Due Date: April 26, Thursday, 23:55
(submit online via ODTU-Class)

CENG 301 Spring 2015-2016

Assignment #2 Insertion Sort with Buckets

In this assignment, your goal is to compare two different insertion sort algorithms. The first one is the ordinary insertion sort meaning each element in the unsorted part of the array is inserted into its correct position one by by, in the sorted part of the array by searching its position in the whole sorted part. It is inserted in that position and all the number to the right of that position is shifted one position to the right. A way to improve Insertion Sort is to organize the sorted part in a number of buckets with equal predefined ranges. For example, for the data range [0:1000] with 10 buckets, our first bucket will store sorted numbers in the range [0:100), then the second bucket will store sorted numbers in the range [100:200) and so on. When a new number is processed from the unsorted portion, it is inserted in the specific bucket whose range incudes that new number. The corresponding bucket of a new number P can be found easily with a constant number of operations in O(1). After all the numbers are processed and inserted in their corresponding buckets, the buckets are simply concatenated one after another in O(N) time, where N is the size of the array to be sorted. In this assignment, you will implement a fixed number of 10 buckets.

You are free to implement the bucketed version of insertion sort using either **arrays** or the **linked list** data structure. Note that although the ranges of buckets are fixed, the number of integers to go in a bucket cannot be known in advance, i.e., not all buckets will have exactly N/10 elements. (In the worst case for example where your data is {1,501,502,…599,1000}, most of the data will be in the 6th bucket). Therefore, if you use an array based implementation, the capacity of all of your buckets should be N considering the worst case scenario. You should also maintain the bucket sizes of each buckets during the sorting process to now the end index of each bucket. The minimum and maximum numbers in the integers to be sorted will not be known in advance, so you will have to first find these by a single pass over your integers in O(N) time. If you prefer a linked list based implementation, you do not need to pre-specify the sizes of buckets (and hence save memory space) and you can avoid shifting of the integers during insertion. In addition, if you also maintain pointers to the end of the linked lists representing the buckets, the final concatenation can be done in O(1) time instead of O(N).

A number of test cases with different size are provided at http://www.ceng.metu.edu.tr/~tcan/ceng301_s1516/Schedule/hw2_test.zip . Each test input contains the number of integers to sort, N, in the first line followed by N lines of integers. Implement the original insertion sort algorithm and the bucket version and compare their running time.

Submission:
Submit your source code named <student_id>.c and a short report that shows the comparison of running times via ODTU-Class before the deadline. Late submissions are allowed with 20 pts penalty per day.