

1：快速排序：

思想：分治

流程：

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

2)：调整区间为，分界点左边的数都 $\leq x$ ，分界点右边的数都 $\geq x$

3)：递归处理左右两段

算法心得1：有关边界问题要背一个模板

模板：

```
1 import java.util.Scanner;
2 public class Main{
3     private static void quicksort(int a[], int l, int r){
4         if(l >= r){
5             return;
6         }
7         int x = a[(l + r) / 2], i = l - 1, j = r + 1;
8         while(i < j){
9             do i++; while(a[i] > x);
10            do j--; while(a[j] < x);
11            if(i < j){
12                int t = a[i];
13                a[i] = a[j];
14                a[j] = t;
15            }
16        }
17        quicksort(a, l, j);
18        quicksort(a, j + 1, r);
19    }
20    public static void main(String[] args){
21        Scanner sc = new Scanner(System.in);
22        int n = sc.nextInt();
23        int[] a = new int[n];
24        for(int i = 0; i < n; i++){
25            a[i] = sc.nextInt();
26        }
27        quicksort(a, 0, n - 1);
28        for(int i = 0; i < n; i++){
```

```

29         System.out.printf("%d ", a[i]);
30     }
31 }
32 }

```

2：归并排序：

思想：分治

流程：

1)：确定分界点： $mid = (l + r) / 2$

2)：递归排序左右部分

3)：归并：合二为一

模板：

```

1 import java.util.Scanner;
2 public class Main{
3     private static void merge_sort(int a[], int l, int r){
4         if(l >= r){
5             return;
6         }
7         int mid = (l + r) / 2;
8         merge_sort(a, l, mid);
9         merge_sort(a, mid + 1, r);
10        int k = 0, i = l, j = mid + 1;
11        int[] temp = new int[r - l + 1];
12        while(i <= mid && j <= r){
13            if(a[i] <= a[j]){
14                temp[k++] = a[i++];
15            }else{
16                temp[k++] = a[j++];
17            }
18        }
19        while(i <= mid){
20            temp[k++] = a[i++];
21        }
22        while(j <= r){
23            temp[k++] = a[j++];
24        }
25        for(int m = l, n = 0; m <= r; m++, n++){

```

```

26         a[m] = temp[n];
27     }
28 }
29 public static void main(String[] args){
30     Scanner sc = new Scanner(System.in);
31     int n = sc.nextInt();
32     int[] a = new int[n];
33     for(int i = 0; i < n; i++){
34         a[i] = sc.nextInt();
35     }
36     merge_sort(a, 0, n - 1);
37     for(int i = 0; i < n; i++){
38         System.out.printf("%d ", a[i]);
39     }
40 }
41 }

```

3: 二分法:

二分的本质: 找到一个性质, 可以将数据一分为二, 一边满足一边不满足

二分每次都会保证新的区间中一定会包含答案 (处理边界问题)

模板:

```

1  boolean check(int x){
2
3  }
4  int bsearch_1(int l, int r){
5      while(l < r){
6          int mid = (l + r) / 2;
7          if(check(mid)){
8              r = mid;
9          }else{
10             l = mid + 1;
11         }
12     }
13     return l;
14 }
15

```

4: 高精度加法:

模板:

```
1  import java.util.Scanner;
2  import java.util.List;
3  import java.util.ArrayList;
4  public class Main{
5      private static List<Integer> add(List<Integer> A, List<Integer> B){
6          List<Integer> C = new ArrayList<>();
7          int t = 0;
8          for(int i = 0; i < A.size() || i < B.size(); i++){
9              if(i < A.size()){
10                 t += A.get(i);
11             }
12             if(i < B.size()){
13                 t += B.get(i);
14             }
15             C.add(t % 10);
16             t /= 10;
17         }
18         if(t != 0){
19             C.add(1);
20         }
21         return C;
22     }
23     public static void main(String[] args){
24         Scanner sc = new Scanner(System.in);
25         String a = sc.next();
26         String b = sc.next();
27         List<Integer> A = new ArrayList<>(a.length());
28         List<Integer> B = new ArrayList<>(b.length());
29         for(int i = a.length() - 1; i >= 0; i--){
30             A.add(a.charAt(i) - '0');
31         }
32         for(int i = b.length() - 1; i >= 0; i--){
33             B.add(b.charAt(i) - '0');
34         }
35         List<Integer> C = add(A, B);
36         for(int i = C.size() - 1; i >= 0; i--){
37             System.out.printf("%d", C.get(i));
```

```

38         }
39     }
40 }

```

5: 前缀和: (作用: 可以快速求出来原数组某一区间的和, 数组下标从1开始)

给定一个原数组, $a_1, a_2, a_3, \dots, a_n$

前缀和 $S_i = a_1 + a_2 + \dots + a_i$, $S_0 = 0$;

