algorithm 函数表

accumulate(iterator beg, iterator end,init)
adjacent_different：
adjacent_find (iterator beg, iterator end)：
binary_search (iterator beg, iterator end, value)：
copy (iterator beg, iterator end,iterator dest)：
copy_backward：
count (iterator beg, iterator end, value)：
count_if (iterator beg, iterator end, _Pred)：
equal：
equal_range：
fill(iterator beg, iterator end, value)：
fill_n：
find (iterator beg, iterator end, value)：
find_if (iterator beg, iterator end, _Pred)：
find_end：
find_first_of：
for_each (iterator beg, iterator end, _fuc) ：
generate：
generate_n：
includes：
inner_product：
inner_merge：
iter_swap：
lexicographical_compare：
lower_bound (iterator beg, iterator end, value)：
max：
max_element：
min：
min_element：
merge (iterator beg1, iterator end1,iterator beg2, iterator end2,iterator dest)：
mismatch：
next_permutation：
nth_element：
partial_sort：
partial_sort_copy：
partial_sum：
partition：
prev_permutation：
random_shuffle (iterator beg, iterator end)：
remove：
remove_copy：
remove_if：
remove_copy_if：
replace (iterator beg, iterator end, oldvalue, newvalue)：
replace_copy：
replace_if (iterator beg, iterator end, _Pred, newvalue)：
replace_copy_if：
reverse (iterator beg, iterator end)：
reverse_copy：
rotate：
rotate_copy：

search：
search_n：
set_difference (iterator beg1, iterator end1,iterator beg2, iterator end2,iterator dest)：
set_intersection (iterator beg1, iterator end1,iterator beg2, iterator end2,iterator dest)：
set_symmetric_difference：
set_union (iterator beg1, iterator end1,iterator beg2, iterator end2,iterator dest)：
sort (iterator beg, iterator end, _Pred)：
stable_partition：
stable_sort：
swap(container a, container b)：
swap_range：
transform (itreator beg1, itreator end1, itreator beg2 ,_fuc)：
unique(iterator it_1,iterator it_2,bool MyFunc)：
unique_copy：
upper_bound：
堆算法：
make_heap：
pop_heap：
push_heap：
sort_heap：

accumulate(iterator beg, iterator end,init)

- iterator 对标志的序列中的元素之和，加到一个由 **init 指定的初始值**上。重载的版本不再做加法，而是传进来的二元操作符被应用到元素上。

adjacent_different：

- 创建一个新序列，该序列的每个新值都代表了当前元素与上一个元素的差。重载版本用指定的二元操作计算相邻元素的差。

adjacent_find (iterator beg, iterator end)：

- 在 iterator 对标志的元素范围内，查找一对相邻的重复元素，如果找到返回一个 ForwardIterator ，指向这对元素的**第一个元素**。否则返回 last 。**重载版本使用输入的二元操作符代替相等的判断。**

binary_search (iterator beg, iterator end, value)：

- 在**有序序列**中查找 value ， 如果**找到返回 true 。重载的版本使用指定的比较函数对象或者函数指针来判断相等。**

copy (iterator beg, iterator end,iterator dest)：

- 复制序列。

copy_backward：

- 除了元素以相反的顺序被拷贝外，别的和 copy 相同。

count (iterator beg, iterator end, value):

- 利用等于操作符，把标志范围类的元素与输入的值进行比较，并返回相等元素的个数。

count_if (iterator beg, iterator end, _Pred):

- 对于标志范围类的元素，应用输入的操作符，并返回结果为 true 的次数。

equal：

- 如果两个序列在范围内的元素都相等，则 equal 返回 true 。重载版本使用输入的操作符代替了默认的等于操作符。

equal_range：

- 返回一对 iterator ，第一个 iterator 表示由 lower_bound 返回的 iterator ，第二个表示由 upper_bound 返回的 iterator 值。

fill(iterator beg, iterator end, value)：

- 将输入的值的拷贝赋给范围内的每个元素。

fill_n：

- 将输入的值赋值给 first 到 frist+n 范围内的元素。

find (iterator beg, iterator end, value):

- 利用底层元素的等于操作符，对范围内的元素与输入的值进行比较。当匹配时，结束搜索，返回该元素的一个 InputIterator 。
- 如果是结构体，**需要重载==**

```
struct node{
    int time;
    int name;
    bool operator == (const struct node &p)
    {
        if(this->name == p.name && this->time == p.time)
            return true;
        else
            return false;
    }
}Node[200];
```

find_if (iterator beg, iterator end, _Pred):

