CS-302 : DESIGN AND ANALYSIS OF ALGORITHMS

PROJECT PART II

REPORT

SANA ALI KHAN

18I-0439

CS-B

1 CONTENTS

2 PROJECT STATEMENT

The 2019–20 coronavirus pandemic is an ongoing pandemic of coronavirus disease 2019 (COVID19) caused by severe acute respiratory syndrome coronavirus 2 (SARS-CoV-2). As of 13 April 2020, more than 1.84 million cases of COVID-19 have been reported in 210 countries and territories, resulting in more than 114,000 deaths. More than 421,000 people have recovered, although there may be a possibility of reinfection.

In this project, you are required to answer queries regarding the COVID-19 pandemic. You are provided with data regarding the pandemic containing information regarding the daily new cases and cumulative cases for each country.

You must read the given dataset and answer the following queries. For each query, think very carefully about your approach in regards to both the time and space complexity of your solution.

---

1) On a given day, find the top 20 countries with the most confirmed cases. **(Efficiently)**

2) Find the country(s) with the highest new cases between two given dates. **(Efficiently)**

3) Find the starting and ending days of the longest spread period for a given country. The spread period is defined as the period where daily new cases tend to increase. They may contain days where new confirmed cases were relatively lower or none at all. For example, [5, 2, 9, 16, 11, 27, 14, 45, 11] has a longest spread period elapsing 7 days from day 2 (2) to day 8 (45). **(Efficiently)**

4) Find the longest daily death toll decrease period for a given country. They may contain days where new confirmed deaths were relatively higher. For example, [9, 5, 1, 16, 11, 23, 8, 3, 27, 14, 45, 11] has a longest daily death toll decrease period of 4 {16, 11, 8, 3} **(Efficiently)**

5) You have decided to help with the relief effort by collaborating with an organization that distributes essential supplies to affected regions. However, they must consider the overhead cost for each country (distance, shipping etc.). They have assigned a score to each country, which is simply the total number of active cases on the latest day (they prefer to help countries with higher cases). A country may or may not be selected for aid. They have gathered their projected costs for each country and have tasked you with finding the highest possible score attainable as well as the countries selected given a budget of 300. **(Efficiently)**

6) We wish to compare the response of any two countries against this virus. To do this, we must compare how similar the change of their daily active cases is. The similarity is measured by the longest number of days their daily active cases share similar values. However, since it is extremely unlikely that two countries have the exact same values for a sequence of days, we define a compare threshold K such that any two countries are said to have 'similar' daily active cases on a given day if their active cases differ <= K. For example, C1 = [0, 0, 0, 1, 6, 10, 17, 27, 48, 94] & C2 = [3, 4, 8, 14, 31, 32, 49] and K = 10 similarity (C1, C2) = 5 (C1[3:7] = C2[0:4]). Given two countries and the compare threshold K, find their similarity. **(Efficiently)**

3 QUERY 1

Objective : On a given day, find the top 20 countries with the most confirmed cases. (Efficiently)

3.1 EXPLANATION:

This query first requires a data structure to hold the given data of a country (on one date) i.e. one row of the data provided in the file "WHO-COVID-19.csv". A user-defined datatype, the countryData structure, is created for this purpose, with the following attributes (methods not included):

```
struct countryData {
	string day;
	string country;
	string countryName;
	string region;
	int deaths;
	int cumDeaths;
	int confirmed;
	int cumConfirmed;
};
```

A class named query1 will use this to solve the given query. It stores a chosen date, and maintains a priority queue for storing countries – this is the main data structure used for this query. The queue used is from the C++ Standard Template Library (STL). The underlying data structure used by this queue is heap.

The time complexity of push() and pop() operations of this queue is of O(log(n)), and the space complexity is n, where n is the number of items in the queue.

Initially, the user chooses a date for which they want the query answered. When the file is read, the data of a row/country is stored in a temporary object of type countryData. If the date stored in it matches the chosen date, then this object is pushed into the queue. This is repeated for every row of the file and results in the queue containing the records of all the countries for that specific date.

The priority of each item in the queue depends on the value of its "cumConfirmed", which is simply the total number of confirmed cases. To find the 20 countries with the most confirmed cases, top() is used to extract the highest priority item for the queue and display its data. That item is then popped, the next one extracted and so on until the top twenty countries have been taken out from the queue.

