Luis Anton P. Imperial BCS32	S-CSPC315 Algorithms and Complexity	241007 – Midterm Technical Assessment Wednesday, October 9, 2024

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 (x_1, y_1) and (x_2, y_2) is calculated as follows.

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

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

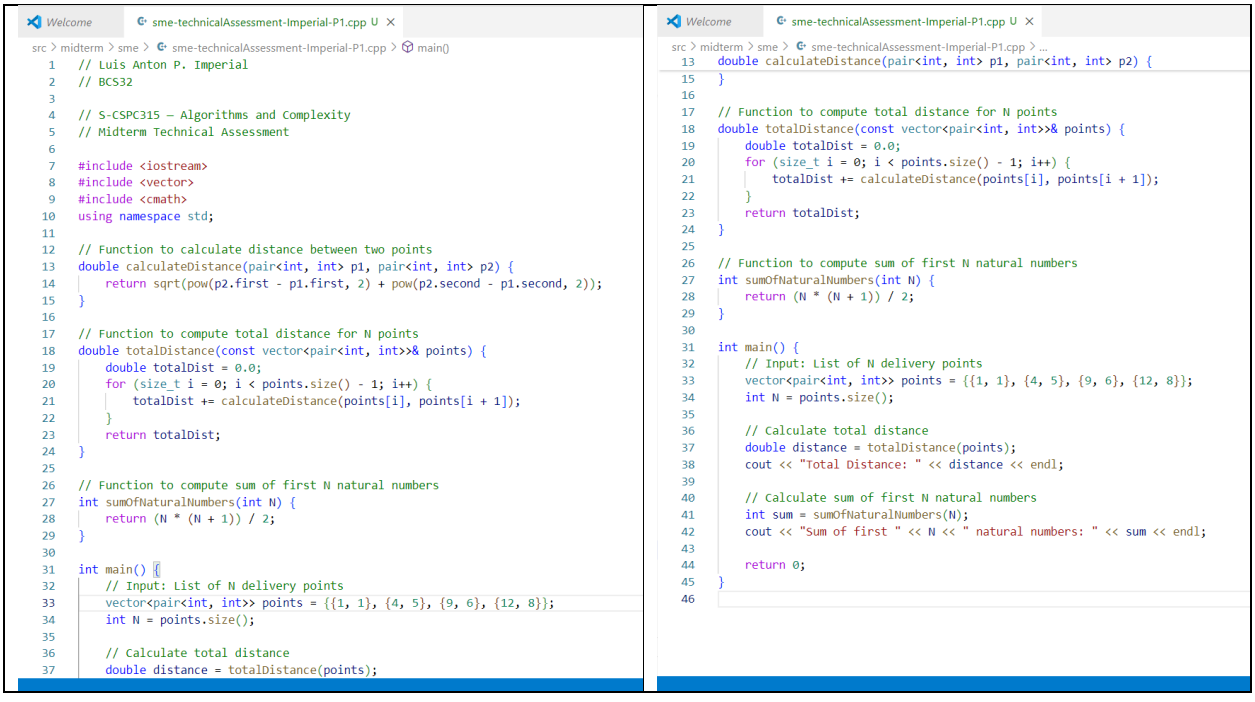
$$\sum_{1}^{N} = \frac{N * (N + 1)}{2}$$

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.

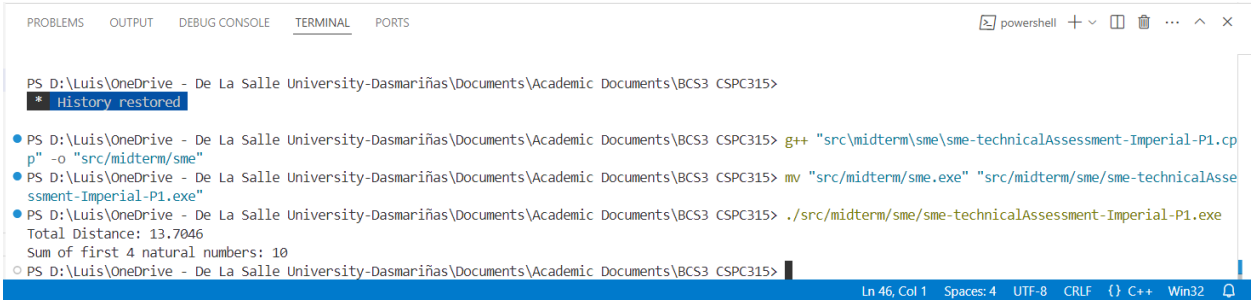
Codebase:

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



```cpp
// Luis Anton P. Imperial
// BCS32

// S-CSPC315 – Algorithms and Complexity
// Midterm Technical Assessment

#include <iostream>
#include <vector>
#include <cmath>
using namespace std;

// Function to calculate distance between two points
double calculateDistance(pair<int, int> p1, pair<int, int> p2) {
    return sqrt(pow(p2.first - p1.first, 2) + pow(p2.second - p1.second, 2));
}

// Function to compute total distance for N points
double totalDistance(const vector<pair<int, int>>& points) {
    double totalDist = 0.0;
    for (size_t i = 0; i < points.size() - 1; i++) {
        totalDist += calculateDistance(points[i], points[i + 1]);
    }
    return totalDist;
}

// Function to compute sum of first N natural numbers
int sumOfNaturalNumbers(int N) {
    return (N * (N + 1)) / 2;
}

int main() {
    // Input: List of N delivery points
    vector<pair<int, int>> points = {{1, 1}, {4, 5}, {9, 6}, {12, 8}};
    int N = points.size();

    // Calculate total distance
    double distance = totalDistance(points);
    cout << "Total Distance: " << distance << endl;

    // Calculate sum of first N natural numbers
    int sum = sumOfNaturalNumbers(N);
    cout << "Sum of first " << N << " natural numbers: " << sum << endl;

    return 0;
}
```

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 $O(n^2)$** while merge sort's TC is only **O(n log n)**.

Input:

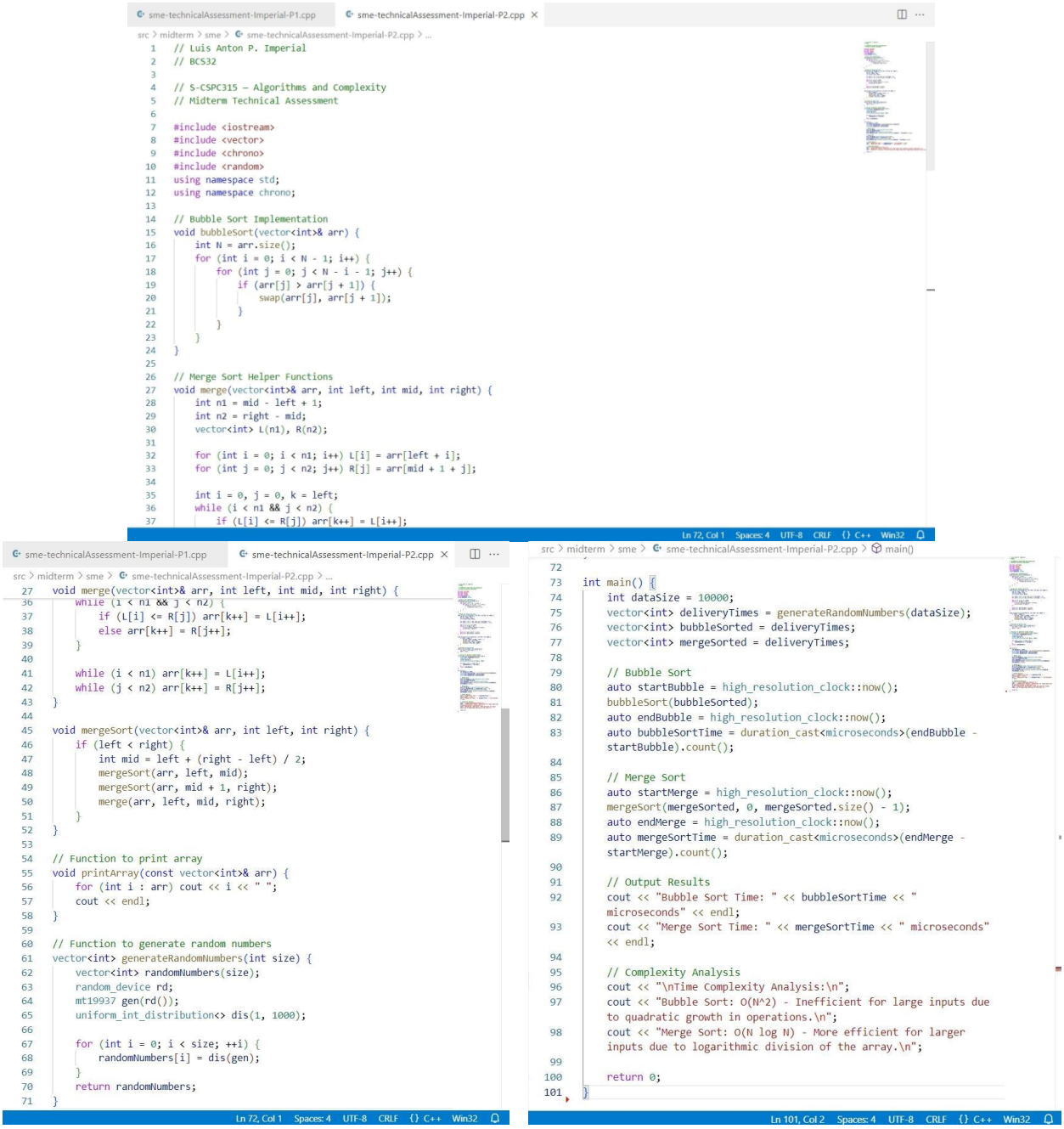
We considered different input sizes for the program. Shown below is the result when:

