EC 504 – Fall 2020 – Homework 2

Due Thursday, Oct. 1, 2020 in the beginning of class. Written and Coding problems submitted in the directory `/projectnb/alg504/yourname/HW2` on your SCC account by Thursday Oct 1, 11:59PM.

GOAL: This a short exercise to learn how to measure and fit performance to curves. Also to present these result in graphical form. The SCC has lots of tools, which you can brose at this link, https://www.bu.edu/tech/support/research/computing-resources/scc/, which you can explore and share tips with us all on Slack A few documents will be put on GItHub under directory `ComputingDocs` as needed. For now

1. Install the very useful (new) interface to CCS is the **VNC Viewer**. Follow the instructions in GitHub in directory `ComputingDocs`.
2. Learn to use the UNIX `gnuplot` tool for graphical and fitting with a few instructions also on GitHub in directory `ComputingDocs` Using VNC this graphics is nicely displayed locally on your laptop.
3. Run the sorting routines to measure performance, graph it as function of the size of list **N**. For **extra credit** estimate errors.

1. The main file on GitHub has a range of sorting algorithms and the ability to construct random lists of varying sizes N. The exercise is to run them for varying the sizes $N = 16, 32, 64, 128, \cdots$, average over at least 100 to get good statistics and verify their scaling empirically counting the swap operations.

   The main code `sorScaling.cpp` runs the example:

   ```
   insertionSort   mergeSort  quickSort shellSort
   ```

   The exercise it to make a table of average performance for all 4 algorithm and plot them to see how the scale with N.

   For the standard $O(N^2)$ search algorithms involve local (nearest neighbor) exchanges of element of the given list

   $$A_{list} = a[0], a[1], a[2], a[3], \cdots, a[N-1] \tag{1}$$

   You should find for `insertonSort` you should verify empirically average the algorithm would have

   $$\text{Number of Exchanges} \quad = \quad \frac{N(N-1)}{4} \tag{2}$$

   For mergeSort it should be exactly $\Theta(NlogN)$ and for quickSort on average $O(NlogN)$. Finally see if you can find the value of the γ for shel sort $O(N^\gamma)$. The data file to feed into gnuplot should be in columns:

```
# N  insertionSort   mergeSort  quickSort shellSort
16   xxxx            xxxx       xxxx      xxxxx
32    xxxx           xxxx       xxxx      xxxxx
64    xxxx          xxxx       xxxx      xxxxx
128   xxxx          xxxx       xxxx      xxxxx
....
```

where `xxxx` are the average values. This is convient for using gnuplot to plot and fit the curves.

Place your final source code, outfile and figures with fits in directory`HW2`. Include the makefile so we can compile and test it.

Extra Credit: If you have time you could add error bars to the average (called σ) defined as mean square deviation a second column next to the tabulate averages `xxxx`. These are define for each algorithm and size N by

$$\sigma^2 = \frac{1}{N_{trials} - 1} \sum_{i=1}^{N_{trials}} (Swaps[i] - AvergeSwap)^2$$

where above we suggested fixing $N_{trials} = 100$. The average numbers of swaps in the 100 trials for each algorithm and size N in the table are:

$$AvergeSwap = \frac{1}{N_{trials}} \sum_{i=1}^{N_{trials}} Swaps[i]$$

For general background information the `sorting.h` files has even more sorting algorithms to play with. We could add others like bucket and improve pivots for quicksort etc.

*Note the usual command `make -k` will run the default `Makefile`. However in these code exercises I have two separate **named makefiles** that need to be run using $-f$ flags:

```
make -f makeFind
and
make -f makeRanSort
```

respectively.