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

CS 314 – Final Exam – Spring 2014

Your Name_____

Your UTEID _____

Instructions:

1. There are **6** questions on this test. 95 points available. Scores will be scaled to 320 points.
2. You have 3 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. When answering coding questions your solution must be as efficient as possible given the constraints 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. What is returned by the method call `a(1)` ? _____

```
public int a(int x) {
     if(x > 20)
          return x;
     else
          return x + a(x * x + 1);
}
```

B. What is output by the method call `b(10)`? _____

```
public void b(int x) {
     if(x <= 0)
          System.out.print("!!");
     else {
          System.out.print(x);
          int temp = x / 2;
          if(temp <= 1)
               temp = 2;
          b(x - temp);
          System.out.print(temp);
     }
}
```

C. What is returned by the method call `c(7)` ? _____

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

D. What is the order (Big O) of method `c`? N = the parameter `n`. _____

E. The following values are inserted in the order shown to a binary search tree using the traditional, naïve insertion algorithm. Draw the resulting tree.

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

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

_____

G. The following values are inserted in the order shown to a min heap using the algorithm demonstrated in class. Draw the resulting min heap.

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

H. What is the order (Big O) of `methodH`? N = `data.length` _____

```
// pre: data.length >= 500
public ArrayList<Integer> methodH(int[] data) {
     ArrayList<Integer> result = new ArrayList<Integer>();

     for(int i = 0; i < 500; i++)
          for(int j = 0; j < 500; j++)
               result.add(data[i] * data[j]);

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

     return result;
}
```

I. What is the order (Big O) of `methodI`? N = `limit` _____

```
// pre: data != null
public Set<Integer> methodI(int limit) {
     Set<Integer> result = new TreeSet<Integer>();
     for(int i = 0; i < limit; i++)
          result.add(i * 10);

     for(int i = 0; i < limit / 2; i++)
          result.add(-i);

     return result;
}
```

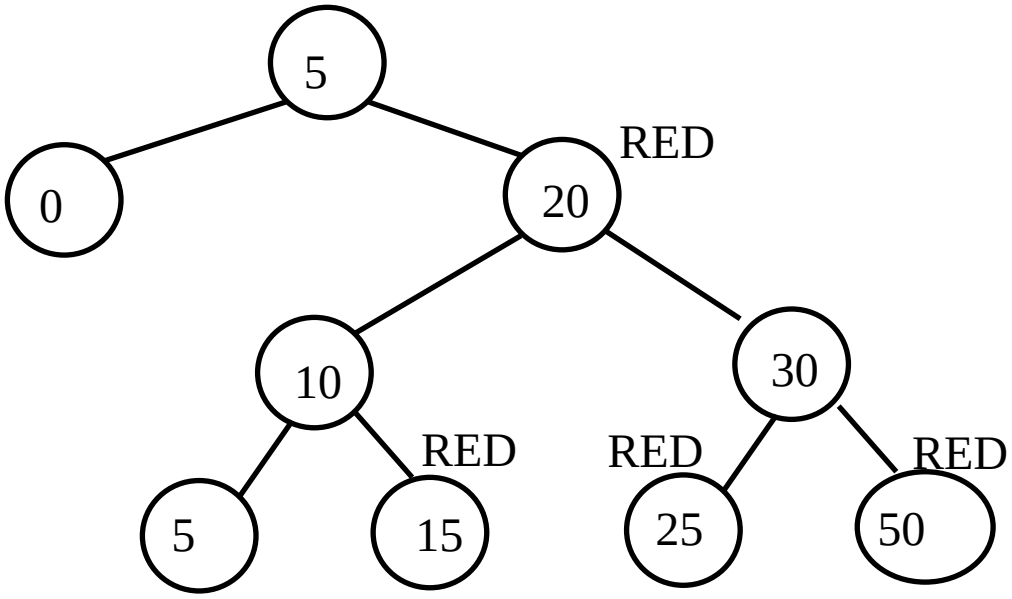
J. What is the order (Big O) of `methodI` if the first line of code is changed to the following?

```
Set<Integer> result = new HashSet<Integer>(); // change
```

N = `limit` _____

K. Consider the following tree. It is not Red-Black tree. Explain all the Red-Black tree requirements that are not met. (Nodes not listed as red are black.)

