Find the largest multiple of 3

Given an array of non-negative integers. Find the largest multiple of 3 that can be formed from array elements.

For example, if the input array is {8, 1, 9}, the output should be “9 8 1″, and if the input array is {8, 1, 7, 6, 0}, output should be “8 7 6 0″.

Method 1 (Brute Force)
The simple & straight forward approach is to generate all the combinations of the elements and keep track of the largest number formed which is divisible by 3.

Time Complexity: O(n x 2^n). There will be 2^n combinations of array elements. To compare each combination with the largest number so far may take O(n) time.
Auxiliary Space: O(n) // to avoid integer overflow, the largest number is assumed to be stored in the form of array.

Method 2 (Tricky)
This problem can be solved efficiently with the help of O(n) extra space. This method is based on the following facts about numbers which are multiple of 3.

1) A number is multiple of 3 if and only if the sum of digits of number is multiple of 3. For example, let us consider 8760, it is a multiple of 3 because sum of digits is 8 + 7+ 6+ 0 = 21, which is a multiple of 3.

2) If a number is multiple of 3, then all permutations of it are also multiple of 3. For example, since 6078 is a multiple of 3, the numbers 8760, 7608, 7068, ….. are also multiples of 3.

3) We get the same remainder when we divide the number and sum of digits of the number. For example, if divide number 151 and sum of it digits 7, by 3, we get the same remainder 1.

What is the idea behind above facts?
The value of 10%3 and 100%3 is 1. The same is true for all the higher powers of 10, because 3 divides 9, 99, 999, … etc.
Let us consider a 3 digit number n to prove above facts. Let the first, second and third digits of n be 'a', 'b' and 'c' respectively. n can be written as

```
n = 100.a + 10.b + c
```

Since (10^x)%3 is 1 for any x, the above expression gives the same remainder as following expression

```
 1.a + 1.b + c
```

So the remainder obtained by sum of digits and 'n' is same.

Following is a solution based on the above observation.

1. Sort the array in non-decreasing order.

2. Take three queues. One for storing elements which on dividing by 3 gives remainder as 0.The second queue stores digits which on dividing by 3 gives remainder as 1. The third queue stores digits which on dividing by 3 gives remainder as 2. Call them as queue0, queue1 and queue2

3. Find the sum of all the digits.

4. Three cases arise:
……**4.1** The sum of digits is divisible by 3. Dequeue all the digits from the three queues. Sort them in non-increasing order. Output the array.

……**4.2** The sum of digits produces remainder 1 when divided by 3.
Remove one item from queue1. If queue1 is empty, remove two items from queue2. If queue2 contains less than two items, the number is not possible.

……**4.3** The sum of digits produces remainder 2 when divided by 3.
Remove one item from queue2. If queue2 is empty, remove two items from queue1. If queue1 contains less than two items, the number is not possible.

5. Finally empty all the queues into an auxiliary array. Sort the auxiliary array in non-increasing order. Output the auxiliary array.

