Definitionen:
Bit Länge der Zahl (N) : k
Distanz (2^k - 1 - N): d

```
Nr      Binär           Adjacent Bits
-----------------------------------------
n(0) = 0
************************************************/



Nr      Binär           Adjacent Bits
0          0            0
1          1            0
-----------------------------------------
n(1) = 0
************************************************/



Nr      Binär           Adjacent Bits
0         00            0
1         01            0
2         10            0
3         11            1
-----------------------------------------
n(2) = 1
************************************************/



Nr      Binär           Adjacent Bits
0        000            0
1        001            0
2        010            0
3        011            1
4        100            0
5        101            0
6        110            1
7        111            2
-----------------------------------------
n(3) = 1 + 1 + (2^3)/4
n(3) = 4
************************************************/



Nr      Binär           Adjacent Bits           d
0       0000            0                       15
1       0001            0                       14
2       0010            0                       13
3       0011            1                       12
4       0100            0                       11
5       0101            0                       10
6       0110            1                       9
7       0111            2                       8
8       1000            0                       7
9       1001            0                       6
10      1010            0                       5
11      1011            1                       4
12      1100            1                       3
13      1101            1                       2
14      1110            2                       1
15      1111            3                       0
-----------------------------------------
n(4) = 4 + 4 + (2^4)/4
n(4) = 12
************************************************/

n(1) = 0
n(2) = 2n(1) + 2^(2-2)
n(3) = 2n(2) + 2^(3-2)
n(k) = 2n(k-1) + 2^(k-2)
```

```
import java.math.BigInteger;
import java.util.Scanner;

public class Main11645 {
private final static BigInteger[] numbers = new BigInteger[64];
        private final static BigInteger two = BigInteger.valueOf(2);
        public static BigInteger n(int i) {
                switch(i) {
                case 0:
                case 1:
                        return BigInteger.ZERO;
                default:
                        return (numbers[i-1].multiply(two)).add(BigInteger.valueOf(2).pow(i-2));
                }
        }

        public static void main(String... args) {
                for(int i = 0; i < 64; i++) {
                        numbers[i] = n(i);
                }
                long curCase = 1;
                Scanner s = new Scanner(System.in);
                for(BigInteger number = s.nextBigInteger(); number.compareTo(BigInteger.ZERO) >= 0; number =
                s.nextBigInteger()) {

                        int index = number.bitLength();


                        BigInteger maxNumber = BigInteger.ONE.shiftLeft(index);
                        BigInteger distance = (maxNumber.subtract(BigInteger.ONE)).subtract(number);
                        System.out.println("Case " + curCase + ": " + getRemaining(number, maxNumber, distance, index));
                        curCase++;
                }
        }

        private static BigInteger getRemaining(BigInteger number, BigInteger maxNumber, BigInteger distance, int index) {
                if(index<2){
                        return numbers[0];
                }
                if(distance.equals(BigInteger.ZERO)) {
                        return numbers[index];
                } else if(distance.equals(maxNumber.shiftRight(2))) {
                        return numbers[index-1].add(numbers[index-2]);
                } else {
                        int newIndex = index-1;
                        BigInteger newNumber = number;
                        if(number.testBit(newIndex)) {
                                newNumber = number.flipBit(newIndex);
                        }
                        BigInteger newMaxNumber = maxNumber.shiftRight(1);
                        BigInteger newDistance = (newMaxNumber.subtract(newNumber)).subtract(BigInteger.ONE);
                        BigInteger add = BigInteger.ZERO;

                        if(number.compareTo(newMaxNumber) >= 0) {
                                add = numbers[index-1];
                        }

                        if(distance.compareTo(maxNumber.shiftRight(2)) > 0) {
                                return add.add(getRemaining(newNumber,newMaxNumber,newDistance,newIndex));
                        } else {
                                return
                                add.add((((maxNumber.shiftRight(2)).subtract(distance).add(getRemaining(newNumber,newMaxNu
                                mber,newDistance,newIndex)))));
                        }
                }

        }
}
```