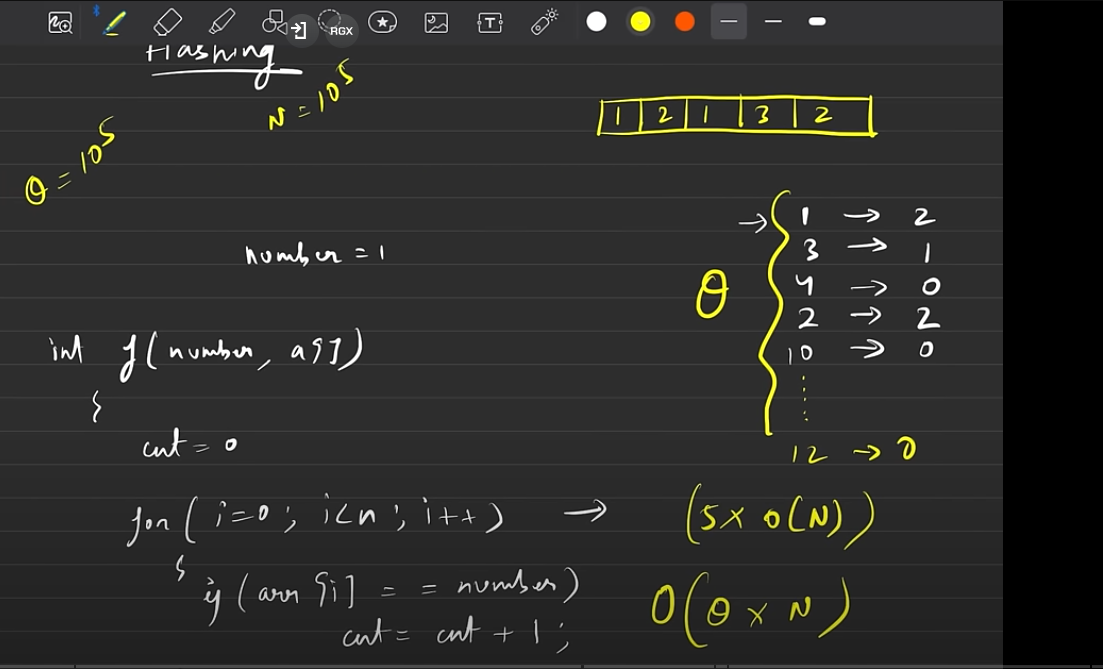
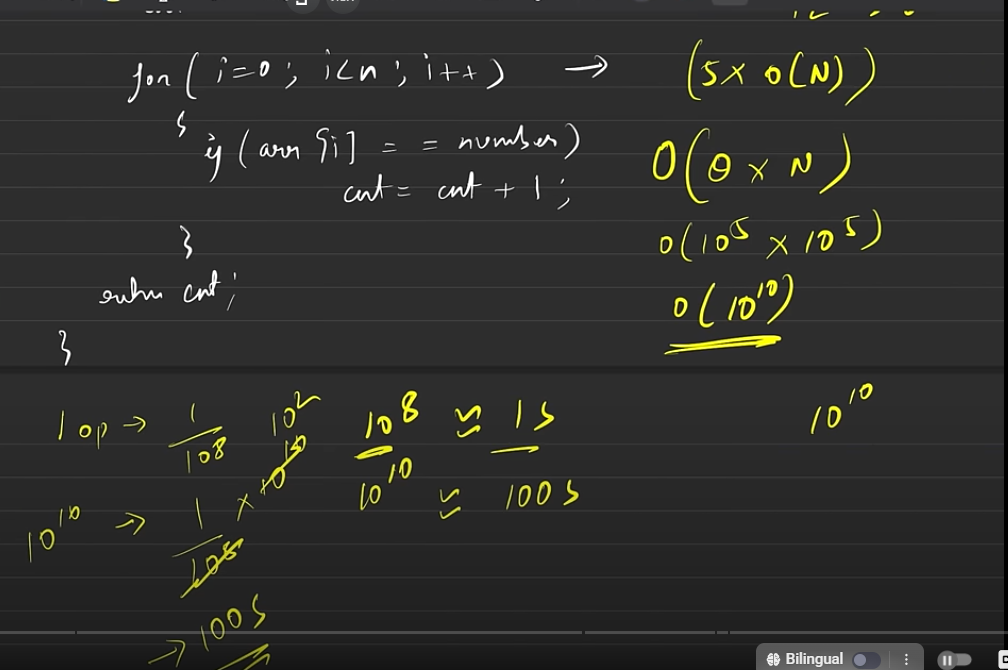


**Refer The Article For Sure (As its imp for revision) and follow the Digital notes along with it:** https://takeuforward.org/hashing/hashing-maps-time-complexity-collisions-division-rule-of-hashing-strivers-a2z-dsa-course/

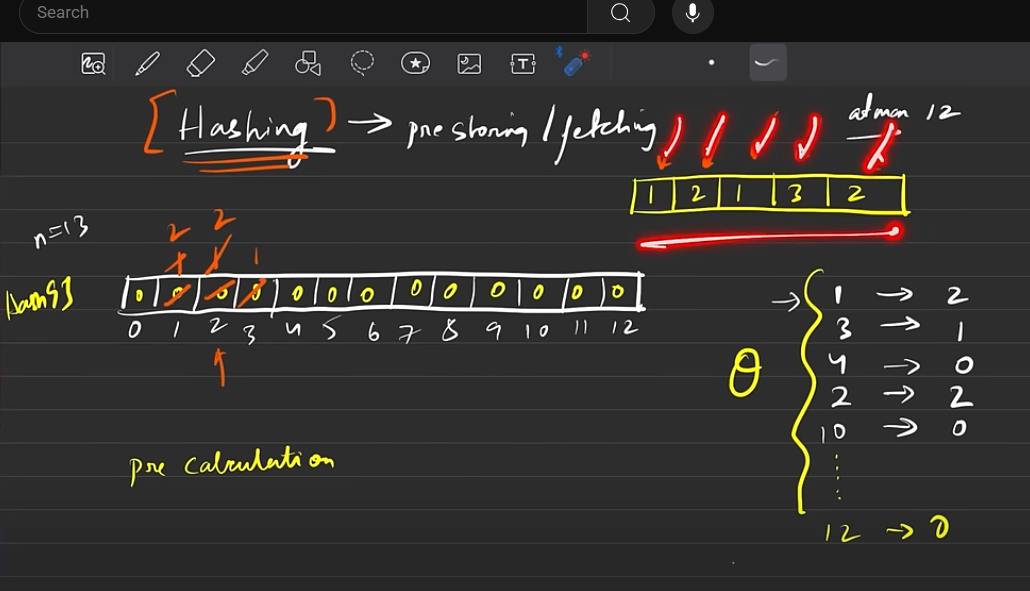
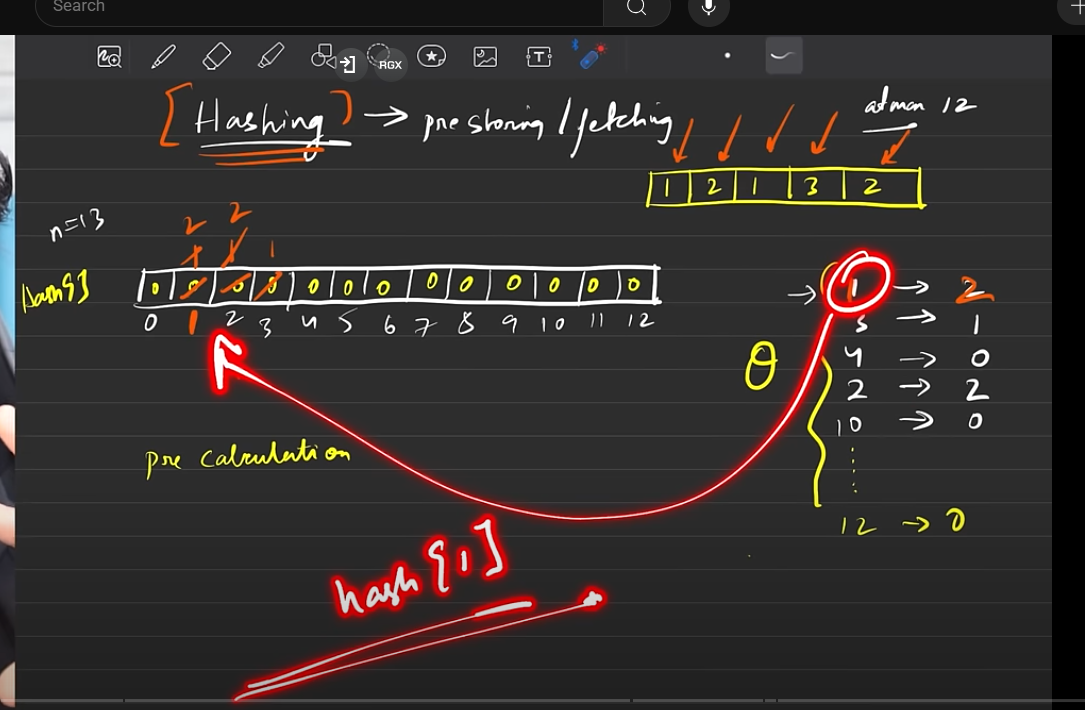




