**Sorting Algorithms (Bubble Sort, Insertion Sort, Selection Sort)**

Given a sequence of elements, sorting them means that ordering them in a particular fashion. In general, when we talk about sorting, we are talking about ordering them in ascending order.

**Example**

Initial Sequence: 3, 4, 1, 6, 2, 5

After Sorting: 1, 2, 3, 4, 5, 6

The values might be integers, or strings, or even other kinds of objects.

The most popular sorting algorithms are:

* [Bubble Sort](https://workat.tech/problem-solving/tutorial/sorting-algorithms-bubble-insertion-selection-sort-veubp86w3e1r#bubble-sort)
* [Selection Sort](https://workat.tech/problem-solving/tutorial/sorting-algorithms-bubble-insertion-selection-sort-veubp86w3e1r#selection-sort)
* [Insertion Sort](https://workat.tech/problem-solving/tutorial/sorting-algorithms-bubble-insertion-selection-sort-veubp86w3e1r#insertion-sort)
* [Quick Sort](https://workat.tech/problem-solving/tutorial/sorting-algorithms-quick-sort-merge-sort-dsa-tutorials-6j3h98lk6j2w#quick-sort)
* [Merge Sort](https://workat.tech/problem-solving/tutorial/sorting-algorithms-quick-sort-merge-sort-dsa-tutorials-6j3h98lk6j2w#merge-sort)

We will be learning about all these algorithms.

Reference table for time complexities of popular sorting algorithms:

| **Algorithm** | **Best Case** | **Average Case** | **Worst Case** |
| --- | --- | --- | --- |
| **Bubble Sort** | O(n) | O(n2) | O(n2) |
| **Selection Sort** | O(n2) | O(n2) | O(n2) |
| **Insertion Sort** | O(n) | O(n2) | O(n2) |
| **Merge Sort** | O(n log(n)) | O(n log(n)) | O(n log(n)) |
| **Quick Sort** | O(n log(n)) | O(n log(n)) | O(n2) |

**Bubble Sort**

**Introduction**

Bubble Sort is one of the simplest sorting algorithms. It works by repeatedly iterating through the list, comparing adjacent elements, and swapping if they are in the wrong order.

Because of its poor performance, it is not practically used and is just used for educational purposes.

**Example**

Initial Array: 6 5 3 1 8 7 2 4

First Iteration:

* **5 6** 3 1 8 7 2 4
* 5 **3 6** 1 8 7 2 4
* 5 3 **1 6** 8 7 2 4
* 5 3 1 **6 8** 7 2 4
* 5 3 1 6 **7 8** 2 4
* 5 3 1 6 7 **2 8** 4
* 5 3 1 6 7 2 **4 8**

Second Iteration:

* **3 5** 1 6 7 2 4 8
* 3 **1 5** 6 7 2 4 8
* 3 1 **5 6** 7 2 4 8
* 3 1 5 **6 7** 2 4 8
* 3 1 5 6 **2 7** 4 8
* 3 1 5 6 2 **4 7** 8
* 3 1 5 6 2 4 **7 8**

Third Iteration:

* **1 3** 5 6 2 4 7 8
* 1 **3 5** 6 2 4 7 8
* 1 3 **5 6** 2 4 7 8
* 1 3 5 **2 6** 4 7 8
* 1 3 5 2 **4 6** 7 8
* 1 3 5 2 4 **6 7** 8
* 1 3 5 2 4 6 **7 8**

Fourth Iteration:

* **1 3** 5 2 4 6 7 8
* 1 **3 5** 2 4 6 7 8
* 1 3 **2 5** 4 6 7 8
* 1 3 2 **4 5** 6 7 8
* 1 3 2 4 **5 6** 7 8
* 1 3 2 4 5 **6 7** 8
* 1 3 2 4 5 6 **7 8**

Fifth Iteration:

* **1 3** 2 4 5 6 7 8
* 1 **2 3** 4 5 6 7 8
* 1 2 **3 4** 5 6 7 8
* 1 2 3 **4 5** 6 7 8
* 1 2 3 4 **5 6** 7 8
* 1 2 3 4 5 **6 7** 8

1 2 3 4 5 6 **7 8**

**Visualization**

**Code**

**C++**

void bubbleSort(int arr[], int n) {

bool swapped;

while (true) {

swapped = false;

for (int i = 1; i < n; i++) {

if (arr[i - 1] > arr[i]) {

int temp = arr[i - 1];

arr[i - 1] = arr[i];

arr[i] = temp;

swapped = true;

}

}

if (swapped == false) {

break;

}

}

}

**Java**

static void bubbleSort(int[] arr) {

int n = arr.length;

boolean swapped;

while (true) {

swapped = false;

for (int i = 1; i < n; i++) {

if (arr[i - 1] > arr[i]) {

int temp = arr[i - 1];

arr[i - 1] = arr[i];

arr[i] = temp;

swapped = true;

}

}

if (swapped == false) {

break;

}

}

}

**Complexity Analysis**

Time Complexity

* Best Case: O(n)
* Average Case: O(n2)
* Worst Case: O(n2)

Space Complexity: O(1)

**Selection Sort**

**Introduction**

Selection Sort is one of the simplest sorting algorithms. It works by creating a sorted array by finding and adding the correct element in each iteration.

