<https://leetcode.com/problems/maximum-candies-allocated-to-k-children/description/>

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 take at most 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 <= candies.length <= 10^5

1 <= candies[i] <= 10^7

1 <= k <= 10^12

**Attempt 1: 2024-12-08**

**Solution 1: Binary Search + Greedy (10 min)**

**Two styles same as** [L1802.Maximum Value at a Given Index in a Bounded Array (Ref.L410)](note://WEB7d060d486a37a8c92696be41f18fb27c)

**Style 1: canDistribute**

class Solution {

    public int maximumCandies(int[] candies, long k) {

        int lo = 1;

        int hi = 1;

        for(int c : candies) {

            hi = Math.max(hi, c);

        }

        // Find upper boundary (since return the maximum

        // number of candies each child can get)

        while(lo <= hi) {

            int mid = lo + (hi - lo) / 2;

            // If find a valid 'mid' as the maximum

            // number of candies each child can get,

            // then move left boundary 'lo' forward

            // to 'mid + 1', which attempts an even

            // larger number, otherwise move right

            // boundary 'hi' backward to 'mid - 1'

            // which attempst a smaller number

            if(canDistribute(candies, k, mid)) {

                lo = mid + 1;

            } else {

                hi = mid - 1;

            }

        }

        return lo - 1;

    }

    private boolean canDistribute(int[] candies, long k, int maxNumOfCandiesEachChildGet) {

        // Use 'long' type because compare with k range 1 <= k <= 10^12

        long count = 0;

        // For each pile, calculate how many children can receive x candies

        for(int c : candies) {

            count += c / maxNumOfCandiesEachChildGet;

            // Sum up these counts and check if the total is >= k

            if(count >= k) {

                return true;

            }

        }

        return false;

    }

}

Time Complexity: O(nlogn)

Space Complexity: O(1)

**Style 2: cannotDistribute**

class Solution {

    public int maximumCandies(int[] candies, long k) {

        int lo = 1;

        int hi = 1;

        for(int c : candies) {

            hi = Math.max(hi, c);

        }

        // Find upper boundary (since return the maximum

        // number of candies each child can get)

        while(lo <= hi) {

            int mid = lo + (hi - lo) / 2;

            // If cannot find a valid 'mid' as the maximum

            // number of candies each child can get, then

            // move right boundary 'hi' backward to 'mid - 1',

            // which attempts an even smaller number,

            // otherwise move left boundary 'lo' forward to

            // 'mid + 1' which attempst a larger number

            if(cannotDistribute(candies, k, mid)) {

                hi = mid - 1;

            } else {

                lo = mid + 1;

            }

        }

        return lo - 1;

    }

    private boolean cannotDistribute(int[] candies, long k, int maxNumOfCandiesEachChildGet) {

        // Use 'long' type because compare with k range 1 <= k <= 10^12

        long count = 0;

        // For each pile, calculate how many children can receive x candies

        for(int c : candies) {

            count += c / maxNumOfCandiesEachChildGet;

        }

        // Sum up these counts and check if the total still < k,

        // if still < k, return true means cannot distribute

        return count < k;

    }

}

Time Complexity: O(nlogn)

Space Complexity: O(1)

**Refer to chatGPT**

This is a **binary search** problem. The goal is to find the largest number of candies xxx such that it is possible to distribute x candies to k children evenly.

**Key Idea:**

Binary Search:

The smallest possible x is 1 (each child gets at least one candy).

The largest possible x is max⁡(candies), as no child can receive more candies than the largest pile.

For each mid-point x in the binary search, check if x candies can be evenly distributed to k children.

Feasibility Check:

Calculate the total number of children that can receive x candies from all piles.

If this total is ≥k, then x is feasible.

class Solution {

public int maximumCandies(int[] candies, long k) {

int left = 1, right = 0;

// Determine the maximum candies in a pile

for (int candy : candies) {

right = Math.max(right, candy);

}

int result = 0;

// Binary search to find the maximum x

while (left <= right) {

int mid = left + (right - left) / 2;

if (canDistribute(candies, k, mid)) {

result = mid; // Update result and try for a larger x

left = mid + 1;

} else {

right = mid - 1; // Decrease x

}

}

return result;

}

private boolean canDistribute(int[] candies, long k, int x) {

long count = 0;

for (int candy : candies) {

count += candy / x; // Number of children that can receive x candies from this pile

}

return count >= k; // Check if we can satisfy k children

}

}

**Explanation:**

Binary Search:

Start with the range [1, max⁡(candies)].

Use binary search to find the maximum x such that x candies can be evenly distributed to k children.

canDistribute **Function**:

For each pile, calculate how many children can receive x candies.

Sum up these counts and check if the total is ≥k.

**Refer to**

[L410.Split Array Largest Sum (Ref.L1011,L704,L1482)](note://WEB4d9d6f26a965381b137615128e2648be)

[L1011.Capacity To Ship Packages Within D Days (Ref.L410,L1482)](note://WEB87ca8f9dd9a14fb4535856a4e0f8f4b1)

[L1283.Find the Smallest Divisor Given a Threshold (Ref.L410,L1011,L1482)](note://76952362C4844CB3B68E28EA09F2EE58)

[L1482.Minimum Number of Days to Make m Bouquets (Ref.L410,L1011)](note://WEBd5ce394160da45a79e92dbbc806111ec)

[L1802.Maximum Value at a Given Index in a Bounded Array (Ref.L410)](note://WEB7d060d486a37a8c92696be41f18fb27c)

[L2064.Minimized Maximum of Products Distributed to Any Store (Ref.L410)](note://WEB2ca95ff77a9fee4bcced70f3e62a41df)