3.2 COMPLEXITY ANALYSIS:

For n number of items in the queue, this means that top() and pop() (for the queue) and display() are performed at most twenty times and at the minimum zero – in the case where there are no records for the date provided. The cost of file-reading is not taken into account.

So, worst case complexity resolves to [(20 * log(n)) + (20 * log(n)) + (20 * 1)]. After disregarding constants, overall complexity of this algorithm is O(log(n)).

```
int count = 1;
while (!countries.empty() and count <= 20) {
    countryData temp = countries.top();
    countries.pop();

    cout << count << ":  ";
    temp.display();
    count++;
}
```

Time complexity	Space complexity
O(log(n))	O(n)

4 QUERY 2

Objective : Find the country(s) with the highest new cases between two given dates. (Efficiently)

4.1 EXPLANATION:

This query first requires a data structure to hold the given data of a country (on one date) i.e. one row of the data provided in the file "WHO-COVID-19.csv". A user-defined datatype, the countryData2 structure, is created for this purpose, with the following attributes (methods not included):

```
struct countryData2 {
	string day;
	string country;
	string countryName;
	string region;
	int confirmed;
	int cumConfirmed;

	int dateDiff;
	int prevC;
	int nextC;

	bool operator < (const countryData2& obj) const {
		return this->dateDiff > obj.dateDiff;
	}
};
```

A class named query2 will use this to solve the given query. It stores the two chosen dates, and maintains a priority queue for storing countries – this is the main data structure used for this query. The queue used is from the C++ Standard Template Library (STL). The underlying data structure used by this queue is heap.

While the queue is a maximum priority queue by default, the comparison operator '<' has been overloaded in the countryData2 struct so that a queue of its objects will be a minimum priority queue.

To start off, the user chooses two dates. When the file is read, the data of a row/country is stored in a temporary object of type countryData1. If the date stored in it matches the first date, then this data is passed to another object, pendingCountry.

From there, successive rows of the file are read until there is either a match for the first date or the second date. If it is the first date, that means that was no record found for the current country (pendingCountry) that corresponded to the second date chosen. In this case, pendingCountry is now reset to the next country with data for the first date (and the next country.)

But if the second date (and the country names of temp and pendingCountry) matches, then the difference between their cumulative cases is calculated, and the confirmed cases on the first date is added to this difference. This figure is now the number of new cases between the two given dates, and is stored in pendingCountry.

Instead of inserting every pendingCountry into the queue and later parsing through it to find the countries with the greatest differences between the dates, it is easier to push countries into it selectively.

This is done by first modifying the queue so that it is a minimum priority queue. If it is empty, then the pendingCountry is pushed into it and the program moves on to the next one. However, if it is not empty, then its top element is peeked at. If the date difference figure (dateDiff) of this element is less than the date difference figure of the pendingCountry, it means that the top element (the minimum in the queue) is no longer relevant to our query and hence, it is removed.

it is possible there are other objects in the queue with date difference figures that may be less than that of the pendingCountry, a there is a loop that goes through the queue and removes all such objects.

```
if (temp.day == date1) {
    pendingCountry = temp;
    pendingCountry.prevC = temp.cumConfirmed;
}

if (temp.day == date2 and temp.country == pendingCountry.country) {
    pendingCountry.nextC = temp.cumConfirmed;
    pendingCountry.dateDiff = pendingCountry.nextC - pendingCountry.prevC;
    pendingCountry.dateDiff += pendingCountry.confirmed;

    if (countries.empty())
        countries.push(pendingCountry);

    else {

        temp = countries.top();

        if (temp.dateDiff == pendingCountry.dateDiff)
            countries.push(pendingCountry);

        else {
            bool flag = true;

            while (temp.dateDiff < pendingCountry.dateDiff) {
                countries.pop();

                if (flag) {
                    countries.push(pendingCountry);
                    flag = false;
                }

                temp = countries.top();
            }
        }
```

This ensures that the queue always has countries with the maximum number of new cases between the given dates.

