Majority Element

Majority Element: A majority element in an array A[] of size n is an element that appears more than n/2 times (and hence there is at most one such element).

Write a function which takes an array and emits the majority element (if it exists), otherwise prints NONE as follows:

```
I/P : 3 3 4 2 4 4 2 4 4
O/P : 4

I/P : 3 3 4 2 4 4 2 4
O/P : NONE
```

Recommended: Please solve it on "*PRACTICE* " first, before moving on to the solution.

METHOD 1 (Basic)

The basic solution is to have two loops and keep track of maximum count for all different elements. If maximum count becomes greater than n/2 then break the loops and return the element having maximum count. If maximum count doesn't become more than n/2 then majority element doesn't exist.

Time Complexity: O(n*n).
Auxiliary Space : O(1).

METHOD 2 (Using Binary Search Tree)

Thanks to Sachin Midha for suggesting this solution. Node of the Binary Search Tree (used in this

approach) will be as follows.

```
struct tree
{
  int element;
  int count;
}BST;
```

Run on IDE

Insert elements in BST one by one and if an element is already present then increment the count of the node. At any stage, if count of a node becomes more than n/2 then return.
The method works well for the cases where n/2+1 occurrences of the majority element is present in the starting of the array, for example {1, 1, 1, 1, 1, 2, 3, 4}.

Time Complexity: If a binary search tree is used then time complexity will be O(n^2). If a self-balancing-binary-search tree is used then O(nlogn)
Auxiliary Space: O(n)

METHOD 3 (Using Moore's Voting Algorithm)
This is a two step process.
1. The first step gives the element that may be majority element in the array. If there is a majority element in an array, then this step will definitely return majority element, otherwise it will return any other element.
2. Check if the element obtained from above step is majority element.This step is necessary as we are not always sure that element return by first step is majority element.

1. Finding a Candidate:
The algorithm for first phase that works in O(n) is known as Moore's Voting Algorithm. Basic idea of the algorithm is if we cancel out each occurrence of an element e with all the other elements that are different from e then e will exist till end if it is a majority element.

```
findCandidate(a[], size)
1.  Initialize index and count of majority element
     maj_index = 0, count = 1
2.  Loop for i = 1 to size - 1
    (a) If a[maj_index] == a[i]
          count++
    (b) Else
        count--;
    (c) If count == 0
          maj_index = i;
          count = 1
3.  Return a[maj_index]
```

Above algorithm loops through each element and maintains a count of a[maj_index], If next element is same then increments the count, if next element is not same then decrements the count, and if the count reaches 0 then changes the maj_index to the current element and sets count to 1.
First Phase algorithm gives us a candidate element. In second phase we need to check if the candidate is really a majority element. Second phase is simple and can be easily done in O(n). We just need to check if count of the candidate element is greater than n/2.

Example:
A[] = 2, 2, 3, 5, 2, 2, 6
Initialize:
maj_index = 0, count = 1 –> candidate '2?
2, 2, 3, 5, 2, 2, 6

Same as a[maj_index] => count = 2
2, 2, 3, 5, 2, 2, 6

Different from a[maj_index] => count = 1
2, 2, 3, 5, 2, 2, 6

Different from a[maj_index] => count = 0
Since count = 0, change candidate for majority element to 5 => maj_index = 3, count = 1
2, 2, 3, 5, 2, 2, 6

Different from a[maj_index] => count = 0
Since count = 0, change candidate for majority element to 2 => maj_index = 4
2, 2, 3, 5, 2, 2, 6

Same as a[maj_index] => count = 2
2, 2, 3, 5, 2, 2, 6

Different from a[maj_index] => count = 1

Finally candidate for majority element is 2.

First step uses Moore's Voting Algorithm to get a candidate for majority element.

2. *Check if the element obtained in step 1 is majority*

```
printMajority (a[], size)
1.  Find the candidate for majority
2.  If candidate is majority. i.e., appears more than n/2 times.
       Print the candidate
3.  Else
       Print "NONE"
```

Implementation of method 3:

C

```c
/* Program for finding out majority element in an array */
# include<stdio.h>
# define bool int

int findCandidate(int *, int);
bool isMajority(int *, int, int);

/* Function to print Majority Element */
void printMajority(int a[], int size)
{
  /* Find the candidate for Majority*/
  int cand = findCandidate(a, size);

  /* Print the candidate if it is Majority*/
  if (isMajority(a, size, cand))
    printf(" %d ", cand);
  else
    printf("No Majority Element");
}

/* Function to find the candidate for Majority */
int findCandidate(int a[], int size)
{
    int maj_index = 0, count = 1;
    int i;
    for (i = 1; i < size; i++)
    {
        if (a[maj_index] == a[i])
            count++;
        else
            count--;
        if (count == 0)
        {
            maj_index = i;
            count = 1;
        }
    }
    return a[maj_index];
}

/* Function to check if the candidate occurs more than n/2 times */
bool isMajority(int a[], int size, int cand)
{
    int i, count = 0;
    for (i = 0; i < size; i++)
      if (a[i] == cand)
         count++;
    if (count > size/2)
       return 1;
    else
       return 0;
}

/* Driver function to test above functions */
int main()
{
    int a[] = {1, 3, 3, 1, 2};
    int size = (sizeof(a))/sizeof(a[0]);
    printMajority(a, size);
    getchar();
    return 0;
}
```

