Monday, December 27, 2021 12:02 AM

Product array puzzle

Easy Accuracy: 47.32% Submissions: 58829 Points: 2

Given an array **nums[]** of size **n**, construct a Product Array **P** (of same size n) such that **P[i]** is equal to the product of all the elements of **nums** except nums[i].

Example 1:

Input:
n = 5
nums[] = {10, 3, 5, 6, 2}
Output:
180 600 360 300 900
Explanation:
For i=0, P[i] = 3*5*6*2 = 180.

For i=1, P[i] = 10*5*6*2 = 600.
For i=2, P[i] = 10*3*6*2 = 360.
For i=3, P[i] = 10*3*5*2 = 300.
For i=4, P[i] = 10*3*5*6 = 900.

Example 2:

Input:
n = 2
nums[] = {12,0}
Output:
0 12
Your Task:

You do not have to read input. Your task is to complete the function **productExceptSelf()** that takes array nums[] and n as input parameters and returns a list of n integers denoting the product array P. If the array has only one element the returned list should should contains one value i.e {1}

Note: Try to solve this problem without using the division operation.

From <https://practice.geeksforgeeks.org/problems/product-array-puzzle4525/1>

Expected Time Complexity: O(n)

Expected Auxiliary Space: O(n)

HINT:

mathematically: $e^{(\ln x)} = x$
in java:
Math.pow(Math.E, (Math.log(x)) == x;

From <https://stackoverflow.com/questions/18209676/how-to-find-antilog-for-a-number-using-java-programm>

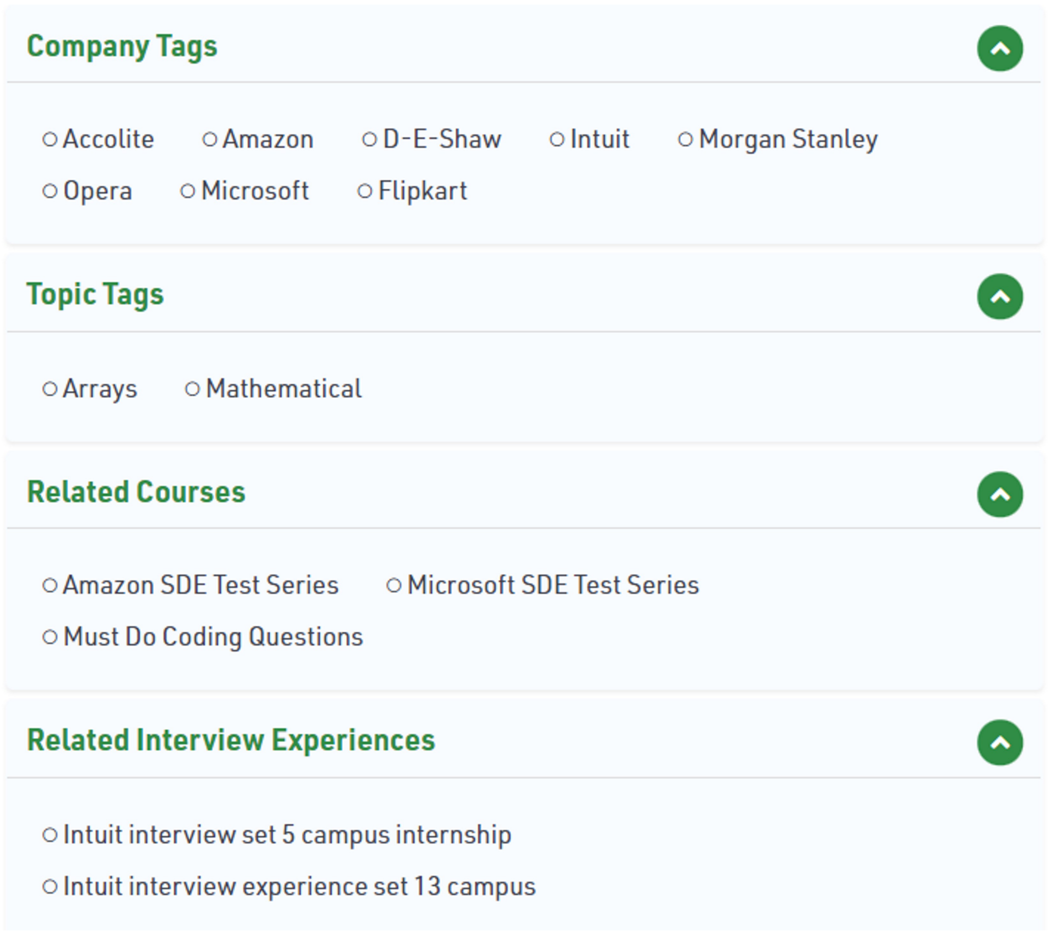
Constraints:

