Merge Sorted Arrays - GeeksforGeeks

Source: https://www.geeksforgeeks.org/merge-k-sorted-arrays/

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 Merge Sorted Arrays Last Updated : 18 Oct, 2025 Given a 2D matrix, mat[][] consisting of sorted arrays, where each row is sorted in non-decreasing order, find a single sorted array that contains all the elements from the matrix. Examples: Input: mat[][] = [[1, 3, 5, 7], [2, 4, 6, 8], [0, 9, 10, 11]] Output: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11] Explanation: Merging all elements from the 3 sorted arrays and sorting them results in: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]. Input: mat[][] = [[1, 2, 3, 4], [2, 2, 3, 4], [5, 5, 6, 6], [7, 8, 9, 9]] Output: [1, 2, 2, 2, 3, 3, 4, 4, 5, 5, 6, 6, 7, 8, 9, 9] Explanation: Merging all elements from the 4 sorted arrays and sorting them results in:[1, 2, 2, 2, 3, 3, 4, 4, 5, 5, 6, 6, 7, 8, 9, 9] . Try it on GfG Practice Table of Content [Naive Approach] Concatenate all and Sort - O(n*log(n)) Time and O(n) Space [Expected Approach 1] Using Merge Sort - Works Better for Equal Sized Arrays [Expected Approach 2] Using Min-Heap - Works better for Different Sized Arrays [Naive Approach] Concatenate all and Sort The idea is to merges sorted arrays by first flattening all the arrays into a single one-dimensional vector. Then sorts this combined vector using the standard sorting algorithm (sort() from the STL). This ensures the final output is a fully sorted array containing all elements from the input arrays. C++ //Driver Code Starts #include <iostream> #include <algorithm> #include <vector> using namespace std ; //Driver Code Ends vector < int > mergeArrays (vector < vector < int >>& mat) { vector < int > res ; // Append all arrays into res for (const auto & vec : mat) { for (int val : vec) res . push_back (val); } // Sort the res sort (res . begin (), res . end ()); return res ; } //Driver Code Starts int main () { vector < vector < int >> mat = {{ 1 , 3 , 5 , 7 }, { 2 , 4 , 6 , 8 }, { 0 , 9 , 10 , 11 }}; vector < int > res = mergeArrays (mat); for (int val : res) { cout << val << " " ; } return 0 ; } //Driver Code Ends Java //Driver Code Starts import java.util.ArrayList ; import java.util.Collections ; class GfG { //Driver Code Ends static ArrayList < Integer > mergeArrays (int [][] mat) { ArrayList < Integer > res = new ArrayList <> (); // Append all arrays into res for (int i = 0 ; i < mat . length ; i ++) { for (int j = 0 ; j < mat [i] . length ; j ++) { res . add (mat [i][j]); } } // Sort the res Collections . sort (res); return res ; } //Driver Code Starts public static void main (String [] args) { int [][] mat = { { 1 , 3 , 5 , 7 }, { 2 , 4 , 6 , 8 }, { 0 , 9 , 10 , 11 } }; ArrayList < Integer > res = mergeArrays (mat); for (int val : res) { System . out . print (val + " "); } } } //Driver Code Ends Python def mergeArrays (mat): res = [] # Append all arrays into res for vec in mat : for val in vec : res . append (val) # Sort the res res . sort () return res if __name__ == "__main__" : #Driver Code Starts mat = [[1 , 3 , 5 , 7], [2 , 4 , 6 , 8], [0 , 9 , 10 , 11]] res = mergeArrays (mat) print (* res) #Driver Code Ends C# //Driver Code Starts using System ; using System.Collections.Generic ; class GFG { //Driver Code Ends static List < int > mergeArrays (int [,] mat) { List < int > res = new List < int > (); int rows = mat . GetLength (0); int cols = mat . GetLength (1); // Append all elements into res for (int i = 0 ; i < rows ; i ++) { for (int j = 0 ; j < cols ; j ++) { res . Add (mat [i , j]); } } // Sort the res res . Sort (); return res ; } //Driver Code Starts static void Main () { int [,] mat = new int [,] { { 1 , 3 , 5 , 7 }, { 2 , 4 , 6 , 8 }, { 0 , 9 , 10 , 11 } }; List < int > res = mergeArrays (mat); foreach (int val in res) { Console . Write (val + " "); } } } //Driver Code Ends JavaScript function mergeArrays (mat) { let res = []; // Append all arrays into res for (let vec of mat) { for (let val of vec) { res . push (val); } } // Sort the res res . sort ((a , b) => a - b); return res ; } // Driver code //Driver Code Starts let mat = [[1 , 3 , 5 , 7], [2 , 4 , 6 , 8], [0 , 9 , 10 , 11]]; let res = mergeArrays (mat); console . log (res . join (" ")); //Driver Code Ends Output 0 1 2 3 4 5 6 7 8 9 10 11 Time Complexity: O(n log(n)), where n is total number of elements Auxiliary Space: O(1) [Expected Approach 1] Using Merge Sort - Works Better for Equal Sized Arrays The idea is to use the Divide and Conquer approach, similar to Merge Sort. We keep

