Bubble Sort - GeeksforGeeks

Source: https://www.geeksforgeeks.org/bubble-sort/

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 Bubble Sort Last Updated : 8 Dec, 2025 Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in the wrong order. This algorithm is not efficient for large data sets as its average and worst-case time complexity are quite high. Sorts the array using multiple passes. After the first pass, the maximum goes to end (its correct position). Same way, after second pass, the second largest goes to second last position and so on. In every pass, process only those that have already not moved to correct position. After k passes, the largest k must have been moved to the last k positions. In a pass, we consider remaining elements and compare all adjacent and swap if larger element is before a smaller element. If we keep doing this, we get the largest (among the remaining elements) at its correct position. Below is the implementation of the bubble sort. It can be optimized by stopping the algorithm if the inner loop didn't cause any swap. Try it on GfG Practice C++ #include <bits/stdc++.h> using namespace std ; // An optimized version of Bubble Sort void bubbleSort (vector < int >& arr) { int n = arr . size (); bool swapped ; for (int i = 0 ; i < n - 1 ; i ++) { swapped = false ; for (int j = 0 ; j < n - i - 1 ; j ++) { if (arr [j] > arr [j + 1]) { swap (arr [j], arr [j + 1]); swapped = true ; } } // If no two elements were swapped, then break if (! swapped) break ; } } // Function to print a vector void printVector (const vector < int >& arr) { for (int num : arr) cout << " " << num ; } int main () { vector < int > arr = { 64 , 34 , 25 , 12 , 22 , 11 , 90 }; bubbleSort (arr); cout << "Sorted array: \n " ; printVector (arr); return 0 ; } C // Optimized implementation of Bubble sort #include <stdbool.h> #include <stdio.h> void swap (int * xp , int * yp){ int temp = * xp ; * xp = * yp ; * yp = temp ; } // An optimized version of Bubble Sort void bubbleSort (int arr [], int n){ int i , j ; bool swapped ; for (i = 0 ; i < n - 1 ; i ++) { swapped = false ; for (j = 0 ; j < n - i - 1 ; j ++) { if (arr [j] > arr [j + 1]) { swap (& arr [j], & arr [j + 1]); swapped = true ; } } // If no two elements were swapped by inner loop, // then break if (swapped == false) break ; } } // Function to print an array void printArray (int arr [], int size){ int i ; for (i = 0 ; i < size ; i ++) printf ("%d " , arr [i]); } int main (){ int arr [] = { 64 , 34 , 25 , 12 , 22 , 11 , 90 }; int n = sizeof (arr) / sizeof (arr [0]); bubbleSort (arr , n); printf ("Sorted array: \n "); printArray (arr , n); return 0 ; } Java // Optimized java implementation of Bubble sort import java.io.* ; class GFG { // An optimized version of Bubble Sort static void bubbleSort (int arr [] , int n){ int i , j , temp ; boolean swapped ; for (i = 0 ; i < n - 1 ; i ++) { swapped = false ; for (j = 0 ; j < n - i - 1 ; j ++) { if (arr [j] > arr [j + 1]) { // Swap arr[j] and arr[j+1] temp = arr [j] ; arr [j] = arr [j + 1] ; arr [j + 1] = temp ; swapped = true ; } } // If no two elements were // swapped by inner loop, then break if (swapped == false) break ; } } // Function to print an array static void printArray (int arr [] , int size){ int i ; for (i = 0 ; i < size ; i ++) System . out . print (arr [i] + " "); System . out . println (); } // Driver program public static void main (String args []){ int arr [] = { 64 , 34 , 25 , 12 , 22 , 11 , 90 }; int n = arr . length ; bubbleSort (arr , n); System . out . println ("Sorted array: "); printArray (arr , n); } } Python # Optimized Python program for implementation of Bubble Sort def bubbleSort (arr): n = len (arr) # Traverse through all array elements for i in range (n): swapped = False # Last i elements are already in place for j in range (0 , n - i - 1): # Traverse the array from 0 to n-i-1 # Swap if the element found is greater # than the next element if arr [j] > arr [j + 1]: arr [j], arr [j + 1] = arr [j + 1], arr [j] swapped = True if (swapped == False): break # Driver code to test above if __name__ == "__main__" : arr = [64 , 34 , 25 , 12 , 22 , 11 , 90] bubbleSort (arr) print ("Sorted array:") for i in range (len (arr)): print (" %d " % arr [i], end = " ") C# // Optimized C# implementation of Bubble sort using System ; class GFG { // An optimized version of Bubble Sort static void bubbleSort (int [] arr , int n){ int i , j , temp ; bool swapped ; for (i = 0 ;

