EGOI 2026 Editorial - Cakes

Task Author: Hazem Issa

Subtask 1: $N = 1$

In this subtask, there is one topping type with count a_1. Since there is only one type, the tastiness of a cake equals the number of toppings placed on it (all are the same type).

To make K_j cakes with equal tastiness b, each cake must contain exactly b toppings, so:

$$a_1 = K_j \cdot b.$$

Thus, the answer is **YES** if and only if a_1 is divisible by K_j which is true if $a_1 \bmod Ki_j = 0$.

Subtask 2: $Q = 1,\ K_j = 2$

In this subtask, we need two cakes with equal tastiness. Let m be the amount of the topping that we have most of, and let s be the amount of the topping we have the second most of or 0 for $N = 1$. We do a case-analysis on whether m is even or odd.

Case 1: m is even

If m is even, we can distribute all the toppings equally among both cakes. This results in a common tastiness of $b = m/2$:

The topping type with count m can contribute b copies to each cake, making both cakes reach tastiness at least b. In addition, the common tastiness will not be larger than b, since any other topping will appear at most $m/2 = b$ times on each cake.

Thus, in this case, the answer is always YES.

Case 2: m is odd

If m is odd, the smallest possible b that can fit all of topping m into two cakes without exceeding b is

$$b = \left\lceil \frac{m}{2} \right\rceil.$$

With this b, the largest type can create only **one** full block of size b, so only one cake can "reach" tastiness b using that type. To make the other cake also reach

b, we need another type with at least b copies:

$$s \geq b \quad \Longleftrightarrow \quad s \geq \left\lceil \frac{m}{2} \right\rceil .$$

Solution

- If m is even: **YES**
- If m is odd: **YES** if and only if $s \geq \lceil m/2 \rceil$, else **NO**

Subtask 3: $Q \leq 5$, small limits $(N \leq 1000),\ (a_i \leq 1000)$

Let $m = \max a_i$. We try all possible values for the common tastiness $b \in [1..m]$ for each query. To check whether a fixed b is possible, we need to make the following checks:

1. **No type exceeds b on any cake (capacity check).**
   A topping type i with count a_i can be distributed across K_j cakes with at most b per cake if and only if $a_i \leq K_j \cdot b$.
   It suffices to check the maximum:

   $$m \leq K_j \cdot b.$$

2. **Every cake achieves tastiness b (winner-block check).**
   A cake that has tastiness b needs at least one topic with exactly b times on it.
   Type i can supply $\lfloor a_i/b \rfloor$ disjoint blocks of size b, each block can serve as the "winner" for one cake.
   So we need:

   $$T(b) = \sum_{i=1}^{N} \left\lfloor \frac{a_i}{b} \right\rfloor \geq K_j.$$

A cake has tastiness b only if *some* topping type appears exactly b times on it and no other topic appears more often. Checking both of the above conditions suffices. For small limits, we can test these for all b for every query in $\mathcal{O}(NQ \max a_i)$.

Solution

For each query, iterate through $b = 1..m$ and check:

- $m \leq K_j b$
- $T(b) \geq K_j$

If any b passes, answer **YES**, otherwise **NO**.

The runningtime of this subtask is $\mathcal{O}(NQ \max(a_i))$.

Subtask 4: $Q \leq 5$

From now on, we want to avoid trying all b.

Observe that due to the capacity check $m \leq K_j \cdot b$, any valid b must satisfy:

$$b \geq \left\lceil \frac{m}{K_j} \right\rceil.$$

Furthermore, the number of winner blocks $T(b) = \sum_{i=1}^{N} \left\lfloor \frac{a_i}{b} \right\rfloor$ is **non-increasing** as b increases: the larger b, the fewer winner blocks we get. So among all feasible b, the best chance to have $T(b) \geq K_j$ is at the smallest one that passes the capacity check: $b_j = \left\lceil \frac{m}{K_j} \right\rceil$.

Solution

It is enough for each query to check just b_j: Output YES if and only if

$$T(b_j) = \sum_{i=1}^{N} \left\lfloor \frac{a_i}{b_j} \right\rfloor \geq K_j.$$

The running time of this subtask is $\mathcal{O}(NQ)$.

Full Solution

Observation 1 (each query depends only on b_j)

Each query K_j only needs:

- $b_j = \left\lceil \frac{m}{K_j} \right\rceil$
- $T(b_j) = \sum \lfloor a_i / b_j \rfloor$

Thus, we do not need to re-compute $T(b_j)$ for every query: Multiple queries may share the same value of b_j. Thus, we can **save** already-computed values of $T(b)$ and reuse them.

Observation 2 (few distinct b_j)

Over the different K_j, the value $\left\lceil \frac{m}{K_j} \right\rceil$ takes only $\mathcal{O}(\sqrt{m})$ distinct values.
So we will compute $T(b)$ only $\mathcal{O}(\sqrt{m})$ times.

Solution

For each query:

1. Compute $b_j = \left\lceil \frac{m}{K_j} \right\rceil$.
2. If $T(b_j)$ was not computed yet, compute it once in $O(N)$.
3. Answer **YES** if and only if $T(b_j) \geq K_j$.

The total running time is

$$\mathcal{O}\big(N \cdot \text{Nr of distinct } b_j + Q\big) = \mathcal{O}(N\sqrt{m} + Q).$$