Maximizing the Profit

locked

by utkarsh028

A hardware company is building a machine with exactly 3 hardware components. There are many components available, and the profit factor of each component is known. The profit obtained by the machine is the product of the profit factors of the 3 hardware components used to build that machine.

However, there is a catch. Three different components with numbers $i < j < k$ can be used to build the machine if and only if their profit factors are $p_i < p_j < p_k$.

Calculate the maximum possible profit that a valid machine consisting of three components can have, or decide that it's impossible to build any machine. Complete the function `maximumProfit` which takes in the integer array denoting the profit factors of all components and returns a single integer denoting the answer.

Input Format

The first line contains a single integer n, denoting the number of available components. Components are numbered 0 to $n-1$.
The second line contains n space-separated integers $p_0, p_1, \ldots, p_{n-1}$, i.e the integer array p denoting the profit factors of the components.

Constraints

- $1 \le n \le 3 \cdot 10^5$
- $-10^6 \le p_i \le 10^6$

Output Format

Print -1 if it's impossible to build any machine. Otherwise, print a single integer denoting the maximum possible profit that a valid machine consisting of 3 components can have.

Sample Input 0

```
7
10 2 -10 6 1 0 8
```

Sample Output 0

```
96
```

Explanation 0

We use 1-based indexing for this explanation.There are $n = 7$ hardware components available.

2nd, 4th and 7th component can be used to build the machine since $p_2 < p_4 < p_7$ (because $p_2 = 2$, $p_4 = 6$ and $p_7 = 8$). The profit obtained by using the machine is $2 \cdot 6 \cdot 8 = 96$, which is maximum possible.

Submissions: 748
Max Score: 50

Difficulty: Medium

Rate This Challenge:

☆ ☆ ☆ ☆ ☆

More

Current Buffer (saved locally, editable)

C++14

```cpp
#include <iostream>
#include <vector>
#include <set>
#include <algorithm>
#include <limits>
typedef long long int int64;

int64 maximumProfit(std::vector<int64>& p) {

    std::multiset<int64> s1;
    std::multiset<int64> s2;

    for(size_t i=0; i < p.size(); i++)
        s2.insert(p[i]);

    int64 ans = std::numeric_limits<int64>::min();
    for(size_t i=0; i < p.size(); i++)
    {
        s2.erase(s2.lower_bound(p[i]));
        if(i==0)
        {
            s1.insert(p[i]);
            continue;
        }

        std::vector<int64> v1;
        std::vector<int64> v2;

        auto it=s1.lower_bound(p[i]);
        if(it!=s1.begin())
        {
            it--;
            v1.push_back((*s1.begin()));
            v1.push_back((*it));
        }

        it=s2.upper_bound(p[i]);
        if(it!=s2.end())
        {
            v2.push_back((*it));
            v2.push_back((*s2.rbegin()));
        }

        for(auto el:v1)
            for(auto el2:v2)
                ans = std::max(ans,el*el2*p[i]);

        s1.insert(p[i]);
    }
    return
    (ans == std::numeric_limits<int64>::min())? -1: ans;
}

int main()
{
    int64 n; std::cin >> n;
    std::vector<int64> vec(n);
    for(auto &it: vec) std::cin >> it;

    std::cout << maximumProfit(vec) << std::endl;
    return 0;
}


```

Line: 1 Col: 1

Upload Code as File Test against custom input Run Code Submit Code