COMS 228: Introduction to Data Structures, Summer 2013
Homework Assignment 2

June 4, 2013

Assignment Overview

Famous inventors and cheese enthusiasts, *Wallace and Gromit*, have created a myriad of gadgets, widgets and trinkets. Their house has become so cluttered with contraptions that they have actually designed a machine to sort through all their inventions. After months of tinkering, Wallace and Gromit have successfully created the Generic-Sorter-3000. The only problem is, Wallace doesn't know Java and Gromit, who graduated from Dogwarts University with a degree in Engineering for Dogs, never took a course in data structures and needs the help of an ace computer programmer to program the logic of the Generic-Sorter-3000. To test their machine, Wallace and Gromit splurged at a Cheese sale that sold packages of Cheese in pairs.

In this assignment, you will practice implementing generic sorting methods while using the Comparable and Comparator interfaces. The methods will sort arrays of type T. The algorithms you will implement are selection sort, insertion sort, and recursive quick sort. You will also implement several methods for choosing a pivot. Finally, using the sorting algorithms, you will perform multiple sorts on an input file containing a list of Cheese names, Cheese weights and Cheese values. You will measure the time cost of each algorithm by counting the number of comparisons that are performed. We highly recommend that you use our textbook's implementation of all three of these sorting algorithms as a model for the methods you will be writing. The sorting methods you write will still be quite different than the book's methods.

Important Note: Each class file you write needs to be included in the package "edu.iastate.cs228.hw2". Your source files should be located in the directory "edu/iastate/cs228/hw2" as specified by your package name. When submitting your homework through Blackboard, please place the "edu" folder in a zip file. Make sure to include your last name in the name of your zip file

Specification for the Cheese Class

Cheese has two properties: weight, represented by a double and typeName, represented by a String. In the Wallace and Gromit universe, there are only three types of Cheese: Cheddar, Swiss and Wensleydale. A constructor for Cheese will take in a double and a String for the name as such

public Cheese(double weight, String typeName)

If the String given to the constructor is not “Cheddar”, “Swiss” or “Wensleydale” (case shouldn’t matter) throw a new *IllegalArgumentException.* Cheese should have five public methods: getters for typeName and weight and a third method called

public double getValue()

This method will compute the value of the Cheese with the given rules

- If the typeName is “Swiss”, getValue will return 2.50*weight
- If the typeName is “Cheddar”, getValue will return 9.50*weight
- If the typeName is “Wensleydale”, getValue will return 13.75*weight

The fourth method should be a toString method that returns the typeName, weight and value all separated by commas.

Lastly, Cheese should be comparable to other Cheeses. That is, Cheese should implement the Comparable interface and needs a compareTo method. compareTo should only compare to other Cheeses and a Cheese with higher value should be ’greater than’ a Cheese will lower value.

Specification for the Pair Class

Pair is a generic class whose two elements can take two different types. To achieve this, we will use two generic type parameters E, T. So the “Pair” class should be in the form of Pair<E, T>. Both E and T must extend the Comparable interface. Also, the Pair class itself must implement the Comparable interface as well. This class must use two private variables, one of type E, the other of type T. Because Pair implements Comparable, you must provide an implementation for the **“compareTo()”** method. The order that is defined by the **compareTo()** method should be the typical ascending order for a collection of pairs. Specifically, this means to order the pairs by their first component. However, if the first component of two pairs are equal, then you order them by the second component. For example, the ascending order for the pairs

(5, 0.1), (4, 4.4), (1, 5.8), (4, 2.6)

is:

(1, 5.8), (4, 2.6), (4, 4.4), (5, 0.1).

The **compareTo()** method for the Pair class will need to make use of the **compareTo()** methods of the type variables E and T. Futhermore, you need to write two getter methods **getFirst()** and **getSecond()** so that others can retrieve the two elements. They are very helpful when you need output the sorted pairs.

Specification for the Sort Class

The sort class contains the following three generic sorting methods:

public static <T> long selectionSort(T[] arr, Comparator<? super T> cp)

public static <T> long insertionSort(T[] arr, Comparator<? super T> cp)

public static <T> long quickSort(T[] arr, Comparator<? super T> cp, String ptype)

Notice that all three methods takes a generic array "**arr**" (the array to sort), and a Comparator object "**cp**" to perform comparisons on the array. The quickSort method also takes a string called "**ptype**" which is used to specify the means of choosing the pivot. Please note that you will have to include several other methods in order to fully implement quick sort.

Each of these sorting methods needs to check if **cp** or **arr** is null, and it must check if the length of arr is 0. If so, then throw a new *IllegalArgumentException.*

Notice that each of the sorting methods returns a **long**. The value to be returned is the number of comparisons performed during the execution of the sorting algorithm. Specifically, you need to return the number of times in which the "**cp.compare()**" method is invoked (do not count the occurrences of **cp.compare()** when trying to calculate the pivot for quick sort). We use **long** here to expect a large number of comparisons needed to sort a very large array.

