TRINITY COLLEGE
DEPARTMENT OF COMPUTER SCIENCE

CPSC 215: Data Structures and Algorithms
Instructor: Dr. Chandranil Chakraborttii
Spring 2025

Laboratory 10

Academic Honesty Policy

In working on programming assignments, you may discuss broad issues of interpretation and understanding and general approaches to a solution. However, conversion to a specific solution or to program code must be your own work. Programming assignments are expected to be the work of the individual student, designed, and coded by him or her alone. Violations are easy to identify and will be dealt with promptly according to the Academic Integrity and Intellectual Dishonesty outlined in the Student Handbook.

- Copying another person's programs or encouraging or assisting another person to commit plagiarism is cheating. In particular, the following activities are strictly prohibited:
- Giving and receiving help in the actual development of code or writing of an assignment.
- Looking at another person's code or showing your code to another person.
- Sharing a copy of all or part of your code regardless of whether that copy is on paper or in a computer file.
- Turning in the work of any other person(s) (former students, friends, textbook authors, people on the Internet, etc.) and representing it as your own work.
- Fabricating compilation or execution results.
- Use of AI ChatBots for any help regarding the assignment is **strictly prohibited.**

The penalty for cheating is a failing grade (F) for the course, and the student is asked to appear at an Academic Dishonesty Hearing. In addition, the College also places a record of the incident in the student's permanent record. It is your responsibility to protect your work from unauthorized access. Do not discard copies of your programs in public places. Do not leave computers unattended and copies of output lying around.

Submission Guidelines:

- Upon the completion of your lab, upload your submission (PerformanceComparison.java and CSV file) to the Moodle course website as a single Zip file.
- See your TA and get your Assignment graded before leaving the lab. Otherwise, inform the instructor.

Lab: AVL Trees and Comparing Data Structures for insertion, searching and deletion

Part A: AVL Tree Rotations and Insertion

In this part, you are asked to complete the AVL Tree insertion logic. An AVL Tree is a self-balancing Binary Search Tree where the difference between heights of left and right subtrees (balance factor) is at most one for all nodes.

Input Files (Starter Code)

- AVLTree.java: AVL Tree implementation (Need to complete)
- AVLTreeTest.java. Tester (No need to change)

Methods to complete:

- private Node<T> rotateRight(Node<T> y)
- private Node<T> rotateLeft(Node<T> x)
- private Node<T> insertRec(Node<T> root, T data)
-

Upon completion:

- Test with AVLTreeTest.java (Sample data inserted: 30, 20, 40, 10, 25, 35, 50)
- AVL Tree demo: https://www.cs.usfca.edu/~galles/visualization/AVLtree.html

---

Pseudocode: rightRotate(y)

`Performs a right rotation around node y.` **`Used when left subtree is too tall (e.g., Left-Left case)`**

- `Let x = y.left`
- `Let T2 = x.right`
- `Rotate:`
  - `Set x.right = y`
  - `Set y.left = T2`
- `Update heights:`
  - `y.height = 1 + max(height(y.left), height(y.right))`
  - `x.height = 1 + max(height(x.left), height(x.right))`
- `Return x (new root of subtree)`

Pseudocode: **leftRotate(x)**

Performs a left rotation around node x. **Used when right subtree is too tall (e.g., Right-Right case)**

- Let y = x.right
- Let T2 = y.left
- Rotate:
  - Set y.left = x
  - Set x.right = T2
- Update heights:
  - x.height = 1 + max(height(x.left), height(x.right))
  - y.height = 1 + max(height(y.left), height(y.right))
- Return y (new root of subtree)

Pseudocode: AVL Tree Insertion

- **Function**: insert(node, key)
  - If node is null:
    - Create and return a new node with key
  - If key is less than node.key:
    - Set node.left to insert(node.left, key)
  - Else if key is greater than node.key:
    - Set node.right to insert(node.right, key)
  - Else:
    - Return node (Duplicate keys not allowed)
  - Update the height of the node
    - node.height = 1 + max(height(node.left), height(node.right))
  - Calculate the balance factor
    - balance = height(node.left) - height(node.right)
  - **Check for imbalance and rotate accordingly**:
    - If balance > 1 and key < node.left.key:
      - Right rotate and return result
    - If balance < -1 and key > node.right.key:
      - Left rotate and return result
    - If balance > 1 and key > node.left.key:
      - Set node.left = left rotate(node.left)
      - Right rotate and return result
    - If balance < -1 and key < node.right.key:
      - Set node.right = right rotate(node.right)
      - Left rotate and return result
  - Return node

