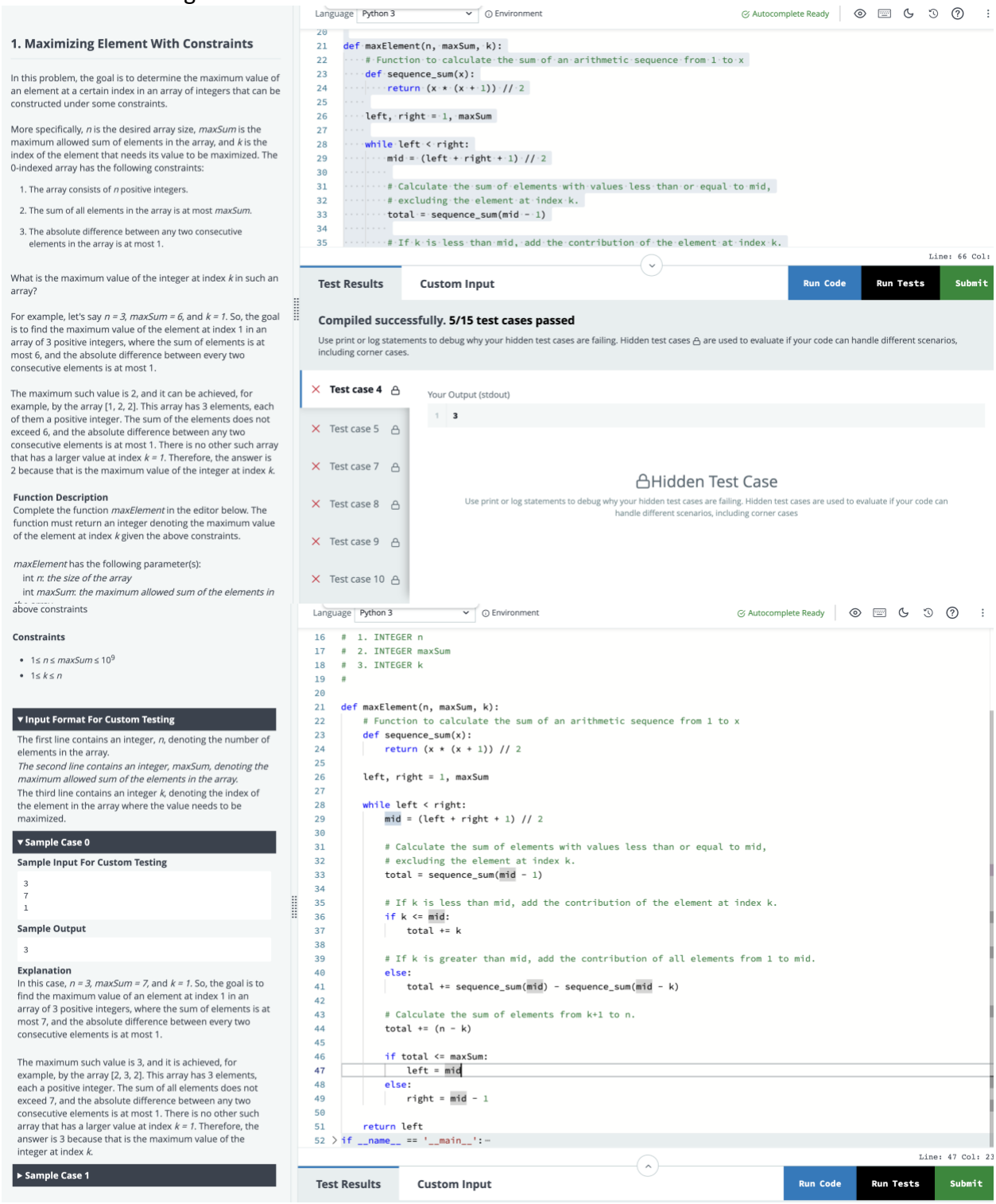
Problem Solving Intermediate

1. Maximizing Element With Constraints



1. Maximizing Element With Constraints

In this problem, the goal is to determine the maximum value of an element at a certain index in an array of integers that can be constructed under some constraints.

More specifically, *n* is the desired array size, *maxSum* is the maximum allowed sum of elements in the array, and *k* is the index of the element that needs its value to be maximized. The 0-indexed array has the following constraints:

1. The array consists of *n* positive integers.
2. The sum of all elements in the array is at most *maxSum*.
3. The absolute difference between any two consecutive elements in the array is at most 1.

