Lab 5: Mix-and-match sort

Professor: Ronaldo Menezes

TA: Ivan Bogun

Department of Computer Science
Florida Institute of Technology

September 22, 2014

1 Problem statement

In this lab the task is to implement a recursive algorithm which is a combination of other sorting algorithms. For this purpose recursive variants of Quicksort, Mergesort and Selections will need to be implemented.

2 Implementation

Use interface *Sort.java* [1] which should be used to implement the classes: *QuickSort.java*, *MergeSort.java*, *SelectionSort.java*.

```
@SuppressWarnings("rawtypes")
public interface Sort {

        // find an index to split the array
        int split(Comparable[] list,int first, int last);

        // sort the array from the index 'from' to the index 'to'
        void sort(Comparable[] list, int from, int to);

        // combine elements after sorting algorithm (merge procedure from the
            merge sort
        // can be used in all algorithms)
        void combine(Comparable[] list, int first, int middle, int last);

        // convenience function which would make the first call to sort the
            array
        void sort(Comparable[] list);
}
```

```
public class MergeSort implements Sort{
// implement MergeSort using method definitions from the Sort interface
}
```

```
public class QuickSort implements Sort{
// implement QuickSort using method definitions from the Sort interface
}
```

```
public class SelectionSort implements Sort{
// implement SelectionSort using method definitions from the Sort interface
}
```

[1] Sort.java

Sorting algorithms will be combined using the following class *MixAndMatchSort.java*[2]. Nothing to implement in this class.

```
public class MixAndMatchSort {

	Sort sort1;
	Sort sort2;
	int basecase;

	public MixAndMatchSort(Sort sort1_,Sort sort2_, int basecase_) {

		this.sort1=sort1_;
		this.sort2=sort2_;
		this.basecase=basecase_;
	}

	@SuppressWarnings("rawtypes")
	public void sort(Comparable[] list){

		this.sort(list, 0, list.length-1);
	}

	@SuppressWarnings("rawtypes")
	public void sort(Comparable[] list,int first, int last) {
		if (first<last) {

			if (last-first>this.basecase) {
				int p=this.sort1.split(list, first, last);

				this.sort(list, first, p);
				this.sort(list, p+1, last);

				this.sort1.combine(list, first, p, last);
			}else{
				int p=this.sort2.split(list, first, last);

				this.sort1.sort(list, first, p);
				this.sort1.sort(list, p+1, last);

				this.sort2.combine(list, first, p, last);
			}
		}
	}
}
```

[2] MixAndMatchSort.java

3 Sample input-output

Create the file *Driver.java* [3] whose modified version will be used for testing.

3.1 Input

```
import java.util.ArrayList;
import java.util.Date;
import java.util.Random;

public class Driver {


	public static class Grade implements Comparable<Grade>{


		ArrayList<Integer> grades;

		public Grade(ArrayList<Integer> grades_) {
			this.grades=grades_;
		}

		@Override
		public int compareTo(Grade other) {

			double currentGPA=0;
			double otherGPA=0;

			for (int i = 0; i < this.grades.size(); i++) {
				currentGPA+=this.grades.get(i);
			}

			currentGPA/=((double)this.grades.size());

			for (int i = 0; i < other.grades.size(); i++) {
				otherGPA+=other.grades.get(i);
			}

			otherGPA/=((double)other.grades.size());


			if (currentGPA>=otherGPA) {
				return 1;
			} else {
				return -1;
```

[3] Driver.java

```
                }

        }


        public String toString(){

                String result="";

                double average=0;
                for (int i = 0; i < this.grades.size(); i++) {
                        result+=this.grades.get(i)+" ";
                        average+=this.grades.get(i);
                }
                result+="    average: "+average/this.grades.size();
                return result;
        }

}

@SuppressWarnings("deprecation")
public static void main(String[] args) {


        MergeSort mergeSort = new MergeSort();
        SelectionSort selectionSort = new SelectionSort();
        QuickSort quickSort = new QuickSort();

        int baseline=5;
        MixAndMatchSort mixAndMatchSort1 = new
            MixAndMatchSort(mergeSort,selectionSort,baseline);
        MixAndMatchSort mixAndMatchSort2 = new
            MixAndMatchSort(mergeSort,quickSort,baseline);
        MixAndMatchSort mixAndMatchSort3 = new
            MixAndMatchSort(selectionSort,quickSort,baseline);


        int n=8;

        // arrays to be sorted
        Integer[] list = new Integer[n];
        Date[] dates = new Date[n];
        Grade[] grades=new Grade[n];

        Random random = new Random(1);


        int numberOfGrades=3;
```

```
            // populates arrays with random numbers, dates, grades
            for (int i = 0; i < n; i++) {

                    list[i]=random.nextInt(100);
                    dates[i]=new Date(random.nextInt(115),
                        random.nextInt(12),
                        random.nextInt(31),random.nextInt(24),random.nextInt(60));

                    ArrayList<Integer> arrayListWithGrade=new
                        ArrayList<Integer>(numberOfGrades);
                    for (int j = 0; j < numberOfGrades; j++) {

                            arrayListWithGrade.add(random.nextInt(100));
                    }
                    Grade grade = new Grade(arrayListWithGrade);
                    grades[i]=grade;

            }

            // sort in place
            mixAndMatchSort1.sort(list);
            mixAndMatchSort2.sort(dates);
            mixAndMatchSort3.sort(grades);

            // print to be sure everything is correct
            Driver.printArray(list,false);
            System.out.println("\n\n");
            Driver.printArray(dates,true);
            System.out.println("\n\n");
            Driver.printArray(grades, true);
    }

    @SuppressWarnings("rawtypes")
    public static void printArray(Comparable[] array,boolean onEveryLine){
            // print array


            for (int i = 0; i < array.length; i++) {
                    if (onEveryLine) {
                            System.out.println(array[i]+" ");
                    }else{
                            System.out.print(array[i]+" ");
                    }

            }System.out.println();
    }
}
```

3.2 Output

```
36 48 53 83 85 89 92 96
```

```
Tue Feb 02 15:42:00 EST 1904
Wed Sep 03 22:59:00 EDT 1919
Mon Apr 26 07:20:00 EDT 1920
Tue Jan 17 10:05:00 EST 1928
Wed Jan 08 05:52:00 EST 1930
Thu Mar 24 08:44:00 EST 1955
Wed Aug 25 14:04:00 EDT 1993
Sat Mar 10 22:00:00 EST 2007
```

```
28 59 16      average: 34.333333333333336
34 6 78      average: 39.333333333333336
6 63 55      average: 41.333333333333336
0 51 86      average: 45.666666666666664
33 76 55      average: 54.666666666666664
34 92 62      average: 62.666666666666664
77 37 77      average: 63.666666666666664
74 59 98      average: 77.0
```

4 Grade breakdown

basis	grade
Implementation	(60)
Each individual sort 5/each (total 3)	20
Each combination 5/each (total 6)	40
Comments	(20)
General	10
Javadocs	10
Overall	(20)
Compiled	5
Style	5
Runtime	10
Total	100