ASSIGNMENT NO 1

Title: Qno-2 Sorting Algorithms Analysis

Made By Abdullah Usman

Submitted to Dr. Rabeeh A. Abbasi

---

1. Overview:

In this part of the assignment, I've implemented 4 types of Sorting Algorithms

- Bubble Sort
- Insertion Sort
- Selection Sort (Correct Method)
- Selection Sort (Incorrect Method)

After writing all three I measured their execution time on different types of arrays. After 10 runs I calculate their average.

2. Creating Arrays:

First array **(Array_A)** is randomly generated for which I used the **rand()** function from library **stdlib.h** , since the rand() function follows a same pattern and generates same values after and after

I also use **srand()** that takes **time(0**) (which is current time) as a parameter and helps generate new random values every time I also defined the range in generating random array function .

Arr_A[i]=rand()% range

Second array **(Array_B)** simply contains ascending order sort values. I run the **for loop** for size of the array and store the value of iterator like **(i)** in my case. But to have different numbers each time the Program is run I passed a variable as an argument in the function. The value of this variable changes everytime program is executed thus creating different elements in array everytime. I read this variable from binary file**(Average_time.dat)** where I store the execution time values of different algorithms for different arrays.

Third array **(Array_C)** contains values sorted in the descending order. The function implementation is same as **Array_B** but instead I start the loop from reverse and store the value of I so that I get the descending sorted array.

3. Calculating Execution Time:

I take 3 different arrays with size 1000.

- Array_A which contains random generated Values.
- Array_B which contains sorted values in ascending order.
- Array_C which contains sorted values in descending order.

To measure the time taken by each algorithm to sort different types of arrays I used the **std::chrono::time_point**. I declared a const auto variable and assign it the value of exact time of the system clock (**std::chrono::high_resolution_clock::now()**) which returns the current time on the system clock. After calling the sort function I did the same. Then I just typecast their value in microseconds then subtracted the start value from end and call the .count() function to convert it into an integer and store its value.

4. Calculating Average:

For Random Array_A

Algorithm Name	Array Type	Array Size	Time Taken (1) ms	Time Taken (2) ms	Time Taken (3)ms	Time Taken (4) ms	Time Taken (5)	Time Taken (6) ms	Time Taken (7) ms	Time Taken (8) ms	Time Taken (9)ms	Time Taken (10) ms	Average Time Taken ms
Selection Sort (C)	Random	1000	1	1	1	1	1	1	1	1	0	1	0.9
Selection Sort (W)	----------	------	2	1	1	1	1	1	1	1	2	1	1.2
Bubble Sort	----------	------	4	4	5	4	7	7	5	5	5	4	5
Insertion Sort	----------	------	0	0	1	0	0	1	1	1	1	0	0.5

For Sorted Array_B

Algorithm Name	Array Type	Array Size	Time Taken (1) ms	Time Taken (2) ms	Time Taken (3)ms	Time Taken (4) ms	Time Taken (5)	Time Taken (6) ms	Time Taken (7) ms	Time Taken (8) ms	Time Taken (9)ms	Time Taken (10) ms	Average Time Taken ms
Selection Sort (C)	Ascending Sort	1000	0	0	0	0	0	0	0	0	0	0	0
Selection Sort (W)	----------	------	1	5	1	1	1	1	1	1	2	1	1.5
Bubble Sort	----------	------	2	2	1	1	1	1	2	2	1	1	1.4
Insertion Sort	----------	------	0	0	0	0	0	0	0	0	0	0	0

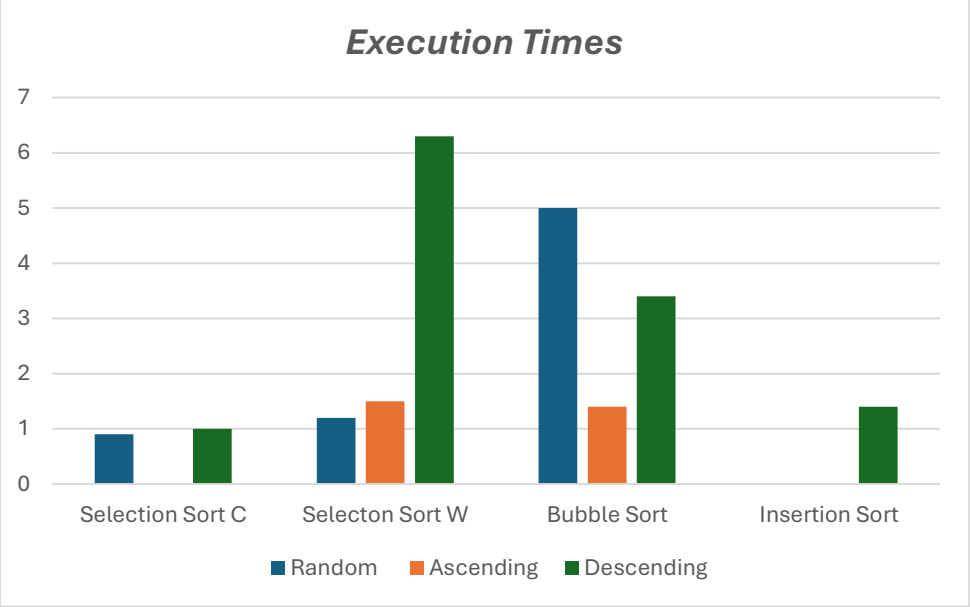
For Descending Array_C

Algorithm Name	Array Type	Array Size	Time Taken (1) ms	Time Taken (2) ms	Time Taken (3)ms	Time Taken (4) ms	Time Taken (5)	Time Taken (6) ms	Time Taken (7) ms	Time Taken (8) ms	Time Taken (9)ms	Time Taken (10) ms	Average Time Taken ms
Selection Sort (C)	Descending Sort	1000	1	1	1	1	1	1	1	1	1	1	1
Selection Sort (W)	----------	------	6	5	6	8	5	7	6	6	6	8	6.3
Bubble Sort	----------	------	5	3	3	3	3	3	4	3	4	3	3.4
Insertion Sort	----------	------	1	1	1	1	2	1	2	1	2	2	1.4

5. How I Got the average result value:

I created 3 arrays. One for Random Array, Second for Ascending Array , Third for Descending array each with size 4. On each index of every array, I am storing the result of taken by 4 different

algorithms. Then I write them inside a binary file (**Average_time.dat**) with each run. First I open the file as input mode and read the number of times the program is executed. And look for any previous data. After reading data in temporary array (if file is not empty) I calculate the time of execution of each array with every algorithm and store in my above arrays. Then I add the temporary array data (which is basically the results of execution of previous program execution). When the program is executed 10^{th} time now after reading and adding the data in my arrays they hold the results of 10 average times thus I divide them by 10 and write all the data in csv file **(Algorithms.csv)**



6. Performance of Algorithms

Best Performance:

Insertion Sort Performs best in **Best Case** and **Random Array.**

Reason: It is because the insertion sort does not perform swaps in any case

Correct Selection Sort also takes 0 milli second in best case.

Bubble Sort Performs Worst in Random case.