dividing the given sorted arrays into smaller groups until we are left with single array. Then, we start merging these single arrays upward in sorted order, gradually combining them until one fully sorted array remains. C++ //Driver Code Starts #include <iostream> #include <vector> using namespace std ; //Driver Code Ends // Function to merge two sorted arrays vector < int > concat (vector < int > & a , vector < int > & b) { int i = 0 , j = 0 ; int n1 = a . size (), n2 = b . size (); vector < int > c ; c . reserve (n1 + n2); // Merge both arrays in sorted order while (i < n1 && j < n2) { if (a [i] < b [j]) c . push_back (a [i ++]); else c . push_back (b [j ++]); } // Add remaining elements of first array while (i < n1) c . push_back (a [i ++]); // Add remaining elements of second array while (j < n2) c . push_back (b [j ++]); return c ; } // Recursively merge K sorted arrays using divide and conquer vector < int > merge (vector < vector < int >> & mat , int lo , int hi) { // Base case if (lo == hi) return mat [lo]; // Divide arrays into two halves int mid = (lo + hi) / 2 ; // Merge left half vector < int > left = merge (mat , lo , mid); // Merge right half vector < int > right = merge (mat , mid + 1 , hi); // Combine both halves return concat (left , right); } // merge all K sorted arrays vector < int > mergeArrays (vector < vector < int >> & mat) { int k = mat . size (); if (k == 0) return {}; return merge (mat , 0 , k - 1); } //Driver Code Starts int main () { vector < vector < int >> mat = { { 1 , 3 , 5 , 7 }, { 2 , 4 , 6 , 8 }, { 0 , 9 , 10 , 11 } }; int k = mat . size (); vector < int > res = mergeArrays (mat); for (int val : res) cout << val << " " ; cout << endl ; return 0 ; } //Driver Code Ends Java //Driver Code Starts import java.util.ArrayList ; class GFG { //Driver Code Ends // Function to merge two sorted arrays static ArrayList < Integer > concat (ArrayList < Integer > a , ArrayList < Integer > b) { int i = 0 , j = 0 ; int n1 = a . size (), n2 = b . size (); ArrayList < Integer > c = new ArrayList <> (); // Merge both arrays in sorted order while (i < n1 && j < n2) { if (a . get (i) < b . get (j)) c . add (a . get (i ++)); else c . add (b . get (j ++)); } // Add remaining elements of first array while (i < n1) c . add (a . get (i ++)); // Add remaining elements of second array while (j < n2) c . add (b . get (j ++)); return c ; } // Recursively merge K sorted arrays using divide and conquer static ArrayList < Integer > merge (ArrayList < ArrayList < Integer >> mat , int lo , int hi) { // Base case if (lo == hi) return mat . get (lo); // Divide arrays into two halves int mid = (lo + hi) / 2 ; // Merge left half ArrayList < Integer > left = merge (mat , lo , mid); // Merge right half ArrayList < Integer > right = merge (mat , mid + 1 , hi); // Combine both halves return concat (left , right); } // merge all K sorted arrays static ArrayList < Integer > mergeArrays (int [][] mat) { int k = mat . length ; if (k == 0) return new ArrayList <> (); // Convert 2D array to ArrayList<ArrayList<Integer>> ArrayList < ArrayList < Integer >> list = new ArrayList <> (); for (int i = 0 ; i < k ; i ++) { ArrayList < Integer > temp = new ArrayList <> (); for (int j = 0 ; j < mat [i] . length ; j ++) { temp . add (mat [i][j]); } list . add (temp); } return merge (list , 0 , k - 1); } //Driver Code Starts public static void main (String [] args) { int [][] mat = { { 1 , 3 , 5 , 7 }, { 2 , 4 , 6 , 8 }, { 0 , 9 , 10 , 11 } }; ArrayList < Integer > res = mergeArrays (mat); for (int val : res) System . out . print (val + " "); System . out . println (); } } //Driver Code Ends Python # Function to merge two sorted arrays def concat (a , b): i = 0 j = 0 n1 = len (a) n2 = len (b) c = [] # Merge both arrays in sorted order while i < n1 and j < n2 : if a [i] < b [j]: c . append (a [i]) i += 1 else : c . append (b [j]) j += 1 # Add remaining elements of first array while i < n1 : c . append (a [i]) i += 1 # Add remaining elements of second array while j < n2 : c . append (b [j]) j += 1 return c # Recursively merge K sorted arrays using divide and conquer def merge (mat , lo , hi): # Base case if lo == hi : return mat [lo] # Divide arrays into two halves mid = (lo + hi) // 2 # Merge left half left = merge (mat , lo , mid) # Merge right half right = merge (mat , mid + 1 , hi) # Combine both halves return concat (left , right) # merge all K sorted arrays def mergeArrays (mat): k = len (mat) if k == 0 : return [] return merge (mat , 0 , k - 1) if __name__ == "__main__" : #Driver Code Starts mat = [[1 , 3 , 5 , 7], [2 , 4 , 6 , 8], [0 , 9 , 10 , 11]] k = len (mat) res = mergeArrays (mat) for val in res : print (val , end = " ") print () #Driver Code Ends C# //Driver Code Starts using System ; using System.Collections.Generic ; class GFG { //Driver Code Ends // Function to merge two sorted arrays static List < int > concat (List < int > a , List < int > b) { int i = 0 , j = 0 ; int n1 = a . Count , n2 = b . Count ; List < int > c = new List < int > (n1 + n2); // Merge both arrays in sorted order while (i < n1 && j < n2) { if (a [i] < b [j]) c . Add (a [i ++]); else c . Add (b [j ++]); } // Add remaining elements of first array while (i < n1) c . Add (a [i ++]); // Add remaining elements of second array while (j < n2) c . Add (b [j ++]); return c ; } // Recursively merge K sorted arrays using divide and conquer static List < int > merge (List < List < int >> mat , int lo , int hi) { // Base case if (lo == hi) return mat [lo]; // Divide arrays into two halves int mid = (lo + hi) / 2 ; // Merge left half List < int > left = merge (mat , lo , mid); // Merge right half List < int > right = merge (mat , mid + 1 , hi); // Combine both halves return concat (left , right); } // merge all K sorted arrays static List < int > mergeArrays (int [,] mat) { int k = mat . GetLength (0); if (k == 0) return new List < int > (); int nCols = mat . GetLength (1); // Convert 2D array to List<List<int>> List < List < int >> list = new List < List < int >> (); for (int i = 0 ; i < k ; i ++) { List < int > temp = new List < int > (); for (int j = 0 ;

j < nCols ; j ++) { temp . Add (mat [i , j]); } list . Add (temp); } return merge (list , 0 , k - 1); } //Driver Code Starts static void Main (string [] args) { int [,] mat = new int [,] { { 1 , 3 , 5 , 7 }, { 2 , 4 , 6 , 8 }, { 0 , 9 , 10 , 11 } }; List < int > res = mergeArrays (mat); foreach (int val in res) Console . Write (val + " "); Console . WriteLine (); } } //Driver Code Ends JavaScript // Function to merge two sorted arrays function concat (a , b) { let i = 0 , j = 0 ; let n1 = a . length , n2 = b . length ; let c = []; // Merge both arrays in sorted order while (i < n1 && j < n2) { if (a [i] < b [j]) c . push (a [i ++]); else c . push (b [j ++]); } // Add remaining elements of first array while (i < n1) c . push (a [i ++]); // Add remaining elements of second array while (j < n2) c . push (b [j ++]); return c ; } // Recursively merge K sorted arrays using divide and conquer function merge (mat , lo , hi) { // Base case if (lo === hi) return mat [lo]; // Divide arrays into two halves let mid = Math . floor ((lo + hi) / 2); // Merge left half let left = merge (mat , lo , mid); // Merge right half let right = merge (mat , mid + 1 , hi); // Combine both halves return concat (left , right); } // merge all K sorted arrays function mergeArrays (mat) { let k = mat . length ; if (k === 0) return []; return merge (mat , 0 , k - 1); } // Driver code //Driver Code Starts let mat = [[1 , 3 , 5 , 7], [2 , 4 , 6 , 8], [0 , 9 , 10 , 11]]; let k = mat . length ; let res = mergeArrays (mat); console . log (res . join (' ')); //Driver Code Ends