Majority Elements(>N/3 times) | Find the elements that appears more than N/3 times in the array

Problem Statement: Given an array of N integers. Find the elements that appear more than **N/3** times in the array. If no such element exists, return an empty vector.

Example 1:

```
Input: N = 5, array[] = {1,2,2,3,2}



Ouput: 2



Explanation: Here we can see that the Count(1) = 1, Count(2) = 3 and Count(3)
= 1.Therefore, the count of 2 is greater than N/3 times. Hence, 2 is the
answer.
```

Example 2:

```
Input:  N = 6, array[] = {11,33,33,11,33,11}



Output: 11 33



Explanation: Here we can see that the Count(11) = 3 and Count(33) = 3.
Therefore, the count of both 11 and 33 is greater than N/3 times. Hence, 11
and 33 is the answer.
```

Solution

***Disclaimer**: Don't jump directly to the solution, try it out yourself first.*

Solution 1: Brute-Force

Approach: Simply count the no. of appearance for each element using nested loops and whenever you find the count of an element greater than N/3 times, that element will be your answer.

Code:

- C++ Code
- Java Code

```cpp
#include <bits/stdc++.h>

using namespace std;
vector < int > majorityElement(int arr[], int n) {
  vector < int > ans;
  for (int i = 0; i < n; i++) {
    int cnt = 1;
    for (int j = i + 1; j < n; j++) {
      if (arr[j] == arr[i])
        cnt++;
    }

    if (cnt > (n / 3))
      ans.push_back(arr[i]);
  }

  return ans;
}

int main() {
  int arr[] = {1,2,2,3,2};
  vector<int> majority;
  majority = majorityElement(arr, 5);
  cout << "The majority element is" << endl;
  set < int > s(majority.begin(), majority.end());
  for (auto it: s) {
    cout << it << " ";
  }
}
```

Output:

The majority element is 2

Time Complexity: O(n^2)

Space Complexity: O(1)

Solution 2: Better Solution

Approach: Traverse the whole array and store the count of every element in a map. After that traverse through the map and whenever you find the count of an element greater than N/3 times, store that element in your answer.

Dry Run: Lets take the example of arr[] = {10,20,40,40,40}, n=5.

First, we create an unordered map to store the count of each element.

Now traverse through the array

1. Found 10 at index 0, increase the value of key 10 in the map by 1.
2. Found 20 at index 1, increase the value of key 20 in the map by 1.
3. Found 40 at index 2, increase the value of key 40 in the map by 1.
4. Found 40 at index 3, increase the value of key 40 in the map by 1.
5. Found 40 at index 4, increase the value of key 40 in the map by 1.

Now, Our map will look like this:

10 -> 1

20 ->1

40 ->3

Now traverse through the map,

We found that the value of key 40 is greater than 2 (N/3). So, 40 is the answer.

Code:

- C++ Code
- Java Code

```
#include <bits/stdc++.h>

using namespace std;
vector < int > majorityElement(int arr[], int n) {
  unordered_map < int, int > mp;
  vector < int > ans;

  for (int i = 0; i < n; i++) {
    mp[arr[i]]++;
  }

  for (auto x: mp) {
    if (x.second > (n / 3))
      ans.push_back(x.first);
  }

  return ans;
}

int main() {
  int arr[] = {1,2,2,3,2};
  vector < int > majority;
  majority = majorityElement(arr, 5);
  cout << "The majority element is " << ;
  for (auto it: majority) {
    cout << it << " ";
  }
}
```

Output: The majority element is: 2

Time Complexity: O(n)

Space Complexity: O(n)

Solution 3: Optimal Solution (Extended Boyer Moore's Voting Algorithm)

Approach + Intuition: In our code, we start with declaring a few variables:

- num1 and num2 will store our currently most frequent and second most frequent element.
- c1 and c2 will store their frequency relatively to other numbers.
- We are sure that there will be a max of 2 elements which occurs > N/3 times because there cannot be if you do a simple math addition.

Let, ele be the element present in the array at any index.

- if ele == num1, so we increment c1.
- if ele == num2, so we increment c2.
- if c1 is 0, so we assign num1 = ele.
- if c2 is 0, so we assign num2 = ele.
- In all the other cases we decrease both c1 and c2.

In the last step, we will run a loop to check if num1 or nums2 are the majority elements or not by running a for loop check.

Intuition: Since it's guaranteed that a number can be a majority element, hence it will always be present at the last block, hence, in turn, will be on nums1 and nums2. For a more detailed explanation, please watch the video below.

Code:

- C++ Code
- Java Code

```
#include <bits/stdc++.h>

using namespace std;
vector < int > majorityElement(int nums[], int n) {
  int sz = n;
  int num1 = -1, num2 = -1, count1 = 0, count2 = 0, i;
  for (i = 0; i < sz; i++) {
    if (nums[i] == num1)
      count1++;
    else if (nums[i] == num2)
      count2++;
    else if (count1 == 0) {
      num1 = nums[i];
      count1 = 1;
    } else if (count2 == 0) {
      num2 = nums[i];
      count2 = 1;
    } else {
      count1--;
      count2--;
    }
  }
  vector < int > ans;
  count1 = count2 = 0;
  for (i = 0; i < sz; i++) {
    if (nums[i] == num1)
      count1++;
    else if (nums[i] == num2)
      count2++;
  }
  if (count1 > sz / 3)
    ans.push_back(num1);
  if (count2 > sz / 3)
    ans.push_back(num2);
  return ans;
}

int main() {
  int arr[] = {1,2,2,3,2};
  vector < int > majority;
  majority = majorityElement(arr, 5);
  cout << "The majority element is ";

  for (auto it: majority) {
    cout << it << " ";
  }
}
```

Output:

The majority element is 2

Time Complexity: O(n)

Space Complexity: O(1)