**VERY**

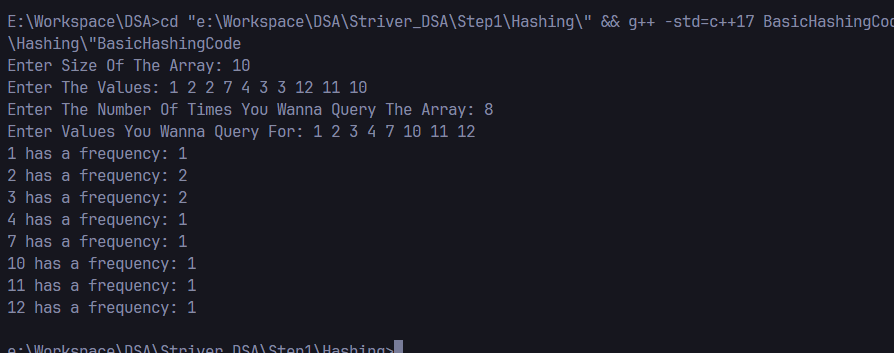
**IMPORTANT**

CALCULATION FOR FINDING OUT THE TIME TAKE TAKEN BY THE ONLINE COMPILER USING THE TIME COMPLEXITY

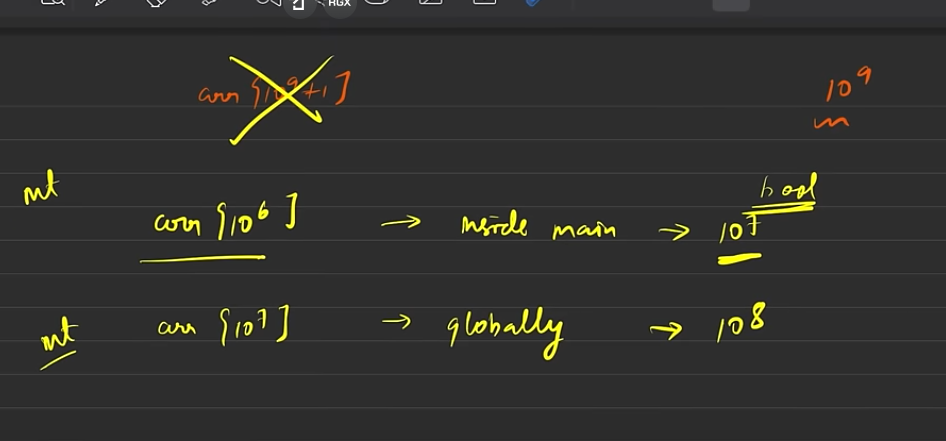


Thus, here a single pre-computation solved my queries in one go. (The pre-computation was to make a separate hash array before so I don’t have to loop every time.

My Code’s Output

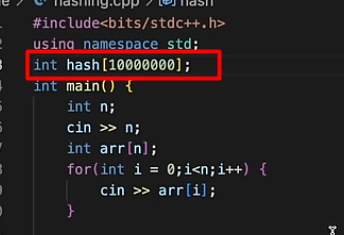


Now we if have an array of 10^9 then we can’t declare it since if we have predefined limits on how much memory can be allocated to an array and we move beyond that then we will get segmentation fault (as it won’t be able to allocated that much memory). Now for int and Boolean the limit is different so you can view them below. (Btw remember that what is given below is not the exact limit rather its around it)





We can also write like this, we had to type cast t to int since 1e5 is actually a double value i.e. 100000.0, thus I had to type-cast it in for arrays

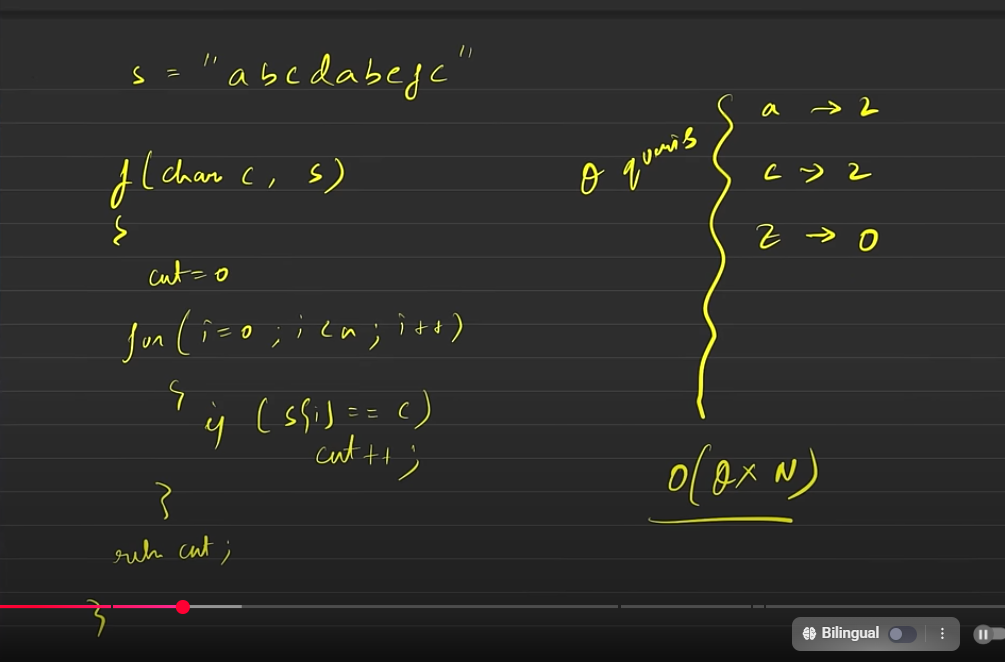


Anything that is declared globally is always initialized to 0 and not as garbage values

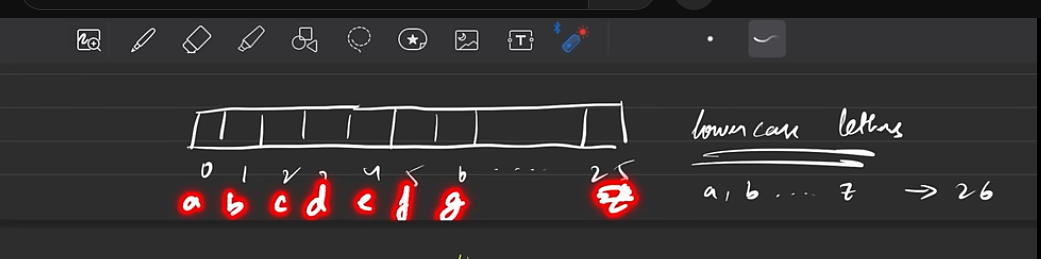
(Valuable Info)

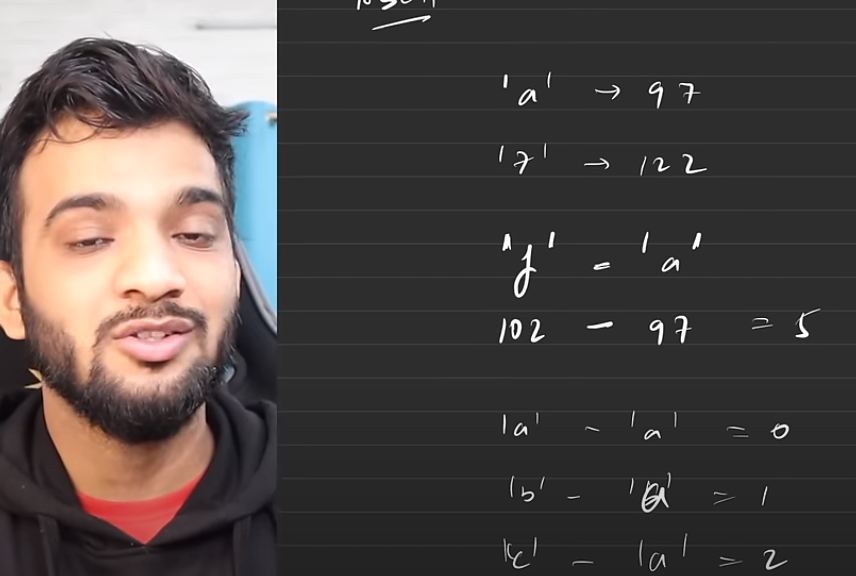
(here btw, the hash name might clash since its declared globally and we already have a predefined has so you should use another name)

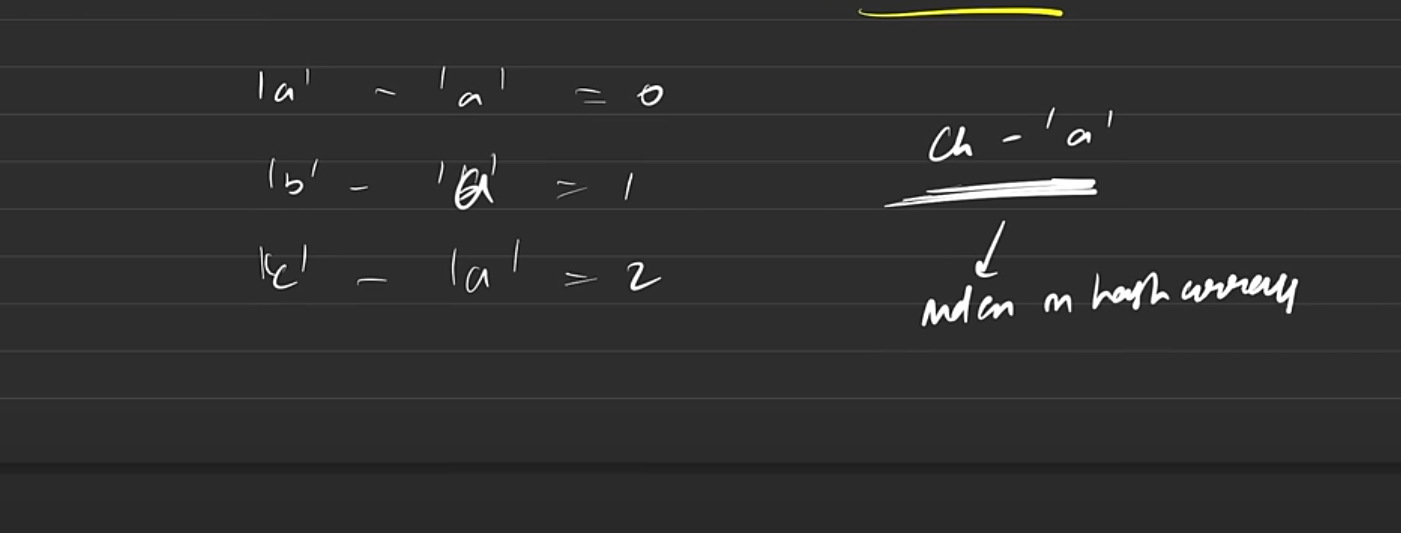
Character Hashing (Just as numbers you might what to ask the frequency for characters (below is the brute force approach to it



