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

CS 314 – Final Exam – Spring 2018

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.)
e. Assume $\log_2(1{,}000) = 10$ and $\log_2(1{,}000{,}000) = 20$.

A. What is the height of a complete binary tree with 120 nodes? _____

B. Min and max heaps typically use arrays as internal storage containers.
Why **don't** binary search trees typically use arrays as internal storage containers?

_____

C. The best case of the quicksort algorithm occurs when the pivot is the median value of the unsorted data. Why don't we find the median and use it as the pivot in order to achieve the best case?

_____

D. What is returned by the method call `d(6)`? _____

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

E. What is output by the method call `e1()`? _____

```
private static int c;

public static void e1() {
     c = 0;
     e2(3);
     System.out.print(c);
     c = 0;
     e2(6);
     System.out.print(" " + c);
}

public static int e2(int x) {
     c++;
     if (x == 0)
          return 1;
     else
          return e2(x - 1) + e2(x - 1);
}
```

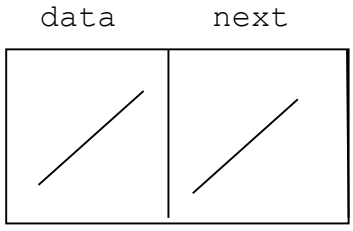
F. The following values are added one at a time, in the order shown from left to right to a binary min heap. The heap uses the insertion algorithm demonstrated in class. Draw the resulting heap.

12, 42, 37, 73, 37, 19

G. Using the techniques developed in class, what is the T(N) of the following code? `N = data.length`.

```
// pre: no elements of data == 0
public static int g(int[] data) {
     int t = 0;
     for (int i = 0; i < data.length; i++) {
          int temp = data[i] % 10;
          t += temp;
          for (int j = data.length - 1; j >= 0; j -= 2) {
               int temp2 = data[i] % data[j];
               t += temp2;
          }
     }
     t += data.length % 10;
     return t;
}
```

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 all 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 singly linked node from the linked list examples we did in class and that the fields of the class are all `public`.




```
Node<String> n1 = new Node<>("AB", null); // params are (data, next)
n1 = new Node<>(n1.data, n1);
Node<String> n2 = new Node<>(null, null);
n1.next.next = n2;
n2.data = n1.data;
```

I. The following values are inserted in the order shown, left to right, to a red-black tree. Draw the resulting tree, including labeling each node's color.

7, 0, 5

J. A method is O(N!). It takes 1 minute for the method to complete when N = 39. What is the expected time for the method to complete when N = 41?

_____

K. The following method takes 0.75 seconds to complete when `words.length` = 500,000. What is the expected time for the method to complete when `words.length` = 2,000,000? In each case all elements of `words` are distinct.

_____

```
public static HashSet<String> k(String[] words) {
     HashSet<String> result = new HashSet<>();
     for (String s : words)
          result.add(s);
     return result;
}
```

L. The following values are inserted in the order shown, left to right, into an initially empty binary search tree using the naive insertion algorithm demonstrated in class. What is the result of a post order traversal of the resulting tree?

-6, 21, 17, 21, 12, 10, 19

_____

M. Given the choice of using an array based list or a singly linked list with references to the first and last nodes in the structure, which would you use as the internal storage container for a queue? Why?

_____

N. Consider the following method.

```
// pre: out != null. out is connected to an initially empty file.
public static void n (PrintStream out) {
     Random r = new Random();
     for (int i = 0; i < 1_000_000; i++) {
          char c = (char) r.nextInt(256);
          out.print(c);
     }
}
```

When compressing the file created by method **n** using the Huffman Encoding algorithm and the standard count format, will the compressed file be larger, smaller, or the same size as the original file? Why?

_____

O. What is output by the following code?

```
Map<String, Integer> m = new TreeMap<>();
m.put("AB", 12);
m.put("IT", 12);
m.put("DO", 5);
m.put("IS", 7);
m.put("AB", m.put("DO", m.get("AB")));
System.out.print(m);
```

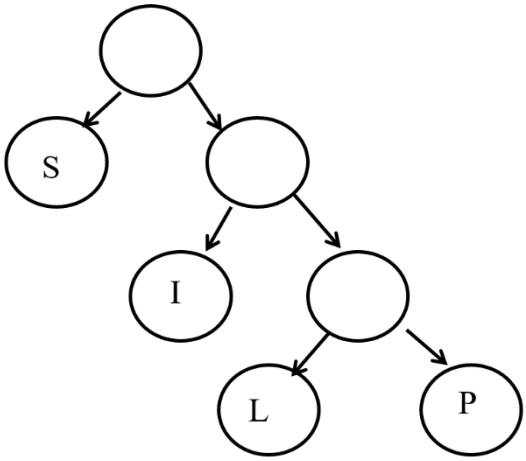
_____

P. The following method takes 40 seconds to complete when `data.length` = 1,000,000. What is the expected time to complete when `data.length` = 2,000,000. The code uses the Java `PriorityQueue` class. The elements in `data` are distinct and in random order.

