C++: Custom Sort via overloading < operator for a class

One way to create a custom sort in C++ is to create a class (or struct) and then overload the < operator, which the sort function uses. This is illustrated in the practice program, sortnames.cpp. In this program, the objects being sorted are people, and each person has a name, age and amount of money. We want to sort the list by money (greatest to least), breaking ties by age (least to greatest). In the class, you define attributes and a constructor. Here is the overloaded operator from that example:

```
bool operator<(const person& other) const {
    if (money != other.money)
        return money > other.money;
    return age < other.age;
}
```

The key here is that this is a method in the class, so it just takes in the other object that gets passed into it, and the method itself is called upon the first object. So when you do a < b, this method is being called on object a, and other is object b. If this object's money is greater than other's, then we want this object to "come first" (be less than) other. If money is equal, we will drop down to the second return, where this comes before other if and only if this object is younger.

To sort, we just do this:

```
sort(list.begin(), list.end());
```

This method is included in algorithm, and we just pass it the portion of the vector we want sorted. (list is a vector of class person.)

C++: Use of STL class priority_queue

By default, a priority queue in C++ is a max queue. Here is how to declare a priority queue of integers:

```
priority_queue<int> pq;
```

Here are the key operations of a priority queue, taken from this site:

https://cplusplus.com/reference/queue/priority_queue/

Member functions

(constructor)	
	Construct priority queue (public member function)
empty	
	Test whether container is empty (public member function)
size	
	Return size (public member function)
top	

	Access top element (public member function)
push	Insert element (public member function)
emplace	Construct and insert element (public member function)
pop	Remove top element (public member function)
swap	Swap contents (public member function)

Of these, we'll mostly use push, top, pop and size. Recall that push, top and pop will all run in O(lg n) time, where n is the number of items in the priority queue.

Let's say we want a min queue of ints. We declare it as follows, by specifying our comparator:

```
priority_queue <int, vector<int>, greater<int>> pq;
```

This basically swaps the comparator operator, putting greater than in place for where the less than operator "would go."

One really important thing to note is that the pop function is void. So, you need to call top first to get the item before you pop it!

Alternatively, if we want to do a priority_queue of a struct, but make it a min queue (for example, Dijkstra's algorithm requires this), then we could do the following:

```
struct estimate {
    int x,y,cost;

    bool operator<(const estimate& other) const {
        return other.cost < cost;
    }
};
```

and then declare an object the usual way:

```
priority_queue<estimate> pq;
```

Here is how to push something onto this priority queue:

```
pq.push(estimate{sX, sY, 0});
```

Here, sX is the x value of the struct, sY is the y value of the struct and 0 is the cost for this struct.

In the practice programs, a solution to the problem Add All is included.

C++: Use of STL class set

A set is a collection of objects without repeats (if you add an item that already exists in a set to it, no change is made to the set). In C++ the amount of time it takes to add an item is O(lg n) and the amount of time it takes to retrieve an item is O(lg n), because a set in C++ is ordered. (In Java and Python sets are unordered and support O(1) run times.)

Full list of method can be found here: https://cplusplus.com/reference/set/set/

Here are the some methods from the set class in C++:

insert	Insert element (public member function)
erase	Erase elements (public member function)
swap	Swap content (public member function)
clear	Clear content (public member function)
emplace	Construct and insert element (public member function)
emplace_hint	Construct and insert element with hint (public member function)

find	Get iterator to element (public member function)
count	Count elements with a specific value (public member function)
lower_bound	Return iterator to lower bound (public member function)
upper_bound	Return iterator to upper bound (public member function)
equal_range	Get range of equal elements (public member function)

Typically, with a set, we want to insert, remove and search for items quickly, where duplicates don't matter. In the posted solution for the problem CD, we simply create an set of ints for Jack's CDs and then we can quickly try to find each of Jill's CDs in the set. Thus, we just use the insert and find methods in the solution. This is fairly typical use of sets.

<u>C++: Use of STL class map</u>
A map is similar to a set, maintaining a set of objects. But in addition, in a map, we are allowed to link each object with an associated piece of data, similar to a dictionary (which stores words, and then links each word to a definition.) We call the pieces of data in the map keys, and the information linked to a key is its associated value. Here are some examples of uses of a map:

1) Mapping names of people to integers, sort of as a code:

Bob → 0
Aline → 1
Cindy → 2
Damarcus → 3

The purpose of this would be then to use an array where index 2, for example refers to Cindy…

2) Mapping names to ages or phone numbers or any other piece of information you want to associate with a person.

3) Mapping names to say, the number of votes they've earned in an election, this is the example program election3.cpp.

Full set of methods can be found here: https://cplusplus.com/reference/map/map/

Here are some methods from the map class in C++:

operator[]	Access element (public member function)
at	Access element (public member function)

Modifiers:

insert	Insert elements (public member function)
erase	Erase elements (public member function)
swap	Swap content (public member function)
clear	Clear content (public member function)
emplace	Construct and insert element (public member function)
emplace_hint	Construct and insert element with hint (public member function)

Observers:

key_comp	
	Return key comparison object (public member function)
value_comp	
	Return value comparison object (public member function)

Operations:

find	
	Get iterator to element (public member function)
count	
	Count elements with a specific key (public member function)
lower_bound	
	Return iterator to lower bound (public member function)
upper_bound	
	Return iterator to upper bound (public member function)
equal_range	
	Get range of equal elements (public member function)

In the example election3.cpp, we map names to the number of votes a name has gotten.

Each time we read in a vote, we check our map to see if that person's gotten a vote before. If they haven't we add them to the map with 1 vote. If they have, we retrieve their number of votes, remove their entry from the map, and then place that person with 1 more vote back into the map. In Python and Java, we could just insert the item and it would overwrite the old one, but in C++, we must remove the item and then reinsert it with the newly mapped value.