

It's teamwork, but simpler, more pleasant and more productive.



Custom Search

Practice



Write an Article



Majority Element

Write a function which takes an array and prints the majority element (if it exists), otherwise prints "No 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).

Examples :

Input : {3, 3, 4, 2, 4, 4, 2, 4, 4}

Output : 4

Input : {3, 3, 4, 2, 4, 4, 2, 4}

Output : No Majority Element

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.

Below is the implementation of the above approach :

C++

```
// C++ program to find Majority
// element in an array
#include <bits/stdc++.h>
using namespace std;

// Function to find Majority element
// in an array
void findMajority(int arr[], int n)
{
    int maxCount = 0;
    int index = -1; // sentinels
    for(int i = 0; i < n; i++)
    {
        int count = 0;
        for(int j = 0; j < n; j++)
        {
            if(arr[i] == arr[j])
                count++;
        }

        // update maxCount if count of
        // current element is greater
        if(count > maxCount)
        {
            maxCount = count;
            index = i;
        }
    }

    // if maxCount is greater than n/2
    // return the corresponding element
    if (maxCount > n/2)
        cout << arr[index] << endl;

    else
        cout << "No Majority Element" << endl;
}

// Driver code
int main()
{
    int arr[] = {1, 1, 2, 1, 3, 5, 1};
    int n = sizeof(arr) / sizeof(arr[0]);

    // Function calling
    findMajority(arr, n);
}
```



```
    return 0;  
}
```



Python 3

```
# Python 3 program to find Majority  
# element in an array  
  
# Function to find Majority  
# element in an array  
def findMajority(arr, n):  
  
    maxCount = 0;  
    index = -1 # sentinels  
    for i in range(n):  
  
        count = 0  
        for j in range(n):  
  
            if(arr[i] == arr[j]):  
                count += 1  
  
        # update maxCount if count of  
        # current element is greater  
        if(count > maxCount):  
  
            maxCount = count  
            index = i  
  
    # if maxCount is greater than n/2  
    # return the corresponding element  
    if (maxCount > n//2):  
        print(arr[index])  
  
    else:  
        print("No Majority Element")  
  
# Driver code  
if __name__ == "__main__":  
    arr = [1, 1, 2, 1, 3, 5, 1]  
    n = len(arr)  
  
    # Function calling  
    findMajority(arr, n)  
  
# This code is contributed  
# by ChitraNayal
```



PHP



```
<?php
// PHP program to find Majority
// element in an array

// Function to find Majority element
// in an array
function findMajority($arr, $n)
{
    $maxCount = 0;
    $index = -1; // sentinels
    for($i = 0; $i < $n; $i++)
    {
        $count = 0;
        for($j = 0; $j < $n; $j++)
        {
            if($arr[$i] == $arr[$j])
                $count++;

            // update maxCount if count of
            // current element is greater
            if($count > $maxCount)
            {
                $maxCount = $count;
                $index = $i;
            }
        }

        // if maxCount is greater than n/2
        // return the corresponding element
        if ($maxCount > $n/2)
            echo $arr[$index] . "\n";
        else
            echo "No Majority Element" . "\n";
    }

    // Driver code
    $arr = array(1, 1, 2, 1, 3, 5, 1);
    $n = sizeof($arr);

    // Function calling
    findMajority($arr, $n);

    // This code is contributed
    // by Akanksha Rai
```

Output :





Iguaz

Desde/po
compran
vuelta (ta

1148 /

*Términos y conc



1

Time Complexity : $O(n*n)$.

Auxiliary Space : $O(1)$.

METHOD 2 (Using Binary Search Tree)

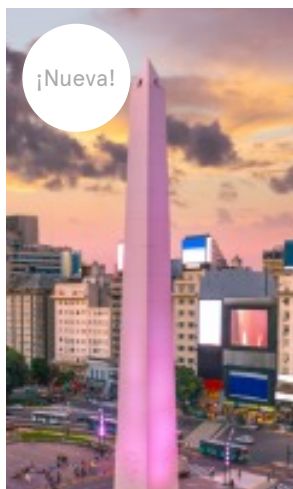
Insert elements in BST one by one and if an element is already present then increment the count of the node. At any stage, if count of a node becomes more than $n/2$ then return.

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 $O(n \log n)$

Auxiliary Space : $O(n)$





Cord

Desde/po
compran
vuelta (ta

998 A

*Términos y conc



METHOD 3 (Using Moore's Voting Algorithm)

This is a two step process.