_____



L. The following code results in a `ClassCastException`. Why?

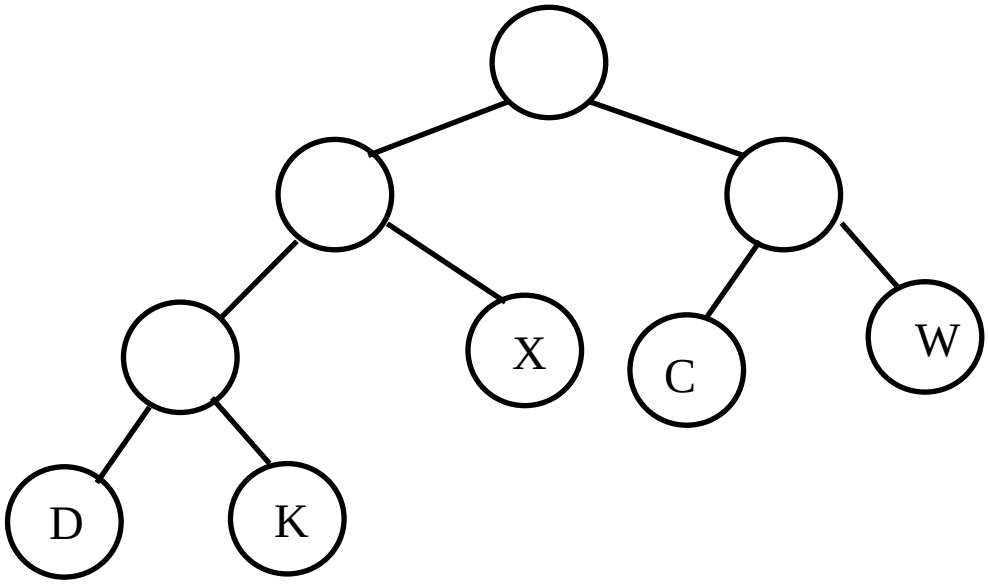
_____

```
ArrayList<Integer> list1 = new ArrayList<Integer>();
ArrayList<Integer> list2 = new ArrayList<Integer>();
list2.add(16);
TreeSet<ArrayList<Integer>> ts = new TreeSet<ArrayList<Integer>>();
ts.add(list2);
ts.add(list1);
```

M. What is output by the following code? _____

```
TreeMap<Integer, String> tm = new TreeMap<Integer, String>();
tm.put(9, "AA");
tm.put(5, "B");
tm.put(-2, "AA");
tm.put(5, "C");
for(int x : tm.keySet())
     System.out.print(x + " " + tm.get(x) + " ");
```

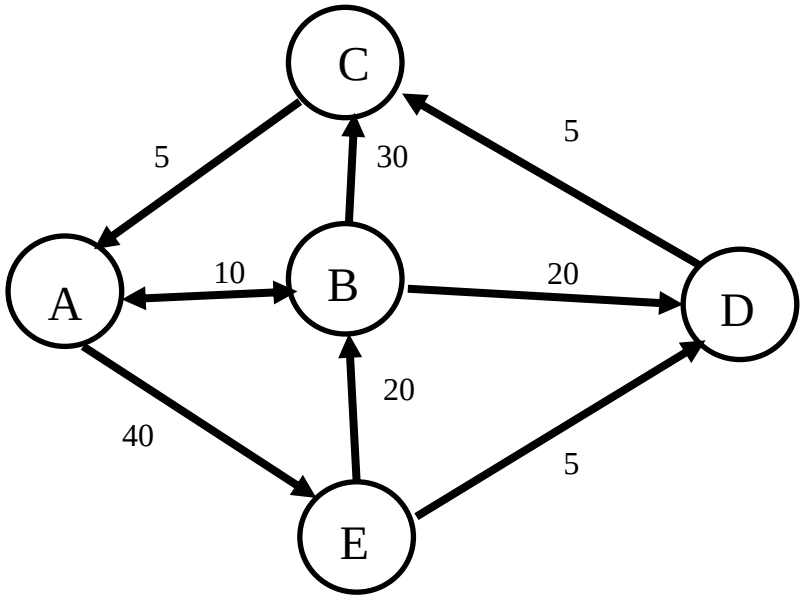
N. Given the following Huffman code tree, what does the given bit stream decode to?



