Check for Majority Element in a sorted array

Question: Write a C function to find if a given integer x appears more than n/2 times in a sorted array of n integers.

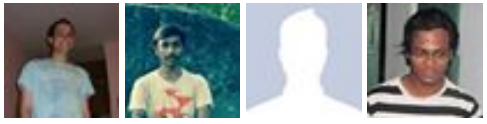
Basically, we need to write a function say isMajority() that takes an array (arr[]), array's size (n) and a number to be searched (x) as parameters and returns true if x is a majority element (present more than n/2 times).

Examples:

```
Input: arr[] = {1, 2, 3, 3, 3, 3, 10}, x = 3
Output: True (x appears more than n/2 times in the given array)

Input: arr[] = {1, 1, 2, 4, 4, 4, 6, 6}, x = 4
Output: False (x doesn't appear more than n/2 times in the given array)

Input: arr[] = {1, 1, 1, 2, 2}, x = 1
Output: True (x appears more than n/2 times in the given array)
```

METHOD 1 (Using Linear Search)
Linearly search for the first occurrence of the element, once you find it (let at index i), check element at index i + n/2. If element is present at i+n/2 then return 1 else return 0.

```
/* Program to check for majority element in a sorted array */
# include <stdio.h>
# include <stdbool.h>

bool isMajority(int arr[], int n, int x)
{
```

Interview Experiences
Advanced Data Structures
Dynamic Programming
Greedy Algorithms
Backtracking
Pattern Searching
Divide & Conquer
Mathematical Algorithms
Recursion

```
  int i;

  /* get last index according to n (even or odd) */
  int last_index = n%2? (n/2+1): (n/2);

  /* search for first occurrence of x in arr[]*/
  for (i = 0; i < last_index; i++)
  {
    /* check if x is present and is present more than n/2 times */
    if (arr[i] == x && arr[i+n/2] == x)
       return 1;
  }
  return 0;
}

/* Driver program to check above function */
int main()
{
   int arr[] ={1, 2, 3, 4, 4, 4, 4};
   int n = sizeof(arr)/sizeof(arr[0]);
   int x = 4;
   if (isMajority(arr, n, x))
     printf("%d appears more than %d times in arr[]", x, n/2);
   else
    printf("%d does not appear more than %d times in arr[]", x, n/2);

   getchar();
   return 0;
}
```

Time Complexity: O(n)

METHOD 2 (Using Binary Search)

Use binary search methodology to find the first occurrence of the given number. The criteria for binary search is important here.

```
/* Program to check for majority element in a sorted array */
# include <stdio.h>;
# include <stdbool.h>

/* If x is present in arr[low...high] then returns the index of
  first occurrence of x, otherwise returns -1 */
int _binarySearch(int arr[], int low, int high, int x);

/* This function returns true if the x is present more than n/2
```

Geometric Algorithms

Popular Posts

All permutations of a given string
Memory Layout of C Programs
Understanding “extern” keyword in C
Median of two sorted arrays
Tree traversal without recursion and without stack!
Structure Member Alignment, Padding and Data Packing
Intersection point of two Linked Lists
Lowest Common Ancestor in a BST.
Check if a binary tree is BST or not
Sorted Linked List to Balanced BST

```
   times in arr[] of size n */
bool isMajority(int arr[], int n, int x)
{
   /* Find the index of first occurrence of x in arr[] */
   int i = _binarySearch(arr, 0, n-1, x);

   /* If element is not present at all, return false*/
   if (i == -1)
     return false;

   /* check if the element is present more than n/2 times */
   if (((i + n/2) <= (n -1)) && arr[i + n/2] == x)
     return true;
   else
     return false;
}

/* If x is present in arr[low...high] then returns the index of
  first occurrence of x, otherwise returns -1 */
int  _binarySearch(int arr[], int low, int high, int x)
{
  if (high >= low)
  {
    int mid = (low + high)/2;  /*low + (high - low)/2;*/

    /* Check if arr[mid] is the first occurrence of x.
        arr[mid] is first occurrence if x is one of the following
        is true:
        (i)  mid == 0 and arr[mid] == x
        (ii) arr[mid-1] < x and arr[mid] == x
     */
    if ( (mid == 0 || x > arr[mid-1]) && (arr[mid] == x) )
      return mid;
    else if (x > arr[mid])
      return _binarySearch(arr, (mid + 1), high, x);
    else
      return _binarySearch(arr, low, (mid -1), x);
  }

  return -1;
}

/* Driver program to check above functions */
int main()
{
   int arr[] = {1, 2, 3, 3, 3, 3, 10};
   int n = sizeof(arr)/sizeof(arr[0]);
```

```
    int x = 3;
    if(isMajority(arr, n, x))
      printf("%d appears more than %d times in arr[]", x, n/2);
    else
     printf("%d does not appear more than %d times in arr[]", x, n/2);

    return 0;
}
```

Time Complexity: O(Logn)
Algorithmic Paradigm: Divide and Conquer

Please write comments if you find any bug in the above program/algorithm or a better way to solve the same problem.

