Selection Sort - GeeksforGeeks

Source: https://www.geeksforgeeks.org/selection-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 Selection Sort Last Updated : 8 Dec, 2025 Selection Sort is a comparison-based sorting algorithm. It sorts by repeatedly selecting the smallest (or largest) element from the unsorted portion and swapping it with the first unsorted element. Find the smallest element and swap it with the first element. This way we get the smallest element at its correct position. Then find the smallest among remaining elements (or second smallest) and swap it with the second element. We keep doing this until we get all elements moved to correct position. Try it on GfG Practice C++ // C++ program to implement Selection Sort #include <bits/stdc++.h> using namespace std ; void selectionSort (vector < int > & arr) { int n = arr . size (); for (int i = 0 ; i < n - 1 ; ++ i) { // Assume the current position holds // the minimum element int min_idx = i ; // Iterate through the unsorted portion // to find the actual minimum for (int j = i + 1 ; j < n ; ++ j) { if (arr [j] < arr [min_idx]) { // Update min_idx if a smaller // element is found min_idx = j ; } } // Move minimum element to its // correct position swap (arr [i], arr [min_idx]); } } void printArray (vector < int > & arr) { for (int & val : arr) { cout << val << " " ; } cout << endl ; } int main () { vector < int > arr = { 64 , 25 , 12 , 22 , 11 }; cout << "Original array: " ; printArray (arr); selectionSort (arr); cout << "Sorted array: " ; printArray (arr); return 0 ; } C // C program for implementation of selection sort #include <stdio.h> void selectionSort (int arr [], int n) { for (int i = 0 ; i < n - 1 ; i ++) { // Assume the current position holds // the minimum element int min_idx = i ; // Iterate through the unsorted portion // to find the actual minimum for (int j = i + 1 ; j < n ; j ++) { if (arr [j] < arr [min_idx]) { // Update min_idx if a smaller element is found min_idx = j ; } } // Move minimum element to its // correct position int temp = arr [i]; arr [i] = arr [min_idx]; arr [min_idx] = temp ; } } void printArray (int arr [], int n) { for (int i = 0 ; i < n ; i ++) { printf ("%d " , arr [i]); } printf (" \n "); } int main () { int arr [] = { 64 , 25 , 12 , 22 , 11 }; int n = sizeof (arr) / sizeof (arr [0]); printf ("Original array: "); printArray (arr , n); selectionSort (arr , n); printf ("Sorted array: "); printArray (arr , n); return 0 ; } Java // Java program for implementation of Selection Sort import java.util.Arrays ; class GfG { static void selectionSort (int [] arr){ int n = arr . length ; for (int i = 0 ; i < n - 1 ; i ++) { // Assume the current position holds // the minimum element int min_idx = i ; // Iterate through the unsorted portion // to find the actual minimum for (int j = i + 1 ; j < n ; j ++) { if (arr [j] < arr [min_idx]) { // Update min_idx if a smaller element // is found min_idx = j ; } } // Move minimum element to its // correct position int temp = arr [i] ; arr [i] = arr [min_idx] ; arr [min_idx] = temp ; } } static void printArray (int [] arr){ for (int val : arr) { System . out . print (val + " "); } System . out . println (); } public static void main (String [] args){ int [] arr = { 64 , 25 , 12 , 22 , 11 }; System . out . print ("Original array: "); printArray (arr); selectionSort (arr); System . out . print ("Sorted array: "); printArray (arr); } } Python # Python program for implementation of Selection # Sort def selection_sort (arr): n = len (arr) for i in range (n - 1): # Assume the current position holds # the minimum element min_idx = i # Iterate through the unsorted portion # to find the actual minimum for j in range (i + 1 , n): if arr [j] < arr [min_idx]: # Update min_idx if a smaller element is found min_idx = j # Move minimum element to its # correct position arr [i], arr [min_idx] = arr [min_idx], arr [i] def print_array (arr): for val in arr : print (val , end = " ") print () if __name__ == "__main__" : arr = [64 , 25 , 12 , 22 , 11] print ("Original array: " , end = "") print_array (arr) selection_sort (arr) print ("Sorted array: " , end = "") print_array (arr) C# // C# program for implementation // of Selection Sort using System ; class GfG { static void selectionSort (int [] arr){ int n = arr . Length ; for (int i = 0 ; i < n - 1 ; i ++) { // Assume the current position holds // the minimum element int min_idx = i ; // Iterate through the unsorted portion // to find the actual minimum for (int j = i + 1 ; j < n ; j ++) { if (arr [j] < arr [