1 <= n <= 1000

0 <= $nums_i$ <= 200

Array may contain duplicates.

From <https://practice.geeksforgeeks.org/problems/product-array-puzzle4525/1>



Company Tags

- Accolite
- Amazon
- D-E-Shaw
- Intuit
- Morgan Stanley
- Opera
- Microsoft
- Flipkart

Topic Tags

- Arrays
- Mathematical

Related Courses

- Amazon SDE Test Series
- Microsoft SDE Test Series
- Must Do Coding Questions

Related Interview Experiences

- Intuit interview set 5 campus internship
- Intuit interview experience set 13 campus

Product of Array except itself

- Difficulty Level : Medium
- Last Updated : 15 Dec, 2021

Given an array arr[] of n integers, construct a Product Array prod[] (of same size) such that prod[i] is equal to the product of all the elements of arr[] except arr[i]. Solve it **without division operator in O(n) time**.

Example :

```
Input: arr[]  = {10, 3, 5, 6, 2}
Output: prod[]  = {180, 600, 360, 300, 900}
3 * 5 * 6 * 2 product of other array
elements except 10 is 180
10 * 5 * 6 * 2 product of other array
```

```
elements except 3 is 600
10 * 3 * 6 * 2 product of other array
elements except 5 is 360
10 * 3 * 5 * 2 product of other array
elements except 6 is 300
10 * 3 * 6 * 5 product of other array
elements except 2 is 900
Input: arr[]  = {1, 2, 3, 4, 5}
Output: prod[]  = {120, 60, 40, 30, 24 }
2 * 3 * 4 * 5  product of other array
elements except 1 is 120
1 * 3 * 4 * 5  product of other array
elements except 2 is 60
1 * 2 * 4 * 5  product of other array
elements except 3 is 40
1 * 2 * 3 * 5  product of other array
elements except 4 is 30
1 * 2 * 3 * 4  product of other array
elements except 5 is 24
```

Recommended: Please solve it on "***PRACTICE*** " first, before moving on to the solution.

Naive Solution:

Approach: Create two extra space, i.e. two extra arrays to store the product of all the array elements from start, up to that index and another array to store the product of all the array elements from the end of the array to that index.

To get the product excluding that index, multiply the prefix product up to index i-1 with the suffix product up to index i+1.

Algorithm:

1. Create two array *prefix* and *suffix* of length *n*, i.e length of the original array, initialize *prefix[0] = 1* and *suffix[n-1] = 1* and also another array to store the product.
2. Traverse the array from second index to end.
3. For every index *i* update *prefix[i]* as *prefix[i] = prefix[i-1] * array[i-1]*, i.e store the product upto *i-1* index from the start of array.
4. Traverse the array from second last index to start.
5. For every index *i* update *suffix[i]* as *suffix[i] = suffix[i+1] * array[i+1]*, i.e store the product upto *i+1* index from the end of array
6. Traverse the array from start to end.
7. For every index *i* the output will be *prefix[i] * suffix[i]*, the product of the array element except that element.

