Problem 3645: Maximum Total from Optimal Activation Order

Problem Information

Difficulty: Medium
Acceptance Rate: 32.34%
Paid Only: No
Tags: Array, Two Pointers, Greedy, Sorting, Heap (Priority Queue)

Problem Description

You are given two integer arrays `value` and `limit`, both of length `n`.

Initially, all elements are **inactive**. You may activate them in any order.

* To activate an inactive element at index `i`, the number of **currently** active elements must be **strictly less** than `limit[i]`. * When you activate the element at index `i`, it adds `value[i]` to the **total** activation value (i.e., the sum of `value[i]` for all elements that have undergone activation operations). * After each activation, if the number of **currently** active elements becomes `x`, then **all** elements `j` with `limit[j] <= x` become **permanently** inactive, even if they are already active.

Return the **maximum** **total** you can obtain by choosing the activation order optimally.

Example 1:

Input: value = [3,5,8], limit = [2,1,3]

Output: 16

Explanation:

One optimal activation order is:

Step | Activated `i` | `value[i]` | Active Before `i` | Active After `i` | Becomes Inactive `j` | Inactive Elements | Total ---|---|---|---|---|---|---|--- 1 | 1 | 5 | 0 | 1 | `j = 1` as `limit[1] = 1` | [1] | 5 2 | 0 | 3 | 0 | 1 | - | [1] | 8 3 | 2 | 8 | 1 | 2 | `j = 0` as `limit[0] = 2` | [0, 1] | 16 Thus, the maximum possible total is 16.

Example 2:

Input: value = [4,2,6], limit = [1,1,1]

Output: 6

Explanation:

One optimal activation order is:

Step | Activated `i` | `value[i]` | Active Before `i` | Active After `i` | Becomes Inactive `j` | Inactive Elements | Total ---|---|---|---|---|---|---|--- 1 | 2 | 6 | 0 | 1 | `j = 0, 1, 2` as `limit[j] = 1` | [0, 1, 2] | 6 Thus, the maximum possible total is 6.

Example 3:

Input: value = [4,1,5,2], limit = [3,3,2,3]

Output: 12

Explanation:

One optimal activation order is:■■■■■■■■**■■■■■■■■**

Step | Activated `i` | `value[i]` | Active Before `i` | Active After `i` | Becomes Inactive `j` | Inactive Elements | Total ---|---|---|---|---|---|---|--- 1 | 2 | 5 | 0 | 1 | - | [ ] | 5 2 | 0 | 4 | 1 | 2 | `j = 2` as `limit[2] = 2` | [2] | 9 3 | 1 | 1 | 1 | 2 | - | [2] | 10 4 | 3 | 2 | 2 | 3 | `j = 0, 1, 3` as `limit[j] = 3` | [0, 1, 2, 3] | 12 Thus, the maximum possible total is 12.

Constraints:

* `1 <= n == value.length == limit.length <= 105` * `1 <= value[i] <= 105` * `1 <= limit[i] <= n`

Code Snippets

C++:

```
class Solution {
public:
long long maxTotal(vector<int>& value, vector<int>& limit) {

}
};
```

Java:

```
class Solution {
public long maxTotal(int[] value, int[] limit) {

}
}
```

Python3:

```
class Solution:
def maxTotal(self, value: List[int], limit: List[int]) -> int:
```