Given bit stream (spaces added for clarity): `1 1 0 1 0 0 0 0 1 0 0 1 1 0`

_____

O. Consider the following weighted, directed Graph:



What is the cost of the lowest cost path from vertex E to vertex A? _____

P. Give the graph in part O, what is the central vertex of the graph? Recall from assignment 12, the central vertex is the one with the lowest average cost per shortest path to the other vertices in the graph.

_____

Q. What is the worst case order of method `q`? `list` is a Java `LinkedList` object.

```
public int q(LinkedList<Integer> list) {

     int result = 0;
     for(int i = 0; i < list.size(); i++) {
          int temp = 0;
          for(int j = i; j < list.size(); j++)
               temp += list.get(j);
          if(list.get(i) > temp)
               result++;
     }
     return result;
}
```

R. You need to encode 25 distinct colors using the fewest bits possible. What is the minimum number of bits you can use to encode the 25 colors?

_____

GO ON TO THE NEXT PAGE.

S. Consider the following class:

```
public class Property {
     private int price;

     public Property(int p) { price = p; }

     public String toString() {return "price: " + price; }

     public int hashCode() {  return 731 * price; }
}
```

Explain why the following code outputs `0` and not `2`.

```
Property p1 = new Property(500);
Property p2 = new Property(450);
Property p3 = new Property(500);
Property p4 = p2;
HashSet<Property> hs = new HashSet<Property>();
hs.add(p1);
hs.add(p2);
if(hs.contains(p3) && hs.contains(p4))
     System.out.println(2);
else
     System.out.println(0);
```

_____

T. What is output by method `t` when called? _____

```
public void t() {
     int[] coins = {3, 7, 11};
     int[] minCoins = new int[17];
     for(int i = 1; i < minCoins.length; i++)
          minCoins[i] = Integer.MAX_VALUE / 2;

     for(int i = 0; i < minCoins.length; i++) {
          for(int j = 0; j < coins.length; j++) {
               int c = coins[j];
               if(c <= i && minCoins[i - c] + 1 < minCoins[i])
                    minCoins[i] = minCoins[i - c] + 1;
          }
     }

     System.out.print(minCoins[16]);
```

```
}
```

2. linked lists - 15 points. Complete the `insertAfterNth` instance method for the `LinkedList314` class. The method inserts a new value in the list after the Nth occurrence of a given target value. If there are less than N occurrences of the target value in the list, then the list is unchanged.

- 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 only has a reference to the first node in the chained structure of nodes.
- When the list is empty, `first` is 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 any other Java classes or native arrays.**

```
public class LinkedList314<E> {

      private Node<E> first; // refers to first 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. of `insertAfterNth(E newValue, E target, int n)`

```
[].insertAfterNth(B, A, 1) ->  resulting list []

[A].insertAfterNth(B, A, 1)  ->  resulting list [A, B]

[A, A].insertAfterNth(B, A, 1)  ->  resulting list [A, B, A]

[A, A].insertAfterNth(B, A, 2)  ->  resulting list [A, A, B]

[A, A].insertAfterNth(B, A, 3)  ->  resulting list [A, A]

[B, B, B, C, G, X].insertAfterNth(B, A, 1)  ->  [B, B, B, C, G, X]

[B, B, B, C, G, X].insertAfterNth(B, A, 2)  ->  [B, B, B, C, G, X]

[B, B, B, C, G, X].insertAfterNth(B, B, 2)  ->  [B, B, B, B, C, G, X]

[B, B, B, C, G, X].insertAfterNth(B, C, 1)  ->  [B, B, B, C, B, G, X]
```

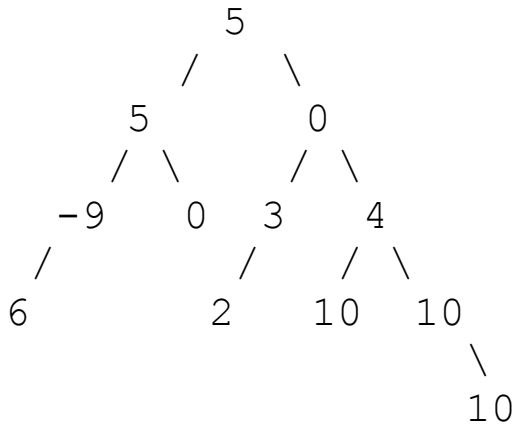
Complete the following method instance method of the `LinkedList314` class.

