Points off	1	2A	2B	2C	3	4A	4B	5	Total off	Net Score

CS 314 – Final – Fall 2012

Your Name___________________________________

Your UTEID ___________________________________

Instructions:

1. There are **<u>5</u>** questions on this exam. The raw point total on the exam is 110.
2. You have 3 hours to complete the exam.
3. You may not use a calculator or any other electronic devices while taking the exam.
4. When writing methods assume the preconditions of the method are met.
5. When writing methods you may add helper methods if you wish.
6. When answering coding questions ensure you follow the restrictions of the question.
7. When you complete the test show the proctor your UTID. Give them the test and any scratch paper. Please leave the room quietly.

1. (2 points each, 40 points total) Short answer. Place you answers on the attached answer sheets.
   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 should 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("TURING")`?

```
public int a(String st) {
     if(st.length() <= 1)
          return st.length() * 2;
     return st.length() + a(st.substring(1)) +
               a(st.substring(2));
}
```

B. What is the order (Big O) of method `b`? N = `data.length`. Assume in the initial call to method `b` the variable `w` stores `0` and the variable `h` stores `data.length - 1`.

```
public int b(int[] data, int w, int h) {
     if(w == h)
          return data[w] % 10;
     else if(w > h)
          return 0;
     else {
          int mid = (w + h) / 2;
          return data[mid] % 10 + b(data, w, mid - 1) +
                    b(data, mid + 1, h);
     }
}
```

C. What is output by the following code?

```
int count = 0;
int total = 0;
int n = 2000;
for(int i = 0; i < n; i++) {
     for(int j = i; j < n; j++) {
          count++;
          total += count + i * j;
     }
}
System.out.print(count);
```

D. Consider the following method from the `Collections` class in the Java standard library.

```
  public static int frequency(Collection<?> c, Object o)
```

Returns the number of elements in the specified collection equal to the specified object.

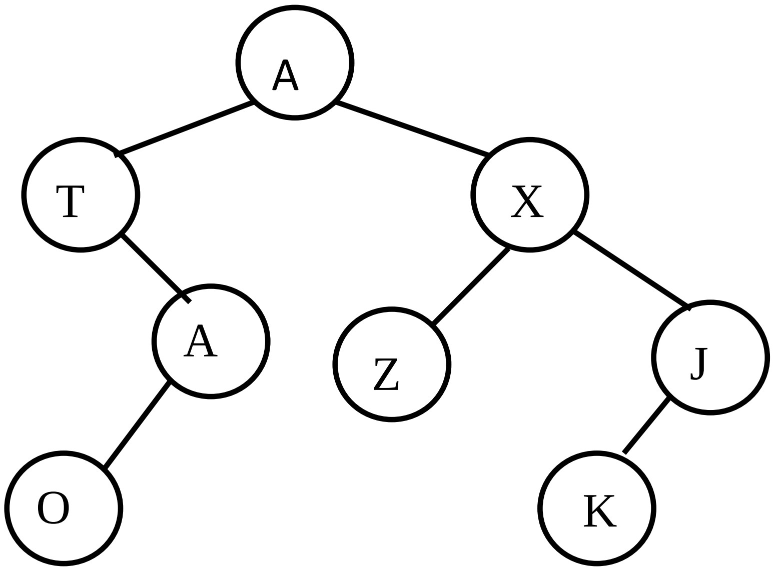
How can the `frequency` method return the correct answer without knowing at compile time what type of `Collection c` is (ArrayList? LinkedList? Hashset? …) and what type of object `o` is? Be specific.

E. The following values are inserted one at a time into a binary search tree using the traditional, naïve algorithm. Draw the resulting tree. `-5, 10, 7, -5, 13, 0`

F. On the answer sheet fill in integer values between 0 and 10 inclusive for the 5 nodes so the resulting tree is a max heap.

G. What is the worst-case order (Big O) to add an element to a hashtable that already contains N elements? The hashtable uses closed address hashing.

