CSE 222 – Data Structures and Algorithms Homework 7 Report

Ahmet Özdemir

Table Of Contents

1- Introduction
2- Main.java
3- AVLTree.java
4- Generate Random Input File
5- Visualization
6- Sample Outputs
7- Usage With Makefile

1- Introduction:

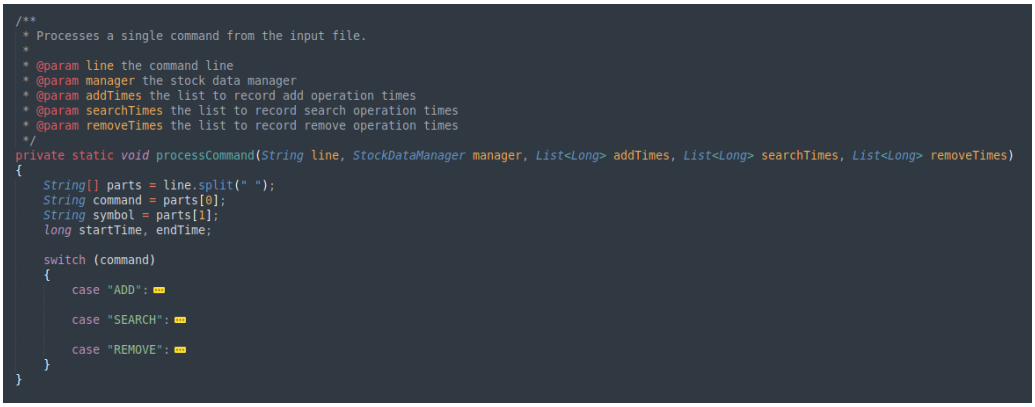
In this report I will explain the program I implemented for the homework. First I will give information about the classes I added and the classes I changed. Then I will explain the general path I followed.

What I understood from the instruction "- The script used to generate random input files." in the assignment document was that the user can ask the program to write a randomly filled input.txt file at any time. This file will be in the format given in the assignment document, completely random and variable. I made some changes in some files and in the Main file.

2- Main.java:

```java
/**
 * The main method to start the application.
 *
 * @param args the command line arguments, expecting the input file path as the only argument
 */
public static void main(String[] args)
{
    if (args.length != 1)
    {
        System.out.println("Usage: java Main <input_file>");
        return;
    }

    String inputFile = args[0];
    StockDataManager manager = new StockDataManager();

    List<Long> addTimes = new ArrayList<>();
    List<Long> searchTimes = new ArrayList<>();
    List<Long> removeTimes = new ArrayList<>();

    try (BufferedReader br = new BufferedReader(new FileReader(inputFile)))
    {
        String line;

        while ((line = br.readLine()) != null)
        {
            processCommand(line, manager, addTimes, searchTimes, removeTimes);
        }
    }

    catch (IOException e)
    {
        e.printStackTrace();
    }

    printAverageTimes(addTimes, searchTimes, removeTimes);
    SwingUtilities.invokeLater(() -> createAndShowGUI(manager, addTimes, searchTimes, removeTimes));
}
```

In main, I first read the input.txt file line by line and then send each line with its content to the processCommands function. While this function performs the necessary operations according to the incoming command, it also keeps the duration of these operations. It keeps these time values in Lists created for add, remove and search.



```java
/**
 * Processes a single command from the input file.
 *
 * @param line the command line
 * @param manager the stock data manager
 * @param addTimes the list to record add operation times
 * @param searchTimes the list to record search operation times
 * @param removeTimes the list to record remove operation times
 */
private static void processCommand(String line, StockDataManager manager, List<Long> addTimes, List<Long> searchTimes, List<Long> removeTimes)
{
    String[] parts = line.split(" ");
    String command = parts[0];
    String symbol = parts[1];
    long startTime, endTime;

    switch (command)
    {
        case "ADD": ...

        case "SEARCH": ...

        case "REMOVE": ...
    }
}
```

processCommand function’s pseudo version.

After these steps, the functions that print the necessary messages to the terminal continue the operations.

