排序算法：只适用于提供random-access iterator的容器：array, vector, deque, string, C-style array。

1. 对所有元素排序，sort

void

sort(RandomAccessIterator beg, RandomAccessIterator end)

void

sort(RandomAccessIterator beg, RandomAccessIterator end,

BinaryPredicate op)

void

stable\_sort(RandomAccessIterator beg, RandomAccessIterator end)

void

stable\_sort(RandomAccessIterator beg, RandomAccessIterator end,

BinaryPredicate op)

sort()和stable\_sort()的第一形式使用operator < 对区间[beg, end)内的所有元素排序。

sort()和stable\_sort()的第二形式使用binary predicate

op(elem1, elem2)

作为排序准则。

sort()和stable\_sort()的区别是，后者保证相等之各元素的相对次序在排序后保持不变。

sort()：平均复杂度：nlogn

stable\_sort：内存足够大，nlogn，内存不够大：nlogn\*logn

例：程序stl\_test76

// sort

*deque*<int> coll;

InsertElements(coll, 1, 9);

InsertElements(coll, 1, 9);

PrintElements(coll, "on entry: ");

*sort*(coll.*begin*(), coll.*end*());

PrintElements(coll, "sorted: ");

*sort*(coll.*begin*(), coll.*end*(), *greater*<int>());

PrintElements(coll, "sorted > : ");

输出为：

on entry: 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9

sorted: 1 1 2 2 3 3 4 4 5 5 6 6 7 7 8 8 9 9

sorted > : 9 9 8 8 7 7 6 6 5 5 4 4 3 3 2 2 1 1

例：程序stl\_test76

// stable\_sort

*vector*<*string*> coll1 = {"1xxx", "2x", "3x", "4x", "5xx",

"6xxxx", "7xx", "8xxx", "10xxx",

"11", "12", "13", "14xx", "15", "16", "17"};

*vector*<*string*> coll2(coll1);

PrintElements(coll1, "on entry:\n");

// 在visual studio中二者貌似没区别

*sort*(coll1.*begin*(), coll1.*end*(), LessLength);

*stable\_sort*(coll2.*begin*(), coll2.*end*(), LessLength);

PrintElements(coll1, "\nwith sort(): \n");

PrintElements(coll2, "\nwith stable\_sort():\n");

输出为：

on entry:

1xxx 2x 3x 4x 5xx 6xxxx 7xx 8xxx 10xxx 11 12 13 14xx 15 16 17

with sort():

2x 3x 4x 11 12 13 15 16 17 5xx 7xx 1xxx 8xxx 14xx 6xxxx 10xxx

with stable\_sort():

2x 3x 4x 11 12 13 15 16 17 5xx 7xx 1xxx 8xxx 14xx 6xxxx 10xxx

1. 局部排序，partial\_sort，复杂度在线性和nlogn之间

void

partial\_sort(RandomAccessIterator beg, RandomAccessIterator sortEnd,

RandomAccessIterator end)

void

partial\_sort(RandomAccessIterator beg, RandomAccessIterator sortEnd,

RandomAccessIterator end, BinaryPredicate op)

第一形式以opertor < 对[beg, end)区间内的元素排序，使[beg, sortEnd)区间内的元素处于已序状态。

第二形式使用binary predicate

op(elem1, elme2)

作为排序准则，使[beg, sortEnd)区间内的元素处于已序状态。

例：程序stl\_test76

// partial\_sort

*deque*<int> coll3;

InsertElements(coll3, 3, 7);

InsertElements(coll3, 2, 6);

InsertElements(coll3, 1, 5);

PrintElements(coll3, "coll3: ");

// sort until the first five elements are sorted

// 对整个序列排序，只需要前5个已序

*partial\_sort*(coll3.*begin*(), coll3.*begin*() + 5, coll3.*end*());

PrintElements(coll3, "after first five elements sorted: ");

//sort inversely until the first five elements are sorted

*partial\_sort*(coll3.*begin*(), coll3.*begin*() + 5,

coll3.*end*(), *greater*<int>());

PrintElements(coll3, "after first five elements: ");

输出为：

coll3: 3 4 5 6 7 2 3 4 5 6 1 2 3 4 5

after first five elements sorted: 1 2 2 3 3 7 6 5 5 6 4 4 3 4 5

after first five elements: 7 6 6 5 5 1 2 2 3 3 4 4 3 4 5

1. 根据第n个元素排序，复杂度平均为线性

void

nth\_element(RandomAccessIterator beg, RandomAccessItertor nth,

RandomAccessIterator end)

void

nth\_element(RandomAccessIterator beg, RandomAccessIterator nth,

RandomAccessIterator end, BinaryPredicate op)

两种形式都对[beg, end)区间内的元素排序，使第n个位置上的元素就位，也就是说所有在位置n之前的元素都小于等于他，所有在位置n后的元素都大于等于它。可以得到“根据n位置上的元素”分割开来的两个子序列（在visual studio中貌似会把整个序列都排序）。

例：程序stl\_test76

// nth\_element

*deque*<int> coll4;

InsertElements(coll4, 3, 7);

InsertElements(coll4, 2, 6);

InsertElements(coll4, 1, 5);

PrintElements(coll4, "coll4: ");

// extract the four lowest elements

// 在visual studio中貌似会把整个

// 序列排序

*nth\_element*(coll4.*begin*(), coll4.*begin*() + 3, coll4.*end*());

*cout* << "the four lowest elements are: ";

*copy*(coll4.*cbegin*(), coll4.*cbegin*() + 4,

*ostream\_iterator*<int>(*cout*, " "));

*cout* << *endl*;

PrintElements(coll4, "coll4: ");

// extract the four highest elements

*nth\_element*(coll4.*begin*(), coll4.*end*() - 4, coll4.*end*());

*cout* << "the four highest elements: ";

*copy*(coll4.*cend*() - 4, coll4.*cend*(), *ostream\_iterator*<int>(*cout*, " "));

*cout* << *endl*;

PrintElements(coll4, "coll4: ");

// extract the four highest elements(second version)

*nth\_element*(coll4.*begin*(), coll4.*begin*() + 3, coll4.*end*(), *greater*<int>());

*cout* << "the four highest element are: ";

*copy*(coll4.*cbegin*(), coll4.*cbegin*() + 4, *ostream\_iterator*<int>(*cout*, " "));

*cout* << *endl*;

PrintElements(coll4, "coll4: ");

输出为：

coll4: 3 4 5 6 7 2 3 4 5 6 1 2 3 4 5

the four lowest elements are: 1 2 2 3

coll4: 1 2 2 3 3 3 4 4 4 5 5 5 6 6 7

the four highest elements: 5 6 6 7

coll4: 1 2 2 3 3 3 4 4 4 5 5 5 6 6 7

the four highest element are: 7 6 6 5

coll4: 7 6 6 5 5 5 4 4 4 3 3 3 2 2 1