Lesson1

> 排序

- 快速排序

-
  ```
  static int n;
  static int[] q = new int[N];

  static void quickSort(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) {
              int t = q[i]; q[i] = q[j]; q[j] = t;
          }
      }

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

- 第k个数

-
  ```
  static int n, k;
  static int[] q = new int[N];

  static int quickSort(int[] q, int l, int r, int k) {
      if (l >= r) return q[l];

      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) {
              int t = q[i]; q[i] = q[j]; q[j] = t;
          }
      }

      if (k-(j-l+1) > 0) return quickSort(q, j+1, r, k-(j-l+1));
      else return quickSort(q, l, j, k);
  }
  ```

- 归并排序

-
  ```
  static int n;
  static int[] q = new int[N], tmp = new int[N];

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

      int mid = l+r>>1, i = l, j = mid+1, k = 0;
      mergeSort(q, l, mid); mergeSort(q, mid+1, r);

      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];
  }
  ```

- 逆序对的数量(考虑三种情况加和)

-
  ```
  static int n;
  static int[] q = new int[N], tmp = new int[N];

  //求逆序对的数量
  static long mergeSort(int[] q, int l, int r) {
  ```

```
    if (l >= r) return 0;

    int mid = l+r>>1, i = l, j = mid+1, k = 0;
    long res = mergeSort(q, l, mid) + mergeSort(q, mid+1, r);

    while (i <= mid && j <= r) {
        if (q[i] <= q[j]) tmp[k++] = q[i++];
        else {
            res += mid-i+1;
            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];

    return res;
}
```

二分

- 有单调性一定可以二分，可二分不一定需要有单调性
- 二分左区间中答案

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

  - 为何 +1 ?

    防止死循环

    例如：l = r - 1，check为true时发生死循环

- 二分右区间中答案

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

- eg
-

```
static int N = 100010;

static int n, m;
static int[] q = new int[N];

public static void main(String[] args) throws Exception {
    ins.nextToken(); n = (int)ins.nval;
    ins.nextToken(); m = (int)ins.nval;

    for (int i=0; i<n; i++) { ins.nextToken(); q[i] = (int)ins.nval; }

    while (m-- > 0) {
        ins.nextToken(); int x = (int)ins.nval;

        int l = 0, r = n-1;
        while (l < r) {     //二分右区间中答案
            int mid = l+r>>1;
            if (q[mid] >= x) r = mid;
            else l = mid+1;
        }

        if (q[l] != x) out.println("-1 -1");
        else {
            out.print(l+" ");

            l = 0; r = n-1;
            while (l < r) {
                int mid = l+r+1>>1;
                if (q[mid] <= x) l = mid;   //和单调性相反，二分左区间中答案
                else r = mid-1;
            }

            out.println(r);
        }
    }

    out.flush();
}
```

- 浮点数二分
-
  ```
  //求数的三次方根
  static double eps = 1e-8;

  public static void main(String[] args) throws Exception {
      double x = in.nextDouble();

      double l = -100, r = 100;
      while (r - l > eps) {
          double mid = (l+r)/2;
          if (mid*mid*mid >= x) r=mid;
          else l = mid;
      }

      out.printf("%.6f", l);

      out.flush();
  }
  ```