Assignment #3

Sorting

PROG 2400: Data Structures

Due by Date As Specified for your section in D2L

Task:

This assignment has two parts. The first is a comparison of six sorting and searching algorithms on contiguous arrays. The second is the construction of an external mergesort.

Part 1: Bubble, Selection, Insertion, Shell, Merge and Quicksorts

Create a console application to compare the time required to conduct sorting activities on a simple array.

Requirements:

- You may **not** use the classes provided in the STL for sorting, but may use them in a supporting role (timers, random numbers and the like.)
- The program must accept an array size from a prompt and create a dynamic unsorted array of that size. Then it must populate this array with a set of numbers between 0 and 32,767, duplicates are permitted. If you use the standard rand() function the range will appropriate but you will have duplicate values.
- Develop each of the six sorting algorithms and apply to a copy of the same, original, unsorted list to create a sorted one. Collect your timings. Run and collect timings as indicated for each sort for each of these array sizes:
  - 1,000 – Do not time this one, use it to create the output file for sort verification. Use some kind of command flag to avoid writing the larger files. (i.e., input 1,000 followed by a W for write, or something similar)
  - 25,000
  - 50,000
  - 75,000
  - 100,000
- Write the sorted list for size 1,000 for each algorithm to its own output file. This will allow verification that each of the sort routines work.
- There are no restrictions on using recursion.
- Capture and display the start and end times for the sorting of each array. Be sure to **exclude** the time for file I/O.
- Graph the times for each sort in Excel on a single graph to compare the times.
- Graph as well the lines for N*N, N*logN and N for comparison with the sorts.

Part 2:
Option 1. External Mergesort

For this subtask, you must build a console-based application or extend the application built in Part 1, to perform an external multi-key mergesort on a file of data representing a phone directory. The provided file is called phonelist.txt, and the columns (fields) are separated by whitespace.

- The columns represent City, LastName, FirstName and PhoneNumber respectively.

Requirements:

- Allow the user to specify 0, 1 or 2 columns (by position number) to act as the primary and secondary sort keys of the final list.
  - A value of 0 uses the default of the first column (city).
  - If one value (1-N) is used, use that column in the file as the primary sort key, reordering he records based on that column' sorted order.
  - If 2 values are used, sort on the first as the primary, then on the second as the secondary key.
- Implement an external mergesort that recursively splits the file in to two intermediate files and then merges all the intermediate files produced (two at a time) to create a new sorted file based on the key(s) provided. The maximum number of records to be in memory at any time is 2, one from each set of files in that phase of the mergesort.
- Write out the merged file. Do not alter the original file.
- Test using all possible column combinations.

Option 2. Radix Sort

For this subtask, you must conduct personal research and implement a radix sort variant which extends the application built in Part 1. Performa all the same timings and capture the information on the common graph(s).

Evaluation:

This assignment is worth 44 marks. Please see the marking rubric below.

Assignment Notes:

The assignment must be demonstrated to the instructor on or before the due date during class.

If your assignment is late please send an e-mail to the instructor, hal.o'connell@nscc.ca, to confirm submission. This e-mail will constitute the timestamp for evaluating any late penalty the assignment may incur.

See the **Marking Rubric** below.

Criteria	Marginal	Developing	Good	Exceptional	Marks
	0	1	2	3	
Bubble Sort	• No bubble sort attempted	• Sort does not create a sorted list due to significant errors in the algorithm	• Sort algorithm has minor errors • Learner cannot fully explain algorithm	• Sort Algorithm complete and correct • Timings captured and displayed • No errors	_____
Selection Sort	• No selection sort attempted	• Sort does not create a sorted list due to significant errors in the algorithm	• Sort algorithm has minor errors • Learner cannot fully explain algorithm	• Sort Algorithm complete and correct • Timings captured and displayed • No errors	_____
Insertion Sort	• No insertion sort attempted	• Sort does not create a sorted list due to significant errors in the algorithm	• Sort algorithm has minor errors • Learner cannot fully explain algorithm	• Sort Algorithm complete and correct • Timings captured and displayed • No errors	_____
Shellsort	• No Shellsort attempted	• Sort does not create a sorted list due to significant errors in the algorithm	• Sort algorithm has minor errors • Learner cannot fully explain algorithm	• Sort Algorithm complete and correct • Timings captured and displayed • No errors	_____
Mergesort	• No merge sort attempted	• Sort does not create a sorted list due to significant errors in the algorithm	• Sort algorithm has minor errors • Learner cannot fully explain algorithm	• Sort Algorithm complete and correct • Timings captured and displayed • No errors	_____
Quicksort	• No quick sort attempted	• Sort does not create a sorted list due to significant errors in the algorithm	• Sort algorithm has minor errors • Learner cannot fully explain algorithm	• Sort Algorithm complete and correct • Timings captured and displayed • No errors	_____
Excel Timing Graph	• No graph submitted	• Graph missing data	• All data present but is done using multiple graphs	• Data presented in a single graph to allow easy comparison of all data	_____
External Mergesort or Radix Sort (Note multiplier)	• No external merge sort attempted • No radix sort attempted	• Sort does not create a sorted list due to significant errors in the algorithm • External MergeSort does not use external files	• Sort algorithm has minor errors or cannot sort on multiple keys • Learner cannot fully explain algorithm	• Sort Algorithm complete and correct • No errors	_____x2
				Sub Total	_____ 27

Criteria	Marginal	Developing	Good	Exceptional	Marks
	0	1	2	3	
Aesthetics	• incorrect or non-existent use of whitespace in output • output is confusing and hard to follow	• fair use of whitespace • most output is clear, but poorly presented	• excellent use of whitespace • output is clear and attractively presented		____
Readability	• source code is poorly organized and very difficult to read	• source code is fairly easy to read, but is hard to follow in some areas	• source code is exceptionally well organized and easy to follow		____
Reusability	• source code cannot be reused • no functions or classes used	• portions of code could be reused with modifications	• source code could be easily reused with few modifications		____
Efficiency	• contains large portions that could have been easily reduced using a different method • too much code is replicated, copy /pasted	• tried some methods to improve efficiency • can explain what they attempted	• very clean and efficient code • can propose new ideas for improvement		____
Comments	- little to no comments used	• not over/under commented • comments are meaningful and easily understood • files and functions have headers • Code is self-documenting	•		____
Naming Convention	• no standard naming convention followed	• industry standard naming convention used throughout the program	•		____
Consistency	• no consistency in formatting or layout of source code	• source code formatting never deviated from the programmer's layout	•		____

SubTotal	11
Assignment Total	38

0 - Assignment not submitted or work not original.