Doremy's Paint 3 - Detailed Revision Notes

Problem: Codeforces 1890A | Rating: 800 | TLE CP-31 Sheet Problem #6

Problem Statement

An array $b_1, b_2, ..., b_n$ of positive integers is **good** if all the sums of two adjacent elements are equal to the same value.

More formally: The array is good if there exists a k such that:

- $b_1 + b_2 = b_2 + b_3 = b_3 + b_4 = ... = b_{n-1} + b_n = k$

Task: Doremy has an array `a` of length `n`. She can permute its elements (change their order) however she wants. Determine if she can make the array good.

Input Format:

- First line: `t` (number of test cases)
- For each test case:
  - First line: `n` (array length)
  - Second line: `n` integers representing the array

Output: Print "YES" if the array can be made good, "NO" otherwise.

Key Intuition & Mathematical Derivation

Step 1: Understanding the Good Array Condition

If an array is good, then:

$$b_1 + b_2 = b_2 + b_3 = b_3 + b_4 = ... = k$$

Step 2: Mathematical Simplification

From the equation $b_1 + b_2 = b_2 + b_3$:

- Cancel b_2 from both sides: $b_1 = b_3$

From $b_2 + b_3 = b_3 + b_4$:

- Cancel b_3 from both sides: $b_2 = b_4$

Continuing this pattern:

- $b_1 = b_3 = b_5 = b_7 = ...$ (all odd-positioned elements are equal)
- $b_2 = b_4 = b_6 = b_8 = ...$ (all even-positioned elements are equal)

Step 3: Critical Observation

The array can have AT MOST 2 unique elements!

- One unique element for odd positions
- One unique element for even positions

Expected Time Complexity Analysis

- **Time Complexity:** O(n) per test case
  - We need to iterate through the array once to count frequencies
  - Map operations are O(1) on average
- **Space Complexity:** O(n) for storing element frequencies
- **Overall:** O(t × n) where t is number of test cases

Complete Solution Approach

Case 1: All Elements are Same

If all elements in the array are identical, then ANY arrangement will be good.
Result: Always print "YES"

Case 2: More than 2 Unique Elements

If there are more than 2 unique elements, it's impossible to make the array good.
Result: Always print "NO"

Case 3: Exactly 2 Unique Elements

This is the critical case. Let's say we have elements `x` and `y` with frequencies `count_x` and `count_y`.

For n elements total:

- Odd positions: `⌈n/2⌉` positions (ceiling of n/2)
- Even positions: `⌊n/2⌋` positions (floor of n/2)

Valid frequency combinations:

1. `count_x = ⌈n/2⌉` and `count_y = ⌊n/2⌋`
2. `count_x = ⌊n/2⌋` and `count_y = ⌈n/2⌉`

Simplified condition: `|count_x - count_y| ≤ 1`

Why This Condition Works

For odd n (e.g., n=5):

- Odd positions: 3 (positions 1,3,5)
- Even positions: 2 (positions 2,4)
- Valid: frequencies (3,2) or (2,3)

For even n (e.g., n=4):

- Odd positions: 2 (positions 1,3)
- Even positions: 2 (positions 2,4)
- Valid: frequencies (2,2) only

Implementation Details

Algorithm Steps:

1. **Count frequencies** of all unique elements using a map/dictionary
2. **Check unique element count:**
    - If 1 unique element → print "YES"
    - If >2 unique elements → print "NO"
    - If exactly 2 unique elements → check frequency condition
3. **For 2 unique elements:** Check if `|freq1 - freq2| ≤ 1`

Code Structure:

```
// Count frequencies
map<int, int> freq;
for(int x : array) {
    freq[x]++;
}

// Check conditions
if(freq.size() == 1) {
    // All same elements
    cout << "YES\n";
} else if(freq.size() > 2) {
    // More than 2 unique elements
    cout << "NO\n";
} else {
    // Exactly 2 unique elements
    vector<int> counts;
    for(auto p : freq) {
        counts.push_back(p.second);
    }
    if(abs(counts[0] - counts[1]) <= 1) {
        cout << "YES\n";
    } else {
```

```
        cout << "NO\n";
    }
}
```

Test Cases Walkthrough

Example 1: [8, 9] (n=2)

- 2 unique elements: 8(freq=1), 9(freq=1)
- |1-1| = 0 ≤ 1 ✓
- **Answer:** YES

Example 2: [1, 1, 2] (n=3)

- 2 unique elements: 1(freq=2), 2(freq=1)
- |2-1| = 1 ≤ 1 ✓
- Arrangement: [1, 2, 1] → sums: 3, 3
- **Answer:** YES

Example 3: [1, 1, 4, 5] (n=4)

- 3 unique elements: impossible to make good
- **Answer:** NO

Example 4: [2, 3, 3, 3, 3] (n=5)

- 2 unique elements: 2(freq=1), 3(freq=4)
- |1-4| = 3 > 1 ✗
- **Answer:** NO

Example 5: [100000, 100000, 100000, 100000] (n=4)

- 1 unique element: all same
- **Answer:** YES

Common Mistakes to Avoid

1. **Not handling the case of all same elements properly**
2. **Forgetting that we can permute elements freely**
3. **Not understanding that odd/even positions must have same elements**
4. **Incorrect frequency difference calculation**
5. **Missing edge cases like n=1 or n=2**

Key Insights for Competitive Programming

1. **Pattern Recognition:** Look for mathematical relationships in constraints
2. **Simplification:** Reduce complex conditions to simpler mathematical expressions
3. **Case Analysis:** Break problem into distinct cases based on input characteristics
4. **Frequency Analysis:** Many CP problems involve counting element frequencies
5. **Permutation Problems:** When elements can be rearranged, focus on what properties must be preserved

Complexity Analysis Summary

- **Time:** O(n log n) due to map operations, can be optimized to O(n) with unordered_map
- **Space:** O(n) for frequency storage
- **Difficulty:** 800-rated problem suitable for beginners learning pattern recognition

This problem teaches fundamental concepts of mathematical proof, case analysis, and the importance of finding invariant properties in permutation problems.