- C++
- C
- Java
- Python3
- C#
- PHP
- Javascript

```java
class ProductArray {
    /* Function to print product array
    for a given array arr[] of size n */
    void productArray(int arr[], int n)
    {

        // Base case
        if (n == 1) {
            System.out.print(0);
            return;
        }
        // Initialize memory to all arrays
        int left[] = new int[n];
        int right[] = new int[n];
        int prod[] = new int[n];

        int i, j;

        /* Left most element of left array
is always 1 */
        left[0] = 1;

        /* Rightmost most element of right
array is always 1 */
        right[n - 1] = 1;

        /* Construct the left array */
        for (i = 1; i < n; i++)
            left[i] = arr[i - 1] * left[i - 1];

        /* Construct the right array */
        for (j = n - 2; j >= 0; j--)
            right[j] = arr[j + 1] * right[j + 1];

        /* Construct the product array using
        left[] and right[] */
        for (i = 0; i < n; i++)
            prod[i] = left[i] * right[i];

        /* print the constructed prod array */
        for (i = 0; i < n; i++)
            System.out.print(prod[i] + " ");

        return;
    }

    /* Driver program to test the above function */
    public static void main(String[] args)
    {
        ProductArray pa = new ProductArray();
        int arr[] = { 10, 3, 5, 6, 2 };
        int n = arr.length;
        System.out.println("The product array is : ");
        pa.productArray(arr, n);
    }
}

// This code has been contributed by Mayank Jaiswal
```

Output

```
The product array is:
180 600 360 300 900
```

- C++
- C
- Java
- Python3
- C#
- PHP
- Javascript

```
class ProductArray {
    /* Function to print product array
    for a given array arr[] of size n */
    void productArray(int arr[], int n)
    {

        // Base case
        if (n == 1) {
            System.out.print(0);
            return;
        }
        // Initialize memory to all arrays
        int left[] = new int[n];
        int right[] = new int[n];
        int prod[] = new int[n];

        int i, j;

        /* Left most element of left array
is always 1 */
        left[0] = 1;

        /* Rightmost most element of right
array is always 1 */
        right[n - 1] = 1;

        /* Construct the left array */
        for (i = 1; i < n; i++)
            left[i] = arr[i - 1] * left[i - 1];

        /* Construct the right array */
        for (j = n - 2; j >= 0; j--)
            right[j] = arr[j + 1] * right[j + 1];

        /* Construct the product array using
        left[] and right[] */
        for (i = 0; i < n; i++)
            prod[i] = left[i] * right[i];

        /* print the constructed prod array */
        for (i = 0; i < n; i++)
            System.out.print(prod[i] + " ");

        return;
    }

    /* Driver program to test the above function */
    public static void main(String[] args)
    {
        ProductArray pa = new ProductArray();
        int arr[] = { 10, 3, 5, 6, 2 };
        int n = arr.length;
        System.out.println("The product array is : ");
        pa.productArray(arr, n);
    }
}

// This code has been contributed by Mayank Jaiswal
```

Complexity Analysis:

- **Time Complexity:** O(n).
  The array needs to be traversed three times, so the time complexity is O(n).
- **Space Complexity:** O(n).
  Two extra arrays and one array to store the output is needed so the space complexity is O(n)
  Note: The above method can be optimized to work in space complexity O(1). Thanks to Dileep for suggesting the below solution.

Efficient solution:

Approach: In the previous solution, two extra arrays were created to store the prefix and suffix, in this solution store the prefix and suffix product in the output array (or product array) itself. Thus reducing the space required.

Algorithm:

8. Create an array *product* and initialize its value to 1 and a variable *temp* = 1.
9. Traverse the array from start to end.
10. For every index *i* update *product[i]* as *product[i] = temp* and *temp = temp * array[i]*, i.e store the product upto *i-1* index from the start of array.
11. initialize temp = 1 and traverse the array from last index to start.
12. For every index *i* update *product[i]* as *product[i] = product[i] * temp* and *temp = temp * array[i]*, i.e multiply with the product upto *i+1* index from the end of array.
13. Print the product array.

