COMP 203 DATA STRUCTURES AND ALGORITHMS

HOMEWORK 4 (Total=100 points)

Deadline: 12.1.2024 23:59

Read the questions and rules carefully. They are clear and well defined.

Rules:

1. **No Cheating:** You are not allowed to collaborate with your friends and use any kind of websites or AI. If your homework gives a sign of any of them, **directly it will be graded as zero**.
2. **Goal:** Please do your homework alone. Our main aim is to **learn** whatever we cover so far.
3. **Submission:** Submit your homework in **2 java files**. **No other file types will be accepted. You will submit only 2 java files. DON'T USE ZIP/RAR etc. In these cases, your points will be deducted by 30%.**
4. **Coding policy:** Explain your code in comments. **This is a must!**
5. **Latency policy:** A 10% deduction will be applied for each day of late submission.

SOLUTION

1.

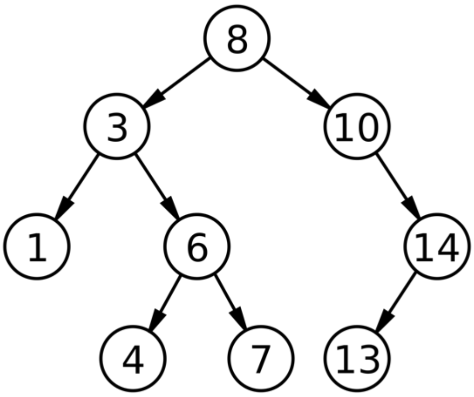
```
public class FixedSizePriorityQueue<T extends Comparable<T>> { //2pt
    private Object[] array;
    private int size;
    private int capacity;

    public FixedSizePriorityQueue(int capacity) { //3pt
        this.array = new Object[capacity];
        this.size = 0;
        this.capacity = capacity;
    }

    public void enqueue(T addedValue) { //10pt
        if (size == capacity) {
            System.out.println("Priority Queue is full. Cannot enqueue element: " + addedValue);
            return;
        }
    int index = findInsertIndex(addedValue);

    // Shift elements to make space for the new element
    for (int i = size - 1; i >= index; i--) {
        array[i + 1] = array[i];
    }

    array[index] = addedValue;
    size++;
}

public T dequeue() { //10pt
    if (size == 0) {
        System.out.println("Priority Queue is empty. Cannot dequeue.");
        return null;
    }

    T dequeuedElement = (T) array[0];

    // Shift elements to fill the gap left by dequeue
    for (int i = 1; i < size; i++) {
        array[i - 1] = array[i];
    }

    size--;

    return dequeuedElement;
}

public int size() { //5pt
    return size;
}

public void printPriorityQueue() {//5pt
    System.out.print("Priority Queue: ");
    for (int i = 0; i < size; i++) {
            System.out.print(array[i] + " ");
        }
        System.out.println();
    }

    private int findInsertIndex(T addedValue) {
        int index = 0;
        while (index < size && ((Comparable<T>) array[index]).compareTo(addedValue) < 0) {
            index++;
        }
        return index;
    }

    public static void main(String[] args) {
        // Creating the given Priority Queue //5pt
        FixedSizePriorityQueue<Integer> priorityQueue = new FixedSizePriorityQueue<>(6);

        priorityQueue.enqueue(4);
        priorityQueue.enqueue(5);
        priorityQueue.enqueue(6);
        priorityQueue.enqueue(7);
        priorityQueue.enqueue(8); //2.5pt
         priorityQueue.enqueue(9);

        // Testing the methods
        priorityQueue.printPriorityQueue();//2.5pt

        System.out.println("Priority Queue size: " + priorityQueue.size()); //2.5pt
        System.out.println("After dequeue: ");

        int q= priorityQueue.dequeue();//2.5pt
         priorityQueue.printPriorityQueue();
        }

}
```

PQ: 4,5,6,7,8,9.

Include comments of your code for each method and class.

Submit FixedSizePriorityQueue.java to Canvas.

2.



Include comments of your code for each method and class.

Submit BinaryTree.java to Canvas.

SOLUTION

```
class Node { //2pt
    int data;
    Node left, right;

    public Node(int data) {//3pt
        this.data = data;
        this.left = this.right = null;
    }
}

class BinaryTree {//2pt
    Node root;
```

```
public BinaryTree() {//3pt
    this.root = null;
}

public boolean isHeap(Node root) { //25pt
    if (root == null) {
        return true;
    }

    // Check the heap property at the current node
    if ((root.left != null && root.left.data > root.data) ||
            (root.right != null && root.right.data > root.data)) {
        return false;
    }

    // Recursively check the heap property for left and right subtrees
    return isHeap(root.left) && isHeap(root.right);
}

public static void main(String[] args) {
    // Creating the given binary tree //10pt
    BinaryTree tree = new BinaryTree();
    tree.root = new Node(8);
    tree.root.left = new Node(3);
    tree.root.right = new Node(10);
    tree.root.left.left = new Node(1);
    tree.root.left.right = new Node(6);
    tree.root.left.right.left = new Node(4);
    tree.root.left.right.right = new Node(7);
    tree.root.right.right = new Node(14);
    tree.root.right.right.left = new Node(13);
    // Testing the isHeap method
    if (tree.isHeap(tree.root)) { //5pt
        System.out.println("The given binary tree is a heap.");
    } else {
        System.out.println("The given binary tree is not a heap.");
        }
    }
}
```