


## Lesson1

排序

- 快速排序

快速排序 —— 分治



① 确定分界点:  $q[l]$ ,  $q[(l+r)/2]$ ,  $q[r]$ , 随机

② 调整区间:  $\leq x$   $\geq x$  ☆

③ 递归处理左右两段

快速排序 —— 分治




① 确定分界点:  $q[l]$ ,  $q[(l+r)/2]$ ,  $q[r]$ , 随机

② 调整区间:  $\leq x$   $\geq x$  ☆

③ 递归处理左右两段

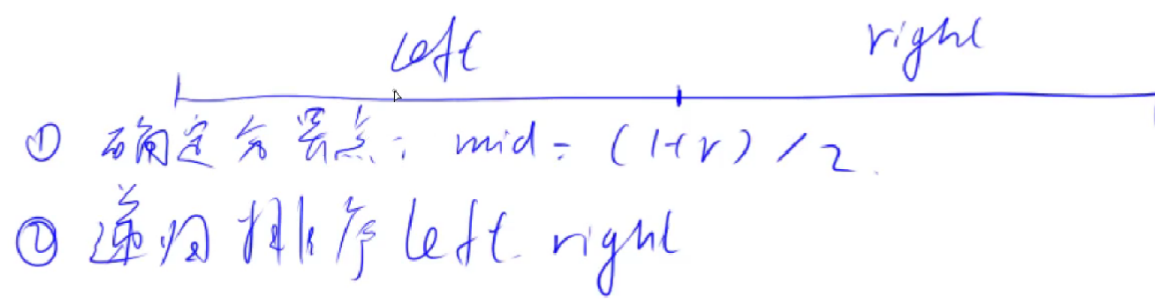
3 1 2 3 5  $x=3$



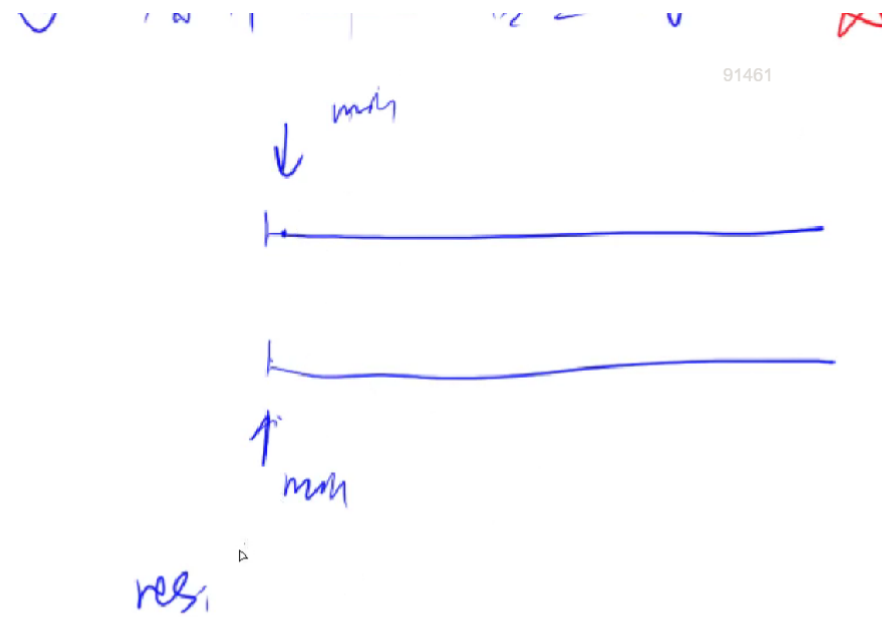
```
1 int n, q[N];
2
3 inline void quick_sort(int *q, int l, int r) {
4     if (l >= r) return;
5
6     int x = q[l+r>>1], i = l - 1, j = r + 1;
7     while (i < j) {
8         do i++; while (q[i] < x);
9         do j--; while (q[j] > x);
10        if (i < j) swap(q[i], q[j]);
11    }
12
13    quick_sort(q, l, j), quick_sort(q, j+1, r);
14 }
```

