A

PROJECT

REPORT

ON

BOYER MOORE'S VOTING ALGORITHM

Submitted in partial fulfillment of the requirement for the IV semester

of

BACHELOR OF TECHNOLOGY

IN

DESIGN ANDANALYSIS OF ALGORITHM

Submitted By:

KUMAR SHASHWAT (RA2011003011262)

NIPURN BHAAL (RA2011003011272)

AYUSH PODDAR (RA2011003011279)

Under the supervision of

Dr. C. Muralidharan

(Asst. Professor)

DEPARTMENT OF COMPUTING AND TECHNOLOGY, SRM

INSTITUTE OF SCIENCE AND TECHNOLOGY SESSION – 2022

18CSC204J- Design and Analysis of Algorithms

Mini Project

Record Work

Registration numbers: RA2011003011262, RA2011003011272, RA2011003011279
Names: KUMAR SHASHWAT, NIPURN BHAAL & AYUSH PODDAR
Semester: 4th
Department: CTECH

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

S.R.M. NAGAR, KATTANKULATHUR -603 203

KANCHEEPURAM DISTRICT

BONAFIDE CERTIFICATE

Register No-RA2011003011262

Certified to be the bonafide record of work done by **KUMAR SHASHWAT** *of* **COMPUTER SCIENCE and ENGINEERING (SECOND YEAR)***, B.Tech Degree course in the Practical* ***18CSC204J – DESIGN AND ANALYSIS OF ALGORITHMS*** *in* ***SRM INSTITUTE OF SCIENCE AND TECHNOLOGY, Kattankulathur*** *during the academic year 2022-2023*

Lab Incharge

Head of the Department

Date:

Submitted for University Examination held in ______________________, SRM INSTITUTE OF SCIENCE AND TECHNOLOGY, Kattankulathur.

Date: **Examiner-1** **Examiner-2**

CERTIFICATE

This is to certify that the project entitled "**Boyer Moore's Voting Algorithm**" carried out by **KUMAR SHASHWAT** under my supervision at Department of Computing and Technology, SRM Institute of Technology, Kattankulathur, Chennai.

The work is original, as it has not been submitted earlier either in part or full for any purpose before.

Dr. C. Muralidharan
(Asst. Professor)

DECLARATION

I, hereby declare that the work presented in this dissertation entitled "**Boyer Moore's Voting Algorithm**" has been done by me and my team, and this dissertation embodies my own work.

Approved By:
Dr. C. Muralidharan

ACKNOWLEDGEMENTS

I would like to thank Dr. C. Muralidharan (Asst. Professor) sir who has been a great inspiration and who have provided sufficient background knowledge and understanding of this subject.

Our humble prostration goes to him, for providing all the necessary resources and environment, which have aided me to complete this project successfully.

Table of Contents

Contribution Table

Name	Reg. no.	Contribution
NIPURN BHAAL	RA2011003011272	Idea for the project.
KUMAR SHASHWAT	RA2011003011262	Implementation of Code
AYUSH PODDAR	RA2011003011279	Preparation of ppt
KUMAR SHASHWAT, NIPURN BHAAL & AYUSH PODDAR	RA2011003011262 , RA2011003011272 & RA2011003011279	Preparation of report and analysis of code

Visualizing Boyer Moore's Voting Algorithm

KUMAR SHASHWAT, NIPURN BHAAL & AYUSH PODDAR

Department of Engineering

SRM Institute of Science and

Technology Chennai-603203

Abstract:

The Boyer–Moore majority vote algorithm is an algorithm for finding the majority of a sequence of elements using linear time and constant space. The Boyer-Moore voting algorithm is one of the popular optimal algorithms which is used to find the majority element among the given elements that have more than N/ 2 occurrences. This works perfectly fine for finding the majority element which takes 2 traversals over the given elements, which works in O(N) time complexity and O(1) space complexity.

Keywords:

Greedy Algorithm, Space Complexity Optimization.

Introduction:

- We have taken this algorithm instead of Divide and Conquer as our approach because it helps us to reduce auxiliary recursion calls as in traditional Divide and Conquer method.

- Among all the algorithmic approaches, the simplest and straightforward approach is the Greedy method. In this approach, the decision is taken on the basis of current available information without worrying about the effect of the current decision in future.

.