- C++
- Java
- Python3
- C#
- PHP
- Javascript

```
class ProductArray {
    void productArray(int arr[], int n)
    {

        // Base case
        if (n == 1) {
            System.out.print("0");
            return;
        }
        int i, temp = 1;

        /* Allocate memory for the product array */
        int prod[] = new int[n];

        /* Initialize the product array as 1 */
        for (int j = 0; j < n; j++)
            prod[j] = 1;

        /* In this loop, temp variable contains product of
           elements on left side excluding arr[i] */
        for (i = 0; i < n; i++) {
            prod[i] = temp;
            temp *= arr[i];
        }

        /* Initialize temp to 1 for product on right side */
        temp = 1;

        /* In this loop, temp variable contains product of
           elements on right side excluding arr[i] */
        for (i = n - 1; i >= 0; i--) {
            prod[i] *= temp;
            temp *= arr[i];
        }

        /* print the constructed prod array */
        for (i = 0; i < n; i++)
            System.out.print(prod[i] + " ");

        return;
    }

    /* Driver program to test above functions */
    public static void main(String[] args)
    {
        ProductArray pa = new ProductArray();
        int arr[] = { 10, 3, 5, 6, 2 };
        int n = arr.length;
        System.out.println("The product array is : ");
        pa.productArray(arr, n);
    }
}

// This code has been contributed by Mayank Jaiswal
```
Output

```
The product array is:
180 600 360 300 900
```
Complexity Analysis:

- **Time Complexity:** O(n).
  The original array needs to be traversed only once, so the time complexity is constant.
- **Space Complexity:** O(n).
  Even though the extra arrays are removed, the space complexity remains O(n), as the product array is still needed.

Another Approach:

Store the product of all the elements is a variable and then iterate the array and add product/current_index_value in a new array. and then return this new array.

Below is the implementation of the above approach:

- C++
- Java
- Python3
- C#
- Javascript

```
// Java program for the above approach
import java.io.*;
import java.util.*;

class Solution {

    public static long[] productExceptSelf(int a[], int n)
    {
        long prod = 1;
        long flag = 0;

        // product of all elements
        for (int i = 0; i < n; i++) {

            // counting number of elements
            // which have value
            // 0
            if (a[i] == 0)
                flag++;
            else
                prod *= a[i];
        }

        // creating a new array of size n
        long arr[] = new long[n];
        for (int i = 0; i < n; i++) {

            // if number of elements in
            // array with value 0
            // is more than 1 than each
            // value in new array
            // will be equal to 0
            if (flag > 1) {
                arr[i] = 0;
            }

            // if no element having value
            // 0 than we will
            // insert product/a[i] in new array
            else if (flag == 0)
                arr[i] = (prod / a[i]);

            // if 1 element of array having
            // value 0 than all
            // the elements except that index
            // value , will be
            // equal to 0
            else if (flag == 1 && a[i] != 0) {
                arr[i] = 0;
            }

            // if(flag == 1 && a[i] == 0)
            else
                arr[i] = prod;
        }
        return arr;
    }

    // Driver Code
    public static void main(String args[])
        throws IOException
    {
        int n = 5;
        int[] array = { 10, 3, 5, 6, 2 };

        Solution ob = new Solution();
        long[] ans = new long[n];
        ans = ob.productExceptSelf(array, n);

        for (int i = 0; i < n; i++) {
            System.out.print(ans[i] + " ");
        }
    }
}

// This code is contributed by Kapil Kumar (kapilkumar2001)
```

Output

```
180 600 360 300 900
```

Time Complexity: O(n)

Space Complexity: O(1)

The original array needs to be traversed only once, so the space complexity is constant.

A product array puzzle | Set 2 (O(1) Space)

- Difficulty Level : Hard
- Last Updated : 19 May, 2021

Given an array arr[] of n integers, construct a Product Array prod[] (of same size) such that prod[i] is equal to the product of all the elements of arr[] except arr[i]. Solve it **without division operator** and in O(n).

Example:

```
Input: arr[] = {10, 3, 5, 6, 2}
Output: prod[] = {180, 600, 360, 300, 900}
The elements of output array are
{3*5*6*2, 10*5*6*2, 10*3*6*2,
10*3*5*2, 10*3*5*6}
Input: arr[] = {1, 2, 1, 3, 4}
Output: prod[] = {24, 12, 24, 8, 6}
The elements of output array are
{3*4*1*2, 1*1*3*4, 4*3*2*1,
1*1*4*2, 1*1*3*2}
```

Recommended: Please solve it on "***PRACTICE*** " first, before moving on to the solution.