3- AVLTree.java:

Firstly, let's explain the Node class and methods in order:

- Node Class: The Node class is a nested class within the AVLTree class. It represents each node in the AVL tree. Each node contains a Stock object, references to its left and right children, and its height.
- Insert(Stock ...): This method inserts a new stock into the AVL tree by calling the private recursive insert method. It starts the insertion process from the root of the tree.
- Insert(Node ..., Stock ...): This is a private recursive method that inserts a new stock into the AVL tree. It compares the stock symbol to decide where to insert the new node. It then updates the height of the current node and balances the tree.
- Delete(String ...): This method deletes a stock from the AVL tree based on its symbol by calling the private recursive delete method. It starts the deletion process from the root of the tree.
- Delete(Node ..., String ...): This is a private recursive method that deletes a stock from the AVL tree. It compares the stock symbol to find the node to delete. If the node to be deleted has two children, it finds the inorder successor to replace it. It then updates the height of the current node and balances the tree.

- Search(String ...): This method searches for a stock in the AVL tree based on its symbol by calling the private recursive search method. It starts the search from the root of the tree.
- Search(Node ..., String ...): This is a private recursive method that searches for a stock in the AVL tree. It compares the stock symbol to search in the left or right subtree. It returns the stock if found, otherwise null.
- MinValueNode(Node ...): This method finds the node with the minimum value in the subtree rooted with the given node. It searches the leftmost subtree as it always contains the minimum value.
- Height(Node ...): This method returns the height of the given node. If the node is null, it returns 0.
- Balance(Node ...): This method balances the AVL tree at the given node. It checks for four cases of imbalance and performs the appropriate rotations to balance the tree.
- GetBalance(Node ...): This method returns the balance factor of the given node, which is the difference in height between the left and right subtrees.
- RightRotate(Node ...): This method performs a right rotation on the subtree rooted with the unbalanced node. It updates the heights of the nodes involved in the rotation and returns the new root of the subtree.
- LeftRotate(Node ...): This method performs a left rotation on the subtree rooted with the unbalanced node. It updates the heights of the nodes involved in the rotation and returns the new root of the subtree.
- InOrderTraversal(): This method performs an in-order traversal of the AVL tree and returns a list of stocks in in-order. It calls the private recursive inOrderTraversal method.
- InOrderTraversal(Node ..., List<Stock ...> ...): This is a private recursive method that performs an in-order traversal of the AVL tree. It adds the stocks to the provided list in in-order sequence.

Some methods of my program in AVLTree class:

```java
/**
 * Recursively inserts a new stock into the AVL tree.
 *
 * @param node The current node in the AVL tree.
 * @param stock The stock to be inserted.
 * @return The updated node.
 */
private Node insert(Node node, Stock stock) // recurs,ve method of inserting
{
    if (node == null)
    {
        return new Node(stock);
    }

    /***** Comparing the stock symbol to decide where to insert the new node *****/

    if (stock.getSymbol().compareTo(node.stock.getSymbol()) < 0)
    {
        node.left = insert(node.left, stock);
    }

    else if (stock.getSymbol().compareTo(node.stock.getSymbol()) > 0)
    {
        node.right = insert(node.right, stock);
    }

    else
    {
        return node; // Duplicate keys are not allowed
    }

    /*************************************************************************/

    // Updating the height of the current node
    node.height = 1 + Math.max(height(node.left), height(node.right));

    return balance(node);
}
```

```java
/**
 * Balances the AVL tree at the given node.
 *
 * @param node The node to be balanced.
 * @return The balanced node.
 */
private Node balance(Node node) // Balance AVL tree
{
    int balance = getBalance(node);

    /* case:
             root
              /
           Left
           /   \
      (Left)
    */
    if (balance > 1 && getBalance(node.left) >= 0)
    {
        return rightRotate(node);
    }

    /* case:
             root
              /
           Left
           /   \
            (Right)
    */
    if (balance > 1 && getBalance(node.left) < 0)
    {
        node.left = leftRotate(node.left);
        return rightRotate(node);
    }

    /* case:
             root
                \
              Right
              /   \
                 (Right)
    */
    if (balance < -1 && getBalance(node.right) <= 0)
    {
        return leftRotate(node);
    }

    /* case:
             root
                \
              Right
              /   \
             (left)
    */
    if (balance < -1 && getBalance(node.right) > 0)
    {
        node.right = rightRotate(node.right);
        return leftRotate(node);
    }
    return node;
}
```

4- Generating Random Input File:

The GenerateRandomInput class is responsible for generating random input commands for a stock management system and writing these commands to a file. The generated

commands include adding, removing, searching, and updating stocks with randomly generated properties. Below is a brief explanation of each component of this class.