Related Tpoics:

- Remove minimum elements from either side such that 2*min becomes more than max
- Divide and Conquer | Set 6 (Search in a Row-wise and Column-wise Sorted 2D Array)
- Bucket Sort
- Kth smallest element in a row-wise and column-wise sorted 2D array | Set 1

Recent Comments

- Find the number of zeroes
- Find if there is a subarray with 0 sum
- Divide and Conquer | Set 5 (Strassen's Matrix Multiplication)
- Count all possible groups of size 2 or 3 that have sum as multiple of 3

Writing code in comment? Please use **ideone.com** and share the link here.

71 Comments GeeksforGeeks

Sort by Newest

AlienOnEarth • 13 days ago

Instead check first and last occurence of the given number in o(log n) time.

OP Coder • 4 months ago

The solution in more simplifies manner

1) i = n/2

2) x = arr[i] It is the only element which may be majority element.

3) find the leftest index of the element x ie leftest_index = left(0, i-1,x)

4)Must for majority rightest_index = n/2 + 1 + index - 1
if (arr[rightest_index] == x) "Its majority";
else "Its is not majority"

Guest • 4 months ago

If the array is sorted and there is a majority element in it, then it will definitely b
return that element.

• Reply • Share ›

OP Coder Guest • 4 months ago

Element at index n/2 may or may not a majority element. To check that
as we have sorted array.

1 2 3 4 5 6 7 for you i t will return 4 but that is not a majority element. I h

• Reply • Share ›

dippi • 7 months ago

if the array is sorted

if(a[n/2]==x && ((a[0]==x && a[n/4]==x) || (a[n]==x && a[3n/4]==x) || (a[n/4]==x
return true
else
return false

O(1) complexity
correct me if i'm wrong :)

2 • Reply • Share ›

Vivek dippi • 6 months ago

no it isn't correct ..
if starting index of 'x' is between 0 to n/4 (suppose n/8) and ending inde
+1)....
your algo will give an incorrect output

• Reply • Share ›

hello dippi • 7 months ago

i think you are wrong. its not necessary for the majority element to star

Guest • 7 months ago

Don't know if I have not understood the question properly or something else.

we need to find Majority element i.e. that appears more than N/2 times for an a
algorithm? it is trivial that middle element will be the majority element.

Minimum frequency of majority element = n/2 + 1.
Now if majority element is also the smallest element then it is present from arr
consider another case where majority element = largest element of array.
then it is present from
arr[n/2...n]
In average case majority element is always present at middle position.

so we need only one operation to compute majority element in a sorted array.

7 ▲ | ▼ • Reply • Share ›

guest ➦ Guest • 7 months ago

if you know it's appearing more than n/2 times then a[n/2] is the one we
appears more than n/2 or not..

it is so simple that if an element appears more than n/2 times then the
is not true.

1 ▲ | ▼ • Reply • Share ›

Jack • 10 months ago

@geeksforgeeks

If the array is sorted, only the element at a[n/2] has the chance of being the ma
So we can compare x with a[n/2].
If they are not equal, x is not a majority element.
If they are equal, we have to check for 2 cases:

*If a[(n/2)-1] and a[n] are also equal to x.
If any of 2 cases is true, x is a majority element. It is not otherwise.

Please correct me if I am wrong anywhere.

1 ▲ | ▼ • Reply • Share ›

shiv kumar gupta ➦ Jack • 8 months ago

ur approach is correct up to "it must be a middle element " but a[0] anc
only possible way !!!
consider 1 2 3 3 3 3 4 none of ur case satisfy but x=3 is TRUE.. so its
take just O(1)) but then u have to check n/2 indices pivoting around a[n

Hope I m right .. check once :)

▲ | ▼ • Reply • Share ›

Ray ➦ shiv kumar gupta • 8 months ago

Jack's approach can be made to work by slightly modifying the
from the middle, N/4 elements either ways.

```
bool majorSearch(int *a, int x, int n, int low, high)
{
if ( (high - low + 1) > n/2)
{
mid = (low + high)/2;
quartLow = mid - n/4;
quartHigh = mid + (n + 2)/4;
if (a[quartLow ] == x && a[quartHigh] == x)
return True;
else if (a[quartLow] == x)
return majorSearch(a, x, n, low, quartHigh - 1);
else if (a[quartHigh] == x)
return majorSearch(a, x, n, quartLow + 1, high);
```

```
}
return False;
}

bool isMajor(int *a, int x)
{
len = sizeof(a)/sizeof(a[0]);
return majorSearch(a, x, len, 0, len - 1);
}
```


wannaC • 10 months ago

What if array is this: 3 3 3 3 4 5 6 and x = 3? How binary search method is goi

```
/* Paste your code here (You may delete these lines if not writing c
```


Alien ↗ wannaC • 8 months ago

It will work because Binary search method is trying to find the first occu
zero will give x only if the input is sorted.

