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 Balanced BST

Part 1: Design of Data Structures(RB Tree)

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

   class **RbTree: BST** {

   public:

   RBTree(void);   // constructor

   RBTree(int arr[], int size);   // constructor to build tree from array

   ~ RBTree (void);   // destructor

   //override the following functions

   void visualizeTree(Node * T); provide visualization of tree on console

   Node* Insert(int x)

   Node * Delete(int x)

   private:

   Node* root;

   };

   class Node{

   int data;

   Node *parent;

   Node *left;

   Node *right;

   bool color; //1 for red, 0 for black, only use it for RB Tree

   };

2. Implement **AVL Tree** class in C++ which must have following functions.

   class **AvlTree: BST** {

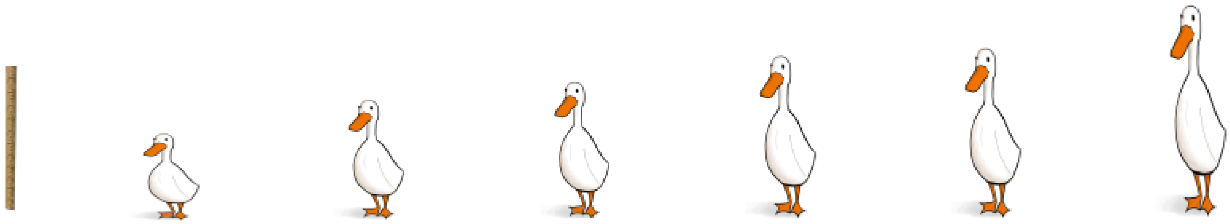
   public:

```
        AvlTree(void);  // constructor
        AvlTree (int arr[], int size);        // constructor to build tree from array
        ~ AvlTree (void);          // destructor
        //override the following functions
        void visualizeTree(Node * T); provide visualization of tree on console
        Node* Insert(int x)
        Node * Delete(int x)

private:
        Node* root;
};
```

Part 2: Application of Custom Data Structures for algorithms

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 run 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.]