A. Constants:

- RANDOM: A Random object seeded with the current time in milliseconds to generate random numbers.

- CHARACTERS: A string containing all uppercase letters used for generating stock symbols.

- OUTPUT_DIR: The directory where the output file will be saved.

- OUTPUT_FILE: The path of the output file.

- MIN_COMMANDS, MAX_COMMANDS: Constants defining the minimum and maximum number of commands to be generated.

- SYMBOL_LENGTH: The length of the stock symbols to be generated.

- symbols: A Set to store unique stock symbols and ensure no duplicates.

B. main(String[] args):

This is the entry point of the program. It determines the number of commands to generate and writes them to the output file.

- The number of commands is randomly chosen between MIN_COMMANDS and MAX_COMMANDS.

- The output directory is created if it does not exist.

- Commands are generated and written to the output file line by line.

C. generateCommand():

This method generates a random stock management command. The possible commands are ADD, REMOVE, SEARCH, and UPDATE.

- ADD: Generates a new unique stock symbol and its properties (price, volume, market capitalization) and adds the symbol to the symbols set.

- REMOVE, SEARCH, UPDATE: Uses a random existing stock symbol from the symbols set for these commands. If the symbols set is empty, a new unique symbol is generated.

D. generateUniqueSymbol():

This method generates a unique stock symbol that is not already present in the symbols set.

E. generateSymbol():

This method generates a random stock symbol of length SYMBOL_LENGTH using characters from the CHARACTERS string.

F. getRandomExistingSymbol():

This method retrieves a random existing stock symbol from the symbols set. If the set is empty, it generates a new unique symbol.

G. generatePrice(), generateVolume(), generateMarketCap():

These methods generate random properties for the stock:

- generatePrice(): Generates a random stock price between 10 and 1000.

- generateVolume(): Generates a random stock volume between 1000 and 1,000,000.

- generateMarketCap(): Generates a random stock market capitalization between 1,000,000 and 1,000,000,000.

Overall, the GenerateRandomInput class provides a way to generate a variety of stock management commands with random but realistic properties and writes these commands to a file for use in testing or demonstration purposes.

```java
/*
    note: we keep these names here because creating random
    names all the time and never using them makes the
    search, remove and update commands ineffective
*/
public static void main(String[] args)
{
    // determines randomly, commands number
    int numCommands = RANDOM.nextInt((MAX_COMMANDS - MIN_COMMANDS) + 1) + MIN_COMMANDS;

    try
    {
        // create output directıry
        Files.createDirectories(Paths.get(OUTPUT_DIR));

        // start the writing output file
        try (BufferedWriter writer = new BufferedWriter(new FileWriter(OUTPUT_FILE)))
        {
            for (int i = 0; i < numCommands; i++) // randomly creating command and add to the file
            {
                String command = generateCommand();
                writer.write(command);
                writer.newLine();
            }
            System.out.println("Random input file generated with " + numCommands + " commands: " + OUTPUT_FILE);
        }
    }

    catch (IOException e)
    {
        e.printStackTrace();
    }
}
```

```java
/**
 * Generates a random command for stock management.
 *
 * @return A string representing the command.
 */
private static String generateCommand() // method for generating random command
{
    String[] commands = {"ADD", "REMOVE", "SEARCH", "UPDATE"}; // command types
    String command = commands[RANDOM.nextInt(commands.length)]; //randomly selecting a command

    switch (command)
    {
        case "ADD":

            String symbolToAdd = generateUniqueSymbol();
            symbols.add(symbolToAdd);

            return String.format("%s %s %.2f %d %d", command, symbolToAdd, generatePrice(), generateVolume(), generateMarketCap());

        case "REMOVE":

            return String.format("%s %s", command, getRandomExistingSymbol());

        case "SEARCH":

            return String.format("%s %s", command, getRandomExistingSymbol());

        case "UPDATE":

            return String.format("%s %s %.2f %d %d", command, getRandomExistingSymbol(), generatePrice(), generateVolume(), generateMarketCap());

        default:

            return "";
    }
}
```