But the drawback of this method is, it can only be used if input is sorte

kuldeepshandilya • 10 months ago

It can be done in only 3 comparisons - check values of array[0], array[mid] anc
'x', just check for one occurrence of x and we have positive response (x occur

If array[mid] == x and (array[0] == x || array[last] == x)
return true.

Plz let me know If i am missing something!!!

Reply • Share ›

geekguy kuldeepshandilya • 10 months ago

try for 1,2,2,2,1 :)

Reply • Share ›

Jack geekguy • 10 months ago

@kuldeepshandilya: This is exactly what came to my mind.
@geekguy: The array has to be sorted. This is not a sorted arra

Reply • Share ›

geekguy Jack • 10 months ago

oops,

consider it 1,2,2,2,3 :)

Reply • Share ›

geekguy Jack • 10 months ago

Oops,

Consider it 1,2,2,2,3 ;)

Reply • Share ›

kuldeepshandilya Jack • 10 months ago

@Jack
I got where we are wrong. We are thinking that if a numl
then either it would be in subarray from 0 to n/2 or n/2+1
But, it can be like {1,2,2,2,3} in which case neither arr[0]
to find first occurrence of x and last occurrence of it.

Reply • Share ›

geekguy • 10 months ago

@geeksforgeeks

The second method does not return majority element. It's only checking that e
the majority element, It will take O(n log n) time.

Please correct me if I am wrong.

• Reply • Share ›

GeeksforGeeks ➚ geekguy • 10 months ago

This post is about finding whether x is majority or not. To find majority,

• Reply • Share ›

geekguy ➚ GeeksforGeeks • 10 months ago

Ohh, My bad.

Thanks for correcting me.

• Reply • Share ›

Ronny • 10 months ago

@geeksforgeeks
There is a bug in the program (linear method).
the condition which computes last_index is incorrect

It should be
int last_index = n%2? (n/2+1): (n/2);
instead of
int last_index = n%2? n/2: (n/2 + 1);

Since n%2 will be true (return 1) if it is odd and for that condition it should be (r

Please correct it.

```
/* Paste your code here (You may delete these lines if not writing c
```

GeeksforGeeks ➚ Ronny • 10 months ago

Ronny: Thanks for pointing this out. We have updated the code. Appre

Akhil • 10 months ago

@geeksforgeeks
In the linear search method the for condition should be
i<=last_index. Otherwise it would fail for some cases.
The code written above fails for
{1,2,3,4,4,4,4}.

Ronny ➚ Akhil • 10 months ago

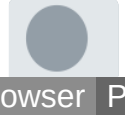
@akhil
The problem is not in the loop condition rather it is in the computation c

the condition which computes last_index is incorrect

It should be
int last_index = n%2? (n/2+1): (n/2);
instead of
int last_index = n%2? n/2: (n/2 + 1);

Since n%2 will be true (return 1) if it is odd and for that condition it shou

You can refer this link where correcting the above condition produces c
http://ideone.com/KhGRGA



ultimate_coder • 11 months ago

Is this code good enough?

```
 bool checkmajority(int a[],int n)
{
    for(int i=0;i<=n/2;i++)
    {
        if(a[i]==a[i+n/2])
        return 1;
    }
    return 0;
}
```

• Reply • Share ›

me.abhinav • a year ago

ASSUMPTION: There exists a majority element for sure.

We can check if a number is the majority element in O(1) time because arr[n/
where n = size of array.

• Reply • Share ›

me.abhinav me.abhinav • a year ago

The must be sorted.

• Reply • Share ›

ramu me.abhinav • 11 months ago

1 3 5

is three majority element even it is mid of sorted array???

• Reply • Share ›

ronny ramu • 11 months ago

guess u missed the ASSUMPTION : the majority eleme

Reply • Share ›

ramu ronny • 11 months ago

hmm if you have already assumed that there exits majo
is wrong.

Reply • Share ›

Abhinav Chauhan • a year ago

ASSUMPTION: There exists a majority element for sure.
We can check if a number is majority element in O(1) time because arr[n/2] w
n = size of array.