Based on the above, below is the implementation:

```
/* A program to find the largest multiple of 3 from an array of elements */
#include <stdio.h>
#include <stdlib.h>

// A queue node
typedef struct Queue
{
    int front;
    int rear;
    int capacity;
    int* array;
} Queue;

// A utility function to create a queue with given capacity
Queue* createQueue( int capacity )
{
    Queue* queue = (Queue *) malloc (sizeof(Queue));
    queue->capacity = capacity;
    queue->front = queue->rear = -1;
    queue->array = (int *) malloc (queue->capacity * sizeof(int));
    return queue;
}

// A utility function to check if queue is empty
int isEmpty (Queue* queue)
{
    return queue->front == -1;
}

// A function to add an item to queue
void Enqueue (Queue* queue, int item)
{
    queue->array[ ++queue->rear ] = item;
    if ( isEmpty(queue) )
        ++queue->front;
}

// A function to remove an item from queue
int Dequeue (Queue* queue)
{
    int item = queue->array[ queue->front ];
    if( queue->front == queue->rear )
        queue->front = queue->rear = -1;
    else
        queue->front++;

    return item;
}
```

```
// A utility function to print array contents
void printArr (int* arr, int size)
{
    int i;
    for (i = 0; i< size; ++i)
        printf ("%d ", arr[i]);
}

/* Following two functions are needed for library function qsort().
   Refer following link for help of qsort()
   http://www.cplusplus.com/reference/clibrary/cstdlib/qsort/ */
int compareAsc( const void* a, const void* b )
{
    return *(int*)a > *(int*)b;
}
int compareDesc( const void* a, const void* b )
{
    return *(int*)a < *(int*)b;
}

// This function puts all elements of 3 queues in the auxiliary array
void populateAux (int* aux, Queue* queue0, Queue* queue1,
                            Queue* queue2, int* top )
{
    // Put all items of first queue in aux[]
    while ( !isEmpty(queue0) )
        aux[ (*top)++ ] = Dequeue( queue0 );

    // Put all items of second queue in aux[]
    while ( !isEmpty(queue1) )
        aux[ (*top)++ ] = Dequeue( queue1 );

    // Put all items of third queue in aux[]
    while ( !isEmpty(queue2) )
        aux[ (*top)++ ] = Dequeue( queue2 );
}

// The main function that finds the largest possible multiple of
// 3 that can be formed by arr[] elements
int findMaxMultupleOf3( int* arr, int size )
{
    // Step 1: sort the array in non-decreasing order
    qsort( arr, size, sizeof( int ), compareAsc );

    // Create 3 queues to store numbers with remainder 0, 1
    // and 2 respectively
    Queue* queue0 = createQueue( size );
    Queue* queue1 = createQueue( size );
    Queue* queue2 = createQueue( size );

    // Step 2 and 3 get the sum of numbers and place them in
    // corresponding queues
    int i, sum;
    for ( i = 0, sum = 0; i < size; ++i )
    {
        sum += arr[i];
        if ( (arr[i] % 3) == 0 )
            Enqueue( queue0, arr[i] );
        else if ( (arr[i] % 3) == 1 )
            Enqueue( queue1, arr[i] );
        else
            Enqueue( queue2, arr[i] );
    }

    // Step 4.2: The sum produces remainder 1
    if ( (sum % 3) == 1 )
    {
        // either remove one item from queue1
        if ( !isEmpty( queue1 ) )
            Dequeue( queue1 );

        // or remove two items from queue2
        else
        {
            if ( !isEmpty( queue2 ) )
                Dequeue( queue2 );
            else
                return 0;

            if ( !isEmpty( queue2 ) )
                Dequeue( queue2 );
            else
                return 0;
        }
    }

    // Step 4.3: The sum produces remainder 2
    else if ((sum % 3) == 2)
    {
        // either remove one item from queue2
        if ( !isEmpty( queue2 ) )
            Dequeue( queue2 );

        // or remove two items from queue1
        else
        {
            if ( !isEmpty( queue1 ) )
                Dequeue( queue1 );
            else
                return 0;

            if ( !isEmpty( queue1 ) )
                Dequeue( queue1 );
            else
                return 0;
        }
    }

    int aux[size], top = 0;

    // Empty all the queues into an auxiliary array.
    populateAux (aux, queue0, queue1, queue2, &top);

    // sort the array in non-increasing order
    qsort (aux, top, sizeof( int ), compareDesc);

    // print the result
    printArr (aux, top);

    return 1;
}

// Driver program to test above functions
int main()
{
    int arr[] = {8, 1, 7, 6, 0};
    int size = sizeof(arr)/sizeof(arr[0]);

    if (findMaxMultupleOf3( arr, size ) == 0)
        printf( "Not Possible" );

    return 0;
}
```

The above method can be optimized in following ways.
1) We can use Heap Sort or Merge Sort to make the time complexity O(nLogn).

2) We can avoid extra space for queues. We know at most two items will be removed from the input array. So we can keep track of two items in two variables.

3) At the end, instead of sorting the array again in descending order, we can print the ascending sorted array in reverse order. While printing in reverse order, we can skip the two elements to be removed.

The above code works only if the input arrays has numbers from 0 to 9. It can be easily extended for any positive integer array. We just have to modify the part where we sort the array in decreasing order, at the end of code.

Time Complexity: O(nLogn), assuming a O(nLogn) algorithm is used for sorting.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Related Topics:

- Find number of days between two given dates

- How to print maximum number of A’s using given four keys
- Write an iterative O(Log y) function for pow(x, y)
- Visa Interview Experience | Set 6 (On-Campus)
- Count number of ways to reach a given score in a game
- Greedy Algorithm for Egyptian Fraction
- Find length of period in decimal value of 1/n
- How to check if an instance of 8 puzzle is solvable?