There is already a discussed O(n) approach in A product array puzzle | set 1. The previous approach uses extra O(n) space for constructing product array.

Solution 1: Using log property.

Approach: In this post, a better approach has been discussed which uses log property to find the product of all elements of the array except at a particular index. This approach uses no extra space.

Use property of log to multiply large numbers

```
x = a * b * c * d
log(x) = log(a * b * c * d)
log(x) = log(a) + log(b) + log(c) + log(d)
x = antilog(log(a) + log(b) + log(c) + log(d))
```
So the idea is simple,

Traverse the array and find the sum of log of all the elements,

```
log(a[0]) + log(a[1]) +
.. + log(a[n-1])
```
Then again traverse through the array and find the product using this formula.

```
antilog((log(a[0]) + log(a[1]) +
 .. + log(a[n-1])) - log(a[i]))
```
This equals to product of all the elements except a[i], i.e antilog(sum- log(a[i])).

- C++
- Java
- Python
- C#
- PHP
- Javascript

```
// Java program for product array puzzle
// with O(n) time and O(1) space.
public class Array_puzzle_2 {

    // epsilon value to maintain precision
    static final double EPS = 1e-9;

    static void productPuzzle(int a[], int n)
    {
        // to hold sum of all values
        double sum = 0;
        for (int i = 0; i < n; i++)
            sum += Math.log10(a[i]);

        // output product for each index
        // anti log to find original product value
        for (int i = 0; i < n; i++)
            System.out.print(
                (int)(EPS
                      + Math.pow(
                            10.00, sum
                                       - Math.log10(a[i])))
                + " ");
    }

    // Driver code
    public static void main(String args[])
    {
        int a[] = { 10, 3, 5, 6, 2 };
        int n = a.length;
        System.out.println("The product array is: ");
        productPuzzle(a, n);
    }
}
// This code is contributed by Sumit Ghosh
```

Output:

```
The product array is:
180 600 360 300 900
```

Complexity Analysis:

- **Time Complexity:** O(n).
  Only two traversals of the array is required.
- **Space Complexity:** O(1).
  No extra space is required.
  This approach is contributed by Abhishek Rajput.

Alternate Approach: Here's another approach to solve the above problem by the use of pow() function, does not use division and works in O(n) time.

Traverse the array and find the product of all the elements in the array. Store the product in a variable.

Then again traverse the array and find the product of all the elements except that number by using the formula (product * pow(a[i], -1))

- C++
- Java
- Python3
- C#
- Javascript

```
// Java program for product array puzzle
// with O(n) time and O(1) space.
public class ArrayPuzzle {

    static void solve(int arr[], int n)
    {
        // Initialize a variable to store the
        // total product of the array elements
        int prod = 1;
        for (int i = 0; i < n; i++)
            prod *= arr[i];

        // we know x/y mathematically is same
        // as x*(y to power -1)
        for (int i = 0; i < n; i++)
            System.out.print(
                (int)prod * Math.pow(arr[i], -1) + " ");
    }

    // Driver code
    public static void main(String args[])
    {
        int arr[] = { 10, 3, 5, 6, 2 };
        int n = arr.length;
        solve(arr, n);
    }
}
// This code is contributed by Sitesh Roy
```

Output:

```
180 600 360 300 900
```

Complexity Analysis:

- **Time complexity:** O(n).
  Only two traversals of the array is required.
- **Space complexity:** O(1).
  No extra space is required.

From <https://www.geeksforgeeks.org/product-array-puzzle-set-2-o1-space/>

From <https://www.geeksforgeeks.org/a-product-array-puzzle/>

A Product Array Puzzle | Set 3

- Difficulty Level : Medium
- Last Updated : 26 Sep, 2021

Given an array **arr[]** consisting of **N** integers, the task is to construct a Product array of the same size without using division (**'/'**) operator such that each array element becomes equal to the product of all the elements of **arr[]** except **arr[i]**.