The sequence (array) is divided into two parts: sorted part and unsorted part. In each iteration, pick the smallest element from the unsorted part, exchange it with the leftmost unsorted element and move the boundary by one place to include another element in the sorted part. After all the iterations, the entire array gets sorted.

**Example**

Initial Array: 3, 4, 1, 6, 2, 5

In each iteration, we will move the smallest element from the unsorted part to the sorted part. The sorted part is marked bold.

Before the first iteration:

3, 4, 1, 6, 2, 5

After iteration 1:

**1**, 4, 3, 6, 2, 5

After iteration 2:

**1, 2**, 3, 6, 4, 5

After iteration 3:

**1, 2, 3**, 6, 4, 5

After iteration 4:

**1, 2, 3, 4**, 6, 5

After iteration 5:

**1, 2, 3, 4, 5**, 6

After iteration 6:

**1, 2, 3, 4, 5, 6**

**Visualization**

**Code**

**C++**

void selectionSort(int arr[], int n) {

for (int i = 0; i < n - 1; i++) {

int minIndex = i;

for (int j = i + 1; j < n; j++) {

if (arr[j] < arr[minIndex]) {

minIndex = j;

}

}

if (minIndex != i) {

int temp = arr[i];

arr[i] = arr[minIndex];

arr[minIndex] = temp;

}

}

}

**Java**

static void selectionSort(int[] arr) {

int n = arr.length;

for (int i = 0; i < n - 1; i++) {

int minIndex = i;

for (int j = i + 1; j < n; j++) {

if (arr[j] < arr[minIndex]) {

minIndex = j;

}

}

if (minIndex != i) {

int temp = arr[i];

arr[i] = arr[minIndex];

arr[minIndex] = temp;

}

}

}

**Complexity Analysis**

Time Complexity

* Best Case: O(n2)
* Average Case: O(n2)
* Worst Case: O(n2)

Space Complexity: O(1)

**Insertion Sort**

**Introduction**

Insertion Sort is one of the simplest sorting algorithms. Most people use this algorithm subconsciously while arranging playing cards in their hands.

The sequence (array) is divided into two parts: sorted part and unsorted part. In each iteration, one element from the unsorted part is moved to the correct position in the sorted part. After all the iterations, the entire array gets sorted.

Insertion Sort is one of the fastest sorting algorithms for sorting **very small input**.

**Example**

Initial Array: 3, 4, 1, 6, 2, 5

In each iteration, we will move one element from the unsorted part to its correct position in the sorted part. The sorted part is marked bold.

Before first iteration:

**3**, 4, 1, 6, 2, 5

After iteration 1:

**3, 4**, 1, 6, 2, 5

After iteration 2:

**1, 3, 4**, 6, 2, 5

After iteration 3:

**1, 3, 4, 6**, 2, 5

After iteration 4:

**1, 2, 3, 4, 6**, 5

After iteration 5:

**1, 2, 3, 4, 5, 6**

**Visualization**

**Code**

**C++**

void insertionSort(int arr[], int n) {

for (int i = 1; i < n; i++) {

int key = arr[i];

int j = i - 1;

while (j >= 0 && arr[j] > key) {

arr[j + 1] = arr[j];

j = j - 1;

}

arr[j + 1] = key;

}

}

**Java**

static void insertionSort(int arr[]) {

int n = arr.length;

for (int i = 1; i < n; i++) {

int key = arr[i];

int j = i - 1;

while (j >= 0 && arr[j] > key) {

arr[j + 1] = arr[j];

j = j - 1;

}

arr[j + 1] = key;

}

}

**Complexity Analysis**

Time Complexity

* Best Case: O(n)
* Average Case: O(n2)
* Worst Case: O(n2)

Space Complexity: O(1)

Part II: [Sorting Algorithms (Quick Sort, Merge Sort) | DSA Tutorials](https://workat.tech/problem-solving/tutorial/sorting-algorithms-quick-sort-merge-sort-dsa-tutorials-6j3h98lk6j2w)

