Exam 2 Summer 2021 Solutions

Q1. True/False and Multiple-Choice

Q1.1. Suppose we have the following operations on a `Stack`:

```
push(10); push(3); push(47); pop(); push(3); pop(); pop(); push(1); pop();
```

If every pop operation prints the corresponding value to the console, what should be printed? 47, 3, 3, 1

Q1.2. In the worst case, a BST can become a linked list. True/False

Q1.3 For a binary tree, its preorder traversal will always be different from its postorder traversal. True/False

Q1.4. All nodes in the left subtree of a binary tree are less than all nodes on the right subtree. True/False

Q1.5. Red-Black tree is NOT a binary search tree. True/False

Q1.6. SKIPPED

Q2.

The following code does not compile. What are the problems and how can you fix them? Assume all necessary functions have been implemented.

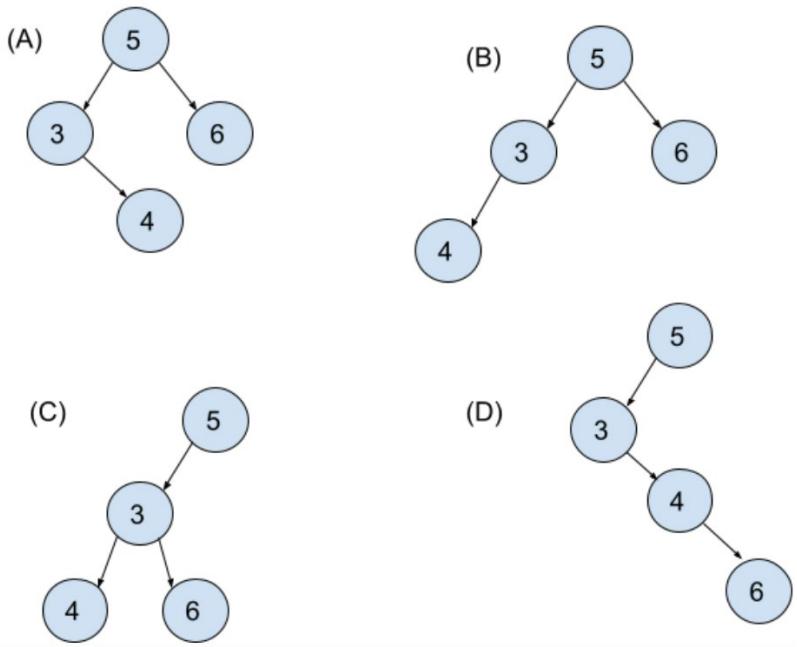
```
public interface Student { ... }

public interface TA extends Student {
    private int hoursPerWeek = 20;
    public void teach();
}

public class SummerTA extends TA {
    ...
    public void teach() { ... }
}
```

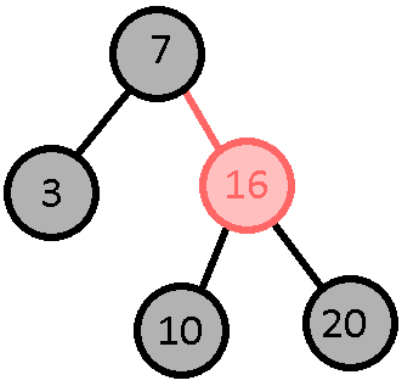
1. `hoursPerWeek` cannot be private. Variables in an interface can only be public, final, or static only.
2. `SummerTA` cannot extend `TA` because `TA` is an interface. `SummerTA` must implement `TA`.

Q3.



Which BST corresponds to the preorder traversal of 5, 3, 4, 6? A

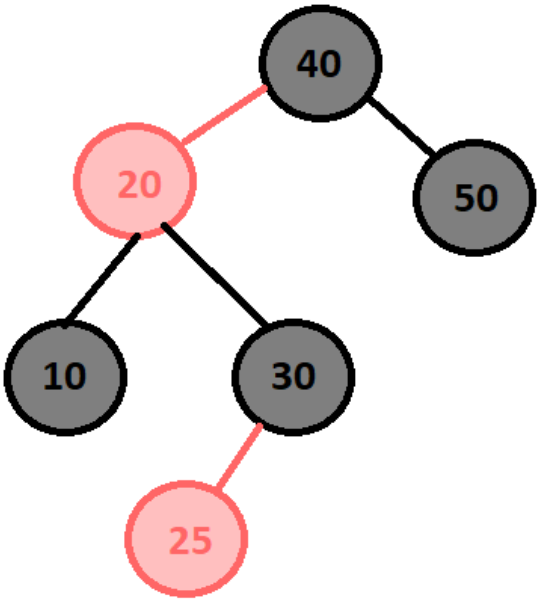
Q4.