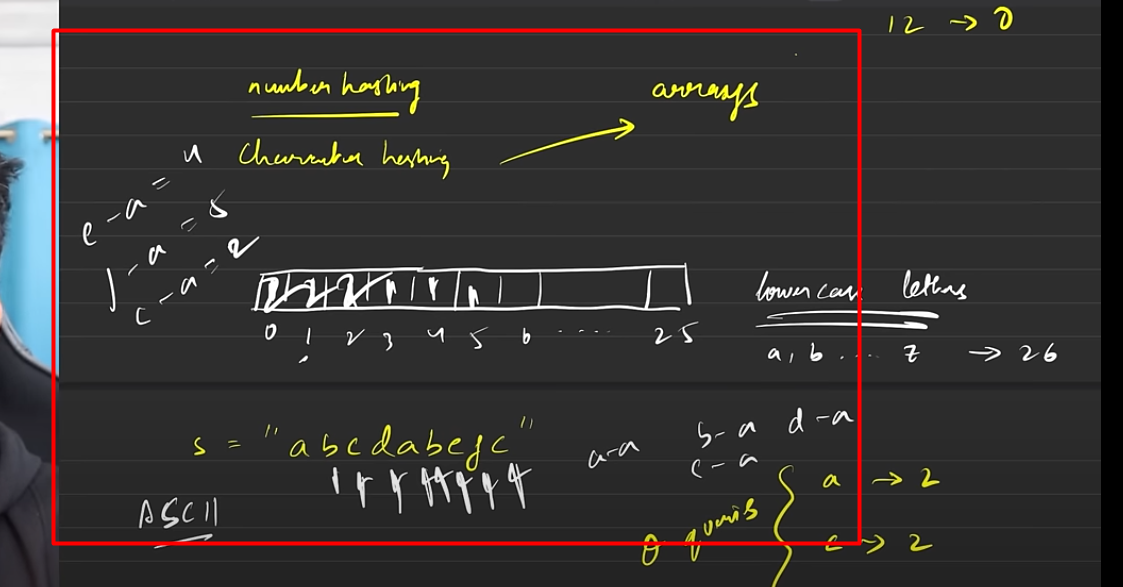
Now, you may ask that just as we did with numbers can we also do it for character i.e. that can we hash them using arrays? And the answer to it is YES. And we’ll be doing this ahead now

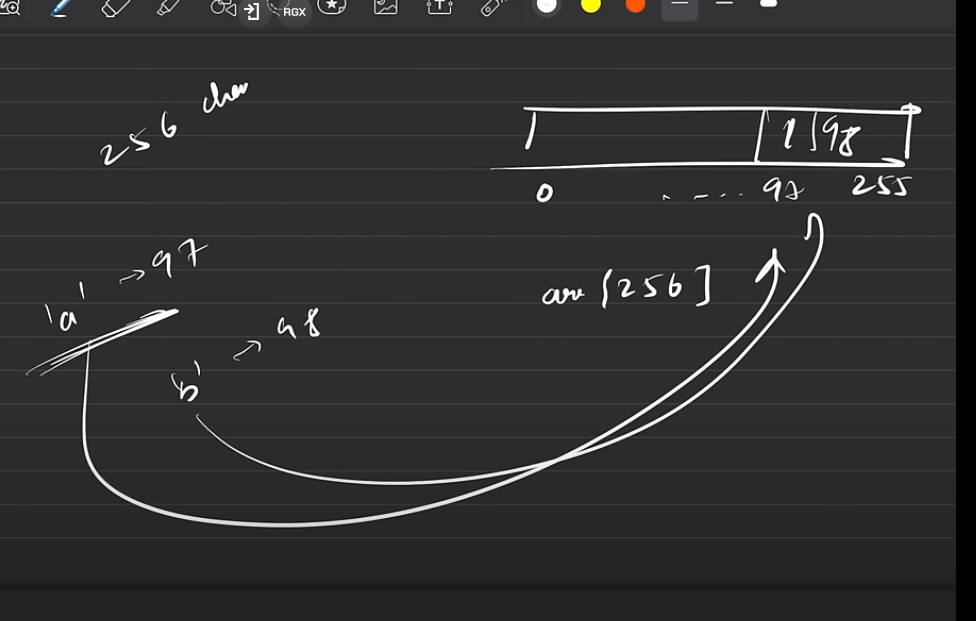




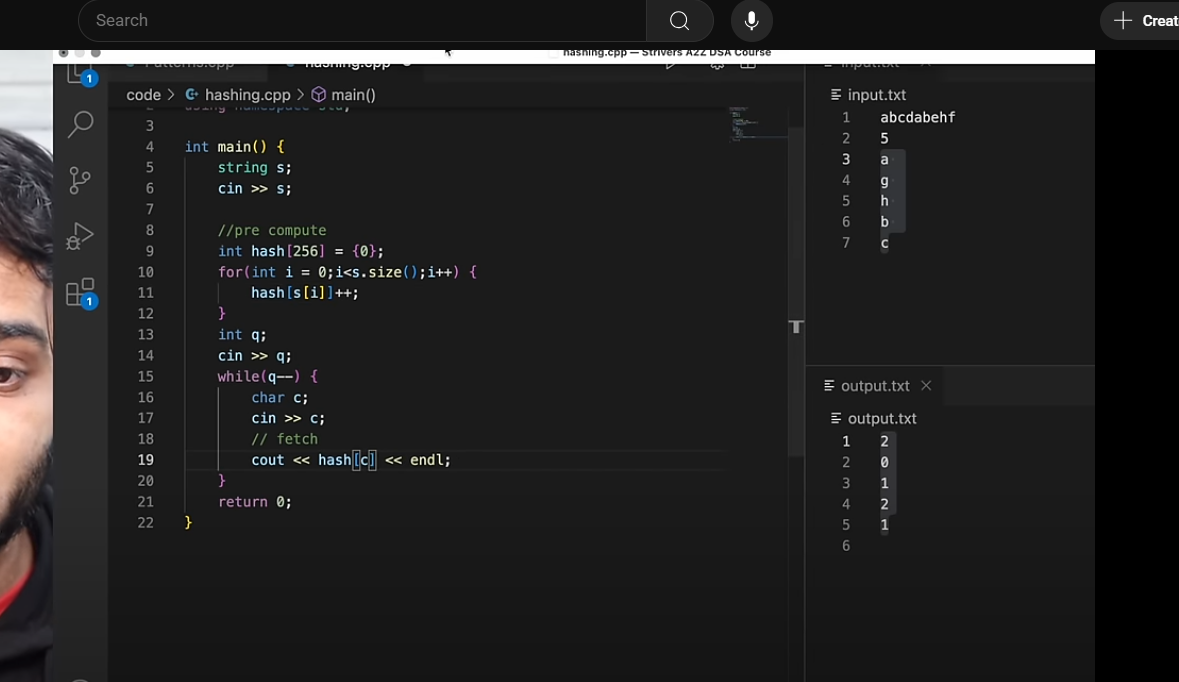
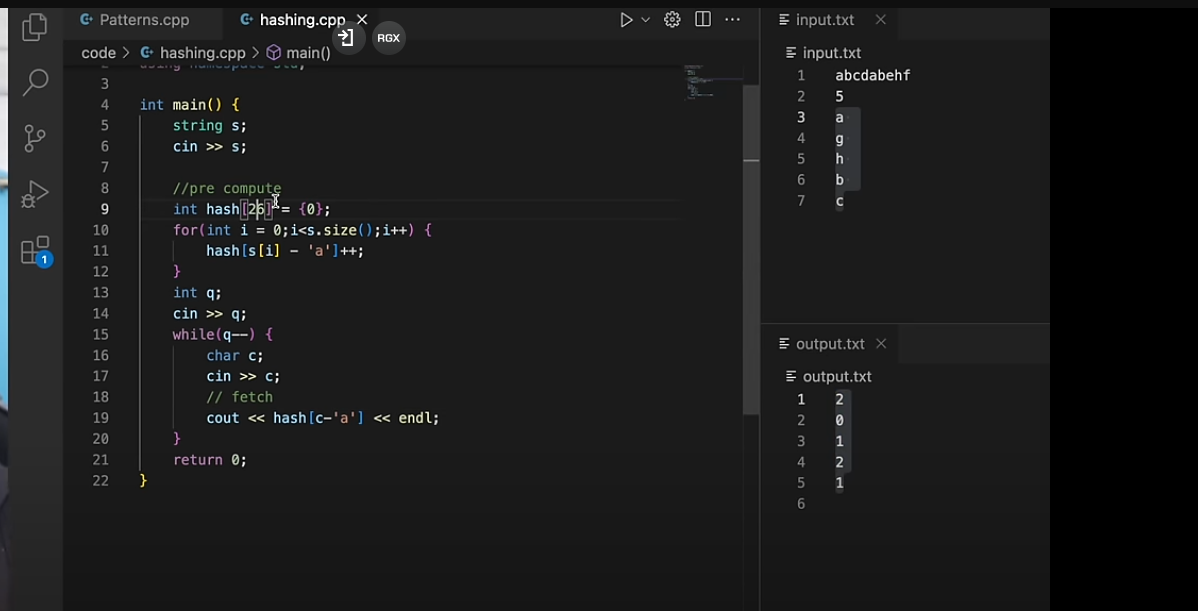


