Implementation of data structures and algorithms
Short Project 3: MergeSort

Version 1.0: Initial description (Thursday, Jan 30).
Due: 11:59 PM, Sun, Feb 9 2020.

Submission procedure: same as usual.

Team task:

1. Implement and compare the running times of the following algorithms on randomly generated arrays: (a) Merge sort (take 2), (b) Merge sort (take 3), (c) Merge sort (take 4), and (d) Merge sort (take 6). Make necessary changes to take 6 version to handle input size that is not power of 2. Do not run more than one algorithm in each trial. In each trial, run only one algorithm, for one value of n, 50 times in a loop, and taking the average time. Try the following values of n: 16M, 32M, 64M, 128M ..., until you get out of memory exception. Submit a report with your observations. Starter code is provided.

Practice task (optional):

2. Write the Merge sort algorithm that works on linked lists. This will be a member function of a linked list class, so that it can work with the internal details of the class. The function should use only O(log n) extra space (mainly for recursion), and not make copies of elements unnecessarily. You can start from the SinglyLinkedList class provided or create your own.

```
static<T extends Comparable<? super T>> void mergeSort(SortableList<T> list)
{ ... }
```

Here is a skeleton of SortableList.java:

```
public class SortableList<T extends Comparable<? super T>> extends
SinglyLinkedList<T> {
    void merge(SortableList<T> otherList) {  // Merge this list with other
list
    }
    void mergeSort() { Sort this list
    }
    public static<T extends Comparable<? super T>> void
mergeSort(SortableList<T> list) {
        list.mergeSort();
    }
}
```