Part A:

1. Write `rotateWithLeftChild`, which takes an avlNode, and rotate with its Left Child, update the height and return the new node.

2. Write a function called `isAVLtree` that check whether current Binary Search Tree is an AVLTree. Assume the current Tree is already a valid Binary Search Tree.

3. Write a function called `printByFrequency` to print all given input names in the list, grouped by number of times each name appears in the list. The runtime must be O(nlogn) where n is the number of names.
   All names with the same number of appearance (frequency) should be printed on the same line alphabetically separated by a comma, and names with more appearance should be printed first. The runtime must be O(nlogn) where n is the number of names in the list. It prints out names only.
   You may use following map to store the string to be printed on the same line.
   Note: Choose the appropriate map class. Add names should be efficient, if use a single String to store more than one name may slow down the runtime (but a StringBuilder is fine).

4. Write a function called `sumZero` that takes and array of integers, then check whether there’re two distinct numbers in the array to sum to zero, if so, return the index of first number. If not, return -1. The expected runtime must be O(n) where n is the array length.

Extra Credit: Write a function called `longestZipZag` that returns how many edge in this longest zigzag path. Write the method in the Btree class

A zig-zag path: 1. From node n, choose either left or right direction. 2. If the node is not null from this direction, there’s one edge, and move to that child node. 3. From this child node, change direction and repeat all the steps. An empty tree’s longestZipZag is 0.

Change the class name as `A3SetMap` follow by your initials.

Finish the AVLtree class for question 1 and Btree for question 2 and extra credit.
Part B

1. What is the minimum number of node in an AVL tree with height of 3, height 5 and height 7? Show the code below and write down the result.

Consider following diagram showing the state of an AVL tree

```
        r
       / \
      l   u
     / \ / \
    f n s w
   / \ \   \`
  e i p z
 / \ \```

Diagram 1

2. Write the inorder traversal of the tree.

3. List all single lower case letters whose insertion into an AVL Tree represented by Diagram 1 above would require a rebalance of the tree. (Insert a – z in alphabetic order. Remember BST contains no duplicate datas).

4. Show how the tree in Diagram 1 is changed when the data element s is removed. Show each state of the tree.

5. What is the worse-case runtime of deleting an element from an AVL tree? What about an unbalanced binary search tree?