Assignment 3 and Paper 1: Which is the best sort of them all?

Out: 25th February 2015 **Due: 17th March 2015 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. Write another version of insertion_sort(array,left,right) that uses insertion sort to sort a subarray arr[l..r].
2. Implement the heapsort algorithm using a binary max-heap in the Sorts class as heapsort(arr,length). You should implement the heap in the array "in-place", and not implement the heap separately and use it as an object to implement heapsort. This will ensure that the heapsort algorithm takes O(1) auxiliary space. You can implement the various operations as private functions in the Sorts class.
3. Implement the recursive merge-sort algorithm in the Sorts class as mergesort(arr,threshold,length). This function uses the mergesort algorithm from class. However if the length of the sub-array is less than or equal to "threshold", it uses the insertion sort algorithm in step 1 to sort that sub-array. Passing "0" for "threshold" means it will always use mergesort, never insertion sort.
4. Implement the quicksort algorithm in the Sorts class as quicksort(arr,threshold,length). This function uses the quicksort algorithm we discussed in class. However, if the length of the sub-array is less than or equal to "threshold", it uses the insertion sort algorithm in step 1 to sort that sub-array. Passing "0" for "threshold" means it will always use quicksort, never insertion sort.
    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.

Keep in mind that all three implementations should work for large arrays (i.e. about 1000000 long). Your implementation should work in all cases (small/big arrays, duplicate numbers, etc.)

You are free to add any private helper functions that you see fit, but make sure your class has the two insertion sorts, the heapsort, the quicksort and the mergesort as the only public functions.

Paper

You will compile a 3-to-5-page report summarizing the performance of each of the above sorting algorithms. This report will report your findings about how much time each of your implementations take with large arrays. The provided file SortsTest.cpp already has code that measures the time taken by a sorting algorithm. Change the call to the sorting function you are timing.

Keep in mind that this is direct time measurement, and so is both unreliable and perhaps unrepeatable. In other words, the same program using the same array will report different times each time you run it. This is NORMAL. However, this is the "gold standard" of determining which sort works best for your environment and data.

The report should summarize the following results:

Section 1: Timings

Draw a graph for each of the following cases, with "size of array" along X axis and "time taken in ms" along Y axis.

1. For arrays of size 100000 to 1000000 in steps of 100000, time taken for insertion sort to sort the array. The curve should be black.
2. For arrays of size 100000 to 1000000 in steps of 100000, time taken for quicksort with median3 partitioning (threshold=0) to sort the array. The curve should be blue.
3. For arrays of size 100000 to 1000000 in steps of 100000, time taken for heapsort to sort the array. The curve should be red.
4. For arrays of size 100000 to 1000000 in steps of 100000, time taken for mergesort (threshold=0) to sort the array. Your curve should be green.
5. Report all the above numbers in a table in your report.

Note: For each of the above trials, run the same algorithm on the same array three times and report the average time.

Conclusion: In a single paragraph, conclude what you learn about these algorithms based on your graph. You should also state the possible reasons for these conclusions. For example, if you find that quicksort is faster than mergesort, comment on why you think this is.

Section 2: Finding the "threshold"

Use an array of size 500,000 for all the timings for this section.

In this section you must attempt to find the maximum value of "threshold" above which using insertion sort for smaller arrays proves to be more expensive than the original sorting algorithm, for quicksort and mergesort.

6. Starting from a threshold of 0 to 200 in steps of 20, graph the times taken by quicksort algorithm with that threshold for the same array of size 500,000. Draw this curve in red, and draw a horizontal black line for the time taken by median3 quicksort algorithm for the same array, but with threshold 0.
7. Starting from a threshold of 0 to 200 in steps of 20, graph the times taken by mergesort algorithm with that threshold for the same array of size 500,000. Draw this curve in blue, and draw a horizontal black line for the time taken by median3 quicksort algorithm for the same array, but with threshold 0.
8. Report all of the above numbers in a table in your report.

Conclusion: In a single paragraph, state the maximum value of "threshold" at which the colored lines cross over the black horizontal lines for each of the sorting algorithms. State how this is practically useful.

Tip: Data collection will go much faster if you modify the main function of SortsTest.cpp 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!). If you modify the program further to write the data out to a .csv file, then importing the data in Excel and drawing graphs will be even easier!

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. **The actual tables of observation should be at the end as an appendix section.**

How to lose points on this report:

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. Pad the report with irrelevant details, like explaining how sorting works or documentation of your code.
4. Make the report longer than 5 pages (excluding the appendix).
5. Use colloquial language or superlatives (e.g. “this algorithm is superb”, “this method bombs”).
6. Conclusions that contradict your data (if your data contradicts expected conclusions say so and then reason why you think that happened).
7. Make spelling or grammar errors.
8. 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 .pdf) summarizing your findings in a single zipped file using Reggienet.