

# 1 Introdução

Esse problema é o problema C do *AtCoder Begginer Contest 125*, da plataforma de programação competitiva AtCoder. Disponível em: [https://atcoder.jp/contests/abc125/tasks/abc125\\_c](https://atcoder.jp/contests/abc125/tasks/abc125_c).

## 2 Solução: código em C++

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 /*
5  left[i] = gcd(a[0], ..., a[i-1])
6  right[i] = gcd(a[i+1], ..., a[n-1])
7  */
8
9 int gcd(int a, int b){
10     int m, M;
11     while (a != 0 && b != 0){
12         m = min(a,b);
13         M = max(a,b);
14         b = M % m;
15         a = m;
16     }
17     return max(a, b);
18 }
19
20 void solve(){
21     int n;
22     cin >> n;
23     int a[n];
24     for (int i = 0; i < n; i++){
25         cin >> a[i];
26     }
27     int left[n], right[n];
28     left[0] = 0; right[n-1] = 0;
29     for (int i = 1; i < n; i++){
30         left[i] = gcd(left[i-1], a[i-1]);
31     }
32     for (int i = n-2; i >= 0; i--){
33         right[i] = gcd(right[i+1], a[i+1]);
34     }
35
36     int ans = 0;
37     for (int i = 0; i < n; i++){
38         ans = max(ans, gcd(left[i], right[i]));
39     }
40     cout << ans;
41 }
```

## 3 Demonstração

### 3.1 Lemas importantes sobre *gcd*

**Teorema 3.1.** A função  $\text{gcd} : \mathbb{N} \rightarrow \mathbb{N}$  é associativa:

$$\text{gcd}(a, \text{gcd}(b, c)) = \text{gcd}(\text{gcd}(a, b), c)$$

Ademais, ela é simétrica em relação a seus argumentos:  $\text{gcd}(a, b) = \text{gcd}(b, a)$ .

*Prova.* Pelo Teorema Fundamental da Aritmética, temos

$$\begin{aligned} a &= \prod_{1 \leq i \leq n} p_i^{\alpha_i} \\ b &= \prod_{1 \leq i \leq n} p_i^{\beta_i} \\ c &= \prod_{1 \leq i \leq n} p_i^{\gamma_i} \end{aligned}$$

em que  $p_1 < \dots < p_n$  são os  $n$ -primeiros primos e um dos três expoentes  $\alpha_n, \beta_n, \gamma_n$  é diferente de 0. Assim o lema segue da **associatividade da função min**:

$$\gcd(a, \gcd(b, c)) = \prod_{1 \leq i \leq n} p_i^{\min(\alpha_i, \min(\beta_i, \gamma_i))} = \prod_{1 \leq i \leq n} p_i^{\min(\min(\alpha_i, \beta_i), \gamma_i)} = \gcd(\gcd(a, b), c)$$

A associatividade da função min, por sua vez, pode ser provada analisando caso a caso dentre as possíveis ordenação de três números, e portanto sua demonstração é omitida aqui.  $\square$

Assim, podemos definir, para todo  $k \geq 2$ , a função  $\gcd_k : \mathbb{N}^k \rightarrow \mathbb{N}$  definida recursivamente por

**Definição 3.1.**

$$\gcd_{k+1}(a_1, \dots, a_{k+1}) = \gcd(\gcd_k(a_1, \dots, a_k), a_{k+1})$$

a qual denotaremos apenas por  $\gcd$ , ficando a quantidade de parâmetros implícita.

Como corolário, vale a seguinte generalização da associatividade:

**Corolário 3.1.1.** Para todos  $1 \leq i < n$ :

$$\gcd_n(a_1, \dots, a_n) = \gcd(\gcd_i(a_1, \dots, a_i), \gcd_{n-i}(a_{i+1}, \dots, a_n))$$

em que definimos  $\gcd_1(a) = a$ .

*Prova.* Provamos por indução sobre  $n - i$ . O caso  $n - i = 1$  segue da própria **definição** de  $\gcd_n$ . Além disso,

$$\begin{aligned} \gcd(\gcd_i(a_1, \dots, a_i), \gcd_{n-i}(a_{i+1}, \dots, a_n)) &\stackrel{\text{Definição 3.1}}{=} \gcd(\gcd_{i-1}(\gcd(a_1, \dots, a_{i-1}), a_i), \gcd_{n-i}(a_{i+1}, \dots, a_n)) \\ &\stackrel{\text{Teorema 3.1}}{=} \gcd(\gcd_{i-1}(a_1, \dots, a_{i-1}), \gcd_{n-i}(a_i, \gcd(a_{i+1}, \dots, a_n))) \\ &= \gcd(\gcd_{i-1}(a_1, \dots, a_{i-1}), \gcd_{n-i}(\gcd(a_{i+1}, \dots, a_n), a_i)) \\ &\stackrel{\text{Definição 3.1}}{=} \gcd(\gcd_{i-1}(a_1, \dots, a_{i-1}), \gcd_{n-i+1}(a_{i+1}, \dots, a_n, a_i)) \end{aligned}$$

porém, como a ordem dos argumentos de  $\gcd_k$  não importa, para qualquer  $k$  natural, então a expressão acima é simplesmente  $\gcd(\gcd_{i-1}(a_1, \dots, a_{i-1}), \gcd_{n-i+1}(a_i, \dots, a_n))$ . Ou seja, usando a hipótese de indução

$$\gcd_n(a_1, \dots, a_n) = \gcd(\gcd_i(a_1, \dots, a_i), \gcd_{n-i}(a_{i+1}, \dots, a_n)) = \gcd(\gcd_{i-1}(a_1, \dots, a_{i-1}), \gcd_{n-i+1}(a_i, \dots, a_n))$$

provando o corolário por indução.  $\square$

**Lema 3.1.** Para todos  $a, b \in \mathbb{N}$ , se  $b \neq 0$  e  $a \equiv r \pmod{b}$  então

$$\gcd(a, b) = \gcd(r, b)$$

**Teorema 3.2.** O seguinte algoritmo para calcular  $\gcd$  de dois números

