Documentation

# First Assignment:

For the first question, implementing the simplyLinkedList was not to challenging because the nodes only go through one direction. Implementing the deleteMin() function had some problems at first but the implementation finished with a while that get through all of the elements and compares them.

For the stack, the implementation was a little bit harder because I needed to work with 2 queues and the solution was to copy the data except for the last element and then removing the last element for the first stack

For the simplyLinkedList swap was hard because we needed to have at least 4 elements to swap and change the node pointer. But for the doblyLinkedList was easier to find the nodes because we can look back at wat was the prior node.

The Bag implementation was the easiest because it is just an array of arrays of the same elements so the only thing to compare is what is the subarray holding.

To make a data structure to behave as a Queue but takes any random element is best to work with indexes so the solution was to make a random number generator that takes input of the size of the list and take that element out of the random queue.

To reverse a DoblyLinkedList we need to get from the last element to the first and is easy because we can get to the next element and the past so the only thing to do is from the last, go to all past elements and add them in that order to a new doblylinkedlist.

For the minStack we use 2 linkedLists that stores all the data and the other stores the minimum element in order so if we remove one element it will always be the minimum element

# Second Assignment:

To get the next element in inorder, preorder, postorder is not to difficult if we make it recursive. The in order and postorder where made recursive because it was easier to add them to a list and the post order, I had some trouble, so I did it with 2 stacks.

To know if we have a BT is also an SBT we only need to compare the elements in a recursive way knowing the left will always be less than the root node and the right node.

To create a hash table that the rule is to add the number in k mod 13 is very easy because we work with an array and the index of that element will only be k mod 13.

To create a Labeled BT that contains the index of inorder, postoder and preorder we need to modify the node so it stores the index and also create a parent class that has methods to add the data to the index in those orders and in all of the cases will always be a counter to add.

# Third Assignment:

For the only implementation the assignment wanted I took a meldable heap from sandoundry so I can implement the remove(u) method. To correctly remove the element, I needed to search for the element to remove and not save it on a queue. Then the only thing left is to add all of the nodes back to the Heap with the queue, so it melds correctly.

https://www.sanfoundry.com/java-program-meldable-heap