Introduction – Problem Statement

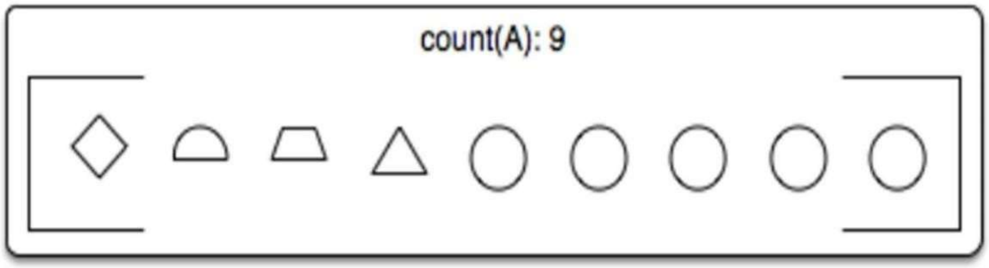
In a general Presidency election, there are N candidates who can a cast a vote either for themselves or any other member. Only two persons designated as President and Vice President have to be elected. President gets the maximum votes followed by the vice president.

Find all candidates that get votes more than **⌊ n/3 ⌋ times**.

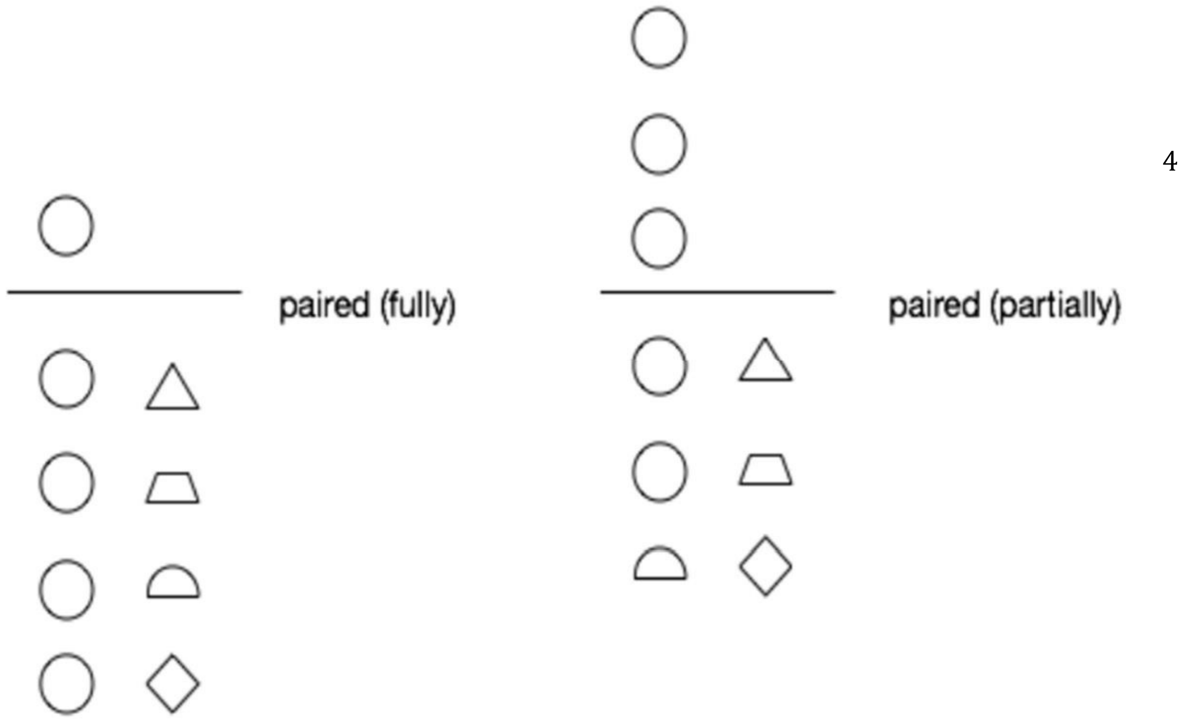
Solve this using a space complexity of **O (1)**.

Problem Explanation

Suppose there are nine elements in array A, and the round one is the majority.

