I was hired as a software engineer for a logistics company to minimize the time and resources spent on deliveries.

# Problem 1: Mathematical Foundations (25 points)

## Formula:

The formula used for the distance (*d*) between two points and is calculated as follows.

Additionally, the summation formula for the first *N* natural numbers is as follows:

## Task:

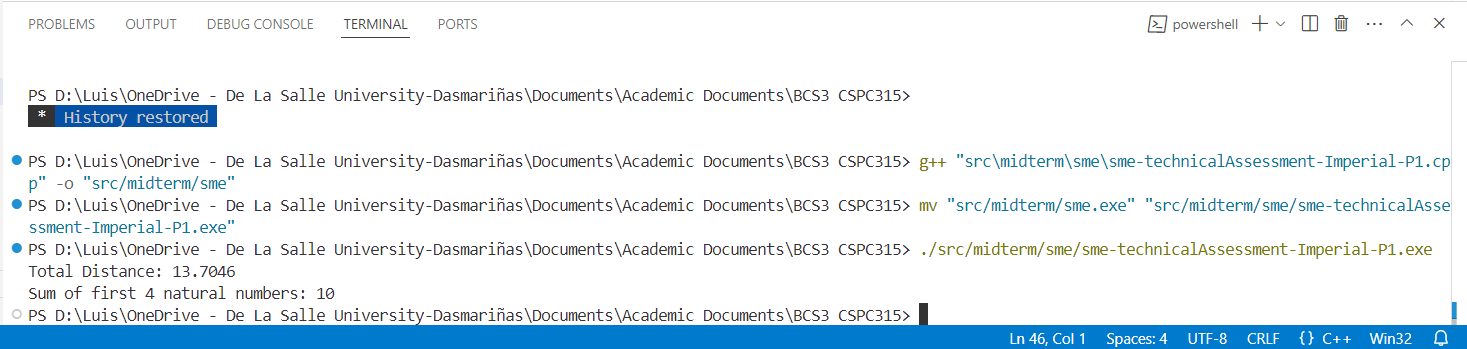
The program I have written computes the total distance traveled, given *N* delivery points in a 2D space, for a given sequence of points.

## Codebase:

*View this code on GitHub at:* [*https://github.com/LuisAPI/BCS3-CSPC315/blob/main/src/midterm/sme/sme-technicalAssessment-Imperial-P1.cpp*](https://github.com/LuisAPI/BCS3-CSPC315/blob/main/src/midterm/sme/sme-technicalAssessment-Imperial-P1.cpp)

|  |  |
| --- | --- |
|  |  |

## Output:



# Problem 2: Algorithmic Complexity (30 points)

## Task:

I used two algorithms, bubble sort and merge sort, in categorizing delivery routes by efficiency. Printed during the computation is the time complexity during the process.

## Result:

For smaller inputs (such as 5 deliveries), the two sorting algorithms finished instantaneously, which is more a reflection of the strength of modern computing devices than of any of the two algorithms.

Bubble sort ended up being less efficient than merge sort when the input was large, meaning it had 10,000 deliveries to go through. It took up 100 times more microseconds than merge sort. This is because **bubble sort has a time complexity of O(n²)** while merge sort’s TC is only **O(n log n)**.

## Input:

We considered different input sizes for the program. Shown below is the result when:

* There is a small array of input consisting of 5 deliveries,
* There is a medium-sized array consisting of 100 deliveries, and
* There is a large array of input consisting of 10,000 deliveries,

with randomly-generated distances.

## Codebase:

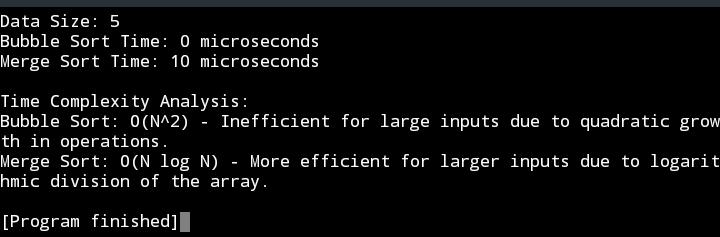
*View the codebase on GitHub at:* [*https://github.com/LuisAPI/BCS3-CSPC315/blob/main/src/midterm/sme/sme-technicalAssessment-Imperial-P2.cpp*](https://github.com/LuisAPI/BCS3-CSPC315/blob/main/src/midterm/sme/sme-technicalAssessment-Imperial-P2.cpp)*.*

|  |  |
| --- | --- |
|  | |
|  |  |

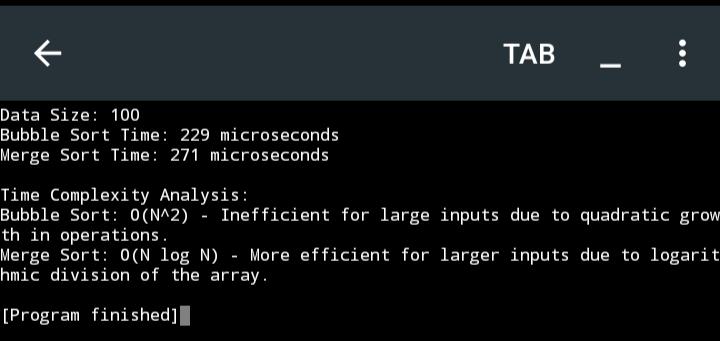
## Output:

*Note that some screenshots were taken on a C++ compiler on dark theme, rather than on light as usual. My day-to-day laptop ran out of charge as I was solving this problem.*

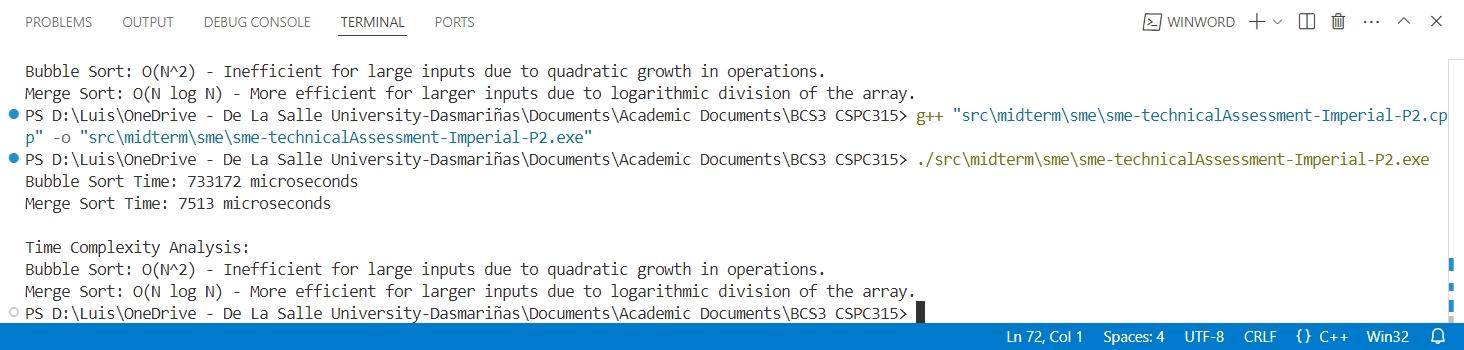
### Output for Small Inputs:



### Output for Medium-Sized Inputs:



### Output for Large Inputs:



# Problem 3: Recursive Algorithms (45 points)

## Task:

For our final problem, I decided to replicate the Tower of Hanoi, an old mathematics puzzle, in C++ script form. The Tower of Hanoi’s goal is, with three towers, to move all discs from one tower to another, following the rule that only one can be moved at a time and that a larger disc cannot be placed on top of a smaller one.

## Process:

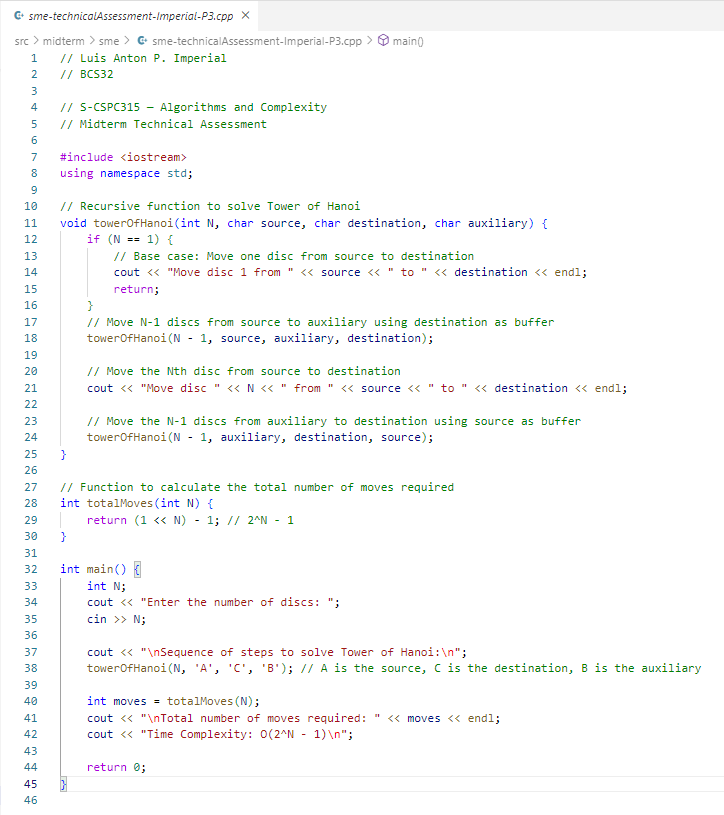
With N as the number of discs to move, we can:

1. Move the top N-1 discs from Tower 1 to Tower 2,
2. Move the Nth disc from Tower 1 to Tower 3, and
3. Move the N-1 discs from the Tower 2 to Tower 3.

The time complexity is .

### Codebase:

*View this on GitHub at:* [*https://github.com/LuisAPI/BCS3-CSPC315/blob/main/src/midterm/sme/sme-technicalAssessment-Imperial-P3.cpp*](https://github.com/LuisAPI/BCS3-CSPC315/blob/main/src/midterm/sme/sme-technicalAssessment-Imperial-P3.cpp)



## Output:

Number of discs entered is 5.

