# CS2710 - Programming and Data Structures Lab

Lab 10 (graded)

Oct 9, 2024

## Instructions

---

- You are expected to solve ALL the problems in the lab, using the local computer, C++ language, and g++ compiler. (Bonus problems can fetch 5% capped bonus, and Optional problems are just for fun and won't be graded.)
- You should submit your code to the course **moodle** on time, i.e., on/before 4.45pm (so that TAs can subsequently grade your submissions for graded lab assignments using the private test cases).
- You must strictly adhere to the following naming convention for your .cpp files and the single .zip file submission. For example, for a Lab Session 10 consisting of 2 questions, a student with roll number CS23B000 should
  - Name their .cpp files as **CS23B000_LAB10_Q1.cpp** and **CS23B000_LAB10_Q2.cpp**
  - Put both these .cpp files in a directory named **CS23B000_LAB10**
  - Zip this directory into a file named **CS23B000_LAB10.zip**
  - Submit only this single .zip file to moodle.
- If you need assistance, ask your TA, not your classmate.
- Internet access and mobile devices are prohibited in the lab.
- Check course Moodle for the public test cases and the evaluation script, which you can use to test your programs.
- Binary search tree is the primary data structure you would implement for the questions in this assignment. You can use string and vector data types for book-keeping or helper tasks. Please clarify with the TAs if you have any questions about which data structures to use or avoid.

---

1. [BINARY SEARCH TREE (BST) IMPLEMENTATION] You are tasked with implementing a Binary Search Tree (BST) as a C++ class. You should specifically implement the following operations as member functions:

   Insert(x): Insert a key x into the BST.

   Delete(x): Remove a key x from the BST. **Use In-order successor to replace the deleted element.**

   **Input Format:**

   - Each operation will either be an insertion **Insert X** or a deletion **Delete X**, where **X** is an integer value.
   - the End line will always be print for printing the BST in **POST ORDER**
   - there might be insertion of duplicate elements to or removal of non-existent elements from the BST; these operations should be ignored without any error messages (recall that keys within a BST are unique).

   **Output Format:**

   - Print one line representing **POST ORDER** traversal of the BST.

   **Constraints:**

   - number of operations $<=$ 1000.
   - value of nodes $\leq$ 1000.

   **Example:**
   **Input**1:
   Insert 7
   Insert 3
   Insert 5
   Delete 3
   Insert 6
   Insert 8
   Delete 9
   Insert 4
   print

   **Output**1:
   4 6 5 8 7

2. [BST FROM **PRE-ORDER** TRAVERSAL] You are provided with a sequence of integers representing the pre-order traversal of a BST. Your task is to:

   1. Construct the BST from the given **Pre-Order** traversal.
   2. Once the BST is constructed, output the **Post-Order** traversal of the tree.

   **Input Format:**

   - Number of integers in sequence
   - A sequence of integers (pre-order traversal) provided as input. All integers are distinct and represent nodes of a valid BST.

   **Output Format:**

   - The **Post-Order** traversal of the constructed BST.

   **Constraints:**

   - number of nodes $<= 1000$.
   - value of nodes $\leq 10000$.

   **Expected Complexity:**

   - Expected: $O(n)$ running time with $O(1)$ extra space for construction of BST, and $O(n)$ extra space for post-order traversal printing.
   - You can also provide implementations that are of different time complexity (like $O(n^2)$), but such solutions will only fetch you partial marks (i.e. for smaller test cases). Solutions with more than constant extra space for construction can also be provided, and again they will fetch only partial marks.

   **Examples:**

   **Input**:
   6
   10 5 1 7 40 50
   **Output**:
   1 7 5 50 40 10

3. [BONUS: HEIGHT-BALANCED BST FROM AN ARRAY] You are given an unsorted array of integers. Your task is to:

   Build a height-balanced (specifically, left-heavy i.e., "(height(leftSubtree) - height(rightSubtree)) $\leq 1$") BST from the elements in the array. A height-balanced BST is defined as a binary tree in which the depth of the two subtrees of every node never differs by more than one.

   Expected time complexity is O(nlogn) and Expected space complexity is O(logn)

   **Input Format:**

   - Number of integers in sequence
   - A single array of distinct integers. The array is not necessarily sorted.

   **Output Format:**

   - The **Pre-Order** traversal of the constructed height-balanced BST.

   **Constraints:**

   - number of nodes $\leq 1000$.
   - value of nodes $\leq 10000$.

   **Examples:**

   **Input**:
   7
   7 2 9 1 5 8 6

   **Output**:
   6 2 1 5 8 7 9

4. [OPTIONAL: QUICK INTERVAL REPORTING] Given a Binary Search Tree (BST) and two integer values $k_1$ and $k_2$, write a function that reports all values in the BST that lie between $k_1$ and $k_2$ (inclusive). The function should operate in $O(\text{height} + K)$ time, where $K$ is the number of keys reported. You can store additional information in each tree node, but the asymptotic running time of other operations like insert/remove should not be affected due to updation of this additional information during the operations.