EECS 2011: Assignment 3

Due: as set in *eClass.*

May be done in groups of up to three students (from either of the sections M and Z)

Motivation

The purpose of this assignment is to evaluate two implementations of binary search trees in terms of their performance for different insertion and deletion operations. The trees will then be tested to implement a *TreeSort* sorting algorithm.

Introduction

In computer science, binary search trees (BST) are a particular type of container: data structures that store "items" (such as numbers, names etc.) in memory. They allow fast lookup, addition and removal of items, and can be used to implement either dynamic sets of items, or lookup tables that allow finding an item by its key.

The included `Tree` interface provides three methods to **add** and **remove** elements to and from the tree. It also provides an iterator that visits the elements in-order, as well as a function `height` that simply returns the height of the tree.

Note that the speed of these operations may strongly depend on the implementation.

Description

In this assignment, you will have to write two implementations of `Tree` interface (provided), one that uses regular (possibly unbalanced) **Binary Search Trees**[1], and one that uses **balanced AVL Trees**[2]. After that, you will have to test the performance of TreeSort[3] when using your implementations. For your convenience, some starting code is provided. Note that it may either not implement some of the required features, or implement them improperly.

Part 1

Implement the necessary public methods in the two implementations (called `A3BSTree` and `A3AVLTree`) of `Tree` interface[4]:

```
public boolean add(E e);

public boolean addAll (Collection<? extends E> c);

public boolean remove(Object o);

public boolean contains(Object o);
```

---

[1] https://en.wikipedia.org/wiki/Binary_search_tree

[2] https://en.wikipedia.org/wiki/AVL_tree

[3] https://en.wikipedia.org/wiki/Tree_sort

[4] The interface effectively [partially] describes a *sorted set* ADT

```
public Iterator<E> iterator();

public int height();

public int size();
```

The classes should use generics. For AVL trees, the complete implementation of the *remove* operation is not required (no rebalancing is needed; the AVL properties may be violated after remove calls). The summarized descriptions of the AVL add and remove algorithms are included for your convenience. You may assume the elements inserted into the tree are going to be `Comparable`.

Of course, you are free to implement any private or protected methods and classes as you see fit. However, you may not have any *public* methods, other than the ones mentioned (or the ones present in the interface or its super classes).

Part 2

Name your class `TreeSort`. The implementation is probably going to resemble that of the priority queue sort of the previous assignment, and will be very short. Note that since your tree (essentially, a set) may not have duplicates, the sorting operation is not going to be defined for arrays with duplicate items[5].

The class should have the following two public methods:

```
static <E extends Comparable <? super E>> void sort (E[] a)

static <E extends Comparable <? super E>> void sort(Tree <E> t, E[] a)
```

See the comments in the code provided to determine their behaviour.

Part 3

Nothing needs to be submitted for this part.

Create some tester class and use it with your tree implementation to investigate its running time complexity as the number of items you insert into the tree increases. Try using values from 10 to 1,000,000 while measuring the running time. Confirm that the running time follows the expected logarithmic behaviour for both `add` and `remove` methods when the items are inserted in random, increasing, and in decreasing order.

NOTES:

1. Do not use *package*-s in your project. Using packages will cost you a 10 % deduction from the assignment mark.
2. Some aspects of your code will be marked automatically (e.g., how it handles boundary cases and error conditions), using a custom tester class. It is also imperative you test your classes. If any of the java files that you submit do not

[5] There are easy way to extend trees to allow to contain duplicates, but this is not going to be required for this assignment.

compile, the whole submission will be given a grade of zero, regardless of how trivial the compiler error is. The code you submit should compile using

```
javac *.java
```

command in either Windows, Linux, or macOS.

3. Your code should include Javadoc comments.
4. Using any `java.util` implementations of `Collection`[6] or `Map`[7] is not allowed (importing the `java.util.Collection` interface is of course fine).

Submission

Find all the `java` files in your project and submit them electronically via eClass (**do not** zip, tar, rar, or 7z them). Only three files are expected, but you may write other classes, if you deem them necessary.

If working in a group, **make only one submission** per group and include a **`group.txt`** file containing the names and the student numbers of the group members. The deadline is firm. Contact your instructor *in advance* if you cannot meet the deadline explaining your circumstances.

Grading

The assignment will be graded using *the Common Grading Scheme for Undergraduate Faculties*[8]. We look at whether the code passes the unit tests, satisfies the requirements of this document, and whether it conforms to the code style rules.

Academic Honesty

Academic honesty will be strictly enforced in this course. Specifically, direct collaboration (e.g., sharing code or answers) is not permitted, and plagiarism detection software will be employed. You are, however, allowed to discuss the questions, ideas, or approaches you take.

Cite all sources you use (online sources – including web sites, old solutions, books, etc.) in your code comments; using textbook examples is allowed, but these must be cited as well.

E.g.,

```
1) /**
     * Constructs an empty list with the specified initial capacity.
     *
     * @param   initialCapacity   the initial capacity of the list
```

[6] https://docs.oracle.com/javase/8/docs/api/java/util/Collection.html

[7] https://docs.oracle.com/javase/8/docs/api/java/util/Map.html

[8] https://secretariat-policies.info.yorku.ca/policies/common-grading-scheme-for-undergraduate-faculties/

```
     * @exception IllegalArgumentException if the specified initial
capacity
     *            is negative
     * uses code from www.xxx.yyy.edu/123.html for initialization
     *
     */


2) //formula based on Jane Smart's thesis, page XX, available at
https://...
        a = b + c;
```

Although using outside sources is allowed – with proper citing, if the amount of non-original work is excessive, your grade may be reduced. You might find the following article useful to have an idea what kind of collaboration is appropriate: https://en.wikipedia.org/wiki/Clean_room_design

You may not post the contents of this assignment, nor solicit solutions on any external resources.