After the program has gone through the file and performed the appropriate insertions and removals from the queue, it simply iterates through the queue and displays the data of the country(s) in it.

```
cout << "\nMaximum new cases between '" << date1 << "' and " << date2
while (!countries.empty()) {

    countryData2 temp = countries.top();
    countries.pop();

    cout << "\n------------------------" << "\n\n";
    cout << "Country: " << temp.country << "\n";
    cout << "Country Name: " << temp.countryName << "\n";
    cout << "Region: " << temp.region << "\n";
    cout << "Cases (" << date1 << "): " << temp.prevC << "\n";
    cout << "Cases (" << date2 << "): " << temp.nextC << "\n";
}
```

This method was chosen because all it needs is a priority queue with the appropriate insertions and removals. By controlling how items are inserted (and their priority), it means that to obtain the end result, all that has to be done is display the values in the queue.

4.2 COMPLEXITY ANALYSIS:

The time complexity of push() and pop() operations of this queue is of O(log(n)), and the space complexity is n, where n is the number of items in the queue.

In the worst case, there are n number of countries whose new cases between the two dates is the maximum i.e. all countries have the required figure equal. So, the cost of removing and displaying all the items from it would be [(n * log(n)) + (n * 1)] – the cost of the display() function is approximated to a 1, a constant.

After disregarding the constants in the equation, it resolves to [(n(log(n)) + (n)] = O(nlog(n)).

Time complexity	Space complexity
O(nlog(n))	O(n)

5 QUERY 3

Objective : Find the starting and ending days of the longest spread period for a given country.

5.1 EXPLANATION:

This query first requires a data structure to hold the given data of a country (on one date) i.e. one row of the data provided in the file "WHO-COVID-19.csv". A user-defined datatype, the countryData3 structure, is created for this purpose, with the following attributes (methods not included):

```
struct countryData3 {
	string day;
	string country;
	string countryName;
	int confirmed;

	int dayNo;
};
```

Extra attributes of a country (used in the previous queries) have been disregarded to save space as there is no use for them.

A class named query3 will use this to solve the given query. It stores a chosen country, and maintains a vector countries for storing countries. There are two dates as well, two sizes, and two dynamic arrays, figures and spread.

Initially, the user chooses a country, then all records of that country are stored in the vector. This vector will be used to find the longest spread period of daily new cases. The algorithm used for this is the longest increasing subsequence., using dynamic programming. This was chosen at it means that the same subsequences do not have to be calculated over repeatedly.

To find this spread, figures and a temp array of type countryData3 is allocated memory, and the figures array is initialized to -1. Now, a nested loop iterates through the vector. The index of the outside loop, is index1 and the index of the inner loop is index2. For every pass of the outside loop, the "confirmed" attribute of the country at index1 of the temp array holds the length of the longest increasing spread up to that very country.

When all the countries are iterated through, this leads to the temp array having the length of the spread at every index, from the start of the array up to the element at that specific index. The outer loop selects a country, and the inner loop iterates through all the previous records in the temp array up to that country.

After the loops are completed, temp holds all the longest increasing periods, and to find the longest one among them, the one with the maximum value of "confirmed" is chosen. However, the figures array is needed now to find the values in this period.

Figures is included in the previously mentioned nested loop, as it is used alongside the temp array. It is used to store the indexes of the second last element of every spread ending at any element in the

temp array i.e. temp[index1]. The index of this second last element is obviously index2 (which iterates through temp until index1 is equal to index2.) The value of index2 is then stored in figures[index1]. This means that any element at any index of figures stores the index at which the maximum value of temp[index1] was found.

To go back through it, we start at the index with the longest length of the increasing spreads (which was found before.) Then the "confirmed" of the element at that index of the vector countries is taken and added to spread array. Then next index is taken from figures, and the next confirmed value and so on, until the spread array has the values of the longest increasing spread.

The starting date of the spread is simply the date of the first element taken from countries and ending date is of the last element.

5.2 COMPLEXITY ANALYSIS:

Space complexity needs to be considered as there are several arrays being maintained. If n is the number of records for a particular country, then the vector countries is of size n. Additionally, all the other arrays used will also be of size n in the worst case. Taking the size of the other variables as 1 (constant), complexity comes to [n + n + n + n + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1] = [4n + 11] = O(n).