Implementing the selection sort and insertion sort algorithms should be fairly straightforward. However, implementing quick sort will be more difficult. Remember, quick sort must be implemented in a recursive manner (thus you may have to write another method called "recursiveQuickSort()" which initiates the recursion). You will also have to implement the "partition" method discussed earlier in class.

Recall that there are many ways to choose a pivot. You will write the quick sort algorithm in such a way that the client can choose between three different ways of selecting the pivot. The first way is to simply select the first element in the subarray to be sorted (this is how we did it in class). The second way is to take the first, middle, and last values in the subarray and let the median of these values be the pivot. The third way is to simply choose a random value in the subarray to be the pivot. The **ptype** string variable is used to determine which way of selecting the pivot the client has chosen. The string **ptype** should be either "first", "median", or "random". If **ptype** is any other string, throw an appropriate exception.

Specification for the SortingClient Class

The "SortingClient" class will require a **main()** method, and will use the generic sorting methods found in the **Sort** class to sort a list of Cheese data found within an input file in increasing order.

The main method takes as input a file that contains a Cheese's name and weight on each line. The program should not ask the user to type in the name of the file. Rather, it should collect the file name by specifying a program argument in Eclipse. You do this by clicking the down arrow on the "run" button, clicking

"Run Configurations...", clicking the "Arguments" tab, and then typing the name of the file in the "Program arguments" textbox. When you specify a program argument, the argument gets stored in the String array found in the parameter of the main method (this array is often called "args"). Your program must look for the file you specified relative to the default working directory. This should be the directory that contains the ".settings", "src", and "bin" folders. Your output files should also be created in this directory.

The input file is formatted in such a way that each line consists of two sets of two strings. Each String in the set is separated by a comma and each set is separated by a "|". The first string is the name of the Cheese, the second string is the Cheese's weight, the last String is the Cheese's value. For example, each line will be formatted as such:

Cheddar,2.91|Cheddar,6.49

Thus, each line consisting of a pair of strings represents an instance of *Pair<Cheese, Cheese>*. The program will need to keep track of the number of lines so that you can create an Array of Cheeses. To declare and initialize an array of type *Pair<Cheese, Cheese>* use the following line of code (you may ignore the warning):

Pair<Cheese, Cheese>[] pairs = (Pair<Cheese, Cheese>[]) new Pair<?,?>[size];

You can first use Scanner to parse each line in order to get the six strings (you will really only need the first two in each set). Your program should check the file to make sure it has been formatted correctly. If the file is empty or has less than three or more than six strings on any given line, then your program needs to output a simple error message informing the user of the problem. After the message is output, the program should simply terminate.

Now, you still need a Comparator object in order to make comparisons (you will pass this object to each sorting method). Do this by creating an anonymous class (read on for a description) that implements the Comparator<Pair<Cheese, Cheese>> interface. This class only needs to implement the "compare" method. This method defines a custom order for Pair<Cheese, Cheese> objects. Specifically, this order should be the normal ascending order for pairs of (Cheese, Cheese).

Next write another Comparator as an anonymous class. This Comparator should be very similar to the one you just wrote. But instead of ordering by normal ascending order for pairs of (Cheese, Cheese), sort by the first Cheese's weight.

Please write your Comparators as an "anonymous class" inside the main method.

An anonymous class is a class you can declare inline. That is, you don't need to make a separate file or make an inner class with a name. For example, if we have an interface **MyInterface** and we want to only make one instance of it, it is often more clean to make an anonymous class as such:

```
MyInterface interface=new MyInterface(){
public void myInterfaceMethod(){} //must implement interface's methods here
}
```

Next, you must perform eight sorts (in ascending order) on the array of pairs. Four sorts will be done using the first Comparator, the other four sorts will be done using the second Comparator. These sorts include selection sort, insertion sort, quick sort (using the first element as the pivot), and quick sort (using the median of the first, middle, and last elements of the subarray to pick the pivot). Sort the Arrays using both Comparators you just wrote. Because you are performing eight sorts, you will want to create additional copies of the array of pairs. Each time you perform the sort, output the results into a separate output file that has an appropriate name. For example, if you perform selection sort, output the results into files called "SelectionSortWeight.txt" and "SelectionSortValue". The format of the output file should almost be identical to the input file except that the Cheese value will be printed (a Cheese name, weight and value separated by commas) and the Cheeses will be sorted. You should also include the number of comparisons made during the sort at the top of the output file. By doing this you can compare the time-cost of each sorting method. In total, you should have four output files.

Data Files

We are including three different test files cheesesSmall.txt, cheesesMedium.txt and cheesesLarge.txt which contains 100,1000 and 10000 lines of Cheese names, weights. You can test your code on this small data set.