**CS311 DSA Project 2**

**Your Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Univ. ID: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Instructions**

* Demo deadline: lab class time, Submission deadline: check **due date** on course webpage
* Remember to comment your code
* Remember to take screenshots of the running results
* Zero mark will be given if your program does not compile, or gets into an infinite loop (does not terminate)
* **Submission guide**
	+ Create a new folder named with CS311\_Proj2\_yourID
	+ Archive your code files and this .doc file into **CS311\_Proj2\_yourID.zip (or .rar)**
	+ Submit **the archived file** to cs\_scu@foxmail.com

**Implementation for Quicksort (See Chapter III-7.5)**

**template <typename E>**

**inline int findpivot(E A[], int i, int j)**

 **{ return (i+j)/2; }**

**template <typename E, typename Comp>**

**inline int partition(E A[], int l, int r, E& pivot) {**

 **do { // Move the bounds inward until they meet**

 **while (Comp::prior(A[++l], pivot)); // Move l right and**

 **while ((l < r) && Comp::prior(pivot, A[--r])); // r left**

 **swap(A, l, r); // Swap out-of-place values**

 **} while (l < r); // Stop when they cross**

 **return l; // Return first position in right partition**

**}**

**template <typename E, typename Comp>**

**void qsort(E A[], int i, int j) { // Quicksort**

 **if (j <= i) return; // Don’t sort 0 or 1 element**

 **int pivotindex = findpivot(A, i, j);**

 **swap(A, pivotindex, j); // Put pivot at end**

 **// k will be the first position in the right subarray**

 **int k = partition<E,Comp>(A, i-1, j, A[j]);**

 **swap(A, k, j); // Put pivot in place**

 **qsort<E,Comp>(A, i, k-1);**

 **qsort<E,Comp>(A, k+1, j);**

}

**Problems**

Starting with the C++ code given above for Quicksort, write a series of Quicksort implementations to test the following optimizations on a wide range of input data sizes. Try these optimizations in various combinations to try and develop the fastest possible Quicksort implementation that you can.

1. Look at more values when selecting a pivot.
2. Do not make a recursive call to **qsort** when the list size falls below a given threshold, and use Insertion Sort to complete the sorting process. Test various values for the threshold size.
3. Eliminate recursion by using a stack and inline functions.

**What to turn in**

1. A cpp file that includes four implementations for the quicksort algorithm: the original algorithm, the optimized sorting algorithm after (a), the sorting algorithm optimized after both (a) and (b), and the sorting algorithm optimized after (a), (b) and (c).
2. This .doc file.

**How to turn in**

See 'Submission guide' above.

**PROJECT REPORT**

**Your Solutions**Show your solutions, e.g., your ideas (no need to show your code here).

**Inputs and Outputs**

1. Your program should sort a dataset of no less than **one thousand** entries. (If less than, your score of this project would be deducted by 30 points)
2. You need to generate a batch of random data, and then use the four quicksort programs you implemented to sort this data.

Command:

Input $n$ to generate a batch of random data with the length $n$. ($n\geq 1000$), then your program should show the runtime of four algorithms in milliseconds.

For example:

Original: 324ms

(a): 300ms

(a) + (b): 290ms

(a) + (b) + (c): 280ms

Some screenshots of the results of your program's execution.