Reply • Share ›

Ujjwal • a year ago

```
boolean check_Majority(int A[], int x)
{
int last=sizeof(A)/sizeof(int);
int mid=last/2;
if(A[mid+1]==A[0] || A[mid-1]==A[last-1]) /*to check whether Majority Element a
if(A[mid]==x) /*if present, should be at mid position*/
return true
else
return false
}

#correct me if i m wrong
```

Reply • Share ›

Ankit Malhotra • a year ago

Approach 2 can be further simplified with the complexity reduced to log(n/2) in
term[n/2] of array can be majority. We check for first appearance of this term i

half at all. Then we simply check term[n-1] to match term[n/2] to see majority.

```
 #include <iostream>
#define MaxCount 101
using namespace::std;
typedef unsigned counter;
typedef long element;

// first insert position in sorted order
// returns false with left = count for item > last
// Check boundaries before use
bool firstsortloc (element terms[], counter n, element x, counter & l
{
  counter r = n, m;
  l = 0;
```

see more

Reply • Share ›

Ankit Malhotra Ankit Malhotra • a year ago

To improve firstsortloc change while condition as follows

```
while (l != r && term[l] != item)
```

Reply • Share ›

Ganesh • a year ago

You can find the java code here for Method 2

[sourcecode language="JAVA"]

/**

* Write a C function to find if a given integer x appears more than n/2 times in

Example:
* Input: arr[] = {1, 2, 3, 3, 3, 3, 10}, x = 3
* Output: True (x appears more than n/2 times in the given array)
* @author GAPIITD
*
*/
public class MajorityElementInSortedArray {

public static void main(String[] args) {
int arr[] = {1, 2, 3, 3, 3, 3, 10};
int no = 3;
System.out.println(isMajority(arr, no));
}

see more

Reply • Share ›

Sreekanth • a year ago
Hi folks,

Saw a better solution using Moore's Algorithm, that is already discussed on thi

http://www.geeksforgeeks.org/m...

Reply • Share ›

aygul Sreekanth • a year ago
Moore's Algorithm, runs with O(N) time. Here the problem is different.
in O(logN)

Reply • Share ›

Shubham Lakhiwal • 2 years ago
Common-sense, x must be the n/2th element to occur more than n/2 times.

Palash ➦ Shubham Lakhiwal • 2 years ago

Even if x is n/2th element, it may or may not be the majority element. T
here.

```
/* Paste your code here (You may delete these lines if not wri
```

• Reply • Share ›

Shubham Lakhiwal ➦ Shubham Lakhiwal • 2 years ago

Oh, it's not sure.

• Reply • Share ›

Rahul Sundar • 2 years ago

How about this non-recursive solution. Here I make use of the fact, that we ha
given number(say x). Please let me know your comments,

```
 int Majority(int a[], int n,int x)
{
        int low = 0,high = n-1;
        int middle = (low+high)/2;

        if(a[middle] != x)
        {
                printf("\n%d is not a majority\n",x);
                return -1;
        }

        for(; low<high; middle = (low+high)/2)
        {
                //check if middle element is x
                if(x==a[middle])
```

see more

• Reply • Share ›

Rahul Sundar ➚ Rahul Sundar • 2 years ago

In the above code, we can improve the condition when [middle-1]==x, I so break out.

```
/* Paste your code here (You may delete these lines if not wri
```

• Reply • Share ›

Rahul Sundar ➚ Rahul Sundar • 2 years ago

The idea is to get the beginning index of the majority. Say for array {1,1 majority. Here we have to find beginning index by doing binary search t

Explanation on the above code:

1. Since we say the majority should be more than n/2. Then it should b condition is checked even before entering binary search logic.
2. In binary search the conditions required are,
Condition1: If middle and middle-1 element are x. Then shift the high to
Condition2: If middle is not x then shift the low to middle (low=middle)

Repeat step 2 till low<high

In Condition1, we can also check if middle-1 is zero. If so we can breal not be present in the above code. We can add this too.

[sourcecode language="C"]
/* Paste your code here (You may delete these lines if not writing code

• Reply • Share ›

I am not able to register my comment

```
/* Paste your code here (You may delete these lines if not writing c
```

• Reply • Share ›

Shyam • 2 years ago

@GeeksforGeeks can you please explain the Binary search method? i am not behind it?

• Reply • Share ›

Shyam ➚ Shyam • 2 years ago

I mean the 2nd Method and not Binary search

• Reply • Share ›

tutum • 2 years ago

just return middle element of the array nothing else

reply me if you have any doubt or found me wrong

• Reply • Share ›

Load more comments