Tags: MathematicalAlgo

Like 1 Tweet 0 +1 0

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

49 Comments GeeksforGeeks Login

Recommend 1 Share Sort by Newest

Join the discussion…

Ashish Jaiswal • 2 months ago

i do not understand for brute force method How is it O(n*2^n) ??
for 1,2,3 : no of combination of digit is 1,2,3,13,12,32,123 which is =7 and not 2^3=8 and for comparision how there is N comparision as there are 2^n-1 elements so comaprision should be more....
Help me plzzz

• Reply • Share ›

StrictMath • 2 months ago

An explanation for whom it may concern.

We have the following observations:

1) If any of the digits in the input array is itself a multiple of 3 (0, 3, 6 or 9), we can keep it in the output array as is. This is because they would not affect the "overall sum modulo 3".

2) Whatever may be the size of the input array, we may need to remove AT MOST two elements, to form the output array. If there existed 3 elements say 'a', 'b', 'c', we would have (a + b + c) modulo 3 equal to 1 or 2, for all three to be removed. But sum of 3 positive numbers cannot be less than 3 unless one of them is a zero, in which case by logic (1) we can keep it in the output array.

So basically the problem boils down to finding either of the following:
A) If the sum of the input array modulo 3 is 1:
Find an element in the array (lowest), which leaves 1 as remainder when divided by 3. If there isn't try to remove 2 elements which leave 2 as the remainder.
B) If the sum of the input array modulo 3 is 2:

First try to remove (if there exists) an element which leaves 2 as remainder when divided by 3. If there isn't one, try to remove 2 elements from the input array such that both of them leave 1 as the remainder.

Note that we first try to remove a single digit, because removing 2 digits will make the output array shorter by one digit.

1 • Reply • Share ›

StrictMath StrictMath • 2 months ago

The proof for (2) may be wrong, however the lemma is correct.

• Reply • Share ›

Neha • 2 months ago

Doesn't work for {3,4,1,6}.
q0= empty.
q1= 1,3
q2= 4,6
sum%3 =2 in the end, hence deque q2.
Now:
q0=empty
q1=1,3
q2=6

num=631.

If I am wrong somewhere, please correct me.

• Reply • Share ›

The_Geek Neha • 2 months ago

Please read the article once again !
Everything that you wrote above is wrong starting from creating the queues to the end.

• Reply • Share ›

Kenneth • 3 months ago

My Solution for not only digit array, but can be any number array. For example:

Original Array is:
510 596 421 37 258 588 326 540 364 175 50 174 140 485
Constructed Maximum Number that is multiple of 3 is:
596 588 540 510 485 421 37 364 326 258 175 174 140

http://ideone.com/txZ88G

• Reply • Share ›

Coding_GG Kenneth • 2 months ago

Wow, generalized the original problem and the implemention is very clear and descriptive. Thanks for sharing.

• Reply • Share ›

areen • 4 months ago

does this work for the input {7,7,7}

• Reply • Share ›

The_Geek areen • 2 months ago

absolutely !

• Reply • Share ›

Rush™ • 4 months ago

Wait! So if the numbers should add up and become a multiple of three, we can sort the array in descending order and that would be our output. { 8, 1, 9 } => Sorted => { 9, 8, 1} which is also the max number that is multiple of 3.

• Reply • Share ›

Ashish Jaiswal Rush™ • 2 months ago

or instead of sorting we can simply print it in reverse in O(n) unlike O(nlogn) in case of merge sort.

• Reply • Share ›

Ekta Goel Rush™ • 4 months ago

this is the same as said above.

• Reply • Share ›

jugal • 7 months ago

We can solve the above problem in O(n) using counting sort. Assuming each element in the array is between 0 to 9.

look at a code on following link.
http://ideone.com/ZoXl5w

• Reply • Share ›

Vishal Srivastava • 7 months ago

```
#include<iostream>

#include<algorithm>

#include<queue>

using namespace std;
```

bool asend(int i, int j)

{

return i>j;

}

bool dsend(int i, int j)

{

return i<j; }="" bool="" divisbleby3(int="" arr[] int="" ele)="" {="" bool="" flag="true;" int=""

see more

• Reply • Share ›

Guest • 8 months ago

@GeeksforGeeks
Here is O(n) solution..
Assuming input contains single digits, sorting is unnecessary.
Declare count_array[10].
Scan the input and count for number of 0's , 1's , 2's ...store in count_array
Find sum = i * a[i] // i=0 to 9
Cases
1. sum % 3 == 0 . Output the number from count_array starting from index=9.

