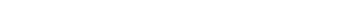


基础算法

快排不稳定: $\rightarrow \langle a_i, i \rangle$ 二元组, 保证值不同 \rightarrow 稳定

1. 快速排序

① 确定分界点 $x = q_L / q_{L+H/2}$

② 调整区间 

③ 递归处理左右两边

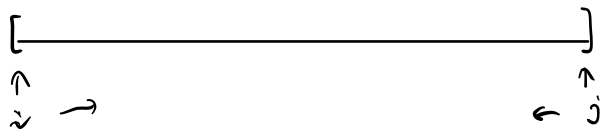
① 开两个额外数组 $a[]$ $b[]$

$$\textcircled{1} \quad q[i \sim r] < \begin{matrix} \overline{q[i]} \leq x & x \rightarrow a[i] \\ q[i] > x & x \rightarrow b[i] \end{matrix}$$

③ $a[] \rightarrow 9[]$ $b[] \rightarrow 9[]$

额外空间

优美写法:



同时向中间走，遇不满尺交换
直到相遇

$$\begin{array}{ccccc} 3 & 1 & 2 & 3 & 5 \\ \uparrow & & & & \uparrow \\ i & & & & j \end{array}$$
$$a_v = 3$$

$\left\{ \begin{array}{l} \text{左边的数永远小于等于} a \\ \text{右边的数永远大于等于} a \end{array} \right.$

⑦

3	1	2	3	5
↑			↑	
i	←	→	j	
swap				

② 3 1 2 3 5

交换后分别往中间移动1位

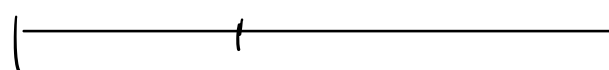
快排模板

```
void quit_sort(int q[], int l, int r){
    if (l >= r) return;
    int i = l-1, j = r+1, x = q[(l+r-1)/2];
    while(i < j){
        do i++; while(q[i] < x);
        do j--; while(q[j] > x);
        if(i < j) swap(q[i], q[j]);
    }
    quit_sort(q, l, j);
    quit_sort(q, j+1, r);
}
```

```
void quit_sort(int q[], int l, int r){
    if (l >= r) return;
    int i = l-1, j = r+1, x = q[(l+r+1)/2];
    while(i < j){
        do i++; while(q[i] < x);
        do j--; while(q[j] > x);
        if(i < j) swap(q[i], q[j]);
    }
    quit_sort(q, l, i-1);
    quit_sort(q, i, r);
}
```

注意边界条件，避免进入死循环

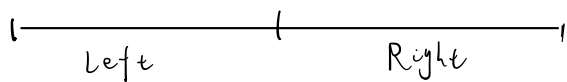
快排平均时间复杂度



划分期望为 $\frac{n}{2}$
期望 $\log_2 n$ 层

→ $n \log_2 n$

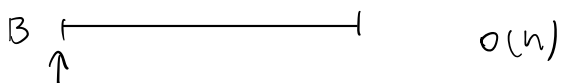
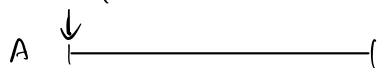
2. 归并排序 (分治) 归并排序 稳定



① 确定分界点 mid (位置)

② 递归排 $left \dots right$

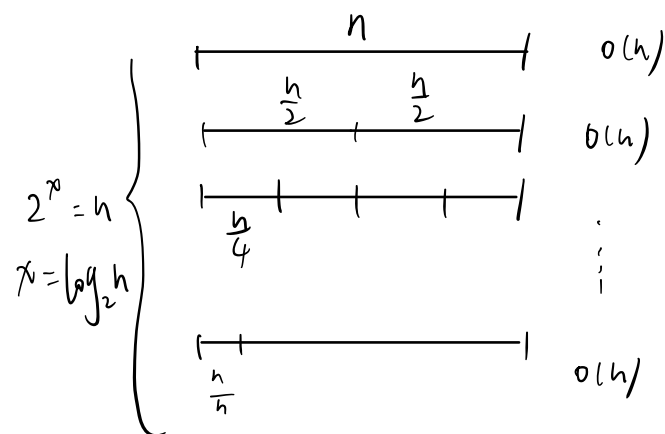
★ ③ 归并 \rightarrow 合二为一 (双指针算法)



res[]