- 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.



```cpp
// Luis Anton P. Imperial
// BCS32

// S-CSPC315 – Algorithms and Complexity
// Midterm Technical Assessment

#include <iostream>
#include <vector>
#include <chrono>
#include <random>
using namespace std;
using namespace chrono;

// Bubble Sort Implementation
void bubbleSort(vector<int>& arr) {
    int N = arr.size();
    for (int i = 0; i < N - 1; i++) {
        for (int j = 0; j < N - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                swap(arr[j], arr[j + 1]);
            }
        }
    }
}

// Merge Sort Helper Functions
void merge(vector<int>& arr, int left, int mid, int right) {
    int n1 = mid - left + 1;
    int n2 = right - mid;
    vector<int> L(n1), R(n2);

    for (int i = 0; i < n1; i++) L[i] = arr[left + i];
    for (int j = 0; j < n2; j++) R[j] = arr[mid + 1 + j];

    int i = 0, j = 0, k = left;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) arr[k++] = L[i++];
        else arr[k++] = R[j++];
    }

    while (i < n1) arr[k++] = L[i++];
    while (j < n2) arr[k++] = R[j++];
}

void mergeSort(vector<int>& arr, int left, int right) {
    if (left < right) {
        int mid = left + (right - left) / 2;
        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);
        merge(arr, left, mid, right);
    }
}

// Function to print array
void printArray(const vector<int>& arr) {
    for (int i : arr) cout << i << " ";
    cout << endl;
}

// Function to generate random numbers
vector<int> generateRandomNumbers(int size) {
    vector<int> randomNumbers(size);
    random_device rd;
    mt19937 gen(rd());
    uniform_int_distribution<> dis(1, 1000);

    for (int i = 0; i < size; ++i) {
        randomNumbers[i] = dis(gen);
    }
    return randomNumbers;
}

int main() {
    int dataSize = 10000;
    vector<int> deliveryTimes = generateRandomNumbers(dataSize);
    vector<int> bubbleSorted = deliveryTimes;
    vector<int> mergeSorted = deliveryTimes;

    // Bubble Sort
    auto startBubble = high_resolution_clock::now();
    bubbleSort(bubbleSorted);
    auto endBubble = high_resolution_clock::now();
    auto bubbleSortTime = duration_cast<microseconds>(endBubble -
    startBubble).count();

    // Merge Sort
    auto startMerge = high_resolution_clock::now();
    mergeSort(mergeSorted, 0, mergeSorted.size() - 1);
    auto endMerge = high_resolution_clock::now();
    auto mergeSortTime = duration_cast<microseconds>(endMerge -
    startMerge).count();

    // Output Results
    cout << "Bubble Sort Time: " << bubbleSortTime << "
    microseconds" << endl;
    cout << "Merge Sort Time: " << mergeSortTime << " microseconds"
    << endl;

    // Complexity Analysis
    cout << "\nTime Complexity Analysis:\n";
    cout << "Bubble Sort: O(N^2) - Inefficient for large inputs due
    to quadratic growth in operations.\n";
    cout << "Merge Sort: O(N log N) - More efficient for larger
    inputs due to logarithmic division of the array.\n";

    return 0;
}
```

