Container with Most Water - GeeksforGeeks

Source: https://www.geeksforgeeks.org/container-with-most-water/

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 Container with Most Water Last Updated : 12 Aug, 2025 Given an array arr[] of non-negative integers, where each element arr[i] represents the height of the vertical lines, find the maximum amount of water that can be contained between any two lines, together with the x-axis. Examples : Input: arr[] = [1, 5, 4, 3] Output: 6 Explanation: 5 and 3 are 2 distance apart. So the size of the base = 2. Height of container = min(5, 3) = 3. So total area = 3 * 2 = 6. Input: arr[] = [3, 1, 2, 4, 5] Output: 12 Explanation: 5 and 3 are 4 distance apart. So the size of the base = 4. Height of container = min(5, 3) = 3. So total area = 4 * 3 = 12. Input: arr[] = [2, 1, 8, 6, 4, 6, 5, 5] Output: 25 Explanation: 8 and 5 are 5 distance apart. So the size of the base = 5. Height of container = min(8, 5) = 5. So, total area = 5 * 5 = 25. Try it on GfG Practice Table of Content [Naive Approach] Checking all possible boundaries - O(n^2) Time and O(1) Space [Expected Approach] Using Two Pointers - O(n) Time and O(1) Space [Naive Approach] Checking all possible boundaries - O(n^2) Time and O(1) Space The idea is to evaluate all possible pairs of lines in the array. For each line arr[i], treat it as the left boundary, and then iterate through all lines to its right (arr[j], where j > i) to consider them as the right boundary. For every such pair, compute the water that can be trapped between them using the formula: min(arr[i], arr[j]) × (j - i) This represents the height (limited by the shorter line) multiplied by the width between the two lines. Finally, keep track of the maximum water obtained among all such pairs. C++ #include <vector> #include <iostream> using namespace std ; int maxWater (vector < int > & arr) { int n = arr . size (); int res = 0 ; for (int i = 0 ; i < n ; i ++) { for (int j = i + 1 ; j < n ; j ++) { // calculate the amount of water int amount = min (arr [i], arr [j]) * (j - i); // keep track of maximum amount of water res = max (amount , res); } } return res ; } int main () { vector < int > arr = { 2 , 1 , 8 , 6 , 4 , 6 , 5 , 5 }; cout << maxWater (arr); } C #include <stdio.h> int maxWater (int arr [], int n) { int res = 0 ; for (int i = 0 ; i < n ; i ++) { for (int j = i + 1 ; j < n ; j ++) { // calculate the amount of water int amount = (arr [i] < arr [j] ? arr [i] : arr [j]) * (j - i); // Keep track of maximum amount of water res = (amount > res ? amount : res); } } return res ; } int main () { int arr [] = { 2 , 1 , 8 , 6 , 4 , 6 , 5 , 5 }; int n = sizeof (arr) / sizeof (arr [0]); printf ("%d" , maxWater (arr , n)); return 0 ; } Java import java.util.* ; class GfG { static int maxWater (int [] arr) { int n = arr . length ; int res = 0 ; for (int i = 0 ; i < n ; i ++) { for (int j = i + 1 ; j < n ; j ++) { // calculate the amount of water int amount = Math . min (arr [i] , arr [j]) * (j - i); // keep track of maximum amount of water res = Math . max (amount , res); } } return res ; } public static void main (String [] args) { int [] arr = { 2 , 1 , 8 , 6 , 4 , 6 , 5 , 5 }; System . out . println (maxWater (arr)); } } Python def maxWater (arr): n = len (arr) res = 0 for i in range (n): for j in range (i + 1 , n): # calculate the amount of water amount = min (arr [i], arr [j]) * (j - i) # keep track of maximum amount of water res = max (amount , res) return res if __name__ == "__main__" : arr = [2 , 1 , 8 , 6 , 4 , 6 , 5 , 5] print (maxWater (arr)) C# using System ; class GfG { static int maxWater (int [] arr) { int n = arr . Length ; int res = 0 ; for (int i = 0 ; i < n ; i ++) { for (int j = i + 1 ; j < n ; j ++) { // calculate the amount of water int amount = Math . Min (arr [i], arr [j]) * (j - i); // keep track of maximum amount of water res = Math . Max (amount , res); } } return res ; } static void Main () { int [] arr = { 2 , 1 , 8 , 6 , 4 , 6 , 5 , 5 }; Console . WriteLine (maxWater (arr)); } } JavaScript function maxWater (arr) { let n = arr . length ; let res = 0 ; for (let i = 0 ; i < n ; i ++) { for (let j = i + 1 ; j < n ; j ++) { // calculate the amount of water let amount = Math . min (arr [i], arr [j]) * (j - i); // keep track of maximum amount of water res = Math . max (amount , res); } } return res ; } // Driver Code let arr = [2 , 1 , 8 , 6 , 4 , 6 , 5 , 5]; console . log (maxWater (arr)); Output 25 [Expected Approach] Using Two Pointers - O(n) Time and O(1) Space The idea is to maintain two pointers: left