_____

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

Q. Consider the following Huffman code tree.



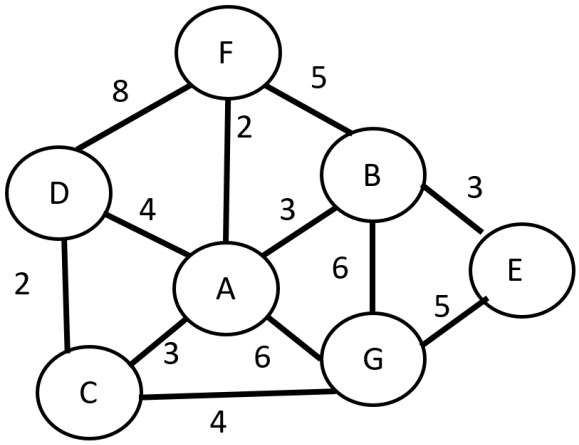
What does the following bit stream decode to using the Huffman code tree to the right?

110100111010111

_____

R. What is the cost of the minimum spanning tree of the following graph?

_____



S. What are the typical differences between a recursive solution for a problem and a dynamic programming solution for the same problem?

_____

T. Can dynamic programming be applied to all computing problems? Why or why not?

_____

Extra Credit - 1 point. What was your favorite assignment and why?

Assignment Number	Topic
1	Code camp, warm up assignment.
2	Creating a stand-alone class - Mathematical Matrices
3	Implementing a program with multiple classes, using ArrayLists - The Name Surfer
4	Using Data Structures - Evil Hangman
5	Linked lists
6	Recursion
7	More recursion, Anagram Solver
8	Using lists, creating data structures - Implementing a Set class
9	Binary Search Trees
10	Huffman Coding
11	Graph Algorithms

Number of favorite assignment: _____

Why? (in a sentence or two)

2. Linked Lists - (16 points) Complete the **`getSubList`** instance method for a linked list class. The method creates and returns new linked list with the elements from the calling list in the given range.

- **<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 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, has its next reference set to `null`.
- You may use the nested `Node` class and the `LinkedList314` constructor.
- **You may not use any other Java classes or methods.**

```
public class LinkedList314<E> {

     private Node<E> first;

     public LinkedList314() { } // default value correct

     private static class Node { // The nested Node class.
          private E data;
          private Node<E> next;
          private Node(E d) { data = d; }
     }
}
```

The `getSubList` method accepts two parameters, `startIndex` and `endIndex`. Just like the `substring` method from the `String` class, the `getSubList` method returns all values from the `startIndex` inclusive to the `endIndex` exclusive. Unlike the `substring` method, `startIndex` and `endIndex` may exceed the number of elements in the list. In this case, only the elements at valid indices are include in the resulting sublist.

Examples of calls to `getSubList(int startIndex, int endIndex)`. In these examples the elements of the list are `Integer` objects.

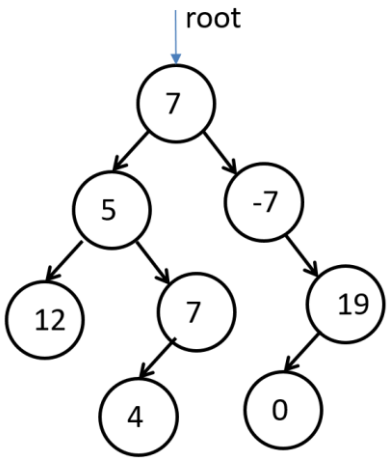
```
[7, 3, 5, 7].getSubList(0, 4) -> returns [7, 3, 5, 7]
[7, 3, 5, 7].getSubList(1, 3) -> returns [3, 5]
[7, 3, 5, 7].getSubList(2, 3) -> returns [5]
[7, 3, 5, 7].getSubList(2, 2) -> returns []
[7, 3, 5, 7].getSubList(2, 6) -> returns [5, 7] (valid values only)
[7, 3, 5, 7].getSubList(7, 10) -> returns [] (no valid values)
[7, 3, 5, 7].getSubList(0, 10) -> returns [7, 3, 5, 7] (valid values
only)
[].getSubList(0, 0) -> returns []
[].getSubList(0, 5) -> returns []
[10, -4].getSublist(2, 2) -> returns []
[10, -4].getSublist(1, 2) -> returns [-4]
[10, -4].getSublist(1, 10) -> returns [-4] (valid value only)
```

