Algorithms Mini Project Design Document

18MCME02, Amruta Jandhyala, IMTech 5th semester

For a description of all functions used, read myheader.h. Also, main is the executable file name.

Implementation 1:

There are a maximum of two command-line arguments: 1. filename 2. fileinputtype
Filename will be passed as command-line argument. Only checks if filename ends with ".txt".
Specify random, asc, or desc in the arguments to ensure fileinputtype is printed correctly.
The size of the file, which is to say the total number of numbers in the input file, must be given as the first line in the input file. Please don't give filenames as "ascorder.txt", "descorder.txt", or "randomlist.txt". This might cause a bug where either of these files will end up blank later.
A menu will be displayed where you can select the sorting algorithm to be implemented.
"output.txt" file will contain the sorted array output. It is compared with ascorder.txt, to make sure it is sorted. A new line will be appended to "outputtable.txt" so that you can see the time taken and other details.
The above method is how I implemented during testing.

Implementation 2:

1. Create a randomlist.txt input file with 10000 random numbers between 0 and 1000.
2. Use shell command "sort" to create an ascending order input file "ascorder.txt" and descending order input file "descorder.txt" out of "randomlist.txt".
3. Do bubble sort on randomlist.txt and write output to a file.
4. Compare this output with ascorder.txt. If same, then store the values into a struct:

```
struct output{
        long int size;
        int array[100000];
        char sortalgo[20]; // bubble sort, merge sort, etc
        char filename[20];
        double timetaken;
        char fileinputtype[30]; // Random list or ascending order or descending order
}o;
```

Display the values in table format in outputtable.txt file.
5. Similarly, do bubble sort on ascorder.txt and descorder.txt and repeat step 3 and 4.
6. Similarly, do selection sort, insertion sort, merge sort, quick sort, and heap sort on randomlist.txt and repeat step 3, 4, and 5.
7. Then add another 500 numbers in the range 0 and 1000 to randomlist.txt. Repeat step 2, 3, 4, 5, 6.
8. Keep repeating step 7 until size of randomlist.txt is 100000.

The algorithms will be written in separate files and functions. The sorting algorithms will take array and size of array as input return modified array as output.

No command-line arguments were given as input for this implementation.
One function file each for insertion sort, bubble sort, quick sort, merge sort, heap sort, and selection sort. One function file for taking file input, storing into array and calling all the sorting algorithms successively. One function to write the sorting algorithm output to a file, checking the correctness, and writing the values to the struct.

Do not follow Implementation 2, it will take 4 hours.