2. sum%3 == 1.
decrease count of 1,4,7,2+2,2+5,5+5,2+8,5+8,8+8 in same order

3.sum%3==2
decrease count of 2,5,8,1+1,1+4,1+7,4+4,4+7,7+7 in same order.

Generate number from count_array

8 • Reply • Share ›

neer1304 → Guest • a month ago

Implementation for the above algo :-
http://ideone.com/6KJfq7

• Reply • Share ›

paramvir → Guest • 7 months ago

very elegant solution. +1

1 • Reply • Share ›

Guest • 8 months ago

implementation without using queue

```
/*http://www.geeksforgeeks.org/f...
#include<iostream>
using namespace std;
void findLargestMultiple(int*,int);
void mergesort(int*,int,int);
void merge(int*,int,int,int);
int main()
{
int a[20]={0},i,n,m;
cout<<endl<<"enter number="" of="" elements:="" ";="" cin="">>n;
cout<<endl<<"enter "<<n<<"="" elements:"<<endl;="" for(i="0;i&lt;n;i++)cin">>a[i];
mergesort(a,0,n-1);
findLargestMultiple(a,n);
return 0;
}
void findLargestMultiple(int *a,int n)
{
```

see more

• Reply • Share ›

SĀnjiv Kumar • 10 months ago

we can implement it without queue
just count no of elements congruent to 1mod3 and 2mod3 and
using some rules ....
O(n) time and O(1)space to find included digits
http://ideone.com/e.js/gDHYDj

• Reply • Share ›

ryan SĀnjiv Kumar • 7 months ago

ur sorting takes n^2 time

• Reply • Share ›

Guest • 10 months ago

can't we do like this?
Example :
[8,1,7,6,0]
1. Sort in descending order = [8,7,6,1,0]

2. if sum % 3 == 0 return list/array
Else:

7 • Reply • Share ›

PRADIP_ACHARJEE • 10 months ago

ok...its a really nice approach.
I think we can choose the element to be discard more easily from queue1 and queue2.
As like:-

Queue0-->> Yeah all elements are considerable as they are divisible by 3.
suppose, queue1 and queue2 has the following elements--->

Queue1--->> 1, 7,7,7 // so size is = 4 ,(produces reminder 1)
Queue2--->> 5,8,8 // so size is = 3 ,(produces reminder 2)

so if you sum up all the elements of Queue1 and Queue2 and divide the sum by 3 ------what will be the remainder??!!!

(size_of_queue1 * 1 + size_of_queue2 * 2) % 3 = (4*1+3*2) %3 = 1

so
Logic-1>> If remainder is 1; Discard an element from queue1.
Logic-2>> if remainder is 2; discard an element from queue2.

Logic-3>> if remainder is 0; No need to discard

see more

• Reply • Share ›

Hawk Eye • 2 years ago
"The above code works only if the input arrays has numbers from 0 to 9. It can be easily extended for any positive integer array."
I&#039m sorry... but I think for other integers it would be a bit difficult...
like if the sum of digits produces remainder 2...
and q1={1,1,...} and q2={23,26,....}
then I think it&#039s optimal to remove two item (1 and 1) from q1 than removing only single item (23) from q2....
correct me if I&#039m wrong... :)
1 • Reply • Share ›

Hawk Eye • 2 years ago
"The above code works only if the input arrays has numbers from 0 to 9. It can be easily extended for any positive integer array."
I&#039m sorry... but I think for other integers it would be a bit difficult...
like if the sum of digits produces remainder 2...
and q1={1,1,...} and q2={23,26,....}
then I think it&#039s optimal to remove two item (1 and 1) from q1 than removing only single item (23) from q2....
correct me if I&#039m wrong... :)
• Reply • Share ›

raghson • 2 years ago

I could not get that why we need to sort the array in ascending order initially. I don't think sorting is needed initially.

