排序

普通排序

P1177 【模板】排序

STL

```
#include<bits/stdc++.h>
using namespace std;
const int N=1e6+10;
int q[N];
int main(){
    int n;cin>>n;
    for(int i=0;i<n;i++){
        cin>>q[i];
    }
    sort(q,q+n);
    for(int i=0;i<n;i++){
        cout<<q[i]<<" ";
    }
    return 0;
}
```

冒泡排序

```
#include <bits/stdc++.h>
using namespace std;
long long n,a[100001];

int main(){
    cin>>n;
    for (int i=1;i<=n;i++) cin>>a[i];

    for (int i=1;i<=n;i++)
        for (int j=1;j<n;j++)
            if (a[j]>a[j+1]) swap(a[j],a[j+1]);

    for (int i=1;i<=n;i++) cout<<a[i]<<' ';
    return 0;
}
```

三路快排

```
#include<bits/stdc++.h>
using namespace std;
const int N=1e6+10;
int a[N],b[N],c[N],d[N];

void quick_sort(int l,int r) {
```

```
    if(l>=r) {
        return;
    }
    int num=(l+r)/2;
    int ind1=0,ind2=0,ind3=0;
    for(int i=l; i<=r; i++) {
        if(a[i]<a[num]) {
            b[ind1++]=a[i];
        }
        if(a[i]==a[num]) {
            c[ind2++]=a[i];
        }
        if(a[i]>a[num]) {
            d[ind3++]=a[i];
        }
    }

    for(int i=l; i<l+ind1; i++) {
        a[i]=b[i-l];
    }
    for(int i=l+ind1; i<l+ind1+ind2; i++) {
        a[i]=c[0];
    }
    for(int i=l+ind1+ind2; i<l+ind1+ind2+ind3; i++) {
        a[i]=d[i-l-ind1-ind2];
    }

    quick_sort(l,l+ind1-1);
    quick_sort(l+ind1+ind2,r);
}

int main() {
    int n;
    cin>>n;
    for(int i=1; i<=n; i++) {
        cin>>a[i];
    }
    quick_sort(1,n);
    for(int i=1; i<=n; i++) {
        cout<<a[i]<<" ";
    }
    return 0;
}
```

简易桶排序

```
#include<iostream>
using namespace std;
int n,b[100000005];
int main(){
  cin>>n;
  for(int i=1;i<=n;i++){
    int x;
    cin>>x;
    b[x]++;
  }
  for(int i=1;i<=100000000;i++){
```

```
    for(int j=1;j<=b[i];j++){
      cout<<i<<' ';
    }
  }
  return 0;
}
```

快速排序

```
#include<bits/stdc++.h>
using namespace std;

const int N=1e6+10;
int a[N];

void quick_sort(int a[],int l,int r) {
    int i=l,j=r,mid=a[(l+r)/2];
    //分成两部分，第一部分从前向后与中止对比
    //第二部分从后向前与中止进行对比
    do {
        while(a[i]<mid) {
            i++;
        }
        while(a[j]>mid) {
            j--;
        }
        //如果两个都跳出，且i<=j那么就将这两个数进行调换
        if(i<=j) {
            swap(a[i],a[j]);
            i++;
            j--;
        }
    } while(i<=j);
    if(l<j) {
        quick_sort(a,l,j);
    }
    if(i<r) {
        quick_sort(a,i,r);
    }
}

int main() {
    int n;
    cin>>n;
    for(int i=1; i<=n; i++) {
        cin>>a[i];
    }
    quick_sort(a,1,n);
    for(int i=1; i<=n; i++) {
        cout<<a[i]<<" ";
    }

    return 0;
}
```

归并排序

```cpp
#include<bits/stdc++.h>
using namespace std;

const int N=1e6+10;
int a[N],tmp[N];

void merge(int l,int mid,int r) {
    int k=l,i=l,j=mid+1;
    //两个部分 从头开始比较 谁小，谁在前
    while(i<=mid&&j<=r) {
        if(a[i]<=a[j]) {
            tmp[k++]=a[i++];
        } else {
            tmp[k++]=a[j++];
        }
    }
    //当其中一个部分被取完 另一部分直接向后依次取出
    while(i<=mid) {
        tmp[k++]=a[i++];
    }
    while(j<=r) {
        tmp[k++]=a[j++];
    }
    //将取出到tmp数组中的排好序的元素 赋值到原a数组中
    for(int i=l; i<=r; i++) {
        a[i]=tmp[i];
    }
}

void merge_sort(int l,int r) {
    if(l>=r) {
        return;
    }
    int mid=(l+r)/2;
    //切割 使其一分为二
    merge_sort(l,mid);
    merge_sort(mid+1,r);
    merge(l,mid,r);
}

int main() {
    int n;
    cin>>n;
    for(int i=1; i<=n; i++) {
        cin>>a[i];
    }
    merge_sort(1,n);

    for(int i=1; i<=n; i++) {
        cout<<a[i]<<" ";
    }

    return 0;
}
```