Some sample output for these parts:

```
ahmete@ahmete-Inspiron-14-5401:~/DERSLER/2_SINIF/spring/Data-Structures/homework/hw7/HW7/HW7/HW7_Demo/src$ make clean
ahmete@ahmete-Inspiron-14-5401:~/DERSLER/2_SINIF/spring/Data-Structures/homework/hw7/HW7/HW7/HW7_Demo/src$ make all
ahmete@ahmete-Inspiron-14-5401:~/DERSLER/2_SINIF/spring/Data-Structures/homework/hw7/HW7/HW7/HW7_Demo/src$ make generate_input
Random input file generated with 467 commands: ../out/production/HW7_Demo/input.txt
ahmete@ahmete-Inspiron-14-5401:~/DERSLER/2_SINIF/spring/Data-Structures/homework/hw7/HW7/HW7/HW7_Demo/src$ make run
java -Xint -cp ../out/production/HW7_Demo Main ../out/production/HW7_Demo/input.txt
Stock not found: LNZB
Stock not found: MFHO
Stock not found: MFHO
Stock not found: XWVM
Stock [symbol=COFZ, price=901.34, volume=827987, marketCap=322006324]
Stock not found: MFHO
Stock not found: MFHO
Stock not found: MFHO
Stock [symbol=COFZ, price=901.34, volume=827987, marketCap=322006324]
Stock not found: MFHO
Stock not found: MFHO
Stock [symbol=COFZ, price=901.34, volume=827987, marketCap=322006324]
Stock [symbol=LGXO, price=444.37, volume=277479, marketCap=455318012]
Stock not found: XWVM
Stock not found: XWVM
Stock [symbol=HHLX, price=813.95, volume=119253, marketCap=392895082]
Stock not found: LGXO
Stock [symbol=KGVT, price=261.49, volume=479882, marketCap=489279513]
Stock not found: LGXO
Stock [symbol=EEEZ, price=347.65, volume=970231, marketCap=457788659]
Stock not found: POXA
Stock not found: XRGW
Stock [symbol=KGVT, price=261.49, volume=479882, marketCap=489279513]
Stock [symbol=ENVI, price=784.29, volume=373739, marketCap=36381140]
Stock not found: UMYJ
Stock not found: BVVJ
Stock not found: MFHO
Stock not found: MFHO
Stock [symbol=KWMG, price=42.48, volume=319441, marketCap=463085262]
Stock not found: POXA
Stock [symbol=UHHW, price=483.7, volume=141354, marketCap=665368752]
Stock [symbol=KWMG, price=42.48, volume=319441, marketCap=463085262]
Stock [symbol=UHHW, price=483.7, volume=141354, marketCap=665368752]
Stock not found: UMYJ
Stock not found: UMYJ
Stock [symbol=XIAN, price=822.83, volume=962071, marketCap=336284346]
Stock not found: HHLX
```

```
Stock not found: HHLX
Stock [symbol=KMFC, price=165.94, volume=393901, marketCap=523706407]
Stock [symbol=RRON, price=233.51, volume=588857, marketCap=388346176]
Stock not found: NBHL
Stock not found: THVT
Stock [symbol=UQUS, price=481.86, volume=114341, marketCap=359309086]
Stock [symbol=HKZJ, price=926.51, volume=488142, marketCap=497117313]
Stock not found: XRGW
Stock not found: EGTW
Stock [symbol=LVBC, price=998.8, volume=102780, marketCap=227641106]
Stock [symbol=DSMZ, price=154.31, volume=475975, marketCap=658581658]
Stock [symbol=XIAN, price=822.83, volume=962071, marketCap=336284346]
Stock not found: UHHW
Stock not found: EGTW
Stock [symbol=HKZJ, price=926.51, volume=488142, marketCap=497117313]
Stock not found: NBHL
Stock [symbol=SFAX, price=597.46, volume=101519, marketCap=155655341]
Stock [symbol=KMFC, price=165.94, volume=393901, marketCap=523706407]
Stock not found: TPOX
Stock [symbol=KWMG, price=42.48, volume=319441, marketCap=463085262]
Stock [symbol=DLOA, price=312.3, volume=390013, marketCap=570625179]
Stock [symbol=DSMZ, price=154.31, volume=475975, marketCap=658581658]
Stock not found: XRGW
Stock not found: NGVP
Stock not found: POXA
Stock [symbol=FMXU, price=140.16, volume=592371, marketCap=639708780]
Stock [symbol=FMXU, price=140.16, volume=592371, marketCap=639708780]
Stock not found: IVXU
Stock not found: WVCS
Stock [symbol=DSMZ, price=154.31, volume=475975, marketCap=658581658]
Stock not found: EGTW
Stock [symbol=FOFL, price=444.54, volume=75400, marketCap=713557031]
Stock [symbol=RYWT, price=574.93, volume=808328, marketCap=167313934]
Stock [symbol=UGGQ, price=189.03, volume=388133, marketCap=664227429]
Stock [symbol=TRKG, price=534.98, volume=612921, marketCap=756249728]
Stock not found: WVCS
Stock [symbol=YHTA, price=643.44, volume=418320, marketCap=870008794]
Stock [symbol=YOKS, price=513.32, volume=448618, marketCap=853905537]
Stock [symbol=RRON, price=233.51, volume=588857, marketCap=388346176]
Stock [symbol=SUAZ, price=512.15, volume=208006, marketCap=743657150]
Stock [symbol=ZMKU, price=191.24, volume=878255, marketCap=450208279]
Stock not found: NBHL
Stock not found: NGVP
```

