ECE 36800 – Data Structures
Programming Assignment 1 – Part 2

Purpose of Assignment 1:
Implement and compare the running time of different sorting algorithms.

Goal of Part 2:
Add the implementation of the following sorting functions: selectionsort, insertionsort, shellsort, quicksort, and mergesort.

What to submit:

1. `sorttest2.cpp`: The test sorting program. Include implementations from your `sorttest1.cpp` from Part 1 and add new contents to handle the sorting. If your `sorttest1.cpp` has errors, please correct those errors first.
2. `sort.cpp`: This file should contain the implementations of the specified sorting functions. Start with the version provided.
3. A word document `proj1p2.docx`: This file should have the following parts:
   - Test result: Test each of the sorting function with your simple text file and record the result. You do not need to show the content of the output text file.
   - Running result: After the above step, test each of the sorting function with the long input file `randomwordlist.txt` and record the result. You do not need to show the content of the output text file.
4. Push all your files under the "proj1/part2" directory before the deadline.

Other files available (please do not modify):

1. `sort.h`: Header file for the sorting functions. NOTE: This header file contains only the prototypes. It does not have any class declarations.
2. `randomwordlist.txt`: a file with a long list of strings to be sorted. The first line of the file contains the number of strings contained in the file, and each of the remaining lines contains a single string.
3. `Makefile:` rules to compile the source files for your convenience.

Guideline:

- As continued from Part 1, after reading in the words from the input file and putting them in a string array, you are going to sort the string array, allowing users to choose from the following sorting functions: bubblesort, selectionsort, insertionsort, shellsort, quicksort, and mergesort. When the program finishes, a text file (e.g, `sortedwordlist.txt`) will be generated, including all the words in alphabetic order, with one word in each line.
- In order to understand how fast your algorithm runs, you will need to print out the running time (in milliseconds) for each sorting algorithm.

Explanations of Implementation:

- `sort.h` must only have one prototype for each sort type. Any other functions (the `swap` function, the `partition` function for quicksort and the `merge` function for mergesort) that are in `sort.cpp` should be declared with "*internal linking*". This hides the function name from the linker, and is accomplished by placing the word `static` before the function prototype and before the function definition.
- `partition` function returns the target location of the pivot. Please make sure to implement the **Hoare** partition scheme.
- The system function `clock()` is used to return an approximation of processor time. This function and related variables are included in `<ctime>` header file. The `clock` function is called twice: once called immediately before the sorting and the other one called immediately after the sorting is done. By subtracting these two clock values, we can get an estimate of the running time for the selected sorting algorithm.

Grading Policy:

Part 2 counts for 60% of the overall points in Programming Assignment 1.

Please make sure your program compiles successfully. If not, 30 points will be deducted.

1. **Executability** (5%):
   - Runtime errors: You program must not have runtime errors (e.g., code crash, infinite loop, reading uninitialized memory, accessing the content of a NULL pointer, etc.).
2. **Programming style** (5%):
   - Code efficiency: Code should use the best approach in every case.
   - Readability: Code should be clean, understandable, and well-organized. Please pay special attention to indentation, use of whitespace, variable naming, and organization.
   - Documentation: Code should be well-commented with file header and comments.
3. **Program Specifications/Correctness** (50%):

   Please refer to the Grading Criteria table for details. Specifically, your program should behave correctly, adhere to the instructions, and pass the test program.

file	**item**	**weight (%)**
sort.cpp	selectionsort function	7
	insertionsort function	7
	shellsort function	7
	partition function	7
	merge function	7
sorttest2.cpp	Correct implementations from part 1	5
proj1part2.docx	test result	5
	running result	5
	total	**50**