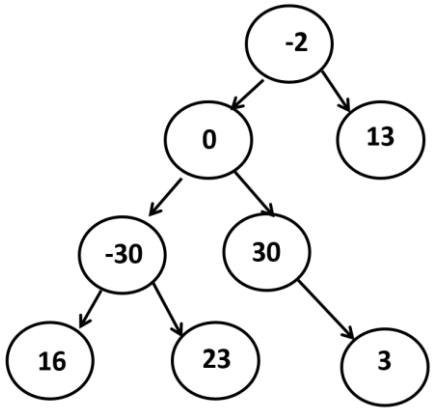
```
/* pre: startIndex >= 0. startIndex <= endIndex
   post: per the problem description. This list is not altered as a
     result of this method call. */
public LinkedList314<E> getSubList(int startIndex, int endIndex) {
```

3. Trees (14 points) - Implement an instance method for a binary tree class that returns the number of internal nodes that contain an odd number. (A number that is not a multiple of 2.) The method name is `numOddInternal`.

```
public class BinaryTree {

     private BNode root; // root == null iff tree is empty

     // nested BNode class
     private static class BNode {

          private int data;
          // left and right child store null if no child on that side
          private BNode left;
          private BNode right;
     }
}
```

Consider the following trees.





`t.numOddInternal() returns 5`
`for the above tree`
`Nodes with 7, 5, -7, 7, 19`

`t.numOddInternal() returns 0`
`for the above tree`

An empty tree always returns a result of 0.

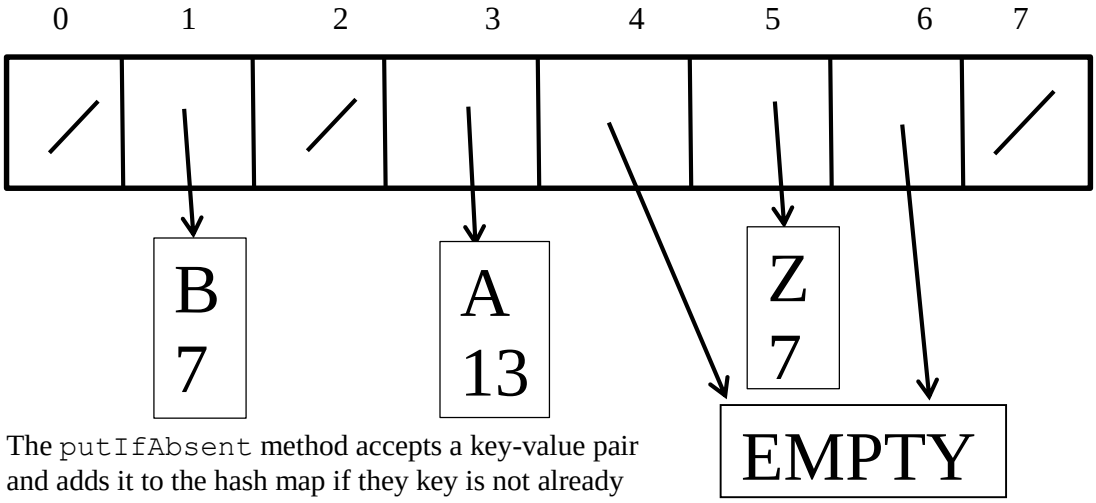
Do not create any new nodes or other data structures. You may use the `BNode` class, but do not use any other Java classes or methods.

```
/*   pre: none
     post: per the problem description.
     This BinaryTree is not altered as a result of this method call.*/
public int numOddInternal() {
```

4. Hash Tables - (16 points) - Complete the `putIfAbsent` instance method for a `HashMap` class. The hash map uses open addressing. Recall, in open addressing each element in the internal array stores a single element. Collisions are resolved by probing for an open spot.This implementation uses linear probing, increasing the index by one with each probe

Elements in the array that have never held a value store `null`. Elements that held a value at one time, but are now open (due to removing the value from the hash map) refer to a special object, named `EMPTY`.

Considering the following example. Assume the keys of this hash map are `String`s and the values are `Integer`s. Elements at indices 4 and 6 use to hold a key-value pair, but do not at the moment. Elements at indices 0, 2, and 7 have never referred to a key-value pair.



The `putIfAbsent` method accepts a key-value pair and adds it to the hash map if they key is not already present in the hash map. If the key is already present the hash map is unchanged.