Selection sort

**C#: Selection sort**

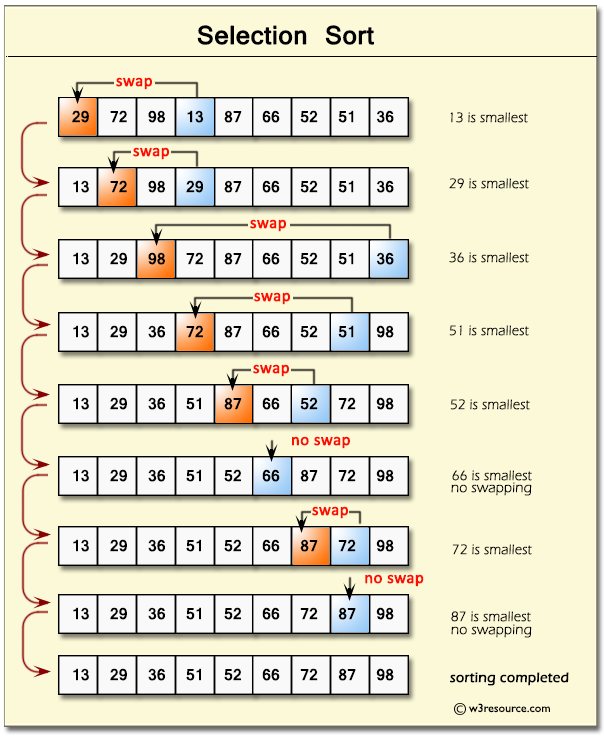
Last update on January 18 2024 06:20:01 (UTC/GMT +8 hours)

C# Sharp Searching and Sorting Algorithm: Exercise-11 with Solution

Write a C# Sharp program to sort a list of elements using the selection sort algorithm.

The selection sort improves on the bubble sort by making only one exchange for every pass through the list.

**Pictorial Presentation : Selection Sort**



**Sample Solution**:-

**C# Sharp Code:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Selection\_Sort

{

class Program

{

static void Main(string[] args)

{

// Create an instance of the Selection\_Sort class with an array of 10 random numbers

Selection\_Sort selection = new Selection\_Sort(10);

// Call the Sort method of the Selection\_Sort class to perform selection sort

selection.Sort();

}

}

class Selection\_Sort

{

private int[] data;

private static Random generator = new Random();

// Constructor to create an array of specified size filled with random numbers

public Selection\_Sort(int size)

{

data = new int[size];

// Fill the array with random numbers between 20 and 90

for (int i = 0; i < size; i++)

{

data[i] = generator.Next(20, 90);

}

}

// Method to perform selection sort

public void Sort()

{

// Display the original array elements before sorting

Console.Write("\nSorted Array Elements :(Step by Step)\n\n");

display\_array\_elements();

int smallest;

// Iterate through the array to perform selection sort

for (int i = 0; i < data.Length - 1; i++)

{

smallest = i;

// Find the index of the smallest element in the unsorted part of the array

for (int index = i + 1; index < data.Length; index++)

{

if (data[index] < data[smallest])

{

smallest = index;

}

}

// Swap the current element with the smallest element found

Swap(i, smallest);

// Display array elements after each swap (step-by-step)

display\_array\_elements();

}

}

// Method to swap two elements in the array

public void Swap(int first, int second)

{

int temporary = data[first];

data[first] = data[second];

data[second] = temporary;

}

// Method to display array elements

public void display\_array\_elements()

{

// Display each element of the array

foreach (var element in data)

{

Console.Write(element + " ");

}

Console.Write("\n\n");

}

}

}

Copy

Sample Output:

Sorted Array Elements :(Step by Step)