So, you can see that by doing this you can get the corresponding index in the hash array



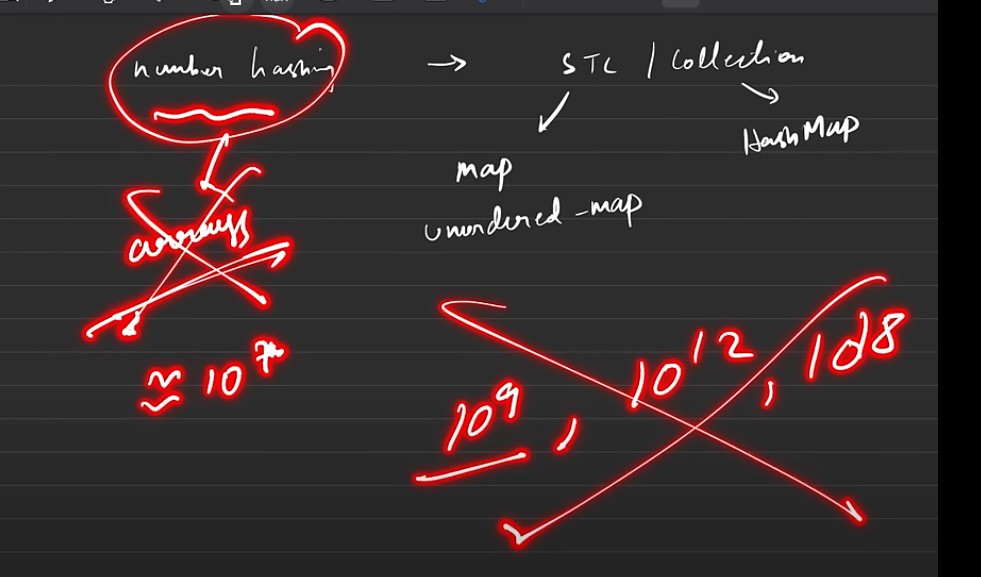


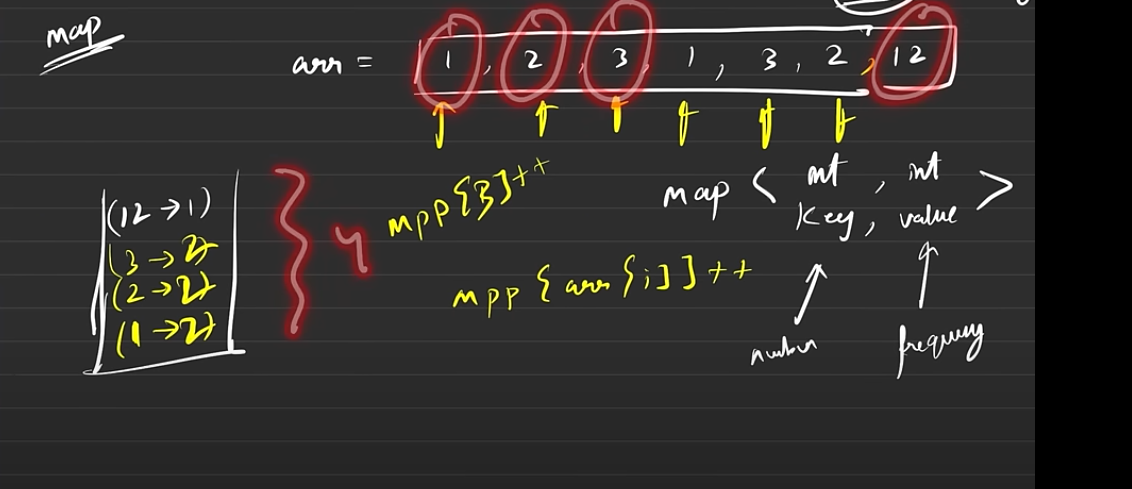
Now, if you want to have uppercase letters then you can just take an array of size 256 accounting for all the 256 characters and this time, you don’t time you don’t even need to do any formula based subtraction achieve the formula, as you can simply iterate over as all the characters are already assigned to a particular value in his 0-255 range like lowercase letters start from 97 already so you can directly match this up now. Earlier we only had to deal with lowercase 26 letters that’s why we needed to make a formula since the size was small and we needed to make it compatible



Now, coming back to the problem that we had in number hashing i.e. at max. we can only have an array of size of 10^7(approximately) (That too when its declared globally as inside main() the size limit is even less. Thus values like 10^9, 10^12, 10^18 are way beyond the limit and its not possible for arrays to store such values thus we cannot hash them through arrays.

And, thus to solve it we have STL(from C++) / Collection (from Java) . Now, from STL we have map and unordered map which can be used for hashing, and then it Java we have Hash Map which has the same concept and it’s the just the index which is different.

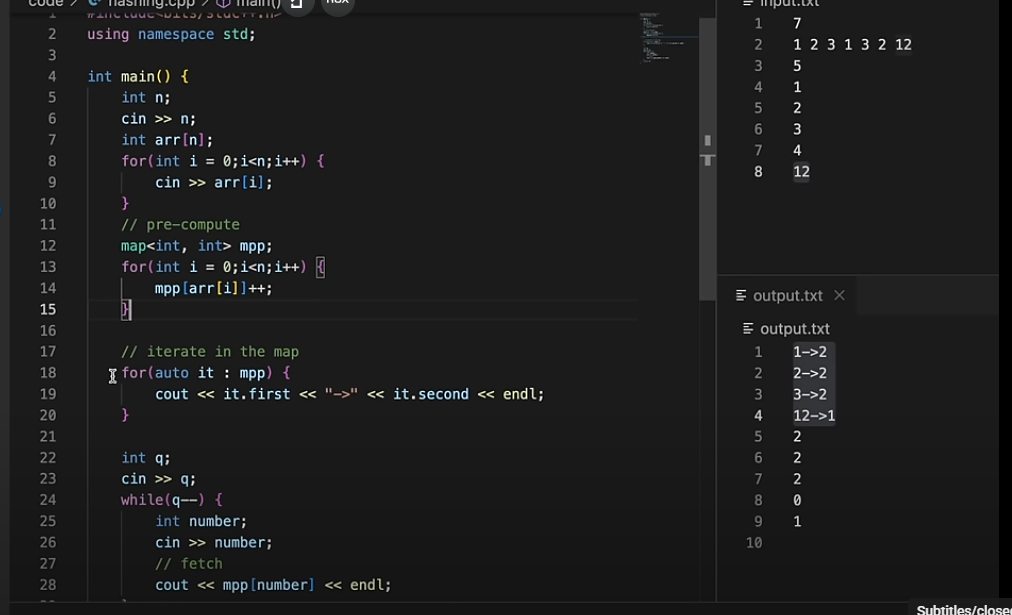
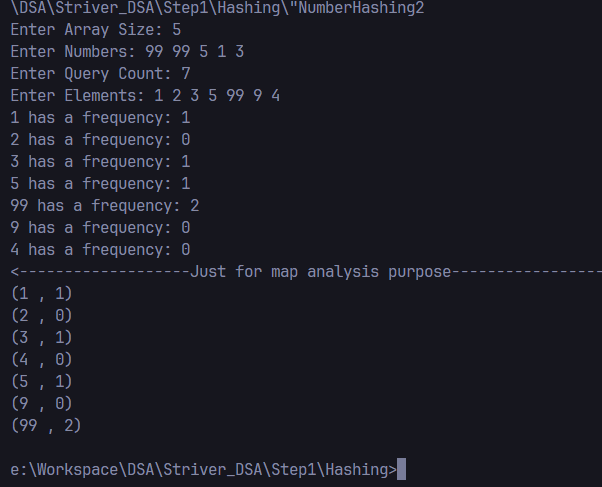




Advantages of a Map Data Structure:

* It only stores the required elements in the map as you can it only stored the elements for which the query was asked for. This makes it slightly beneficial than arrays as it takes a little less memory(since arrays have to make space for all the elements until the maximum to ensure its index exists.)
* For all the elements which are not present (i.e. key is not found) it automatically returns there value as 0 by default.
* So, basically in a map the number is stored as the key and its corresponding frequency is stored as its value associated with it. Thus, using the number you can directly access the frequency just like indexing.
* Besides, we don’t have to initialize it with anything as mentioned in the second point, it takes the default value as 0 but if it finds it in the main array so we can directly increment that value.

Ex: Here, as you can see the max. element is 12 so in array hashing you would’ve ended up taking the size as 13 for array. But, in map you are just required to store 4 elements here(because that is what is required and rest are just repeated iterations of them).  
Suppose you do mpp[4] then it automatically throws you 0 but still it doesn’t stores 4 in the map (until you query for it or you increase it)( Look at the output below).



