里得算法，又称为辗转相除法。

具体做法为：如果 q 和 r 分别是 m 除以 n 的商及余数，即 $m=nq+r$，那么 m 和 n 的最大公约数等于 n 和 r 的最大公约数。详细证明略。

具体请参看如下代码中的 gcd 方法。

```
public int gcd(int m, int n) {
        return n == 0 ? m : gcd(n, m % n);
}
```

有关阶乘的两个问题

【题目】

给定一个非负整数 N，返回 $N!$结果的末尾为 0 的数量。

例如：3!=6，结果的末尾没有 0，则返回 0。5!=120，结果的末尾有 1 个 0，返回 1。1000000000!，结果的末尾有 249999998 个 0，返回 249999998。

【进阶题目】

给定一个非负整数 N，如果用二进制数表达 $N!$的结果，返回最低位的 1 在哪个位置上，认为最右的位置为位置 0。

例如：1!=1，最低位的 1 在 0 位置上。2!=2，最低位的 1 在 1 位置上。1000000000!，最低位的 1 在 999999987 位置上。

【难度】

原问题　尉 ★★☆☆

进阶问题　校 ★★★☆

【解答】

无论是原问题还是进阶问题，通过算出真实的阶乘结果后再处理的方法无疑是不合适的，因为阶乘的结果通常很大，非常容易溢出，而且会增加计算的复杂性。

先来介绍原问题的一个普通解法。对原问题来说，$N!$结果的末尾有多少个 0 的问题可以转换为 1，2，3，…，N-1，N 的序列中一共有多少个因子 5。这是因为 $1\times2\times3\times\cdots\times N$ 的过程中，因子 2 的数目比因子 5 的数目多，所以不管有多少个因子 5，都有足够的因子 2

与其相乘得到 10。所以只要找出 1~N 所有的数中，一共含有多少个因子 5 就可以。具体参看如下代码中的 zeroNum1 方法。

```
public int zeroNum1(int num) {
        if (num < 0) {
                return 0;
        }
        int res = 0;
        int cur = 0;
        for (int i = 5; i < num + 1; i = i + 5) {
                cur = i;
                while (cur % 5 == 0) {
                        res++;
                        cur /= 5;
                }
        }
        return res;
}
```

以上方法的效率并不高，对每一个数 i 来说，处理的代价是 $\log i$（以 5 为底），一共有 $O(N)$个数。所以时间复杂度为 $O(N\log N)$。

现在介绍原问题的最优解。我们把 1~N 的数列出来。1，2，3，4，5，6，7，8，9，10…，15…，20…，25…，30…，35…，40…，45…，50…，75…，100…，125…

读者观察一下上面的数就会发现：

若每 5 个含有 0 个因子 5 的数(1，2，3，4，5)组成一组，这一组中的第 5 个数就含有 5^1 的因子(5)。若每 5 个含有 1 个因子 5 的数(5，10，15，20，25)组成一组，这一组中的第 5 个数就含有 5^2 的因子(25)。若每 5 个含有 2 个因子 5 的数(25，50，75，100，125)组成一组，这一组中的第 5 个数就含有 5^3 的因子(125)。若每 5 个含有 i 个因子 5 的数组成一组，这一组中的第 5 个数就含有 5^{i+1} 的因子……

所以，如果把 N!的结果中因子 5 的总个数记为 Z，就可以得到如下关系：

$Z = N/5 + N/(5^2) + N/(5^3) + ... + N/(5^i)$ (i 一直增长，直到 $5^i>N$)。

用上文的例子来理解就是，1~N 中有 $N/5$ 个数，这每个数都能贡献一个 5；然后 1~N 中有 $N/(5^2)$个数，这每个数又都能贡献一个 5……。具体请参看如下代码中的 zeroNum2 方法：

```
public int zeroNum2(int num) {
        if (num < 0) {
                return 0;
        }
        int res = 0;
        while (num != 0) {
                res += num / 5;
```

```
            num /= 5;
        }
        return res;
    }
```

