2070. Most Beautiful Item for Each Query

You are given a 2D integer array `items` where `items[i] = [pricei, beautyi]` denotes the **price** and **beauty** of an item respectively.

You are also given a **0-indexed** integer array `queries`. For each `queries[j]`, you want to determine the **maximum beauty** of an item whose **price** is **less than or equal** to `queries[j]`. If no such item exists, then the answer to this query is `0`.

Return *an array* `answer` *of the same length as* `queries` *where* `answer[j]` *is the answer to the* `j`th *query*.

Example 1:

```
Input: items = [[1,2],[3,2],[2,4],[5,6],[3,5]], queries = [1,2,3,4,5,6]
Output: [2,4,5,5,6,6]
Explanation:
- For queries[0]=1, [1,2] is the only item which has price <= 1. Hence, the answer for this query is 2.
- For queries[1]=2, the items which can be considered are [1,2] and [2,4].
  The maximum beauty among them is 4.
- For queries[2]=3 and queries[3]=4, the items which can be considered are [1,2], [3,2], [2,4], and [3,5].
  The maximum beauty among them is 5.
- For queries[4]=5 and queries[5]=6, all items can be considered.
  Hence, the answer for them is the maximum beauty of all items, i.e., 6.
```

Example 2:

```
Input: items = [[1,2],[1,2],[1,3],[1,4]], queries = [1]
Output: [4]
Explanation:
The price of every item is equal to 1, so we choose the item with the maximum beauty 4.
Note that multiple items can have the same price and/or beauty.
```

Example 3:

```
Input: items = [[10,1000]], queries = [5]
Output: [0]
Explanation:
No item has a price less than or equal to 5, so no item can be chosen.
Hence, the answer to the query is 0.
```

Constraints:

- `1 <= items.length, queries.length <= 10^5`
- `items[i].length == 2`
- `1 <= pricei, beautyi, queries[j] <= 10^9`

2070. Most Beautiful Item for Each Query

```
/*
   Sorting items+prefix max + binary search
   Time complexity: O(3n+ nlogn + mlogn) = O((n+m)logn)
   Space complexity: O(3n)
*/
typedef std::vector<int> vi;
typedef std::vector<vi> vvi;
class Solution {
   public:
      vi maximumBeauty(vvi& items, vi& queries){
         // Sort the items base on prices
         std::sort(items.begin(),items.end());

         // Create two independant arrays for prices and beauties
         // to facilate searching
         vi prices,beauties;
         for(auto& item: items){
            prices.push_back(item[0]);
            beauties.push_back(item[1]);
         };

         int n=beauties.size();

         // Create prefix max array for beauties
         // to get the max beauty of each query in constant time
         vi prefix_max(n);
         prefix_max[0]=beauties[0];
         for(int i=1;i<n;++i) prefix_max[i]=std::max(prefix_max[i-1],beauties[i]);

         vi ans;
         // For each price query
         for(auto& q: queries){
            int max_beauty=0; // Its max beauty is by default 0
            // Perform an upper bound binary search on prices
            int i=std::upper_bound(prices.begin(),prices.end(),q)-prices.begin();
            // Because, we're looking for the upper bound
            // the max beauty is at index (i-1)
            if(i-1>=0) max_beauty=prefix_max[i-1];
            ans.push_back(max_beauty); // add the max beauty of the query q to the answer
         }
         return ans;
      }};
```