CIS 435 Programming Exercise Module 4 (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 Heap Sort/ Quick Sort (20 points)

Provide an implementation of the heap sort algorithm of CLRS with the following syntax and behavior.

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

Provide an implementation of the quick sort algorithm (the one of the textbook) with the following syntax and behavior; the quick sort must adhere to the description of the textbook as far as the operation of Partition are concerned.

```
void quick_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 `heap_sort`/ `quick_sort` are similar to those of the ANSI C standard library function `qsort`.

If you also implement a non-recursive heap-sort (i.e. Heapify is iterative) your receive **10 additional bonus points**. You can only receive full bonus points for a correct non-recursive implementation; partial credit of bonus points will not be assigned. The bonus points can increase your intake to 40 points from the programming module giving you up to 60 points for HW4.

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

4 Part B: Experimental Results (10 points)

Run your implementations with the testing functions provided in `main()` of `sortg.c` in `testing.tar` on 4 different data-sets and 5 problem sizes.
1. Use the following problem sizes:

1. $n = 4000$.
2. $n = 16000$.
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 for large n, execution time seems to exceed 5 minutes on your machine, abort execution and estimate approximate running time and include this time in the corresponding table. Indicate, however, that your entry is estimated and not actual running time.