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 <= x，分界点右边的数都 >= 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：

```
import java.util.Scanner;
public class Main{
    private static void quicksort(int a[], int l, int r){
        if(l >= r){
            return;
        }
        int x = a[(l + r) / 2], i = l - 1, j = r + 1;
        while(i < j){
            do i++; while(a[i] > x);
            do j--; while(a[j] < x);
            if(i < j){
                int t = a[i];
                a[i] = a[j];
                a[j] = t;
            }
        }
        quicksort(a, l, j);
        quicksort(a, j + 1, r);
    }
    public static void main(String[] args){
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int[] a = new int[n];
        for(int i = 0; i < n; i++){
            a[i] = sc.nextInt();
        }
        quicksort(a, 0, n - 1);
        for(int i = 0; i < n; i++){
```

```
            System.out.printf("%d ", a[i]);
        }
    }
}
```

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：

```
import java.util.Scanner;
public class Main{
    private static void merge_sort(int a[], int l, int r){
        if(l >= r){
            return;
        }
        int mid = (l + r) / 2;
        merge_sort(a, l, mid);
        merge_sort(a, mid + 1, r);
        int k = 0, i = l, j = mid + 1;
        int[] temp = new int[r - l + 1];
        while(i <= mid && j <= r){
            if(a[i] <= a[j]){
                temp[k++] = a[i++];
            }else{
                temp[k++] = a[j++];
            }
        }
        while(i <= mid){
            temp[k++] = a[i++];
        }
        while(j <= r){
            temp[k++] = a[j++];
        }
        for(int m = l, n = 0; m <= r; m++, n++){
```

```
            a[m] = temp[n];
        }
    }
    public static void main(String[] args){
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int[] a = new int[n];
        for(int i = 0; i < n; i++){
            a[i] = sc.nextInt();
        }
        merge_sort(a, 0, n - 1);
        for(int i = 0; i < n; i++){
            System.out.printf("%d ", a[i]);
        }
    }
}
```

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：

```
boolean check(int x){

}
int bsearch_1(int l, int r){
    while(l < r){
        int mid = (l + r) / 2;
        if(check(mid)){
            r = mid;
        }else{
            l = mid + 1;
        }
    }
    return l;
}

```

4：高精度加法：

模板：

```java
import java.util.Scanner;
import java.util.List;
import java.util.ArrayList;
public class Main{
    private static List<Integer> add(List<Integer> A, List<Integer> B){
        List<Integer> C = new ArrayList<>();
        int t = 0;
        for(int i = 0; i < A.size() || i < B.size(); i++){
            if(i < A.size()){
                t += A.get(i);
            }
            if(i < B.size()){
                t += B.get(i);
            }
            C.add(t % 10);
            t /= 10;
        }
        if(t != 0){
            C.add(1);
        }
        return C;
    }
    public static void main(String[] args){
        Scanner sc = new Scanner(System.in);
        String a = sc.next();
        String b = sc.next();
        List<Integer> A = new ArrayList<>(a.length());
        List<Integer> B = new ArrayList<>(b.length());
        for(int i = a.length() - 1; i >= 0; i--){
            A.add(a.charAt(i) - '0');
        }
        for(int i = b.length() - 1; i >= 0; i--){
            B.add(b.charAt(i) - '0');
        }
        List<Integer> C = add(A, B);
        for(int i = C.size() - 1; i >= 0; i--){
            System.out.printf("%d", C.get(i));
```

```
        }
    }
}
```

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，a1, a2, a3.....an

前缀和 Si = a1 + a2 + ... + ai , S0 = 0;

模板：

```java
import java.util.Scanner;
public class Main{
    public static void main(String[] args){
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int m = sc.nextInt();
        int[] a = new int[n + 1];
        int[] s = new int[n + 1];
        for(int i = 1; i <= n; i++){
            a[i] = sc.nextInt();
        }
        for(int i = 1; i <= n; i++){
            s[i] = s[i - 1] + a[i];
        }
        while(m -- > 0){
            int l = sc.nextInt();
            int r = sc.nextInt();
            System.out.println(s[r] - s[l - 1]);
        }
    }
}
```

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：

```
import java.util.Scanner;
public class Main{
    public static void main(String[] args){
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int m = sc.nextInt();
        int q = sc.nextInt();
        int[][] a = new int[n + 1][m + 1];
        int[][] s = new int[n + 1][m + 1];
        for(int i = 1; i <= n; i++){
            for(int j = 1; j <= m; j++){
                a[i][j] = sc.nextInt();
            }
        }
        for(int i = 1; i <= n; i++){
            for(int j = 1; j <= m; j++){
                s[i][j] = s[i - 1][j] + s[i][j - 1] - s[i - 1][j - 1] + a[i][j];

            }
```

```
        }
        while(q-- > 0){
            int x1 = sc.nextInt();
            int y1 = sc.nextInt();
            int x2 = sc.nextInt();
            int y2 = sc.nextInt();
            System.out.println(s[x2][y2] - s[x1 -1][y2] - s[x2][y1 -1] + s[x1 -1][y1 -1]);
        }
    }
}
```

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],,,,,, a[n]；然后构造一个数组 b : b[1] ,b[2] , b[3],,,,,, b[i]；使得：a[i] = b[1] + b[2]+ b[3] +,,,,,, + 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 ,,,,,, 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,,,,,, 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,,,,,,,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：

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

```
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int m = sc.nextInt();
        int[] a = new int[100010];
        int[] b = new int[100010];
        for(int i = 1; i <= n; i++){
            a[i] = sc.nextInt();
            insert(b, i, i, a[i]);
        }
        while(m -- > 0){
            int l = sc.nextInt();
            int r = sc.nextInt();
            int c = sc.nextInt();
            insert(b, l, r, c);
        }
        for(int i = 1; i <= n; i++){
            b[i] += b[i - 1];
            System.out.printf("%d ", b[i]);
        }
    }
}
```

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让原二维数组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：

```
import java.util.Scanner;
public class Main{
    private  static void insert(int[][] b, int x1 ,int y1, int x2, int y2, int c){
        b[x1][y1] += c;
        b[x1][y2+1] -= c;
        b[x2+1][y1] -= c;
        b[x2+1][y2+1] += c;
    }
    public static void main(String[] args){
        Scanner sc  = new Scanner(System.in);
        int n = sc.nextInt();
        int m = sc.nextInt();
        int q = sc.nextInt();
        int[][] a = new int[1010][1010];
        int[][] b = new int[1010][1010];
        for(int i = 1 ; i <= n ; i ++ ){
            for(int j = 1 ; j <= m ; j ++ ){
                a[i][j] = sc.nextInt();
                insert(b, i, j, i, j, a[i][j]);
            }
        }
        while(q-- > 0){
            int x1 = sc.nextInt();
            int y1 = sc.nextInt();
            int x2 = sc.nextInt();
            int y2 = sc.nextInt();
            int c = sc.nextInt();
            insert(b, x1, y1, x2, y2, c);
        }
```

```
        for(int i = 1 ; i <= n ; i ++ ){
            for(int j = 1 ; j <= m ; j ++ ){
                a[i][j] = a[i-1][j] + a[i][j-1] - a[i-1][j-1] + b[i][j];
                System.out.print(a[i][j] + " ");
              }
           System.out.println();
        }
    }
}
```