When calculating time complexity, the cost of file reading is disregarded. The cost of the algorithm itself will be polynomial as there are several loops.

```
countryData3* temp = new countryData3[size_];
figures = new int[size_];

for (int i = 0; i < size_; i++) {
    figures[i] = -1;
    temp[i].confirmed = 1;
}
```

This loop that initializes the two arrays runs up to size_, which is the size of the vector i.e. n. Complexity of this loop is then equal to n.

```
int index1 = 0;

while (index1 < size_) {

    int index2 = 0;
    int prev = countries[index1].confirmed;

    while (index2 < index1) {

        if (prev > countries[index2].confirmed) {

            int next = temp[index2].confirmed + 1;

            if (temp[index1].confirmed < next) {

                temp[index1].confirmed = next;
                figures[index1] = index2;
            }
        }

        index2++;
    }

    index1++;
}
```

This is the nested loop that computes the longest spreads and their respective indexes. The outer loop runs up to size_ (that is, n) and for every iteration of this loop, the inner loop runs up to the index of the outer loop. Outer index will be 0, 1, 2....n.

So, complexity :

= [n * (0 + 1 + 2 + 3 + ... + n)]

= [0 + n + 2n + 3n + ... + n * n].

After disregarding constants, this is equal to n^2.

The loops after this are less important regarding complexity, but for the sake of detail:

```
bool flag = true;
for (int i = 0; i < size_; i++) {
    int index = figures[i];

    if (index >= 0) {
    //  cout << countries[index].confirmed << " ";

        if (flag) {
            sDate = countries[index].day;
            flag = false;
        }
    }
}
```

```
int max = INT_MIN;
int index;

for (int i = 0; i < size_; i++) {
    if (temp[i].confirmed > max) {
        max = temp[i].confirmed;
        index = i;
    }
}
```

These two loops each have n number of iterations.

```
for (int i = 0; i < spreadSize / 2; i++) {
    int temp = spread[i];
    spread[i] = spread[spreadSize - 1 - i];
    spread[spreadSize - 1 - i] = temp;
}
```

This loop simply reverses the spread array, so that the values of the longest spread period are in the correct order. It runs up to spreadSize/2. As spreadSize may be n in the worst case, it means this loop essentially has a complexity of n/2.

Time complexity of all these loops resolve to:

= [n + n^2 + n + n + n/2]

= [7n/2 + n^2]

= [n^2]

= O(n^2)

Time complexity	Space complexity
O(n^2)	O(n)

6 QUERY 4

Objective : Find the longest daily death toll decrease period for a given country.

6.1 EXPLANATION:

This query first requires a data structure to hold the given data of a country (on one date) i.e. one row of the data provided in the file "WHO-COVID-19.csv". A user-defined datatype, the countryData4 structure, is created for this purpose, with the following attributes (methods not included):

```
struct countryData4 {
	string country;

	vector<countryData4> countries;

	int size_;

	string sDate;
	string eDate;

	int* spread;
	int spreadSize;
};
```

Extra attributes of a country (used in the previous queries) have been disregarded to save space as there is no use for them.

A class named query4 will use this to solve the given query. It stores a chosen country, and maintains a vector countries for storing countries. There are two dates as well, two sizes, and a dynamic array, spread, which will contain the value of the longest period.

Initially, the user chooses a country, then all records of that country are stored in the vector. This vector will be used to find the longest decreasing spread period of daily death toll. The algorithm used for this is the longest decreasing subsequence., using dynamic programming. This was chosen at it means that the same subsequences do not have to be calculated over repeatedly.

To solve the query, an array of vectors is created, of type countryData4. This array will maintain all possible subsequences once it is computed. The initial assumption is that every daily death figure forms a subsequence on its own (hence the size of the array being n, which is the number of countries in the countries vector.)

Computation of the spread is started of by pushing the first record of countries vector into the deathTolls vector. After this, a nested loop iterates through the vector. For every pass of the outer loop (using index1), the inner loop goes through each element/vector in deathTolls up to the one at index1. Then, the sizes of the vectors at those two indexes of deathTolls are compared. If the next is greater in size than the previous, then the countries at those indexes are compared. If the deaths at

the second index are greater than those at the first index, then the country vector at first index is set equal to the one at the second.

