EECS 2011: Assignment 2

Due: as set in *eClass.*

May be done in groups of up to three students

> *TL;DR: Implement two variations of merge sort (recursive and iterative) and investigate their running time for sorting 10, 100, … items. Submit 2 java classes. Cheating not allowed.*

Motivation

The purpose of this assignment is to compare the performance of recursive and non-recursive implementations of a common sorting algorithm, and to compare it to the performance achievable with the sorting implementation already available in Java standard libraries.

Introduction

In this assignment you will first need to implement, or to adapt the existing implementations, of merge sort. Being more than 75 years old, it's among the early sub-quadratic sorting algorithms.

In addition to the lecture notes, you might find the following resources useful:

Sorting chapter of the Algorithms textbook
https://algs4.cs.princeton.edu/20sorting/

Wikipedia article on merge sort

https://en.wikipedia.org/wiki/Merge_sort

Implementation

Here, you have to create two classes. Name them `A2` and `Part2` and put them in the *default* package.

Part 1

In `A2` class, implement the following public methods:

```
1. public static void mergeSortIterative (int[] a)
2. public static void mergeSortRecursive 1(int[] a)
```

The class should not have any other non-private methods, nor any public constructors – that is, it should not be possible to create A2 objects.

Other than the running time, the behaviour of the methods in your implementation should be similar to that of the `sort` method in `java.util.Arrays`; please refer to the class' API online). The time complexity, of course, may vary. You may use (after some adapting) the code from other sources. In particular, you might find the examples posted at https://algs4.cs.princeton.edu/20sorting/ useful.

Part 2

In your `Part2` tester class you will investigate how the running time complexity of the sorting methods you implemented compares with that of quicksort (the default algorithm for sorting arrays of primitive types in Java) as the number of items in the arrays you're trying to sort increases. Try using values from 10 to 1,000,000 (or higher) while measuring the running time. Confirm that the running time follows the expected behaviour. Pick the maximum value of n such that the whole test is no more than 10 seconds on your computer.

Output the time to the console in a format similar to this (the time values are completely arbitrary):

```
                mergeSortIter   mergeSortRecur  sort(int)
n = 10          1 ms            1 ms            1 ms
n = 100         19 ms           21 ms           20 ms
n = 1K          100 ms          111 ms          122 ms
n = 10K         1000 ms         1101 ms         1102 ms
n = 100K        1 ms            11001 ms        11001 ms
n = 1M          1 ms            100011 ms       100011 ms
…
```

Feel free to improve the layout of the table as you see fit (consider using printf() or String.format() methods).

Suggestion

For each input size *n*, create three identical arrays of *n* random `int` values (alternatively, create one array and make two more copies of it using `Arrays.copyOf`). Use the arrays as inputs to the respective methods (2 that you implemented, and the `sort` method from the `java.util.Arrays` class).

Part 3

Nothing needs to be submitted here. Observe the behaviour of your merge sort implementation relative to the other sort methods.

Is recursion appropriate when implementing merge sort?

Why would one use merge sort instead of quicksort?

Is recursion appropriate when implementing algorithms like selection sort? "Put the smallest item in front, then sort recursively the remaining $n - 1$ elements." Explain.

NOTES:

1. Do not use *package*-s in your project (put your classes in the `default` package). 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 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. Although you may use the code from the outside sources for Part 1, you must cite those sources.

Submission

Find all the `java` files in your project and submit them electronically via eClass (do not zip, tar, rar, or 7z them). Only 2 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. The extension may or may not be granted, at instructor's discretion.

Grading

The assignment will be graded using *the Common Grading Scheme for Undergraduate Faculties*[1]. 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
     * @exception IllegalArgumentException if the specified initial capacity
     *            is negative
     * uses code from www.xxx.yyy.edu/123.html for initialization
     *
     */
```

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

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

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