# Splitting Numbers

We define the operation of splitting a binary number `n` into two numbers `a(n)` and `b(n)` as follows. Let `0 <= i1 < i2 < ... < ik` be the indices of the bits (with the least significant bit having index 0) in `n` that are 1. Then the indices of the bits of `a(n)` that are 1 are `i1`, `i3`, `i5`, ... and the indices of the bits of `b(n)` that are 1 are `i2`, `i4`, `i6`, ...

For example, if `n` is `110110101` in binary then, again in binary, `a = 010010001` and `b= 100100100`.

### Input

The input consists of a single integer `n` between `1` and `2^31 - 1` written in standard decimal (base 10) format on a single line.

### Output

The output consists of a single **line**, containing the integers `a(n)` and `b(n)` separated by a single space. Both `a(n)` and `b(n)` should be written in decimal format.

### Example 1

```
Input:
6
Output:
2 4
```

### Example 2

```
Input:
7
Output:
5 2
```

### Example 3

```
Input:
13
Output:
9 4
```

# Big- or Little- Endian

All computer memory is organized into 8-bit bytes. For integer values that require a type with more than 8-bits, such as the typical 2-byte, 4-byte, and 8-byte integral types available on most modern hardware, there is no universal agreement as to how to order those bytes and two incompatible storage schemes exist.

The first stores integers as groups of consecutive 8-bit bytes with the least significant byte occupying the lowest memory address within the group and the most significant byte occupying the highest memory address.

The second is just the reverse; the least significant byte is stored in the highest memory address within the group, and the most significant byte is stored in the lowest memory address.

Those schemes have been dubbed Little Endian and Big Endian, respectively.

We assume that signed integers are stored using "two's complement" representation.

When binary integer data is shared between a Little Endian and Big Endian machine, a data conversion must be performed which involves reversing the bytes of the data. Once the bytes have been reversed the integer is then correctly interpreted by the hardware as the original value from the opposite-endian machine. Your task is to write a program that will read a list of integers and report the integers that the binary representations of the input integers would represent on an opposite-endian machine.

**Input**

A number `n`, `-2147483648 <= n <= 2147483647`.

**Output**

The number `n` followed by the phrase "converts to" followed by the other endian-value (single space between the three parts).

**Example 1**

```
Input:
123456789

Output:
123456789 converts to 365779719
```

**Example 2**

```
Input:
-123456789

Output:
-123456789 converts to -349002504
```

**Example 3**

```
Input:
1

Output:
1 converts to 16777216
```

**Example 4**

```
Input:
16777216

Output:
16777216 converts to 1
```

# Base Conversion

Write a program to convert a whole number specified in any base (2..16) to a whole number in any other base (2..16). *Digits* above 9 are represented by single capital letters: 10 by A, 11 by B, 12 by C, 13 by D, 14 by E and 15 by F.

**Input**

The input consists of three values: - a positive integer indicating the base of the number - a positive integer indicating the base we wish to convert to - the actual number in the firs base that we wish to convert to the second base (this number will have letters representing any digits higher than 9; **it may have invalid *digits***)

**Output**

The output should consist of the original number followed by the string "base", followed by the original base number, followed by the string "=" followed by the converted number, followed by the string "base" followed by the new base.

If the original number is invalid, output the statement:

`originalValue` is an illegal base `originalBase` number

where `originalValue` is replaced by the value to be converted and `originalBase` is replaced by the original base value.