• Reply • Share ›

Anon raghson • a year ago

so that we do not remove larger numbers for smaller numbers :)

• Reply • Share ›

thepace • 2 years ago

Hi,
I think the above solution can be simplified with the above solution of counting sort.
Here is my solution: "http://codepad.org/UdX8EOs7"
Steps:
a)Get the input..<output: arr[])="" b)do="" counting="" sort.<output:="" count_sort[])="" c)calculate="" the="" two="" values:="" i)rem_one:="" stores="" the="" number="" of="" numbers="" with="" remainder="" 1;="" i)rem_two:="" stores="" the="" number="" of="" numbers="" with="" remainder="" 2;="" d)adjust="" count_sort="" array="" so="" that="" its="" divisible="" by="" 3.="" func:="" make_rem_zero(..)="" step1:="" calculate="" the="" final="" remainder="((rem_two&lt;&lt;1)+rem_one)%3;" step2:="" if="" final_rem="1" if="" rem_one="">0 => decrease count_sort[3*i+1]
else if rem_two>1 => decrease count_sort[3*i+2] twice.
else return false;
else
If rem_two>0 => decrease count_sort[3*i+2]
else if rem_one>1 => decrease count_sort[3*i+1] twice.
else return false;

e)Print count_sort array with the "i" value for count_sort[i] times.

Complexity: O(n);

• Reply • Share ›

Nguyen Ngoc Hoang • 2 years ago

No need to calculate the sum, just calculate the number of elements n1 and n2 in queue 1 and 2. Then, sum = n1 + 2 * n2.

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

• Reply • Share ›

Nguyen Ngoc Hoang Nguyen Ngoc Hoang • 2 years ago

```
// Step 2 and 3 get the sum of numbers and place them in
   // corresponding queues
```

```
    int i, sum;
    for ( i = 0, sum = 0; i < size; ++i )
    {
        //sum += arr[i];
        if ( (arr[i] % 3) == 0 )
            Enqueue( queue0, arr[i] );
        else if ( (arr[i] % 3) == 1 ) {
            sum ++;
            Enqueue( queue1, arr[i] );
        }
        else{
            Enqueue( queue2, arr[i] );
            sum += 2;
        }
    }
```

• Reply • Share ›

Jagan • 2 years ago

i have a solution which does not need sorting. i am not sure if it is correct.

1. find the smallest and 2nd smallest number in the given unsorted array, WHICH ARE NOT MULTIPLES OF 3. (o(n) steps).

2. find sum of all numbers.
if sum%3==0, return.
else
sumnew=sum-smallest;
check if sumnew%3==0

else
sumnew=sum-second smallest;
check if sumnew%3==0

else
if (smallest+second smallest)%3!=0
sumnew=sum-(smallest+second smallest);

I think this takes o(n) + constant steps with constant(2) extra space.

• Reply • Share ›

Aashish Jagan • 2 years ago

Without sorting, how can we ensure that the number so formed is the largest?
We need sorting at least once, either prior to applying algo or post to applying algo.

• Reply • Share ›

Jenish • 2 years ago

Below is the method which I think will do the required stuff. To avoid sorting method.. I have assumed that array is already sorted when this method is called.

```
void findMaxMultupleOf3( int[] arr, int size )
       {
               int[] temp = new int[arr.length];
               int totalSum = 0;
               int[] count = {0,0,0};
               //Sort array here if its not sorted already

               for(int i =0;i<arr.length;i++){
                       temp[i]=arr[i]%3;
                       totalSum += arr[i];
                       count[temp[i]]++;
               }
               int toBeRemoved = 0;
               if(totalSum%3 == 1){
                       if(count[1]>0){
```

see more

• Reply • Share ›

VCD • 2 years ago

The code here is wrong, if arr[]=(81,9) the result must 9 81 not 81 9.

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

• Reply • Share ›

Kartik → VCD • 2 years ago

Please take a closer look at the post. It says The above code works only if the input arrays has numbers from 0 to 9. It can be easily extended for any positive integer array. We just have to modify the part where we sort the array in decreasing order, at the end of code.

1 • Reply • Share ›

helper → Kartik • 6 months ago

:D then also the post is inefficient(if not wrong). coz they have written the worst case complexity as O(nlgn) but its O(n)

• Reply • Share ›

nm • 2 years ago

Shouldnt the brute force time complexity be O(2^n * nlgn). Where nlgn is used to sort every combination?

• Reply • Share ›

Kartik nm • 2 years ago

@nm: The time complexity should be O(n x 2^n). It will take at most O(n) time to compare the current combination with the largest so far. We just have compare element by element. Please let me know if you think otherwise.

• Reply • Share ›

Worldcreator • 2 years ago

```
 #include<stdio.h>
#define MAX 10000000
int arr[MAX];
int main()
{
   int i,k,j,sum=0;

    for(i=0;i<MAX;i++)
    {   scanf("%d",&arr[i]);
        sum+=arr[i];
    }

    apply merge sort or quick sort to sort the array

    for(i=0;i<MAX;i++)
        printf("%d  ",arr[i]);

    if(sum%3==0){
```

see more

• Reply • Share ›

sk Worldcreator • 2 years ago

```
 How will it work for 5,3,2 O/P should be 3
can u please elaborate.
```

• Reply • Share ›

Worldcreator Worldcreator • 2 years ago

no need to take extra space, just we can do as i did ...

