LEBANESE AMERICAN UNIVERSITY
School of Arts and Science
Department of Computer Science and Mathematics

CSC 310: Algorithms and Data Structures

Lab III

22. Sep. 2015

This lab deals with sorting algorithms, where you shall be given an array of integers and you have to sort it in increasing order. Each problem requires you to implement a different sorting algorithm. The following sample input and output is common to all problems.

Input

All input is read from a file named "lab3.in".

The first line of input is an integer *T* representing the number of test cases.
Each test case is made up of two lines. The first line has an integer *N* representing the number of elements in the array. The second line contains *N* integers representing the values in the array.

Output

For each test case, output the sorted array.

Sample Input	**Sample Output**
3 7 25 13 10 30 15 27 37 4 6 7 8 9 6 10 7 15 13 4 6	10 13 15 25 27 30 37 6 7 8 9 4 6 7 10 13 15

Problem 1

Given an array of integers, write a program that sorts the array using *Selection Sort*.

The algorithm works by first selecting the minimum number in the array, then swapping it with the beginning index. After the swap is done, increment the beginning index by one, and redo the previous operation until you reach the end of the array.

Problem 2

Given an array of integers, write a program that sorts the array using *Insertion Sort*.

For each integer in the array, as long as it is less than the value to its left, you keep swapping in a left-wise manner. For example, consider the following array.

7 8 6 9

Assume the algorithm is now at position 2, where array[2] = 6. As long as 6 is less than the value to its left, we swap it. Therefore, the following changes will be applied to the array.

7 6 8 9
6 7 8 9

Problem 3

Given an array of integers, write a program that sorts the array using *Bubble Sort*.

The algorithm works by keeping track of swap operations. If you did not swap any numbers during a loop over the array, then the array is sorted and the algorithm is done. The idea is to check each pair of adjacent elements. If one is greater than the other, then you swap their positions. If no swapping took place, you stop.

Problem 4

Given an array of integers, write a program that sorts the array using *Counting Sort*.

This algorithm is faster than previous algorithms, but it requires an extra auxiliary array that acts as a histogram. First of all, you are required to find the maximum value in the input array. Then, create an array whose length is equal to the maximum value. Loop over the input array and increase the count of each integer in the histogram array. Once done, copy the values according to their count back to the input array. For example, assume we have the following array.

7 7 2 4 1 2

The maximum value in that array is 7, so our histogram array will have a length equal to 7 (or 8 if the array is zero-based).

0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0

When looping over the input array, increment the index of its current element by one. The histogram array will become as follows.

0	1	2	3	4	5	6	7

0	1	2	0	1	0	0	2

Now, loop over the histogram array and copy the indices *count* number of time back to the input array. The input array will become sorted.