**Time complexity of map data structure:**

The total time complexity will be O(N \* time taken by map data structure).

Storing(i.e. insertion) and fetching(i.e. retrieval) in a C++ map, both take always O(logN) time complexity, where N = the size of the map.

But the unordered\_map in C++ and HashMap in Java, both take O(1) time complexity to perform storing(i.e. insertion) and fetching(i.e. retrieval). Now, it is valid for the best case and the average case.

But in the worst case, this time complexity will be O(N) for unordered\_map. Now, the worst case occurs very very rarely. It almost never happens and most of the time, we will be using unordered\_map.

**Note:** *Our first priority will be always to use unordered\_map and then map.* *If unordered\_map gives a time limit exceeded error(TLE), we will then use the map.*

The time complexity in the worst case is O(N) because of the internal collision.

In order to understand collision properly, we need to understand the concept of how the hashing work with an optimized space.

**What is collision & How the hashing works:**

Hashing is done using several methods. Among them, the three most common ones are

* **Division method**
* **Folding method**
* **Mid-Square method**

Here, we are only interested to discuss the division method. The rest two methods may be important for college exams but not much important in terms of interviews or coding rounds.

The map data structures in the C++ STL or in Java are implemented using different and complex methods. We need not know about them. But here we are going to discuss the **division method** so that we can understand the collision properly.

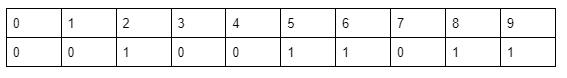
**Division Method:**

Let’s discuss it considering the following example:

Assume, we are given an array: **[2, 5, 16, 28, 139]**. Here, we can apply array hashing, and to use that, we need to create an array of size 140. Now, what to do if we are given a constraint that **we cannot use an array whose length is greater than 10**?

In order to solve this we will use the division method. We will simply consider the modulo 10 of each element of the array(element % 10) and we will hash(pre-store and fetch) the elements on the basis of the modulo value i.e. the remainder. The steps will look like the following:  
Pre storing: hash[arr[i]%10] += 1 and Fetching: hash[number%10]

Now if we apply this method to the given array, the hash array will look like the following:



This is how the division method works. We simply reduce the array elements and apply array hashing.

Let’s discuss the questions that come up in the mind:

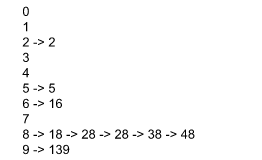
**Question: What if two or more array elements give the same remainder for modulo 10?**

In this case, we apply the **linear chaining** method. This method is implemented using Linked List which will be discussed later in another article. Here, we just need to understand the logic. While storing the elements we will maintain a chain(i.e. inserting the element itself to the corresponding index instead of just keeping the count) for each index(i.e. the remainder we get). And in that chains, we will store the elements in a sorted fashion.

Let’s understand it considering the following example:

Given array: **[2, 5, 16, 28, 139, 38, 48, 28, 18]**

In this array 28, 38, 48, and 18 are giving the same value for modulo 10. So, we will apply linear chaining. The hash array will look like the following:



Now to get the frequency of a number, we will first go to (number % 10) indexed chain and count the frequency of that number.

**Note:** *We can choose to take modulo of any number as per our need. Here for example we have taken the number 10.*

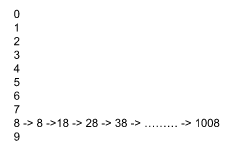
**Collision:**

Now, if we are applying linear chaining and division rule and we find that all elements of an array get stored in a single index, then we will call it a case of collision.

**Example:**

Given array: [8, 18, 28, 38, 48, 58, 68, ….., 1008]

If we apply the methods and take modulo 10 for every number, the hash array will look like the following:



Now, while fetching we have to traverse N times(N = size of the given array) to find the frequency of an element. This is when the worst case happens and the time complexity becomes O(N). But this happens very very rarely.

***Whatever method the map is using, if all the elements go to the same hash index, we will call it a case of collision.***

**Note:***In the map data structure, the data type of key can be anything like int, double, pair<int, int>, etc. But for unordered\_map the data type is limited to integer, double, string, etc. We cannot have an unordered\_map whose key is pair<int, int>.*