50 70 50 80 89 25 78 58 83 73

25 70 50 80 89 50 78 58 83 73

25 50 70 80 89 50 78 58 83 73

25 50 50 80 89 70 78 58 83 73

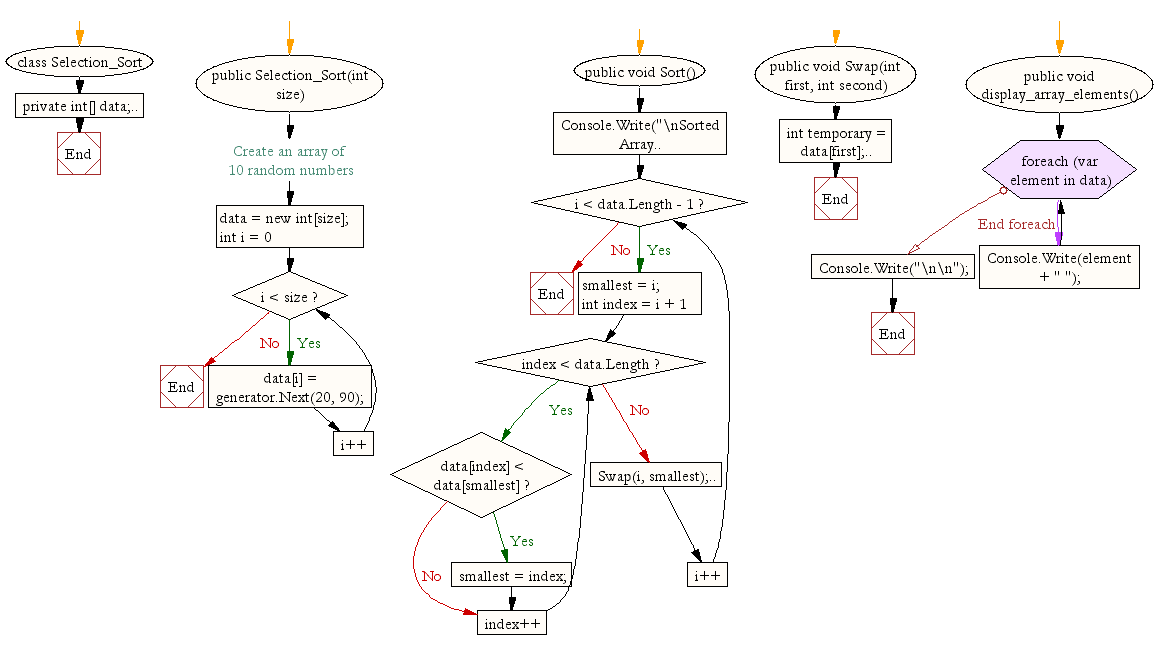
25 50 50 58 89 70 78 80 83 73

25 50 50 58 70 89 78 80 83 73

25 50 50 58 70 73 78 80 83 89

25 50 50 58 70 73 78 80 83 89

**Flowchart:**



**Insertion sort :**

**Insertion Sort in C#**

[Csharp](https://www.tutorialspoint.com/articles/category/Csharp)[Programming](https://www.tutorialspoint.com/articles/category/Programming)[Server Side Programming](https://www.tutorialspoint.com/articles/category/Server-Side-Programming)

Insertion Sort is a sorting algorithm that takes an element at a time and inserts it in its correct position in the array. This process is continued until the array is sorted.

A program that demonstrates insertion sort in C# is given as follows.

Example

using System;

namespace InsertionSortDemo {

   class Example {

      static void Main(string[] args) {

         int[] arr = new int[10] { 23, 9, 85, 12, 99, 34, 60, 15, 100, 1 };

         int n = 10, i, j, val, flag;

         Console.WriteLine("Insertion Sort");

         Console.Write("Initial array is: ");

         for (i = 0; i < n; i++) {

            Console.Write(arr[i] + " ");

         }

         for (i = 1; i < n; i++) {

            val = arr[i];

            flag = 0;

            for (j = i - 1; j >= 0 && flag != 1; ) {

               if (val < arr[j]) {

                  arr[j + 1] = arr[j];

                  j--;

                  arr[j + 1] = val;

               }

               else flag = 1;

            }

         }

         Console.Write("  
Sorted Array is: ");

         for (i = 0; i < n; i++) {

            Console.Write(arr[i] + " ");

         }

      }

   }

}

Output

The output of the above program is as follows.

Insertion Sort

Initial array is: 23 9 85 12 99 34 60 15 100 1

Sorted Array is: 1 9 12 15 23 34 60 85 99 100

Now, let us understand the above program.

First the array is initialized and its value is printed using a for loop. This can be seen in the following code snippet −

int[] arr = new int[10] { 23, 9, 85, 12, 99, 34, 60, 15, 100, 1 };

int n = 10, i, j, val, flag;

Console.WriteLine("Insertion Sort");

Console.Write("Initial array is: ");

for (i = 0; i < n; i++) {

   Console.Write(arr[i] + " ");

}

A nested for loop is used for the actual sorting process. In each pass of the outer for loop, the current element is inserted into its correct position in the array. This process continues until the array is sorted. This can be seen in the following code snippet.

for (i = 1; i < n; i++) {

   val = arr[i];

   flag = 0;

   for (j = i - 1; j >= 0 && flag != 1; ) {

      if (val < arr[j]) {

         arr[j + 1] = arr[j];

         j--;

         arr[j + 1] = val;

      } else flag = 1;

   }

}

Finally, the sorted array is displayed. This can be seen in the following code snippet.

Console.Write("  
Sorted Array is: ");

for (i = 0; i < n; i++) {

   Console.Write(arr[i] + " ");

}