i < n - 1 ; i ++) { swapped = false ; for (j = 0 ; j < n - i - 1 ; j ++) { if (arr [j] > arr [j + 1]) { // Swap arr[j] and arr[j+1] temp = arr [j]; arr [j] = arr [j + 1]; arr [j + 1] = temp ; swapped = true ; } } // If no two elements were // swapped by inner loop, then break if (swapped == false) break ; } } // Function to print an array static void printArray (int [] arr , int size){ int i ; for (i = 0 ; i < size ; i ++) Console . Write (arr [i] + " "); Console . WriteLine (); } // Driver method public static void Main (){ int [] arr = { 64 , 34 , 25 , 12 , 22 , 11 , 90 }; int n = arr . Length ; bubbleSort (arr , n); Console . WriteLine ("Sorted array:"); printArray (arr , n); } } JavaScript // Optimized javaScript implementation // of Bubble sort function bubbleSort (arr , n){ var i , j , temp ; var swapped ; for (i = 0 ; i < n - 1 ; i ++){ swapped = false ; for (j = 0 ; j < n - i - 1 ; j ++){ if (arr [j] > arr [j + 1]) { // Swap arr[j] and arr[j+1] temp = arr [j]; arr [j] = arr [j + 1]; arr [j + 1] = temp ; swapped = true ; } } // IF no two elements were // swapped by inner loop, then break if (swapped == false) break ; } } // Function to print an array function printArray (arr , size){ var i ; for (i = 0 ; i < size ; i ++) console . log (arr [i] + " "); } // Driver program var arr = [64 , 34 , 25 , 12 , 22 , 11 , 90]; var n = arr . length ; bubbleSort (arr , n); console . log ("Sorted array: "); printArray (arr , n); PHP <?php // PHP Optimized implementation // of Bubble sort function bubbleSort (& $arr) { $n = sizeof ($arr); // Traverse through all array elements for ($i = 0 ; $i < $n ; $i ++) { $swapped = False ; // Last i elements are already // in place for ($j = 0 ; $j < $n - $i - 1 ; $j ++) { // Traverse the array from 0 to // n-i-1. Swap if the element // found is greater than the // next element if ($arr [$j] > $arr [$j + 1]) { $t = $arr [$j]; $arr [$j] = $arr [$j + 1]; $arr [$j + 1] = $t ; $swapped = True ; } } // If no two elements were swapped // by inner loop, then break if ($swapped == False) break ; } } // Driver code $arr = array (64 , 34 , 25 , 12 , 22 , 11 , 90); $len = sizeof ($arr); bubbleSort ($arr); echo "Sorted array: \n " ; for ($i = 0 ; $i < $len ; $i ++) echo $arr [$i] . " " ; // This code is contributed by ChitraNayal. ?> Output Sorted array: 11 12 22 25 34 64 90 Complexity Analysis of Bubble Sort: Time Complexity: O(n 2) Auxiliary Space: O(1) Please refer Complexity Analysis of Bubble Sort for details. Advantages of Bubble Sort: Bubble sort is easy to understand and implement. It does not require any additional memory space. It is a stable sorting algorithm, meaning that elements with the same key value maintain their relative order in the sorted output. Disadvantages of Bubble Sort: Bubble sort has a time complexity of O(n 2) which makes it very slow for large data sets. Bubble sort has almost no or limited real world applications. It is mostly used in academics to teach different ways of sorting. Recursive Bubble Sort Coding practice for sorting Quiz on Bubble Sort Complexity Analysis of Bubble Sort Comment Article Tags: Article Tags: Sorting DSA redBus Algorithms-BubbleSort BubbleSort + 1 More