🧠 LeetCode 1636 – Sort Array by Increasing Frequency

📌 Problem Statement:

Given an array `nums`, sort it in:

- **increasing order of frequency**
- If frequency is same, then in **decreasing order of value**

💡 Thought Process (By Priyansh 👑)

🔧 Step-by-step Logic:

1. **Frequency Count**:
    - Use `unordered_map` to store each number and its frequency.

    ```
    for(int num : nums){
        mp[num]++;
    }
    ```

2. **Custom Min-Heap**:
    - Push `{frequency, value}` into `priority_queue` using custom comparator.
    - Comparator ensures:
        - Lower frequency comes first
        - If frequency is same, higher value comes first

    ```
    struct comp {
        bool operator()(pair<int, int>& a, pair<int, int>& b) {
    ```

```
        if (a.first == b.first)
            return a.second < b.second; // prefer higher value
        return a.first > b.first;       // lower freq comes first
    }
};
```

3. **Build Answer**:
    - While priority queue has elements, pop top
    - Push its value into answer `frequency` number of times.

✅ Final Code (Tera Original Code):

```
class Solution {
public:
    struct comp {
        bool operator()(pair<int, int>& a, pair<int, int>& b) {
            if (a.first == b.first)
                return a.second < b.second; // prefer higher value if same frequency
            return a.first > b.first;       // min-heap based on frequency
        }
    };

    vector<int> frequencySort(vector<int>& nums) {
        typedef pair<int,int> p;
        unordered_map<int,int>mp;

        // mapping done: every value has a frequency
        for(int num:nums){
            mp[num]++;
        }

        vector<int>ans;
        priority_queue<p,vector<p>,comp> pq;
        // push {frequency, value} into min-heap
        for(auto x:mp){
            int value = x.first;
            int frequency = x.second;
            pq.push({frequency,value});
        }

        // build final sorted answer
        while(pq.size()>0){
            int value = pq.top().second;
            int frequency = pq.top().first;
            for(int i=0;i<frequency;i++){
                ans.push_back(value);
            }
            pq.pop();
        }

        return ans;
    }
};
```

🧪 Dry Run Example:

Input: `[2,3,1,3,2]`

Frequencies:

- 2 → 2 times
- 3 → 2 times
- 1 → 1 time

Min-Heap Push:

- `{2,2}` , `{2,3}` , `{1,1}`

Pop order (based on comparator):

1. `{1,1}` → `1`

2. `{2,3}` → `3 3`
3. `{2,2}` → `2 2`

✅ **Output:** `[1,3,3,2,2]`

⏱️ Time & Space Complexity:

Operation	Complexity
Frequency map build	`O(n)`
Heap operations	`O(n log n)`
Output construction	`O(n)`
Total Time	`O(n log n)`
Space Used	`O(n)`