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:

```
Data Size: 5
Bubble Sort Time: 0 microseconds
Merge Sort Time: 10 microseconds

Time Complexity Analysis:
Bubble Sort: O(N^2) - Inefficient for large inputs due to quadratic grow
th in operations.
Merge Sort: O(N log N) - More efficient for larger inputs due to logarit
hmic division of the array.
```

Output for Medium-Sized Inputs:

```
Data Size: 100
Bubble Sort Time: 229 microseconds
Merge Sort Time: 271 microseconds

Time Complexity Analysis:
Bubble Sort: O(N^2) - Inefficient for large inputs due to quadratic grow
th in operations.
Merge Sort: O(N log N) - More efficient for larger inputs due to logarit
hmic division of the array.
```

Output for Large Inputs:

```
Bubble Sort Time: 733172 microseconds
Merge Sort Time: 7513 microseconds

Time Complexity Analysis:
Bubble Sort: O(N^2) - Inefficient for large inputs due to quadratic growth in operations.
Merge Sort: O(N log N) - More efficient for larger inputs due to logarithmic division of the array._
```

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 $O(2^N - 1)$.

Codebase:

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

sme-technicalAssessment-Imperial-P3.cpp

src > midterm > sme > sme-technicalAssessment-Imperial-P3.cpp > main()

```cpp
// Luis Anton P. Imperial
// BCS32

// S-CSPC315 – Algorithms and Complexity
// Midterm Technical Assessment

#include <iostream>
using namespace std;

// Recursive function to solve Tower of Hanoi
void towerOfHanoi(int N, char source, char destination, char auxiliary) {
    if (N == 1) {
        // Base case: Move one disc from source to destination
        cout << "Move disc 1 from " << source << " to " << destination << endl;
        return;
    }
    // Move N-1 discs from source to auxiliary using destination as buffer
    towerOfHanoi(N - 1, source, auxiliary, destination);

    // Move the Nth disc from source to destination
    cout << "Move disc " << N << " from " << source << " to " << destination << endl;

    // Move the N-1 discs from auxiliary to destination using source as buffer
    towerOfHanoi(N - 1, auxiliary, destination, source);
}

// Function to calculate the total number of moves required
int totalMoves(int N) {
    return (1 << N) - 1; // 2^N - 1
}

int main() {
    int N;
    cout << "Enter the number of discs: ";
    cin >> N;

    cout << "\nSequence of steps to solve Tower of Hanoi:\n";
    towerOfHanoi(N, 'A', 'C', 'B'); // A is the source, C is the destination, B is the auxiliary

    int moves = totalMoves(N);
    cout << "\nTotal number of moves required: " << moves << endl;
    cout << "Time Complexity: O(2^N - 1)\n";

    return 0;
}

```

Output:

Number of discs entered is 5.

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS powershell

```
PS D:\Luis\OneDrive - De La Salle University-Dasmariñas\Documents\Academic Documents\BCS3 CSPC315> g++ "src\midterm\sme\sme-technicalAssessment-Imperial-P3.cpp" -o "src\midterm\sme\sme-technicalAssessment-Imperial-P3.exe"
PS D:\Luis\OneDrive - De La Salle University-Dasmariñas\Documents\Academic Documents\BCS3 CSPC315> ./src\midterm\sme\sme-technicalAssessment-Imperial-P3.exe
Enter the number of discs: 5

Sequence of steps to solve Tower of Hanoi:
Move disc 1 from A to C
Move disc 2 from A to B
Move disc 1 from C to B
Move disc 3 from A to C
Move disc 1 from B to A
Move disc 2 from B to C
Move disc 1 from A to C
Move disc 4 from A to B
Move disc 1 from C to B
Move disc 2 from C to A
Move disc 1 from B to A
Move disc 3 from C to B
Move disc 1 from A to C
Move disc 2 from A to B
Move disc 1 from C to B
Move disc 5 from A to C
Move disc 1 from B to A
Move disc 2 from B to C
Move disc 1 from A to C
Move disc 3 from B to A
Move disc 1 from C to B
Move disc 2 from C to A
Move disc 1 from B to A
Move disc 4 from B to C
Move disc 1 from C to B
Move disc 3 from A to C
Move disc 1 from B to A
Move disc 2 from B to C
Move disc 1 from A to C

Total number of moves required: 31
Time Complexity: O(2^N - 1)
PS D:\Luis\OneDrive - De La Salle University-Dasmariñas\Documents\Academic Documents\BCS3 CSPC315>
```