What this means is that if the second subsequence is found to be greater (and if the deaths were higher as well), then it replaces the previous ongoing one in deathTolls. After a pass of the inner loop, the country at that index is just pushed back into the current subsquence – the one at deathTolls[index1.

Once the nested loop is computed, all that remains is to find the vector with the largest size. As each vector contains a subsequence of deaths, the one with greatest size will the be the longest decreasing subsequence. The values from this vector are passed to the spread array and its contents displayed.

6.2 COMPLEXITY ANALYSIS:

Considering space complexity, there is a vector for countries, an array for the spread, and an array of vectors. If variables of constant complexity are taken into account as well, overall complexity is:

[n + n + n * n + 1 + 1 + 1 + 1] = [n^2 + 3n + 4] = O(n^2).

The reason for the n^2 factor is that the array of vectors is of size n. In the worst case, each of the vectors in this array will contain a subsequence of size n. Therefore, n * n.

When calculating time complexity, the cost of file reading is disregarded. The cost of the algorithm itself will be polynomial as there are several loops.

```
int index1 = 1;

while (index1 < size_)  {

    int index2 = 0;

    while (index2 < index1) {

        if (allDeathTolls[index2].size() > allDeathTolls[index1].size()) { // ne

            if (countries[index2].deaths > countries[index1].deaths) { // checki
                allDeathTolls[index1] = allDeathTolls[index2];
            }
        }

        index2++;
    }

    allDeathTolls[index1].push_back(countries[index1]);

    index1++;
}

// finding the index of the longest death spread
```

This is the nested loop that computes the longest spreads and their respective indexes. The outer loop runs up to size_ (that is, n) and for every iteration of this loop, the inner loop runs up to the index of the outer loop. Outer index will be 0, 1, 2....n.

So, complexity :

= [n * (0 + 1 + 2 + 3 + ... + n)]

= [0 + n + 2n + 3n + ... + n * n].

After disregarding constants, this is equal to n^2.

```
// finding the index of the longest death spread

int max = allDeathTolls[0].size();
int index = 0;

for (int i = 1; i < size_; i++) {
    if (max < allDeathTolls[i].size()) {
        max = allDeathTolls[i].size();
        index = i;
    }
}
```

This loop is used to find the subsquence vector with the greatest number of elements. As it requires iterating throughout the whole deathTolls array, its complexity is n.

```
    // passing longest spread to the spread array

    spread = new int[allDeathTolls[index].size()];

    for (int i = 0; i < allDeathTolls[index].size(); i++) {
        spread[spreadSize] = allDeathTolls[index][i].deaths;
        spreadSize++;
    }

    cout << endl;
}
```

This loop iterates through the longest subsequence vector, and passes its values to the spread array. It runs up to the size of the vector, which will be n in the worst case. So, complexity of the loop is n.

```
void displayResults() {

    cout << "Longest  daily death toll decrease period for " << country << ":\n";
    cout << "-----------------------------------------------------------------" << "\n";

//  cout << "Starting date:\t" << sDate << endl;
//  cout << "Ending date:\t" << eDate << endl;
    cout << "Size:\t" << spreadSize << "\n";

    cout << "Values:\t[";

    for (int i = 0; i < spreadSize; i++) {
        cout << spread[i];

        if (i == spreadSize - 1)
            cout << "]\n";
        else
            cout << ", ";
    }
}
```

Displaying the longest spread is done by looping through the spread array. In the worst case, the size of it will be n and the complexity of the loop will then be n.

Time complexity of all these loops resolve to:

= [n^2 + n + n + n]

= [3n n^2]

= [n^2]

= O(n^2)

Time complexity	Space complexity
O(n^2)	O(n^2)

7 QUERY 5

Objective : You have decided to help with the relief effort by collaborating with an organization that distributes essential supplies to affected regions. However, they must consider the overhead cost for each country (distance, shipping etc.). They have assigned a score to each country, which is simply the total number of active cases on the latest day (they prefer to help countries with higher cases). A country may or may not be selected for aid. They have gathered their projected costs for each country and have tasked you with finding the highest possible score attainable as well as the countries selected given a budget of 300. (Efficiently)

7.1 EXPLANATION:

This query first requires a data structure to hold the given data of a country (on one date) i.e. one row of the data provided in the file "WHO-COVID-19.csv". A user-defined datatype, the countryData5 structure, is created for this purpose, with the following attributes (methods not included):

```
struct countryData5 {
	string country;
	int weight;
	int score;
};
```

Extra attributes of a country (used in the previous queries) have been disregarded to save space as there is no use for them. Weight here is the weight assigned to each country, and score is the number of active cases on the latest date recorded yet.