What is the maximum value of the integer at index *k* in such an array?

For example, let's say *n = 3*, *maxSum = 6*, and *k = 1*. So, the goal is to find the maximum value of the element at index 1 in an array of 3 positive integers, where the sum of elements is at most 6, and the absolute difference between every two consecutive elements is at most 1.

The maximum such value is 2, and it can be achieved, for example, by the array [1, 2, 2]. This array has 3 elements, each of them a positive integer. The sum of the elements does not exceed 6, and the absolute difference between any two consecutive elements is at most 1. There is no other such array that has a larger value at index *k = 1*. Therefore, the answer is 2 because that is the maximum value of the integer at index *k*.

Function Description
Complete the function *maxElement* in the editor below. The function must return an integer denoting the maximum value of the element at index *k* given the above constraints.

maxElement has the following parameter(s):
int *n*: *the size of the array*
int *maxSum*: *the maximum allowed sum of the elements in the array* above constraints

Constraints

- $1 \le n \le maxSum \le 10^9$
- $1 \le k \le n$

▾ Input Format For Custom Testing

The first line contains an integer, *n*, denoting the number of elements in the array.
The second line contains an integer, maxSum, denoting the maximum allowed sum of the elements in the array.
The third line contains an integer *k*, denoting the index of the element in the array where the value needs to be maximized.

▾ Sample Case 0

Sample Input For Custom Testing

```
3
7
1
```

Sample Output

```
3
```

Explanation
In this case, *n = 3*, *maxSum = 7*, and *k = 1*. So, the goal is to find the maximum value of an element at index 1 in an array of 3 positive integers, where the sum of elements is at most 7, and the absolute difference between every two consecutive elements is at most 1.

The maximum such value is 3, and it is achieved, for example, by the array [2, 3, 2]. This array has 3 elements, each a positive integer. The sum of all elements does not exceed 7, and the absolute difference between any two consecutive elements is at most 1. There is no other such array that has a larger value at index *k = 1*. Therefore, the answer is 3 because that is the maximum value of the integer at index *k*.

▸ Sample Case 1

Language: Python 3 | Environment | Autocomplete Ready

```
def maxElement(n, maxSum, k):
    # Function to calculate the sum of an arithmetic sequence from 1 to x
    def sequence_sum(x):
        return (x * (x + 1)) // 2

    left, right = 1, maxSum

    while left < right:
        mid = (left + right + 1) // 2

        # Calculate the sum of elements with values less than or equal to mid,
        # excluding the element at index k.
        total = sequence_sum(mid - 1)

        # If k is less than mid, add the contribution of the element at index k.
```

Line: 66 Col:

Test Results | Custom Input | Run Code | Run Tests | Submit

Compiled successfully. 5/15 test cases passed

Use print or log statements to debug why your hidden test cases are failing. Hidden test cases are used to evaluate if your code can handle different scenarios, including corner cases.

- ✕ Test case 4
- ✕ Test case 5
- ✕ Test case 7
- ✕ Test case 8
- ✕ Test case 9
- ✕ Test case 10

Your Output (stdout)

```
3
```

Hidden Test Case

Use print or log statements to debug why your hidden test cases are failing. Hidden test cases are used to evaluate if your code can handle different scenarios, including corner cases

Language: Python 3 | Environment | Autocomplete Ready

```
#  1. INTEGER n
#  2. INTEGER maxSum
#  3. INTEGER k
#

def maxElement(n, maxSum, k):
    # Function to calculate the sum of an arithmetic sequence from 1 to x
    def sequence_sum(x):
        return (x * (x + 1)) // 2

    left, right = 1, maxSum

    while left < right:
        mid = (left + right + 1) // 2

        # Calculate the sum of elements with values less than or equal to mid,
        # excluding the element at index k.
        total = sequence_sum(mid - 1)

        # If k is less than mid, add the contribution of the element at index k.
        if k <= mid:
            total += k

        # If k is greater than mid, add the contribution of all elements from 1 to mid.
        else:
            total += sequence_sum(mid) - sequence_sum(mid - k)

        # Calculate the sum of elements from k+1 to n.
        total += (n - k)

        if total <= maxSum:
            left = mid
        else:
            right = mid - 1

    return left
if __name__ == '__main__':
```

Line: 47 Col: 23

Test Results | Custom Input | Run Code | Run Tests | Submit

1

Sample Output

```
3
```

Explanation
In this case, *n = 3, maxSum = 7,* and *k = 1*. So, the goal is to find the maximum value of an element at index 1 in an array of 3 positive integers, where the sum of elements is at most 7, and the absolute difference between every two consecutive elements is at most 1.