For H and I consider the following binary tree:



H. What is the result of a preorder traversal of the tree?

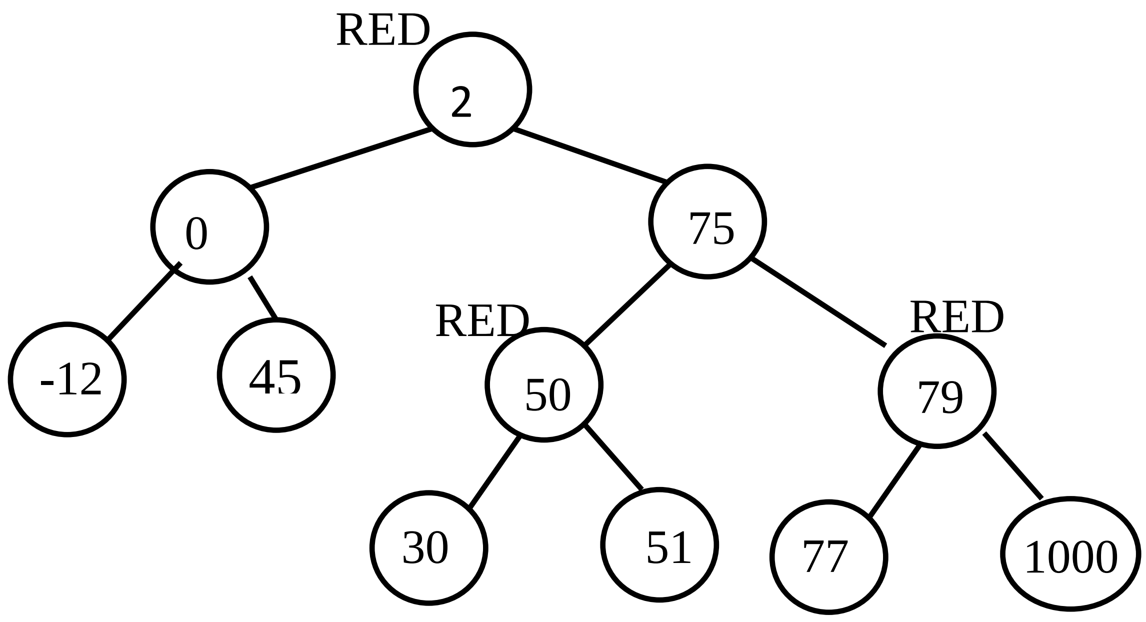
I. What is the result of a postorder traversal of the tree?

J. What is the order (Big O) of method `j` if `set` is a `TreeSet` from the Java standard library? What is the order (Big O) of method `j` if `set` is a `HashSet` from the Java standard library? N = `num`. Assume the `nextInt` method is O(1).

```
// pre: set != null, set.size() == 0
public void j(Set<Integer> set, int num, Random r) {
     for(int i = 0; i < num; i++) {
          set.add(r.nextInt());
          set.remove(r.nextint());
          set.add(r.nextint());
     }
}
```

K. Suppose you want to encode 60 different names using a fixed length encoding scheme. Represent each name with a unique combination of bits that is the same length as all of the other name codes. No other information about the name is stored. You simply need a unique binary code for each name. What is the minimum number of bits per name required to have a unique code for each name?

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



M. What is output by the following client code?

```
ArrayList<Integer> list = new ArrayList<Integer>();
list.add(12);
method_l(list);
System.out.println(list);

public static void method_l(ArrayList<Integer> arl) {
     arl.add(7);
     arl.add(12);
     arl.add(5);
     Collections.sort(arl);
     arl = new ArrayList<Integer>();
     arl.add(9);
     arl.add(-5);
}
```

N. A method that sorts arrays of `int`s uses the mergesort algorithm.

It takes the method 20 seconds to sort an array of 1,000,000 distinct `int`s in descending order.

What is the expected run time when the method sorts an array of 2,000,000 distinct `int`s in descending order?

O. Consider the following potential `hashCode` method for an array based list class. The list does not allow `null`s to be stored as elements of the list.