A class named query5 will use this to solve the given query. It stores a limit (that is, the budget), and maintains a vector countries for storing countries.

The algorithm to be applied here is knapsack, specifically the 0/1 knapsack (a country can either be selected or not selected. Additionally, each country can only be selected once.) This will be solved by dynamic programming. Each country has a weight and a score, and a combination of countries has to be selected such that cumulative weight does not exceed the limit and the total score is as large as possible.

To start off, there needs to be a table that maintains all the possible solutions. Every row represents a collection of items from all the rows before it. E.g. row 4 would hold items from rows 0, 1, 2, 3. Whereas each column number represents the weight limit of knapsack. E.g. column 4 considers the total weight as 4. This means that any one cell at row r and column c is the highest score that can be achieved with items from rows 0-r and maximum capacity weight of c. Therefore, if we have n number of items and a limit of 300, then the highest score possible would be computed at table[n][300].

Keeping this in consideration, the table has to have n + 1 rows and limit + 1 number of columns, where n is the number of countries in the country vector. The table is initialized so that every cell is zero.

Now, to iterate through the table using nested loops. For every iteration of the outer loop, a row i is taken and its columns computed, using the values from the previous row, which uses the values from the previous row and so on...so that every row is computed once only.

When computing each cell in that specific row, first its weight is compared with the capacity (j) of that column, so that an entry may be discarded if its inclusion would violate the capacity. If it is not discarded, then two things have to be compared: the score of including that entry and the score of not including it. Whichever is larger is set as the value of that cell in the table.

In this way, every cell is computed until the whole table is set. To display the values, we have to retrace back from the maximum value, which is a at table[n][limit]. If an cell has the same value as the cell above it (same column), this means that it was not selected, so it can be passed over and tracing has to continue from the cell above it. This continues until the whole column is traversed.

7.2 Complexity Analysis:

Considering space complexity, there is a vector for countries, a variable for limit/budget, and matrix used to compute the highest score. If n is the number of countries and m is the budget (which will be > 0), then the matrix takes up space of [n * m]. Factoring in the other variables leads to a complexity of:

[n + 1 + n * m + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1] = [n*m + n + 11] = O(n*m).

Regarding time complexity, there are several loops to be taken into account.

```
// allocating memory
int** table = new int* [size_ + 1];

for (int i = 0; i < size_ + 1; i++)
    table[i] = new int[limit + 1];

// initially whole table is set to zero
for (int i = 0; i <= size_; i++) {
    for (int j = 0; j <= limit; j++) {
        table[i][j] = 0;
    }
}
```

The first loop runs n + 1 times, and the second runs (n + 1) * (m + 1) times.

Adding these two:

= n + 1 + n*m + n + m + 1

= n*m + m + 2n + 2

= n*m

```
// now all possible options will be calculated

int i = 1;

while (i < size_ + 1) {
    int j = 0;
    int prevIndex = i - 1;

    while (j < limit + 1) {
        table[i][j] = table[prevIndex][j];

        int current = table[i][j];
        int prevScore = countries[prevIndex].score;
        int prevWeight = countries[prevIndex].weight;

        if (j >= countries[prevIndex].weight) {
            if (current < table[prevIndex][j - prevWeight] + prevScore)
                table[i][j] = table[prevIndex][j - prevWeight] + prevScore;
        }

        j++;
    }

    i++;
}
```

Nested loops where the outer loop runs n + 1 times, and the inner one runs (m + 1) times.

