CODESHOWS TREE

Let us first show that in an optimal Codeshows tree only the root or a leaf can hold a value other than one of `d[i]`. Suppose that we have an inner vertex different from the root which holds a number `X` not equal to any of `d[i]`. Then we can exclude this vertex from the tree and tie its children to its parent without violating any of the tree's properties.

Hence, our tree consists of the root, vertices with numbers `d[i]` tied to each other or to the root, and leaves, which are tied to vertices with numbers `d[i]` and contain these numbers' prime factorizations. The exception is the case when one of `d[i]` is written in root itself, and the case when some `d[i]`'s are prime themselves. Also note that in general case it's easy to count how many leaves the tree will have. This count is equal to the sum of exponents of primes in prime factorizations of those `d[i]`'s which are the children of the root.

Since `n <= 8`, we can build all trees which satisfy the observations we made. Let's sort numbers `d[i]` in descending order and recursively choose a parent for each of them from the vertices already present in the tree. Of course, tying a number `X` to some vertex `v` is only possible if the product of `X` and the numbers in children of `v` divides the number in `v` itself. For `d[1]`, we have a choice — we can make it the root of the tree or a child of the root (in this case the root will hold a nominal infinity which is divisible by any number). For every next `d[i]`, the choice is whether to tie it to the root or a vertex containing one of the previous numbers. Therefore, we only consider `O(N!)` trees in total.

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

using namespace std;

int primeDivisors[8];
int primeCount;
int rec(int cur, vector<long long> &a, vector<long long> &tree, int rootChildrenNum, int
rootChildrenSum)
{
    if (cur == a.size())
        return rootChildrenSum + (rootChildrenNum > 1) + (int)a.size() - primeCount;

    int best = (int)1e9;
    for(int i = 0; i < tree.size(); i++)
        if (tree[i] % a[cur] == 0)
        {
            tree[i] /= a[cur];
            tree.push_back(a[cur]);
            best = min(best, rec(cur + 1, a, tree, rootChildrenNum + (i == 0),
                rootChildrenSum + (i == 0 ? primeDivisors[cur] : 0)));
            tree[i] *= a[cur];
            tree.pop_back();
        }
    return best;
}

int main()
{
    int n, i, j;
    cin >> n;
    vector<long long> a(n);
    for(i = 0; i < n; i++)
        cin >> a[i];
    sort(a.begin(), a.end(), greater<long long>());

    for(i = 0; i < n; i++)
    {
        long long x = a[i];
        for(j = 2; 1LL * j * j <= x; j++)
            while(x % j == 0)
            {
                ++primeDivisors[i];
                x /= j;
            }
        if (x > 1)
            ++primeDivisors[i];
        if (primeDivisors[i] == 1)
            primeCount++;
    }

    vector<long long> tree;
    tree.push_back(0);
    cout << rec(0, a, tree, 0, 0) << endl;

    return 0;
}
```