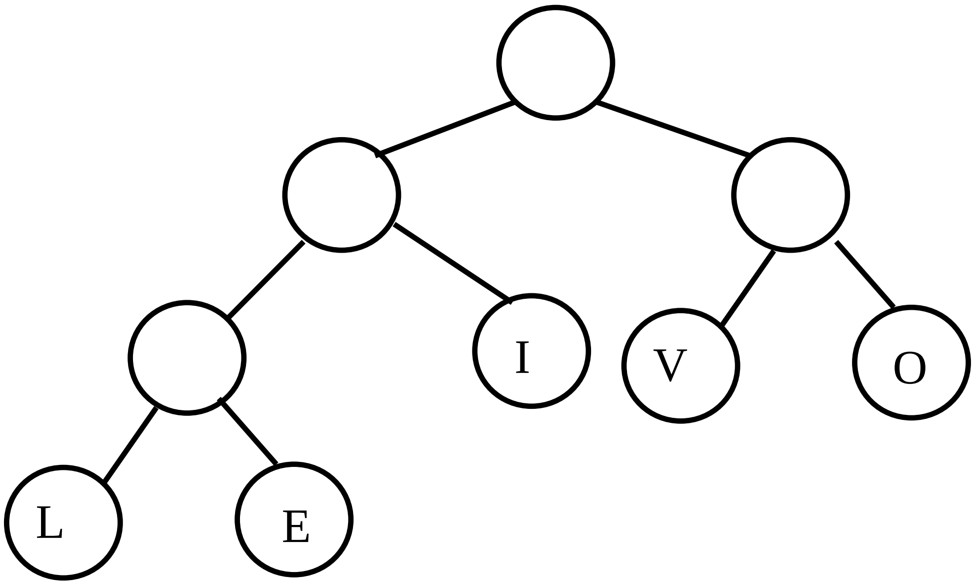
```
public class GenericList<E> {

     private int size;
     private E[] con;
     private int hashcode;

     public int hashCode() {
          if(hashcode != 0) {
               for(int i = 0; i < size; i++) {
                    hashcode += con[i].hashCode();
               }
          }
          return hashcode;
     }
}
```

What is the most significant problem of this `hashCode` method for the `GenericList` class?

P. The following values are added one at time in the order shown to a min heap using the algorithm demonstrated in class. Draw the resulting min heap.

```
12, 6, 9, 11, 15
```

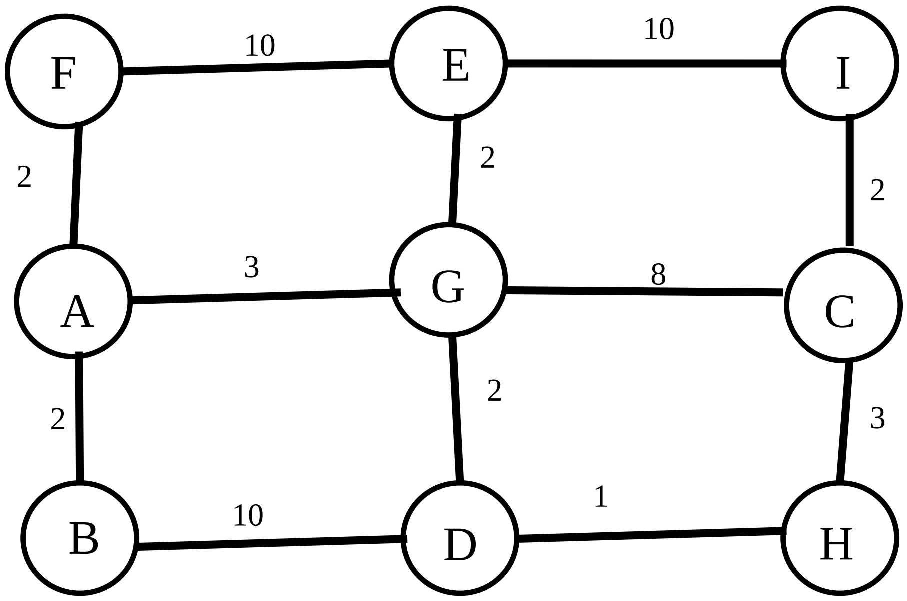
Q. Consider the following Huffman code tree.



Given the above tree what is the result of encoding `OLIVE`? List the correct sequence of 0s and 1s.

R. Are most real world systems modeled by graphs spares or dense?
What is the biggest disadvantage of implementing a spares graph with an adjacency matrix?

S. Consider the following undirected, weighted graph. What is the cost of the lowest cost path from vertex B to vertex I?



T. In the Java standard library the `LinkedList` class implements the `Queue` interface but the `ArrayList` class does not? Using Big O, explain why the `ArrayList` class does not implement the `Queue` interface.

EXTRA CREDIT (1 POINT)

Two UTCS Faculty Members have won the Turing Award, the "Nobel Prize" of Computer Science. What are their last names? (Put answer on the answer sheet.)

