

Question 1. Which of the following sets of binary codes could not be a Huffman code for any message? Explain/justify your answer.

a) 11, 101, 100, 01, 000

b) 0, 10, 110, 1110, 1111

c) 10, 01, 010, 110, 111

Question 2. In this question, you are asked to construct some Huffman codes for a specific message.

- a) Construct a Huffman code for the message “bananarama”. Show both the tree you construct and the binary codes used for each of the 5 symbols in the message. How many bits are required to represent the message using this code? (Just the message. Not the description of the code itself.)
- b) You might have noticed that at some points in the process of constructing the first tree, you had the ability to choose several different sets of characters to merge. See what happens if you choose differently. In particular, try to make choices that produce a Huffman tree with a different structure than the tree you constructed for part a. Show both the tree you construct and the binary codes used for each of the 5 symbols in the message. How many bits are required to represent the message bananarama” using this code?

Question 3. In this question, we would like you to construct a Huffman code given information about the probabilities of letter occurrences, rather than an actual message.

- a) Give a Huffman code for messages composed of the six symbols a, b, c, d, e, and f assuming that the letters appear in messages with the following probabilities:

Letter	Probability	Letter	Probability	Letter	Probability
a	7/30	b	10/30	c	3/30
d	4/30	e	4/30	f	2/30

- b) What is the expected number of bits per symbol when the code you constructed is used to encode messages with the given distribution?

Question 4. For this question, we would like you to investigate an alternative to Huffman codes known as Shannon-Fano codes. The Huffman algorithm builds a tree from the bottom up. It starts by merging single letters into pairs. Then, it repeatedly joins small collections of letters to form larger collections. The Shannon-Fano algorithm works instead from the top down. It starts by splitting the letters into two sets, then splits each of these sets into smaller sets until eventually all the letters have been split into sets of their own. Each split leads to the construction of a branch in the tree. When the algorithm splits a set of letters into two smaller subsets it first sorts the letters in decreasing order by their frequencies and second, finds the point in the sorted list where the sum of the frequencies on one side roughly equals the sum of the frequencies on the other side. We give a complete description of the Shannon-Fano algorithm along with an example execution of it on the message “bananarama” in the Appendix at the end of this homework.

Sometimes the Shannon-Fano algorithm produces a code that is as efficient as the Huffman code, but not always. We want you to show an example message where the code produced by the Huffman algorithm is more efficient than the code produced by the Shannon-Fano algorithm.

Question 5. Consider the following simple Java program (with the import statements removed to save space). You should recognize this program from lecture. It displays a button on the screen. Initially the button’s label is “Click Here”. Each time the button is clicked, a JLabel displaying the text “That tickles” is added to the display and the label on the button is replaced by the text “Click Here.”

```
public class TouchyButtonAgain extends GUIManager {
    private JButton myButton;

    public TouchyButtonAgain() {
        this.createWindow( 170, 300 );

        contentPane.add( new JLabel( "Click on the button below" ) );
```

```
        myButton = new JButton( "Click Here" );
        contentPane.add( myButton );
    }

    public void buttonClicked( ) {
        contentPane.add( new JLabel( "That tickles!" ) );
        myButton.setText( "Click again" );
    }
}
```

Below, you will find several programs that are similar but not identical to the program shown above. The differences all involve the declarations and uses of variable and parameter names. Some of these programs will function exactly like the program shown above. Others will work differently or not at all.

For each program, indicate whether a) it will work like the original program, b) a syntax error will be detected as soon as you compile the program, or c) the program will run, but it will not behave like the original (perhaps causing an error to occur while it runs). In cases b and c, briefly describe the error or change in the program's behavior.

While we certainly cannot stop you from actually typing in and running these programs, this assignment will be of much greater value if you answer the questions by simply carefully examining their text and applying your knowledge of how Java programs are interpreted.

```
a) public class TouchyButtonAgain extends GUIManager {
    private JButton myButton;

    public TouchyButtonAgain() {
        this.createWindow( 170, 300 );

        contentPane.add( new JLabel( "Click on the button below" ) );

        myButton = new JButton( "Click Here" );
        contentPane.add( new JButton( "Click Here" ) );
    }

    public void buttonClicked( ) {
        contentPane.add( new JLabel( "That tickles!" ) );
        myButton.setText( "Click again" );
    }
}
```

```
b) public class TouchyButtonAgain extends GUIManager {

    public TouchyButtonAgain() {
        JButton myButton;

        this.createWindow( 170, 300 );

        contentPane.add( new JLabel( "Click on the button below" ) );

        myButton = new JButton( "Click Here" );
        contentPane.add( myButton );
    }

    public void buttonClicked( ) {
        contentPane.add( new JLabel( "That tickles!" ) );
        myButton.setText( "Click again" );
    }
}
```

```
    }  
  }  
  
c) public class TouchyButtonAgain extends GUIManager {  
    private JButton myButton;  
  
    public TouchyButtonAgain() {  
        this.createWindow( 170, 300 );  
  
        contentPane.add( new JLabel( "Click on the button below" ) );  
  
        contentPane.add( new JButton( "Click Here" ) );  
    }  
  
    public void buttonClicked( ) {  
        contentPane.add( new JLabel( "That tickles!" ) );  
        myButton.setText( "Click again" );  
    }  
}  
  
d) public class TouchyButtonAgain extends GUIManager {  
  
    public TouchyButtonAgain() {  
        this.createWindow( 170, 300 );  
  
        contentPane.add( new JLabel( "Click on the button below" ) );  
  
        contentPane.add( new JButton( "Click Here" ) );  
    }  
  
    public void buttonClicked( JButton myButton ) {  
        contentPane.add( new JLabel( "That tickles!" ) );  
        myButton.setText( "Click again" );  
    }  
}
```