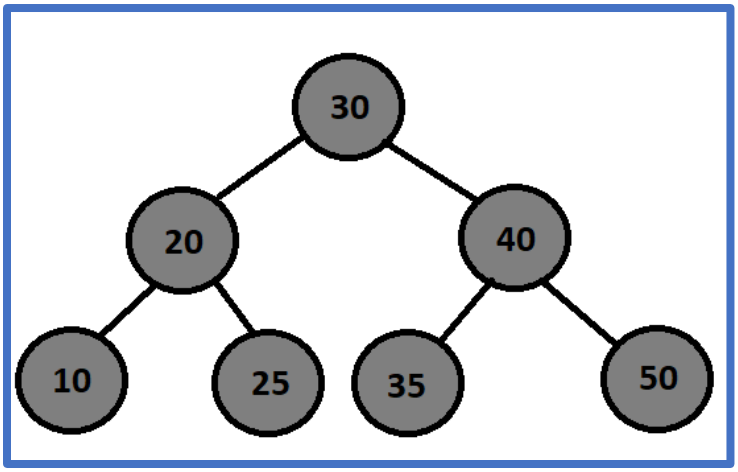
What's the issue with the following Red-Black Tree? Right-learning red edge

Q5.

Consider the Red-Black Tree given below:



Suppose we want to insert `35`. After performing the necessary rotations and color swaps, what does our Red-Black Tree look like?



Q6.

What does `foo(Node r)` do?

```
void foo(Node r) {
    if (r == null) return;
    Queue<Node> q = new Queue<Node>();
    Stack<Node> s = new Stack<Node>();
    q.enqueue(r);
    while (!q.isEmpty()) {
        Node t = q.dequeue();
        s.push(t);
        if (t.left != null) q.push(t.left);
        if (t.right != null) q.push(t.right);
    }
```

```
    while (!s.isEmpty()) {
      System.out.print(s.pop());
    }
}
```

Prints each key in a bottom-up level-order traversal of a tree.

Q7. Binary Tree Traversal

Suppose you create a Binary Search Tree by inserting elements 15, 18, 8, 10, 20, 7, 16, 25 in this order. Write the preorder, inorder, postorder traversal for the resultant tree.

Preorder: 15, 8, 7, 10, 18, 16, 20, 25

Inorder: 7, 8, 10, 15, 16, 18, 20, 25

Postorder: 7, 10, 8, 16, 25, 20, 18, 15

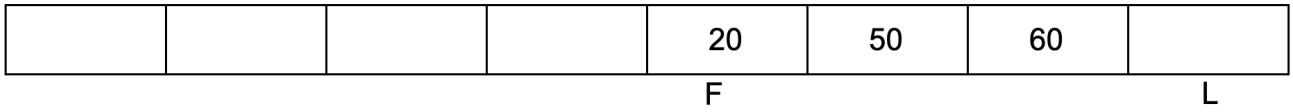
Q8.

```
int doMagic(int a) {
    if (a <= 0) {
        return 0;
    } else {
        return doMagic(a-1) + doMagic(a-2) + 2*a;
    }
}
```

What is the output of `doMagic(3)`? 14

Q9.

Observe the following circular array queue:



If 20 is the first item in the queue and 60 is the last item in the queue, give the values in the queue and where the `front` and `end` of the queue are after we perform the following operations:

`enqueue(10); enqueue(15); dequeue(); dequeue();`

A	B	C	D	E	F	G	H

A. 15
B. L
C.
D.
E.
F. 60, F
G. 10

Q10.

```
//Node Class
public class Node {
    public int data;
    public Node left, right;
}

/**
* This method will return the number of nodes in a binary tree
* @param root of the tree.
* Note: DO NOT create a helper function
*/
public int size(Node root) {
    if(root == null) {
        return 0;
    } else {
        return 1 + size(root.left) + size(root.right);
    }
}
```