NOTE : This Method only works when we are given that majority element do exist in the array , otherwise this method won't work , as in the problem definition we said that majority element may or may not exist but for applying this approach you can assume that majority element do exist in the given input array

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 candidate for majority 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 that 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 $a[\text{maj_index}]$. If the next element is same then increment the count, if the next element is not same then decrement the count, and if the count reaches 0 then changes the maj_index to the current element and set the count again to 1. So, the first phase of the algorithm gives us a candidate element.

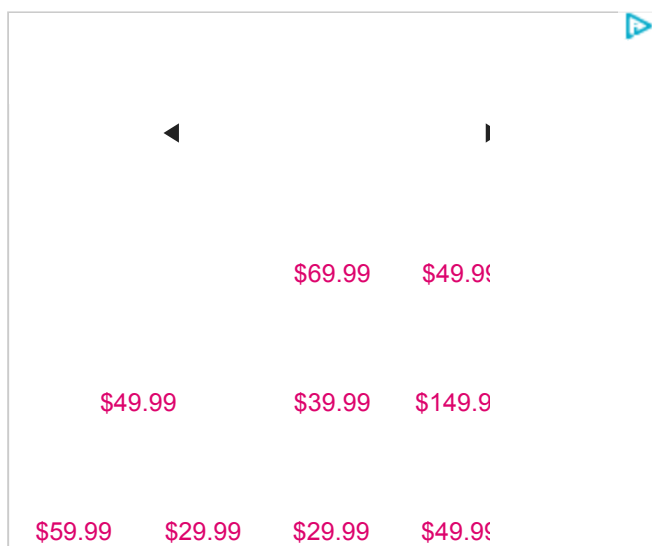
In the second phase we need to check if the candidate is really a majority element. Second phase is simple and can be easily done in $O(n)$. We just need to check if count of the candidate element is greater than $n/2$.

Example :

Let the array be $A[] = 2, 2, 3, 5, 2, 2, 6$

- Initialize $\text{maj_index} = 0$, $\text{count} = 1$
- Next element is 2, which is same as $a[\text{maj_index}] \Rightarrow \text{count} = 2$
- Next element is 3, which is different from $a[\text{maj_index}] \Rightarrow \text{count} = 1$
- Next element is 5, which is different from $a[\text{maj_index}] \Rightarrow \text{count} = 0$
Since $\text{count} = 0$, change candidate for majority element to 5 $\Rightarrow \text{maj_index} = 3$, $\text{count} = 1$
- Next element is 2, which is different from $a[\text{maj_index}] \Rightarrow \text{count} = 0$
Since $\text{count} = 0$, change candidate for majority element to 2 $\Rightarrow \text{maj_index} = 4$
- Next element is 2, which is same as $a[\text{maj_index}] \Rightarrow \text{count} = 2$
- Next element is 6, which is different from $a[\text{maj_index}] \Rightarrow \text{count} = 1$
- Finally candidate for majority element is 2.

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

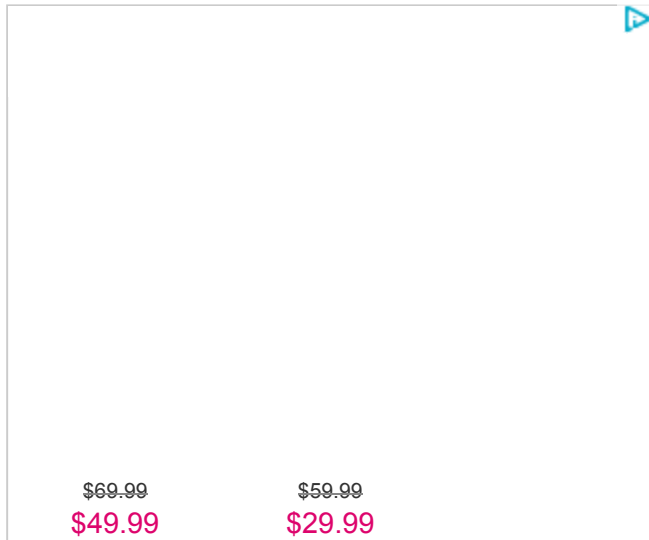




2. Check if the element obtained in step 1 is majority element or not :

```
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 "No Majority Element"



Below is the implementation of the above approach :

C++



```
/* C++ Program for finding out
   majority element in an array */
#include <bits/stdc++.h>
using namespace std;

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

        if (a[i] == cand)
            count++;

    if (count > size/2)
        return 1;

    else
        return 0;
}

/* 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))
        cout << " " << cand << " ";

    else
        cout << "No Majority Element";
}

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

    // Function calling
    printMajority(a, size);

    return 0;
}
```

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;  
}
```



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
```

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)
```

C#

```
// C# Program for finding out majority element in an array
using System;
```

```
class GFG
{
    /* Function to print Majority Element */
    static 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))
            Console.Write(" " + cand + " ");
        else
            Console.Write("No Majority Element");
    }

    /* Function to find the candidate for Majority */
    static 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
    static 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 true;
else
    return false;
}

// Driver Code
public static void Main()
{
    int []a = {1, 3, 3, 1, 2};
    int size = a.Length;
    printMajority(a, size);
}

// This code is contributed by Sam007
```

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).

