Okay, let's talk about vectors. So we talked about arrays. Arrays are a block of memory. Vectors are implemented in standard template library and they're similar to arrays, but have the added advantage that we don't have to specify the size at the very beginning and we can just keep expanding it so they are probably the best data structure to use. So let's look at vectors and starting with some vector operations so we can declare a vector.

Here we are declaring a vector with the sizes of 10, 20, 30, 40.

We can get the size of the vector. And once we have a vector, we can access all of its elements using a index based for loop, or we can use a range based for loop. The other thing that we can do with vectors is we can find the maximum elements using the standard template library function max element. So for that one we are going to give two iterators the beginning of the vector and the end of of the vector. An iterator is, allows us to go through a container without having to know the underlying data structure.

So we want to go from beginning to end, look at the maximum value and it returns an iterator. It's called it max. And we can dereference it and say, hey, the maximum element is going to be 50. And similarly, we do the same thing for the minimum elements. We also have other standard template library functions.

One of them is Olaf, where we are looking at all of the items in the vector and all of them have to return true. We would provide once again iterator to the beginning of the vector, iterator to the end of the vector. And here we have a lambda function that's going to return true if the number is even. Of course, we can do the same thing with if we have an external function such as is even is even is defined over here and it just says hey, it takes an integer and returns true if we have an even number. So whether we use all of and with iterators and a lambda function, or all of with iterators and an external function, the result is going to be the same.

We can sort a vector which is very useful. Once we have sorted it, we can do binary search on it. Binary search is going to be more efficient. It's going to be log n rather, rather, rather than an unsorted vector, we'd have to go through order N to find something. And we can see that we find our element 20, we can push back on onto the back vector, extending the size of the vector.

And so we are adding 6 and 70. Our vector wasn't Empty before, shouldn't be empty now. And we can even pop the back of our vector. And so right now we have 10 to 70. Once we pop our vector, 70 is gone.

So we want to add things to the back of the vector and pop things from the back of the vector if we need to. The other thing we can do is we can put an object in a vector. Let's say we have this class B I have it should be B int created. Anyway, we have this class B and we can create a B object object by giving it an integer or make a copy of it. And so we create our objects B object with one, then we create a B object with 10.

Then we create a vector of B objects and we push these objects. So one thing to notice is when we did the push with the object, we ended up making a copy of the objects. So when we are pushing something onto the vector, when we are putting an object into the vector, we are making a copy of the vector. So now we are setting a zero of X to to be seven. A zero was the one that was at that set vector is vector A index zero.

So what we, after modifying this, the one that is inside the vector, because it's a copy, didn't get modified. So now we have the A0 that we had created and the copy that is in in the vector. So we have two separate versions for simple objects. It may not make sense, but it's important to remember that if you put an object into a vector, we are putting a copy of the object and not the object itself. We can also use vectors as a map, and this comes in very conveniently when we are dealing with ASCII numbers.

The ASCII values, the kind of printable characters we see, go from 0 to 255 fin that lowercase A to Z corresponds to 97 to 122, uppercase A to Z correspond to 65 to 90. You don't really have to know these numbers, but you do need to know that they will convert to an integer. So if you have some kind of a text with a bunch of ASCII numbers we can go through and for each of the characters in the text we can increase the frequency. So we are calculating the frequency for all of the characters that is part of the text and hopefully we'll get there soon. It is a long piece of text.

Ah, finally. And so now we have the frequency frequency map using our vector and we're also going to look and keep track of what's the most frequent one. So we can say hey, we are only interested in characters from A to Z. We don't have to go through the whole set if we wanted to. And we say hey, if the character is has appeared, we print how many times it has appeared.

So we go through all of the characters from A to Z. Essentially it's as if we are going from 97 to 122, but our for loop doesn't care. We can use the same plus plus notation as part of it. And we are at x, so we are most frequent character e with frequency of 4, and those are the basic vector operations. There is additional operations for vectors such as being able to insert into the middle or erase something from from the middle, but those are not as efficient.

The most effic operations are adding to the back of the vector or given an index jumping to that index for vector.