First of all, time analysis is not completed because there is some problems in graph and expected results do not match . So I only implement first 5 parts of homework.

RandomInputFileGenerator Class Report

1. Introduction

- Briefly introduce the purpose of the **RandomInputFileGenerator** class.
- Mention that it generates a random input file containing commands for testing the **StockDataManager** class.

2. Purpose

- Describe the main purpose of the class: to create random input files for testing stock data management functionality.
- Explain that it generates commands for adding, removing, searching, and updating stock data.

3. Implementation Overview

- Highlight key components of the class:
    - Constants for symbol length and methods for generating commands.
    - Usage of **Random** object for generating random values.
    - Utilization of sets to track added and removed symbols.
    - Methods for generating add, remove, search, and update commands.
    - File writing using **FileWriter**.
- Discuss how the class utilizes loops to generate the specified number of each type of command.

4. Methodology

- Explain each method briefly:
    - **generateRandomInputFile**: Main method that orchestrates the generation of the random input file.
    - **generateAddCommand**: Generates an 'ADD' command with random stock data.
    - **generateRemoveCommand**: Generates a 'REMOVE' command for existing symbols.
    - **generateSearchCommand**: Generates a 'SEARCH' command for existing or new symbols.
    - **generateUpdateCommand**: Generates an 'UPDATE' command for existing symbols with new data.
    - Utility methods for generating random symbols and selecting random elements from sets.
    -

5. Sample Usage

- Provide a brief example demonstrating how to use the **RandomInputFileGenerator** class to create a random input file.

Node Class:

- Represents a node in the AVL tree.
- Contains references to left and right child nodes, as well as the height of the node.
- Stock object is stored within each node.

AVLTree Class:

- Maintains a reference to the root node of the AVL tree.

Public Methods:

1. **insert(Stock stock)**:
    - Inserts a stock into the AVL tree while maintaining the AVL property.
    - Utilizes the private method **insert(Node node, Stock stock)** for recursive insertion and balancing.
2. **delete(String symbol)**:
    - Deletes a stock from the AVL tree based on its symbol.
    - Utilizes the private method **deleteNode(Node node, String symbol)** for recursive deletion and balancing.
3. **search(String symbol)**:
    - Searches for a stock in the AVL tree based on its symbol.
    - Utilizes the private method **search(Node node, String symbol)** for recursive search.
4. **inOrderTraversal()**:
    - Initiates an in-order traversal of the AVL tree.
    - Utilizes the private method **inOrderTraversal(Node node)** for recursive traversal.

Private Methods:

1. **insert(Node node, Stock stock)**:
    - Recursive method to insert a stock into the AVL tree.
    - Balances the tree after insertion.
2. **deleteNode(Node node, String symbol)**:

   - Recursive method to delete a node from the AVL tree based on its symbol.
   - Balances the tree after deletion.

3. **search(Node node, String symbol)**:
   - Recursive method to search for a stock in the AVL tree.
4. **balance(Node node)**:
   - Balances the AVL tree by performing rotations.
   - Utilizes left and right rotations (**leftRotate(Node x)** and **rightRotate(Node y)**).
5. **height(Node N)**:
   - Calculates the height of a given node in the AVL tree.
6. **getBalance(Node N)**:
   - Calculates the balance factor of a given node.
7. **updateHeight(Node node)**:
   - Updates the height of a given node.

Rotations:

- Left and right rotations are performed to maintain the balance of the AVL tree during insertion and deletion operations.

Output:

- The **inOrderTraversal()** method allows for the traversal of the AVL tree, printing out the stocks in sorted order.