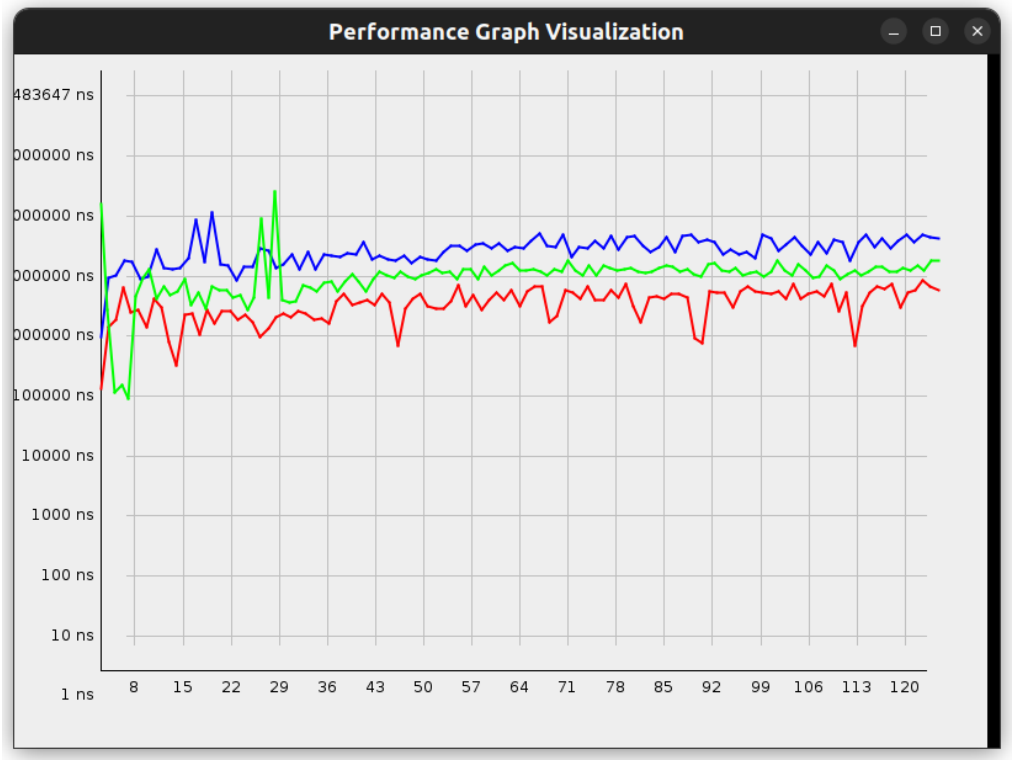
```
Stock [symbol=RRON, price=233.51, volume=588857, marketCap=388346176]
Stock [symbol=SUAZ, price=512.15, volume=208006, marketCap=743657150]
Stock [symbol=ZMKU, price=191.24, volume=878255, marketCap=450208279]
Stock not found: NBHL
Stock not found: NGVP
Stock [symbol=NNET, price=651.96, volume=601482, marketCap=632095476]
Stock [symbol=UQUS, price=481.86, volume=114341, marketCap=359309086]
Stock [symbol=KGVT, price=261.49, volume=479882, marketCap=489279513]
Stock [symbol=MIPX, price=94.12, volume=455058, marketCap=686631139]
Stock not found: BVVJ
Stock not found: YOKS
Stock [symbol=ZFBD, price=834.13, volume=340756, marketCap=640957114]
Stock not found: FKYB
Stock not found: HKZJ
Stock not found: SZTF
Stock not found: EGTW
Stock not found: DTSA
Stock not found: SZTF
Stock [symbol=YMDJ, price=870.59, volume=801171, marketCap=442001034]
Stock not found: FNEW
Stock [symbol=DLOA, price=312.3, volume=390013, marketCap=570625179]
Stock not found: TPOX
Stock not found: TPOX
Stock [symbol=BIID, price=135.12, volume=887614, marketCap=933896399]
Stock not found: JRIO
Stock [symbol=NITQ, price=720.81, volume=206515, marketCap=869138966]
Stock not found: FHGJ
Stock [symbol=KGVT, price=261.49, volume=479882, marketCap=489279513]
Stock [symbol=LPSV, price=959.19, volume=33200, marketCap=44888837]
Stock not found: UHHW
Stock not found: IJRT
Stock [symbol=YUIM, price=417.23, volume=518614, marketCap=369870647]
Stock not found: TPOX
Stock [symbol=ICWZ, price=407.06, volume=229668, marketCap=107863224]
Stock not found: RRON
Stock [symbol=ZPEZ, price=79.34, volume=560053, marketCap=849318385]
Stock not found: SWDI
Stock not found: XIAN
Stock [symbol=LPSV, price=959.19, volume=33200, marketCap=44888837]
Average ADD time: 14881.935185185184 ns
Average SEARCH time: 3453.0619469026547 ns
Average REMOVE time: 6698.5203252032525 ns
ahmete@ahmete-Inspiron-14-5401:~/DERSLER/2_SINIF/spring/Data-Structures/
```