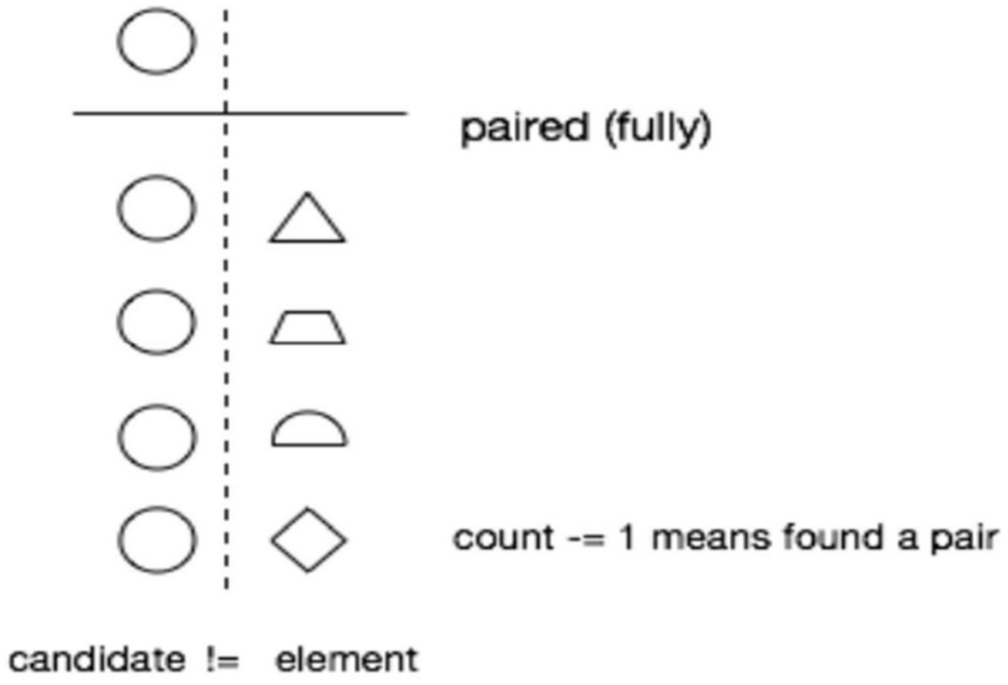


No matter in what order we select element from the array, we can only get two results.

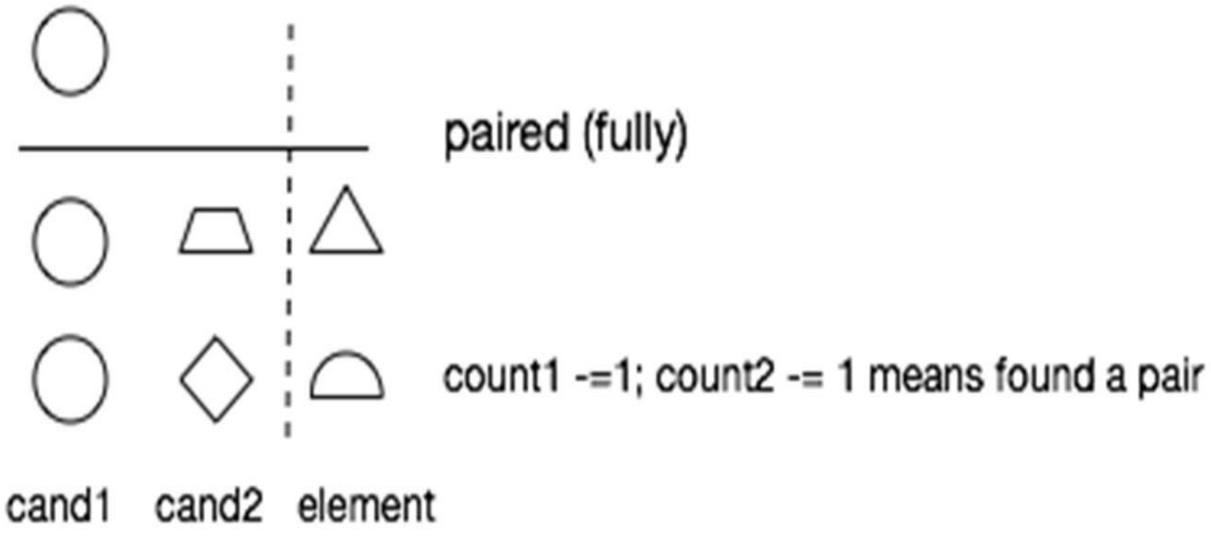


Compared to fully pairing, it is a little wasting of the partially pairing as there are some round ones are not paired (definitely it would be left). So, under the condition that the majority element exists, we could only think about the **fully pairing situation**. (It's useful when dealing with **n/3** situation)

We can consider either column as the candidate, and it's intuitive for me to get understand thatthe code means found a pair.



