基础算法

排序

快速排序

思想

1. 确定分界点
2. **调整区间**
3. 递归处理左右两段

代码

```
void quick_sort(int q[], int l, int r) {
    if (l >= r) return;

    int i = l - 1, j = r + 1, x = q[l + r >> 1]; // x 一定不能取 q[r]
    while(i < j) {
        do ++i; while(q[i] < x);
        do --j; while(q[j] > x);
        if (i < j) swap(q[i], q[j]);
    }

    quick_sort(q, l, j), quick_sort(q, j + 1, r); // j, j + 1
}
```

归并排序

思想

1. 确定分界点
2. 递归排序
3. **归并——合二为一**

代码

```
int tmp[MaxN];

void merge_sort(int q[], int l, int r) {
    if (l >= r) return;

    int mid = l + r >> 1;
    merge_sort(q, l, mid);
    merge_sort(q, mid + 1, r);

    int k = 0, i = l, j = mid + 1;
    while(i <= mid && j <= r) {
        if (q[i] <= q[j]) tmp[k++] = q[i++];
        else tmp[k++] = q[j++];
    }
```

```
    while(i <= mid) tmp[k++] = q[i++];
    while(j <= r) tmp[k++] =  q[j++];

    for (i = 1, j = 0; i <= r; i++, j++) q[i] = tmp[j];
}
```

sort

```
#include <algorithm>
void sort(RandomIt first, RandomIt last); // 默认升序
void sort(RandomIt first, RandomIt last, Compare comp);

bool compare(Student a, Student b) {
    return a.score > b.score;
}
```

二分

整数二分

思想

本质并非利用单调性，而是利用某位置的左侧不满足某一性质，而右侧都满足某一性质。

代码

```
bool check(int x) {/* ... */} // 检查 x 是否满足某种性质

// 区间 [l, r] 被划分成 [l, mid] 和 [mid + 1, r] 时使用，右半边最左侧
int bsearch_1(int l, int r) {
    while(l < r) {
        int mid = l + r >> 1;
        if (check(mid)) r = mid;
        else l = mid + 1;
    }
    return l;
}

// 区间 [l, r] 被划分成 [l, mid - 1] 和 [mid, r] 时使用，左半边最右侧
int bsearch_2(int l, int r) {
    while(l < r) {
        int mid = l + r + 1 >> 1;
        if (check(mid)) l = mid;
        else r = mid - 1;
    }
    return l;
}
```

浮点数二分

```
bool check(double x) {/* ... */} // 检查 x 是否满足某种性质

double bsearch_3(double l, double r) {
    const double eps = 1e-6; // eps 表示精度
    while (r - l > eps) {
        double mid = l + r >> 1;
        if (check(mid)) r = mid;
        else l = mid;
    }
    return l;
}
```

高精度

思想

- 存储：小端存储，方便进位

代码

高精度加法

```
// C = A + B, A >= 0, B >= 0
vector<int> add(vector<int> &A, vector<int> &B) {
    if (A.size() < B.size()) return add(B, A);

    vector<int> C;
    int t = 0;
    for (int i = 0; i < A.size(); i++) {
        t += A[i];
        if (i < B.size()) t += B[i];
        C.push_back(t % 10);
        t /= 10;
    }

    if (t) C.push_back(t);
    return C;
}
```

高精度减法

```
// C = A - B, A >= B, A >= 0, B >= 0
vector<int> sub(vector<int> &A, vector<int> &B) {
    vector<int> C;
    for (int i = 0, t = 0; i < A.size(); i++) {
        t = A[i] - t;
        if (i < B.size()) t -= B[i];
        C.push_back((t + 10) % 10);
        if (t < 0) t = 1;
        else t = 0;
    }
    while(C.size() > 1 && C.back() == 0) C.pop_back();
    return C;
}
```

高精度乘低精度

```
// C = A * b, A >= 0, b >= 0
vector<int> mul(vector<int> &A, int b) {
    vector<int> C;

    int t = 0;
    for (int i = 0; i < A.size() || t != 0; i++) {
        if (i < A.size()) t += A[i] * b;
        C.push_back(t % 10);
        t /= 10;
    }

    while (C.size() > 1 && C.back() == 0) C.pop_back();

    return C;
}
```

高精度除以低精度

```
// A /b = C ... r, A >= 0, b > 0
vector<int> div(vector<int> &A, int b, int &r) {
    vector<int> C;
    r = 0;
    for (int i = A.size() - 1; i >= 0; i--) {
        r = r * 10 + A[i];
        C.push_back(r / b);
        r %= b;
    }
    reverse(C.begin(), C.end());
    while (C.size() > 1 && C.back() == 0) C.pop_back();

    return C;
}
```

前缀和与差分

前缀和

一维前缀和

```
S[i] = a[1] + a[2] + ... + a[i];
a[l] + ... + a[r] = S[r] - S[l - 1];
```

二维前缀和

```
S[i, j] = 第i行第j列格子左上部分所有元素的和;

以(x1, y1)为左上角，(x2, y2)为右下角的子矩阵的和为:
    S[x2, y2] - S[x1 - 1, y2] - S[x2, y1 - 1] + S[x1 - 1, y1 - 1];
```

差分

前缀和的逆运算

一维差分

- 用 $O(1)$ 的时间给原数组某个区间的所有元素全部加上同一个值。
- 初始化：以数组区间长度为 1 进行插入。

```
给区间 [l, r] 中的每个数加上 c:
    B[l] += c, B[r + 1] -= c
```

二维差分

- 初始化：以 $1 * 1$ 矩阵进行插入。

```
给以 (x1, y1) 为左上角，(x2, y2) 为右下角的子矩阵中的所有元素加上 c:
    B[x1, y1] += c, B[x2 + 1, y1] -= c, B[x1, y2 + 1] -= c, B[x2, y2] += c
```

双指针算法

核心思想：将枚举数量由$O(n^2)$变为$O(n)$

分类：

- 对于一个序列，用两个指针维护一段区间
- 对于两个序列，维护某种持续，比如归并排序中合并两个有序序列的操作

伪代码：

```
for (int i = 0, j = 0; i < n; i++) {
    while (j < i && check(i, j)) j++;
    // 具体问题的逻辑
}
```

位运算

- n 的二进制表示中第 k 位数字：$n >> k \& 1$

- n 的最后一位1：$lowbit(n) = n \ \& \ -n$

离散化

```
vector<int> alls; // 存储所有待离散化的值
sort(alls.begin(), alls.end()); // 将所有值排序
alls.erase(unique(alls.befin(), alls.end()), alls.end()); // 去掉重复元素

// 二分求出 x 对应的离散化的值
int find(int x) // 找到第一个大于等于 x 的位置
{
    int l = 0, r = alls.size - 1;
    while(l < r) {
        int mid = (l + r) >> 1;
        if (alls[mid] >= x) r = mid;
        else l = mid + 1;
    }
    return r + 1; // 映射到 1, 2, ..., n
}
```

区间合并

思想：贪心策略

```
// 将所有存在交集的区间合并
void merge(vector<PII> &segs) {
    vector<PII> res;

    sort(segs.begin(), segs.end());

    int st = -2e9, ed = -2e9;
    for (auto seg : segs) {
        if (ed < seg.first) {
            if (st != -2e9) res.push_back({st, ed});
            st = seg.first, ed = seg.second;
        } else {
            ed = max(ed, seg.second);
        }
    }

    if (st != -2e9) res.push_back({st, ed});
    segs = res;
}
```