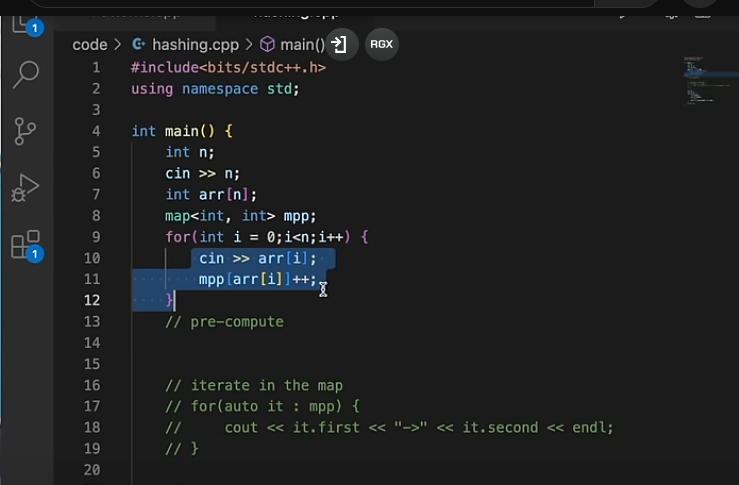
My Code’s Results. (Refer VS Code for code (file: NumberHashing2)

Striver’s Code And Results

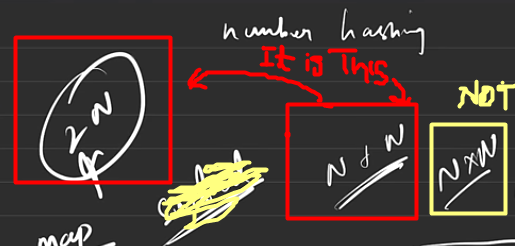
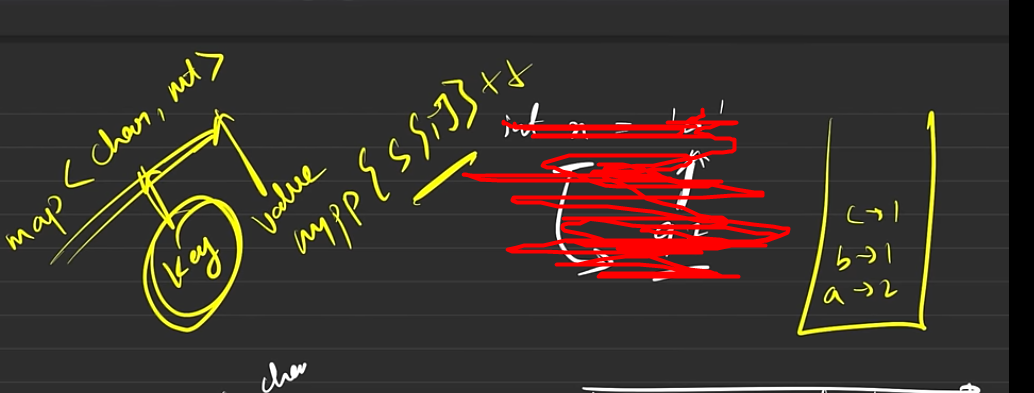
From, the given output, you can also conclude that map stores everything in the sorted order (of keys)

(This is an important point which is proved here in both images as you can see the iteration is Striver’s output and the ‘Just for map analysis purpose’ section my code which actually shows the key-value pairs that exist in map).

Some more insights



We can also do pre-computation while taking the input too. Although it will save us 1 for loop but it doesn’t change the anything much as the TC still remains the same as it is N+N and not NXN. As, NXN generates a bigger value and in 2N we ignore the constant i.e it same as O(N) only.



Now, we can use map for character hashing too.

Where it keys become the character and the frequency becomes the value. (But, remember, here the character is not stored as ascii value but rather as char itself.

Key Insight:

* So depending whatever you’re trying to hash that will become your key
* And what you’re storing that will become your value. (Since, over here we’re storing the frequency that’s why we did ++).

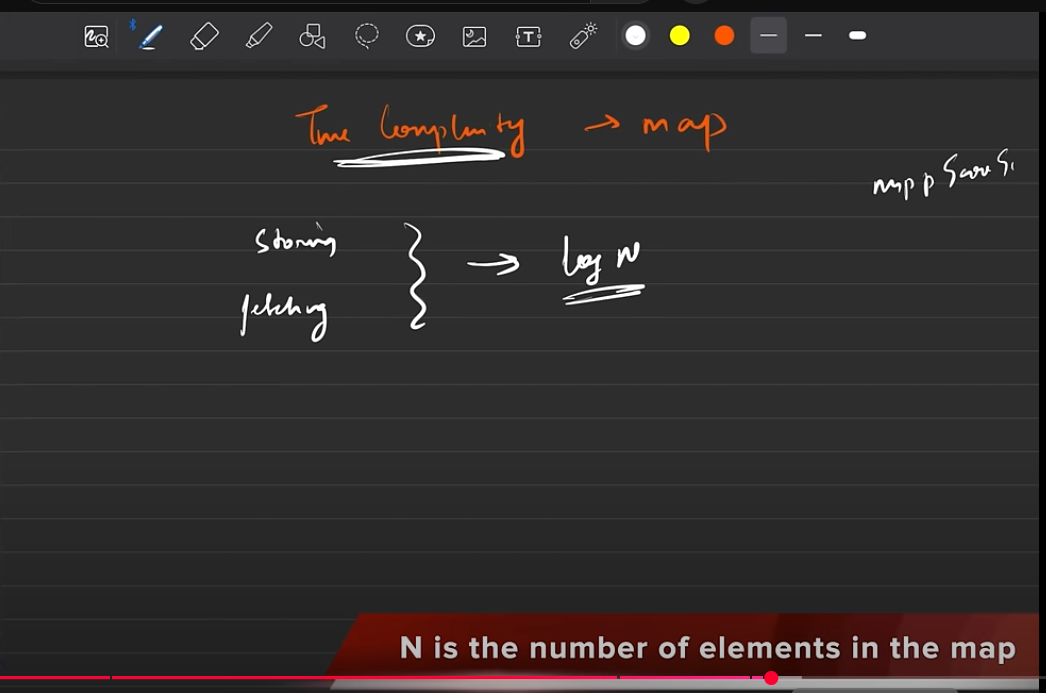
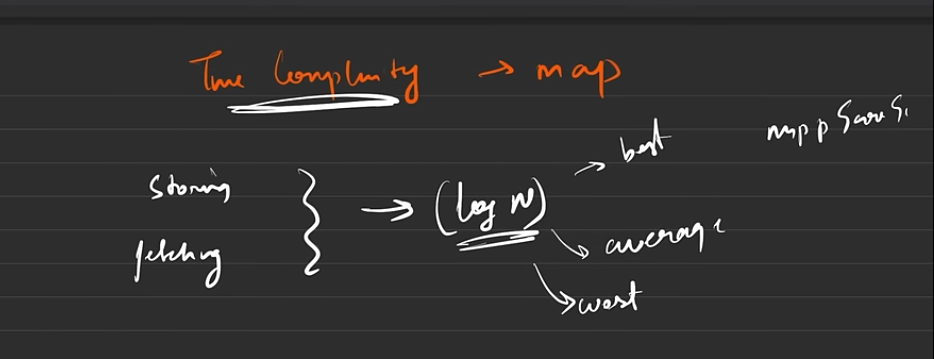
In Java, we have Hashmap which works in the similar fashion.

Storing: Like when we do ++ for storing the frequency

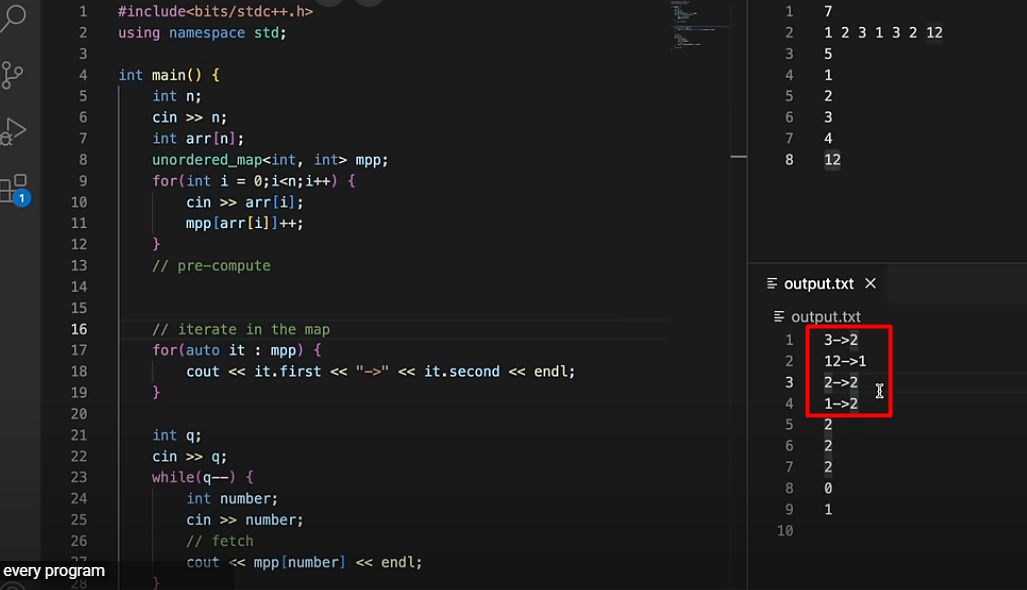
Fetching: Like when we do mpp[arr[i]]