Sample Output

```
AVL Tree Display:
Please read tree right (Root) to left (Leaves)
        50
    40
        35
30
        25
    20
        10
```

Part B: Performance Comparison of Data Structures

Input Files (Starter Code)

- AVLTree.java: AVL Tree implementation (Use from above)
- BinarySearchTree.java: BST implementation (No Need to change)
- Queue.java: Queue implementation (No Need to change)
- Heap.java: Heap implementation (No Need to change)
- Stack.java: Stack implementation (No Need to change)
- HashTable.java: Hash Table implementation (No Need to change)
- SplayTree.java: Splay Tree implementation (No Need to change)
- PerformanceComparison.java **Need to create** (Check instructions below)

Tasks:

- **Review** the implemented classes.
- Complete the **Lab_10 worksheet (Expectations section).**
- Create class called **PerformanceComparison.java** to compare insertion of N elements, then deletion and searching of N/2 elements using the data structures provided above.
- **Generate Random Elements:** Create a generateRandomElements method to generate an array of random elements of a given size. Use the Random class for generating random numbers. You may take help from previous labs. The method signature should be - **int[] generateRandomElements(int size)**
- **Main Method:** Implement main method to conduct performance comparison tests for each data structure. In the main method, perform the following steps for each data structure:
    - Create an array of list sizes to test: {1000, 10000, 100000, 500000}
    - For each list size in the array:
        - Generate N random elements using the generateRandomElements method.

    - Create an instance of the data structure to be tested (e.g., AVLTree, BinarySearchTree, etc.).
    - Measure the time taken for insertion of elements using the System.nanoTime() method before and after the insertion loop.
    - Measure the time taken for deletion of n/2 elements using the System.nanoTime() method before and after the deletion loop.
    - Measure the time taken for searching of n/2 elements using the System.nanoTime() method before and after the search loop.
    - Print the timing results for each operation.
- **Record Timing:** Record the timing results for each data structure and each operation (insertion, deletion, and searching) in a table format. You can use a simple text-based table for recording the timings.
- **Repeat for Each Data Structure:** Repeat steps 3-5 for each data structure (AVLTree, BinarySearchTree, HashTable, Queue, and Stack).
- **Ranking and Reasoning:** After conducting the performance tests, rank the data structures based on their performance for each operation. Provide reasoning for the ranking, explaining why certain data structures performed better or worse than others.
- **Complete the Lab_12 worksheet (Reality section)**

