EC 504 – Spring 2023 – Homework 2

Due Wednesday, Feb 22, 11:59PM, 2023. Submit the coding and graphic in the directory `/projectnb/ec504rb/students/yourname/HW2` on your SCC account. The figures will be returned to gradescope. More information in class next week on what to return!

GOAL: This a short exercise to learn how to measure and fit performance to curves. Also to present The result will be presented in graphical form. On the EC504 GitHUB at Plotting_and_Fitting there is information on how to use gnuplot and even in **LittleGnuplot.pdf** instructions on how to install gnuplot in your laptop. This is convenient but all can be done easily using gnuplot on the SCC login in with `ssh -Y username@scc1.bu.edu`. For convenience you may also install gnuplot on your laptop. Use Slack to exchange helpful hints on graphing!

1 Sorting Problem

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 a range of the sizes $N = 16, 32, 64, 128, \cdots$, average over at an ensemble of cases (as much as 100). to get good statistics. Report the average behavior a function of N for each sorting method as to determine the scaling empirically as function of the number the mean number of swap operations vs N.

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

```
insertionSort   mergeSort  quickSort shellSort
```

There are Two part. They differ only in changing the number of **Ntrials** . In **Part 1** setting **Ntrials= 1** you benchmark sorting algorithms for a given random sequence as integers as function of it size **N**. In **Part 2** you do this for **Ntrials = 10 to 100** randomly permuting each sequence and averaging over these trials to find their mean value and Root Mean Square Deviation (rms or error).

The second find use this error to for the fitting the curves.

1.1 Part 1: Sorting Plotting Exercise

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 exercise it to modify the main file to build an output data file to plot. The code will be tested to see that it does give output but the grade is mostly on the figure and explanation put in gradescope.

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

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

This output file can be make by a `hack` by printing to the standard output. Just run the code in a terminal (aka shell) with `$./sort > datafile.txt` Then you take what you need using an editor. This is useful quick trick, however you should really set up a separate **output** file. This is necessary if you want submit you code in queue. To set up a output file see the example to do this on GitHub at `HW1_codes/makeSortedList.cpp` (Hey basic software technique. Steal method from other codes!)

You should turn in on gradescope figures and fits for all 4 sorting as function of size N. Can comment on how well the theoretical scaling is verified numerically. It is up to you to do this as you wish. You may choose to combine different algorityms on the same plot.

1.2 Part 2: Get better Statistics and Fit with Error bars

Next you should also run statistics to estimate error bars and to a least square fit to estimate the goodness of fit. This requires adding error bars to the figure and fit. The mean value is the average of **Ntrials** for different permutations of each sequence of size **N** and finding the standard deviation (or error) as the mean square deviation (called error or σ). Now you must plot and fit using these error bars. These for each algorithm and size N by

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

where above we suggested fixing $N_{trials} = 10 - 100$ range. The average numbers of swaps in the 100 trials for each algorithm and size N in the table are:

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

You will want to have your code compute the standard error and put into another column in your output file. By the way all these analysis skill will likely come in handy for the team project.

Not you want a enlarge output table something like. (In fact you can use this for both parts just setting **Ntrial = 1** for Part 1 above.

```
# N  insertMean insertErr | mergeMean  mergeErr | ...
16   xxxx       xxxx         xxxx        xxxx   ....
32   xxxx       xxxx         xxxx        xxxx   ....
64   xxxx       xxxx         xxxx        xxxx   ....
128  xxxx       xxxx         xxxx        xxxx   ....
....
```

I used a print statements C style `fprintf` I O which is easier to format. The syntax

```
 fprintf(cFile,"#                N                |
                      MeanInsert       rmsInsert  |
                      MeanMerge        rmsMerge   |
                      MeanQuck         rmsQuick   |
                      MeanShell        rmShell    |\n ");

// Compute loops and averaging.

fprintf(cFile,"%20.15d  %15.5e  %15.5e  %15.5e  %15.5e
                        %15.5e  %15.5e  %15.5e  %15.5e \n",
                    N, MeanInsert,  rmsInsert, MeanMerge,  rmsMerge,
                       MeanQuick,  rmsQuick,  MeanShell,  rmsShell );
```

The syntax for opening and closing a file are:

```
FILE* cFile;
cFile = fopen ("Plotfile.txt","w+");

// accumulated data from sorting algorithm
// and print to file.

fclose(cFile);
```

You should turn in on gradescope figures and fits for all 4 sorting including errors to give a chisqr of fit. Can comment on how well the theoretical scaling is verified numerically. It is up to you to do this as you wish. You may choose to combine different algorithms on the same plot.

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

COMMENT: In real time in class there will be help next week on all that is need to do these plots and fits.