2. (Trees - 20 points) 2.A: (5 points) Write an instance method for a `HuffmanCodeTree` class from assignment 11 that determines if the path defined by a given code is a valid path from the root of the tree to a leaf or not.

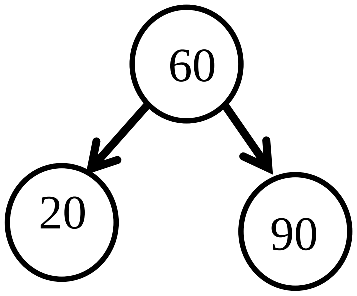
You may use methods from the `String` class and fields from the `HuffNode` class, but no other classes or methods. Your solution shall be as efficient as possible given the constraints.

Examples using the Huffman code tree from question 1.Q:

```
validPath("00") -> returns false, not a valid path from root to a leaf
validPath("10") -> returns true
validPath("000") -> returns true
validPath("101001") -> returns false
validPath("1") -> returns false

public class HuffmanCodeTree {

     private HuffNode root; // never equals null

     private static class HuffNode {
          public int chunk;
          public int freq;
          public HuffNode left, right;
     }

     // complete the following method
     // pre: code != null, code.length() >= 1
     // code only contains '0's and '1's
     public boolean validPath(String code) {
```

2.B: (5 points) Write an instance method for an `IntBST` class that determines the number of elements in the tree that are evenly divisible (no remainder) by a given integer. The `IntBST` only stores `int`s. Hint: add a helper method.

Examples based on tree to the right:



```
numDivisible(20) -> 2
numDivisible(-10) -> 3
numDivisible(1) -> 3
numDivisible(7) -> 0
```

You may not use any other classes or methods except the the `BSTNode` class.
As always, you may add your own helper methods.
Your solution shall be as efficient as possible given the constraints.

```
public class IntBST {

      private BSTNode root; // root == null if size == 0
      private int size;

      // recall outer class can access private fields in nested class
      private static class BSTNode {
            private int val;
            private BSTNode left, right;
      }

      // pre: num != 0
      public int numDivisible(int num) {
```

2.C: (10 Points) Recall the red rule for a red black tree states the children of a red node, if they exist, must be black. Write an instance method that verifies if the red rule is met by all the red nodes in a red black tree.

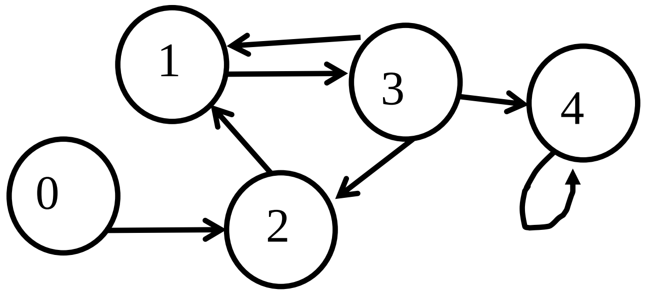
You may not use any other classes or methods except the `RBNode` class.
As always, you may add your own helper methods. (Hint, you want a helper method.)
Your solution shall be as efficient as possible given the constraints.

```
public class RedBlackTree<E extends Comparable<E>> {

     private RBNode<E> root; // if size == 0, root == null
     private int size;

     // recall outer class can access private fields in nested class
     private static class RBNode<E> {

          private E data;

          // true if node is red, false if black
          private boolean isRedNode;

          private RBNode<E> left, right;
     }

     // Complete the following method.
     // returns true if red rule is met by this RedBlackTree,
     // false otherwise
     private boolean redRuleMet() {
```

COMPLETE THE `redRuleMet` METHOD ON THE NEXT PAGE.

```
// returns true if red rule is met by this RedBlackTree,
// false otherwise
private boolean redRuleMet() {
```

3. (Graphs - 15 points) Implement a method in a `Graph` class that determines if a path exists from one vertex to another. The Graph is directed and unweighted. The Graph uses an adjacency matrix of `boolean`s to represent the edges between vertices. Consider the example below. The graph below on the left is represented by adjacency matrix on the right. t = true, edge exists from vertex with row number to vertex with column number. The vertices are designated with an int from `0` to (`N - 1)` where N is the number of vertices in the graph.



