CS261L Data Structures and Algorithms (Pr)

Lab Manual (Week 10)

Instructor:

- Mr. Samyan Qayyum Wahla

Registration No._____

Name: _____

Guide Lines/Instructions:

You may talk with your fellow CS261-ers about the problems. However:

- Try the problems on your own *before* collaborating.
- Write up your answers yourself, in your own words. You should never share your typed-up solutions with your collaborators.
- If you collaborated, list the names of the students you collaborated with at the beginning of each problem.

Today's Task:

- Design of Data Structures(Binary Search Tree)

Part 1: Design of Data Structures(LinkedList, Stack Queue)

1. Implement **Binary Search Tree** class in C++ which must have following functions.

```
class BST {
public:
        BST(void);       // constructor
        BST(int arr[], int size);  // constructor to build tree from array, try to make it balanced by choosing
random root
        ~BST(void);      // destructor
        bool isEmpty() { return root == NULL; }
        Node* T getTree() { return root; }
        Node* insertNode(int x);          //insert in BST
        Node* findNode(int x); //search for data value x in the BST, if not found, return the last value before null
        bool deleteNode(int x); //delete all occurrences of x (use transplant as helping procedure in book)
        void inOrderTraversal (Node* T); //prints using in order traversal technique
        void preOrderTraversal (Node * T); //prints using in pre traversal technique
        void postOrderTraversal (Node* T); //prints using in post traversal technique
        int NumberOfNodes(Node* T); //recursive procedure to find number of nodes in tree T
        int Height(Node* T); //recursive procedure to calculate the height in tree T
        bool isBST(Node * T);
        void LeafNodes(Node* T); //print leaf nodes of the tree
        bool isSparseTree(Node *T); return tree in case tree is filled less than 50%
        void visualizeTree(Node * T); provide visualization of tree on console

private:
        Node* root;
};
```

```
class Node{
        int data;
        Node *parent;
        Node *left;
        Node *right;
};
```

Part 2: Application of Custom Data Structures for algorithms

1. Provide Console based application for the user to use the functions of the tree. You can design your own menu
2. Provide ability to save tree and then load from the same state.