```
from math import log2


def firstExercise(n):
    # We xor all the elements, those who appear an even number of times cancel out, the r
emaining one will be the odd one out
    vector = []
    for x in range(0, n):
        y = int(input(f"vectorelement {x} = "))
        vector.append(y)

    finalvalue = 0
    for value in vector:
        finalvalue = finalvalue ^ value
    print(f"Number Odd Times {finalvalue}")


def secondExercise(n):
    # We select each subset based on the binary representation of a number
    # We declare a final variable
    final = 0
    # And create the number until we count to
    k = (1 << (n))
    # We will load our elements inside an array
    vector = []
    for x in range(1, n+1):
        y = int(input(f"vectorelement {x} = "))
        vector.append(y)
    # And for each number we get its binary representation, each number of 1 representing
 an occurrence in the current subset
    for a in range(1, k):
        copya = a # We have a copy of number a
        contor_position = 0
        print("(", end=" ")
        while copya > 0:
            # We can xor the values dirrectly as we get them because of associativity
            if copya & 1 == 1 and contor_position < len(vector):
                final = (final ^ vector[contor_position])
                print(f"{vector[contor_position]}, ", end=" ")
            # We consume the number bitshifting it to the right
            copya = (copya >> 1)
            contor_position = (contor_position + 1)
        print(")")
    print(f"Final value of all the elements xorred is: {final}")

def thirdExercise():
    x = int(input("x = "))
    y = int(input("y = "))

    print (f"X: {bin(x)}")
    print (f"Y: {bin(y)}")
    t = x ^ y
    k = 0
    # Our bit mask contains all the bits that do not have common values in both numbers (
marked as 1)
    # And we simply determine the number of bits of 1 in the representation
    while t > 0:
        t = t & (t-1) # We eliminate the last bit of the number
        k = k + 1
    print(f"Number of bits from x switched to become y: {k}")

# 4. Fie ð\235\221\233 un numÄ\203r natural. SÄ\203 se determine cea mai micÄ\203 putere
a lui 2 mai mare sau egalÄ\203 decÃ¢t  numÄ\203rul ð\235\221\233.

def fourthExerise(number):
    # We need to get the position of the last bit of 1 from the representation
    counter = 0
    cntchange = 0
    # We get the position of the last value of 1 in the representation
    while number != 0:
        if (number & 1 == 1):
            finalpos = counter
            cntchange = cntchange + 1
        number = number >> 1
        counter = counter + 1
    if cntchange > 1:
        # If there is more than 1 value of 1, we consider the number as not being a power
 of 2 -> we show the number bitshifted (counter+1) times
        print(f"Power of 2 greater than or equal to number: {1 << (counter)}")
    else:
        # Here, the number is a power of 2 and can be shown as it is (or as 1 bitshifted
counter times)
        print(f"Power of 2 greater than or equal to number: {1 << (counter-1)}")

def fifthExercise(n):
    print(f"Binary: {bin(n)}")

    # Now we count the position of the last 1 in the representation of n
    copyn = n
    # We get the total number of bits from the representation
    positionoflast1 = int(1 + log2(n))
    # And create the bitmask, all values being 1
    bitmask = (1 << positionoflast1) - 1

    # By xorring with the bitmask, we invert the bits
    copyn = copyn ^ bitmask

    print(f"Binary: {bin(bitmask)}")
    print(f"Copyn Binary: {bin(copyn)}")

    # And now we just count the number of values of 1 from the new representation
    k = 0
    while (copyn > 0):
        copyn = (copyn & (copyn-1)) # We eliminate the lsb of 1
        k = k + 1
    print(f"Number of zeros from binary representation: {k}")

def main():
    x = int(input("x = "))
    fifthExercise(x)

if __name__ == "__main__":
    main();
```