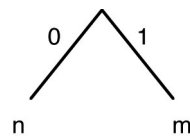
Appendix

Here is a formal description of the Shannon-Fano algorithm.

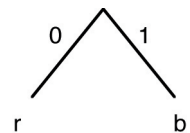
1. Order all the symbols in the set by their frequency.
2. If the set is a single symbol then produce the “tree” consisting of just that symbol.
3. Otherwise, split the ordered list of symbols into two subsets X and Y such that
 - (a) all of the letters in X occur before the letters in Y ,
 - (b) the sum of the frequencies of the letters in X is at least the sum of the frequencies of the letters in Y , but
 - (c) the size of X is as small as it can be subject to (b).
4. Now generate trees for X and Y independently using the same algorithm. Build a tree by joining the two subtrees that result as branches under a single tree node. Label the branch for the first subset as 0 and the branch for the second subset as 1.

For clarity, we illustrate an application of the Shannon-Fano algorithm to the example message: *bananarama*.

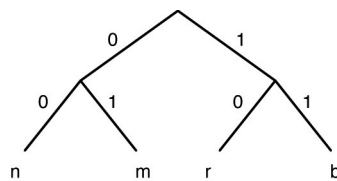
- Step 1 produces the list $\{a, n, m, r, b\}$ since a appears most frequently and $b, r,$ and m appear once each.
- Step 2 does not apply at this point since the set $\{a, n, m, r, b\}$ contains 5 symbols.
- The first letter in the list $\{a, n, m, r, b\}$, the single symbol a , has frequency 5. The frequencies of the remaining symbols also sum to 5 so following step 3 we split the list of 5 symbols into the subset $\{a\}$ and the subset $\{n, m, r, b\}$.
- As stated in step 4, we now apply the steps over again – first to build a tree for the subset $\{a\}$ and then to build one for the subset $\{n, m, r, b\}$.
- Building a tree for $\{a\}$ is handled by step 2. Since there is only one symbol, the tree is trivial. It just consists of the symbol a .
- To build a tree for the list $\{n, m, r, b\}$ we start with step 1. Luckily, the list is already in order and it is not of length 1. Therefore, we skip to step 3. This calls for us to divide the list up into two subsets again. This time, the first subset will be $\{n, m\}$ since it has total frequency 3 which is just more than half of the total frequency of 5 for the set $\{n, m, r, b\}$.
- Now, we go through the algorithm again for the sets $\{n, m\}$ and $\{r, b\}$. Luckily, sets of length two are handled fairly simple. The only way a set of two items can be divided into two subsets that satisfy the conditions in step 3 is to place one symbol in each subset. For example, $\{n, m\}$ is broken up into $\{n\}$ and $\{m\}$. Each of these lists falls under step 2. So we end up with two trivial trees. Step 4 combines them under a branch with the 0 edge going to n and the 1 edge going toward m :



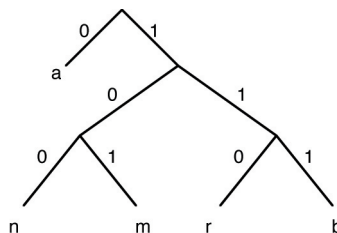
Similarly, the list $\{r, b\}$ will result in the building of the tree:



Now, we get to apply Step 4 again to these two trees, since they are the trees built from the prefix and suffix of the list $\{n, m, r, b\}$. The result is to build the tree:



Finally, we follow Step 4 one last time to combine the trivial tree built for the prefix $\{a\}$ with the tree we just obtained for $\{n, m, r, b\}$ to obtain the complete tree:



Notice that the the Shannon-Fano tree for *bananarama* is very similar to the Huffman tree you created in Question ???. But, as we note above, this is not always the case.