COSC1285/2123: Algorithms & Analysis
Laboratory 6

Topic

Sorting.

Objective

Students who complete this lab should:

- Learn to implement three fundamental sorting algorithms.

Introduction

In this lab exercise you are to implement three sorting algorithms, mergesort, quicksort and cocktail sort.

Provided Code

`SortDemo.java` reads a sequence of white-space separated integers from file and sorts them into ascending numerical order. `SortDemo.java` allows you to sort the input using several different sorting algorithms. Most of the algorithms were covered in the lecture and in the text book.

The program is based on assignment one and is divided into the following modules:

file	description
`SortDemo.java`	Code to read data from disk into the set. Also performs timings. No need to modify this file.
`BubbleSort1.java`	Implementation of standard bubble sort. No need to modify this file.
`MergeSort.java`	Implementation of mergesort. You should complete the implementation of the `sort` and helper methods.
`QuickSort.java`	Implementation of quicksort. You should complete the implementation of the `sort` and helper methods.
`CocktailSort.java`	Implementation of cocktail sort. You should complete the implementation of `sort` method.

Compile the program using the following command:

```
javac *.java
```

Run the command using the following parameters:

```
$ java -Xss5m SortDemo
USAGE: SortDemo [sort method] [input file]
  sort methods [bubble, quick, merge, cocktail]
EXAMPLE: SortDemo quick random.txt
```

© 2019 RMIT University

Note, you should specific *-Xss5m* as a recursive quicksort for reversed input is likely to need a larger call stack size.

The `SortDemo.java` program supports the following algorithms:

algorithm	description
`bubble sort`	Standard bubble sort algorithm.
`cocktail sort`	Cocktail sort is a variation of the standard bubble sort algorithm, such as the one provided in Section 3.1 of the textbook. The standard bubble sort algorithm traverses the array left-to-right, swapping records that are out of place. Cocktail sort modifies the standard bubble sort algorithm so that it instead alternately traverses the array from left-to-right and then from right-to-left.
`quick sort`	Your quick sort implementation.
`merge sort`	Your merge sort implementation.

Task

Your task in this lab exercise is to implement the following sorting algorithms in `libsort.c`:

Merge sort is to be implemented in `sort_merge()`. Mergesort is described in more detail in the lecture 5 slides 7 to 28 and Section 5.1 in the textbook.

Quick sort is to be implemented in `sort_quick()`. Quicksort is described in more detail in the lecture 5 slides 29 to 63 and Section 5.2 in the textbook.

Cocktail sort is to be implemented in `sort_cocktail()`. See http://en.wikipedia.org/wiki/Cocktail_sort for details.

After implementing the three algorithms run each of the algorithms on the following test files and compare the run times:

file	description
`debug.txt`	file with few items for testing purposes.
`random.txt`	file with items in random initial order.
`nearlysorted.txt`	file with items nearly ordered.
`reversed.txt`	file with items in reversed sorted order.
`fewunique.txt`	file with very few unique items

Once you implemented and tested all your algorithms on the given test files try implementing the following optimizations and rerun each algorithm on the test files:

Quick sort pivot element selection: Selecting a good pivot can improve the speed of the algorithm for certain data sets. Use the median-of-three strategy to choose the pivot element (see slide 63 in the lecture notes and page 180 of book).

© 2019 RMIT University

Merge sort skip merge step: Merging two subarrays that do not overlap is trivial, i.e., if the largest element of first subarray (last element in subarray) is less than the smallest element in the second subarray (first element in subarray), then the two subarrays do not overlap. Implement this strategy for nearly ordered lists/arrays.

Cocktail sort skip already sorted elements: The first rightward pass will shift the largest element to its correct place at the end, and the following leftward pass will shift the smallest element to its correct place at the beginning. The second complete pass will shift the second largest and second smallest elements to their correct places, and so on. After i passes, the first i and the last i elements in the list are in their correct positions, and do not need to be checked. By shortening the part of the list that is sorted each time, the number of operations can be halved. [1]

Question: Did any of the algorithms improve significantly? If yes why do you think that is?

Challenges

- How do your experimental results compare to the known theoretical efficiencies of the algorithms?
- Describe the performance of cocktail sort as compared to bubble sort.

[1] from: http://en.wikipedia.org/wiki/Cocktail_sort

© 2019 RMIT University