TRINITY COLLEGE

DEPARTMENT OF COMPUTER SCIENCE

CPSC 215: Data Structures and Algorithms

Instructor: Dr. Chandranil Chakraborttii

Spring 2025

Assignment 7

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 this 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.

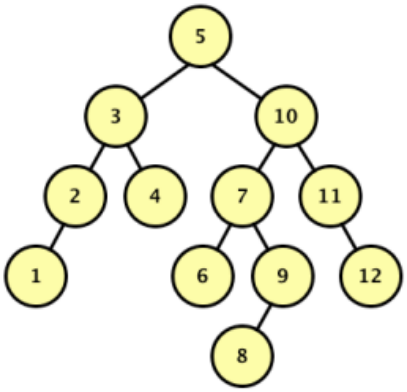
Objective:

- Understand the structure and properties of B trees.
- Explore different graph representations and graph traversals.

Part I: AVL Trees

(a) Build an AVL tree after inserting each of the following keys: 9, 27, 50, 15, 2, 21, and 36 are inserted, in that order, into an initially empty AVL tree. Show tree results after each insertion and make clear any rotations that must be performed at each step.

(b) Given the following AVL Tree:



Draw the resulting BST after 5 is removed, but before any rebalancing takes place. Label each node in the resulting tree with its balance factor. Replace a node with both children using an appropriate value from the node's left child.

(c) Now rebalance the tree that results from (i). Draw another new tree for each rotation that occurs when rebalancing the AVL Tree (you only need to draw one tree that results from an RL or LR rotation). You do not need to label these trees with balance factors.

Part II: Faster Sorting Methods: Galactic Sorting Challenge

Optimize three sorting algorithms (Quick Sort, Merge Sort, Radix Sort) to handle specific cosmic data patterns encountered in interplanetary databases. Implement an adaptive sorting system that automatically selects the best algorithm for different data characteristics.

Input Files:

You'll find these complete generic implementations in your starter code:

- **QuickSort.java** - Standard implementation
- **MergeSort.java** - Classic merge sort
- **RadixSort.java** - Base-10 implementation
- **GalacticSortingTest.java** - Main method to test implementations

Tasks:

Part A: Quantum Quick Sort - Modify Quick Sort to optimize for planetary system IDs (numbers with meaningful digit patterns)

Requirements:

- Review starter code
- Implement gravitational pivot selection
- Calculate "weight" as sum of digits (e.g., 142 → 1+4+2=7)
- Choose pivot as median of three elements' weights.

Part B: Nebula Merge Sort - Enhance Merge Sort for nebula density readings (floating-point numbers with clustered values)

Requirements:

- Review starter code and start with base **mergeSort()**
- Implement density-aware merging:
    - Calculate subarray density (average value)
- Implement Adaptive Merge Logic
    - Skip merge if adjacent subarrays are already ordered.

Part C: Galactic Radix Sort - Optimize Radix Sort for star classifications (hexadecimal values) and cosmic distances (large numbers with duplicates)

Requirements:

- Review starter code
- Implement base-16 sorting in **GalacticRadixSort.java**
- Handle negative numbers using offset adjustment
- Add duplicate optimization:

    private void handleDuplicates(int[] arr) { ... }

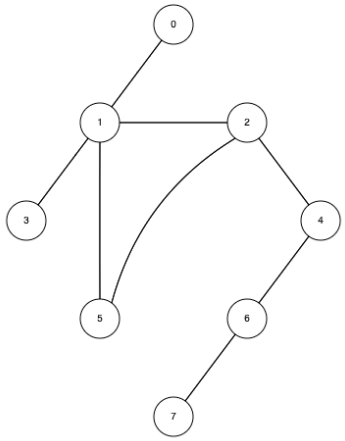
Part D: Cosmic Data Analysis - Test your implementation with **GalacticSortingTest.java**

Steps:

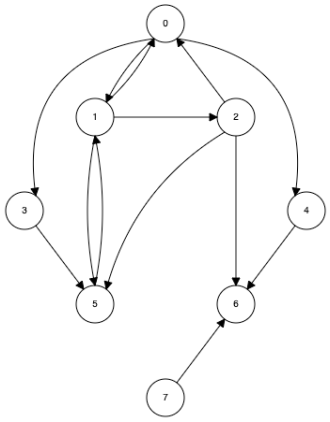
- Random assortment of 1 million celestial object IDs
- Nearly-sorted list of 500,000 planetary ages (in Earth years)
- Reverse-sorted list of 750,000 star temperatures
- List of 2 million galaxy distances with many duplicates (multiple galaxies at similar distances)
- For each algorithm and dataset, analyze:
- Execution time
- Number of comparisons/operations
- Efficiency in handling the specific data pattern.

Part III: Graph Traversals

1. Using an appropriate data structure, show the order in which nodes are visited during a **depth-first search (DFS)** traversal of a given graph, starting from node 1?



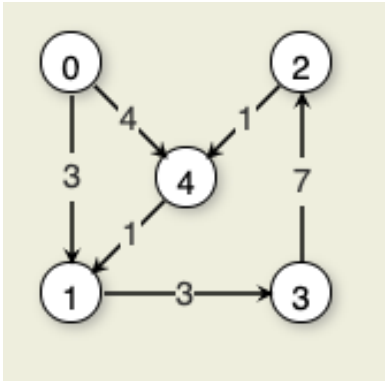
2. Using an appropriate data structure, show the order in which nodes are visited during a **breadth-first search (BFS)** traversal of a given graph, starting from node 0? **Which vertices cannot be reached by BFS?**



Part IV: Graph Representations

1. Represent the following graph using the following:
   (a) Adjacency Matrix
   (b) Adjacency Graph

2. (a) Assume that a vertex index requires two bytes, a pointer requires four bytes, and an edge weight requires two bytes. How much memory will be required to store the directed graph using an Adjacency Matrix and an Adjacency Graph?
   (b) Repeat assuming the graph is undirected.



Sample Output

```
(base) nilchakraborttii@nilchakraborttii's-MacBook-Air P3 % javac *.java
Note: MergeSort.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
(base) nilchakraborttii@nilchakraborttii's-MacBook-Air P3 % java GalacticSortingTest
=== GALACTIC SORTING CHALLENGE: PERFORMANCE ANALYSIS ===

Generating cosmic datasets...
All datasets generated successfully!

Testing algorithms on Random Celestial Objects...
  Nebula Merge Sort: 487ms, Comparisons: 18321952
  Galactic Radix Sort: 37ms
Testing algorithms on Planetary Ages...
  Nebula Merge Sort: 153ms, Comparisons: 7919704
  Galactic Radix Sort: 13ms
Testing algorithms on Star Temperatures...
  Nebula Merge Sort: 201ms, Comparisons: 7180608
  Galactic Radix Sort: 11ms
Testing algorithms on Galaxy Distances...
  Nebula Merge Sort: 871ms, Comparisons: 38732790
  Galactic Radix Sort: 22ms
(base) nilchakraborttii@nilchakraborttii's-MacBook-Air P3 %
```

Submission Guidelines:

- Use a pen and paper for this assignment (Recommended). Scan and upload your submission. **Make sure your writing is clear and easy to read.**
- Alternately, you may use an online format and submit a PDF.
- Upon the completion of your lab, arrange each section into individual folders (PI, PII, PIII), then compress these folders into a single Zip file. Upload the consolidated submission to the Moodle course website.