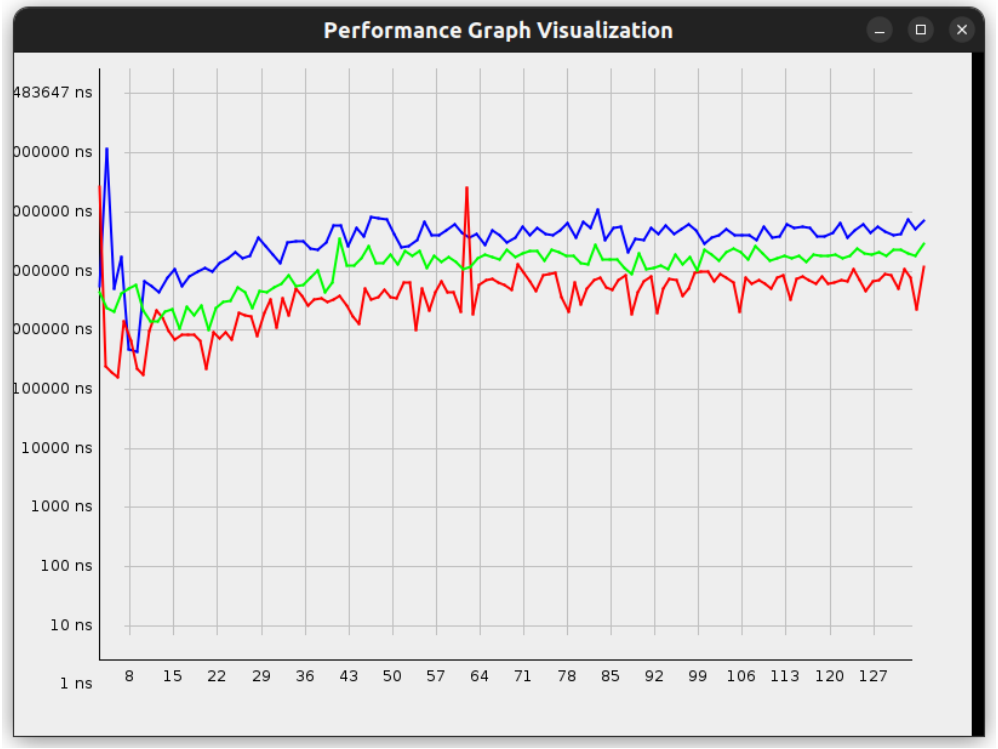
5- Visualization:

In this section, based on the runtimes of the ADD, REMOVE and SEARCH operators, one graph of each is printed in a single table. Here the graphs do not exhibit a perfect O(logn) behaviour. However, a careful look at the graphs will show that the graphs are logn graphs when a few radical changes are ignored.

6- Sample Outputs:

Now let's look at some output examples:


Performance Graph Visualization
483647 ns
000000 ns
000000 ns
000000 ns
000000 ns
100000 ns
10000 ns
1000 ns
100 ns
10 ns
1 ns
8 15 22 29 36 43 50 57 64 71 78 85 92 99 106 113 120


Performance Graph Visualization
483647 ns
000000 ns
000000 ns
000000 ns
000000 ns
100000 ns
10000 ns
1000 ns
100 ns
10 ns
1 ns
8 15 22 29 36 43 50 57 64 71 78 85 92 99 106 113 120 127

7- Usage With Makefile:

This Makefile is designed to automate the process of compiling, cleaning, running, and documenting a Java project. Below are the available targets and their descriptions:

Targets:

A. all

- Description: Compiles all Java source files found in the source directory (SRC_DIR).
- Usage: make all
- Important Notes: The compiled classes are placed in the output directory (OUT_DIR).

B. clean

- Description: Cleans the output directory by removing all compiled class files and generated input files.
- Usage: make clean
- Important Notes: Also removes the documentation directory (DOC_DIR) and the generated input file.

C. run

- Description: Runs the main class with the input file.
- Usage: make run
- Important Notes: Requires the input file (input.txt) to be present in the output directory. Compiles the project if not already compiled.

D. build_and_run

- Description: Cleans the output directory, compiles all Java source files, and runs the main class with the input file.
- Usage: make build_and_run
- Important Notes: Combines clean, all, and run targets to perform a full build and run sequence.

E. javadoc

- Description: Generates Javadoc documentation for the Java source files.

- Usage: make javadoc

- Important Notes: The generated documentation is placed in the documentation output directory (DOC_DIR).

F. generate_input

- Description: Generates a random input file for the stock management system.

- Usage: make generate_input

- Important Notes: The input file is generated in the output directory (OUT_DIR).

G. full_build_and_run

- Description: Cleans the output directory, compiles all Java source files, generates Javadoc documentation, generates the random input file, and runs the main class.

- Usage: make full_build_and_run

- Important Notes: Combines clean, all, javadoc, generate_input, and run targets to perform a complete build, documentation, and run sequence.

.PHONY: This directive marks the targets that are not actual files, ensuring that make will run the corresponding commands even if a file with the target's name exists.

```
# Java compiler
JAVAC = javac
# Java documentation tool
JAVADOC = javadoc
# Source directory
SRC_DIR = .
# Output directory for compiled classes
OUT_DIR = ../out/production/HW7_Demo
# Documentation output directory
DOC_DIR = ../doc

# List of all Java source files
SOURCES = $(wildcard $(SRC_DIR)/*.java)

# Rule to compile all Java files
all: $(SOURCES)
    @$(JAVAC) -d $(OUT_DIR) $(SOURCES)

# Rule to clean the output directory
clean:
    @rm -rf $(OUT_DIR)/*.class $(DOC_DIR) $(OUT_DIR)/input.txt

# Rule to run the main class with input file
run: all
    java -Xint -cp $(OUT_DIR) Main $(OUT_DIR)/input.txt

# Rule to compile and run
build_and_run: clean all run

# Rule to generate Javadoc
javadoc:
    @$(JAVADOC) -d $(DOC_DIR) $(SOURCES)

# Rule to generate random input file
generate_input: all
    @java -cp $(OUT_DIR) GenerateRandomInput

# Rule to clean, compile, generate documentation, and run
full_build_and_run: clean all javadoc generate_input run

.PHONY: all clean run build_and_run javadoc generate_input full_build_and_run

```