|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | | **Kingdom of Saudi Arabia**  **Ministry of Education**  **University of Jeddah**  **College of Science and Computer Engineering** |  | **المملكة العربية السعودية**  **وزارة التعليم**  **جامعة جدّة**  **كلية علوم و هندسة الحاسب** | |  |  |

**CCCS 314 – Design and Analysis of Algorithms**

**LAB 1**

**Topic:**

1. **Design of algorithm to find greatest common divisor**
2. **Implementation of Euclid’s algorithm**
3. **Time efficiency analysis and algorithm improvement**
4. **Empirical Analysis of Performance and Algorithm Visualization**

**Total Marks: 4**

**Student Name: bassam alghamdi**

**Student ID: 2141362**

**Marks:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Exercises | 1 | 2 | 3 | 4 | Total |
| Allocated | 1 | 1 | 1 |  | 4 |
| Obtained |  |  |  |  |  |
| **CLO, PLO** | 2.1, S1 | 2.1, S1 | 2.1, S1 | 2.1, S1 |  |

**CLO** **Marks:**

|  |  |  |
| --- | --- | --- |
|  | CLO2.1, S1 | Total |
| Allocated | 4 | 4 |
| Obtained |  |  |

**Exercise 1:** **(Design of algorithm to find greatest common divisor)**

In mathematics, the **greatest common divisor** (**gcd**) of two or more integers is the largest positive integer that divides each of the integers.

For example, the **gcd** of 8 and 12 is 4. Why?

Divisors of 8 are 1, 2, 4, 8.

Divisors of 12 are 1, 2, 4, 6, 12

Thus, the common divisors of 8 and 12 are 1, 2, 4.

Out of these common divisors, the greatest one is 4. Therefore, the greatest common divisor (**gcd**) of 8 and 12 is 4.

Write a programming code for a function FindGCD(*m,n*) that find the greatest common divisor. You can use any language of Java/C++/Python/Octave.

**[CLO2.1, S1, 1 Mark]**

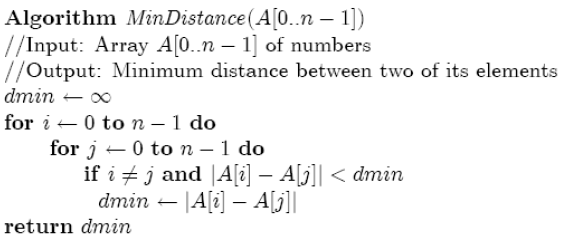
|  |
| --- |
| Find GCD Algorithm :  Step 1 Make an array to store common divisors of two integers m, n.  Step 2 Check all the integers from 1 to minimun(m,n) whether they divide both m, n. If yes, add it to the array.  Step 3 Return the maximum number in the array. |

|  |
| --- |
| Write your programming code here :  def gcd(a, b):  while b != 0:  a, b = b, a % b  return a |

**Exercise 3:** **(Time efficiency analysis and algorithm improvement)**

Consider the following algorithm for finding the distance between the two closest elements in an array of numbers.

**[CLO2.1, S1, 1 Mark]**



1. How many comparisons does the above algorithm has?

Answer: the algorithm has n^2 - n comparisons.

1. Make as many improvements as you can in this algorithm to improve the time efficiency. If you need to, you can change the whole algorithm; if not, improve the implementation given. Then, explain why your algorithm is more efficient than the above algorithm.

|  |
| --- |
| Write your improved algorithm here :  def min\_distance\_sorted(A):  n = len(A)  A.sort()  dmin = float('inf')  for i in range(n - 1):  if abs(A[i] - A[i+1]) < dmin:  dmin = abs(A[i] - A[i+1])  return dmin  This algorithm is more efficient than the original algorithm because it avoids comparing every pair of elements in the input list, and instead finds the minimum absolute difference between adjacent elements in a single pass through the sorted list. |

**Exercise 4:** **(Empirical Analysis of Performance and Algorithm Visualization)**

**[CLO2.1, S1, 1 Mark]**

1. Take an available implementation of the Selection Sort algorithm. Run it on a randomly generated data of fixed size and note down the execution time. Repeat the experiment 10 times on the same data and fill the table below.

|  |  |
| --- | --- |
| Trial # | Execution Time (ms) |
| 1 | Execution time: 1.14226 milliseconds |
| 2 | Execution time: 1.55807 milliseconds |
| 3 | Execution time: 1.45650 milliseconds |
| 4 | Execution time: 1.02782 milliseconds |
| 5 | Execution time: 0.99444 milliseconds |
| 6 | Execution time: 1.02568 milliseconds |
| 7 | Execution time: 1.25074 milliseconds |
| 8 | Execution time: 1.12963 milliseconds |
| 9 | Execution time: 0.52071 milliseconds |
| 10 | Execution time: 1.27077 milliseconds |