```

1      int gcd(int a, int b){
2          while (a != 0 and b != 0){
3              int m = min(a, b);
4              int M = max(a, b);
5              a = M % m;
6              b = m;
7          }
8      return max(a, b);
9  }
```

termina com o resultado correto em  $O(\log \min(a, b))$ .

*Prova: **Complexidade.*** Seja  $a_i, b_i$  os valores das variáveis após a  $i$ -ésima iteração, e  $a_0 = a, b_0 = b$  seus valores iniciais.

A partir da primeira iteração temos como invariante de loop que  $a_i \leq b_i$ , uma vez que  $a_i \leftarrow M \bmod m \leq m$  e  $b_i \leftarrow m$ . Assim, podemos supor sem perda de generalidade que  $a \leq b$  inicialmente (do contrário, basta 1 iteração para valer tal desigualdade).

Após uma iteração temos  $a_{i+1} = b_i \bmod a_i$  e  $b_{i+1} = a_i$ , e logo, após duas iterações temos que

$$a_{i+2} = a_i \bmod (b_i \bmod a_i) \quad (1)$$

$$b_{i+2} = b_i \bmod a_i \quad (2)$$

Seja  $b_i = q_i a_i + r_i$ . Então temos dois casos:

**Caso 1:** Se  $r_i \leq \frac{a_i}{2}$  então

$$a_{i+2} = a_i \bmod r_i < r_i \leq \frac{a_i}{2}$$

**Caso 2:** Se  $r_i > \frac{a_i}{2}$ , então

$$a_{i+2} = a_i \bmod r_i = a_i - r_i < a_i - \frac{a_i}{2} = \frac{a_i}{2}$$

De qualquer forma temos  $a_{i+2} \leq 2^{-1} a_i$ . Logo, por indução

$$a_{2i} \leq 2^{-i} a_0$$

Como  $a_{2i}$  é um inteiro não-negativo, temos que  $i > \log_2(a_0)$  implica que  $a_{2i} < 1$ , e portanto  $a_{2i} = 0$ , o que faz que o algoritmo termine. Como supomos sem perda de generalidade  $a_0 \leq b_0$ , isso mostra que o algoritmo termina em no máximo  $2(\log_2(a_0) + 1) = 2(\log_2(\min(a, b)) + 1)$  iterações, isto é, possui complexidade  $O(\log \min(a, b))$ .  $\square$

*Prova: **Corretude.*** Precisamos mostrar apenas a seguinte invariante de loop: após  $i$ -ésima iteração

$$\gcd(a_i, b_i) = \gcd(a, b)$$

O caso inicial  $i = 0$  segue da definição dos termos iniciais da sequência, pois  $a_0 := a$  e  $b_0 := b$ . Suponha, então, que existe  $i > 0$  tal que a invariante de loop segue válida para todo  $0 \leq j \leq i$ . Conforme mostramos na parte anterior, e pela condição do loop while, temos  $0 < a_i \leq b_i$ , e portanto

