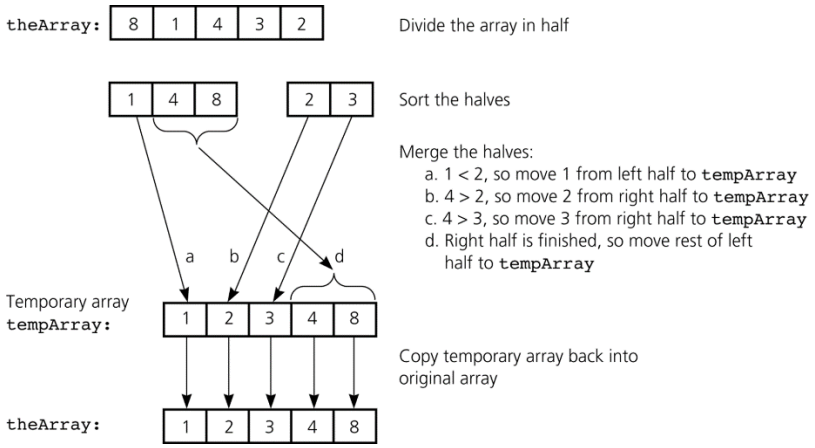
# SYSC 2100 Algorithms and Data Structures
## Summer 2017
## Assignment 4: Sorting and Trees
## Due: August 9, 2017 at 6 pm

1. One of the divide-and-conquer sorting algorithms that we learnt in the class is *Mergesort*, which has an elegent recursive formulation and is highly efficient. *Mergesort* is a recursive sorting algorithm that always provides the same performance, regardless of the initial order. Given a sortable list, *Mergesort* algorithm divide the list into halves, sort each half, and then merge the sorted halfs into one sorted list. An example of *Mergesort* algorithm is as follows:




Implement the *Mergesort* algorithm, which will accept an unsorted array as an argument, and then will sort the array in ascending order. Example specification can be as follows:

```
public static void mergeSort(T[] theArray);
```

Using your implementation of *Mergesort* algorithm, sort the following array of names based on the first character and demonstrate the output.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Lisa | Adam | John | Vicky | George | Beth | Kate | Aaron | Jinny |

**Submission Requirements**: Submit your assignment (the source files and byte code) using ***cuLearn***. Your program should compile and run as is in the default lab environment, and the code should be well documented. Submit all files without using any archive or compression as separate files. The main program should be called **`TestMergeSort.java`**.

2. You are required to implement a way to manage a Table/Dictionary, with operations on that ADT specified in the interface *Dictionary.java*. The Table stores *Strings*, referenced by *SortableStrings* (Strings that implement the *Sortable* interface). Your implementation should internally use a Binary Search Tree (BST). The trees in turn are to be implemented using references.

   The course website in cuLearn provides you a source file *DictionaryTest.java* with the main routine that instantiates a dictionary and invokes the methods defined in the interface *Dictionary* to add and deletes entries, print the tree, search for entries, and to print the depth of the underlying binary tree. The course website also provides the interface definition *Dictionary.java*. You have to provide an implementation of *BSTDictionary*, which implement this interface. To get you started, the course website also provides source files for the *Sortable* interface and the *SortableString* class, as well as the source for a *BSTNode.*

Binary Search Trees are described in the textbook in Section 11.3, starting on page 551 (2nd edition) or page 594 (3rd edition). Note that unlike the discussion in the textbook, the *Dictionary* interface manages a table where the search key of type *K* is a separate data item from the entry of type *E* (with both being stored in a tree node). Also, the key type *K* is not of type *KeyedItem* but has to implement/extend the interface *Sortable*.

**Submission Requirements**: Submit your assignment (the source files) using ***cuLearn***. Your program should compile and run as is in the default lab environment, and the code should be well documented. Submit all files without using any archive or compression as separate files. Submit your solution by providing the source files for *BSTDictionary* and other auxiliary classes. When combined with the source files provided on the website, your submission should compile and run as is, and the code should be well documented.

**Extra Part (no marks allocated)**

Implement Table/Dictionary with AVL trees. AVL trees are discussed in the textbook in Section 13.1, starting at page 689/page 700. We will only discuss them briefly in class. A more in-depth description of AVL trees is available at http://en.wikipedia.org/wiki/AVL_tree. The class *AVLDictionary* should built and maintain a balanced binary search tree. To get you started on this part, the course website provides the source for a class *AVLNode*. To test and compare the two different dictionary implementations, use the *DictionaryAdvancedTest.java* source file. When running the main routine, you should be able to verify that the depth of the AVL tree is significantly less than the depth of the BST tree.

Marks will be based on:

| | 25% | 50% | 75% | 100% |
|---|---|---|---|---|
| Part 1 program (5 marks) | Compiles | Runs, but incomplete. | Recursive, but sorting is partially accurate. | Recursive and sorting is accurate. |
| Part 2 program (10 marks) | Compiles | Runs, but incomplete. | Most of the functionality is available in BST. | **BSTDictionary** dumps correctly. |
| Readability (3 marks) | Hard to decipher. | Dead or useless code or variables, reinventing wheel, methods that do too much | Follows conventions, proper whitespace and indenting, appropriate naming of variables and methods. | Includes logging or testing code (commented out or disabled for submission) |
| Comments (1 marks) | Occasional | Lots, but extraneous | Some | Javadoc comments for all member functions, classes. Minimal and cohesive comments elsewhere. |
| Completeness of your submission (1 marks) | Something in CULearn | Any extra effort to extract code (RAR, JAR, ZIP files, incorrect naming). | Somewhat adheres to the submission requirements. | Strictly adheres to the submission requirements. |

The due date is based on the time of the ***cuLearn*** server and will be strictly enforced. If you are concerned about missing the deadline, here is a tip: multiple submissions are allowed. You can always submit a (partial) solution early, and resubmit an improved solution later. This way, you will reduce the risk of running late, for whatever reason (slow computers/networks, unsynchronized clocks, failure of the Internet connection at home, etc.).

In ***cuLearn***, you can manage the submission until the deadline, taking it back, deleting/adding files, etc, and resubmitting it. The system also provides online feedback whether you submitted something for an assignment. It may take a while to learn the submission process, so I would encourage you to experiment with it early and contact the TA(s) in case you have problems, as only assignments properly and timely submitted using ***cuLearn*** will be marked and will earn you assignment credits.