3.Hard\02.Majority_element_2.cpp

```cpp
/*
QUESTION:
Given an integer array of size n, find all elements that appear more than ⌊ n/3 ⌋ times.

Example 1:
Input: nums = [3,2,3]
Output: [3]

Example 2:
Input: nums = [1]
Output: [1]

APPROACH:
To find all elements that appear more than ⌊ n/3 ⌋ times, we can use the Boyer-Moore Majority Vote algorithm. This algorithm helps us find potential candidates that could appear more than ⌊ n/3 ⌋ times in a single pass. After finding the candidates, we count their occurrences and return the elements that meet the criteria.

1. Initialize two candidate variables, c1 and c2, and their corresponding vote counters, vote1 and vote2.
2. Iterate through the array:
   - If the current element matches c1, increment vote1.
   - Else if the current element matches c2, increment vote2.
   - Else if vote1 is 0, assign the current element to c1 and set vote1 to 1.
   - Else if vote2 is 0, assign the current element to c2 and set vote2 to 1.
   - Else, decrement both vote1 and vote2.
3. After finding the potential candidates, count the occurrences of each candidate using cnt1 and cnt2.
4. If cnt1 is greater than ⌊ n/3 ⌋, add c1 to the result vector.
5. If cnt2 is greater than ⌊ n/3 ⌋ and c2 is different from c1, add c2 to the result vector.
6. Return the result vector containing the elements that appear more than ⌊ n/3 ⌋ times.


*/

vector<int> majorityElement(vector<int> &nums)
{
    int c1 = 0, c2 = 0, vote1 = 0, vote2 = 0;

    // Finding potential candidates
    for (int i = 0; i < nums.size(); i++)
    {
        if (c1 == nums[i])
        {
            vote1++;
        }
        else if (c2 == nums[i])
        {
            vote2++;
        }
        else if (vote1 == 0)
        {
            c1 = nums[i];
            vote1 = 1;
        }
        else if (vote2 == 0)
        {
            c2 = nums[i];
            vote2 = 1;
        }
        else
        {
            vote1--;
            vote2--;
        }
    }

    vector<int> ans;
    int cnt1 = 0, cnt2 = 0;

    // Counting occurrences of potential candidates
    for (auto it : nums)
    {
        if (it == c1)
        {
            cnt1++;
        }
        if (it == c2)
        {
            cnt2++;
        }
    }

    // Checking if candidates appear more than ⌊ n/3 ⌋ times
    if (cnt1 > nums.size() / 3)
    {
        ans.push_back(c1);
    }
    if (cnt2 > nums.size() / 3 && c2 != c1)
    {
        ans.push_back(c2);
    }

    return ans;
}

// TIME COMPLEXITY: O(n), where n is the size of the input array.
// SPACE COMPLEXITY: O(1), as we are using a constant amount of extra space.
```