排序

使用c++STL库的sort方法可以通过 `#include <algorithm>` 来引入。

- 可以排序数组，vector等数据结构。
- 可以自定义排序规则。

排序数组

```
int arr[6] = {2,4,6,1,3,5};
sort(arr, arr + 6); // 第一个参数填起始地址，第二个参数填写最后一个元素的后一个位置的地址，相当于左闭右开
```

> **Note**
>
> C语言语法： `arr` 是一个数组名，作为函数参数时会退化成数组中第一个元素的地址

排序动态数组

```
vector<int> vec = {2,4,6,1,3,5};
sort(vec.begin(), vec.end()); // 两个参数都是迭代器
```

> **Note**
>
> C++语法：函数重载，不同的函数使用同一个名字

自定义比较规则

sort底层是快速排序，是一种基于比较的排序。sort本身是不稳定的，但是可以通过一定的手段实现稳定排序，例如结构体加入序号信息。

```
/*第一个参数和第二个参数不交换时，需返回true*/
bool compare(int lhs, int rhs) {
    return lhs >= rhs; // 第一个参数比第二个参数大，则不交换
}
sort(vec.begin(), vec.end(), compare);
```

compare函数设计

1. 返回bool类型
2. 函数名自定义，保持和sort的第三个参数一致即可
3. 两个参数的类型和容器元素类型一致
4. 当左边和右边不发生交换时，返回true

map按值比较

```
bool compare(pair lhs, pair, rhs) {
    ...
}

// 需要先将map中的pair放到vector中，然后对vector进行排序
map<int, int> map1 = {{1,1}, {2,2}, {3,3}};
vector<pair<int, int>> vec(map1.begin(), map1.end()); // 也可以遍历map，把pair push到vec中
sort(vec.begin(), vec.end(), compare);

//之后对vec进行操作即可
```

查找

顺序查找

STL库中的查找函数需导入 `#include <algorithm>`

`find(begIt, endIt, x)`：查找x在数组中的位置。如果x存在，返回的是x位置的迭代器；如果不存在，返回的是数组最后一个元素后一个位置的迭代器。

`begIt` 数组第一个位置的迭代器，如果查找的是静态数组，应为数组第一个元素的地址

`endIt` 数组最后一个位置的后一个位置的迭代器， 如果是静态数组，应为数组最后一个元素后一个位置的地址

`x` 所要查找的元素

```
it = find(vec.begin(), vec.end(), x);
if (it == vec.end()) {
    // 元素不存在
} else {
    // 元素相对于数组起始位置的偏移量为it - vec.begin()
    printf("%d\n", it - vec.begin()); // 输出元素x所在的下标
}
```

二分查找

只能在有序的数据结构中使用

```
int left = 0, right = arr.length - 1;
while (left <= right) {
    int mid = (left + right) / 2;
    if(arr[mid] == x) {
        // 查找成功
    } else if (arr[mid] < x) {
        left = mid + 1;
    } else if (arr[mid] > x) {
        right = mid - 1;
    }
}
// 查找失败
```

使用map取代二分查找

把所有查找的数据放到`map`里，map的底层是红黑树，查找是O(logn)，和二分查找一样。

如果放到`unordered_map`，它的底层是哈希表，查找是O(1)，代价是更多的额外空间。

```cpp
map<int, int> findA;
for(int i = 0; i < n; i++) {
    findA.insert(arr[i], i);
}

if(findA.find(b) == findA.end()) {
    // b在数组中不存在
} else {
    // b在数组中存在
    printf("%d\n", findA[b]);
}
```