Complexity:

= (n + 1) * (m + 1)

= n*m + n + m + 1

= n*m + m + n + 1

= n*m

```
    cout << "\nHighest possible score for a budget of " << this->limit << " = " <<

    cout << "\nCountries selected:\n\n";

    cout << "Weight" << "\t|\t" << "Score" << "\t|\t" << "Country" << "\n";
    cout << "-----------------------------------------------------\n";

    for (int i = size_ - 1; i > 0; i--) {
        if (table[i + 1][limit] != table[i][limit]) {
            this->display(i);
            limit -= countries[i].weight;
        }
    }
}
```

This loop runs n – 1 times to display the selected countries. As the statements inside it are of constant complexity, their cost is disregarded and the overall complexity of the loop is (n-1)

Time complexity of all these loops resolve to:

= [(n * m) + (n * m) + (n – 1)] = [2(n * m) = n – 1] = O(n * m)

Time complexity	Space complexity
O(n * m)	O(n * m)

8 OUTPUTS OF ALL QUERIES

8.1 QUERY 1:

```
Running query 1:
----------------
Enter the target date (DD-MM-YY): 10-03-20

Top 20 countries with the most confirmed cases on 10-03-20:
```

	Day	Country	Region	Deaths	cumDeaths	Confirmed	cumConfirmed	Country Name
1:	10-03-20	CN	WPRO	17	3140	20	80924	China
2:	10-03-20	IT	EURO	168	631	977	10149	Italy
3:	10-03-20	IR	EMRO	54	291	881	8042	Iran (Islamic Republic of)
4:	10-03-20	KR	WPRO	3	54	131	7513	Republic of Korea
5:	10-03-20	FR	EURO	3	33	372	1774	France
6:	10-03-20	ES	EURO	8	36	615	1639	Spain
7:	10-03-20	DE	EURO	0	2	157	1296	Germany
8:	10-03-20	US	AMRO	6	25	224	696	United States of America
9:	10-03-20	JP	WPRO	0	9	0	514	Japan
10:	10-03-20	CH	EURO	1	3	159	491	Switzerland
11:	10-03-20	NL	EURO	1	4	117	382	Netherlands
12:	10-03-20	GB	EURO	3	6	50	373	The United Kingdom
13:	10-03-20	SE	EURO	0	0	78	326	Sweden
14:	10-03-20	NO	EURO	0	0	85	277	Norway
15:	10-03-20	BE	EURO	0	0	67	267	Belgium
16:	10-03-20	DK	EURO	0	0	172	262	Denmark
17:	10-03-20	AT	EURO	0	0	70	182	Austria
18:	10-03-20	SG	WPRO	0	0	6	166	Singapore
19:	10-03-20	MY	WPRO	0	0	0	117	Malaysia
20:	10-03-20	BH	EMRO	0	0	15	110	Bahrain

8.2 QUERY 2:

```
Running query 2:
----------------
Enter the first date (DD-MM-YY): 01-03-20
Enter the second date (DD-MM-YY): 01-04-20

Maximum new cases between '01-03-20' and 01-04-20': 67309

--------------------------

Country: DE
Country Name: Germany
Region: EURO
Cases (01-03-20): 129
Cases (01-04-20): 67366
```

8.3 QUERY 3:

```
Running query 3:
---------------
Enter the name of a country: China

Longest spread period for china:
----------------------------------------
Starting date:  12-01-20
Ending date:    13-02-20
Size:   20
Values: [0, 4, 17, 59, 77, 93, 152, 261, 462, 688, 776, 1462, 1741, 1984, 2101, 2590, 2827, 3233, 3892, 15152]
```

8.4 QUERY 4:

```
Running query 4:
----------------
Enter the name of a country: Italy

Longest  daily death toll decrease period for italy:
------------------------------------------------------------
Size:   9
Values: [971, 887, 758, 727, 681, 636, 604, 540, 431]
```

8.5 QUERY 5:

```
Running query 5:
---------------

Highest possible score for a budget of 300 = 1281006

Countries selected:

Weight  |       Score   |       Country
-------------------------------------------------
50      |       578199  |       United States of America
15      |       67874   |       Turkey
20      |       85612   |       The United Kingdom
70      |       159054  |       Spain
5       |       6297    |       Saudi Arabia
5       |       10073   |       Peru
25      |       143508  |       Italy
15      |       12074   |       Israel
55      |       126881  |       Germany
30      |       88009   |       France
10      |       3425    |       Dominican Republic
```