Run on IDE

Java

```java
/* Program for finding out majority element in an array */

class MajorityElement
{
    /* Function to print Majority Element */
    void printMajority(int a[], int size)
    {
        /* Find the candidate for Majority*/
        int cand = findCandidate(a, size);

        /* Print the candidate if it is Majority*/
        if (isMajority(a, size, cand))
            System.out.println(" " + cand + " ");
        else
            System.out.println("No Majority Element");
    }

    /* Function to find the candidate for Majority */
    int findCandidate(int a[], int size)
    {
        int maj_index = 0, count = 1;
        int i;
        for (i = 1; i < size; i++)
        {
            if (a[maj_index] == a[i])
                count++;
            else
                count--;
            if (count == 0)
            {
                maj_index = i;
                count = 1;
            }
        }
        return a[maj_index];
    }

    /* Function to check if the candidate occurs more
       than n/2 times */
    boolean isMajority(int a[], int size, int cand)
    {
        int i, count = 0;
        for (i = 0; i < size; i++)
        {
            if (a[i] == cand)
                count++;
        }
        if (count > size / 2)
            return true;
        else
            return false;
    }

    /* Driver program to test the above functions */
    public static void main(String[] args)
    {
        MajorityElement majorelement = new MajorityElement();
        int a[] = new int[]{1, 3, 3, 1, 2};
        int size = a.length;
        majorelement.printMajority(a, size);
    }
}

// This code has been contributed by Mayank Jaiswal
```

Run on IDE

Python

```
# Program for finding out majority element in an array

# Function to find the candidate for Majority
def findCandidate(A):
    maj_index = 0
    count = 1
    for i in range(len(A)):
        if A[maj_index] == A[i]:
            count += 1
        else:
            count -= 1
        if count == 0:
            maj_index = i
            count = 1
    return A[maj_index]

# Function to check if the candidate occurs more than n/2 times
def isMajority(A, cand):
    count = 0
    for i in range(len(A)):
        if A[i] == cand:
            count += 1
    if count > len(A)/2:
        return True
    else:
        return False

# Function to print Majority Element
def printMajority(A):
    # Find the candidate for Majority
    cand = findCandidate(A)

    # Print the candidate if it is Majority
    if isMajority(A, cand) == True:
        print(cand)
    else:
        print("No Majority Element")

# Driver program to test above functions
A = [1, 3, 3, 1, 2]
printMajority(A)
```

Run on IDE

Output:

```
No Majority Element
```

Time Complexity: O(n)

Auxiliary Space : O(1)

METHOD 4 (Using Hashmap) : This method is somewhat similar to Moore voting algorithm in terms of time complexity, but in this case there is no need of second step of Moore voting algorithm.But as

usual, here space complexity becomes O(n).
In Hashmap(key-value pair), at value,maintain a count for each element(key) and whenever count is greater than half of array length, we are just returning that key(majority element).Thanks Ashwani Tanwar,Karan Malhotra for suggesting this.

Time Complexity: O(n)

Auxiliary Space: O(n)

Below is java implementation.

```
import java.util.HashMap;

/* Program for finding out majority element in an array */

class MajorityElement
{
    private static void findMajority(int[] arr)
    {
        HashMap<Integer,Integer> map = new HashMap<Integer, Integer>();

        for(int i = 0; i < arr.length; i++) {
            if (map.containsKey(arr[i])) {
                    int count = map.get(arr[i]) +1;
                    if (count > arr.length /2) {
                        System.out.println("Majority found :- " + arr[i]);
                        return;
                    } else
                        map.put(arr[i], count);

            }
            else
                map.put(arr[i],1);
            }
            System.out.println(" No Majority element");
    }


    /* Driver program to test the above functions */
    public static void main(String[] args)
    {
        int a[] = new int[]{2,2,2,2,5,5,2,3,3};

        findMajority(a);
    }
}
// This code is contributed by  karan malhotra
```

Run on IDE

Output:

```
Majority found :- 2
```

Now give a try to below question

Given an array of 2n elements of which n elements are same and the remaining n elements are all different. Write a C program to find out the value which is present n times in the array. There is no restriction on the elements in the array. They are random (In particular they not sequential).

Asked in: Accolite,Amazon,D-E-Shaw,Microsoft

GATE CS Corner Company Wise Coding Practice

Arrays | Majority Element | Moore's Voting Algorithm

Login to Improve this Article

Please write to us at contribute@geeksforgeeks.org to report any issue with the above content.

Recommended Posts:

Find the Number Occurring Odd Number of Times
Given an array A[] and a number x, check for pair in A[] with sum as x
Search an element in a sorted and rotated array
Check for Majority Element in a sorted array
Merge an array of size n into another array of size m+n
K maximum sums of non-overlapping contiguous sub-arrays
Kronecker Product of two matrices

Number of local extrema in an array
Sparse Matrix Representations | Set 3 (CSR)
Possible to make a divisible by 3 number using all digits in an array

(Login to Rate)

2.6 Average Difficulty : **2.6/5.0**
Based on **310** vote(s)

Basic Easy Medium Hard Expert

Add to TODO List
Mark as DONE

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments Share this post!