The maximum such value is 3, and it is achieved, for example, by the array [2, 3, 2]. This array has 3 elements, each a positive integer. The sum of all elements does not exceed 7, and the absolute difference between any two consecutive elements is at most 1. There is no other such array that has a larger value at index *k = 1*. Therefore, the answer is 3 because that is the maximum value of the integer at index *k*.

▼ Sample Case 1

Sample Input For Custom Testing

```
4
6
2
```

Sample Output

```
2
```

Explanation
In this case, *n = 4, maxSum = 6,* and *k = 2*. So, the goal is to find the maximum value of an element at index 2 in an array of 4 positive integers, where the sum of elements is at most 6, and the absolute difference between every two consecutive elements is at most 1.

The maximum such value is 2, and it is achieved, for example, by the array [1, 1, 2, 1]. This array has 4 elements, each a positive integer. The sum of all elements does not exceed 6, and the absolute difference between any two consecutive elements is at most 1. There is no other such array that has a larger value at index *k = 2*. Therefore, the answer is 2 because that is the maximum value of the integer at index *k*.

Language Python 3 | Environment | Autocomplete Ready

```python
#  1. INTEGER n
#  2. INTEGER maxSum
#  3. INTEGER k
#

def maxElement(n, maxSum, k):
    # Function to calculate the sum of an arithmetic sequence from 1 to x
    def sequence_sum(x):
        return (x * (x + 1)) // 2

    left, right = 1, maxSum

    while left < right:
        mid = (left + right + 1) // 2

        # Calculate the sum of elements with values less than or equal to mid,
        # excluding the element at index k.
        total = sequence_sum(mid - 1)

        # If k is less than mid, add the contribution of the element at index k.
        if k <= mid:
            total += k

        # If k is greater than mid, add the contribution of all elements from 1 to mid.
        else:
            total += sequence_sum(mid) - sequence_sum(mid - k)

        # Calculate the sum of elements from k+1 to n.
        total += (n - k)

        if total <= maxSum:
            left = mid
        else:
            right = mid - 1

    return left
if __name__ == '__main__':
```

Line: 52 Col: 27

Test Results | Custom Input | Run Code | Run Tests | Submit

1. Maximizing Element With Constraints

In this problem, the goal is to determine the maximum value of an element at a certain index in an array of integers that can be constructed under some constraints.

More specifically, *n* is the desired array size, *maxSum* is the maximum allowed sum of elements in the array, and *k* is the index of the element that needs its value to be maximized. The 0-indexed array has the following constraints:

1. The array consists of *n* positive integers.
2. The sum of all elements in the array is at most *maxSum*.
3. The absolute difference between any two consecutive elements in the array is at most 1.

What is the maximum value of the integer at index *k* in such an array?

For example, let's say *n = 3, maxSum = 6,* and *k = 1*. So, the goal is to find the maximum value of the element at index 1 in an array of 3 positive integers, where the sum of elements is at most 6, and the absolute difference between every two consecutive elements is at most 1.

The maximum such value is 2, and it can be achieved, for example, by the array [1, 2, 2]. This array has 3 elements, each of them a positive integer. The sum of the elements does not exceed 6, and the absolute difference between any two consecutive elements is at most 1. There is no other such array that has a larger value at index *k = 1*. Therefore, the answer is 2 because that is the maximum value of the integer at index *k*.

Function Description
Complete the function *maxElement* in the editor below. The function must return an integer denoting the maximum value of the element at index *k* given the above constraints.

maxElement has the following parameter(s):
int *n*: *the size of the array*
int *maxSum*: *the maximum allowed sum of the elements in the array*

Language Python 3 | Environment | Autocomplete Ready

```python
        # If k is greater than mid, add the contribution of all elements from 1 to mid.
        else:
            total += sequence_sum(mid) - sequence_sum(mid - k)

        # Calculate the sum of elements from k+1 to n.
        total += (n - k)

        if total <= maxSum:
            left = mid
        else:
            right = mid - 1

    return left
if __name__ == '__main__':
```

Line: 52 Col: 27

Test Results | Custom Input | Run Code | Run Tests | Submit

Compiled successfully. 5/15 test cases passed

Use print or log statements to debug why your hidden test cases are failing. Hidden test cases are used to evaluate if your code can handle different scenarios, including corner cases.