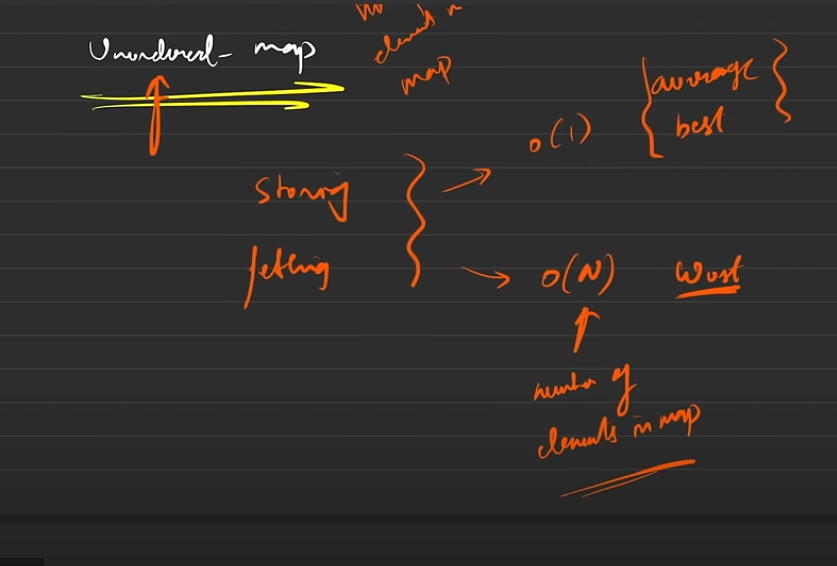
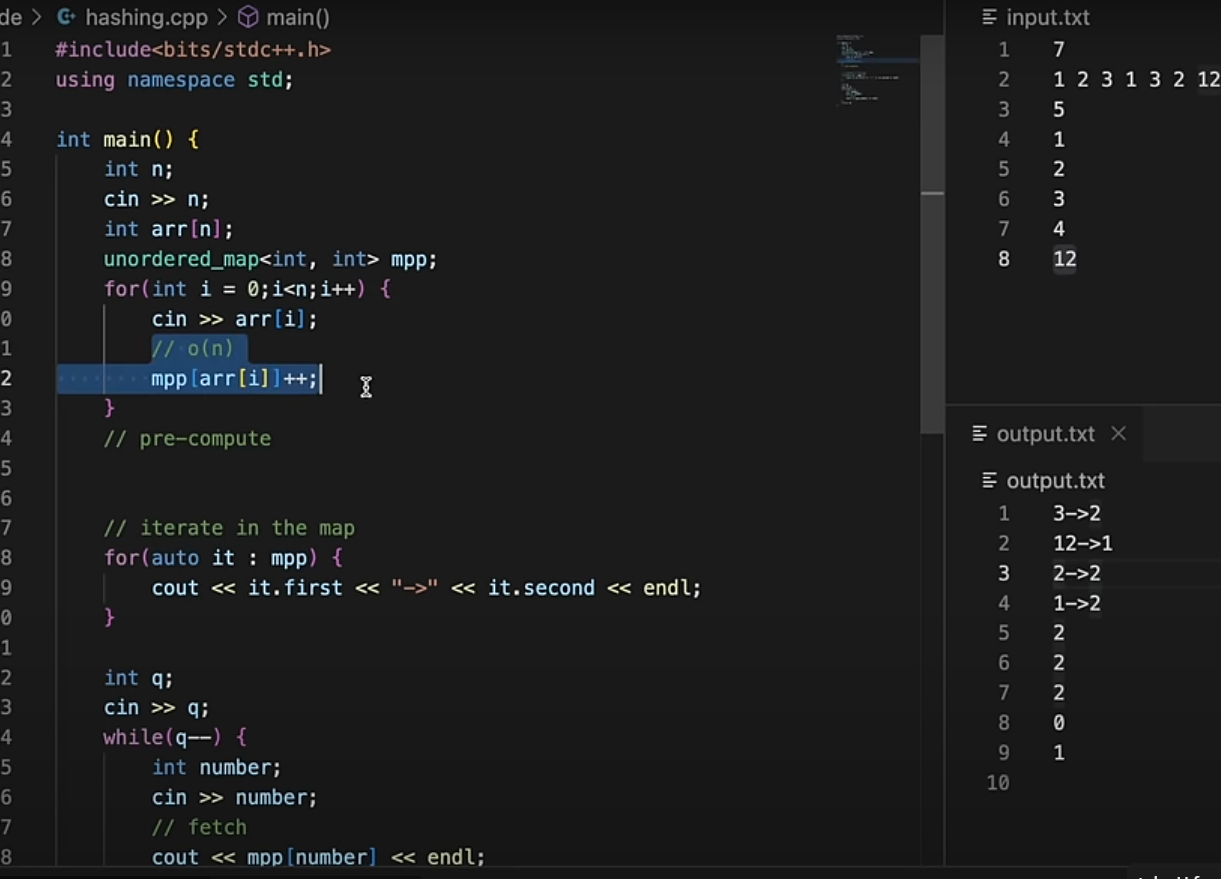
(we’re actually fetching a value from the map)

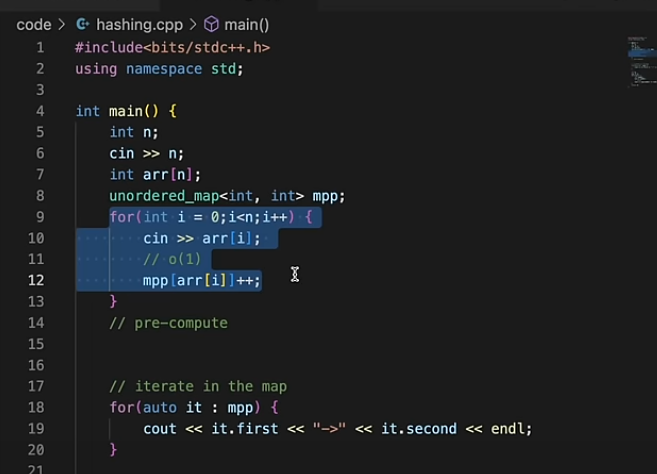
So, both of these operations take O(logN) TC in all cases



Using Unordered Map (So as the name recommends its order cannot be decided







Most Cases

Worst Case

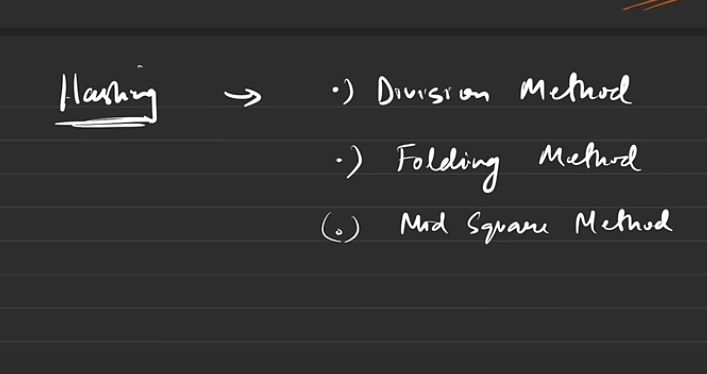
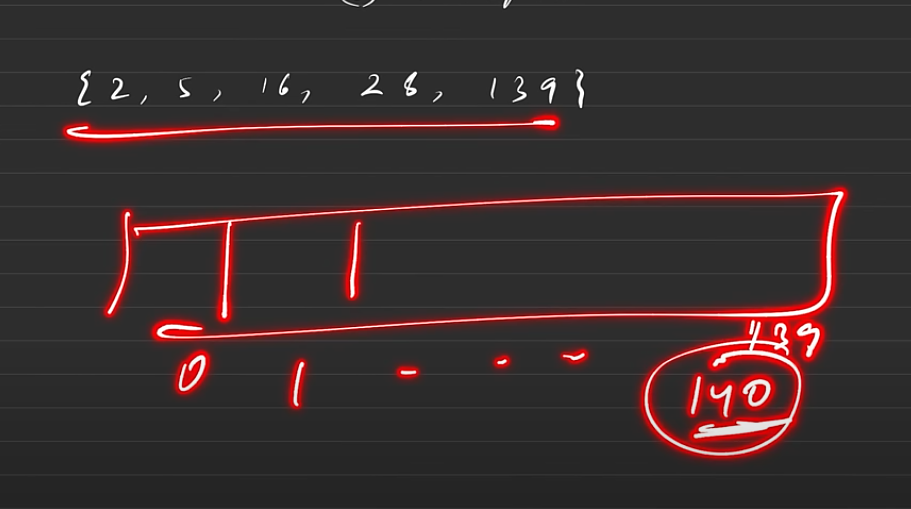
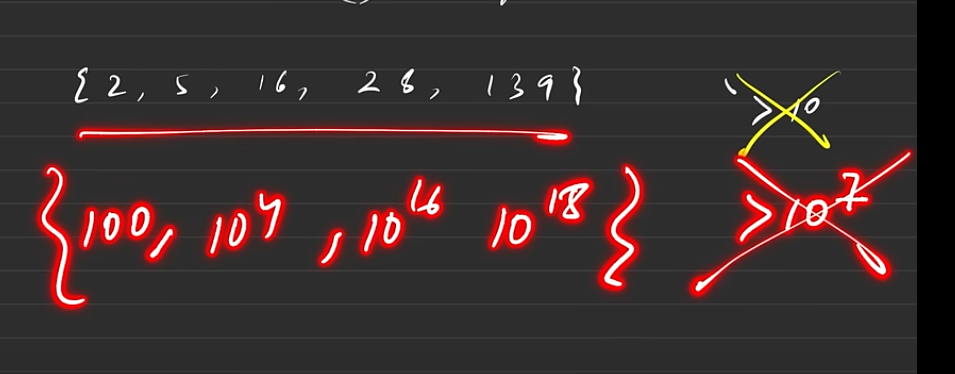
By code failing we mean, suppose while using un-ordered map if the TC of problem is fixed. So, if the worst case happens then this code will take O(n\*n)[worst case] instead of O(n\*1) [in most cases] which would make the code go in TLE(Time Limit Exceeded).

