# Department of Computer Science and Software Engineering
# Concordia University
## COMP 352: Data Structure and Algorithms
## Fall 2017
## Assignment 3
## Due date and time: Monday November 13th, 2017 by midnight

**Written Questions (50 marks):**

### Question 1

a) Draw a single binary tree that gave the following traversals:
   Inorder: T N C K V A S M W Q B R L
   Postorder: T C N V S A W M B Q L R K

b) Assume that the binary tree from the above part (a) of this question is stored in an array-list as a complete binary tree as discussed in class. Specify the contents of such an array-list for this tree.

### Question 2

a) Draw the min-heap that results from running the bottom-up heap construction algorithm on the following list of values:
12 24 15 17 16 13 11 18 11 14 21 4 8 3 12.
Starting from the bottom layer, use the values from left to right as specified above. Show intermediate steps and the final tree representing the min-heap. Afterwards perform the operation removeMin 4 times and show the resulting min-heap after each step.

b) Create again a min-heap using the list of values from the above part (a) of this question but this time you have to insert these values step by step using the order from left to right as shown in the above question. Show the tree after each step and the final tree representing the min-heap.

### Question 3

Give an algorithm for computing the depths of all the nodes of a tree T, where *n* is the number of nodes of *T*.

a) What is the complexity of your algorithm in terms of Big-O?
b) What is the best possible complexity *that you believe* can be achieved when solving such problem? Explain why.
c) If your algorithm does not achieve this best complexity, can you design a better algorithm to achieve such complexity! If so, give this algorithm as a second solution.

**Programming Questions (50 marks):**

The purpose of the programming questions in 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.

Binary Search Trees (BST) are 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 `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.
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 either does not implement some features or implements them improperly.

**Question 1:**

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

```
public void add(E e);
public void addAll (Collection<? extends E> c);
public boolean remove(Object o);
public Iterator<E> iterator();
public int height();
public int size();
```

The classes should use generics. The AVL *`add`* and *`remove`* operation should keep the tree balanced. It should also be possible to have duplicate items in the trees (something that binary search trees normally do not allow); think about how you can work around it.
You are free to implement any private or protected methods and classes as you see needed. However, you may not have any public methods other than the ones mentioned (or the ones present in the interface or its super classes).

**Question 2**

Name your class `TreeSort.`
The class should have the following methods:

```
public static <E> void sort( E[] a);
public static <E> void sort(Tree <E> tree, E[] a);
```

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

[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

**Question 3:**

Name your class `SortTester`
Take both of your tree implementations and compare them when used to implement a TreeSort sorting algorithm.

For numbers $N$ = {10, 100, 1000, 10000, 100000, 1000000}

a) Starting with **unsorted** *arrays* of *N*, measure how long it takes to sort such an array using regular BSTs and balanced AVL trees.

b) Starting with **reverse-sorted** *arrays* of *N*, measure how long it takes to sort such an array using regular BSTs and balanced AVL trees.

Produce the following table (the timing values below are just an illustration and do not relate to any real measurements):

```
N = 10:
BST                 213 ms
AVL                 432 ms
BST(rev-sorted)     543 ms
AVL(rev-sorted)     876 ms

N = 100:
…
N = 1000:
…
<repeat for all values of N>
```

Save the result of your program execution in a file `testrun.txt` and submit it together with your other files.

**Important Requirements:**

1. Make sure you reset the timer (or save the intermediate time before the next measurement); i.e., make sure your measured time contains only the time to perform one set of operations that was supposed to be timed.

2. In case the operations for big N numbers take too long (e.g., more than 50 s) you may reduce the number to a smaller one or eliminate it (so that you will have a range from, say, 1 to 100000).

3. Do not use `packages` in these programming questions of your assignment (put your classes in a `default` package). Using packages will cost you a 40 % deduction from the assignment mark.

4. Name your classes as specified. Using incorrect names will cost you a 20 % deduction from the assignment mark.

5. Your code should handle boundary cases and error conditions. It is also imperative that you test your classes. If any of the java files that you submit do not compile, the whole submission will be given a grade of zero, regardless of how trivial the compiler error is.

6. In this programming assignment, you are required to submit the commented Java source files, the compiled files (.class files), and the testrun text files.

**Submission:**

- The **written questions part** must be done **individually** and uploaded to "theory assignment 3" via EAS, name your file *A3_studentID. pdf*, where *studentID* is your ID number. For this third assignment, student 123456 would submit a file named A3_123456.pdf. All your answers to written questions must be in PDF (no scans of handwriting) or text formats only. **Please be concise and brief (about ¼ of a page for each question).**
- The **programming part** must be done **in a team of 2 students max,** For the Java programs, you must submit the source files together with the compiled files. The solutions to all the programming questions should be zipped together into one .zip or .tar.gz, name your file *A3_studentID1, studentID2, studentID3. zip*, where *studentID1, studentID2, studentID3, are the IDs of the three students who did the programming part together,* uploaded to "Programming assignment 3" via EAS.

**Note**: **Assignment not submitted by the due date and in the correct format will not be graded – NO EXCEPTIONS!!!!**