模板:

```

1  import java.util.Scanner;
2  public class Main{
3      public static void main(String[] args){
4          Scanner sc = new Scanner(System.in);
5          int n = sc.nextInt();
6          int m = sc.nextInt();
7          int[] a = new int[n + 1];
8          int[] s = new int[n + 1];
9          for(int i = 1; i <= n; i++){
10             a[i] = sc.nextInt();
11         }
12         for(int i = 1; i <= n; i++){
13             s[i] = s[i - 1] + a[i];
14         }
15         while(m -- > 0){
16             int l = sc.nextInt();
17             int r = sc.nextInt();
18             System.out.println(s[r] - s[l - 1]);
19         }
20     }
21 }

```

前缀和可以扩展到二维子矩阵求前缀和:

二维前缀和推导:

如图:

紫色面积是指 $(1, 1)$ 左上角到 $(i, j - 1)$ 右下角的矩形面积, 绿色面积是指 $(1, 1)$ 左上角到 $(i - 1, j)$ 右下角的矩形面积, 每一个颜色的矩形面积都代表了它所包围元素的和

从图中我们很容易看出，整个外围蓝色矩形面积 $s[i][j]$ = 绿色面积 $s[i-1][j]$ + 紫色面积 $s[i][j-1]$ - 重复加的红色面积 $s[i-1][j-1]$ + 小方块面积 $a[i][j]$

因此得出二维前缀和预处理公式：

$$s[i][j] = s[i-1][j] + s[i][j-1] - s[i-1][j-1] + a[i][j]$$

接下来去求以 $(x1, y1)$ 为左上角和以 $(x2, y2)$ 为右下角的矩阵的元素的和

如图：

紫色面积是指 $(1, 1)$ 左上角到 $(x1-1, y2)$ 右下角的矩形面积，黄色面积是指 $(1, 1)$ 左上角到 $(x2, y1-1)$ 右下角的矩形面积

不难找出：

绿色矩形的面积 = 整个外围面积 $s[x2, y2]$ - 黄色面积 $s[x2, y1-1]$ - 紫色面积 $s[x1-1, y2]$ + 重复减去的红色面积 $s[x1-1, y1-1]$

因此二维前缀和的结论为：

以 $(x1, y1)$ 为左上角， $(x2, y2)$ 为右下角的子矩阵的和为：

$$s[x2, y2] - s[x2, y1-1] - s[x1-1, y2] + s[x1-1, y1-1]$$

模板：

```
1 import java.util.Scanner;
2 public class Main{
3     public static void main(String[] args){
4         Scanner sc = new Scanner(System.in);
5         int n = sc.nextInt();
6         int m = sc.nextInt();
7         int q = sc.nextInt();
8         int[][] a = new int[n+1][m+1];
9         int[][] s = new int[n+1][m+1];
10        for(int i = 1; i <= n; i++){
11            for(int j = 1; j <= m; j++){
12                a[i][j] = sc.nextInt();
13            }
14        }
15        for(int i = 1; i <= n; i++){
16            for(int j = 1; j <= m; j++){
17                s[i][j] = s[i-1][j] + s[i][j-1] - s[i-1][j-1] + a[i][j];
18            }
19        }
20    }
21 }
```

```

19     }
20     while(q-- > 0){
21         int x1 = sc.nextInt();
22         int y1 = sc.nextInt();
23         int x2 = sc.nextInt();
24         int y2 = sc.nextInt();
25         System.out.println(s[x2][y2] - s[x1 - 1][y2] - s[x2][y1 - 1] + s[x1 - 1][y1
-1]);
26     }
27 }
28 }

```

6: 差分:

类似于数学中的求导和积分，可以看成前缀和的逆运算

差分数组:

首先给定一个原数组 $a: a[1], a[2], a[3], \dots, a[n]$; 然后构造一个数组 $b: b[1], b[2], b[3], \dots, b[i]$; 使得: $a[i] = b[1] + b[2] + b[3] + \dots + b[i]$

也就是说, a 数组是 b 数组的前缀和数组, 反过来把 b 数组叫做 a 数组的差分数组。换句话说, 每一个 $a[i]$ 都是 b 数组中从头开始的一段区间和

给定区间 $[l, r]$, 若想把 a 数组中的 $[l, r]$ 区间中的每一个数都加上 c , 即 $a[l] + c, a[l+1] + c, a[l+2] + c, \dots, a[r] + c$; 首先让差分 b 数组中的 $b[l] + c$, a 数组变成 $a[l] + c, a[l+1] + c, \dots, a[n] + c$; 然后打个补丁, $b[r+1] - c$, a 数组变成 $a[r+1] - c, a[r+2] - c, \dots, a[n] - c$