```
/* pre:  newValue != null, target != null, n > 0
   post: newValue inserted into this list after the nth occurrence of
     target

   If there are not n occurrences of target in this LinkedList314
   then this LinkedList314 is unchanged. */
public void insertAfterNth(E newValue, E target, int n) {
```

3. binary trees - 15 points. Consider a binary tree that contains integers. The binary tree is ***not*** a binary search tree. Write a method that returns `true` if there is a non-empty path from the overall root of a tree to a descendant node in which the sum of the data stored in the nodes in the path equals a target value. For this question the root is considered a descendent of itself. (A path can consist of just the root node.)

Consider the following tree and various target values



target of 10 -> true: root(5) -> left (5)
target of 5-> true: root(5)
target of 0 -> false
target of 27 -> false
target of 19 -> true: root(5) -> right(0) -> right(4)
-> left(10) = 5 + 0 + 4 + 10 = 19
target of 7 -> true: root(5) -> left(5) -> left(-9)
-> left(6) = 5 + 5 + (-9) + 6 = 7
target of 14 -> false

The path must start at the root and move to descendant nodes. The path cannot go back up the tree.

Use the following `BinaryNode` class:

```
public class BinaryNode {
     public int getData();
     public BinaryNode getLeft();
     public BinaryNode getRight();

     public void setData(int n);
     public void setLeft(BinaryNode left);
     public void setRight(BinaryNode right);
}
```

Use the following `BinaryTree` class:

```
public class BinaryTree {
     private BinaryNode root; // if tree is empty root == null
}
```

You may not use any other Java classes or methods other than the `BinaryNode` class.

You may not use any other methods from the `BinaryTree` class unless you implement them yourself as a part of your answer.

Complete the following instance method for the `BianryTree` class.

```
/* pre: none
   post: returns true if there is some non-empty path from the overall
root of a tree to a descendant node in which the sum of the data
stored in the nodes in the path equals target. For this method the
root is considered a descendant of itself. */
public boolean pathFromRootExists(int target) {
```

4. Huffman coding and maps - 15 points. In the Huffman coding project you decoded the compressed file by rebuilding the tree. An alternative to using the Huffman code tree is to use a map where the keys are the compressed codes and the value is the original integer value from the uncompressed file. This approach works because no code for a value is the prefix to any other code.

In the "Eerie eyes seen near lake." example the code for 'e' (ASCII value 101) is 10.

No other codes in that example start with 10. So if we have built up the code word "10" from the compressed file we know to write the integer 101 (the value for 'e') to output and then start over with a blank code.

Complete the following method to uncompress a Huffman encoded file.

