Assignment 1 (Due: Oct 3) [CLO 3]

Instructions:

- Start from day 1. Submit to MS Teams before due time. Do not delay submission for the last day(s). Late submissions will not be accepted.
- Before submission, remove all the debugging and temporary files. Only submit the .cpp and .h files (no visual studio or other files). Delete the .vs or any other hidden folder before submission.
- Select .cpp and .h files and compress them using your full registration number and name, (e.g., `04072412007-Ali-Ahmad.zip`).
- Avoid using conio.h, as it is not part of standard C++. Don't use clear screen function. Don't use getch or other windows-specific/non-standard functions.
- The source code should be properly indented and commented.
- Any genuine efforts in each part, would result in at least 50% marks (for that part). Make sure you put your best efforts into every part. Each part carries its own marks.
- You are getting 50% marks for any genuine efforts in all the parts to encourage you to learn, even if your program does not compile and is full of bugs. Therefore, please do not plagiarize! Plagiarism includes taking or giving help in any form including but not limited to code, concept or idea for the solution, algorithm, or pseudocode. Taking help from any source including but not limited to classmates, seniors, internet, or GenAI (Chat GPT, Gemini, Meta, Deepseek, etc.) is strictly prohibited. In case your code is plagiarized, you'll get -50% absolute marks of the whole assignment. For example, if the assignment is of 100 marks, you will get -50 marks. Even a single plagiarized statement will count as plagiarism for the whole assignment. Plagiarism in two assignments may result in getting failed in the course.

This assignment will help us in understanding the relation between execution time and time complexity. We will measure the execution times of algorithms having different best case time complexities. And measure the execution times of algorithms having same worst-case time complexities, but different approaches (like using algorithms which do less or more swaps). You will implement the following algorithms:

a. Correct selection sort (in which at most O(n) swaps are performed in best case)
b. Wrong selection sort (in which O(n^2) swaps are performed in worst case)
c. Bubble sort (stop sorting, if there is no swap in an iteration)
d. Insertion sort (do not do any swapping, just shift the values to appropriate positions)
e. For having average, best, and worst cases, create three arrays A, B and C.
   - Array A should contain random values. You can use *std::rand()* function for generating random values: https://en.cppreference.com/w/cpp/numeric/random/rand.html
   - Array B should contain values sorted in ascending order.
   - Array C should contain values sorted in descending order.
f. You also need to learn how to measure execution time, this can be done using *chrono*: https://en.cppreference.com/w/cpp/chrono
g. Now do the following tasks:
   i. Measure how much time each of the four algorithms takes to sort the random array (A). Make sure, you generate a new random array before calling the sort function. Only measure the time for sorting the array (not for the whole program or generating arrays).
   ii. Measure how much time each of the three algorithms takes to sort an already sorted (ascending) array (B).
   iii. Measure how much time each of the three algorithms takes to sort an already sorted (descending) array (C). Do not reuse this array for next experiments, as the array would become sorted.
   iv. Instead of getting results for a single run, repeat the experiments 10 times (each time with different numbers in the arrays) and take the average time of 10 runs.
   v. There should be a single program to do all the tasks. There should be no input taken from the user. The size of array should be given as a command line argument.
   vi. Generate a CSV file for results. The CSV file should have the columns. (Algorithm, Array Size, Array Type (RAND/ASC/DESC), Average Time Taken). The name of the results file should be given as command line argument. For example, if your compiled program is called sorting.exe, running the command *sorting.exe 10000 results.csv* should create arrays of size 10000 each and store the results in results.csv
   vii. Run the experiments for at least 100,000 values (as it might take sever hours to run all the experiments, you can design your program such that it resumes the experiments, if the computer shuts down or the program gets closed).
   viii. Open the .csv file in MS-Excel (or LibreOffice Calc). Analyze it. Discuss in detail which algorithms perform the best under what conditions? And why? Which are worst? And why? Write a detailed report supported by appropriate tables and charts showing the performance of different algorithms.