Q11. Print a Binary Tree

```
//Tree Class
public class Tree {
    public class TreeNode {
        public TreeNode left;
        public int data;
        public TreeNode right;
    }

    private TreeNode root;

    public TreeNode getRoot() {
        return this.root;
    }

/**
    5
   / \
  3   7
 / \ / \
1  4 6  8

* This method will return a string of all the data in the tree in ascending order.
* For example, if t was the tree above, it would return `1345678`
*/
 public String inOrderToString() {
    return inOrderToString(root);
 }

 private String inOrderToString(TreeNode n) {
```

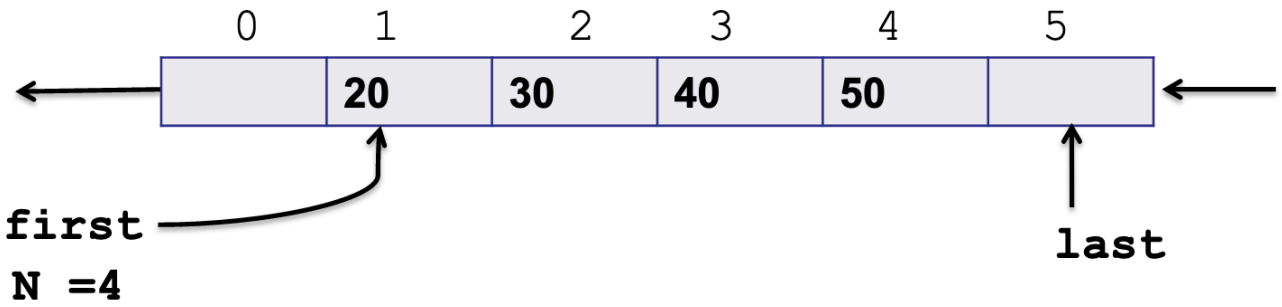
```
    if(n == null) {
        return “”;
    }
    return inOrderToString(n.left) + n.data + inOrderToString(n.right);
 }
}
```

Q12.

SKIPPED

Q13.

Below is the code for enqueuing an item into a circular array-based queue. `N` is the number of items currently on the queue, `q` is the array representing the queue, `last` is the index of the next free position of the queue and `resize()` doubles the size of the queue and copies the contents over into the new array, maintaining queue order and setting `first` and `last` appropriately. You should only resize when the circular array is full. Here is the queue after we enqueue 10, 20, 30, 40, 50, and dequeue one item.



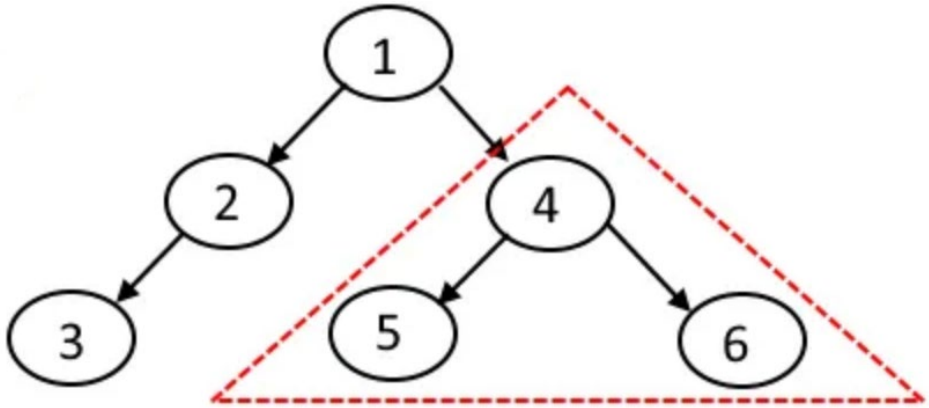

```
public void enqueue(E item) {
    //Check if resize is required
    if (N == q.length) {
      resize();
    }
    //Add the item to the queue
    q[last++] = item;

    //Wrap around
    if (last >= q.length) {
      last = 0;
    }

    N++;
}
```

Q14.

Here's a function that deletes the subtrees of a BST node whose data is equal to the given key. For example: `setNull(4)` will delete the subtrees 5 and 6.



Assume keys are unique.

```
public class Node {
    public Node left;
    public int data;
    public Node right;
}

public void setNull(int key) {
    root = setNull(root, key);
}
private Node setNull(Node r, int key) {
    if (r ==  null)
        return null;
    if (key == r.data) {
        return null;
    } else if (key < r.data) {
        r.left = setNull(r.left, key);
    } else {
        r.right = setNull(r.right, key);
    }
    return r;
}
```