Examples:

Input: *arr[] = {10, 3, 5, 6, 2}*

Output: *180 600 360 300 900*

Explanation:

*3 * 5 * 6 * 2 is the product of all array elements except 10 is 180*

*10 * 5 * 6 * 2 is the product of all array elements except 3 is 600.*

*10 * 3 * 6 * 2 is the product of all array elements except 5 is 360.*

*10 * 3 * 5 * 2 is the product of all array elements except 6 is 300.*

*10 * 3 * 6 * 5 is the product of all array elements except 2 is 9.*

Input: *arr[] = {1, 2, 1, 3, 4}*

Output: *24 12 24 8 6*

Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.

Approach: The idea is to use log() and exp() functions instead of **log10()** and pow(). Below are some observations regarding the same:

- Suppose **M** is the multiplication of all the array elements then the element of output array at **ith** position will be equal **M/arr[i].**
- The divisions of two numbers can be performed by using the property of logarithm and exp functions.
-
-
- The logarithmic function is not defined for numbers less than zero so to maintain the such cases separately.
  Follow the steps below to solve the problem:

- Initialize two variables, say **product = 1** and **Z = 1**, to store the product of array and count of zero elements.
- Traverse the array and multiply the product by **arr[i]** if **arr[i]** is not equal to **0**. Otherwise, increment count of **Z** by **one**.
- Traverse the array **arr[]** and perform the following:
- If **Z** is **1** and **arr[i]** is not zero then update **arr[i]** as **arr[i] = 0** and continue.
- Otherwise, if **Z** is **1** and **arr[i]** is **0** then update **arr[i]** as **product** and continue.
- Otherwise, if **Z** is greater than **1** then assign **arr[i]** as **0** and continue.
- Now find the value of **abs(product)/abs(arr[i])** using the formula discussed above and store it in a variable say **curr**.
- If the value of **arr[i]** and **product** is negative or if **arr[i]** and **product** is positive then assign **arr[i]** as **curr**.
- Otherwise, assign **arr[i]** as **-1*curr**.
- After completing the above steps, print the array **arr[]**.
  Below is the implementation of the above approach:

- C++
- Java
- Python3
- C#
- Javascript

```java
// Java program to implement
// the above approach
import java.util.*;

class GFG{

// Function to form product array
// with O(n) time and O(1) space
static void productExceptSelf(int arr[],
                              int N)
{
    // Stores the product of array
    int product = 1;

    // Stores the count of zeros
    int z = 0;

    // Traverse the array
    for (int i = 0; i < N; i++) {

        // If arr[i] is not zero
        if (arr[i] != 0)
            product *= arr[i];

        // If arr[i] is zero then
        // increment count of z by 1
        if (arr[i] == 0)
            z += 1;
    }

    // Stores the absolute value
    // of the product
    int a = Math.abs(product);
    for (int i = 0; i < N; i++) {

        // If Z is equal to 1
        if (z == 1) {

            // If arr[i] is not zero
            if (arr[i] != 0)
                arr[i] = 0;

            // Else
            else
                arr[i] = product;
            continue;
        }

        // If count of 0s at least 2
        else if (z > 1) {

            // Assign arr[i] = 0
            arr[i] = 0;
            continue;
        }

        // Store absolute value of arr[i]
        int b = Math.abs(arr[i]);

        // Find the value of a/b
        int curr = (int)Math.round(Math.exp(Math.log(a) -
Math.log(b)));

        // If arr[i] and product both
        // are less than zero
        if (arr[i] < 0 && product < 0)
            arr[i] = curr;

        // If arr[i] and product both
        // are greater than zero
        else if (arr[i] > 0 && product > 0)
            arr[i] = curr;

        // Else
        else
            arr[i] = -1 * curr;
    }

    // Traverse the array arr[]
    for (int i = 0; i < N; i++) {
        System.out.print(arr[i] + " ");
    }
}

// Driver Code
public static void main(String args[])
{
    int arr[] = { 10, 3, 5, 6, 2 };
    int N = arr.length;

    // Function Call
    productExceptSelf(arr, N);
}
}

// This code is contributed by splevel62.
```

Output:

```
180 600 360 300 900
```

Time Complexity: *O(N)*

Auxiliary Space: *O(1)*

Alternate Approaches: Please refer to the previous posts of this article for alternate approaches:

From <https://www.geeksforgeeks.org/a-product-array-puzzle-set-3/?ref=rp>