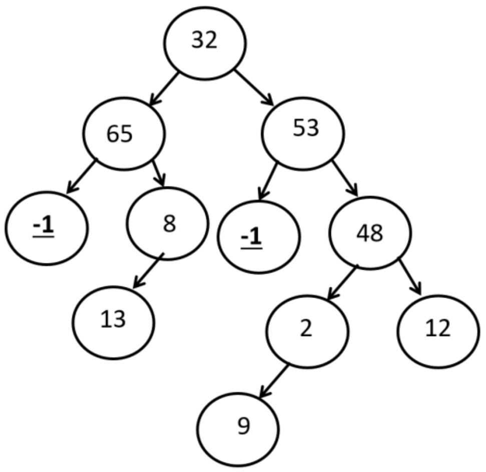
Complete the `insertAndRemove` method for the `LinkedList314` class on the next page

```
/* pre: insertAfter != null, insertVal != null, remove != null
     insertAfter, insertVal, and remove are all distinct values.
   post: per the question description. */
public int insertAndRemove(E insertAfter, E insertVal, E remove) {
```

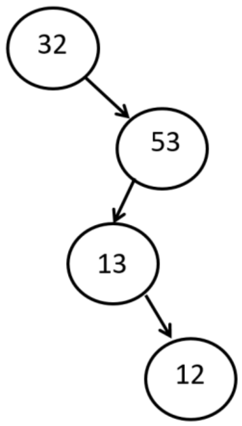
3. Trees -16 points. Complete a method for a binary tree class that ensures the tree is perfect up to and including a given depth. Recall a perfect tree has the maximum number of nodes at every level. For this method you are only ensuring the tree is perfect up to and including the given depth.

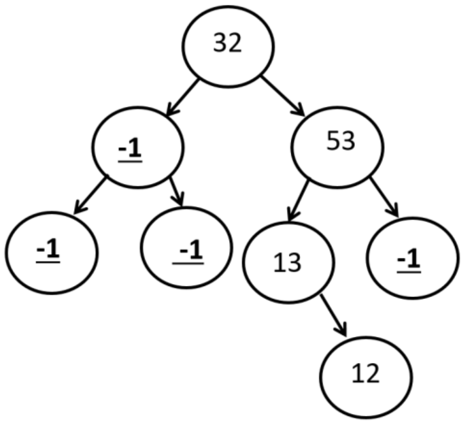
For example, given the tree on the left and a depth of 2, the resulting tree is shown on the right. The value to place in the new nodes is passed as a parameter to the method.





Another example with the depth equal to 2. Original tree on the left and resulting tree on the right.





Use the given **`BinaryTree`** and **`BNode`** class. Do not use any other Java classes or methods.

```
public class BinaryTree {

     private int size; // number of elements in this binary tree
     private BNode root; //root of tree. root == null iff size == 0

     private static class BNode {

          private int data;
          private BNode left; // left child, null if no left child
          private BNode right; // right child, null no rt. child

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

Complete the method on the next page.

```
/* pre: depth >= 0
   post: Per the problem description. size updated
     Returns the number of nodes added to the tree*/
private int makePerfectToGivenDepth(int depth,int newVal) {
```

4. Maps- 16 points. Reference counting is a garbage collection technique. We track the number of references to each object. When the reference count for an object is 0, no other objects are referring to the object and it can be deleted.

In this question we use a map. The keys are integers that represent objects. The associated value is an array of **`int`**s with at least one element. The first value in the array is the number of references to the object represented by the key. There may be other elements in the array. The array elements beyond the first represent the objects the key itself references.

Note, in the example, not all objects are stored in the map. Therefore, an object's reference count may be greater than 0 even though none of the objects in the map refer to the object. Likewise, an object may have references to objects that do not appear in the map. Also note in the example all of the values are sorted for ease of viewing, but this may not be the case in the actual map.

Consider the following small example.

123	[1, 457, 755, 900]
222	[1]
335	[2]
457	[1, 335, 900]
755	[3, 222]
900	[4]

If we are told to decrement the reference counter for object 335 (`-335`), its reference count goes from 2 to 1. No other change needs to take place.

If we are told to increment the reference count for 755, its reference count goes from 3 to 4. No other changes need to take place.

If we are told to increment the reference count for 800 we must add that key to the map and create an array of length 1 with a value equal to 1.

If we are told to decrement the reference count of 222 (`-222`) we must remove that element from the map because its reference count becomes 0.

Finally, it we are told to decrement the reference count of 123 (`-123`), its count goes from 1 to 0. It must be removed from the map and the objects it references must have their reference counts decremented. 755 and 900 each have their reference counts decremented. So does 457, which reduces its count to 0, so now it must be removed and the objects it references, 335 and 900 have their counts decremented.

Here is the map after performing **<u>only</u>** the last change, decrementing the reference count for 123:

222	[1]
335	[1]
755	[2, 222]
900	[2]

123 and 457 have been removed from the map.

For this question you are passed an **`int`** representing the object reference to alter and a **`Map<Integer, int[]>`** representing the objects, their reference counts, and the objects they in turn reference. The **`int`** will be positive if we are to increment the reference count and negative if we are to decrement the reference count.

You may use the following methods:
`Map`: `put(K key, V value), V get(K key), V remove(K key)` removes the given key

The only data structure you may create are arrays of ints of length one when adding a value to the map that was not already present.

Do not create or use any other data structures in your answer.