结构体排序

重载运算符排序

重载运算符 `<`:

- `bool operator<(const node &x) const`: 这个成员函数重载了小于运算符，使得可以使用 `<` 直接比较两个 `node` 对象。
- 函数内部的逻辑 `return x.score < score;` 表示比较时，分数大的对象会被认为是"更小"的。这是因为在排序时，通常希望分数高的排在前面。

```
struct node{
    string name;
    int score;
    //分数大的排前面
    bool operator<(const node &x) const{
        return x.score < score;
    }
};
```

通过重载运算符，可以利用标准库中的排序算法（如 `std::sort`）直接对 `node` 类型的数组或向量进行排序。高分的 `node` 会排在前面。

CMP函数实现

```
// 比较函数，返回 true 表示第一个参数在排序中排在第二个参数前面
bool cmp(const node &a, const node &b) {
    return a.score > b.score; // 分数大的排前面
}
```

使用一个比较函数 `cmp` 来替代重载运算符 `<`，这在某些情况下可能更方便，尤其是需要与 STL 算法一起使用时。下面是如何实现一个比较函数的示例：

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

struct node {
    string name;
    int score;
};

// 比较函数，返回 true 表示第一个参数在排序中排在第二个参数前面
bool cmp(const node &a, const node &b) {
    return a.score > b.score; // 分数大的排前面
}

int main() {
    vector<node> students = {{"Alice", 85}, {"Bob", 95}, {"Charlie", 80}};

    // 使用比较函数进行排序
    sort(students.begin(), students.end(), cmp);
```

```
    for (const auto &student : students) {
        cout << student.name << ": " << student.score << endl;
    }

    return 0;
}
```

优先队列排序

- 优先队列

优先队列和普通队列区别较大。优先队列，是一个可以**自动排序**的队列。也就是说，先进去的不一定先出来，而是通过比较大小来决定出队顺序的。

优先队列和堆是类似的，当然，你也可以把它理解为一个堆。

优先队列默认以`vector`的作为底层数据结构来实现，当然，也可以以`list`作为底层数据结构实现。

下面是优先队列的声明方式。

```
// priority_queue<数据类型> 变量名
// 认真研究过上面也应该明白是什么意思了

priority_queue<int> q; <=> priority_queue<int,vector<int>,less<int> > q;

// 上面两种声明方法是等价的。在定义时若不指明类型，默认为大根堆（降序排序）

priority_queue<int,vector<int>,greater<int> > q;

//这个是小根堆（升序排序）的声明方式
```

优先队列的成员函数：

```
priority_queue<int> q; //这里直接声明成小根堆了

q.top(); //取出队首元素，和普通队列不太一样

q.pop(); //清除队首元素

q.size(); //返回当前队列内元素个数（容器大小）

q.push(x); //向队列加入新的元素 x

q.empty(); //判断队列是否为空，空则返回1，否则返回0
```

优先队列不仅可以对普通的数据类型排序，还可以对你自定义的数据类型排序，只不过需要自己规定排序方式，方法如下。

```
struct cmp// 方法一：写一个比较函数，定义比较顺序。
{
    bool operator () (int x,int y)
   {
```

```
        return x<y;
    }
}

priority_queue<int,vector<int>,cmp> q;//和上面声明小根堆的方法是等价的，这里只是展示写比较函数的方法。



struct node{ //方法二：重载小于符号（可以理解为重新定义小于符号）
    int x,y,sum;
    bool operator<(const node &xx) const{
        return xx.sum<sum;  //对于每一个队内的node类型元素，谁的 sum 小谁在前面
    }
}t,tt;

priority_queue<node> q;
```

优先队列的常用成员函数：

```
priority_queue<int> q; //这里直接声明成小根堆了

q.top(); //取出队首元素，和普通队列不太一样

q.pop(); //清除队首元素

q.size(); //返回当前队列内元素个数（容器大小）

q.push(x); //向队列加入新的元素x

q.empty(); //判断队列是否为空，空则返回1，否则返回0
```

优先队列的增改复杂度为 O(logn)，删查为 O(1)