So here comes the n/3 problem, we would only think about the fully pairing situation. If the over one third majority exists, it should be left after pairing.



Q/A Related to Problem Statement

Why would we use three elements as a pair?

- ✓ Because it makes sure that in fully pairing the count of majority element equals n/3.

Why we have implemented Greedy Approach?

- ✓ Among all the algorithmic approaches, the simplest and straightforward approach is theGreedy method. In this approach, the decision is taken on the basis of current available information without worrying about the effect of the current decision in future.
- ✓ Greedy algorithms build a solution part by part, choosing the next part in such a way, thatit gives an immediate benefit. This approach never reconsiders the choices taken previously. This approach is mainly used to solve optimization problems. Greedy methodis easy to implement and quite efficient in most of the cases. Hence, we can say that Greedy algorithm is an algorithmic paradigm based on heuristic that follows local optimal choice at each step with the hope of finding global optimal solution.
- ✓ In many problems, it does not produce an optimal solution though it gives anapproximate (near optimal) solution in a reasonable time.

Explanation of Algorithm with Example

If N=15, then

Frequency (ME) = N/3 = 5

In this case, we realize that we can have 3 cases.

Case 1: 0 people winning: Everyone gets unique votes or some get 2 etc.

Case 2: Person winning: President who gets vote like 6 and rest all get less than 6.

For example: 1 1 1 1 1 1 2 2 3 3 4 4 5 5 6

President: 1 with 6 votes

Case 3: 2 people winning: Vice President both get equal to 6 or one of them gets more than 6.

For Example: 1 1 1 1 1 1 1 2 2 2 2 2 2 8 8

President: 1 with 7 votes

Vice President: 2 with 6 votes

Rest: 2 votes

Visualizing Boyer Moore's Voting Algorithm with OneStatement

What will President and Vice President say?

- President will say that "I won with a difference of 5".
- Vice President will say that "I won with a difference of 4"

Till this point I want you to realise that Vote of Majority Element is only affected by Minority elements. By people who are not winning.

Hence in this case we have to maintain two independent counters keeping track of **ME1** an **ME2.**

The only thing to note here is that when you reach a Minority element, that minority element will affect both President and Vice President's vote.

That's why when we hit a minority element, we reduce count for both the Majority Elements as it is affecting winning difference of both the winning Candidates

Understanding through the help of Example

Example

Index: 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4

Given Array: 1 2 3 1 1 1 7 7 1 1 7 5 7 7 7

cnt1: 1

cnt2: 1

Count 1 is for ME1->

Initialized to 0

Count 2 is for ME2->

Initialized by 0

At i=0: We get First Candidate

At i: 1: This is not equal to our first candidate so new counter initialized by 1

Index: 0 1 2 3 4 5 6 7 8 9 0 1 2 3
4

Given Array: 1 2 3 1 1 1 7 7 1 1
7 5 7 7 7

cnt1: .1 0

cnt2: ….1 0

At i=2: This vote is a loss for both the candidates BUT not in the long run. That's what we aretrying to do. We want our counters to be greater than 1 for that candidate to win.

