Most Beautiful Item For Each Query

solved by	Senan
Platform	LeetCode
difficulty	Medium
Serial	2070
tags	Sorting
language	C++
solved on	@12/11/2024
link	https://leetcode.com/problems/most-beautiful-item-for-each-query/description/
Completion	☑

Intuition

We want to find the maximum beauty value for a given price, but prices and corresponding beauty values can vary, so a binary search can efficiently find the maximum beauty for each query. To optimize this, we store the maximum beauty seen so far at each unique price point in a separate list, allowing us to quickly look up the maximum beauty value for any query.

Approach

1. **Sort Items**: First, sort the `items` array by price.
2. **Build `store`**: Create a `store` list where each entry contains a unique price and the maximum beauty up to that price. This allows us to use binary search later to efficiently find the best beauty value for each price in `queries`.
    - Traverse the sorted `items`, and for each unique price, update the maximum beauty found so far.
3. **Binary Search for Queries**: For each price in `queries`, use binary search on the `store` list to find the highest possible beauty for that price or less.
4. **Return Results**: Store the results for each query in an answer array and return it.

Complexity

Time Complexity:

- Sorting `items` takes **O(n log n)**, where **n** is the number of items.
- Building the `store` list takes **O(n)**, as each item is processed once.
- For each query, binary search on the `store` takes **O(log n)**, so for **q** queries, it takes **O(q log n)**.
- Overall time complexity: **O(n log n + q log n)**.

Space Complexity:

- The `store` list requires **O(n)** space to store the unique price-beauty pairs.
- Answer array requires **O(q)** space for **q** queries.

- Overall space complexity: **O(n + q)**.

Code

```
class Solution {
    int binarySearch(int num, vector<pair<int, int>>& store) {
        int low = 0;
        int high = store.size() - 1;

        while (low <= high) {
            int mid = low + (high - low) / 2;
            if (store[mid].first <= num) {
                low = mid + 1;
            } else {
                high = mid - 1;
            }
        }

        return store[high].second;
    }

public:
    vector<int> maximumBeauty(vector<vector<int>>& items, vector<int>& queries) {
        vector<int> answer;
        vector<pair<int, int>> store;

        // Sort items by price (first element)
        sort(items.begin(), items.end());
        store.push_back({INT_MIN, 0});

        // Build `store` with maximum beauty values up to each unique price
        for (int i = 0; i < items.size(); i++) {
            int maxPrice = max(items[i][1], store.back().second);
            if (store.back().first < items[i][0]) {
                store.push_back({items[i][0], maxPrice});
            } else {
                store.back().second = maxPrice;
            }
        }

        // Process each query using binary search
        for (auto query : queries) {
            answer.push_back(binarySearch(query, store));
        }
        return answer;
    }
};
```