	0	1	2	3	4
0	f	f	t	f	f
1	f	f	f	t	f
2	f	t	f	f	f
3	f	t	t	f	t
4	f	f	f	f	t

You may use a single `HashSet` of `Integers` in your solution, but no other classes or methods.
As always, you may write your own helper methods if you wish.
Your solution shall be as efficient as possible given the constraints.

Example of calls to `pathExists`.

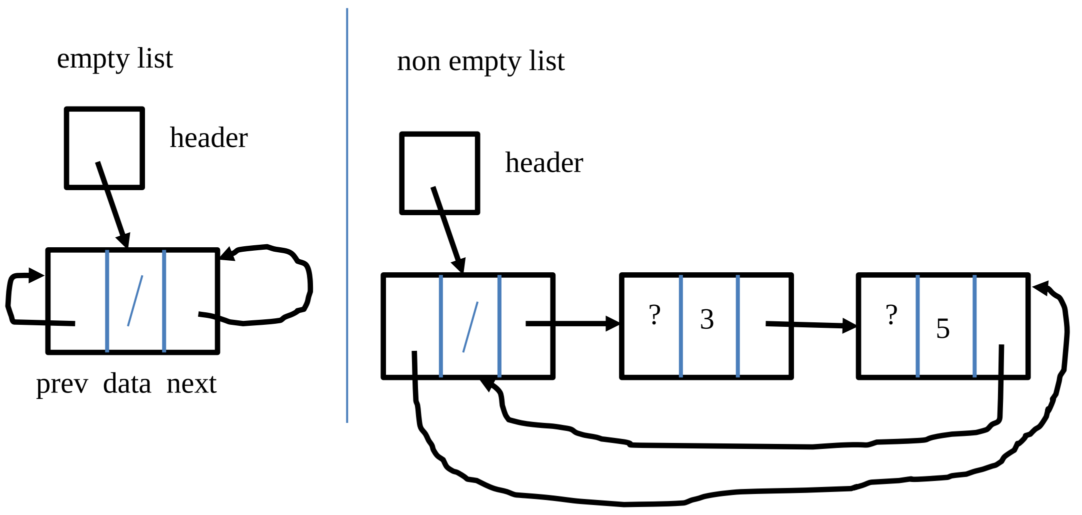
```
pathExists(0, 0) -> false                pathExists(0, 1) -> true
pathExists(0, 4) -> true                 pathExists(4, 4) -> true
pathExists(4, 1) -> false                pathExists(3, 4) -> true
pathExists(4, 3) -> false                pathExists(2, 0) -> false
```

```
public class Graph {

     // a square matrix with no extra capacity
     // if graph is empty, adjMat = null
     private boolean[][] adjMat;

     public int numVertices() {
          if(adjMat == null)
               return 0;
          else
               return adjMat.length;
     }

     // complete the following method
     // pre: 0 <= start < numVertices(), 0 <= dest < numVertices()
     // post: return true if a path exists from start to dest
     public boolean pathExists(int start, int dest) {

     // complete this method on the next page
// pre: 0 <= start < numVertices(), 0 <= dest < numVertices()
// post: return true if a path exists from start to dest,
// false otherwise
public boolean pathExists(int start, int dest) {
```

4. (Linked Lists - 20 points)

A (10 points): Write an instance method that finds and repairs any incorrect previous references in a double linked list. The doubly linked list has the following properties:

- The list uses doubly linked nodes that store one piece of data, a reference to the next node in the list, and a reference to the previous node in the list.
- The list maintains a reference to a header node. The next reference in the header node refers to the first node in the list with data. The previous reference in the header node refers to the last node in the list with data.
- If the list is empty, the header's next and previous references refer to the same node as header.
- The list does not maintain a size instance variable.



`

Your method shall find and repair all previous references in the linked list that are not set correctly.

You may not use any other classes (including arrays) or methods except the `Node` class.
Your solution shall be as efficient as possible given the constraints.

```
public class LinkedList<E> {

     private Node<E> header; // never null, always refers to same node

     private class static Node<E> {
          private Node<E> prev, next;
          private E data;
     }

     // pre: all next references set correctly, header node's
     // previous reference set correctly
     // post: return true if any references were initially
     //   incorrect. All incorrect references repaired.
     private boolean repairPreviousReferences() {
```

```
// pre: all next references set correctly, header node's
// previous reference set correctly
// post: return true if any references were initially
//   incorrect. All incorrect references repaired.
private boolean repairPreviousReferences() {
```

4.B (10 points) Implement an instance method for a singly linked list of `int`s that adds a node to the end of the list that is sum of the elements initially in the list.

