Home Algorithms DS GATE Interview Corner Q&A C C++ Java Books Contribute Ask a Q About
Array Bit Magic C/C++ Articles GFacts Linked List MCQ Misc Output String Tree Graph

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.

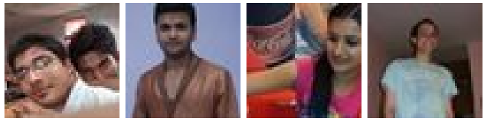
Time Complexity: O(n x 2^n). There will be 2^n combinations of array elements. To compare each combination with the largest number so far may take O(n) time.
Auxiliary Space: O(n) // to avoid integer overflow, the largest number is assumed to be stored in the form of array.

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



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

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

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

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 elemen
#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 )
```

705 Subscribe

Recent Comments

Abhi You live US or India?

Google (Mountain View) interview · 15 minutes ago

Aman Hi, Why arent we checking for conditions...

Write a C program to Delete a Tree. · 55 minutes ago

kzs please provide solution for the problem...

Backtracking | Set 2 (Rat in a Maze) · 58 minutes ago

Sanjay Agarwal bool tree::Root_to_leaf_path_given_sum(tree...

Root to leaf path sum equal to a given number · 1 hour ago

GOPI GOPINATH @admin Highlight this sentence "We can easily...

Count trailing zeroes in factorial of a number · 1 hour ago

newCoder3006 If the array contains negative numbers also. We...

Find subarray with given sum · 1 hour ago

```
{
    // 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 Tpoics:

- Backtracking | Set 8 (Solving Cryptarithmetic Puzzles)
- Tail Recursion
- Find if two rectangles overlap
- Analysis of Algorithm | Set 4 (Solving Recurrences)
- Print all possible paths from top left to bottom right of a mXn matrix
- Generate all unique partitions of an integer
- Russian Peasant Multiplication
- Closest Pair of Points | O(nlogn) Implementation

1 Tweet 0 0

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

26 Comments **GeeksforGeeks**

Sort by Newest ▾

Join the discussion…

Hawk Eye • 9 months ago

"The above code works only if the input arrays has numbers from 0 to 9. It car
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 remov
correct me if I&#039m wrong... :)

Hawk Eye • 9 months ago

"The above code works only if the input arrays has numbers from 0 to 9. It car
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...

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 remov
correct me if I&#039m wrong... :)

raghson • 10 months ago

I could not get that why we need to sort the array in ascending order initially. I c

Anon ➦ raghson • 24 days ago

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

thepace • a year ago

Hi,
I think the above solution can be simplified with the above solution of counting
Here is my solution: "http://codepad.org/UdX8EOs7"
Steps:
a)Get the input..<output: arr[])="" b)do="" counting="" sort.<output:="" count_s
values:="" i)rem_one:="" stores="" the="" number="" of="" numbers="" with="
stores="" the="" number="" of="" numbers="" with="" remainder="" 2;="" d)adj
that="" its="" divisible="" by="" 3.="" func:="" make_rem_zero(..)="" step1:="" c
remainder="((rem_two&lt;&lt;1)+rem_one)%3;" step2:="" if="" final_rem="1" if=
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);

Nguyen Ngoc Hoang • a year ago

No need to calculate the sum, just calculate the number of elements n1 and n
2 * n2.

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

Nguyen Ngoc Hoang ➦ Nguyen Ngoc Hoang • a year 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;
       }
   }
```

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, WHI
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.

Aashish ➦ Jagan • 2 years ago

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

Jenish • 2 years ago

Below is the method which I think will do the required stuff. To avoid sorting m
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

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 c
```

Kartik ➜ VCD • 2 years ago

Please take a closer look at the post. It says The above code works or
0 to 9. It can be easily extended for any positive integer array. We just l
the array in decreasing order, at the end of code.

nm • 2 years ago

ShoudInt the brute force time complexity be O(2^n * nlgn) . Where nlgn is use

Kartik nm • 2 years ago

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

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]);
```

see more

sk Worldcreator • 2 years ago

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

Worldcreator Worldcreator • 2 years ago

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

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
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 re
4.b If element not found in step 4.a. start from the end and search for first two
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 (s

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

Kartik ➦ hary · 2 years ago

Thanks for compiling the complete approach with all optimizations and original post.

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 co
```

Anil · 2 years ago

I don't know why you are using mergesort or heapsort where you can easily us

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

Kartik ➦ Anil · 2 years ago

Counting sort is a good option when array elements are in the range fr cannot be extended for an array like {1222, 12, 234, 999}

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).
```

```
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

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 sta
```

gu ➦ sk • 2 years ago

Your solution is not correct, because the remainder of the sum can no
For example, if the array is {2,3,5}, with sum of 10, the remainder of th
with the remainder of 1 or 4 or 7.Because the remainder of their sum is
and the remainder 2 from 5.

Sorry for my poor English :)

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

sk ➦ gu • 2 years ago

```
Yes, u r right
```

sk ➦ sk • 2 years ago

complexity: nlogn

No auxiliary space