Thus, its recommended to always do questions firstly using Unordered\_map and if that fails(i.e. it gives you TLE) then only you should switch back to map.

Thus, unordered-map should be your first preference as the worst case happens very very rarely.

Now, the advantage of having an unordered-map is that is most of cases it only take O(1) TC for both storing and fetching whereas in the Worst Case (that happens very rarely i.e. once in blue moon (happens when the problem setup is made in a way to make our code fail) then it would take about O(n) TC.

Now, if you’ll just replace the data structure to unordered map then you’ll realize that it works the same. Instead of storing the key-value pairs in a sorted fashion (like the former one) it stores it in a random fashion which is dependent on the compiler (Although you might say that on your system it gives the same result but it can give different results on some other system because it would have a different compiler.



But, suppose we have restriction that we cannot exceed the size of array beyond 10. (Now, although there are much smaller values here, we can map it to bigger values as mentioned below in the red, like there the size cannot extend beyond 10^7 and max element is 10^18.

Division Method

Ideal Way To do array hashing of the given elements

So, now how is an internal map designed?

Now, map data structures can be implemented in different ways by any of the three methods given on the left side.

So, they might be using any of them and they are implemented in very complex way so you don’t need to know about them.

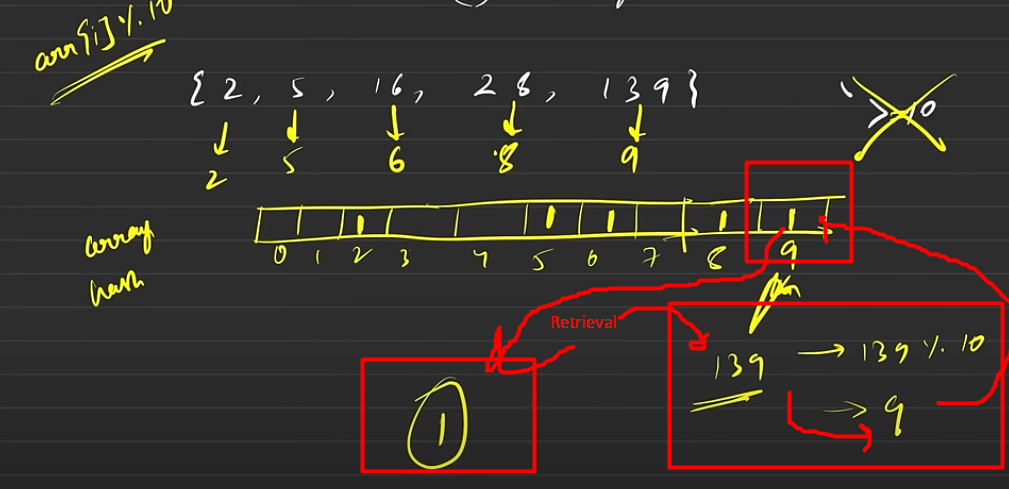
However, knowing about Division method will help you get idea about collisions

Now, if you remember, the moment it crosses 10^7 an array hashing cannot be done. So, how does map DS achieve this, how is a map DS created?(Now, although in interviews they don’t ask that much and even if they do telling them, about division method works fine for the rest two methods you just need to know name (not required) but you must understand the Division method)

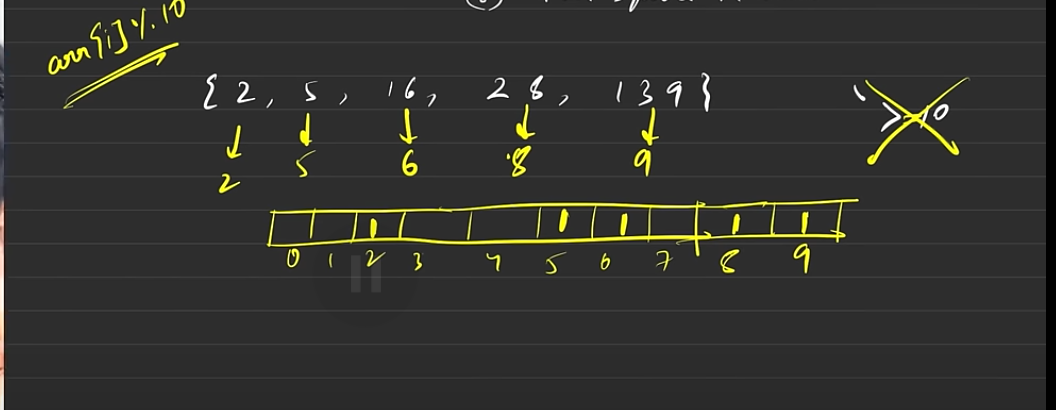
**Why does the Worst Case happens?**

It happens because of Internal **Collisions**.

Now, since we are taking about collisions let’s explore them now but before that let’s see how hashing works



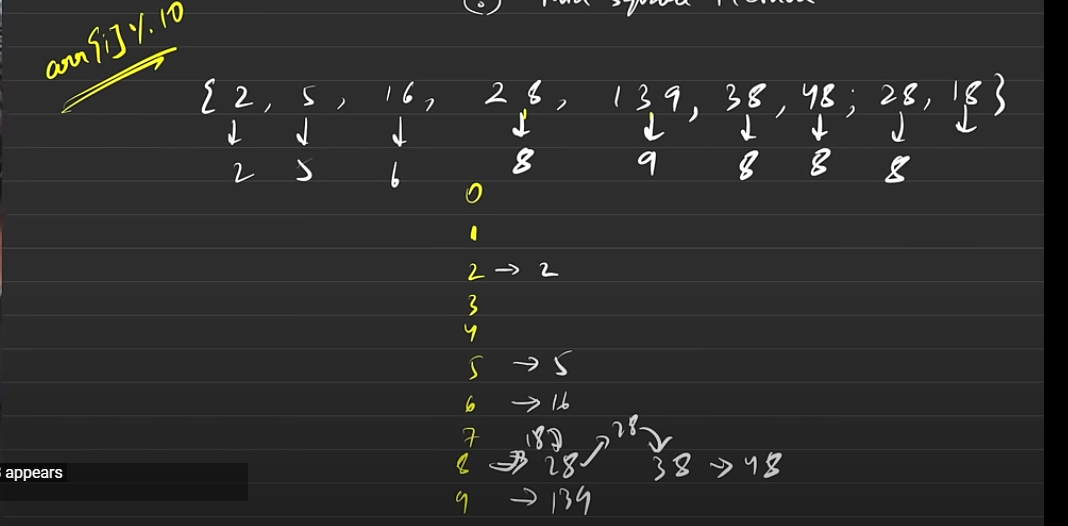
Now, we will apply Division rule to deal with this situation



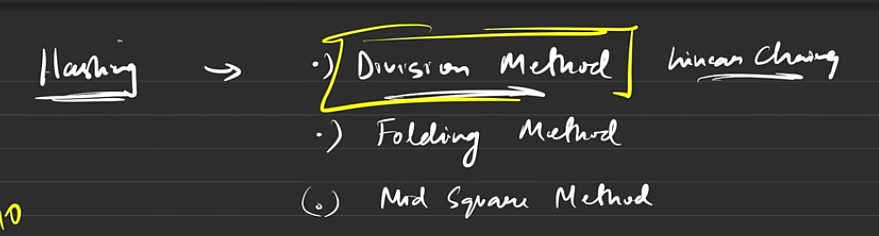
So, in the above images we were able to hash the values using division method.

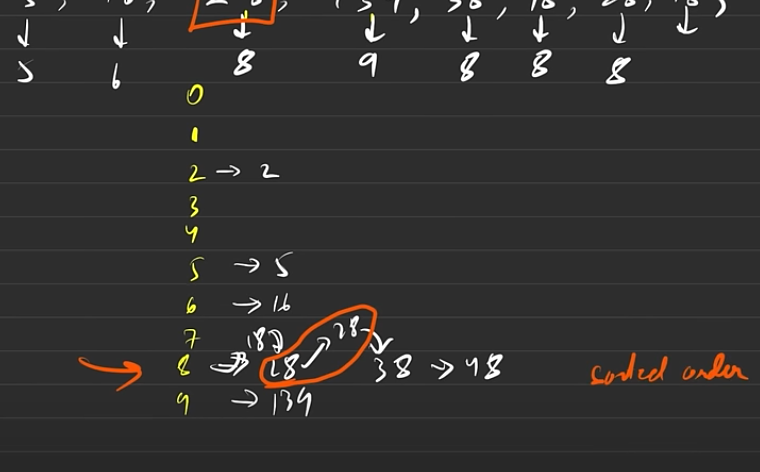
In the latter image we also tried to retrieve how many times do 139 appears, for which we do 139%10 and then use the resultant value(i.e. 9) as the index to find its frequency from the hash array(where it was already stored)

Now, this is a very subtle and simple example (Just for the Idea but there are many complications to it).



Here, you can see that approach is slightly changed here, now we are not focusing on the frequency of the numbers as of now, we are focusing if they are colliding or not. Now if the hash index value comes to be same for more than one number then these numbers are chained together in a sorted fashion which is achieved through Linked List. Now, this is what we call as Linear Chaining.

