Majority Element

Tuesday, December 21, 2021 4:22 PM

Majority Element

Medium Accuracy: 48.6% Submissions: 100k+ Points: 4

Given an array **A** of **N** elements. Find the majority element in the array. A majority element in an array A of size N is an **element that appears more than N/2 times in the array**.

Example 1:

Input:
N = 3
A[] = {1,2,3}
Output:
-1
Explanation:
Since, each element in
{1,2,3} appears only once so there
is no majority element.

Example 2:

Input:
N = 5
A[] = {3,1,3,3,2}
Output:
3
Explanation:
Since, 3 is present more
than N/2 times, so it is
the majority element.

Your Task:

The task is to complete the function **majorityElement**() which returns the majority element in the array. If no majority exists, return -1.

Expected Time Complexity: O(N).

Expected Auxiliary Space: O(1).

Constraints:

$1 \leq N \leq 10^7$

$0 \leq A_i \leq 10^6$

From <https://practice.geeksforgeeks.org/problems/majority-element-1587115620/1>

View Bookmarked Problems

Company Tags

Accolite Amazon D-E-Shaw Microsoft Nagarro
Google Atlassian Flipkart

View Bookmarked Problems

Company Tags

Accolite Amazon D-E-Shaw Microsoft Nagarro
Google Atlassian Flipkart

- Maq software interview experience set 9 on campus for se 1
- Samsung r d interview questions

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
```

```
Explanation: The frequency of 4 is 5 which is greater
than the half of the size of the array size.
```

```
Input : {3, 3, 4, 2, 4, 4, 2, 4}
Output : No Majority Element
Explanation: There is no element whose frequency is
greater than the half of the size of the array size.
```

METHOD 1

- **Approach:** The basic solution is to have two loops and keep track of the 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 the maximum count doesn't become more than n/2 then the majority element doesn't exist.
- **Algorithm:**
  1. Create a variable to store the max count, *count = 0*
  2. Traverse through the array from start to end.
  3. For every element in the array run another loop to find the count of similar elements in the given array.
  4. If the count is greater than the max count update the max count and store the index in another variable.
  5. If the maximum count is greater than the half the size of the array, print the element. Else print there is no majority element.

Below is the implementation of the above idea:

```
    // 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)
    System.out.println(arr[index]);
  else
    System.out.println("No Majority Element");
}
// Driver code
publicstaticvoidmain(String[] args)
{
  intarr[] ={ 1, 1, 2, 1, 3, 5, 1};
  intn=arr.length;
  // Function calling
  findMajority(arr, n);
}
// This code is contributed by ajit.
}
```

Run

Python3C#PHPJavascript

Output

```
1
```

Complexity Analysis:

- **Time Complexity:** O(n*n).
  A nested loop is needed where both the loops traverse the array from start to end, so the time complexity is O(n^2).
- **Auxiliary Space:** O(1).
  As no extra space is required for any operation so the space complexity is constant.

From <https://practice.geeksforgeeks.org/problems/majority-element-1587115620/1#>

METHOD 2 (Using Binary Search Tree)

- **Approach:** 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 the count of a node becomes more than n/2 then return.
- **Algorithm:**
  1. Create a binary search tree, if same element is entered in the binary search tree the frequency of the node is increased.
  2. traverse the array and insert the element in the binary search tree.
  3. If the maximum frequency of any node is greater than the half the size of the array, then perform a inorder traversal and find the node with frequency greater than half
  4. Else print No majority Element.

Below is the implementation of the above idea:

```
   inorder(root.left, s);
   if(root.c>(s/ 2))
      System.out.println(root.key+"\n");
   inorder(root.right, s);
}
}
// Driver Code
publicstaticvoidmain(String[] args)
{
inta[] ={ 1, 3, 3, 3, 2};
intsize=a.length;
Noderoot=null;
for(inti=0; i<size; i++)
{
   root=insert(root, a[i]);
}
// Function call
if(ma>(size/ 2))
   inorder(root, size);
else
   System.out.println("No majority element\n");
}
}
// This code is contributed by avanitrachhadiya2155
```

Run
Python3C#
Output
3

Complexity Analysis:

- **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 it will be O(nlogn)
- **Auxiliary Space:** O(n).
  As extra space is needed to store the array in tree.

From <https://practice.geeksforgeeks.org/problems/majority-element-1587115620/1#>

METHOD 3 (Using Moore's Voting Algorithm):

- **Approach:** This is a two-step process.
  - The first step gives the element that maybe the 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.
  - Check if the element obtained from the above step is majority element. This step is necessary as there might be no majority element.

- **Algorithm:**
  1. Loop through each element and maintains a count of majority element, and a majority index, *maj_index*
  2. If the next element is same then increment the count if the next element is not same then decrement the count.
  3. if the count reaches 0 then changes the maj_index to the current element and set the count again to 1.
  4. Now again traverse through the array and find the count of majority element found.
  5. If the count is greater than half the size of the array, print the element
  6. Else print that there is no majority element

Below is the implementation of above idea:

```
/* 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 code */
public static void main(String[] args)
{
  MajorityElement majorelement
    =new MajorityElement();
  int a[] =new int[] { 1, 3, 3, 1, 2};

  // Function call
  int size=a.length;
  majorelement.printMajority(a, size);
}
}
// This code has been contributed by Mayank Jaiswal
```

Run

Python C# PHP Javascript

Output

```
No Majority Element
```

Complexity Analysis:

- **Time Complexity:** O(n).
  As two traversal of the array is needed, so the time complexity is linear.
- **Auxiliary Space:** O(1).
  As no extra space is required.

From <https://practice.geeksforgeeks.org/problems/majority-element-1587115620/1#>