Assignment 9: Reading and Programming Project due 11/7

Reading Assignment

Read Chapter 11, *Recursion*.

Read the handout on MergeSort distributed in class (note that this handout is not available on-line).

Programming Assignment

In this week’s assignment you will implement a sorting algorithm: BubbleSort or MergeSort.

BubbleSort algorithm is described in the Handout on Sorting. MergeSort is presented in a separate handout (section from a textbook by R. Shackelford distributed in class) and was covered in class.

BubbleSort is a simple algorithm and its implementation is relatively straightforward. However it is does not have the optimal running time for sorting. Its worst case time complexity is $O(n^2)$.

MergeSort is a faster algorithm - it runs in time $O(n \lg(n))$. That running time cannot be beaten - it turns out that if the range of array elements is not known in advance sorting $n$ elements cannot be done faster than in time $O(n \lg(n))$ in the worst case. However MergeSort requires an additional array for merging, and thus is less memory efficient than other sorting algorithms we studied. Implementation of MergeSort involves recursion and is more complex than that of BubbleSort, but if you choose to implement MergeSort - your program would have the optimal worst case performance.

Programming Project

**Sort:** implement BubbleSort or MergeSort due 11:00 p.m. on Thursday, 11/7 worth 10 points

In this assignment you will be sorting an array of numbers entered by the user. After reading how many numbers there are in the list (that could be any number greater than 0) your program must let the user enter exactly that number of integers and then sort them using either BubbleSort or MergeSort and output the numbers in non-descending order.

**On the design of MergeSort.**

As the implementation of MergeSort is somewhat harder than that of BubbleSort I have supplied an unfinished program ([http://cis.bentley.edu/tbabaian/cs230/wks/MergeSortProg.java](http://cis.bentley.edu/tbabaian/cs230/wks/MergeSortProg.java)) that embodies some of the program design features suggested below.

A part of implementation of MergeSort is the procedure that merges two already sorted parts of the original array. We’ll call a part of the original array *subarray*. The beginning and end of each subarray is identified by the start and end indecies. Notice that the two merged subarrays are always consecutive, i.e. the second one starts right after the first subarray ends. Merging of two separately sorted subarrays into one is done
in a separate auxiliary array. Once the merging is complete in the auxiliary array, the elements from the auxiliary array must be copied into the original array.

The design of the MergeSortProg application includes a static method called Merge that performs the procedure above. The strategy used to merge two subarrays is the same as the one used in the implementation of an earlier project Merge (solution available from http://cis.bentley.edu/tbabaian/cs230/hws/mergesol.html). However the details of the implementation differ, since in MergeSort the procedure must merge two subarrays of the same array rather than two different arrays.

To carry out this task, the Merge method should be passed the array itself, as well as the boundaries (starting and ending indices) of the subarrays to be merged. The proposed design includes static method Merge that has the following parameters

1. the original array int[] arr,
2. start that denotes the start index of the first subarray,
3. mid that denotes the end index of the first array. At the same time mid+1 is the start index of the second subarray.
4. end that denotes the end index of the second subarray.

You may define other methods in your implementation of the MergeSort.