Detailed System Requirements:

ReturnBST methods takes a BinaryTree and an array then returns a binary search tree which has elements that array's has.

Theese classes written in Java 11 so it can be run in anyplace that java can run. Program starts and prints 3 different BinaryTrees then passes them to the ReturnBST method and prints the resulting trees again. After that prints 3 different BinarySearchTrees and prints them before and after passing them to the ReturnAVL tree.

BinaryTree and BinarySearchTree implementations comes from the book.

Question 1:

ReturnBST method takes a BinaryTree and an array then returns a binary search tree which has elements that array's has. To accomplish this, first the array is sorted (By Array.sort() method which has Θ(nlogn) complexity) than it is implemented inorder traverse on the BinaryTree (to get binary search tree, a node must be bigger than left node and smaller than right node. Inorder traver first goes left branch than root than right branch recursively so it will be easy to put the values to the nodes in order.) and while doing this elements of the array are put to the nodes one by one

Algorithm Analysis:

Best case : Θ (nlogn) -> comes from Arrat.sort()

Worst case: : Θ (nlogn) -> comes from Arrat.sort()

Question 2:

ReturnAVLTree method takes a BinarySearchTree as input and returns an AVL tree as output. To do this it recursively does inorder search and finds every subtree which is unbalanced. Then according to unbalance situtation it implements four different case.
Left Left Case -> Right Rotate

Left Right Case -> Left(rotation of left node)-Right Rotate(rotation of imblance node)

Right Right Case -> Left Rotate

Right Left Case -> Right(rotation of left node)-Left Rotate(rotation of imblance node)

Rotataions are made by the LeftRotate and RightRotate methods. In these methods I implemented the rotations slightly different than usual because these classes are outside the BinarySeacrhTree therefore there was some restirictions (to not break the link of the root of the subtree from the rest of the tree I didn't change it's location. Instead I only rotate subtrees of that root and at the and I swapped the value of the root appropriately).

So ReturnAVLTree uses 4 different helper method. One of them is ReturnAVLTreeHelper which ReturnAVLTree calls is recursively. It call GetHeight function to get the height difference of the nodes to implement accurate rotations. Then according to height differences it call LeftRotate and RightRotate methods to get AVL trees.

Algorithm Analysis:

LeftRotate and RightRotate methods takes constant time (Θ (1)).
GetHeight method takes linear time (Θ (n)) since it goes every node of to tree.
ReturnAVLTreeHelper method calls theese methods but if we ignore the complexity of theese called methods it takes constant time (Θ (1)).
ReturnAVLTree recursively goes every subtree of the tree and calls ReturnAVLTreeHelper method. If we ignore the complexity of called method it takes linear time (Θ (n)).

So;

Best case : Θ (n^2)

Worst case: : Θ (n^2)

Test Cases:

For the first question three different BinaryTree and three different array which has same elements with theese BinaryTrees are constructed. BinaryTrees are printed with toString method to show their structure and then printed again after being passes to ReturnBST method as parameter with their corresponding array to show ReturnBST method is working and turns BinaryTrees into BinarySearchTrees.

For the second question three different BinarySearchTree are constructed randomly then they printed with toString method to show their structure. Then they printed again after being passes to ReturnAVLTree method as parameter to show ReturnAVLTree method is working and turns BinarySearchTrees into AVL trees.

```
1. BinaryTree:

2
 5
  1
   null
   null
  9
   null
   null
 8
  7
   null
   null
  6
   null
   null

1. BinaryTree to BinarySeachTree:

6
 2
  1
   null
   null
  5
   null
   null
 8
  7
   null
   null
  9
   null
   null
```

2. BinarySeachTree:

```
59
 35
  null
  49
   38
    null
    45
     null
     null
   null
 65
  null
  79
   null
   null
```

2. BinarySeachTree to AVL tree:

```
49
 38
  35
   null
   null
  45
   null
   null
 65
  59
   null
   null
  79
   null
   null
```