2226. Maximum Candies Allocated to K Children

You are given a **0-indexed** integer array `candies`. Each element in the array denotes a pile of candies of size `candies[i]`. You can divide each pile into any number of **sub piles**, but you **cannot** merge two piles together.

You are also given an integer `k`. You should allocate piles of candies to `k` children such that each child gets the **same** number of candies. Each child can be allocated candies from **only one** pile of candies and some piles of candies may go unused.
Return *the* ***maximum number of candies*** *each child can get.*

Example 1:

```
Input: candies = [5,8,6], k = 3
Output: 5
Explanation: We can divide candies[1] into 2 piles of size 5 and 3, and candies[2]
into 2 piles of size 5 and 1. We now have five piles of candies of sizes 5, 5, 3,
5, and 1. We can allocate the 3 piles of size 5 to 3 children. It can be proven
that each child cannot receive more than 5 candies.
```

Example 2:

```
Input: candies = [2,5], k = 11
Output: 0
Explanation: There are 11 children but only 7 candies in total, so it is impossible
to ensure each child receives at least one candy. Thus, each child gets no candy
and the answer is 0.
```

Constraints:

- $1 \le candies.length \le 10^5$
- $1 \le candies[i] \le 10^7$
- $1 \le k \le 10^{12}$

Overview

We are given an array called `candies`, where each element `candies[i]` represents the number of candies in the `i-th` pile. We also have an integer `k`, which denotes the number of children we must give candies to. Our goal is to find the greatest number of candies each child can get, following these rules:

- Each child must get the same number of candies.
- Each child's candies must come from just one pile. We can divide candies from a pile among multiple children, but we cannot combine candies from different piles for one child.

Note that we do not have to use all the candies from any given pile—some candies from a pile or even entire piles may remain unused.

To better understand the task, let's go through an example.

2226. Maximum Candies Allocated to K Children

```
/*
    Binary search on the answer
    Time complexity: O(n+n log hi)
    Space complexity: O(1)
*/
typedef long long ll;
class Solution {
    public:
        int maximumCandies(vector<int>& candies_piles, ll k) {
            // Get the highest value in the list of candies piles
            int hi=*std::max_element(candies_piles.begin(),candies_piles.end());

            // Function to check if we can allocate amount of value of candies,
            // `candies_number` to k children
            auto is_possible=[&](int candies_number)->bool{
                ll nb_candies=0;
                for(auto& candies: candies_piles){
                    nb_candies+=candies*1ll/candies_number;
                    if(nb_candies>=k) return true;
                }
                return false;
            };

            // Using binary search, look for the greatest number of candies that
            // could be possible to be allocate to the k children
            int lo=1;
            while(lo<=hi){
                int candies_number=(lo+hi)>>1;
                // If it is possible, search further
                if(is_possible(candies_number)) lo=candies_number+1;

                // Otherwise, go backward
                else hi=candies_number-1;
            }

            // -1, because it initialized to 1
            return lo-1;
        }
};
```