• Reply • Share ›

hary • 2 years ago

It seems - as already called out - one does not need any queue.

I feel the following steps are sufficient enough for the solution
1. Sort in descending order.
2. find the sum of the digits.
3. if ((sum % 3) == 0) return array
4. if ((sum % 3) == 1)
There exists at least 1 elements which gives a remainder of 1 when divided by 3 or at least 2 elements which lead to a remainder of 2 each (when divided by 3).

4.a start from the end and find out the 1 element which has a remainder of 1 remove it (set this to -1)
4.b If element not found in step 4.a. start from the end and search for first two elements that have a remainder of 2 each when divided by 3 and set them to -1.
4.c If 4.a. and 4.b yield nothing i.e. no array item set to -1 you are in a mess (something wrong right from the start)

5. if ((sum % 3) == 2) there exists at least one element with remainder as 2 when divided by 3 or at least two elements each yielding remainder as 1 when divided by 3.
We have 5.a, 5.b similar to 4.a and 4.b

2 • Reply • Share ›

Kartik hary • 2 years ago
Thanks for compiling the complete approach with all optimizations and suggestions. We will add it to the original post.

• Reply • Share ›

vikramgoyal • 2 years ago
For the case when array contains only digits we can use counting sort.

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

• Reply • Share ›

Anil • 2 years ago
I don't know why you are using mergesort or heapsort where you can easily use counting sort which is of O(n) time.

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

• Reply • Share ›

Kartik Anil • 2 years ago
Counting sort is a good option when array elements are in the range from 0 to 9. The counting sort solution cannot be extended for an array like {1222, 12, 234, 999}

^ | ⌄ • Reply • Share ›

sk • 2 years ago

```
 What abt this:
1. Sort in non decreasing order
2. find sum of(S) all digits
3. find remainder(S%3).
4. if remainder is 0. return value
else
  Just need to search a digit from end to start such that rem or rem+3 or rem+6 is present


Ex. 8,1,7,6,0
after step:
1. 87610
2. 22
3. 1
4. need to search for first occurrence of 1 or 4 or 7 from end to start, 1 is present so re
```

^ | ⌄ • Reply • Share ›

gu ➦ sk • 2 years ago

Your solution is not correct, because the remainder of the sum can not only generate from one digit.
For example, if the array is {2,3,5}, with sum of 10, the remainder of their sum is 1. But there is not any digit with the remainder of 1 or 4 or 7.Because the remainder of their sum is caused by the remainder 2 from 2 and the remainder 2 from 5.

Sorry for my poor English :)

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

^ | ⌄ • Reply • Share ›

sk ➦ gu • 2 years ago

```
Yes, u r right
```

^ | ⌄ • Reply • Share ›

sk ➦ sk • 2 years ago

complexity: nlogn
No auxiliary space

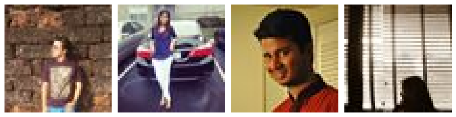
^ | ⌄ • Reply • Share ›

Subscribe Add Disqus to your site Privacy

-
- GeeksforGeeks

  Like

  93,005 people like GeeksforGeeks.

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

- Follow @GeeksforGeeks Subscribe

- ## Recent Comments

  - Jerry Goyal

    instead, use a global variable flag to check...

    Write C Code to Determine if Two Trees are Identical · 25 minutes ago

  - THE SAINT

    Recursive Solution: int palindrome = 1; int...

    Function to check if a singly linked list is palindrome · 48 minutes ago

  - Aryan Parker

    In the above code if we change the order of the...

    Construct Tree from given Inorder and Preorder traversals · 1 hour ago

  - cfh

    (low+high) can overflow but (high-low) can not.

    Find the missing number in Arithmetic Progression · 1 hour ago

  - Anand Barnwal

    In order to maintain the same complexity, you...

    Given an array A[] and a number x, check for pair in A[] with sum as x · 2 hours ago

  - Elangovan Manickam

@Anand, How we can identify the number to add...

Given an array A[] and a number x, check for pair in A[] with sum as x · 3 hours ago

-