CS 229 HW3: Trees

Long explanation: implement a Ternary Search Tree. Nodes have three children, and optionally a reference to the parent. Nodes store Color (red, green, and blue values). This Tree is sorted by Blueness. The child to the left of ***every*** node ***always*** has a **smaller** blue value, the child to the right always has a **larger** blue value, and the child in the middle always has an **equal** blue value.

Here are the functions you need to implement:

- (20%) Right clicking one of the top buttons removes the first occurrence of that Color from the tree. The tree stays valid after removal (hint: often, a Node fills the removed one's place)
  - Start searching at the root. You must perform in log(N) time for full credit (e.g. if the tree has 4 "tiers", it takes no more than 4 iterations to find the Node with that Color).
- (10%) Left clicking a Node in the tree adds a duplicate Node (same color, same border, ID is the next ID available) to the tree in O(1) time. Note that duplicates always add to the middle.
- (10%) Right clicking a Node selects it
  - (10%) Right clicking another Node attempts to swap their positions. If the swap would cause the tree to be invalid, does nothing.
  - (20%) Pressing the "a" key re-creates the tree with the selected Node as the root. All Nodes must keep their Color, background Color, ID, etc.
  - (5%) Deselect the Node if the user clicks into empty space
- (25%) The space key re-balances the tree so it is the optimal height. For example, a tree with 13 Nodes can be expressed as a 3 tier tree (1 + 3 + 9 = 13). A tree with 14-40 Nodes needs 4 tiers. All Nodes must keep their Color, background Color, ID, etc.

This is basically a Binary Search Tree (BST) combined with a Linked List for Nodes that are identical in terms of the search criteria. It's a BST where all duplicates end up in a queue-like structure attached at the middle Node. Its slightly more complicated, but it performs about the same as a BST.

Currently you have the display for the tree, and left clicking one of the colored buttons at the top adds a Node to the tree. The top buttons are ordered by blueness.

This homework is best solved using a lot of recursion. There are a couple of examples in the functions for adding or drawing the Nodes. You might also check the book (see the syllabus) or do your own research for these functions for a Binary Search Tree and adjust accordingly (they are very similar).

Some other information you should know about the starter:

- Every button has a randomly generated colored outline: this isn't part of the sorting mechanism, but Nodes generated by some button also inherit the outline color, which can help you differentiate Nodes with similar shades.
- Nodes all have an ID, this just counts up by one. Again, this is just for you to see which Node is which among identically colored Nodes.
- Once you are 4+ tiers down, Nodes will start to overlap or go offscreen.

Extra credit (15%) Convert the underlying mechanism to the store the tree into an array of Nodes. Nodes no longer have a left/middle/right/parent reference. They are stored in the array. You can give the array a fixed length to hold up 5 tiers, and do nothing if an insertion would exceed that.

As always, up to 5% bonus for exceptional documentation.