Output 0 1 2 3 4 5 6 7 8 9 10 11 Time Complexity: O(n*log■k), k is number of rows and n is total number of elements. Auxiliary Space: O(n) [Expected Approach 2] Using Min-Heap - Works better for Different Sized Arrays The idea is to merge sorted arrays using a Min Heap. We start by inserting the first element of each array into the heap. The smallest element is always at the top, so we remove it and add it to the output array. Then we insert the next element from the same array into the heap. We repeat this process until the heap is empty. This ensures that we always pick the smallest available element, producing a fully sorted merged array efficiently. C++ //Driver Code Starts #include <iostream> #include <queue> #include <vector> using namespace std ; //Driver Code Ends vector < int > mergeArrays (vector < vector < int >> & mat){ int k = mat . size (); vector < int > output ; // Min-heap: {value, {array index, element index}} priority_queue < pair < int , pair < int , int >> , vector < pair < int , pair < int , int >>> , greater < pair < int , pair < int , int >>>> minHeap ; // Push first element of each array for (int i = 0 ; i < k ; ++ i){ if (! mat [i]. empty ()){ minHeap . push ({ mat [i][0], { i , 0 }}); } } // Merge all elements while (! minHeap . empty ()){ auto top = minHeap . top (); minHeap . pop (); int val = top . first ; int i = top . second . first ; int j = top . second . second ; output . push_back (val); // Push next element from same array if (j + 1 < mat [i]. size ()){ minHeap . push ({ mat [i][j + 1], { i , j + 1 }}); } } return output ; } //Driver Code Starts int main (){ vector < vector < int >> mat = {{ 1 , 3 , 5 , 7 }, { 2 , 4 , 6 , 8 }, { 0 , 9 , 10 , 11 }}; vector < int > result = mergeArrays (mat); for (int x : result){ cout << x << " " ; } cout << endl ; return 0 ; } //Driver Code Ends Java //Driver Code Starts import java.util.ArrayList ; import java.util.List ; import java.util.PriorityQueue ; import java.util.Comparator ; class GFG { //Driver Code Ends static ArrayList < Integer > mergeArrays (int [][] mat){ int k = mat . length ; ArrayList < Integer > output = new ArrayList <> (); // Min-heap: {value, {array index, element index}} PriorityQueue < int []> minHeap = new PriorityQueue <> (Comparator . comparingInt (a -> a [0])); // Push first element of each array for (int i = 0 ; i < k ; ++ i){ if (mat [i] . length > 0){ minHeap . add (new int [] { mat [i][0] , i , 0 }); } } // Merge all elements while (! minHeap . isEmpty ()){ int [] top = minHeap . poll (); int val = top [0] ; int i = top [1] ; int j = top [2] ; output . add (val); // Push next element from same array if (j + 1 < mat [i] . length){ minHeap . add (new int [] { mat [i][j + 1] , i , j + 1 }); } } return output ; } //Driver Code Starts public static void main (String [] args){ int [][] mat = { { 1 , 3 , 5 , 7 }, { 2 , 4 , 6 , 8 }, { 0 , 9 , 10 , 11 } }; ArrayList < Integer > result = mergeArrays (mat); for (int x : result){ System . out . print (x + " "); } System . out . println (); } } //Driver Code Ends Python #Driver Code Starts import heapq #Driver Code Ends def mergeArrays (mat): k = len (mat) output = [] # Min-heap: (value, (array index, element index)) minHeap = [] # Push first element of each array for i in range (k): if len (mat [i]) > 0 : heapq . heappush (minHeap , (mat [i][0], i , 0)) # Merge all elements while minHeap : val , i , j = heapq . heappop (minHeap) output . append (val) # Push next element from same array if j + 1 < len (mat [i]): heapq . heappush (minHeap , (mat [i][j + 1], i , j + 1)) return output #Driver Code Starts if __name__ == "__main__" : mat = [[1 , 3 , 5 , 7], [2 , 4 , 6 , 8], [0 , 9 , 10 , 11]] result = mergeArrays (mat) print (" " . join (map (str , result))) #Driver Code Ends C# //Driver Code Starts using System ; using System.Collections.Generic ; // Custom Min-Heap Priority Queue class MinHeap { private List < (int val , int arrIdx , int elemIdx) > heap ; public MinHeap () { heap = new List < (int , int , int) > (); } private void Swap (int i , int j) { var temp = heap [i]; heap [i] = heap [j]; heap [j] = temp ; } private void HeapifyUp (int index) { while (index > 0) { int parent = (index - 1) / 2 ; if (heap [index]. val < heap [parent]. val) { Swap (index , parent); index = parent ; } else break ; } } private void HeapifyDown (int index) { int last = heap . Count - 1 ; while (true) { int left = 2 * index + 1 ; int right = 2 * index + 2 ; int smallest = index ; if (left <= last && heap [left]. val