- ✗ Test case 14
- ✓ Test case 0
- ✓ **Test case 1**
- ✓ Test case 2
- ✓ Test case 3
- ✓ Test case 6

Input (stdin) — Run as Custom Input | Download

```
4
6
2
```

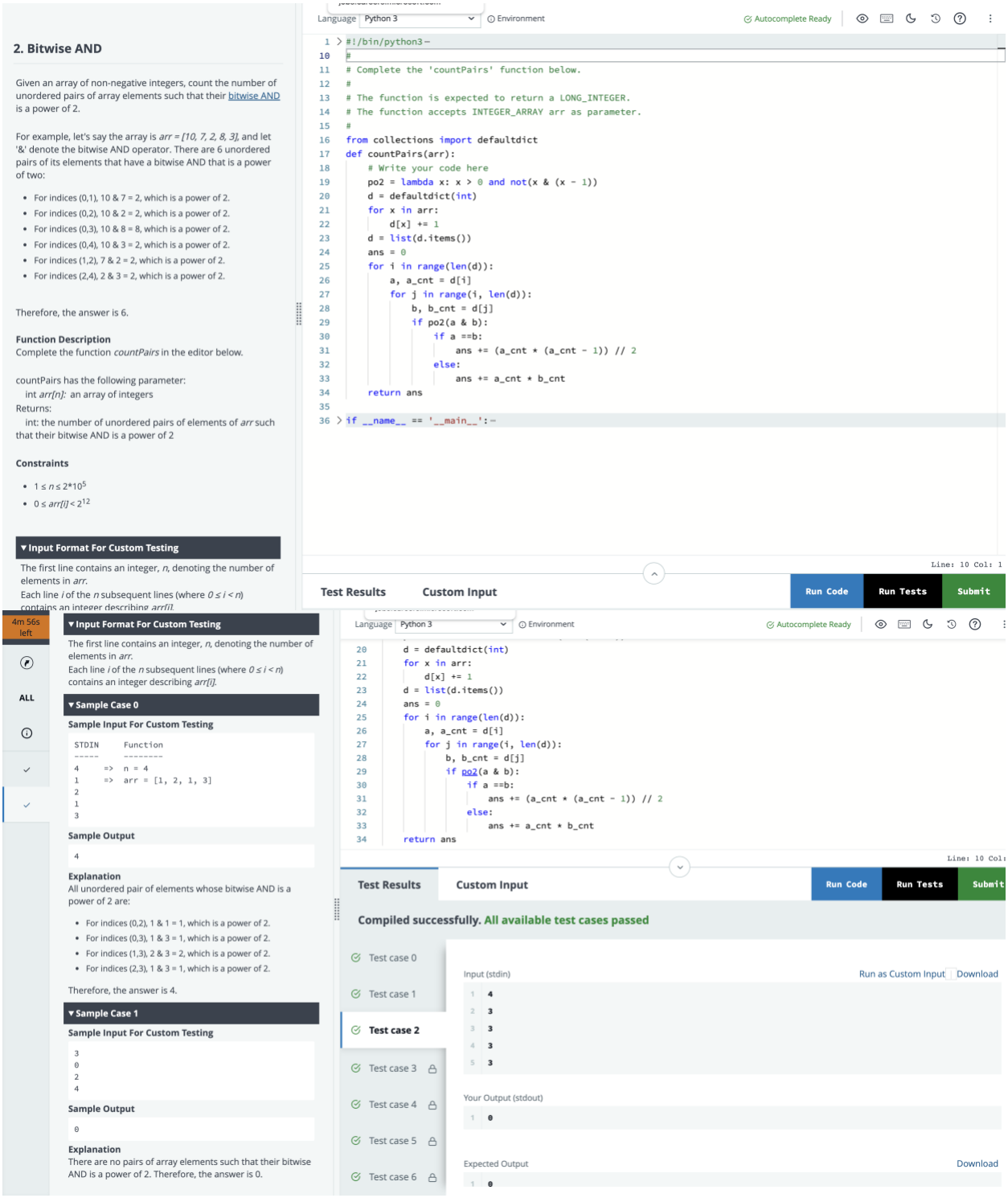
Your Output (stdout)

```
2
```

Expected Output — Download

```
2
```

2. Bitwise AND



2. Bitwise AND

Given an array of non-negative integers, count the number of unordered pairs of array elements such that their bitwise AND is a power of 2.