```
public void uncompress(Map<String, Integer> codes, BitInputStream in,
                       BitOutputStream out) {
```

Recall the following method from the `BitInputStream` class:

`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.

Recall the following method from the `BitOutputStream` class:

`void`	**`writeBits`**`(int howManyBits, int value)` Write specified number of bits from value to a file.

When writing out integer values from the map of codes, write out the value using the `BITS_PER_WORD` constant. The method you are writing is in a class that implements the `IHuffConstants` interface so you can refer to the constant without a class or interface name.

You may use `String`s and `String` concatenation to build up codes from the input file.

You may use any methods from the `Map` interface.

Do not create any additional data structures other than `String`s in your method.

The `BitInputStream` is positioned at the start of the input file.

The `BitOutputStream` is already connected to the output file.

Stop writing and close the `BitOutputStream` when the code for the `PSEUDO_EOF` is read in from the `BitInputStream`.

The keys in the map of codes are String representations of the compressed binary codes and the values are the original integer value. Given the "Eerie eyes seen near lake." example the map contains a key "10" and the associated value is 101, the ASCII code for 'e'. Likewise the map contains a key "11111" and the

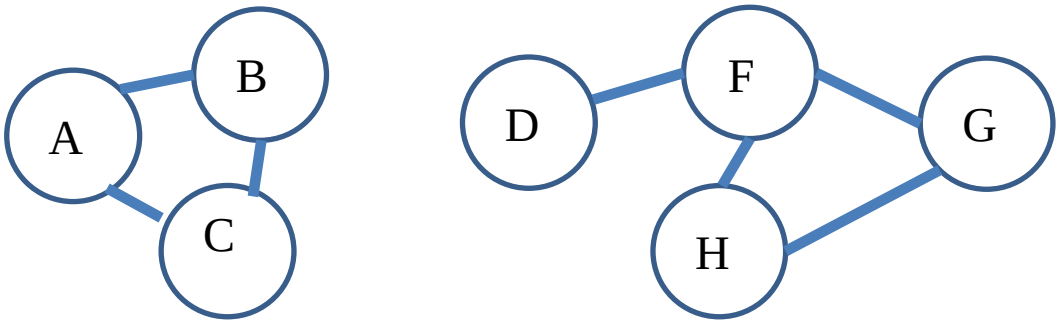
associated value is 69, the ASCII code for 'E' along with the codes for all the other values in the original file. The map will also contain the code for the `PSEUDO_EOF` value.

```
/* pre: 1. The keys in codes are String representations of the
compressed binary codes in the input file. The associated values are
the uncompressed value to be written to the output file.
2. in is positioned at the start of the input file.
3. out is positioned at the start of the output file.
post: per the question description. */
public void uncompress(Map<String, Integer> codes, BitInputStream in,
                       BitOutputStream out) {
```

5. graphs - 15 points. Write a method to determine if an undirected graph is *connected*. Two vertices, A and B, are connected if there is a path in the graph from A to B. In the example below D and H are connected because there is a path between them even though no edge exists between D and H.

The graph itself is connected if every pair of vertices in the graph is connected.

Consider the graph from question 1.O. If the edges were undirected the graph would be connected. The following undirected graph is not connected. There are no paths from the part of the graph with vertices A, B, and C to the part of the graph with vertices D, F, G, and H.



Use the `Graph` class from assignment 12 and complete a method that returns `true` if the `Graph` is connected, `false` otherwise.

In order to represent an undirected graph, if `Vertex` A contains an edge in its adjacency list to `Vertex` B, then `Vertex` B will have an edge in its adjacency list back to `Vertex` A.

Important: You may use the `Map`, `List`, `Vertex`, and `Edge` classes and objects of those types that already exist in the `Graph`, `Vertex`, and `Edge` classes, but you MAY NOT create any new objects or data structures in your solution other than `Iterators`.

Recall these parts of the `Graph`, `Vertex`, and `Edge` classes:

```
public class Graph {

    // The vertices in the graph.
    private Map<String, Vertex> vertices;

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

    private static class Edge {
        private Vertex dest;
        private double cost;
    }

    // go on to the next page
```

```
// pre: the Graph is not empty
public boolean isConnected() {
    clearAll(); // sets all Vertex objects' scratch value to 0
    Vertex start = vertices.get(vertices.keySet().iterator().next());
    int[] numVisited = {0};
    return connectedHelper(start, numVisited);
}

// complete the following method
private boolean connectedHelper(Vertex currentVertex, int numVisited)
```

6. data structures - 15 points. Write the dequeue method for a priority queue that uses a binary search tree as the internal storage container. The front element of the queue is the smallest element based on the `Comparable` interface.

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

     private BSTNode<E> root; // root == null if tree empty

     // no other instance variables

     private static class BSTNode<E extends Comparable<? super E>> {
          // recall you can access these directly from the
          // TreePriorityQueue class
          private E data;
          private BSTNode<E> left;
          private BSTNode<E> right;
     }
```

Important: You may not use recursion in your solution.
You may not use any other Java classes or methods besides the `BSTNode` class.

There are no precoditions to the method. If the priority queue is empty, then the method returns `null`.

Complete the instance method for the `TreePriorityQueue` class on the next page.

```
/* pre: none
   post: remove and return the front (smallest) element in this
          priority queue.
         If the priority queue is empty this method returns null.
*/
public E dequeue {
```