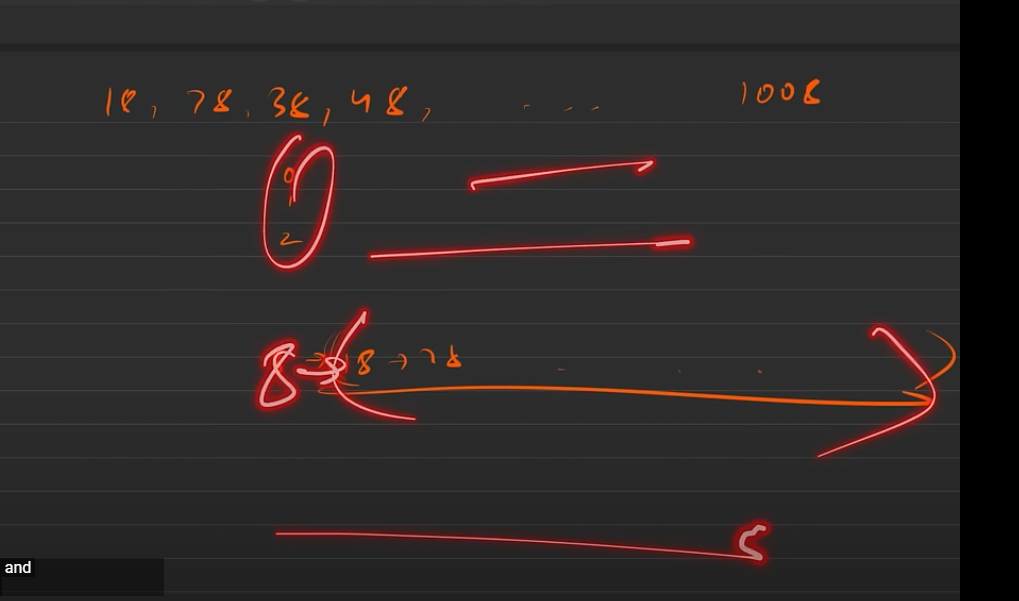
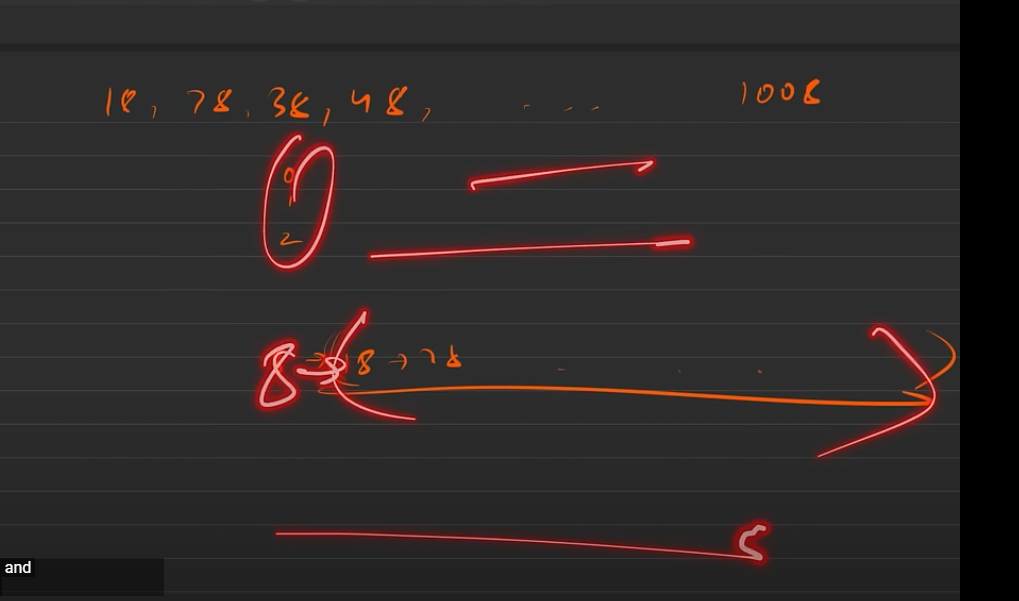




Now, suppose if someone comes and asks that how many times does 28 appears then, it will go to the index 8 and there since it knows that there are limited numbers stored in a sorted fashion so there are various searching algo. which are applied internally like you can easily do a binary search which can find and tell that there are 2 numbers and it can this in minimal time. Thus, chaining can be done at any index and you can always keep them in limited memory storage.

Now we don’t need to implement this, we just have to understand the concepts.

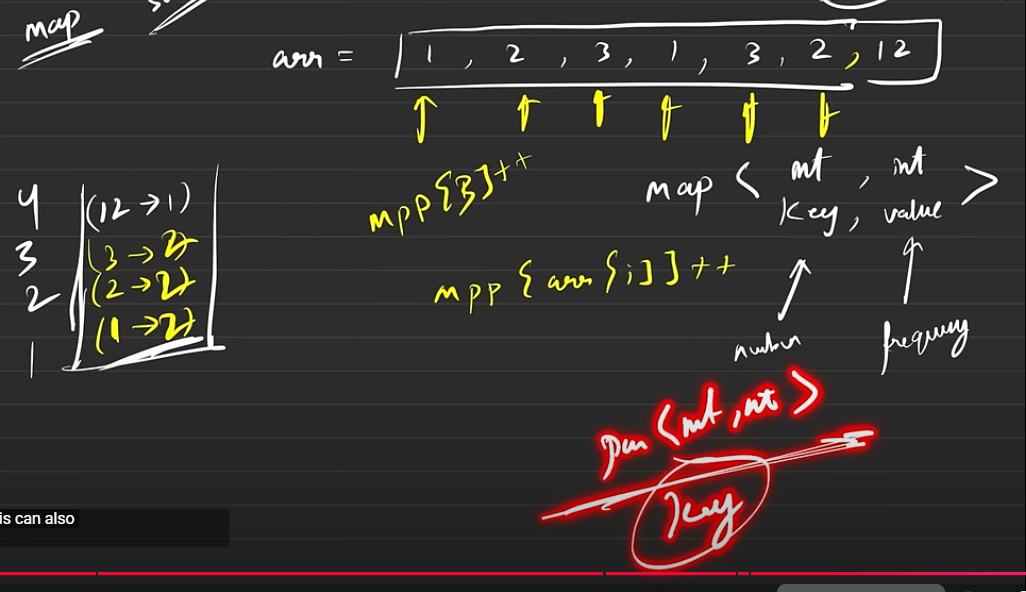
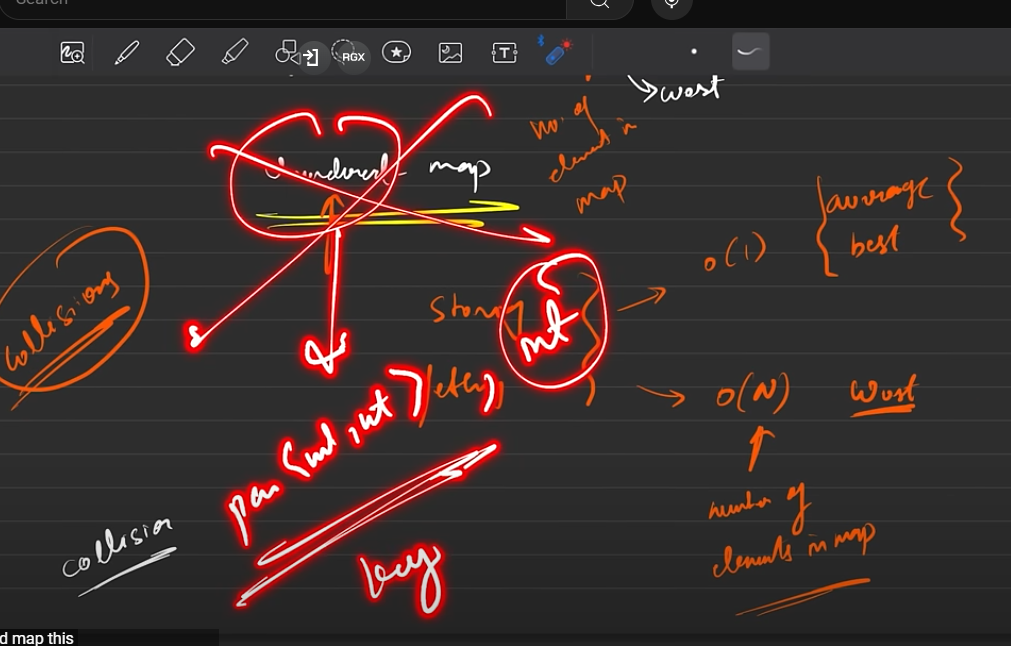
**Remember**: It’s not necessary that they use division method internally, they use a method which is a mixture of everything and they decide on the basis of the input given. So, whatever hash they are using internally cannot be determined by you. And, even then it is susceptible to fall for collisions which we are going to discuss now.



**Case Of Collision**

Imagine an array, where element went to the same hash space no other memories were used and they form a huge chain at a single hash space. This is what we call that Collision happened (All values end up at the same hash index). And this happen, then finding the frequency for the element is tough.

And this where the, Worst Case for Unordered\_map happens i.e. O(N) and that is why we say that its extremely rare to happen. (As the problems has to be extremely brilliant to make this case happen).



Any Datatype and Data-Structure can be a key in a map. In fact, same is the case with values.

Ex: It can be double, char, string or even a map etc.

However, we can only have individual datatypes as key in an Unordered map.  
Ex: You cannot have key with datatype pair and value int as mentioned in the image in UM. But, datatypes like int, char, double, string etc. are fine as keys. And same as map **values do NOT have the same restrictions as keys.**

Some More Insights and Clarification