CS111-Design and Analysis of Algorithms
2nd Semester, 2024-2025

Programming Project 1- Empirical Analysis of Sorting Algorithms

This project involves conducting an empirical analysis of six sorting algorithms—Selection Sort, Bubble Sort, Insertion Sort, Mergesort, Quicksort, and Heapsort—by sorting randomly generated integers and evaluating their performance.

Project Requirements

A. Desired Program Behavior

1. **User Input**
   - The program prompts the user to enter a value `N` (number of integers to be sorted).
   - The user selects one of two data generation methods:

     a) **Randomly Generated Integers**
     - The program generates N integers uniformly at random from the range `[0, MAX_RANGE]` (`MAX_RANGE` is an `unsigned long int` constant).

     b) **Increasing Sequence**
     - The program prompts the user to enter a starting value `X` (a positive number).
     - It generates an increasing sequence where:
       - The first element is `X`,
       - The second element is `X+1`,
       - The third element is `X+2`, and so on, for `N` elements.
2. **Sorting and Output**
   - The program sorts the generated integers using all six sorting algorithms. Each sorting algorithms should be implemented as functions not as individual programs.
   - It writes both the original and sorted values to an output file.
3. **Performance Measurement**
   - The program measures and outputs the computation time for sorting `N` integers for each algorithm.
   - Computation time should exclude user interaction, number generation, and file I/O.
   - The `clock()` function from `<time.h>` should be used to measure execution time:

```
#include <time.h>

clock_t start, end;
double cpu_time_used;

start = clock();
// Sorting operation
end = clock();
cpu_time_used = ((double) (end-start)) / CLOCKS_PER_SEC;
```

B. Empirical Analysis of Running Time

Ensure that you run the program on the same computer throughout all experiments to prevent variations in running time due to differences in system specifications.

1. **Random Input Case:**
   - Compute the **average execution time over five runs** for different values of `N`.
   - Select five different values of N, i.e. `N=10`, `N=100`, `N=1000`, `N=10000`, `N=100000`, `N=1000000`.
   - For each `N`, run the program **five times**, each time generating a new random array.
   - Record the execution time for each sorting algorithm.
   - Compute the **average execution time** over the five runs.
   - Present the results in a table format.

     Example Table for Random Input:

Result of the Experiment for Random Input, N=10

Sorting Algorithm	Run 1	Run 2	Run 3	Run 4	Run 5	Avg Time for N=10
Selection Sort	0.045s	0.048s	0.047s	0.046s	0.049s	0.047s
Bubble Sort	0.120s	0.122s	0.121s	0.125s	0.119s	0.121s
...	...	...	...	...	...	...

2. **Sorted Input Case:**
   - Repeat the experiment using **already sorted input (increasing sequence)**.
   - Record the execution times and compute the average over five runs.
   - Present results in a table similar to the one above.

     Example Table for Sorted Input:

Result of the Experiment for Sorted Input, N=10

Sorting Algorithm	Run 1	Run 2	Run 3	Run 4	Run 5	Avg Time for N=10
Selection Sort	0.005s	0.006s	0.005s	0.005s	0.006s	0.005s
Bubble Sort	0.001s	0.001s	0.001s	0.001s	0.001s	0.001s
...	...	...	...	...	...	...

3. **Graphical Representation:**
   - The final results should be presented in **both table and graphical formats**.
   - The rows in the table represent different values of `N`, while the columns represent the average execution times for each sorting algorithm.
   - A graph should plot `N` on the x-axis and the average execution time on the y-axis, with different lines representing different sorting algorithms.

Table Format:

Average Running Time for an Input Array that is Random

N	Selection Sort	Bubble Sort	Insertion sort	Mergesort	Quicksort	Heapsort
10						
100						
1000						
10000						
100000						
1000000						

Average Running Time for an Input Array that is Sorted

N	Selection Sort	Bubble Sort	Insertion sort	Mergesort	Quicksort	Heapsort
10						
100						
1000						
10000						
100000						
1000000						

To be submitted are the following:

- Source code in C or C++
- A very **detailed written report** which contains the following:
  - documentation of your program describing the structure of your code
  - screenshots showing the actual execution of the program
  - analysis of the outputs as specified above(discuss the results of your analysis and provide your conclusion, details about the specifications of the computer used should be provided also)
  - challenges you have encountered and the participation/contribution of each member in making the project

Note: Submissions consisting solely of source code will not be accepted. A program demonstration and defense are required.