

1. 位运算

符号	运算
&	与
	或
~	非
^	异或
>>	右移
<<	左移

常用操作：

1. 求 x 的第 k 位数字 $x \gg k \& 1$
2. $\text{lowbit}(x) = x \& -x$ ，返回 x 的最后一位 1

2. 常用库函数

2.1 reverse 翻转

翻转一个 `vector`：

```
reverse(a.begin(), a.end());
```

翻转一个数组，元素存放在下标 $1 \sim n$ ：

```
reverse(a + 1, a + n + 1);
```

2.2 unique 去重

返回去重（只去掉相邻的相同元素）之后的尾迭代器（或指针），仍然为前闭后开，即这个迭代器是去重之后末尾元素的下一个位置。该函数常用于离散化，利用迭代器（或指针）的减法，可计算出去重后的元素个数。

把一个 `vector` 去重：

```
int m = unique(a.begin(), a.end()) - a.begin();
```

把一个数组去重，元素存放在下标 $1 \sim n$ ：

```
int m = unique(a + 1, a + n + 1) - (a + 1);
```

2.3 random_shuffle 随机打乱

用法与 `reverse` 相同。

2.4 sort

对两个迭代器（或指针）指定的部分进行快速排序。可以在第三个参数传入定义大小比较的函数，或者重载“小于号”运算符。

把一个 `int` 数组（元素存放在下标 `1 ~ n`）从大到小排序，传入比较函数：

```
int a[MAX_SIZE];
bool cmp(int a, int b)
{
    return a > b;
}
sort(a + 1, a + n + 1, cmp);
```

把自定义的结构体 `vector` 排序，重载“小于号”运算符：

```
struct Rec
{
    int id, x, y;
};

vector<Rec> a;

bool operator <(const Rec &a, const Rec &b)
{
    return a.x < b.x || a.x == b.x && a.y < b.y;
}

sort(a.begin(), a.end());
```

2.5 `lower_bound/upper_bound` 二分

`lower_bound` 的第三个参数传入一个元素 `x`，在两个迭代器（指针）指定的部分上执行二分查找，返回指向第一个大于等于 `x` 的元素的位置的迭代器（指针）。

`upper_bound` 的用法和 `lower_bound` 大致相同，唯一的区别是查找第一个大于 `x` 的元素。当然，两个迭代器（指针）指定的部分应该是提前排好序的。

在有序 `int` 数组（元素存放在下标 `1 ~ n`）中查找大于等于 `x` 的最小整数的下标：

```
int i = lower_bound(a + 1, a + 1 + n, x) - a;
```

在有序 `vector<int>` 中查找小于等于 `x` 的最大整数（假设一定存在）：

```
int y = *--upper_bound(a.begin(), a.end(), x);
```