- 使用输入的函数替代了等于操作符执行了 find 。
- _Pred 函数或者谓词（返回值为bool类型的仿函数）

find_end：

- 在范围内查找"由输入的另外一个 iterator 对标志的第二个序列"的最后一次出现。重载版本中使用了用户输入的操作符替代等于操作。

find_first_of：

- 在范围内查找“由输入的另外一个 iterator 对标志的第二个序列”中的任意一个元素的第一次出现。重载版本中使用了用户自定义的操作符。

for_each（iterator beg, iterator end, _fuc）：

- 依次对范围内的所有元素执行输入的函数。
- _fuc可以是一般函数，也可以是重载括号的仿函数，但是传递仿函数的时候是传递一个类。

generate：

- 通过对输入的函数 gen 的连续调用来填充指定的范围。

generate_n：

- 填充 n 个元素。

includes：

- 判断 [first1, last1) 的一个元素是否被包含在另外一个序列中。使用底层元素的 <= 操作符，重载版本使用用户输入的函数。

inner_product：

- 对两个序列做内积 (对应的元素相乘，再求和)，并将内积加到一个输入的的初始值上。重载版本使用了用户定义的操作。

inner_merge：

- 合并两个排过序的连续序列，结果序列覆盖了两端范围，重载版本使用输入的操作进行排序。

iter_swap：

- 交换两个 ForwardIterator 的值。

lexicographical_compare：

- 比较两个序列。重载版本使用了用户自定义的比较操作。

lower_bound (iterator beg, iterator end, value)：

- 返回一个 iterator，它指向在范围内的有序序列中可以插入指定值而不破坏容器顺序的第一个位置。重载函数使用了自定义的比较操作。

max：

- 返回两个元素中的较大的一个，重载版本使用了自定义的比较操作。

max_element：

- 返回一个 iterator，指出序列中最大的元素。重载版本使用自定义的比较操作。

min：

- 两个元素中的较小者。重载版本使用自定义的比较操作

min_element：

- 类似与 max_element，不过返回最小的元素。

merge (iterator beg1, iterator end1,iterator beg2, iterator end2,iterator dest)：

- 合并两个**有序序列**，并存放到另外一个序列中。重载版本使用自定义的比较。

mismatch：

- 并行的比较两个序列，指出第一个不匹配的位置，它返回一对 iterator，标志第一个不匹配的元素位置。如果都匹配，返回每个容器的 last 。重载版本使用自定义的比较操作。

next_permutation：

- 取出当前范围内的排列，并将其重新排序为下一个排列。重载版本使用自定义的比较操作。

nth_element：

- 将范围内的序列重新排序，使所有小于第 n 个元素的元素都出现在它前面，而大于它的都出现在后面，重载版本使用了自定义的比较操作。

partial_sort：

- 对整个序列做部分排序，被排序元素的个数正好可以被放到范围内。重载版本使用自定义的比较操作。

partial_sort_copy：

- 与 partial_sort 相同，除了将经过排序的序列复制到另外一个容器。

partial_sum：

- 创建一个新的元素序列，其中每个元素的值代表了范围内该位置之前所有元素之和。重载版本使用了自定义操作替代加法。

partition：

- 对范围内元素重新排序，使用输入的函数，把计算结果为 true 的元素都放在结果为 false 的元素之前。

prev_permutation：

- 取出范围内的序列并将它重新排序为上一个序列。如果不存在上一个序列则返回 false 。重载版本使用自定义的比较操作。

random_shuffle (iterator beg, iterator end)：

- 对范围内的元素随机调整次序。重载版本输入一个随机数产生操作。

remove：

- 删除在范围内的所有等于指定的元素，注意，该函数并不真正删除元素。内置数组不适合使用 remove 和 remove_if 函数。

remove_copy：

- 将所有不匹配的元素都复制到一个指定容器，返回的 OutputIterator 指向被拷贝的末元素的下一个位置。

remove_if：

- 删除所有范围内输入操作结果为 true 的元素。

remove_copy_if：

- 将所有不匹配的元素拷贝到一个指定容器。

replace (iterator beg, iterator end, oldvalue, newvalue):

