Minimized Maximum Of Products Distributed To Any Store

solved by	Senan
Platform	LeetCode
difficulty	Medium
# Serial	2064
tags	Binary Search
language	C++
solved on	@14/11/2024
link	https://leetcode.com/problems/minimized-maximum-of-products-distributed-to-any-store/description/
Completion	☑

Intuition

The problem is to distribute products across stores such that the maximum number of products any store has is minimized. To achieve this, we can consider using binary search on the maximum number of products any store can have, and then check if this maximum is feasible with the given number of stores (`n`).

Approach

1. **Binary Search on the Maximum Products per Store**:
   - Define `low` as 1 (the smallest possible maximum for any store) and `high` as the maximum possible products per store, set to a high number (`1e5` here), as an upper bound.
   - Use binary search to narrow down the feasible maximum number of products per store.
2. **Helper Function `getNumStores`**:
   - This function calculates the number of stores needed if we set the maximum number of products each store can handle to `numProducts`.
   - For each quantity, determine the number of stores required using `ceil(quantity / numProducts)`, and sum up these values.
3. **Binary Search Condition**:
   - For each midpoint (`mid`), use `getNumStores(mid, quantities)` to check if the number of stores required is less than or equal to `n`.
   - If `getNumStores(mid, quantities) <= n`, then `mid` is a feasible solution, so continue the search on the left half by setting `high = mid - 1`.
   - Otherwise, search in the right half by setting `low = mid + 1`.
4. **Return Result**:
   - The loop exits when `low` is the minimum feasible value for the maximum products per store.

Complexity

Time Complexity:

- **Binary Search**: **O(**$\log(\text{high})$**)**, where `high` is the upper bound (`1e5`).
- **Helper Function**: For each midpoint in the binary search, `getNumStores` takes **O(m)** time, where `m` is the number of elements in `quantities`.
- **Overall Complexity**: **O(m** $\log(\text{high})$**)**.

Space Complexity:

- **O(1)**, as we only use a few variables for calculations.

Code

```
class Solution {
    int getNumStores(int numProducts, vector<int>& quantities){
        int stores = 0;
        for(auto &quantity: quantities)
            stores += ceil(float(quantity) / numProducts);
        return stores;
    }
public:
    int minimizedMaximum(int n, vector<int>& quantities) {
        int low = 1, high = 1e5;
        while (low <= high) {
            int mid = (low + high) / 2;
            if (getNumStores(mid, quantities) <= n) high = mid - 1;
            else low = mid + 1;
        }
        return low;
    }
};
```