Candidate Assessment Questions

I.

a. Create a Node and a Binary Search Tree (BST) class in JavaScript. Node should have **three public variables** an d **one public function**. The BST class should have **at least one private variable** and **only two public methods**. Y ou may implement additional methods if needed but they must not be public. Do not worry about balancing the tree. Explain how you handle private vs public variables and functions in JavaScript and comment your code to indicate what the algorithm is doing.

Node
+ left : Node + right : Node + value : int
+ height()

BST
- root : Node
+ add(value : int) + height() : int

b. Once you've created your classes, use the following comma separated list as input and build the BST. Print the height of the tree to the console or web page at the end.

Data

```
// Given data
var data = "5,3,4,2,12,22,55,66,15,28,9";
```

II.

```
<ul class="sc-menu">
 <li>Item 1</li>
 <li>Item 2</li>
 <li>Item 3</li>
 <li>Item 4</li>
 <li>Item 5</li>
</ul>
```

a. Given the previous HTML code snippet, write the CSS that displays it as a vertical navigation menu with a width equal to 100 pixels and no defined height. For each item, the background should be a shade of black with white text for each label. There should be 10 pixels between each item. On roll over, the background should change to white and the text should change to the same shade of black you assigned the background by default.

b. Add the ability to display the menu horizontally, where the width of the entire menu is 469 pixels and height of the entire menu is 60 pixels without changing the class name or the class markup you coded in the previous step

c. Display the third menu item as selected by default with a background color of blue and white for the label.

III.

The following block of code creates a 25x25 checker board. When your move the mouse over the board, the box under the mouse is colored green. However, this code is inefficient and the jQuery could use some work.

```
$( 'body' ).append( '<div />' );
$( 'body' ).find( '> div' ).css( {
    width: 250,
    height: 250
} );

for ( var i = 0; i < 625; i++ ) {
    $( 'body' ).find( '> div' ).append( '<div />' );
}

$( 'div > div' )
    .css( 'background', '#000000' )
    .css( 'width', 8 )
    .css( 'height', 8 )
    .css( 'float', 'left' )
    .css( 'margin', 1 );

for ( var j = 0; j < 625; j++ ) {
    if ( j % 2 == 0 ) {
        $( 'div > div:eq(' + j + ')' ).css( 'background', '#999999' );
    }
}

$( 'div > div' ).mouseover( function() {
    $( this ).css( 'background', '#00ff00' );
} );

$( 'div > div' ).mouseout( function() {
    for ( var j = 0; j < 625; j++ ) {
        if ( j % 2 == 0 ) {
            $( 'div > div:eq(' + j + ')' ).css( 'background', '#999999' );
        } else {
            $( 'div > div:eq(' + j + ')' ).css( 'background', '#000000' );
        }
    }
} );
```

a. Please clean up the code and make it run more efficiently. Feel free to modify the code and DOM as needed. However the DOM structure (a single parent div containing separate child divs) should remain throughout the entirety of this problem.

b. Once the code is in optimal shape, a cross-hair effect should be applied to the matrix. That is to say, all boxes in the same row and column as the mouse should be colored green. Moving the mouse should act similarly in that all

of the boxes revert to their original color. Furthermore, clicking should "lock in" the boxes, which are part of the cross-hair, color to red so that moving the mouse does not revert them to black or gray (red becomes their new original color). The cross-hair effect should still change red boxes to green as needed.

c. Finally, implement functionality which resets the matrix to its default checker board coloring when the backspace button is pressed.