直到一个指针到终点

归并排序 时间复杂度



$\Rightarrow n \log_2 n$ 时间复杂度

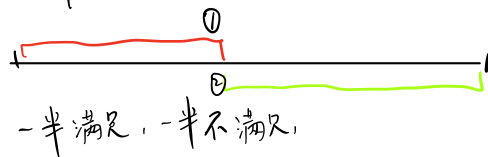
模板:

```
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 i = l,j=mid+1,k=0;
    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];
}
```

整数二分 (如果有单调, 则一定可以二分, 但二分并不一定要单调)

二分的本质:

某种性质

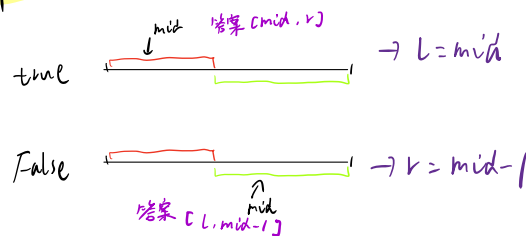


则二分可寻找红/绿的边界

两个模板

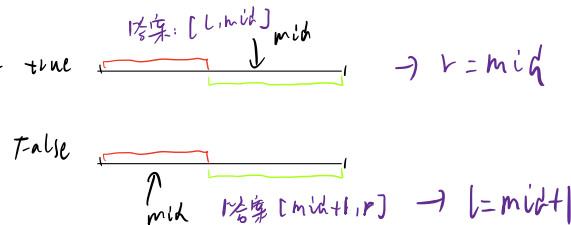
① 找红色边界

$mid = (l+r+1)/2$
if (check(mid))



② 找绿色边界

$mid = (l+r)/2$
if (check(mid))



① 为何补上 $(l+r+1)/2 + 1$

例 $l = r - 1$

如果不补上 +1

$$\Rightarrow mid = \frac{l+l+1}{2} = l = mid$$

死循环

如果补上 +1

则 $mid = r \rightarrow l = r \rightarrow$ 结束
返回 r

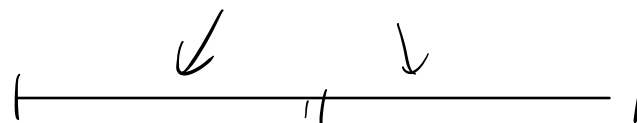
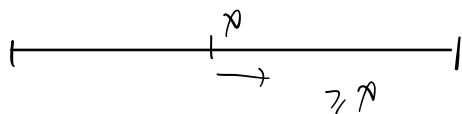
② 如何选用模板?

$l = mid$ 时 \rightarrow 补上 1

	1	2	2	3	3	4
idx	0	1	2	3	4	5

求 3 的起始坐标与终止坐标

如何 check



[每次选择答案所在区间进行处理]

$\log_2 n$ 复杂度

[浮点数 = 分不需要处理边界]

可严格二分 \rightarrow 终止条件 $r - l < \epsilon$

牛顿迭代法

789. 数的范围

```
#include <iostream>
#include <cstring>
#include <algorithm>
using namespace std;

const int N = 100010;
int n,m,k;
int q[N];

int main()
{
    scanf("%d%d", &n, &m);
    for (int i = 0; i < n; i ++ ) scanf("%d", &q[i]);
    while(m--)
    {
        scanf("%d", &k);

        int l=0,r=n-1;
        while(l<r)
        {
            int mid = l+r>>1;
            if(q[mid] < k) l = mid+1;
            else r = mid;
        }
        if(q[r] != k) {
            cout<<"-1 -1"<<endl;
            continue;}
        cout<<r<<" ";

        l = 0,r=n-1;
        while(l<r)
        {
            int mid = l+r+1>>1;
            if (q[mid] > k ) r = mid -1;
            else l = mid;
        }
        cout<<l<<endl;
    }
    return 0;
}
```