如何快速判断正整数是2的N次幂

这个问题可能很多面试的人都遇到过，很多人可能想利用循环来判断，代码可能如下所示：

```
public static boolean isPowOfTwo(int n) {
    int temp = 0;
    for (int i = 1; ; i++) {
        temp = (int) Math.pow(2, i);
        if (temp >= n)
            break;
    }
    if (temp == n) return true;
    else return false;
}
```

上面的代码简单明了。但是，这样的方案效率比较低。我们仔细分析一下，正整数是2的n次幂他有什么规律？2^0=1,2^1=2,2^2=4,2^3=8....这样看是没有什么规律的。但是如果将2的幂次方写成二进制形式后，很容易就会发现有以下两个特点：

1、2^0=1 -> 0001,2^1=2 -> 0010,2^2=4 -> 0100,2^3=8 -> 1000二进制中只有一个1，并且1后面跟了n个0。

2、如果将这个数减去1后会发现，仅有的那个1会变为0，而原来的那n个0会变为1；因此将原来的数与去减去1后的数字进行与运算后会发现为零（(x & x- 1) == 0）。

原因：因为2^n换算是二进制为10.....0这样的形式，2^n-1的二进制为0111...1，两个二进制求与结果为0，例如：16的二进制为10000；15=01111，两者相与的结果为0。计算如下：

```
  10000
 &01111
 -------
  00000
```

所以可以用下面算法实现

```
public static boolean isPowerOfTwo(int x) {
  return x > 0 & (x & (x - 1)) == 0;
}
```

很简单的一行代码就实现了。细心的读者可能会问：2的n次幂二进制始终都是只有一个1，其它的位数都为0，是否可以判断给定的数转换为二进制来判断其中是否只有1个1来得出给定数是否为2的n次幂呢？答案是不能。因为Integer.MIN_VALUE的二进制只有1个1，但是Integer.MIN_VALUE并不是2的n次幂，所以不能用上面方式来实现。（完）