CIS 435 Programming Exercise Module 3 (30points)

1 What to turn in

Follow the guidelines of Handout 10 dated January 23, 2004. Submissions that deviate from these guidelines will be assigned 0 points.

2 What to implement

Implementation is required for the function described in part A.

3 Part A: Implementation of a non-recursive Merge Sort (25 points)

Provide an implementation of a non-recursive merge sort algorithm with the following syntax and behavior. The trick in dealing with a non-recursive (i.e. iterative) merge-sort lies in the splitting of the keys at iteration i of the standard recursive algorithm. Let us assume that n is a power of two. Your implementation should work for any value of n, however. If you understand the description below, then you can easily remove this assumption for n. At iteration 1, i.e. the first recursive call we have $2 = 2^1$ arrays to sort one array starting at location 0 (C++ boundaries) and one array starting at location $n/2$, which is $n/2^1$ positions away. At iteration 2 we have $4 = 2^2$ arrays to sort of size $n/2^2$ each starting at indices 0, $0+n/4 = n/4$, $0+2n/4 = n/2$, and $0+3n/4 = 3n/4$. Use this information to unfold the recursion. Conversely, when we merge, we merge the two arrays starting at positions 0 and $n/4$, each of size $n/4$, into one starting at 0 of size $n/2$, and the two arrays of size $n/4$ in positions $n/2$ and $3n/4$ into one starting at $n/2$ of size $n/2$.

NOTE: If you submitted a non-recursive implementation as Part of HW2, you can resubmit it and gain additional points!.

```
void nrmrg_sort(void *keys, int n, int size, int (*compare) ( ) );
```

`keys` is a pointer to the input array. Each element of the array is a datatype whose length in bytes is `size`. The length of the array (input size) is `n`. `compare` is a pointer to a function that returns an integer. Its two arguments are pointers to `void` as well. Depending on whether the first argument of `compare` is greater, equal, or less than the second, `compare` returns a positive, 0 or negative number. The parameters of `nrmrg_sort` are similar to those of the ANSI C standard library function `qsort` (Review the evaluation quiz as well).

4 Part B: Experimental Results (5 points)

Run your implementations with the testing functions provided in `main()` of `sortg.c` in `testing.tar` on 4 different data-sets and 6 problem sizes.
1. Use the following problem sizes:

1. $n = 8000$.
2. $n = 32000$.
3. $n = 64000$.
4. $n = 128000$.
5. $n = 256000$.

2. The four different data sets consist of the following test instances.

   1. An array of integers where all the element are the same (say n).

   2. A sorted array of integers where the i-th element of the array is i.

   3. A reverse sorted array of integers where the i-th element of the array is $n - i$.

   4. An array whose elements are randomly chosen using function `random` (see `sortg.c` on how to setup such an array).

   Describe in tabular form the running time of your implementation for each input instance. A timing of the execution of any function can be obtained similarly to the one provided in `sortg.c`

   Remarks.

   The table to be reported for part B should be included at the end of the submitted source code file as comment.

   If you think the running time takes more than a reasonable amount of time (say one or two minutes), try to extrapolate the running time for that problem size and input and indicate so in the compiled table.