Index: 0 1 2 3 4 5 6 7 8 9 0 1 2 3
4

Given Array: 1 2 3 1 1 1 7 7 1 1
7 5 7 7 7

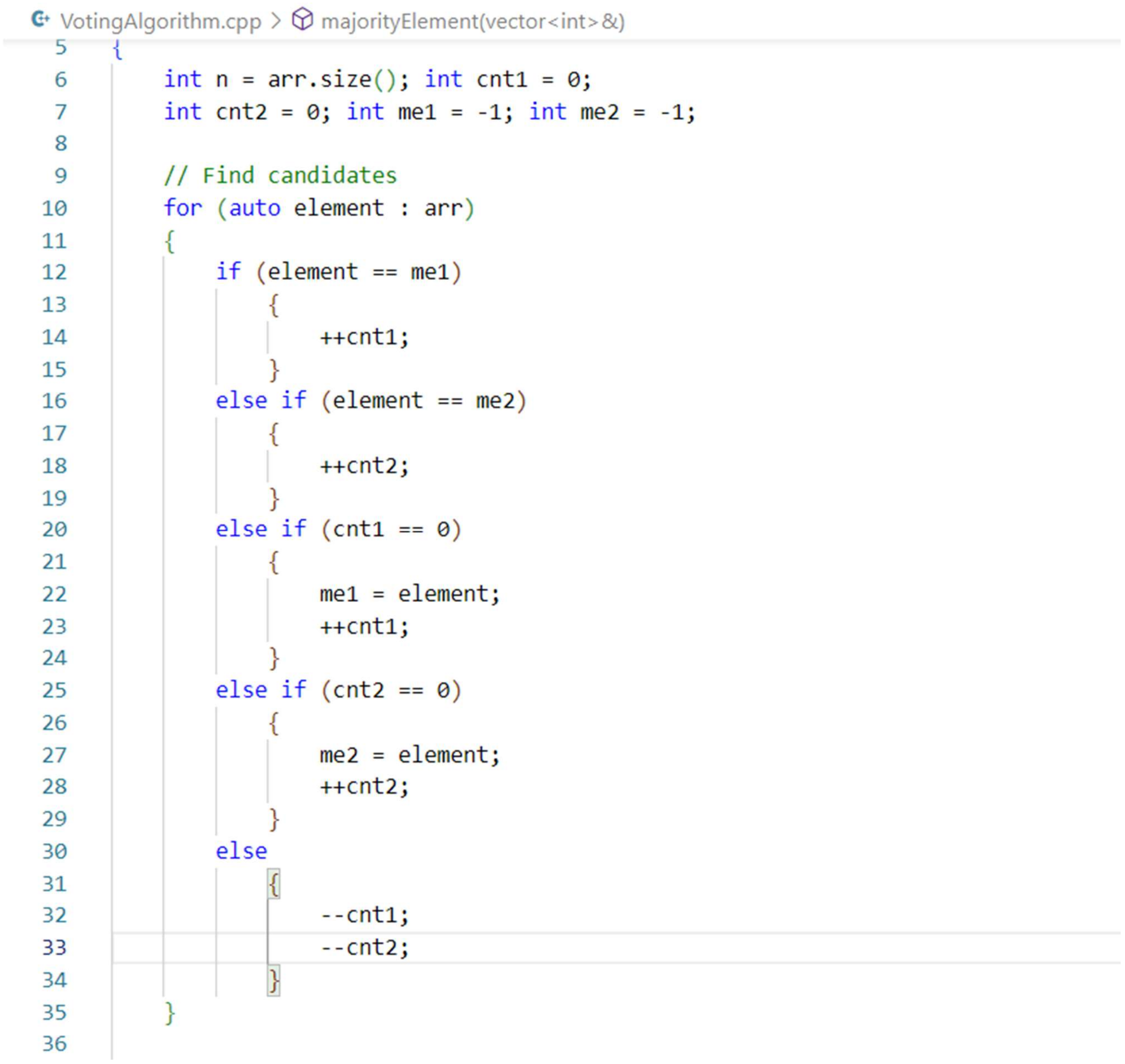
cnt1: 1….0 1 2 3…   4 5….4

cnt2: ….1.0……….1 2…3 2 3 4 5

Make sure to update new ME1 and ME2 when counters == 0.

Make sure to check whether these really are Majority Elements because this algorithm will giveyou ME1 and ME2 in any case.

