Problem Solving

Problem: Make It Beautiful

(Problem Source: https://codeforces.com/problemset/problem/2118/C)

You are given an array a of n integers.
We define the beauty of a number x to be the number of 1 bits in its binary representation.
We define the beauty of an array to be the sum of beauties of the numbers it contains.

In one operation, you can select an index I ($1 \le i \le n$) and increase a_i by 1.

Find the maximum beauty of the array after doing **at most** k operations.

Input:
Each test contains multiple test cases. The first line contains the number of test cases t ($1 \le t \le 5000$). The description of the test cases follows:

The first line of each test case contains two integers n and k ($1 \le n \le 5000$, $0 \le k \le 10^{18}$) — the length of the array and the maximal number of operations.

The second line of each test case contains n integers $a_1, a_2, \ldots a_n$ ($0 \le a_i \le 10^9$) —denoting the array a.
It is guaranteed that the sum of n over all test cases does not exceed 5000.

Output: For each test case, output a single integer, the maximum beauty after at most k operations.

Example

Input	
2	**Output**
5 2	
0 1 7 2 4	8
5 3	
0 1 7 2 4	9

The Solution

Algorithm: Maximum Beauty of an Array

Input:

- t — number of test cases
- For each test case:
  - n — size of array
  - k — maximum number of operations
  - a[1..n] — array of integers

Output:

- Maximum beauty of the array after at most k increment operations

Algorithm: Maximum Beauty of an Array

Steps:

1. Read Input

- Read the number of test cases t.
- For each test case, read n, k, and the array a.

2. Compute Initial Beauty

- Initialize beauty = 0.
- For each element x in a, add the number of 1 bits in its binary representation to beauty.
  - (__builtin_popcountll(x) in C++ or equivalent)

3. Perform Increment Operations Greedily

- While k > 0:
  1. Initialize index = -1.
  2. For each element a[i] in the array:
     - Compute before = number of 1s in a[i]
     - Compute after = number of 1s in a[i] + 1
     - If after > before, set index = i and break the loop.
  3. If index == -1, no further gain is possible → break.
  4. Otherwise:
     - Increment a[index] by 1.
     - Increment beauty by 1.
     - Decrement k by 1.

4. Output Result

- Print the final value of beauty for the current test case.

5. Repeat for all test cases

```cpp
#include <iostream>
#include <vector>
using namespace std;

int main() {
    int t;
    cin >> t;
    while (t--)
    {
        long long n, k;
        cin >> n >> k;
        vector<long long> a(n);
        long long beauty = 0;
        for (long long i = 0; i < n; i++)
        {
            cin >> a[i];
            beauty += __builtin_popcountll(a[i]);
        }
        while (k > 0)
        {
            long long index = -1;
            for (long long i = 0; i < n; i++)
            {
                long long before = __builtin_popcountll(a[i]);
                long long after = __builtin_popcountll(a[i] + 1);
                if (after > before)
                {
                    index = i;
                    break;
                }
            }
            if (index == -1) break;

            a[index]++;
            beauty++;
            k--;
        }

        cout << beauty << "\n";
    }
}
```

Time Complexity

Case	Complexity Description	Big-O
Best Case	Few increments, each O(n) scan	O(n)
Worst Case	Huge k → many increments, each O(n) scan	O(n × k)
Average Case	Moderate k, bounded by bits	O(n × k)