The doubling ratio for my implementation of BTKeysAtSameLevel2 should be 2.0.

The method that does all the heavy lifting is called buildTree. The method takes in a string s and returns a Node. I leveraged a Stack in order to make my algorithm linear. We enter into a for loop that goes through the length of the string s that was passed in as a parameter which costs linear time. Then the next major step is entering into the addCurrToParent method which takes in the current node we are up to and our stack of Nodes. In addCurrToParent we check if there is a parent Node by checking if the stack is empty which is a constant time operation. If it does not have a parent we return from the method. If the Node does have a parent then we check if parent's Node left child is null. If it is Null then we set parent.left to the current Node that we passed into the method. Otherwise, we set parent.right to the current Node. Both of these operations are constant time. Once we exit from that method and enter back into buildTree we push the current Node to the stack which again is a constant time operation. Then we set the root if it is null to the current Node. Finally we return the root Node.

The method that puts all the elements into a list of lists is called addLevelOrder and it takes in a Node root as the parameter. I took advantage of a Que to make this method also linear. Meaning while the que isn't empty I enter a for loop of the size of the que.

To summarize in the buildTree method the most expensive operation is traversing the length of a string s. In the addCurrToParent method all the operations are constant. The addLevelOrder method is linear times a constant because the for loop is in a while loop.