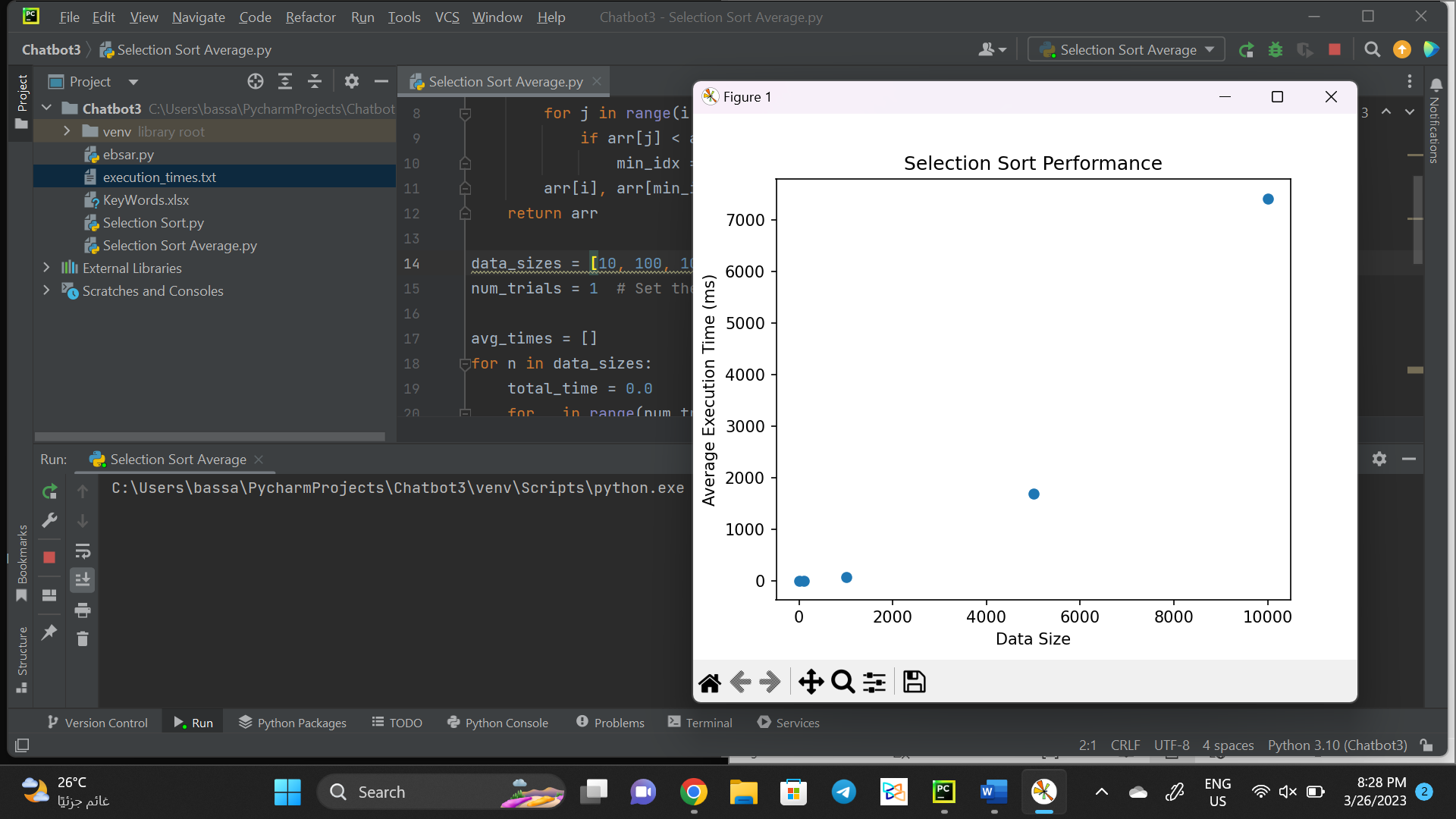
1. Why is the execution time different for each trial?

|  |
| --- |
| Due to factors such as CPU load, caching, and other system events. |

1. A good way to approximate the execution time is to take the average time of multiple trials. Do this for different data sizes and fill the table below:

|  |  |
| --- | --- |
| Data Size (number of elements) | Average Execution Time of 10 trials (ms) |
| 10 | Average execution time for n=10: 0.000000 milliseconds |
| 100 | Average execution time for n=100: 0.262427 milliseconds |
| 1000 | Average execution time for n=1000: 35.542464 milliseconds |
| 5000 | Average execution time for n=5000: 905.038142 milliseconds |
| 10,000 | Average execution time for n=10000: 3829.093146 milliseconds |
| 50,000 | Average execution time for n=50000: 209078.652382 milliseconds |
| 100,000 | Average execution time for n=100000: 838408.197975 milliseconds |
| 500,000 | Average execution time for n=500000: 147833631.808281 milliseconds |
| 1,000,000 | Average execution time for n=1000000: 568031964.302444 milliseconds |

1. Plot the data in the table above. Average time along y-axis and the data size along x-axis.



1. The time complexity of **Selection** **Sort** algorithm is O(n^2). Plot it and verify that the curve looks like the one in (d).