min_idx]) { // Update min_idx if a smaller element // is found min_idx = j ; } } // Move minimum element to its // correct position int temp = arr [i]; arr [i] = arr [min_idx]; arr [min_idx] = temp ; } } static void printArray (int [] arr){ foreach (int val in arr){ Console . Write (val + " "); } Console . WriteLine (); } public static void Main (){ int [] arr = { 64 , 25 , 12 , 22 , 11 }; Console . Write ("Original array: "); printArray (arr); selectionSort (arr); Console . Write ("Sorted array: "); printArray (arr); } } JavaScript function selectionSort (arr) { let n = arr . length ; for (let i = 0 ; i < n - 1 ; i ++) { // Assume the current position holds // the minimum element let min_idx = i ; // Iterate through the unsorted portion // to find the actual minimum for (let j = i + 1 ; j < n ; j ++) { if (arr [j] < arr [min_idx]) { // Update min_idx if a smaller element is found min_idx = j ; } } // Move minimum element to its // correct position let temp = arr [i]; arr [i] = arr [min_idx]; arr [min_idx] = temp ; } } function printArray (arr) { for (let val of arr) { process . stdout . write (val + " "); } console . log (); } // Driver function const arr = [64 , 25 , 12 , 22 , 11]; console . log ("Original array: "); printArray (arr); selectionSort (arr); console . log ("Sorted array: "); printArray (arr); PHP <?php function selectionSort (& $arr) { $n = count ($arr); for ($i = 0 ; $i < $n - 1 ; $i ++) { // Assume the current position holds // the minimum element $min_idx = $i ; // Iterate through the unsorted portion // to find the actual minimum for ($j = $i + 1 ; $j < $n ; $j ++) { if ($arr [$j] < $arr [$min_idx]) { // Update min_idx if a smaller element is found $min_idx = $j ; } } // Move minimum element to its // correct position $temp = $arr [$i]; $arr [$i] = $arr [$min_idx]; $arr [$min_idx] = $temp ; } } function printArray ($arr) { foreach ($arr as $val) { echo $val . " " ; } echo " \n " ; } $arr = [64 , 25 , 12 , 22 , 11]; echo "Original array: " ; printArray ($arr); selectionSort ($arr); echo "Sorted array: " ; printArray ($arr); ?> Output Original vector: 64 25 12 22 11 Sorted vector: 11 12 22 25 64 Complexity Analysis of Selection Sort Time Complexity: O(n 2) ,as there are two nested loops: One loop to select an element of Array one by one = O(n) Another loop to compare that element with every other Array element = O(n) Therefore overall complexity = O(n) * O(n) = O(n*n) = O(n 2) Auxiliary Space: O(1) as the only extra memory used is for temporary variables. Advantages of Selection Sort Easy to understand and implement, making it ideal for teaching basic sorting concepts. Requires only a constant O(1) extra memory space. It requires less number of swaps (or memory writes) compared to many other standard algorithms. Only cycle sort beats it in terms of memory writes. Therefore it can be simple algorithm choice when memory writes are costly. Disadvantages of the Selection Sort Selection sort has a time complexity of O(n^2) makes it slower compared to algorithms like Quick Sort or Merge Sort . Does not maintain the relative order of equal elements which means it is not stable. Applications of Selection Sort Perfect for teaching fundamental sorting mechanisms and algorithm design. Suitable for small lists where the overhead of more complex algorithms isn't justified and memory writing is costly as it requires less memory writes compared to other standard sorting algorithms. Heap Sort algorithm is based on Selection Sort. Question 1: Is Selection Sort a stable sorting algorithm? Answer: No, Selection Sort is not stable as it may change the relative order of equal elements. Question 2: What is the time complexity of Selection Sort? Answer: Selection Sort has a time complexity of O(n^2) in the best, average, and worst cases. Question 3: Does Selection Sort require extra memory? Answer: No, Selection Sort is an in-place sorting algorithm and requires only O(1) additional space. Question 4: When is it best to use Selection Sort? Answer: Selection Sort is best used for small datasets, educational purposes, or when memory usage needs to be minimal. Question 5: How does Selection Sort differ from Bubble Sort? Answer: Selection Sort selects the minimum element and places it in the correct position with fewer swaps, while Bubble Sort repeatedly swaps adjacent elements to sort the array. Comment Article Tags: Article Tags: Sorting DSA Medlife