pointer at the beginning of the array and right pointer at the end of the array. These pointers act as the container's sides so we can calculate the amount of water stored between them using the formula: min(arr[left], arr[right]) * (right - left) . After calculating the amount of water for the given sides, we can have three cases: arr[left] < arr[right]: This means that we have calculated the water stored for the container of height arr[left], so increment left by 1. arr[left] >= arr[right]: This means that we have calculated the water stored for the container of height arr[right], so decrement right by 1. Repeat the above process till left is less than right keeping track of the maximum water stored as the result. Why are we moving the pointer which is pointing to the shorter line? We are moving the pointer pointing to the shorter line to potentially find a taller line and increase the container's height. Moving the pointer to the taller line would only reduce the width, but the height cannot increase because of the shorter line, thus decreasing the amount of water. C++ #include <vector> #include <iostream> using namespace std ; int maxWater (vector < int > & arr) { int left = 0 , right = arr . size () - 1 ; int res = 0 ; while (left < right) { // find the water stored in the container between // arr[left] and arr[right] int water = min (arr [left], arr [right]) * (right - left); res = max (res , water); if (arr [left] < arr [right]) left += 1 ; else right -= 1 ; } return res ; } int main () { vector < int > arr = { 2 , 1 , 8 , 6 , 4 , 6 , 5 , 5 }; cout << maxWater (arr); } Java class GfG { static int maxWater (int [] arr) { int left = 0 , right = arr . length - 1 ; int res = 0 ; while (left < right) { // find the water stored in the container between // arr[left] and arr[right] int water = Math . min (arr [left] , arr [right]) * (right - left); res = Math . max (res , water); if (arr [left] < arr [right]) left += 1 ; else right -= 1 ; } return res ; } public static void main (String [] args) { int [] arr = { 2 , 1 , 8 , 6 , 4 , 6 , 5 , 5 }; System . out . println (maxWater (arr)); } } Python def maxWater (arr): left = 0 right = len (arr) - 1 res = 0 while left < right : # find the water stored in the container between # arr[left] and arr[right] water = min (arr [left], arr [right]) * (right - left) res = max (res , water) if arr [left] < arr [right]: left += 1 else : right -= 1 return res if __name__ == "__main__" : arr = [2 , 1 , 8 , 6 , 4 , 6 , 5 , 5] print (maxWater (arr)) C# using System ; using System.Collections.Generic ; class GfG { static int maxWater (int [] arr) { int left = 0 , right = arr . Length - 1 ; int res = 0 ; while (left < right) { // find the water stored in the container between // arr[left] and arr[right] int water = Math . Min (arr [left], arr [right]) * (right - left); res = Math . Max (res , water); if (arr [left] < arr [right]) left += 1 ; else right -= 1 ; } return res ; } static void Main () { int [] arr = { 2 , 1 , 8 , 6 , 4 , 6 , 5 , 5 }; Console . WriteLine (maxWater (arr)); } } JavaScript function maxWater (arr) { let left = 0 , right = arr . length - 1 ; let res = 0 ; while (left < right) { // find the water stored in the container between // arr[left] and arr[right] let water = Math . min (arr [left], arr [right]) * (right - left); res = Math . max (res , water); if (arr [left] < arr [right]) left += 1 ; else right -= 1 ; } return res ; } // Driver Code let arr = [2 , 1 , 8 , 6 , 4 , 6 , 5 , 5]; console . log (maxWater (arr)); Output 25 Comment Article Tags: Article Tags: DSA programming-puzzle