可以看到，如果一共有 N 个数，最优解的时间复杂度为 $O(\log N)$，以 5 为底。

进阶问题。本书提供两种方法，先来介绍解法一。与原问题的解法类似，最低位的 1 在哪个位置上，完全取决于 1~N 的数中因子 2 有多少个，因为只要出现一个因子 2，最低位的 1 就会向左位移一位。所以，如果把 N!的结果中因子 2 的总个数记为 Z，我们就可以得到如下关系 $Z = N/2 + N/4 + N/8 + \cdots + N/(2^i)$ (i 一直增长，直到 $2^i>N$)。具体请参看如下代码中的 rightOne1 方法。

```
    public int rightOne1(int num) {
        if (num < 1) {
            return -1;
        }
        int res = 0;
        while (num != 0) {
            num >>>= 1;
            res += num;
        }
        return res;
    }
```

再来介绍解法二。如果把 N!的结果中因子 2 的总个数记为 Z，把 N 的二进制数表达式中 1 的个数记为 m，还存在如下一个关系 $Z = N - m$，也就是可以证明 $N/2 + N/4 + N/8 + \cdots = N-m$。注意，这里的/不是数学上的除法，而是计算科学中的除法，即结果要向下取整。首先，如果一个整数 K 正好为 2 的某次方($K=2^i$)，那么求和公式 $K/2+K/4+K/8+\cdots = K/2+K/4+K/8+\cdots+1$，也就是在 $K=2^i$ 时，计算科学中的除法和数学上的除法等效。所以根据等比数列求和公式 S=(末项×公比-首项)/(公比-1)，可以得到 $K/2+K/4+K/8+\cdots=K-1$。

如果在 N 的二进制表达中有 m 个 1，那么 N 可以表达为：$N=K1+K2+K3+\cdots+Km$，其中的所有 K 都等于 2 的某次方，例如，N=10110 时，N=10000+100+10。于是有 $N/2+N/4+\cdots=(K1+K2+K3+\cdots+Km)/2+(K1+K2+K3+\cdots+Km)/4+\cdots=K1/2+K1/4+K1/8+\cdots+1+K2/2+K2/4+\cdots+1+\cdots+Km/2+Km/4+\cdots+1$。

$K1$，$K2$，…，Km 都等于 2 的某次方。所以等式右边$=K1-1+K2-1+K3-1+\cdots+Km-1=(K1+\cdots+Km)-m=N-m$。至此，$Z=N-m$ 证明完毕。具体过程请参看如下代码中 rightOne2 方法。

```
    public int rightOne2(int num) {
        if (num < 1) {
            return -1;
        }
```

```
		int ones = 0;
		int tmp = num;
		while (tmp != 0) {
			ones += (tmp & 1) != 0 ? 1 : 0;
			tmp >>>= 1;
		}
		return num - ones;
	}
```

判断一个点是否在矩形内部

【题目】

在二维坐标系中，所有的值都是double类型，那么一个矩形可以由4个点来代表，$(x1,y1)$为最左的点、$(x2,y2)$为最上的点、$(x3,y3)$为最下的点、$(x4,y4)$为最右的点。给定 4 个点代表的矩形，再给定一个点(x,y)，判断(x,y)是否在矩形中。

【难度】

尉 ★★☆☆

【解答】

本题的解法有很多种，本书提供的方法先解决如果矩形的边不是平行于 x 轴就是平行于 y 轴的情况下，该如何判断点(x,y)是否在其中，具体请参看如下代码中的 isInside 方法。

```
	public boolean isInside(double x1, double y1, double x4, double y4,
			double x, double y) {
		if (x <= x1) {
			return false;
		}
		if (x >= x4) {
			return false;
		}
		if (y >= y1) {
			return false;
		}
		if (y <= y4) {
			return false;
		}
		return true;
	}
```

这种情况是比较简单的，因为矩形的边不是平行于 x 轴就是平行于 y 轴，所以判断该