- The list uses singly linked nodes
- The list has references to the first and last node in the list. There is no size variable
- If the list is empty first and last are both set to `null`.
- The last node in the list (if it exists) has its next reference set to `null`.

Example of method:

```
[].addSumEnd -> [0]
[0].addSumEnd -> [0, 0]
[1, 4, 1].addSumEnd -> [1, 4, 1, 6]
[1, 4, 1, 6].addSumEnd -> [1, 4, 1, 6, 12]
```

You may not use any other classes (including arrays) or methods except the `Node` class.
Your solution shall be as efficient as possible given the constraints.

```
public class IntList {
      private Node first, last;

      private static class Node {
            private int data;
            private Node next;

            public Node(int data, Node next)
      }

      // complete the following method. There are no preconditions.
      public void addSumEnd() {
```

5. (Other Data Structures - Deques, 15 points) A *double ended queue* or *deque* is a data structure that allows elements to be added, accessed, and removed from either end of the structure, but nowhere else.

Examples of calls to `addFront` and `addBack` on a deque.

```
[].addFront(3) -> [3]
[3].addFront(7) -> [7, 3]
[7, 3].addBack(3) -> [7, 3, 3]
[7, 3, 3].addFront(5) -> [5, 7, 3, 3]
```

Implement the `addFront` and `addBack` methods for a `BufferedDeque` class that uses a native array as its internal storage container. The capacity of the `Deque` is specified in the constructor and cannot be altered.

You may not use any other classes or methods except native arrays.
Your solution shall be as efficient as possible given the constraints.

```
public class BufferedDeque<E> {

     private int first; // refers to index of front element
     private int last; // refers to index of last element
     private int size; // number of elements in this BufferedDeque
     private E[] con; // internal storage container

     // pre: capacity > 0
     // you may NOT alter the constructor
     public BufferedDeque(int capacity) {
          size = 0;
          first = 0;
          last = -1;
          con = (E[]) (new Object[capacity]);
     }

     // You may not alter these methods.
     // Your implementations of addFront and addBack must not
     // cause these methods to generate any kind of error.

     // pre: size > 0
     public E getFirst() { return con[first]; }
     public E getLast() { return con[last]; }
```

(3 points) What should the precondition of the `addFront` and `addBack` methods be? What action should the methods take if the precondition is violated?

Implement the `addFront` and `addBack` instance methods for the `BufferedDeque` class. Do not check your precondition or write any code to deal with the violation of your precondition.

```
public void addFront(E val) {
```

```
public void addBack(E val) {
```

(2 points) How would you have to alter your methods if the deque was not buffered? In other words, what would have to change if there were no limit on the number of items that could be added to the deque?

(2 points) If the dequeue was not buffered and you were allowed to use a data structure other than a native array what would it be? Use Big O as part of your justification

Question 1 Answer Sheet.

Name____ UTEID ________________________

A.

____

B.

____

C.

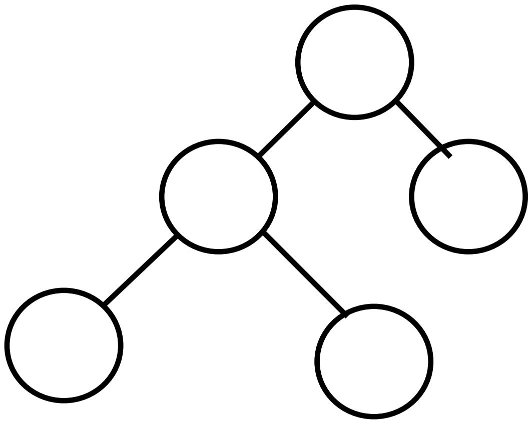
____

D.

____

E.

____



F.

G.

H.

I.

J. `TreeSet:`

`HashSet:`

K.

L.

M.

N. ______________________________

O. ______________________________

P. ______________________________

Q. ______________________________

Most real world graphs are: sparse dense (circle one)

Biggest disadvantage of using an adjacency matrix for a sparse graph:

R. ______________________________

S. ______________________________

T. ______________________________

One current and one former UTCS faculty members won the Turing Award, the "Nobel Prize" of Computer Science. What are their last names?

____