```
/* pre:  refMap != null, objectRef != 0
   post: per the problem statement. */
public void updateRefCount(int objectRef, Map<Integer, int[]> refMap) {
```

5. Graphs - 16 points. Complete an instance method for the **`Graph`** class that determines if a path exists from one vertex to another, along with its cost, using the *depth first search algorithm* and recursive backtracking. Recall, depth first searching explores as far as possible away from a vertex before backtracking.

Recall the following classes:

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

     // for all vertices, set scratch to 0 and prev to null.
    private void clearAll()

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

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

You may use the **`get`** method from the **`Map`** class and the **`get`** and **`size`** methods from the **`List`** interface. You may use the for-each loop.

You may use the **`clearAll`** method from the **`Graph`** class.

Do not add or remove any elements from the **`Graph's`** **`Map`** of vertices.

When complete the path from the start vertex to the destination vertex is stored via the **`prev`** references in the **`Vertex`** objects in the path, if one exists.

All costs in the **`Graph`** are **`> 0`**.

The method returns the cost of the path found or **`-1.0`** if there is no path from the start vertex to the destination vertex. Note, we are not finding the shortest (lowest cost) path.

Do not create ANY additional data structures.

```
/* pre: start != null, dest != null, !start.equals(dest)
   post: Returns the cost of the first path found from start to dest
if one exists or -1.0 if no path exists. The method performs a depth
first search */
public double dfsPathCost(String start, String dest) {
```

6. Encoding - 16 points. Run length encoding, encodes data by listing the count of a value and the value instead of each individual value.

For example, the data **`111111111`** is encoded as **`9 1`**. Run length encoding can lead to compression if a file has long runs of the same value such as images with a limited number of colors, especially black and white images.

Write a method that determines the difference between the number of bits in a file and the number of bits it would take to encode the file with run length encoding. Our run length encoding format uses an 8-bit integer for the run length and a single bit for the value.

Using 8 bits for our run lengths mean we have values of 0 to 255. However, since we don't need to represent a run length of 0, each 8-bit integer actually represents a count of one more. 0 indicates a run length of run, 1 indicates a run length of 2, and so forth.

For example, a file of five hundred 1's followed by three 0's, followed by one hundred 1's would be encoded with the following binary (spaces added for clarity) using our run length encoding scheme.

```
11111111 1 11110011 1 00000010 0 01100011 1
```

The data above, when interpreted as base 10 values is **`255 1 243 1 2 0 99 1`**. Recall each run length value would be increased by 1 by the decoder.

You may use the **`BitInputReader`** from the Huffman Encoding assignment and its **`readBits`** method:

```
public int readBits(int howManyBits)
```

Returns the number of bits requested as rightmost bits in returned value, returns -1 if not enough bits available to satisfy the request.

The source file will have at least 1 byte of data.

Complete the method on the next page.

Do not use any other Java classes or objects.

Question. (2 points). What average run length is required so the technique in this question results in compression? Explain your answer.

____

```
/* in != null, at least one byte of data in source
   Return the difference between the number of bits in the file in is
connected to and the number of bits required to encode the same file
use the run length encoding technique described in the question. */
public int bitsSaved(BitInputReader in) {
```