Data Structure	Lab 8
Spring 2021	Friday, June 4

Lab Goals:

- Project Screens – HomePage (**Deadline**: 16th June, 2021)
- Git Understanding
- Pre-lecture Exercises
- Understanding and implementation of Binary Search Tree (**Deadline**: Friday, June 11)
- Homework Questions (**Deadline**: Friday, June 11)

Project Rubrics:

	Implementation
5	Code repository created on GitLab + Both members share the code via Git + Both members use commit/push and pull techniques for code sharing **+** Rubric IV requirements.
4	Properly formatted and visually pleasant UI/Template integrated **+** Rubric III requirements.
3	Home Screen with user interface and backend **+** Rubric II requirements**.**
2	Home Screen with user interface + Rubric I requirements
1	Login Screen with user interface created
0	No screen created

Lab Rubrics:

"No plagiarism" text is present in all rubrics.

	Implementation
5	Completed Q5 with expected outcome + Rubric IV requirements**.**
4	Completed Q4 with expected outcome + Rubric III requirements**.**
3	Completed Q3 with expected outcome + Rubric II requirements**.**
2	Binary Tree created via Linklist in C++ **+ All BST functions tested in main (code written/commented) +** Rubric I requirements**.**

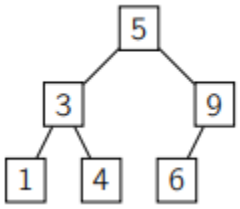
1	Only Node class and BST manually created in main. No separate class for BST.
0	Lab missed or solved none of the problems

Project Screens:

Students must create their HomePage screens of their project and get themselves evaluated in two weeks.

Pre-lecture Exercise:

In this pre-lecture exercise, we'll recall (or see for the first time) binary search trees (BSTs). Here is an example of a binary search tree:

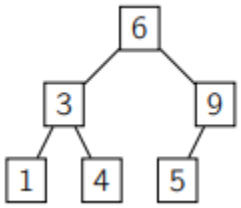


T is a binary search tree.

It is a binary tree, where each node has a key, that has the BST property:

For any node n, n.key is larger than all of the keys in the subtree under n's left child, and is smaller than all of the keys in the subtree under n's right child.

For example the following is not a BST (why not?):



T′ is a **NOT** a binary search tree.

If you have never seen binary search trees before, you might want to read up on them in CLRS ahead of time (Sections 12.1, 12.2, 12.3) before lecture. Answer the following exercises, which refer to the BST T drawn above.

1. Suppose you wanted to insert a node with key 7 into T. Where should it go? How about a node with key 8? Try to think about how you'd write an algorithm to insert a node with any given key into T.

2. Suppose you wanted to delete the node with key 4 from T. What would happen? How about the node with key 3? Try to think about how you'd write an algorithm to delete a node with any given key from T.

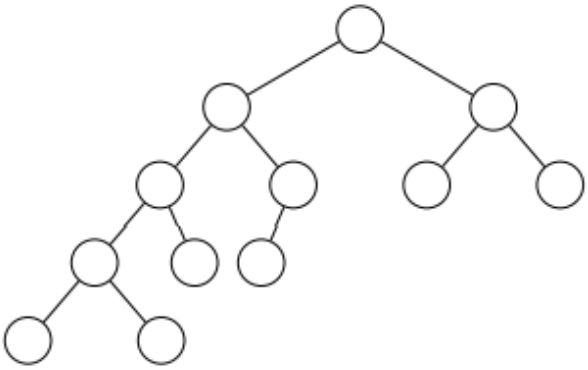
Homework Question:

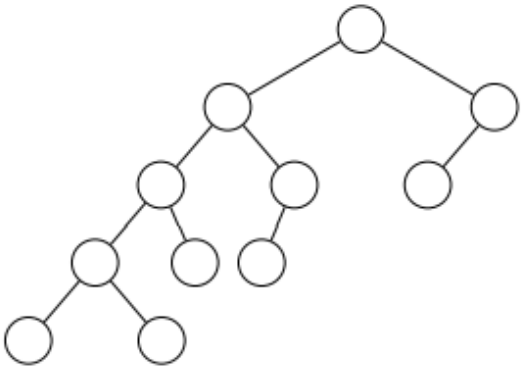
1. Implement **Binary Search Tree** class in C++ which must have following functions.

```
class BST {

public:

        BST(void);      // constructor

        ~BST(void);                             // destructor

        bool isEmpty() { return head == NULL; }

        Node* insertNode(int x);        //insert in BST

        bool findNode(int x);  //search for data value x in the BST

        bool deleteNode(int x); //delete all occurrences of x

        void inOrderTraversal (void); //prints using in order traversal technique

        void preOrderTraversal (void); //prints using in pre traversal technique

        void postOrderTraversal (void); //prints using in post traversal technique

private:

        Node* head;

};

class Node{

        int data;

        Node *left;

        Node *right;

};
```

