CMPT225, Spring 2025
Homework Assignment 3
Due date: Friday, October 31, 2025, 23:59

You need to implement the following classes:
Part 1:
- *sorting.MySortingAlgs.java*
Part 2:
- *binarytree.BTNode.java*
- *binarytree.BinaryTree.java*
You may add more classes to your solution if necessary.
Note that all files in your solution must be under the correct package (folder)**.**

Submit all your files in **assignment3.zip** to CourSys. Make sure your zip file can be unzipped using the command "*unzip assignment3.zip*" in CSIL. All your solutions in the zip file must be under the **src** folder.
1. See the provided example for the expected format.
2. Use the provided *check_format.sh* to check that your zip has the expected folder structure.

Compilation: Your code MUST compile in CSIL using javac.
Make sure that your code compiles without warnings/errors.
If the code does not compile in CSIL the grade on that part will be 0 (zero).
Even if you can't solve a problem completely, make sure it compiles.

The assignment will be graded mostly **automatically**, with some exceptions.

Discussion with others: You may discuss the assignment with your classmates/tutors (or anyone else), but coding must be entirely your own.

References: You may use textbooks, wiki, stack overflow, geeksforgeeks, etc. If you do, please specify the references in comments. Asking others for solutions (online or in person) is prohibited.

Readability: Your code should be readable using the standard Java conventions. Add comments wherever is necessary. If needed, write helper functions or add classes to improve readability.

Do not add main() to your solutions. The main() method will be in the test files.
Examples of such tests are provided in *TestSorting.java* and *TestBinaryTree.java*.

Warnings: Warnings during compilation will reduce points. More importantly, they indicate that something is probably wrong with the code.

Testing: Test your code. Examples of tests are included. Your code will be tested using the provided tests as well as additional tests. You should create more tests to check your solution.

Good luck!

Part 1 [75 points] - the class BinaryTree

Write the following methods for the class *BinaryTree*.

[10 points] `public int numberOfLeaves()`
Returns the number of leaves in the tree.

[15 points] `public boolean equals(Object other)`
Overrides the equal method. The method checks if this tree is equal to the other tree.
That is, it checks that *other* is a BinaryTree with the same tree structure, and all data is the same, comparing data using == operator.

[20 points] `public int countDepthK(int k)`
Returns the number of nodes at distance k from the root.

[30 points] `public Iterator<T> preOrderIterator()`
The method returns an iterator that performs the preorder traversal on the tree.

The iterator must be dynamic in the following sense: if after the iterator is created, and the tree changes in some part that has not been processed by the iterator yet, then the iterator "will see" these changes and output the values in the updated tree when reaching that part of the tree. However, if you change the nodes that have already been processed, then it will have no effect on the iterator.
** Be careful, do not change the nodes that are currently being processed, as this may have unexpected effects

This means you cannot simply create the list with the preorder traversal of the tree in the constructor of the iterator. It will not work in the dynamic sense.

You will probably need to implement the iterator in a new class. Please declare the class in the package **binarytree**. Make sure to submit the java file implementing the iterator.

Part 2 [25 points] - Sorting algorithms

Implement the following methods in the class *sorting.MySortingAlgs.*

[10 points] `public static void sortStrings(String[] a)`
    The method gets an array of strings and sorts it. In the sorted array the strings are arranged
    as follows:
    - Shorter strings need to come before the longer strings
    - For two strings of equal length, use the function String.compare() to decide which
      one needs to come before the other.
    - Specifically, if str1.compare(str2)<0, then str1 should be before str2 in the sorted
      array.
    - If str1.compare(str2)>0, then str2 should be before str1 in the sorted array.
    One way to solve this problem is to use the method
    `Arrary.sort(T[] a, Comparator<? super T> c)`, which sorts the given array
    according to the order induced by the specified comparator.
    https://docs.oracle.com/javase/8/docs/api/java/util/Comparator.html

[15 points] `public static <T extends Comparable<T>>`
                `void merge(T[] a, int start, int mid, int end)`
    The method gets an array of objects of type T, such that
    - arr[start...mid-1] is sorted and
    - arr[mid...end] is sorted
    The method merges the two parts into a sorted subarray arr[start...end]
    All elements outside arr[start...end] remain unchanged