We are going to talk about two very related data structures, sets and maps. So let's start with the basic sets object. So the set object is really useful when you want to know if an object is in the set or not in the set. So we can think of the set as some kind of a bucket that we can keep putting objects in and we can quickly check if the object is in the set or not. The set doesn't keep track of duplicates.

So if you have the same object or the same number put into the set multiple times, you'll still have only one way, only one of those items in the set. So a set can be useful, for example, for getting rid of duplicates. If you have a whole bunch of integers in a vector and there are some duplicates, you can take all of these integers, put them in a set, and then once you take them out, there are no more duplicates. The way we create a set using set followed by the data type we are going to use in this case integer. And there are two versions of set in C standard template library there is the regular set and then unordered set.

The set data structure is implemented using a binary tree, specifically red and black trees. So the insertion, deletion, search, all of those operations take order log n. So and because it's organized as a tree structure that the items in the set are also ordered from smallest to largest. So we insert into the sets 100, 20 and 50, and we can check using S count. And S count is either going to return one or zero. I don't know why they called it count since it only returns 0 or 1 and 1 indicates that it's in the set.

The next version of C, C20 or later, it introduced the concept of contains. So you could use the, you could use S contains to check if an item is in the set or not. But right now I believe I'm using C17, so we are going to skip over that part. And now that we have put some stuff into the sets, we can go through it and when we print it out, we get them in sorted order.

The next data structure map can be seen as an extension of the set. It's got a set of keys and values associated with the keys. So in a map the keys are unique. And in this case we have a keys that are going to be integers and the values, well, the values can be anything. But there is, for each key there is going to be a value associated with it.

And once again it's kept internally in a red and black tree binary Search tree structure. So operations are order log n. So we insert things into the tree using the square brackets, the index operator. So we can insert M100 equals string hundreds, M square bracket 20 equals string 20, M50 equals 50 and so on. And once again, just like the sets, we can check if something is in the map using dot count, dot count double equals one to check if something is in there. And if you wanted to get the value of the item, we would use M square bracket 20 and that's would give us the string 20 as part of it.

Since we are not using C20, it's going to skip over the next piece of code. And if we want to iterate through the map, we are going to get pairs and pairs is going to have a first and a second, a key and a value as part of it. And and just like the set, these are ordered from smallest to largest based on the key.

The unordered set uses a hash table underneath it, so it uses a hash function. Hash functions are usually mathematical functions that given a value, they do a computation and try to map it into one of the buckets in the hash table. And so as a result of that, the time complexity for insertion as well as erasing or lookup is going to be order one on average rather than order log n. With an unordered sets, we have the same structure. We can insert things into our unordered sets. We can check if something is in there using count and we can have a range based for loop to go through all the items.

But in this case we don't know what order they are going to come out. They could come out in any kind of order. So this time it comes out as 50, 20 and 100. So it came out in neither ascending or descending order. There is no ordering how the unordered set keeps it.

In general, if we don't need an ordering, we would use unordered set rather than a set, because unordered set is faster.

And just like the unordered sets, we also have the unordered map which has a key and a value pair. And internally it uses a hash table. So we have the average time complexity of order one constant time. We insert items using the square brackets. Once again, we check if an item is in there using the unordered map dot count operation and checking if that gives us one.

And when we take the items out we are going to have pairs, key and value pairs. And once again they come out in any possible order. There is no ordering for them. And that's it for sets, maps, unordered sets and unordered maps.