# Department of Computing

# School of Electrical Engineering and Computer Science

**CS-250: Data Structure and Algorithms**

**Class: BSCS**

**Muhammad Haadhee Sheeraz Mian**

**478359**

# Assignment # 2

# Course Instructor: Dr. Farzana Jabeen

**Tasks : Binary search Tree implementation using Stacks**

**Note:**

* Submit code for the traversal of BST using Stack
* #include <iostream>
* #include <stack>
* using namespace std;
* // Node structure
* struct Node {
* int data;
* Node\* left;
* Node\* right;
* Node(int val) : data(val), left(NULL), right(NULL) {}
* };
* // In-order traversal using stack
* void inorderTraversal(Node\* root) {
* stack<Node\*> st;
* Node\* current = root;
* while (current != NULL || !st.empty()) {
* while (current != NULL) {
* st.push(current);
* current = current->left;
* }
* current = st.top();
* st.pop();
* cout << current->data << " ";
* current = current->right;
* }
* }
* // Helper to insert node in BST
* Node\* insert(Node\* root, int val) {
* if (!root) return new Node(val);
* if (val < root->data)
* root->left = insert(root->left, val);
* else
* root->right = insert(root->right, val);
* return root;
* }
* int main() {
* Node\* root = NULL;
* root = insert(root, 50);
* insert(root, 30);
* insert(root, 70);
* insert(root, 20);
* insert(root, 40);
* insert(root, 60);
* insert(root, 80);
* cout << "In-order Traversal (using stack): ";
* inorderTraversal(root);
* return 0;
* }

A screenshot of a computer screen

AI-generated content may be incorrect.

* Also calculate the complexity of the algorithm and compare it with linked list solution.

**Time Complexity:**

* **O(n)**, where n is the number of nodes.
* Every node is visited exactly once in the traversal.

**Space Complexity:**

* **O(h)**, where h is the height of the BST.
* Worst case (skewed tree): **O(n)**
* Best case (balanced tree): **O(log n)**
* Stack holds at most h nodes at a time.

| **Comparison table** | **BST (Using Stack)** | **Linked List (Traversal)** |
| --- | --- | --- |
| Structure | Hierarchical (Tree) | Linear |
| Time Complexity | O(n) | O(n) |
| Space Complexity | O(h) (stack) | O(1) |
| Traversal Type | In-order / Pre / Post | Single-directional |
| Use Cases | Sorted search, Range queries | Simple linear iteration |
| Insert/Delete Speed | O(log n) avg (BST) | O(1) insert/delete at head |
| Sorted Order Access | Efficient in BST | Inefficient in Linked List |

* **Identify the problems with this implementation as well.**

**Non-tail recursion overhead**:

* Recursive version may cause stack overflow for deep trees. Stack-based version solves this but still uses space proportional to tree height.

**Not memory-optimal**:

* Stack-based traversal is less space-efficient compared to Morris Traversal (which uses O(1) space with threaded trees).

**Imbalanced BSTs**:

* Time and space complexity degrade to O(n) for skewed trees (like linked lists).

**Limited to one traversal at a time**:

* Can’t traverse multiple parts simultaneously unless modified.

**More boilerplate code** :

* Stack-based traversal requires more boilerplate code compared to recursion.

Deadline : Sunday 3rd Aug, 2025