Programming Assignment 1

Choice of data Structures

1. The town Data Structure

Towns are of type struct:

Struct allow us to have a collection of members/ elements, with different types and different lengths

Access to struct elements is almost as fast as access to individual variables

A struct is also easy to modify

Structure also suitable for implementing binary trees

2. The collection of Towns

All towns are stored in an unordered_map stl data structure:

- Maps are convenient for storing data due to their ability to access data in a fast way
- Maps also avoid data duplicates (By key) therefore you don't have to check every time you are adding data.
- Unordered maps have even more advantages due to their improved asymptotic performance as compared. to maps. This makes unordered maps faster than maps for almost all operations

For all the basic operations on an unordered_map needed in this implementation, the asymptotic performance is almost constant meaning O(1)

Here are examples below

- Insert element - O(1)
- Access element by Key - O(1)

3. Temporary data storage

Many functions in this implementation request data that should be returned as a vector, or as a result of an operation on a vector, say, minimum values, maximum values, sorted data and other operations that fall in this category.

This requires data to be fetched from the main data structure (unordered_map) into a temporary storage where it arranged and converted into a form that will enable the intended processing operation

This requires a data structure that allows duplicates, since our data may contain duplicates, but also has reasonable performance to avoid slowing us down

A vector - std::vector<>

For all such operations a vector is used. For data for which we need to keep track of its unique identifier (key) we use a pair to store a pair(std::pair<>) and implement a Vector of pairs

A piar - std::pair<type1, type2>

A pair provides a simple way to store two heterogeneous objects as a single unit.

Pair can be assigned, copied, and compared

Easy access to pair elements using the dot(.) operator is equivalent to accessing normal variables

Vectors have advantages over other data structures that are capable of implementing this functionality (say lists or arrays). These include

- Simple storage and inhouse memory management
- Ability and simplicity of adding elements
- Accessing and searching for elements is reasonably quick

Majority of the operations on vectors implemented in this exercise have constant asymptotic performance O(1) with a few having linear performance O(n). Here are examples below.

- Insert Tail - O(1)
- Remove Tail - O(1)
- Search by Index - O(1)
- Search by key - O(n)

Vectors are also reasonably efficient when working with stl algorithms such as sort(), reverse() and finding minimums and maximum values

The last reason for using vectors is because some function calls already return a vector data structure. Therefore, if these functions are called within another function, a vector must be used for that case