$b[l] + c$, 效果使得 a 数组中 $a[l]$ 及以后的数都加上了 c (红色部分), 但我们只要求 l 到 r 区间加上 c , 因此还需要执行 $b[r+1] - c$, 让 a 数组中 $a[r+1]$ 及往后的区间再减去 c (绿色部分), 这样对于 $a[r]$ 以后区间的数相当于没有发生改变

因此可以得出一维差分结论: 若给 a 数组中的 $[l, r]$ 区间中的每一个数都加上 c , 只需对差分数组 b 做 $b[l] += c, b[r+1] -= c$

模板:

```

1  import java.util.Scanner;
2  public class Main{
3      private static void insert(int[] b, int l, int r, int c){
4          b[l] += c;
5          b[r + 1] -= c;
6      }
7      public static void main(String[] args){

```

```

8         Scanner sc = new Scanner(System.in);
9         int n = sc.nextInt();
10        int m = sc.nextInt();
11        int[] a = new int[100010];
12        int[] b = new int[100010];
13        for(int i = 1; i <= n; i++){
14            a[i] = sc.nextInt();
15            insert(b, i, i, a[i]);
16        }
17        while(m -- > 0){
18            int l = sc.nextInt();
19            int r = sc.nextInt();
20            int c = sc.nextInt();
21            insert(b, l, r, c);
22        }
23        for(int i = 1; i <= n; i++){
24            b[i] += b[i - 1];
25            System.out.printf("%d ", b[i]);
26        }
27    }
28 }

```

差分数组可以扩展到二维差分， $a[][]$ 数组是 $b[][]$ 数组的前缀和数组，那么 $b[][]$ 是 $a[][]$ 的差分数组

构造二维差分数组目的是为了原二维数组 a 中所选中子矩阵中的每一个元素加上 c 的操作，已知原数组 a 中被选中的子矩阵为以 $(x1, y1)$ 为左上角，以 $(x2, y2)$ 为右下角所围成的矩形区域， a 数组是 b 数组的前缀和数组，对 b 数组的 $b[i][j]$ 的修改，会影响到 a 数组中从 $a[i][j]$ 及往后的每一个数

假定我们已经构造好了 b 数组，类比一维差分，执行以下操作来使被选中的子矩阵中的每个元素的值加上 c

$b[x1][y1] += c, b[x1][y2+1] -= c, b[x2+1][y1] -= c, b[x2+1][y2+1] += c$

画图来理解一下操作过程：

$b[x1][y1] += c$; 对应图1 , 让整个 a 数组中蓝色矩形面积的元素都加上了 c

$b[x1][y2+1] -= c$; 对应图2 , 让整个 a 数组中绿色矩形面积的元素再减去 c , 使其内元素不发生改变

$b[x2+1][y1] -= c$; 对应图3 , 让整个 a 数组中紫色矩形面积的元素再减去 c , 使其内元素不发生改变

$b[x2+1][y2+1] += c$; 对应图4,,让整个a数组中红色矩形面积的元素再加上c, 红色内的相当于被减了两次, 再加上一次c, 才能使其恢复

可以先假想a数组为空, 那么b数组一开始也为空, 但是实际上a数组并不为空, 因此我们每次让以(i, j)为左上角到以(i, j)为右下角面积内元素(其实就是一个小方格的面积)去插入 $c=a[i][j]$, 等价于原数组a中(i, j) 到(i, j)范围内 加上了 $a[i][j]$, 因此执行n*m次插入操作, 就成功构建了差分b数组

模板:

```
1 import java.util.Scanner;
2 public class Main{
3     private static void insert(int[][] b, int x1 ,int y1, int x2, int y2, int c){
4         b[x1][y1] += c;
5         b[x1][y2+1] -= c;
6         b[x2+1][y1] -= c;
7         b[x2+1][y2+1] += c;
8     }
9     public static void main(String[] args){
10         Scanner sc = new Scanner(System.in);
11         int n = sc.nextInt();
12         int m = sc.nextInt();
13         int q = sc.nextInt();
14         int[][] a = new int[1010][1010];
15         int[][] b = new int[1010][1010];
16         for(int i = 1 ; i <= n ; i ++ ){
17             for(int j = 1 ; j <= m ; j ++ ){
18                 a[i][j] = sc.nextInt();
19                 insert(b, i, j, i, j, a[i][j]);
20             }
21         }
22         while(q-- > 0){
23             int x1 = sc.nextInt();
24             int y1 = sc.nextInt();
25             int x2 = sc.nextInt();
26             int y2 = sc.nextInt();
27             int c = sc.nextInt();
28             insert(b, x1, y1, x2, y2, c);
29         }
```

```
30     for(int i = 1 ; i <= n ; i ++ ){
31         for(int j = 1 ; j <= m ; j ++ ){
32             a[i][j] = a[i-1][j] + a[i][j-1] - a[i-1][j-1] + b[i][j];
33             System.out.print(a[i][j] + " ");
34         }
35         System.out.println();
36     }
37 }
38 }
```