Question 1. The analysis of Ethernet performance in the paper by Metcalfe and Boggs makes the simplifying assumption that each computer will transmit during a given slot with probability $\frac{1}{N}$ where N is the number of stations trying to transmit. In reality, we know that the probability that a station will transmit depends on the number of collisions it has experienced rather than on N. If a computer has experienced K collisions, it chooses a random delay value from the set $\{0, 1S, 2S, \ldots, (2^K - 1)S\}$. Therefore, the probability it transmits in any given slot is $\frac{1}{2^K}$.

Metcalfe and Boggs make this simplifying assumption in order to obtain a formula that approximates the expected number of slots that will be wasted either by being left idle or because a collision occurs during the slot. Any formula for the expected number of wasted slots that accurately reflects the actual dynamics of the exponential backoff algorithm would be extremely complicated. To appreciate this, we would like you to determine a formula for the expected number of slots that would be wasted in a single round of a very specific collision resolution scenario.

Suppose that two computers, A and B, attempt to transmit simultaneously on an idle Ethernet. The process of collision resolution in which they will participate can be divided into "rounds." In round 0, both stations transmit immediately and collide. In round 1, they both randomly choose a delay from the set $\{0, S\}$ and transmit after this delay. In this round, the probability of a collision is $\frac{1}{2}$. If they collide again in round 1, then they engage in another round (round 2) in which they choose delays from the set $\{0, S, 2S, 3S\}$. The total number of slots wasted during this process will be the sum of the slots wasted in each round. It is obvious that the maximum number of slots wasted in round K is 2^K. The actual number of slots wasted in a given round, however, depends on the random delays selected.

For example, in round 1, there are four equally likely outcomes. Station A may choose delay 0 while station B chooses S. In this case, A will immediately transmit successfully and no slots will be wasted.. Similarly, if B chooses delay 0 and A chooses S, no slots will be wasted. On the other hand, if both A and B choose 0, they will both transmit immediately and collide. After this collision, they will both move on to round 2 immediately, so round 1 will only waste 1 slot. On the other hand, if the both choose a delay of S, one slot will be wasted by being left idle and another slot will be wasted due to collision. To compute the expected number of slots wasted in round 1, we simply sum the number of slots wasted in each of the four possible outcomes time the probability of each outcome ($\frac{1}{4}$) to obtain:

$$0 \times \frac{1}{4} + 0 \times \frac{1}{4} + 1 \times \frac{1}{4} + 2 \times \frac{1}{4} = \frac{3}{4}$$

That is, the expected number of slots that will be wasted during round 1 will be $\frac{3}{4}$.

Assuming that A and B do collide during round 1 and move on to round 2, determine the expected number of slots that will be wasted during round 2. Justify your answer.

Question 2. The code for most of the URLList class we discussed in class is shown below.

```
// Class used to hold a list of Strings interpreted as web site addresses.
public class URLList  {

  private boolean isEmpty = false;     // true if nothing in list
  private String firstSite;       // The first web site in the list
  private URLList restOfSites;  // The rest of the list of web sites


  // Create an empty list
  public URLList( ) {
      isEmpty = true;
  }


  // Create a larger list from a new website and an existing list
  public URLList( String newSite, URLList existingList ) {
      firstSite = newSite;
      restOfSites = existingList;
  }
```

```
  // determines whether the collection contains a given entry
  public boolean contains( String site ) {
    if ( isEmpty ) {
        return false;
    } else if ( firstSite.equals( site ) ) {
        return true;
    } else {
        return restOfSites.contains( site );
    }
  }

  .....   // a few other methods we do not care about in this question ...

}
```

For this question, we want you to consider several alternative definitions for contains that might have been used in this class. For each alternative, indicate whether it would i) work fine, ii) work, but not as efficiently as the original, iii) compile but not work correctly, or iv) not even compile. Briefly explain your answer. Hint: Although URLList is certainly an example of a recursive class, dont panic. This question is more about if statements than about recursive classes.

a)
```
public boolean contains( String site ) {
  if ( firstSite.equals( site ) ) {
      return true;
  } else if ( isEmpty ) {
      return false;
  } else {
      return restOfSites.contains( site );
  }
}
```

b)
```
public boolean contains( String site ) {
  if ( isEmpty ) {
      return false;
  } else if ( ! restOfSites.contains( site )  ) {
      return firstSite.equals( site );
  } else {
      return true;
  }
}
```

c)
```
public boolean contains( String site ) {
  return !isEmpty && ( firstSite.equals(site) || restOfSites.contains(site) );
}
```

d)
```
public boolean contains( String site ) {
  if ( isEmpty ) {
      return false;
  } else if ( ! firstSite.equals( site ) ) {
      return restOfSites.contains( site );
  }
}
```

Question 3. On the following two pages, you will find definitions of two classes that could be used in the Java IM clients you constructed in labs 4 and 5. The first is a definition of a ChatWindow class. It should look very similar

to what you used in lab. The second class is named TOCSendIMPacket. It is not quite complete. It is missing the declarations of its instance variables, local variables, and formal parameters.