$$\gcd(a_{i+1}, b_{i+1}) = \gcd(b_i \bmod a_i, a_i) = \gcd(b_i, a_i) = \gcd(a_i, b_i)$$

pelo Lema 3.1. Pela parte anterior da demonstração, após a iteração final  $i_0$  temos  $a_{i_0} = 0$  ou  $b_{i_0} = 0$ , e portanto

$$\gcd(a, b) \stackrel{\text{invariante}}{=} \gcd(a_{i_0}, b_{i_0}) = \max(a_{i_0}, b_{i_0})$$

provando a corretude do algoritmo.  $\square$

## 3.2 Demonstrando o algoritmo

**Teorema 3.3.** *O algoritmo da Seção 2 fornece a resposta correta em  $O(n \log M)$ , onde  $M = \max(a[0], \dots, a[n-1])$ .*

*Prova.* Temos que a resposta que deve ser fornecida, dado um vetor  $a[]$  de  $n$  inteiros positivos, é

$$S = \max_{b \in \mathbb{Z}_{\geq 0}} \max_{1 \leq i \leq n} \gcd(a[0], \dots, a[i-1], b, a[i+1], \dots, a[n-1])$$

Porém, temos que para todo  $b \geq 0$  inteiro:

$$\begin{aligned} \gcd(a[0], \dots, a[i-1], b, a[i+1], \dots, a[n-1]) &= \gcd(a[0], \dots, a[i-1], a[i+1], \dots, a[n-1], b) \\ &\stackrel{\text{Definição 3.1}}{=} \gcd(\gcd(a[0], \dots, a[i-1], a[i+1], \dots, a[n-1]), b) \\ &\leq \gcd(a[0], \dots, a[i-1], a[i+1], \dots, a[n-1]) \\ &\stackrel{\text{Corolário 3.1.1}}{=} \gcd(\gcd(a[0], \dots, a[i-1]), \gcd(a[i+1], \dots, a[n-1])) \end{aligned}$$

com **caso de igualdade quando, por exemplo**,  $b = \gcd(a[0], \dots, a[i-1], a[i+1], \dots, a[n-1])$ .

Assim, podemos definir os seguintes vetores  $L[n]$ ,  $R[n]$ :

$$\begin{aligned} L[0] &= 0 \\ L[i+1] &= \gcd(L[i], a[i]) \\ R[n-1] &= 0 \\ R[i-1] &= \gcd(R[i], a[i]) \end{aligned}$$

e pela associatividade (Teorema 3.1) teremos

$$\begin{aligned} L[i] &= \gcd(a[0], \dots, a[i-1]) \\ R[i] &= \gcd(a[n-1], \dots, a[n-i]) \end{aligned}$$

Ademais, novamente pelo Teorema 3.1, podemos calcular ambos os vetor em  $O(n) \cdot O(\gcd) = O(n \log M)$  onde  $M = \max(a[0], \dots, a[n-1])$ .

Por fim, teremos que

$$\begin{aligned} S &= \max_{0 \leq i \leq n-1} \gcd(\gcd(a[0], \dots, a[i-1]), \gcd(a[i+1], \dots, a[n-1])) \\ &= \max_{0 \leq i \leq n-1} \gcd(L[i], R[n-i]) \end{aligned}$$

o que novamente pode ser calculado em  $O(n \log M)$ .

Portanto, o código da Seção 2, que é exatamente a implementação do algoritmo descrito, fornece uma solução correta em  $O(n \log M)$ .  $\square$