Definitely not a Nim Game

This problem was worth 1500 points and was ranked easy.
The author of this problem is Aryan V S.

Note: GitHub does not support LaTex in Markdown. If you want a more readable version of the problem, download the PDF file instead.

Story

I wanted to come up with a problem that had the GCD of an array as the solution but was hard enough to figure out. I started creating the problem from the solution. I wasn't very sure about the correctness of the solution, and so I asked help from a few very experienced Competitive Programmers on a private server. One of them suggested that the problem could also be solved using a Nim Game solution but didn't provide any further insight. It was probably a joke or maybe it was serious, I'll never know. I've spent quite some time studying Nim Games but I didn't see any resemblance, and so this problem is definitely not a Nim Game :) The title was only to confuse other participants who have studied about Nim Games.

Also, I really did not expect someone to solve the problem in under 60 minutes from the start of the contest. Great work Raghav S K, who solved it in 16 minutes from the start of the contest!

Statement

Aryan and Dhruv are observing a game being played. There are n people that are playing the game right now. The i^{th} person has a_i strength.

The rules of the game are as follows:

- There are infinitely many rounds to be played until a winner can be determined
- In every round, two random people are selected to fight each other. One of them chooses to attack and one of them chooses to defend. The strength of the person that chooses to defend reduces by the strength of the person who attacks. Formally, if person A with strength x and person B with strength y are playing the current round, and say person A chooses to attack while person B chooses to defend, then the strength of person B decreases to $y - x$ after the round
- A person is eliminated if they have non-positive strength i.e. ≤ 0
- The winner of the round is the last person standing

Aryan is interested in finding the minimum possible strength that the winner can have. Dhruv is interested in finding the maximum possible strength that the winner can have. Please help them pursue their interests!

Input Format

The first line of the input contains a single integer n.

The second line contains n space separated integers a_i.

Output Format

Output two space separated single integers - the minimum and maximum health that the winner can have.

Constraints

$2 \leq n \leq 3 \cdot 10^5$

$1 \leq a_i \leq 10^{18}$

Sample Tests

Sample Test 1

Input

```
5
10 20 15 50 30
```

Output

```
5 50
```

Sample Test 2

Input

```
3
2 3 2
```

Output

```
1 3
```

Solution

Time Complexity: $O(n + log(max(a_i)))$ or simply $O(n)$

Memory Complexity: $O(1)$ or $O(n)$

Firstly, finding the maximum possible health that the last man standing will have is simple. We can always choose the person with maximum strength to attack others with lower strength until he's the only one left. Therefore, the maximum strength that the winner can have is $max(a_i)$. Now, onto finding the minimum possible strength.

Let's, without the loss of generality, assume that if some person is attacked and their strength becomes negative, we can simply consider their strength to be 0. This way they are still eliminated but it allows us to observe an important property.

In the statement, we are allowed to do the following operation: select two distinct i and j indices randomly and do $a_i = a_i - a_j$, where a_j is the strength of the attacker and a_i is the strength of the defender. Also, if both a_i and a_j is divisible by some x, then $a_i - a_j$ is also divisible by x. If we find the minimum possible value of x, we will have our answer. Why? Because it would always be possible to obtain x after some of the above subtraction operations, and because x is the minimum such value.

After some operations, we can always make it so that all values can be represented as some linear function of all other values of the array. We want to solve for the constants of these linear equations such that $n - 1$ values are non-positive and 1 value is positive (and is also the minimum value possible).

Now, think about what the strengths look like after the last round of battle when finding the minimum, also keeping in mind the assumption from the first line of the solution. It would look something like: $[0,\ 0,\ \ldots,\ x,\ \ldots,\ 0,\ 0]$. That is, the minimum health x and a bunch of 0s. Consider this as the final state.

To achieve the final state, we must have had a previous state that looked something like: $[0,\ 0,\ \ldots,\ p,\ q,\ \ldots,\ 0,\ 0]$ where $max(p,\ q) - k \cdot min(p,\ q) = x$ for some value k. Without loss of generality, you can write x as a linear equation representation of the array elements following similar steps.

Now, we have a bunch of equations to represent x with. However, we still do not know the value of x. It turns out that the $gcd(a_i)$ (Greatest Common Divisor) always satisfies all the equations when set as the minimum. I will not show all the mathematical details; you should try working out a little bit on paper about the same.

Let's see why. Here are some properties of the gcd function:

- $gcd(0,\ X) = X$
- $gcd(X,\ Y) = gcd(Y,\ X)$
- $gcd(X,\ Y) = gcd(Y,\ X - Y)$ (assuming $Y \leq X$)
- $gcd(X,\ Y) = gcd(Y,\ X\ \%\ Y) = gcd(Y,\ X - i \cdot Y)$ for some i
- $gcd(X,\ Y,\ Z) = gcd(X,\ gcd(Y,\ Z))$

All these properties can also be observed in the way we create our linear equations. At each step, we try and represent x as some function in a way similar to how gcd works. After making all these observations, one can conclude that the $gcd(a_i)$ must be the minimal possible strength the winner can have.

Solution in C++:

```cpp
#include <bits/stdc++.h>

int main () {
  std::ios::sync_with_stdio(false);
  std::cin.tie(nullptr);

  int n;
  std::cin >> n;

  int64_t min = 0;
  int64_t max = 0;
```

```
  for (int i = 0; i < n; ++i) {
    int64_t a;
    std::cin >> a;
    min = std::gcd(min, a);
    max = std::max(max, a);
  }

  std::cout << min << ' ' << max << '\n';

  return 0;
}
```

Solution in Python:

```
import math

n = int(input())
a = list(map(int, input().split()))

print(math.gcd(*a), max(a))
```