2. For each of the following examples, if the nodes can be colored red or black to make a legitimate red-black tree, then give such a coloring. If not, then say that they cannot. (For reference, the LATEX code to make these trees is included at the end of the PSET, as well as instructions about how to color the nodes red or black. If you use this code, make sure

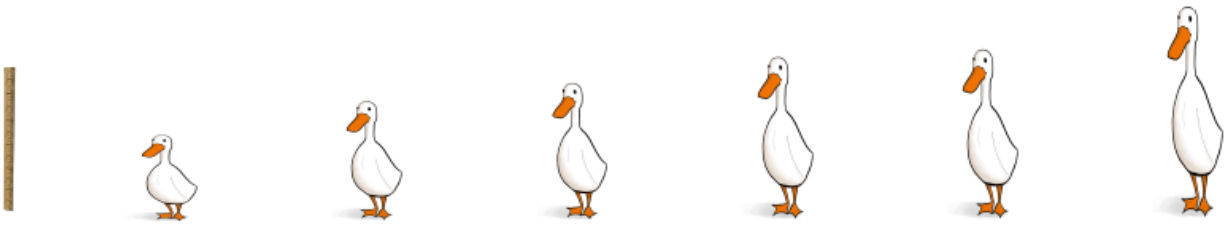
you include \usepackage{tikz} before the \begin{document} command. You are also welcome to take a screenshot and then color the trees in MSPaint, or make a hand-drawn copy and take a photo, or...)





[**We are expecting:** For each tree, either an image of a colored-in red-black tree or a statement "No such red-black tree." No justification is required.]

3. **The impossible job interview:** You're interviewing for your dream job at an ecological ethical tech company with healthy snacks. You already passed 28 stages of interviews, and your final interviewer asks you to design a binary search tree data structure that performs INSERT operations in $O(\sqrt{\log n})$ time using a comparison-based algorithm. Design such a data structure or prove that this is impossible. [**We are expecting**: If possible: An English description of the algorithm and a running time analysis. If impossible: A formal proof that this is impossible.]
4. Suppose that n ducks are standing in a line, ordered from shortest to tallest (not necessarily of unique height).



You have a measuring stick of a certain height, and you would like to identify a duck which is the same height as the stick, or else report that there is no such duck. The only operation you are allowed to do is compareToStick(j), where $j \in \{0, \ldots, n-1\}$, which returns taller if the j'th duck is taller than the stick, shorter if the j'th duck is shorter than the stick, and the same if the j'th duck is the same height as the stick. You forgot to bring a piece of paper, so you can only remember a constant number of integers in $\{0, \ldots, n-1\}$ at a time.

(a) Give an algorithm which either finds a duck the same height as the stick, or else returns "No such duck," in the model above which uses O(log(n)) comparisons. [We are expecting: Pseudocode AND an English description of your algorithm. You do not need to justify the correctness or runtime.]

(b) Prove that any algorithm in this model of computation must use $\Omega(\log(n))$ comparisons. [We are expecting: A short but convincing argument.]

5. [**Goose**!] A goose comes to you with the following claim. They say that they have come up with a new kind of binary search tree, called gooseTree, even better than red-black trees! More precisely, gooseTree is a data structure that stores comparable elements in a binary search tree. It might also store other auxiliary information, but the goose won't tell you how it works. The goose claims that gooseTree supports the following operations:

- gooseInsert(k) inserts an item with key k into the gooseTree, maintaining the BST property. It does not return anything. It runs in time O(1).
- gooseSearch(k) finds and returns a pointer to node with key k, if it exists in the tree. It runs in time O(log(n)).
- gooseDelete(k) removes and returns a pointer to an item with key k, if it exists in the tree, maintaining the BST property. It runs in time O(log(n)).

Above, n is the number of items stored in the gooseTree. The goose says that all these operations are deterministic, and that gooseTree can handle arbitrary comparable objects. You think the goose's logic is a bit loosey-goosey. How do you know the goose is wrong?

Notes:

- You may use results or algorithms that we have seen in class without further justification.
- Since the gooseTree is still a kind of binary search tree, you can access the root of gooseTree by calling gooseTree.root().

[**We are expecting:** Formally prove that the goose is wrong by showing that we can solve an algorithmic problem that we know the lower bound for with this data structure.]