- 归并排序

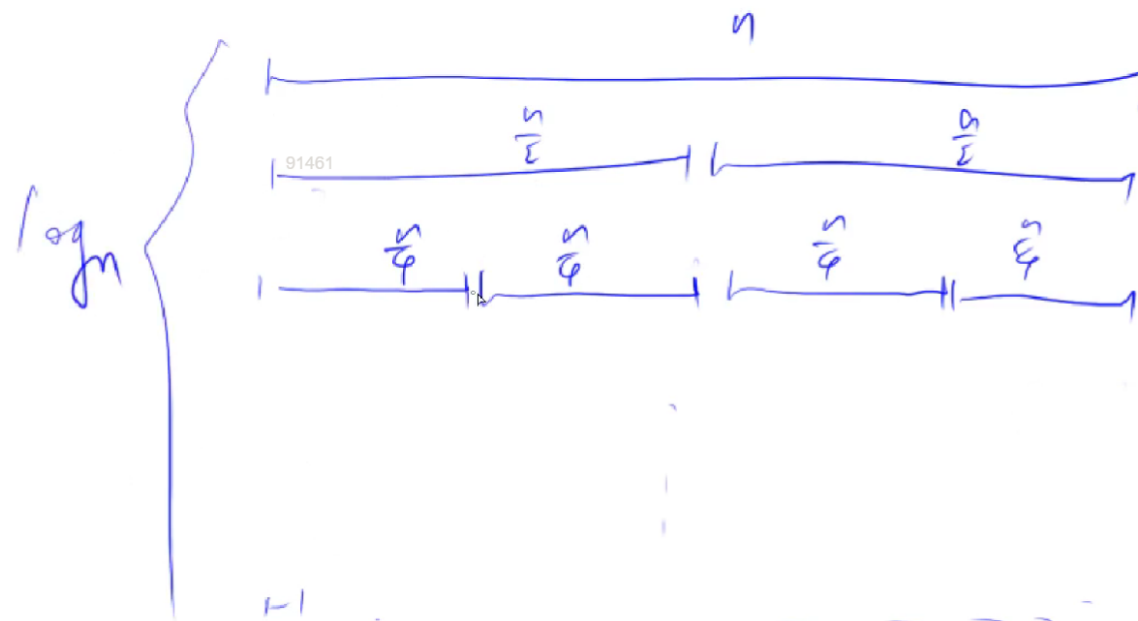
算法2: 归并排序 —— 分治



③ 归并 —— 合二为一 ☆



③ 归并 —— 合二为一



```

1 int n, q[N], tmp[N];
2
3 inline void merge_sort(int *q, int l, int r) {
4     if (l >= r) return;
5
6     int mid = l + r >> 1;
7     merge_sort(q, l, mid), merge_sort(q, mid+1, r);
8
9     int k = 0, i = l, j = mid+1;
10    while (i <= mid && j <= r) {
11        if (q[i] <= q[j]) tmp[k++] = q[i++];
12        else tmp[k++] = q[j++];
13    }
14
15    while (i <= mid) tmp[k++] = q[i++];
16    while (j <= r) tmp[k++] = q[j++];

```

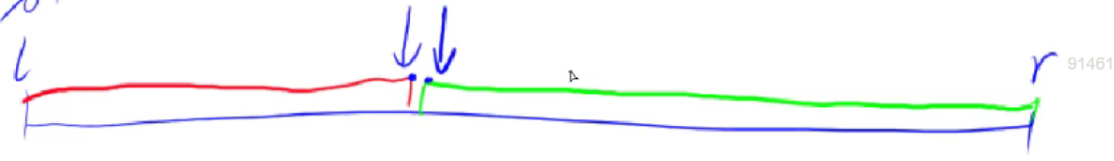
```

17
18     for (int i=l, j=0; i<=r; i++, j++) q[i] = tmp[j];
19 }

```

二分

整数 = 分



- 有单调性一定可以二分，可二分不一定需要有单调性

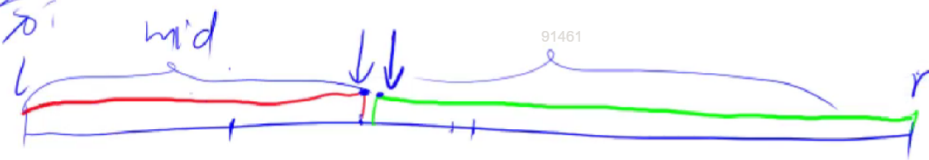
整数 = 分

~~二分~~



- 二分左区间中答案

整数 = 分



$$\textcircled{1} \quad mid = \frac{l+r+1}{2}$$

if (check(mid)) {  
     true [mid, r]: l = mid;  
     false [l, mid-1]: r = mid-1;  
}

```

1 while (l < r) {
2     mid = l + r + 1 >> 1;
3     if (check(mid))
4         true: ans in [mid, r], 更新方式 l = mid
5         false: ans in [l, mid-1], 更新方式 r = mid - 1;
6 }

```

- 为何 +1?

防止死循环

例如:  $l = r - 1$ , check为true时发生死循环

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

if (check(mid)) {  
     true [mid, r]: l = mid = l  
     false [l, mid-1]: r = mid-1  
}

- 二分右区间中答案

Handwritten diagram illustrating binary search on a range [l, r]. The range is divided into two halves, each labeled 'mid'. The search process is shown with arrows indicating the update of l and r.

