MEDIUM 2:

PROBLEM STATEMENT:

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

SOLUTION:

import java.util.ArrayList;

import java.util.List;

public class MajorityElement {

public static List<Integer> majorityElements(int[] nums) {

List<Integer> result = new ArrayList<>();

int candidate1 = 0, count1 = 0;

int candidate2 = 0, count2 = 0;

// Phase 1: Find potential candidates

for (int num : nums) {

if (num == candidate1) {

count1++;

} else if (num == candidate2) {

count2++;

} else if (count1 == 0) {

candidate1 = num;

count1 = 1;

} else if (count2 == 0) {

candidate2 = num;

count2 = 1;

} else {

count1--;

count2--;

}

}

// Phase 2: Verify candidates

count1 = count2 = 0;

for (int num : nums) {

if (num == candidate1) {

count1++;

} else if (num == candidate2) {

count2++;

}

}

// Check if candidates appear more than n/3 times

if (count1 > nums.length / 3) {

result.add(candidate1);

}

if (count2 > nums.length / 3) {

result.add(candidate2);

}

return result;

}

public static void main(String[] args) {

// Test cases

int[] nums1 = {3, 2, 3};

System.out.println(majorityElements(nums1)); // Output: [3]

int[] nums2 = {1};

System.out.println(majorityElements(nums2)); // Output: [1]

int[] nums3 = {1, 2};

System.out.println(majorityElements(nums3)); // Output: [1, 2]

}

}

SUMMARY AND ALGORITHM:

Summary:

The provided Java code defines a class named `MajorityElement` that contains a method `majorityElements` designed to find majority elements in an array. A majority element is defined as an element that appears more than n/3 times, where n is the length of the array. The main method includes several test cases to demonstrate the functionality of the `majorityElements` method.

Algorithm:

1. majorityElements Method:

- Initialize variables `candidate1`, `count1`, `candidate2`, and `count2` to keep track of two potential majority candidates and their counts.

- In the first phase, iterate through the array (`nums`) to find potential candidates:

- If the current number is equal to `candidate1`, increment `count1`.

- If the current number is equal to `candidate2`, increment `count2`.

- If `count1` is 0, assign the current number to `candidate1` and set `count1` to 1.

- If `count2` is 0, assign the current number to `candidate2` and set `count2` to 1.

- If neither `count1` nor `count2` is 0, decrement both counts.

- In the second phase, reiterate through the array to verify the counts of the potential candidates (`count1` and `count2`).

- If the count of a candidate is greater than `nums.length / 3`, add the candidate to the result list.

- Return the result list containing majority elements.

2. main Method:

- Test cases are provided to demonstrate the functionality of the `majorityElements` method.

- The results of the test cases are printed using `System.out.println`.

Example Test Cases and Expected Outputs:

int[] nums1 = {3, 2, 3};

System.out.println(majorityElements(nums1)); // Output: [3]

int[] nums2 = {1};

System.out.println(majorityElements(nums2)); // Output: [1]

int[] nums3 = {1, 2};

System.out.println(majorityElements(nums3)); // Output: [1, 2]

These test cases cover scenarios with different array lengths and compositions, and the `majorityElements` method is expected to accurately identify majority elements based on the defined criteria.