

Reverse an Array in groups of given size - GeeksforGeeks

Source: <https://www.geeksforgeeks.org/reverse-an-array-in-groups-of-given-size/>

Courses Tutorials Practice Jobs DSA Tutorial Interview Questions Quizzes Must Do Advanced DSA System Design Aptitude Puzzles Interview Corner DSA Python Technical Scripter 2026 Explore DSA Fundamentals Logic Building Problems Analysis of Algorithms Data Structures Array Data Structure String in Data Structure Hashing in Data Structure Linked List Data Structure Stack Data Structure Queue Data Structure Tree Data Structure Graph Data Structure Trie Data Structure Algorithms Searching Algorithms Sorting Algorithms Introduction to Recursion Greedy Algorithms Tutorial Graph Algorithms Dynamic Programming or DP Bitwise Algorithms Advanced Segment Tree Binary Indexed Tree or Fenwick Tree Square Root (Sqrt) Decomposition Algorithm Binary Lifting Geometry Interview Preparation Interview Corner GfG160 Practice Problem GeeksforGeeks Practice - Leading Online Coding Platform Problem of The Day - Develop the Habit of Coding DSA Course 90% Refund Reverse an Array in groups of given size Last Updated : 22 Jul, 2025 Given an array arr[] and an integer k , find the array after reversing every subarray of consecutive k elements in place. If the last subarray has fewer than k elements, reverse it as it is. Modify the array in place, do not return anything. Examples: Input: arr[] = [1, 2, 3, 4, 5, 6, 7, 8], k = 3 Output: [3, 2, 1, 6, 5, 4, 8, 7] Explanation: Elements is reversed: [1, 2, 3] → [3, 2, 1], [4, 5, 6] → [6, 5, 4], and the last group [7, 8](size < 3) is reversed as [8, 7]. Input: arr[] = [1, 2, 3, 4, 5], k = 3 Output: [3, 2, 1, 5, 4] Explanation: First group consists of elements 1, 2, 3. Second group consists of 4, 5. Input: arr[] = [5, 6, 8, 9], k = 5 Output: [9, 8, 6, 5] Explanation: Since k is greater than array size, the entire array is reversed. Try it on GfG Practice [Approach] Fixed-Size Group Reversal Idea is to consider every sub-array of size k starting from the beginning of the array and reverse it. We need to handle some special cases. => If k is not a multiple of n where n is the size of the array, for the last group we will have less than k elements left, we need to reverse all remaining elements. => If k = 1, the array should remain unchanged. If k >= n, we reverse all elements present in the array. To reverse a subarray, maintain two pointers: left and right. Now, swap the elements at left and right pointers and increment left by 1 and decrement right by 1. Repeat till left and right pointers don't cross each other. Working: C++ #include <iostream> #include <vector> using namespace std ; void reverseInGroups (vector < int >& arr , int k) { // Get the size of the array int n = arr . size () ; for (int i = 0 ; i < n ; i += k) { int left = i ; // to handle case when k is not multiple of n int right = min (i + k - 1 , n - 1) ; // reverse the sub-array [left, right] while (left < right) { swap (arr [left ++], arr [right --]); } } } int main () { vector < int > arr = { 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 } ; int k = 3 ; reverseInGroups (arr , k) ; for (int num : arr) cout << num << " " ; return 0 ; } C #include <stdio.h> void reverseInGroups (int arr [] , int n , int k) { for (int i = 0 ; i < n ; i += k) { int left = i ; int right ; // to handle case when k is not multiple // of n if (i + k - 1 < n - 1) right = i + k - 1 ; else right = n - 1 ; // reverse the sub-array [left, right] while (left < right) { // swap int temp = arr [left]; arr [left] = arr [right]; arr [right] = temp ; left ++ ; right -- ; } } } int main () { int arr [] = { 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 } ; int k = 3 ; int n = sizeof (arr) / sizeof (arr [0]); reverseInGroups (arr , n , k) ; for (int i = 0 ; i < n ; i ++) printf ("%d " , arr [i]); return 0 ; } Java class GfG { static void reverseInGroups (int [] arr , int k){ int n = arr . length ; for (int i = 0 ; i < n ; i += k) { int left = i ; int right = Math . min (i + k - 1 , n - 1) ; // reverse the sub-array while (left < right) { int temp = arr [left] ; arr [left] = arr [right] ; arr [right] = temp ; left ++ ; right -- ; } } } public static void main (String [] args) { int [] arr = { 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 } ; int k = 3 ; reverseInGroups (arr , k) ; for (int num : arr) { System . out . print (num + " "); } } } Python def reverseInGroups (arr , k): i = 0 # get the size of the array n = len (arr) while i < n : left = i # To handle case when k is not # multiple of n right = min (i + k - 1 , n - 1) # reverse the sub-array [left, right] while left < right : arr [left], arr [right] = arr [right], arr [left] left += 1 right -= 1 i += k if __name__ == "__main__" : arr = [1 , 2 , 3 , 4 , 5 , 6 , 7 , 8] k = 3 reverseInGroups (arr , k) print (" " . join (map (str , arr))) C# using System ; class GfG { public static void reverseInGroups (int [] arr , int k){ int n = arr . Length ; for (int i = 0 ; i < n ; i += k) { int left = i ; // to handle case when k is // not multiple of n int right = Math . Min (i + k - 1 , n - 1) ; int temp ; // reverse the sub-array [left, right] while (left < right) { temp = arr [left] ; arr [left] = arr [right] ; arr [right] = temp ; left += 1 ; right -= 1 ; } } public static void Main (string [] args){ int [] arr = new int [] { 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 } ; int k = 3 ; int n = arr . Length ; reverseInGroups (arr , k) ; for (int i = 0 ; i < n ; i ++) { Console .

Write (arr [i] + " "); } } JavaScript function reverseInGroups (arr , k) { let n = arr . length ; for (let i = 0 ; i < n ; i += k) { let left = i ; // to handle case when k is not // multiple of n let right = Math . min (i + k - 1 , n - 1); // reverse the sub-array [left, right] while (left < right) { // Swap elements [arr [left], arr [right]] = [arr [right], arr [left]]; left += 1 ; right -= 1 ; } } return arr ; } // Driver Code let arr = [1 , 2 , 3 , 4 , 5 , 6 , 7 , 8]; let k = 3 ; let arr1 = reverseInGroups (arr , k); console . log (arr1 . join (" ")); Output 3 2 1 6 5 4 8 7 Time Complexity: O(n), we go through the entire array just once, reversing elements in groups of size k. Since we don't revisit any element, the total work done grows linearly with the size of the array. So, if the array has n elements, it takes roughly n steps. Auxiliary Space: O(1), the reversal is done directly within the original array using just a few extra variables. Comment Article Tags: Article Tags: DSA Arrays