Sample Output (Your results may be different)

```
(base) nilchakraborttii@Administrator's-MacBook-Air Solution %
javac *.java
Note: Some input files use unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
(base) nilchakraborttii@Administrator's-MacBook-Air Solution %
java PerformanceComparison
Heaps
List Size: 1000
Time for inserting elements: 518833 nanoseconds
Time for deleting n/2 elements: 923625 nanoseconds
Time for searching n/2 elements: 3995625 nanoseconds

List Size: 10000
Time for inserting elements: 821291 nanoseconds
Time for deleting n/2 elements: 2098375 nanoseconds
Time for searching n/2 elements: 34254125 nanoseconds

List Size: 100000
Time for inserting elements: 5012084 nanoseconds
Time for deleting n/2 elements: 11928750 nanoseconds
Time for searching n/2 elements: 3038612083 nanoseconds

List Size: 500000
Time for inserting elements: 13279250 nanoseconds
Time for deleting n/2 elements: 63197750 nanoseconds
Time for searching n/2 elements: 82491973750 nanoseconds

AVL Trees
List Size: 1000
Time for inserting elements: 1236417 nanoseconds
Time for searching n/2 elements: 284750 nanoseconds

Time for deleting n/2 elements: 686416 nanoseconds
List Size: 10000
Time for inserting elements: 2517666 nanoseconds
Time for searching n/2 elements: 3151042 nanoseconds

Time for deleting n/2 elements: 3797167 nanoseconds
List Size: 100000
Time for inserting elements: 10078917 nanoseconds
Time for searching n/2 elements: 5506834 nanoseconds

Time for deleting n/2 elements: 927125 nanoseconds
List Size: 500000
Time for inserting elements: 36699208 nanoseconds
Time for searching n/2 elements: 13245958 nanoseconds

Time for deleting n/2 elements: 1313417 nanoseconds
```

```
Binary Search Trees
List Size: 1000
Time for inserting elements: 920042 nanoseconds
Time for searching n/2 elements: 366917 nanoseconds

Time for deleting n/2 elements: 360250 nanoseconds
List Size: 10000
Time for inserting elements: 1386292 nanoseconds
Time for searching n/2 elements: 3997667 nanoseconds

Time for deleting n/2 elements: 9102250 nanoseconds
List Size: 100000
Time for inserting elements: 8684500 nanoseconds
Time for searching n/2 elements: 6279959 nanoseconds

Time for deleting n/2 elements: 939750 nanoseconds
List Size: 500000
Time for inserting elements: 29274834 nanoseconds
Time for searching n/2 elements: 13822375 nanoseconds

Time for deleting n/2 elements: 1460041 nanoseconds
Queues
List Size: 1000
Time for inserting elements: 331958 nanoseconds
Time for searching n/2 elements: 2187042 nanoseconds

Time for deleting n/2 elements: 32791 nanoseconds
List Size: 10000
Time for inserting elements: 394583 nanoseconds
Time for searching n/2 elements: 20313708 nanoseconds

Time for deleting n/2 elements: 102292 nanoseconds
List Size: 100000
Time for inserting elements: 3372500 nanoseconds
Time for searching n/2 elements: 89024542 nanoseconds

Time for deleting n/2 elements: 278500 nanoseconds
List Size: 500000
Time for inserting elements: 3602584 nanoseconds
Time for searching n/2 elements: 467101500 nanoseconds

Time for deleting n/2 elements: 1349459 nanoseconds
Stacks
List Size: 1000
Time for pushing elements: 376416 nanoseconds
Time for searching n/2 elements: 5352625 nanoseconds
Time for popping n/2 elements: 32542 nanoseconds

List Size: 10000
Time for pushing elements: 389333 nanoseconds
Time for searching n/2 elements: 24010875 nanoseconds
Time for popping n/2 elements: 98833 nanoseconds

List Size: 100000
Time for pushing elements: 3202000 nanoseconds
Time for searching n/2 elements: 92018125 nanoseconds
Time for popping n/2 elements: 41 nanoseconds

List Size: 500000
Time for pushing elements: 2311959 nanoseconds
Time for searching n/2 elements: 454743708 nanoseconds
Time for popping n/2 elements: 42 nanoseconds



 Hash Tables
 List Size: 1000
 Time for inserting elements: 370500 nanoseconds
 Time for searching n/2 elements: 283084 nanoseconds

 Time for deleting n/2 elements: 47916 nanoseconds
 List Size: 10000
 Time for inserting elements: 23308583 nanoseconds
 Time for searching n/2 elements: 6719334 nanoseconds

 Time for deleting n/2 elements: 115083 nanoseconds
 List Size: 100000
 Time for inserting elements: 3029875459 nanoseconds
 Time for searching n/2 elements: 14622458 nanoseconds

 Time for deleting n/2 elements: 293709 nanoseconds
 List Size: 500000
 Time for inserting elements: 301926760625 nanoseconds
 Time for searching n/2 elements: 74388750 nanoseconds

 Time for deleting n/2 elements: 1536125 nanoseconds
```

Submission Guidelines:

- Make sure you completed all sections with “To do” statements.
- Make sure you have added JavaDoc in your code for each class you need to complete.
- Upon the completion of your lab, upload your submission to the Moodle course website as a single Zip file.
- See your TA and get your Assignment graded before leaving the lab. Otherwise, inform the instructor.