For this problem, we want you to add the missing declarations. We have placed shaded bubbles at all the points in the code where the missing declarations could go. You will not need to fill in all of the bubbles we have provided, but you should not need to place declarations anywhere outside these bubbles.

Each name should be declared as locally as possible. That is, if the program would work if a given name was declared either as a local variable or an instance variable, then you should declare it as a local variable. The names for which declarations must be provided are:

- destination
- buddysName
- message
- messageInHTML
- lessPos
- greaterPos
- original
- updated

You should be able to complete this problem without fully understanding what the classes do, but to help you out, we provide the following explanation.

The TOCSendIMPacket constructor expects parameter values that specify the essential details required to send an IM message, the screen name of the person the message should be sent to and the text of the message. The toString method associated with the class then returns the packet that should be sent to the AOL server. That is, if you created a TOCSendIMPacket by saying

```
packet = new TOCSendIMPacket( thommurtagh, Hi Tom );
```

you could later sent the appropriate text to the server by saying

```
toServer.out.printPacket( packet.toString() );
```

Real IM clients encode the messages they send in HTML. The programs you created in lab did not do this. The TOCSendIMPacket fixes this by adding some simple HTML to the messages being sent. It adds ¡html¿¡body¿ to the beginning of each message and ¡/body¿¡/html¿ to the end. In addition, if any less than signs were included as part of the original message, they are replaced by the characters HTML uses to encode less than signs, <. Similarly, any greater than signs are replaced by >. As a result, if you created a TOCSendIMPacket by saying

```
packet = new TOCSendIMPacket( thommurtagh, Hi >Tom< );
```

and then later said

```
String text = packet.toString();
```

the variable text would be associated with the String:

```
toc2_send_im thommurtagh <html><body>Hi &gt;Tom&lt;</body></html>
```

```
import TOCtools.*;
import squint.*;
import javax.swing.*;

public class ChatWindow extends GUIManager {
    private final int WINDOW_WIDTH = 550, WINDOW_HEIGHT = 350;

    // Dimensions of GUI components for text
    private final int TEXT_WIDTH = 40;
    private final int TEXT_AREA_HEIGHT = 15;

    // Area used to display IM messages (and other messages send by server)
    private JTextArea dialogue = new JTextArea( TEXT_AREA_HEIGHT, TEXT_WIDTH );

    // The connection to the server
    private FLAPConnection toAOL;

    // Field used to enter messages
    private JTextField message = new JTextField( TEXT_WIDTH - 10 );

    // Buddy's name
    private String buddy;

    // User's name
    private String user;


    // Create a new chat window
    public ChatWindow(String screenName, String buddyName, FLAPConnection connection ) {
        this.createWindow( WINDOW_WIDTH, WINDOW_HEIGHT, buddyName );

        buddy = buddyName;
        user = screenName;
        toAOL = connection;

        dialogue.setEditable( false );
        contentPane.add( new JScrollPane( dialogue ) );
        contentPane.add( new JLabel( "Message: " ) );
        contentPane.add( message );
    }

    // Send a message to the window's buddy
    public void textEntered( ) {
        TOCServerPacket packet;
        dialogue.append( user + ": " + message.getText() + "\n" );
        packet = new TOCSendIMPacket( buddy, message.getText() );
        toAOL.sendFlapData( packet.toString() );
    }

    // Display a message received from the buddy
    public void messageReceived( String message ) {
        dialogue.append( buddy + ": " + message + "\n" );
    }
}
```

```
public class TOCSendIMPacket {



 // Create a new packet
    public TOCSendIMPacket(                                                    ) {


        destination = buddysName;
        messageInHTML = addHTML( message );
    }

    // Return the packet in text form
    public String toString(                                                    ) {


        return "toc2_send_im " + destination + " \"" + messageInHTML + "\"";
    }

    // Place HTML tags around the message body and replace and < and > in the orignal
    // message with the HTML sequences used to encode them ( &lt; and &gt;).
    private String addHTML(                                                    ) {

        // Look for the first < and > signs
        lessPos = original.indexOf( "<" );
        greaterPos = original.indexOf( ">" );
        updated = "";

        while ( lessPos != -1 || greaterPos != -1 ) {

            // Replace the first < or > found
            if ( lessPos != -1 && (lessPos<greaterPos || greaterPos == -1) ) {
                // a < was found before any >
                updated = updated + original.substring(0, lessPos ) + "&lt;";
                original = original.substring( lessPos + 1 );
            } else {
                // a > was found before any <
                updated = updated + original.substring(0, greaterPos ) + "&gt;";
                original = original.substring( greaterPos + 1 );
            }

            // Look for the first remaining < and > signs
            lessPos = original.indexOf( "<" );
            greaterPos = original.indexOf( ">" );
        }

        // Add the tags to indicate the beginning and end of the HTML for the message
        return "<html><body>" + updated + original + "</html></body>";
    }
}
```