①  $mid = \frac{l+r+1}{2}$   
 if (check(mid)) { true [mid, r]: l = mid;  
 false [l, mid-1]: r = mid-1

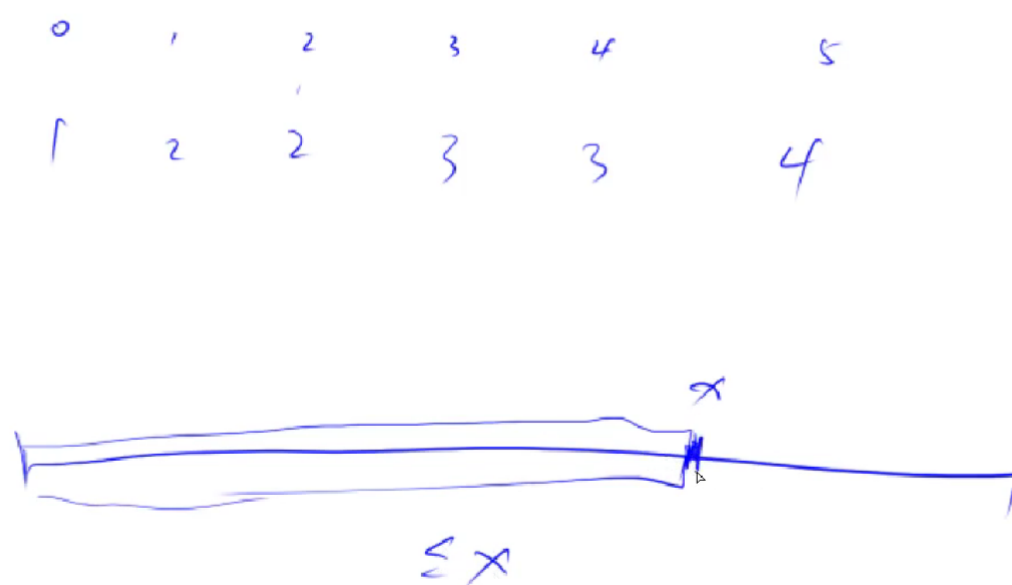
②  $mid = \frac{l+r}{2}$   
 if (check(mid)) { true [l, mid]: r = mid;  
 false [mid+1, r]: l = mid+1

```

1 while (l < r) {
2     mid = l + r >> 1;
3     if (check(mid))
4         true: ans in [l, mid], 更新方式 r = mid
5         false: ans in [mid+1, r], 更新方式 l = mid + 1;
6 }

```

• eg



```

1 #include <iostream>
2
3 using namespace std;
4
5 const int N = 1e5+10;
6
7 int n, m;
8 int q[N];
9
10 int main(void) {
11     scanf("%d%d", &n, &m);
12
13     for (int i=0; i<n; i++) scanf("%d", &q[i]);
14
15     while (m--) {
16         int x; scanf("%d", &x);
17
18         int l = 0, r = n - 1;
19         while (l < r) {
20             int mid = l + r >> 1;
21             if (q[mid] >= x) r = mid;
22             else l = mid + 1;
23         }
24
25         if (q[l] != x) cout << "-1 -1" << endl;
26         else {
27             cout << l << " ";

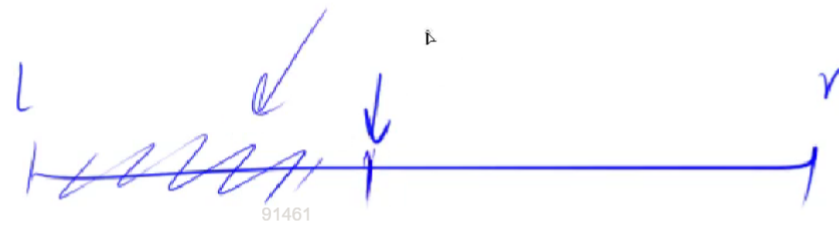
```

```

28
29     l = 0, r = n - 1;
30     while (l < r) {
31         int mid = l + r + 1 >> 1;
32         if (q[mid] <= x) l = mid;
33         else r = mid - 1;
34     }
35
36     cout << l << endl;
37 }
38 }
39
40 return 0;
41 }

```

- 浮点数二分



$$r - l \leq 10^{-6}$$

- ```

1  #求数的三次方根
2  #include <iostream>
3
4  using namespace std;
5
6  int main(void) {
7      double x;
8      scanf("%lf", &x);
9
10     double l = -100, r = 100;
11
12     while (r - l > 1e-8) {
13         double mid = (l + r) / 2;
14         if (mid * mid * mid >= x) r = mid;
15         else l = mid;
16     }
17
18     printf("%.6lf", l);
19
20     return 0;
21 }

```