Data Structure and Algorithms

Lab8

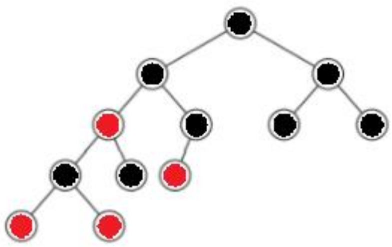
Submitted to:	**Ms. Sehar Waqar**
Submitted by:	**SABA**
Roll no:	**2019-CE-04**
Date:	**11June,2021**

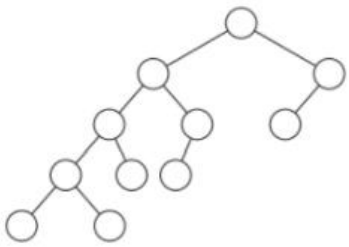
Department of Computer Engineering

UET,Lahore

Question no 2:

For each tree, either an image of a coloured-in red-black tree or a statement "No such red-black tree." No justification is required.





No such red-black tree.

Question no 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($\sqrt{logn}$) 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.]

Ans: It's impossible to design a binary search tree data structure that performs INSERT operations in O($\sqrt{logn}$) time using a comparison-based algorithm.

Algorithm:

```
Node* insertNode(Node* root, int key){
        Node *y = NULL;
        Node *x = root;
        Node *new_node = new Node();
        new_node-> key = key;
        new_node->left = NULL;
        new_node-> right = NULL;
```

```
while (x != NULL){
        y = x;
        if (key < x->key){
            x = x->left;}
        else{
            x = x->right;}
 }
 if (y==NULL){
        root = new_node; }
 else if (key < y->key){
        y->left = new_node; }
  else {
         y->right = new_node; }
  return root;
}
```

Running Time: It takes O(logn) time equals to height of the tree.

Question no 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.]

Answer:

```
def compareToStick(X):
    stick_height = 10;
    if X == stick_height:
        return "same";
    if X > stick_height:
        return "taller";
    if X < stick_height:
        return "smaller";
```

```
def ducks_heigh (ducks):
    n = len(ducks)
    if (n == 1):
        if compareToStick(ducks[0]) == "same":
            return ducks[0]
    else:
        return "No such duck"
    if (n == 0):
        return "No such duck"
    L = ducks[:round(n/2)]
    R = ducks[round(n/2):n]
    mid = round(n/2)
    if compareToStick(ducks[mid]) == "same":
        return ducks[mid]
    if compareToStick(ducks[mid])== "taller":
        return ducks_heigh (L)
    if compareToStick(ducks[mid]) == "smaller":
        return ducks_heigh (R)
```

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

Answer:

In each step, this algorithm eliminates half from the array of ducks.

Question no 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().

<u>Answer:</u>

If we want to insert a key in BST, it takes O(logn) time in average case as I described before and O(n) in worst case. But goose claims that gooseInsert(k) takes O(1) time which is wrong. While running time of our algorithms and gooseSearch(k) and gooseDelete(k) takes the same time. So, goose claims true in case of search and delete operations.
