

APS 105 – Computer Fundamentals

Lab #8: Sorting

Fall 1999

(To be completed *before* your lab period, week of November 1.)

Objective: To experiment with several searching and sorting algorithms and measure their running time on different data set sizes.

1 Sorting Arrays

For this part of the assignment you will be running two different sorting algorithms on the same sets of data and measuring how much work each algorithm does. To measure the amount of work performed, you should “instrument” your code so that a count of the number of comparisons between two array elements is obtained. This count will be the measure of the work performed. (You are not to count all comparisons – only those between members of the array are considered as part of the work.)

To compare the efficiency of the sorts, your program will sort arrays of length 1000, 2000, ..., 10000 (i.e. all multiples of 1000 between 1000 and 10000 inclusive). If you wish, you may try other array sizes. For each array, you are to perform the following sequence of steps:

1. Generate an array of random numbers; double values in the range [0, 1). Use the method `Math.random()` to give you a random number.
2. Make a copy of the array and sort it using the *selection sort* algorithm.
3. Make a copy of the array and sort it using the *quick sort* algorithm.
4. Output the size of the array and the amount of work each algorithm performed.

Plot the results of your experiment. You may plot your results by hand on a piece of paper, or you may use a plotting package such as `xgraph`. To learn how to use `xgraph`, type `man xgraph` in a Unix window.

Note 1: it is *very important* that you make a copy of the array before sorting; you want both sorting methods to sort the same initial data.

Note 2: your program make take a long time to run on `skule.ccf`, so you may get a message about your CPU time limit exceeded. If this happens, change your program to sort only one array size at a time.

2 A Simple Phone Book

For this part of the lab, you are to create a simple telephone book program. The book will be represented by an array of 50 `Entry` objects, with each object having a surname, a first name, and a telephone number. The `Entry` class **must** provide a constructor of the following format:

```
Entry (String surname, String first name, String phonenumber)
```

1

Since you will be sorting them, the class will also need to provide methods to determine the ordering of two `Entry` objects. For simplicity, you may assume that surnames are always unique (that is, no two instances of `Entry` will have the same surname).

To make it easier to mark your lab, and save you time entering data, you will need to copy two files:

```
/share/copy/aps105/lab8.java  
/share/copy/aps105/phonedata
```

Use `Lab8.java` as the starting point for your program, which supplies an `Entry` class and a `PhoneBook` class. In the `PhoneBook` class, you will find the `main()` method as well as the method `getData()`, which returns an array of 50 *fully initialized* `Entry` objects. The provided method works by reading its input from the `Phonedata` file. Do not be concerned if you do not understand the code contained in `getData` since it uses Java methods which will not be covered in the course. You will have to complete the `main` method, and supply new methods in the `PhoneBook` and `Entry` classes. Do not modify these classes other than adding new methods (i.e., do not add or change the visibility of any instance variables, or add any new class variables).

Once you have used `getData` to build an array of `Entry` objects, you will have to sort the array. Modify a copy of the `quicksort` algorithm that you wrote for part 1 to do this. You will need to provide a very simple user interface that lets the user enter a surname and then looks up that surname in the array. Since the array has been sorted, you are to use a binary search. To illustrate the operation of the binary search, it is to print out the surname of each `Entry`, as it searches the array. The following example illustrates the required behaviour when applied to an abbreviated phone book of 50 entries. The special surname `Exit` is used to end the program.

```
*** Phone Number Locator ***  
Surname to Lookup? Dhaliwal  
Checking Lalat  
Checking Goguen  
Checking Davils  
Checking Fisher  
Checking Dhaliwal  
Dhaliwal, Amrit: 876-9965  
  
Surname to Lookup? Lebeau  
Checking Lalat  
Checking Goguen  
Checking Khan  
Checking Longridge  
Checking King  
Checking Le  
Not found.  
  
Surname to Lookup? Exit  
Program terminating.
```

3 Optional Part

You may extend part 1 by implementing additional sorting algorithms. There are many algorithm textbooks available in the library which contain sorting algorithms. Some suggested algorithms you may look up and implement are: Bubble sort, Shaker sort, Merge sort, and Insertion sort. You could also investigate the effectiveness sorting algorithms on already sorted data, almost sorted data, and data sorted in reverse order (some of the results may surprise you).

2