Let's talk about priority queues. So priority queues are useful when we have a set of items and we want them sorted possibly from largest to smallest, and we want to keep picking up the largest one again and again. And if we want to insert a new one, well, we want to make sure that it remains sorted. So the way we do priority queues, let's start with the default priority queue and which is going to use the standard template libraries priority queue and data structure. ISP define priority queue and we decide that there is going to be a set of integers put into it.

If internally priority queue is implemented using a max heap, a binary tree where the parent node is always greater than or equal to its child nodes and the top element is always going to be the largest element in the queue. So getting the top element is a order order one operation, and then it takes another order log n operation to reprioritize the queue. So once we have established our priority queue, we are going to put 30, 20 and then 50 into it. And now the size of our priority queue is going to be three. And regardless of how we have in which order we have input things into the priority queue, it should be from the largest to the smallest.

So while it's not empty, we'll get the top element. So we get 50, then 30 and then 20 out of our priority queue. Just like with queues, priorityqueue top accesses the top element and the dot pop is the one that's going to remove the top element out.

Next we are going to look at reversed priority queue. So this is the case when we want items from smallest to largest. So once again we are going to define our priority queue. It's going to have a set of integers and as a part of the definition we need to tell what the internal data structure is going to be. We say it's going to be a vector.

And I have always chosen vector, I've never chosen anything else. Not really sure why we get a choice. And since we want things to be smaller to larger, we are going to say we want the greater relationship between the items. You can think of it as item is item greater than another, item greater than another, item greater than. And this last item over here in the smallest one is the one that we are going to be picking up.

So once again we are inserting three, pushing 30, 20 and 50. Our size of the priority queue is once again three. And while it's not empty, we take out the elements one by one, and now they come out in the order 20, 30, 50 smallest to largest. Sometimes we might want to put more complex objects into a priority queue. For example, we might want to put a pair into the queue.

So a pair is a very simple structure. It essentially takes two elements and then just kind of wraps them up and you can access them using dot first and dot second. And when you're creating a pair, you can also use curly braces, or you can use make underscore pair to make the pair. When pairs are sorted, they'll get sorted by the first element, and if it's the same, then they'll get sorted by the second element. So once again, we push 30, 20 and 50 into our priority queue.

We have a priority queue of once again and three. And while it's not empty, we can take out our elements. Now, each of the elements we take out has two parts, the dot first and the dot second part that comes out. So this is useful when you want to have two bits of information that need to travel together. Maybe one of the items is the priority of the item, the other one is something else about the item as part of it.

We can also put objects into a priority queue. So let's say we want to create a priority queue of A objects. But. But before we start pushing A objects, let's have a look at the A data structure. The A class we just have a constructor and A has a public integer X as part of it.

Because we are going to put it into the priority queue. We need to define operator less than as part of it. So we define operator less than, and in this case we define it as X is less than the other dot X. And so that we have a way of comparing 2A objects with that out of the way, and we can push objects a 30, a 20, a 50 into our priority queue and then one by one take them out of the priority queue. Of course, the problem is with putting an object into priority queue, just like putting other containers into other objects into containers, is you end up making a copy of the object.

So in the case of class A, we may not care about it, but for more complex objects, we do not want to make copies of the objects. At the same time, we can't really put a pointer into the priority queue. Well, we could, but then the default priority queue will look at that pointer and look at it as a unsigned 64 bit number and would use the memory addresses to do the sorting and not the objects as part of it. So if you want to put pointers into a priority queue, the way we have to declare it is a Little bit more complicated. So we are setting up a priority queue.

It's going to have B prime objects. The container we want to use is going to be a vector of B prime objects. And the next thing we do is we are declaring and that the comparison is going to be of type this, my object comparison and my object comparison is is this function that's defined over here where it takes two pointers to different B objects and then compares the x values of the left hand side and the right hand side. And once again, we are using less than because we want to create a max heap as part of it. So declare type is not something that you see very frequently.

What it does is it tells the compiler saying, hey, look at this variable or look at whatever is inside the parentheses and give me the type of that. And a lot of times this is used when you have a complex type as part of it. And so now that we have set up the priority queue, we make sure that this comparator is going to get used to do the comparison. The other way we could set up our priority queue would be by explicitly saying, hey, we are going to have a pointer to a function that returns a boolean and takes B pointer and B pointer, so that takes 2B pointer objects and returns a boolean, and we are going to have a pointer to that function. So either way will work as part of it.

And we are passing the address of the function so that priority queue can call that function. So now that we have created our priority queue, we push B objects. Now these are dynamic objects created on the heap using new with 30, 20 and 50, we have priority queue with three elements and we can look at the top element and as expected, we get the top element to be 50 and the next element after that is going to be 30, and so on. So this way we are able to work with putting pointers into priority queue and making sure that the sorting is using the actual object and whatever value we want from that object rather than the memory address of the object. And that's it for priority queues.