For example, assume we try to put the key value pair of A-17. The key A is already present so the hash map is unchanged. Assume we try to add the key-value pair AA-13. Assume you determine the index for AA is 5. That is taken so probing goes to index 6. That spot is available and AA is not present in the hash map, so the key-value pair of AA is added at index 6.

```
public class HashMap<K, V> {
      private static final Object EMPTY = new Object();

      // resize array when load factor >= LOAD_LIMIT
      private final double LOAD_LIMIT;
      private int size; // number of key-value pairs in this map
      private Pair[] con; // hash map container

      /* Create new, larger container array, rehashes Pairs, and sets
         con equal to the new array. */
      private void resize()


      // Nested Pair class
      private static class Pair {
            private K key;
            private V value;
            private Pair(K k, V v) {key = k; value = v;}
      }
```

```
}
```

Complete the `putIfAbsent` method for `HashMap` class. You may use the `HashMap resize` method, the nested `Pair` class, the `hashCode` method from the `Object` class, and methods from the `Math` class.

```
/* pre: key != null, val != null
   post: per the problem description. Return true if this HashMap was altered
     as a result of this method, false otherwise. */
public boolean putIfAbsent(K key, V val) {
```

5. Encoding - 16 points. Error correction codes are a way of sending binary data to help overcome noise or errors in transmission.

A very simple technique, is to repeat each bit of data three times. When using this technique, a **0** becomes **000** and a **1** becomes **111**. This is known as a (3, 1) repetition code.

The receiver reads three bits at a time and examines the bits. If all three bits match, the intended bit is assumed to be error free. If there is a difference in the triplet, the intended bit is assumed to be the one that occurred 2 out of 3 times. For example:

Received Triplet	Assumed Value	Received Triplet	Assumed Value
000	**0**	**111**	**1**
001	**0**	**110**	**1**
010	**0**	**101**	**1**
100	**0**	**011**	**1**

Write a method that accepts a **BitInputStream** connected to a source encoded with the (3, 1) error correction repetition code described above. It writes out the decoded data via a **BitOutputStream**.

For example, given the following input bits (spaces added for clarity) the output would be as shown (again, spaces added for clarity):

000 111 110 111 111 000 010 011 -> 0 1 1 1 1 0 0 1

The method returns the number of triplets that were not consistent. **000** and **111** are the only two consistent triplets. The above example results in a return value of 3. (The three inconsistent triples in the input data are underlined for clairty.)

If the number of bits in the source the **BitInputStream** is connected to is not a multiple of 3, throw an `IOException`.

You may use the **BitInputStream** and **BitOutputStream** classes from the Huffman Encoding assignment and their **readBits** and **writeBits** methods:

```
public int readBits(int howManyBits) // from BitInputStream
```
Returns the number of bits requested as rightmost bits in returned value, returns -1 if not enough bits available to satisfy the request.

```
public void writeBits(int howManyBits, int value) // BitOutputStream
```
Write specified number of bits from `value`. The rightmost `howManyBits` of `value` are written.

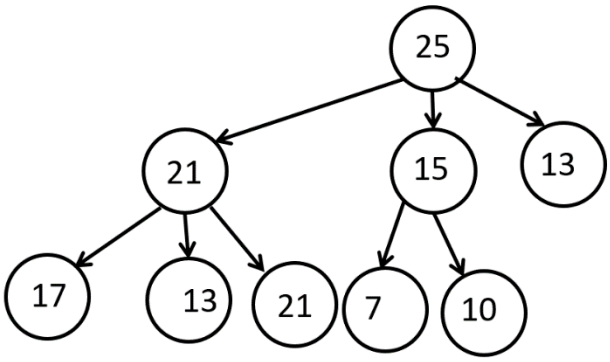
You may, but are not required, to create and use one array of `int`s if you believe it will help simplify your solution.

Do not use any other Java classes or methods.

```
/* in != null, out != null, in and out are already connected to the
     input and output files.
   post: Per the problem description. */
public int decodeTriplets(BitInputStream in, BitOutputStream out) {
```

6. Heaps - (18 points) - In class we presented binary heaps. A three heap use nodes with a maximum of three children instead of two.

For example:



In the three heap to the left, 25 is the root of the tree.

The heap is a **max heap,** as opposed to the min heap demonstrated in class.

In a max heap, the value in each node is **greater than or equal to** all the values in its descendant nodes.

In the three heap, each node has three children: left, center, and right child.

The heap is still a complete tree. Nodes are added left to right, top to bottom. In the example above, if a node was added it must be the right child of the node that contains 15 in order to maintain the complete property.

Complete the `add` method for a `Heap` class that uses a native array of `Comparable`s as its internal storage container.

The heap shall store the value of the root of the tree at index 1 in the array. The element at index 0 of the array is unused.

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

     private int size; // number of elements in heap
     private E[] con; // holds the elements of the heap

     // pre: val != null, post: val added to this max heap
     public void add(E val) {

}
```

You may use native arrays and the `compareTo` method from the `Comparable` interface.

Do not use any other Java classes or methods.

Complete the method on the next page.

```
// pre: val != null, post: val added to this max heap
public void add(E val) {
```