< heap [smallest]. val) smallest = left ; if (right <= last && heap [right]. val < heap [smallest]. val) smallest = right ; if (smallest != index) { Swap (index , smallest); index = smallest ; } else break ; } } public void Add ((int val , int arrIdx , int elemIdx) item) { heap . Add (item); HeapifyUp (heap . Count - 1); } public (int val , int arrIdx , int elemIdx) Pop () { var top = heap [0]; heap [0] = heap [heap . Count - 1]; heap . RemoveAt (heap . Count - 1); HeapifyDown (0); return top ; } public int Count () { return heap . Count ; } } class GFG { //Driver Code Ends static List < int > mergeArrays (int [,] mat) { int k = mat . GetLength (0); int nCols = mat . GetLength (1); List < int > output = new List < int > (); // Min-heap: {value, array index, element index} MinHeap minHeap = new MinHeap (); // Push first element of each array for (int i = 0 ; i < k ; i ++) { if (nCols > 0) minHeap . Add ((mat [i , 0], i , 0)); } // Merge all elements while (minHeap . Count () > 0) { var top = minHeap . Pop (); int val = top . val ; int i = top . arrIdx ; int j = top . elemIdx ; output . Add (val); // Push next element from same array if (j + 1 < nCols) minHeap . Add ((mat [i , j + 1], i , j + 1)); } return output ; } //Driver Code Starts static void Main () { int [,] mat = new int [,] { { 1 , 3 , 5 , 7 }, { 2 , 4 , 6 , 8 }, { 0 , 9 , 10 , 11 } }; List < int > result = mergeArrays (mat); foreach (int x in result) Console . Write (x + " "); Console . WriteLine (); } } //Driver Code Ends JavaScript //Driver Code Starts // Custom Min-Heap Priority Queue class MinHeap { constructor () { this . heap = []; } swap (i , j) { [this . heap [i], this . heap [j]] = [this . heap [j], this . heap [i]]; } heapifyUp (index) { while (index > 0) { let parent = Math . floor ((index - 1) / 2); if (this . heap [index][0] < this . heap [parent][0]) { this . swap (index , parent); index = parent ; } else break ; } } heapifyDown (index) { let last = this . heap . length - 1 ; while (true) { let left = 2 * index + 1 ; let right = 2 * index + 2 ; let smallest = index ; if (left <= last && this . heap [left][0] < this . heap [smallest][0]) smallest = left ; if (right <= last && this . heap [right][0] < this . heap [smallest][0]) smallest = right ; if (smallest !== index) { this . swap (index , smallest); index = smallest ; } else break ; } } add (item) { this . heap . push (item); this . heapifyUp (this . heap . length - 1); } pop () { const top = this . heap [0]; this . heap [0] = this . heap [this . heap . length - 1]; this . heap . pop (); this . heapifyDown (0); return top ; } size () { return this . heap . length ; } } //Driver Code Ends function mergeArrays (mat) { let k = mat . length ; let output = []; // Min-heap: [value, array index, element index] const minHeap = new MinHeap (); // Push first element of each array for (let i = 0 ; i < k ; i ++) { if (mat [i]. length > 0) { minHeap . add ([mat [i][0], i , 0]); } } // Merge all elements while (minHeap . size () > 0) { let [val , i , j] = minHeap . pop (); output . push (val); // Push next element from same array if (j + 1 < mat [i]. length) { minHeap . add ([mat [i][j + 1], i , j + 1]); } } return output ; } //Driver Code Starts // Driver code let mat = [[1 , 3 , 5 , 7], [2 , 4 , 6 , 8], [0 , 9 , 10 , 11]]; let result = mergeArrays (mat); console . log (result . join (' ')); //Driver Code Ends Output 0 1 2 3 4 5 6 7 8 9 10 11 Time Complexity: O(n*log(k)), k is number of rows and n is total number of elements. Auxiliary Space: O(k) Comment Article Tags: Article Tags: Sorting Heap DSA Arrays Microsoft Amazon Flipkart VMWare Citrix Merge Sort + 6 More