For example, let's say the array is *arr = [10, 7, 2, 8, 3]*, and let '&' denote the bitwise AND operator. There are 6 unordered pairs of its elements that have a bitwise AND that is a power of two:

- For indices (0,1), 10 & 7 = 2, which is a power of 2.
- For indices (0,2), 10 & 2 = 2, which is a power of 2.
- For indices (0,3), 10 & 8 = 8, which is a power of 2.
- For indices (0,4), 10 & 3 = 2, which is a power of 2.
- For indices (1,2), 7 & 2 = 2, which is a power of 2.
- For indices (2,4), 2 & 3 = 2, which is a power of 2.

Therefore, the answer is 6.

Function Description
Complete the function *countPairs* in the editor below.

countPairs has the following parameter:
int *arr[n]*: an array of integers
Returns:
int: the number of unordered pairs of elements of *arr* such that their bitwise AND is a power of 2

Constraints

- $1 \le n \le 2*10^5$
- $0 \le arr[i] < 2^{12}$

▼ Input Format For Custom Testing

The first line contains an integer, *n*, denoting the number of elements in *arr*.
Each line *i* of the *n* subsequent lines (where $0 \le i < n$) contains an integer describing *arr[i]*.

Language: Python 3 | Environment | Autocomplete Ready

```
#!/bin/python3
#
# Complete the 'countPairs' function below.
#
# The function is expected to return a LONG_INTEGER.
# The function accepts INTEGER_ARRAY arr as parameter.
#
from collections import defaultdict
def countPairs(arr):
    # Write your code here
    po2 = lambda x: x > 0 and not(x & (x - 1))
    d = defaultdict(int)
    for x in arr:
        d[x] += 1
    d = list(d.items())
    ans = 0
    for i in range(len(d)):
        a, a_cnt = d[i]
        for j in range(i, len(d)):
            b, b_cnt = d[j]
            if po2(a & b):
                if a ==b:
                    ans += (a_cnt * (a_cnt - 1)) // 2
                else:
                    ans += a_cnt * b_cnt
    return ans

if __name__ == '__main__':
```

Line: 10 Col: 1

Test Results | Custom Input | Run Code | Run Tests | Submit

4m 56s left

ALL

▼ Input Format For Custom Testing

The first line contains an integer, *n*, denoting the number of elements in *arr*.
Each line *i* of the *n* subsequent lines (where $0 \le i < n$) contains an integer describing *arr[i]*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN       Function
-----       --------
4       =>  n = 4
1       =>  arr = [1, 2, 1, 3]
2
1
3
```

Sample Output

```
4
```

Explanation
All unordered pair of elements whose bitwise AND is a power of 2 are:

- For indices (0,2), 1 & 1 = 1, which is a power of 2.
- For indices (0,3), 1 & 3 = 1, which is a power of 2.
- For indices (1,3), 2 & 3 = 2, which is a power of 2.
- For indices (2,3), 1 & 3 = 1, which is a power of 2.

Therefore, the answer is 4.

▼ Sample Case 1

Sample Input For Custom Testing

```
3
0
2
4
```

Sample Output

```
0
```

Explanation
There are no pairs of array elements such that their bitwise AND is a power of 2. Therefore, the answer is 0.

Language: Python 3 | Environment | Autocomplete Ready

```
    d = defaultdict(int)
    for x in arr:
        d[x] += 1
    d = list(d.items())
    ans = 0
    for i in range(len(d)):
        a, a_cnt = d[i]
        for j in range(i, len(d)):
            b, b_cnt = d[j]
            if po2(a & b):
                if a ==b:
                    ans += (a_cnt * (a_cnt - 1)) // 2
                else:
                    ans += a_cnt * b_cnt
    return ans
```

Line: 10 Col:

Test Results | Custom Input | Run Code | Run Tests | Submit

Compiled successfully. All available test cases passed

Test case 0
Test case 1
Test case 2
Test case 3
Test case 4
Test case 5
Test case 6

Input (stdin) — Run as Custom Input | Download

```
4
3
3
3
3
```

Your Output (stdout)

```
0
```

Expected Output — Download

```
0
```