**Example 1**

```
Input:
2 10 10101
Output:
10101 base 2 = 21 base 10
```

**Example 2**

```
Input:
5 3 126
Output:
126 is an illegal base 5 number
```

**Example 3**

```
Input:
15 11 A4C
Output:
A4C base 15 = 1821 base 11
```

# No-Blink Contest

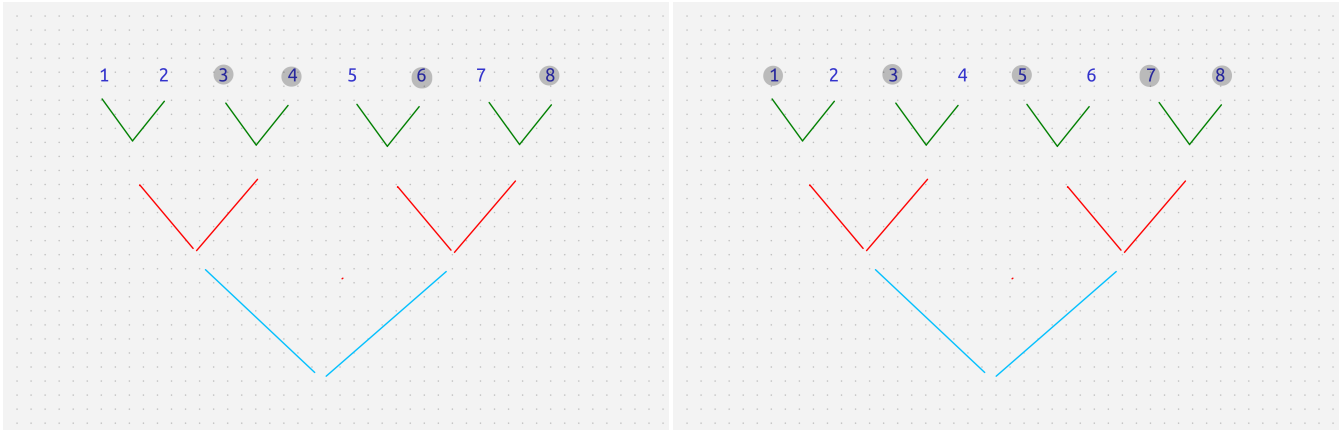
NYU has been planning a no-blink contest for the start of the freshman week. In a no-blink contest, one contestant faces the other contestant in a match and the one who blinks first loses. All 2^S first year students were expected to participate. Because of flight delays due to some hurricane activity at the end of August, many students were not able to arrive on time. The organizers of the contest spent a lot of time selecting the pairs for initial round of matches (based on their roommate preferences specified in the college applications) and did not want to rearrange the entire contest last minute. Instead, they decided to allow *free passes* in cases when one of the contestants is missing: - If both contestants are present, then the regular no-blink match will happen. - If only one contestant arrived on time for the contest, they will be automatically declared a winner, i.e., get a *free pass*. - If no contestants arrived on time then there is no match between them.

If the students arrive after the start of the first round of matches, they cannot join in the later rounds.

In the figures below we start with 2^3 students in each case. In the situation on the left four students did not arrive on time and in the situation on the right fives five students are missing the contest. These students are marked with gray circles around their number.

In the example on the left students 1 and 2 play a regular match. But when one of them advances to the next level there will be no other contestant since students 3 and 4 are both missing (this is a *free pass* match). Both students 5 and 7 have *free pass* matches in the first round and compete against each other in the second round. There will be two contestants for the final match. So there are total of three *free pass* matches.

In the example on the right, students 2, 4, and 6 all win by *free pass* matches in the first round. Students 2 and 4 will compete against each other, but student 5 gets a *free pass* match in the second round as well and competes only in the final round. So there are total of four *free pass* matches.




Given the list of students who could not arrive on time, calculate the number of the *free pass* matches in the whole contest.

### Input

Each test begins with two integers: `1 <= S <= 10` and `0 <= M <= 2^S`. `2^S` is the number of students originally scheduled for the contest. `M` is the number of students missing. The next line contains `M` integers, denoting the students who have not arrived. The students are numbered `1` to `2^S`.

### Output

The number of *free pass* matches.

### Example 1

```
Input:
3 4
3 4 6 8
Output:
3
```

**Example 2**

```
Input:
3 5
1 3 5 7 8
Output:
4
```

**Example 3**

```
Input:
2 1
2
Output:
1
```

# Prime Pair

A prime pair is a pair of the form (p, p + 2) where p and (p + 2) are both prime. The first few prime pairs are (3, 5), (5, 7), (11, 13). In this problem, you are given an integer N and you should find the N'th prime pair.

**Input** The input will contain a single integer N (1 <= N <= 100,000).

**Output** The output should be a single **line**, the N'th prime pair formatted as `(p1, p2)` (there is a space after the comma). It is guaranteed that primes in the 100,000th prime pair are less than 20,000,000.

**Example 1**

```
Input:
1

Output:
(3, 5)
```

**Example 2**

```
Input:
2

Output:
(5, 7)
```

**Example 3**

```
Input:
3

Output:
(11, 13)
```

**Example 4**

```
Input:
4

Output:
(17, 19)
```