Longest Harmonious Subsequence

Author:	Prakash JC

Problem Statement

We define a harmonious array as an array where the difference between its maximum value and its minimum value is **exactly** `1`.

Given an integer array `nums`, return the length of its longest harmonious subsequence among all its possible subsequences.

Example 1:

Input: nums = [1,3,2,2,5,2,3,7]

Output: 5

Explanation: The longest harmonious subsequence is `[3,2,2,2,3]`.

Example 2:

Input: nums = [1,2,3,4]

Output: 2

Explanation: The longest harmonious subsequences are `[1,2]`, `[2,3]`, and `[3,4]`, all of which have a length of 2.

Example 3:

Input: nums = [1,1,1,1]

Output: 0

Explanation: No harmonic subsequence exists.

Editorial

Brute Force Approach

Approach:

- Generate **all possible subsequences** using recursion or bitmasking.
- For each subsequence:
  - Find `max` and `min` values.
  - If `max - min == 1`, check and update maximum length.
- Return the longest length found.

Code

C++:

```
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    int findLHS(vector<int>& nums) {
        int n = nums.size(), maxLen = 0;
        int total = 1 << n;  // 2^n possible subsequences

        for (int mask = 1; mask < total; ++mask) {
            vector<int> subseq;
            for (int i = 0; i < n; ++i) {
                if (mask & (1 << i)) subseq.push_back(nums[i]);
            }
            int mini = *min_element(subseq.begin(), subseq.end());
            int maxi = *max_element(subseq.begin(), subseq.end());
            if (maxi - mini == 1) {
                maxLen = max(maxLen, (int)subseq.size());
            }
        }
        return maxLen;
    }
};

int main() {
    vector<int> nums = {1,3,2,2,5,2,3,7};
    Solution s1;
    cout << s1.findLHS(nums) << endl;
    return 0;
}
```

Python: Usually not implemented — impractical due to exponential growth.

Complexity Analysis:

- **Time Complexity**: `O(2^n * n)` — exponential due to all subsequences.
- **Space Complexity**: `O(n)` for temporary subsequence storage.

Better Approach (Sorting + Scanning)

Approach

- Sort the array.
- Use two pointers to maintain a window:
    - Move `start` when the difference between current and start > 1.
    - If `max - min == 1`, update the result.

- This avoids generating subsequences.

Code

C++:

```
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    int findLHS(vector<int>& nums) {
        sort(nums.begin(), nums.end());
        int start = 0, maxLen = 0;
        for (int end = 0; end < nums.size(); ++end) {
            while (nums[end] - nums[start] > 1) ++start;
            if (nums[end] - nums[start] == 1)
                maxLen = max(maxLen, end - start + 1);
        }
        return maxLen;
    }
};

int main() {
    vector<int> nums = {1,3,2,2,5,2,3,7};
    Solution s1;
    cout << s1.findLHS(nums) << endl;
    return 0;
}
```

Python:

```
def findLHS(nums):
    nums.sort()
    start, res = 0, 0
    for end in range(len(nums)):
        while nums[end] - nums[start] > 1:
            start += 1
        if nums[end] - nums[start] == 1:
            res = max(res, end - start + 1)
    return res
```

Complexity Analysis:

- **Time Complexity**: `O(n log n)` for sorting + `O(n)` scanning.
- **Space Complexity**: `O(1)` (no extra space apart from sorting).

Optimal Approach (Hashmap Frequency Count)

Approach

- Use a `unordered_map` to store frequency of each element.
- For each element `x`, check if `x+1` exists.
- If yes, `freq+ freq[x+1]` is a valid subsequence.
- Keep track of max length.

Code

C++:

```
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    int findLHS(vector<int>& nums) {
        unordered_map<int, int> freq;
        for (int num : nums) freq[num]++;
        int maxLen = 0;
        for (auto& [num, count] : freq) {
            if (freq.count(num + 1)) {
                maxLen = max(maxLen, count + freq[num + 1]);
            }
        }
        return maxLen;
    }
};

int main() {
    vector<int> nums = {1,3,2,2,5,2,3,7};
    Solution s1;
    cout << s1.findLHS(nums) << endl;
    return 0;
}
```

Python:

```
def findLHS(nums):
    from collections import Counter
    freq = Counter(nums)
    res = 0
    for key in freq:
        if key + 1 in freq:
            res = max(res, freq[key] + freq[key+1])
    return res
```

Complexity Analysis:

- **Time Complexity**: `O(n)` for building hashmap and iterating over it.
- **Space Complexity**: `O(n)` for the hashmap.

THE END