Time Complexity : $O(n)$

Auxiliary Space : $O(n)$

Below is the implementation.

Java

```
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
```

C#

```
// C# Program for finding out majority
// element in an array
using System;
using System.Collections.Generic;

class GFG
{
    private static void findMajority(int[] arr)
    {
        Dictionary<int,
            int> map = new Dictionary<int,
                int>();

        for (int i = 0; i < arr.Length; i++)
        {
            if (map.ContainsKey(arr[i]))
            {
                int count = map[arr[i]] + 1;
                if (count > arr.Length / 2)
```

```
        {
            Console.WriteLine("Majority found :- " +
                              arr[i]);
            return;
        }
        else
        {
            map[arr[i]] = count;
        }
    }
    else
    {
        map[arr[i]] = 1;
    }
}
Console.WriteLine(" No Majority element");
}

// Driver Code
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 Shrikant13
```

Output:

Majority found :- 2

Thanks Ashwani Tanwar, Karan Malhotra for suggesting this.

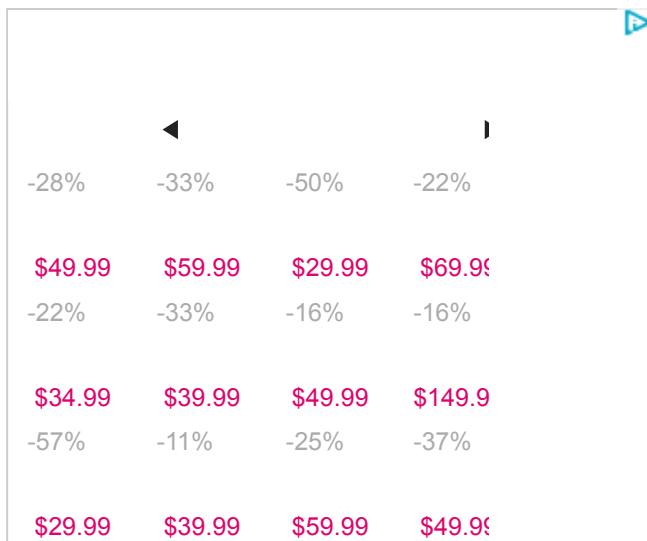


Majority Element | GeeksforGeeks



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).



-28%	-33%	-50%	-22%
\$49.99	\$59.99	\$29.99	\$69.99
-22%	-33%	-16%	-16%
\$34.99	\$39.99	\$49.99	\$149.99
-57%	-11%	-25%	-37%
\$29.99	\$39.99	\$59.99	\$49.99

Recommended Posts:

[Check if an array has a majority element](#)

[Majority element in a circular array of 0's and 1's](#)

[Check for Majority Element in a sorted array](#)

[Majority Element | Set-2 \(Misra-Gries Family of Algorithms\)](#)

Replace each element by the difference of the total size of the array and frequency of that element
Range Query on array whose each element is XOR of index value and previous element
Find last element after deleting every second element in array of n integers
Replace every array element by Bitwise Xor of previous and next element
Longest Subarray with first element greater than or equal to Last element
Replace every element with the greatest element on its left side
Replace every element with the smallest element on its left side
Replace every element of the array by its previous element
Replace every element with the greatest element on right side
Replace every element with the least greater element on its right
Replace every element of the array by its next element

Improved By : tolani62, Ita_c, shrikanth13, Akanksha_Rai

Article Tags : Arrays Accolite Amazon D-E-Shaw Majority Element Microsoft Moore's Voting Algorithm

Practice Tags : Accolite Microsoft Amazon D-E-Shaw Arrays



Be the First to upvote.

2.6

☐ To-do ☐ Done

Based on **362** vote(s)

Feedback

Notes

Improve Article

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

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

Load Comments

Share this post!





A computer science portal for geeks

710-B, Advant Navis Business Park,
Sector-142, Noida, Uttar Pradesh - 201305
feedback@geeksforgeeks.org

COMPANY

About Us
Careers
Privacy Policy
Contact Us

PRACTICE

Company-wise
Topic-wise
Contests
Subjective Questions

LEARN

Algorithms
Data Structures
Languages
CS Subjects
Video Tutorials

CONTRIBUTE

Write an Article
Write Interview Experience
Internships
Videos

@geeksforgeeks, Some rights reserved

