Assignment 4: Which is the best sort of them all?

Out: 5th March 2014 **Due: 21st March 2014 at 11:55pm**

In this assignment you will implement some sorting algorithms and compare them in order to understand them and determine which sort implementation is the best.

A Sorts class has been provided to you that currently implements the insertion sort algorithm. A main function tests this algorithm using an input file. The main function reads an array from a file and sorts it using insertion sort. To run the program after compiling it, type "./program-name –i file-name". Several input files have been provided to you.

What to do

1. Implement the quicksort algorithm in the Sorts class that takes only a double array and its size as its parameters.
    a. Write a pivot choosing function median3(array,left,right) that chooses the median of the first, middle and the last element of a sub-array as its pivot.
2. Implement the heapsort algorithm in the Sorts class that takes only a double array and its size as its parameters.
3. Implement the bottom-up non-recursive merge-sort algorithm in the Sorts class that takes only a double array and its size as its parameters. To summarize, here is how the bottom-up mergesort works:
    a. Start with step size of 1.
    b. Go through the array from left to right, identifying and merging adjacent sub-arrays of the given step size two at a time.
    c. Double the step size. If the step size is less than n, continue with step b, otherwise quit.

    For example, if you have an array of size 9, then the sub-arrays you will merge will be in order {step=1, (0-0,1-1), (2-2,3-3), (4-4,5-5), (6-6,7-7)}, {step=2, (0-1,2-3), (4-5,6-7)}, {step=4, (0-3,4-7)}, {step=8, (0-7,8-8)}.

    Keep in mind that all three implementations should work for large arrays (i.e. about 100000 long). Your implementation should work in all cases (small/big arrays, duplicate numbers, etc.)

    You are free to add any helper functions that you see fit, but make sure your class has the above three functions exactly as specified.

Paper

You will compile a 5-page report summarizing the performance of each of the above sorting algorithms. This report will report your findings about how many comparisons and assignments each of the above sorting variants perform. The actual time taken by these algorithms is usually influenced the most by these two factors.

In order to measure them, modify your program as follows:

1. Write a class that stores and maintains the number of comparisons and number of assignments (i.e. the number of times something in the array is written to that or any other location).
    a. A comparison is any comparison between two array elements. This DOES NOT include other comparisons (e.g. i<n).
    b. An assignment is an assignment TO or FROM an array element. This DOES NOT include other assignments (e.g. i=0).
2. Create a global object of this class outside your main function.

3. In the Sorts.cpp file, declare this global object as “extern ClassName objName” keeping the object name the same as you defined in step 2. This will make the global object accessible in the Sorts.cpp file.
4. Modify all your implementations above (including the insertion sort implementation) so that it records every relevant comparison and assignment done by the algorithm.

Using your new program, collect data for the report as follows.

The report should summarize the following results:

1. For each of the insertion sort, quicksort, mergesort and heapsort algorithms, find the correlation between number of comparisons and the size of the array, and number of assignments and the size of the array. Proceed as follows.
    a. Read in one of the two random data files.
    b. Choose subarrays of varying sizes 10, 100, 200, ….from the above data and use insertion sort, quicksort, heapsort and mergesort to sort them, recording the number of assignments and comparisons for each run. <u>Make sure you take enough samples to plot your measurements</u>. Make sure that you give the same input arrays to each of the four sorts, so that the results are comparable to each other.
    c. Report all the above numbers in a table in your report.
    d. Plot a graph with ‘n’ on its X-axis and the number of comparisons on the Y-axis, with blue points for quicksort, red points for mergesort, green points for heapsort and black points for insertion sort.
    e. Plot a graph with ‘n’ on its X-axis and the number of assignments on the Y-axis, with blue points for quicksort, red points for mergesort, green points for heapsort and black points for insertion sort.
    f. Comment on your observations, arguing what they say about each sorting strategy.

Make sure your graph shows the entire range of array sizes. If curves are overlapping over a region you wish to comment on, add another graph that zooms into that range only.

2.
    a. Assign a cost of “1” per comparison and “2” per assignment to compute the final cost for each of the 3 sorting strategies. Plot a graph with ‘n’ on its X-axis and cost on the Y-axis, with blue points for quicksort, red points for mergesort and black points for insertion sort.
    b. Assign a cost of “2” per comparison and “1” per assignment to compute the final cost for each of the 4 sorting strategies. Plot a graph with ‘n’ on its X-axis and cost on the Y-axis, with blue points for quicksort, red points for mergesort and black points for insertion sort.
    c. Step ‘g’ simulates how C++ works (comparisons are cheaper than object assignments) and step ‘h’ simulates how Java works (comparisons are more expensive than assignments). Suggest which strategy will be most useful.

Hint: Data collection will go <u>much</u> faster if you modify your main function so that it runs many of these experiments with a single run of your program (manually changing size of the array or sorting strategy will take a very long time to collect data!)

Note: Your analysis above will yield the best results if you pay attention to implementing these algorithms efficiently. Keep this in mind when you implement them.

Here is how you can draw the above plots using Excel (draw scatter plots): http://office.microsoft.com/en-us/excel-help/creating-xy-scatter-and-line-charts-HA001054840.aspx

Formatting of the report

The report should be neatly divided into sections. It should start with a short description of this experiment and what the report contains. Each conclusion should be mentioned clearly. **<u>The actual tables of observation should be at the end as an appendix section.</u>**

<u>How to lose points on this report:</u>

1. The report has no title and author.
2. Write the entire report in 1-2 gigantic paragraphs with tables and figures in between.
3. Writing your conclusions in paragraph form instead of point/bullet form.
4. Pad the report with irrelevant details, like explaining how sorting works or documentation of your code.
5. Make the report longer than 5 pages (excluding the appendix).
6. Use colloquial language or superlatives (e.g. “this algorithm is superb”, “this method sucks”).
7. Conclusions that contradict your data (if your data contradicts expected conclusions say so and then reason why you think that happened).
8. Make spelling or grammar errors.
9. Plots are hand-drawn that are scanned and pasted in the document.

What to submit

Please submit all .cpp and .h files, the Makefile, the data files and the report (in .doc/.docx/.pdf) summarizing your findings in a single zipped file using Reggienet.