function template

std::sort

<algorithm>

default (1)	`template <class RandomAccessIterator>` `  void sort (RandomAccessIterator first, RandomAccessIterator last);`
custom (2)	`template <class RandomAccessIterator, class Compare>` `  void sort (RandomAccessIterator first, RandomAccessIterator last, Compare comp);`

Sort elements in range

Sorts the elements in the range `[first,last)` into ascending order.

The elements are compared using `operator<` for the first version, and *comp* for the second.

Equivalent elements are not guaranteed to keep their original relative order (see `stable_sort`).

Parameters

`first, last`
Random-access iterators to the initial and final positions of the sequence to be sorted. The range used is `[first,last)`, which contains all the elements between *first* and *last*, including the element pointed by *first* but not the element pointed by *last*.
`RandomAccessIterator` shall point to a type for which `swap` is properly defined and which is both *move-constructible* and *move-assignable*.

`comp`
Binary function that accepts two elements in the range as arguments, and returns a value convertible to `bool`. The value returned indicates whether the element passed as first argument is considered to go before the second in the specific *strict weak ordering* it defines.
The function shall not modify any of its arguments.
This can either be a function pointer or a function object.

Return value

none

Example

```
// sort algorithm example
#include <iostream>     // std::cout
#include <algorithm>    // std::sort
#include <vector>       // std::vector

bool myfunction (int i,int j) { return (i<j); }

struct myclass {
  bool operator() (int i,int j) { return (i<j);}
} myobject;

int main () {
  int myints[] = {32,71,12,45,26,80,53,33};
  std::vector<int> myvector (myints, myints+8);               // 32 71 12 45 26 80 53 33

  // using default comparison (operator <):
  std::sort (myvector.begin(), myvector.begin()+4);           //(12 32 45 71)26 80 53 33

  // using function as comp
  std::sort (myvector.begin()+4, myvector.end(), myfunction); // 12 32 45 71(26 33 53 80)

  // using object as comp
  std::sort (myvector.begin(), myvector.end(), myobject);     //(12 26 32 33 45 53 71 80)

  // print out content:
  std::cout << "myvector contains:";
  for (std::vector<int>::iterator it=myvector.begin(); it!=myvector.end(); ++it)
    std::cout << ' ' << *it;
  std::cout << '\n';

  return 0;
}
```

Output:

```
myvector contains: 12 26 32 33 45 53 71 80
```

Complexity

On average, linearithmic in the distance between *first* and *last*: Performs approximately $N*\log_2(N)$ (where *N* is this distance) comparisons of elements, and up to that many element swaps (or moves).

Data races

The objects in the range `[first,last)` are modified.

Exceptions

Throws if any of the element comparisons, the element swaps (or moves) or the operations on iterators throws.
Note that invalid arguments cause *undefined behavior*.

See also

stable_sort	Sort elements preserving order of equivalents (function template)
partial_sort	Partially sort elements in range (function template)
search	Search range for subsequence (function template)

reverse	Reverse range (function template)