- 将范围内的所有等于 old_value 的元素都用 new_value 替代。

replace_copy：

- 与 replace 类似，不过将结果写入另外一个容器。

replace_if (iterator beg, iterator end, _Pred, newvalue):

- 将范围内的所有操作结果为 true 的元素用新值替代。

replace_copy_if：

- 类似与 replace_if，不过将结果写入另外一个容器。

reverse (iterator beg, iterator end):

- 将范围内元素重新按反序排列。

reverse_copy：

- 类似与 reverse，不过将结果写入另外一个容器。

rotate：

- 将范围内的元素移到容器末尾，由 middle 指向的元素成为容器第一个元素。

rotate_copy：

- 类似与 rotate，不过将结果写入另外一个容器。

search：

- 给出了两个范围，返回一个 iterator，指向在范围内第一次出现子序列的位置。重载版本使用自定义的比较操作。

search_n：

- 在范围内查找 value 出现 n 次的子序列。重载版本使用自定义的比较操作。

set_difference (iterator beg1, iterator end1,iterator beg2, iterator end2,iterator dest):

- 构造一个**排过序的序列**，其中的元素出现在第一个序列中，但是不包含在第二个序列中。重载版本使用自定义的比较操作。

set_intersection (iterator beg1, iterator end1,iterator beg2, iterator end2,iterator dest):

- 构造一个**排过序的序列**，其中的元素在两个序列中都存在。重载版本使用自定义的比较操作。
- ```
  set_intersection(v1.begin(),v1.end(),v2.begin(),v2.end(),inserter(v3,v3.begin()));
  ```
  ```

set_symmetric_difference：

- 构造一个**排过序的序列**，其中的元素在第一个序列中出现，但是不出现在第二个序列中。重载版本使用自定义的比较操作。

set_union (iterator beg1, iterator end1,iterator beg2, iterator end2,iterator dest)：

- 构造一个排过序的序列，它包含两个序列中的所有的不重复元素。重载版本使用自定义的比较操作。

sort (iterator beg, iterator end, _Pred)：

- 以升序重新排列范围内的元素，重载版本使用了自定义的比较操作。

stable_partition：

- 与 partition 类似，不过它不保证保留容器中的相对顺序。

stable_sort：

- 类似与 sort，不过保留相等元素之间的顺序关系。

swap(container a, container b)：

- 交换存储在两个对象中的值。

swap_range：

- 将在范围内的元素与另外一个序列的元素值进行交换。

transform (itreator beg1, itreator end1, itreator beg2 ,_fuc)：

- beg1 源容器起始迭代器，end1源容器结束迭代器， beg2 目标容器起始迭代器， _fuc函数或函数对象。
- 注意目标容器需要提前开辟好空间，否则会运行失败
- 将输入的操作作用在范围内的每个元素上，并产生一个新的序列。重载版本将操作作用在一对元素上，另外一个元素来自输入的另外一个序列。结果输出到指定的容器。

unique(iterator it_1,iterator it_2,bool MyFunc)：

- 清除序列中重复的元素，和 remove 类似，它也不能真正的删除元素。重载版本使用了自定义的操作。
- ```
  v.erase(unique(v.begin(),v.end()),v.end());
  ```

unique_copy：

- 类似与 unique，不过它把结果输出到另外一个容器。

upper_bound：

- 返回一个 iterator，它指向在范围内的有序序列中插入 value 而不破坏容器顺序的最后一个位置，该位置标志了一个大于 value 的值。重载版本使用了输入的比较操作。

堆算法：

- C++ 标准库提供的是 max-heap 。一共由以下 4 个泛型堆算法。
  ```

make_heap：

- 把范围内的元素生成一个堆。重载版本使用自定义的比较操作。

pop_heap：

- 并不是真正的把最大元素从堆中弹出，而是重新排序堆。它把 first 和 last-1 交换，然后重新做成一个堆。可以使用容器的 back 来访问被“弹出”的元素或者使用 pop_back 来真正的删除。重载版本使用自定义的比较操作。

push_heap：

- 假设 first 到 last-1 是一个有效的堆，要被加入堆的元素在位置 last-1 ，重新生成堆。在指向该函数前，必须先把元素插入容器后。重载版本使用指定的比较。

sort_heap：

- 对范围内的序列重新排序，它假设该序列是个有序的堆。重载版本使用自定义的比较操