For this problem, we are not going to use a "typical" divide and conquer method that runs $O(nlogn)$ time and uses $O(logn)$ space. Instead, we use a different approach: Since there are at most 2 voters to be returned, we create 2 slots storing their name (id) and 2 variables storing the corresponding count. For all candidates in the ballot box, if one is the same as the one in first or second slot, we add 1 to the corresponding count; if there is a slot empty (i.e., corresponding count equals to 0), we assign this candidate to that slot; for all other cases, we subtract 1 from both counts. After one round, we have two slots with names and counts. We set these counts 0 again and we count the actual votes for these two slots using another loop. Finally we return the one with votes greater than $n/3$ where n is the total votes.

The algorithm is stated as follow:

Algorithm 1 Algorithm for problem 2

```
Set first_slot = Null
Set second_slot = Null
Set first_slot_count = 0
Set second_slot_count = 0
For candidate in all_candidate_array:
    If candidate == first_slot:
        first_slot_count += 1
    Elif candidate == second_slot:
        second_slot_count += 1
    Elif first_slot_count == 0:
        first_slot_count = candidate
    Elif second_slot_count == 0:
        second_slot_count = candidate
    Else:
        first_slot_count -= 1
        second_slot_count -= 1
Endfor
first_slot_count = 0
second_slot_count = 0
For candidate in all_candidate_array:
    If candidate == first_slot:
        first_slot_count += 1
    Elif candidate == second_slot:
        second_slot_count += 1
Endfor
Return first_slot if first_slot_count > n/3 & second_slot_count ≤ n/3
Return second_slot if first_slot_count ≤ n/3 & second_slot_count > n/3
Return first_slot & second_slot if first_slot_count > n/3 & second_slot_count > n/3
Return None if first_slot_count ≤ n/3 & second_slot_count ≤ n/3
```

Proof of Correctness:

We need to prove that these 2 slots can capture all candidates with votes greater than $n/3$. Suppose we have a candidate have slightly more than $n/3$ votes (say $[n/3]+1$). The during the first loop, the count of this candidate will add 1 for $[n/3]-1$ times. But there are about 2n/3 times that this count can be subtracted by 1. **How can we guarantee that it cannot be subtracted to 0?** (i.e., slot taken by others) Well, as long as a second candidate repeats, the other slot count will add 1 and the first slot will not be subtracted. So the worst case is that there is no repeat for any other candidates (i.e., all other candidates has the vote exactly one). However, we noticed from the algorithm that when the other slot is empty, the first slot count will not be subtracted as well. If all other candidates have exactly vote 1, the count of the second slot will be like (1,0,1,0,1,0,......). When it comes to 1, the first slot count will be subtracted by 1; when it comes to 0, there will be no subtraction. Since the number of other candidates is less than $2n/3$, the number of subtractions will be less than $2n/3/2 = n/3$. Since the first candidate has $[n/3]+1$ votes, his slot cannot be taken even if in the worst case. Hence, we guarantee that the slots will capture all candidates who has more than $n/3$ votes. **Then why would we need the second loop?** That is because we have 2 slots, if we have only one candidates or even no one satisfying the condition, the other slot(s) may also be taken. So for this round we need to count the exact number of counts for these two slots, and if any of them is greater than $n/3$, the candidate(s) will be the one we need. Hence, we proved that this algorithm is correct.

Complexity Analysis:

Time complexity: For the first loop, we only have "if" statements that compare the current candidate with the ones in the slots which costs constant time as well as the assigning or adding values. Hence for the entire loop we spend $O(n)$ time. The second loop is the same, where we only have comparisons and add values. Hence, the total time complexity is $\mathbf{O(n)}$.

Space complexity: In addition to the read-only array, we only allocate 4 blocks of memories that store 4 variables (2 slots and 2 counts). Hence the space complexity is constant $\mathbf{O(1)}$!