目录

[快速排序 2](#_Toc6749)

[归并排序 3](#_Toc11744)

[二分 4](#_Toc4865)

[▪ 整数二分 4](#_Toc7380)

[▪ 浮点数二分 6](#_Toc19237)

[高精度 6](#_Toc13954)

[前缀和与差分 6](#_Toc17515)

[双指针算法 6](#_Toc20960)

[位运算 6](#_Toc20066)

[离散化 6](#_Toc19180)

[▪ 区间合并 6](#_Toc7440)

# 快速排序

#### 分治思想，不稳定，平均时间复杂度 ，最坏情况下

### 确定分界点q[l], q[(l+r)/2], q[r]

### 调整区间 | <=x | >=x |

#### 双指针

| <=x | | >=x |

i j

#### 交换q[i],q[j]

### 递归左右两端

#### [785. 快速排序](https://www.acwing.com/problem/content/description/787/)

//快速排序算法

#include <iostream>

using namespace  std;

const int N = 1e6+9;

int n;

int q[N];

void quick\_sort(int q[],int l ,int r)

{

    if(l>=r) return ;

    int x = q[(l+r)>>1] ,i=l-1,j=r+1;

    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);

}

int main()

{

    scanf("%d",&n);

    for(int i=0;i<n;i++) scanf("%d",&q[i]);

    quick\_sort(q,0,n-1);

    for(int i=0;i<n;i++) printf("%d ",q[i]);

    return 0;

} *// namespace  std;*

#### 第k个数

# 归并排序

#### 分治思想，稳定，

### | left | right |

### 确定分界点 (l+r)/2

### 递归排序左右两端

### 归并--合二为一

#### 双指针

#### left ↓

#### i=l

#### right ↓ ----------

j=mid

#### 比较把较小的拿到tmp[]数组

Res：···· ----------

#### 787. 归并排序

#include <iostream>

using namespace  std;

const int N = 1e6+9;

int n;

int q[N],tmp[N];

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=l,j=0;i<=r;i++,j++) q[i]=tmp[j];

}

int main()

{

    scanf("%d",&n);

    for(int i=0;i<n;i++) scanf("%d",&q[i]);

    merge\_sort(q,0,n-1);

    for(int i=0;i<n;i++) printf("%d ",q[i]);

    return 0;

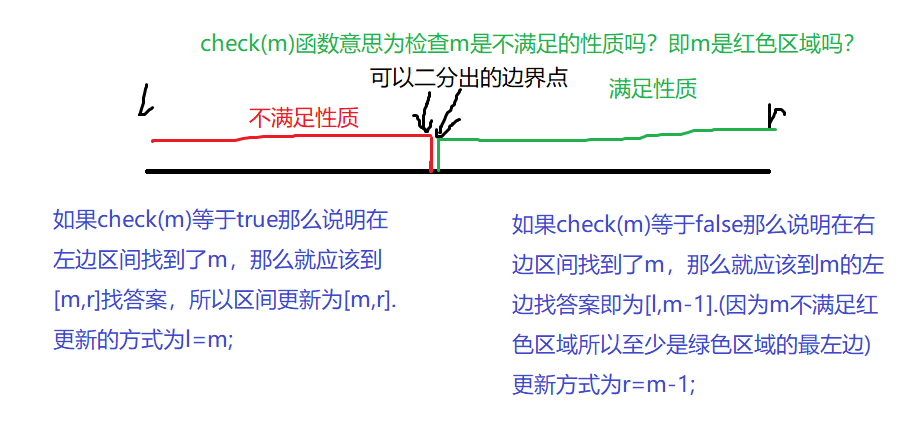
} *// namespace  std;*

# 二分

## 整数二分

|--------------------------||--------------------------|

L r



#### l=mid的情况，中间点的更新方式是m=(l+r+1)/2

#### r=mid的情况，中间点的更新方式是m=(l+r)/2

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：*

int bsearch\_1(int l, int r)

{

    while (l < r)

    {

        int mid = l + r >> 1;

        if (check(mid)) r = mid;    *// check()判断mid是否满足性质*

        else l = mid + 1;

    }

    return l;

}

*// 区间[l, r]被划分成[l, mid - 1]和[mx*

*id, r]时使用：*

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

double bsearch\_3(double l, double r)

{

const double eps

 = 1e-6;   *// eps 表示精度，取决于题目对精度的要求*

    while (r - l > eps)

    {

        double mid = (l + r) / 2;

        if (check(mid)) r = mid;

        else l = mid; //不用加一减一

    }

    return l;

}

# 高精度

#### 大数<10^6位，小数<10^9

#### A+B A-B A\*a A/a

### 大整数的存储，**逆序存储**，便于运算和存储

### 模拟全加器

bool cmp(vector<int> A,vector<int> B)

{

    if(A.size()>B.size) return A.size()>B.size();

    for(int i=0;i<A.size()-1;i++)

    {

        if(A[i]!=B[i]) return A[i]>B[i];

    }

    return true;

}

#### 1.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]; //t是进位

        if (i < B.size()) t += B[i]; //当B没有加完

        C.push\_back(t % 10);

        t /= 10;

    }

    if (t) C.push\_back(t); //最高位的进位

    return C;

}

#### 2.高精度减法

*// C = A - B, 满足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;

}

#### 3.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; 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;

}

#### 4.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

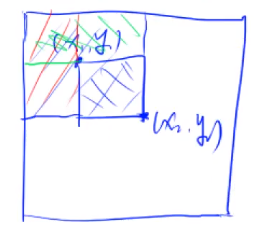
## 前缀和

#### 前缀和从1开始

### 定义一个sum[]数组，sum[i]代表a数组中前i个数的和

### sum[r]-sum[l-1]

二维前缀和：求子矩阵的和



S[i, j] = 第i行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

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

#### 差分数列得到原数列 for(int i=1;i<=n;i++) b[i]+=b[i-1];

### [(x1,y1),(x2,y2)]中的每个数加上c:

    b[x1][y1] += c;

    b[x2 + 1][y1] -= c;

    b[x1][y2 + 1] -= c;

    b[x2 + 1][y2 + 1] += c;

#### 差分矩阵得到原矩阵

    for (int i = 1; i <= n; i ++ )

        for (int j = 1; j <= m; j ++ )

**b[i][j] += b[i - 1][j] + b[i][j - 1] - b[i - 1][j - 1];**

# 双指针算法

for (int i = 0, j = 0; i < n; i ++ )

{

    while (j < i && check(i, j)) j ++ ;

*// 具体问题的逻辑*

}

常见问题分类：

    (1) 对于一个序列，用两个指针维护一段区间

    (2) 对于两个序列，维护某种次序，比如归并排序中合并两个有序序列的操作

# 位运算

#### 补码=反码+1

#### -x = ~x+1

int lowbit(int x) //返回最低位的1所对应的值

{

    return x&(-x);

}

# 离散化

#### 数据离散，数据的绝对关系不紧要

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.begin(), 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

*// 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;

}