Source Code

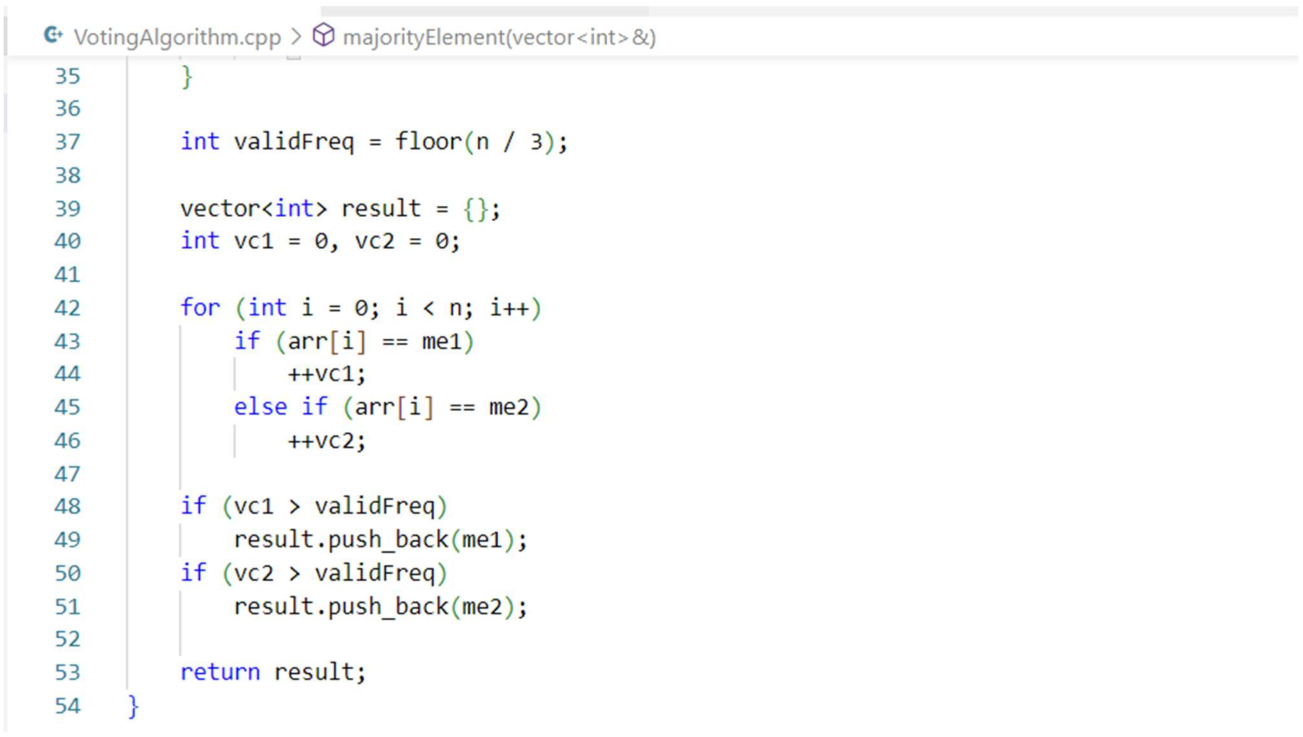


VotingAlgorithm.cpp > majorityElement(vector<int>&)

```
{
    int n = arr.size(); int cnt1 = 0;
    int cnt2 = 0; int me1 = -1; int me2 = -1;

    // Find candidates
    for (auto element : arr)
    {
        if (element == me1)
            {
                ++cnt1;
            }
        else if (element == me2)
            {
                ++cnt2;
            }
        else if (cnt1 == 0)
            {
                me1 = element;
                ++cnt1;
            }
        else if (cnt2 == 0)
            {
                me2 = element;
                ++cnt2;
            }
        else
            {
                --cnt1;
                --cnt2;
            }
    }

```



VotingAlgorithm.cpp > majorityElement(vector<int>&)

```cpp
    }

    int validFreq = floor(n / 3);

    vector<int> result = {};
    int vc1 = 0, vc2 = 0;

    for (int i = 0; i < n; i++)
        if (arr[i] == me1)
            ++vc1;
        else if (arr[i] == me2)
            ++vc2;

    if (vc1 > validFreq)
        result.push_back(me1);
    if (vc2 > validFreq)
        result.push_back(me2);

    return result;
}
```

Time Complexity

Time Complexity will remain **O(n)**

Space Complexity

Space complexity will be **O(1)**

Conclusion

Problem could be solved using Divide and conquer, but greedy helps us to reduce auxiliary recursion calls as in traditional Divide and Conquer method.
The amount of memory that the algorithm needs is the space for one element and one counter. In the random access model of computing usually used for the analysis of algorithms, each of these values can be stored in a machine word and the total space needed is O(1). If an array index is needed to keep track of the algorithm's position in the input sequence, it doesn't change the overall constant space bound. The algorithm's bit complexity (the space it would need, for instance, on a Turing machine) is higher, the sum of the binary logarithms of the input length and the size of the universe from which the elements are drawn. Both the random access model and bit complexity analyses only count the working storage of the algorithm, and not the storage for the input sequence itself.

References

- http://goo.gl/64Nams

- http://www.cs.rug.nl/~wim/pub/whh348.pdf

- https://discuss.leetcode.com/topic/17564/boyer-moore-majority-vote-algorithm-and-my- elaboration