Problem D. Game With Numbers

Author: Matvii Aslandukov
Developer: Matvii Aslandukov
Editorialist: Matvii Aslandukov

Subtask 1. In the first subtask $m = 1$ so you can find two sums s_1 and s_2: the sum of all elements that are divisible by b_1 and the sum of all elements that are not divisible by b_1. Then the answer is $\min(s_1, s_2)$ because the first player wants to minimize the sum of the remaining elements. Time complexity: $O(n)$.

Lemma 1. Before describing the solution for each individual subtask let's prove the following lemma: if both players can remove all the elements from the array a using only their own rounds, then the answer is 0. Indeed, in such a case both players can make the sum of the remaining elements in the array a equal to 0 regardless of the actions of the second player. The first player wants to minimize the sum, therefore the answer is not bigger than zero. At the same time, the second player wants to maximize the sum, therefore the answer is not less than zero. Thus the only possible answer is 0.

Subtask 2. In the second subtask two own rounds are enough for the first player to remove all the elements: he can remove all the numbers that are divisible by b_1 and are not divisible by b_3. At the same time, only one own round is enough for the second player to remove all the elements: he can make an opposite operation to the first player due to the condition $b_1 = b_2$. Thus by the lemma 1 the answer is 0 for $m > 2$. For $m = 2$ the answer is $\max(0, \min(s_1, s_2))$. Time complexity: $O(n)$.

Subtask 3. In the third subtask two own rounds are enough for each player to remove all the elements: the first player can remove all the numbers that are divisible by b_1 and are not divisible by b_3, and the second player can do the same using b_2 and b_4. Thus by the lemma 1 the answer is 0 for $m > 3$. For $m \leq 3$ you can either solve the problem recursively by considering each possible game scenario or consider up to 8 cases manually. Time complexity: $O(n)$.

Subtask 4. In the fourth subtask no additional observations are required. The entire game can be simulated recursively: you can represent the state of the game as a pair $(operations, pos)$, where $operations$ is an array with chosen operations and pos is a current round. Then at each state of the recursion you can try to make all two possible operations and select the best one. The total number of states is $O(2^m)$ and for each final state with $pos = m$ you can calculate the sum of the remaining elements in $O(nm)$. Therefore such a solution works in $O(2^m \cdot nm)$.

Subtask 5. In the fifth subtask you can modify recursion in the following way: instead of representing the state of the game as a pair $(operations, pos)$ you can represent it as a pair (a, pos), where a is an array with remaining elements and pos is a current round. Then at each state of the recursion you can try to make all two possible operations and select the best one. The total number of states is $O(2^m)$ but the total size of all arrays a across all states is only $O(nm)$ due to the fact that each initial element of the array a is contained in exactly m states. Therefore such a solution works in $O(2^m + nm)$. See the following code on `Python` for a better understanding.

```
def solve(a, pos):
    if pos == len(b):
        return sum(a)
    na = [[], []]
    for x in a:
        na[(x % b[pos]) == 0].append(x)
    return [min, max][pos % 2](solve(na[0], pos + 1), solve(na[1], pos + 1))

n, m = map(int, input().split())
a = list(map(int, input().split()))
b = list(map(int, input().split()))
print(solve(a, 0))
```

Subtask 6. In the sixth subtask you can notice that the majority of $O(2^m)$ states inside the recursion have an empty array a that allows to immediately return 0 as a result:

```
def solve(a, pos):
    if len(a) == 0:
        return 0
    ...
```

Such optimization gives time complexity $O(nm)$ which is enough for the sixth subtask.

Subtask 7. In the seventh subtask $a_i \geq 1$ that means that the final sum of the remaining elements is always non-negative. It simplifies the proof of the lemma 1 because now the goal of the first player is to remove all the elements from the array a. Also it can be proven that the answer is equal to 0 when $m \geq 19$ under the constraint $a_i \leq 10^9$. Such proof is left as an exercise for the reader (see bonus section). Such a fact means that the only difference from the fifth subtask is that we can just output 0 when $m > 20$.

Subtask 8. In the eighth subtask the second player can always remove all the elements by making the opposite second operation. It means that the answer is always non-negative. When the answer is positive the only strategy for the second player is to repeat the corresponding operations of the first player. It allows us to speed up the solution from the fifth subtask to $O(2^{m/2} + nm)$ which is ok for the $m \leq 40$. See the solution for the next subtask to understand what to do in case $m > 40$.

Subtask 9. For the full solution you can notice that $O(\log n)$ own rounds are enough for each player to remove all the elements. Indeed, at each round one of the two possible operations removes at least half of all remaining elements, therefore $\lceil \log_2 n \rceil$ rounds are always enough to remove all the elements. It means that the only difference from the sixth subtask is that we can just output 0 when $m > 100$ by the lemma 1.

Bonus. What is the maximum value of m where the answer is not equal to zero?