# **Алгоритм сортування вибірки**

# **(Selection Sort Algorithm)**

**Сортування вибором(Selection Sort Algorithm)** - це простий і ефективний алгоритм сортування, який працює шляхом багаторазового вибору найменшого (або найбільшого) елемента з невідсортованої частини списку і переміщення його до відсортованої частини списку. Алгоритм багаторазово вибирає найменший (або найбільший) елемент з невідсортованої частини списку і міняє його місцями з першим елементом невідсортованої частини. Цей процес повторюється для решти не відсортованої частини списку, поки весь список не буде відсортовано. Одна з варіацій сортування вибором називається "Двонаправлене сортування вибором", яке проходить через список елементів, чергуючи найменший і найбільший елементи, таким чином алгоритм може бути швидким у деяких випадках.

Алгоритм підтримує два підмасиви у заданому масиві.

* Підмасив, який вже відсортовано.
* Підмасив, що залишився невідсортованим.

На кожній ітерації сортування вибором вибирається мінімальний елемент (за зростанням) з не відсортованого підмасиву і переміщується на початок не відсортованого підмасиву. Після кожної ітерації розмір відсортованого підмасиву збільшується на одиницю, а розмір не відсортованого підмасиву зменшується на одиницю.

***Як працює вибіркове сортування?***

Розглянемо наступний масив як приклад: arr[] = {64, 25, 12, 22, 11}

***Перший прохід:***

Для першої позиції у відсортованому масиві послідовно обходиться весь масив від індексу 0 до 4. У першій позиції, де зараз зберігається 64, після обходу всього масиву стає зрозуміло, що 11 є найменшим значенням.

| **64** | 25 | 12 | 22 | 11 |
| --- | --- | --- | --- | --- |

Таким чином, замініть 64 на 11. Після однієї ітерації 11, яке є найменшим значенням у масиві, як правило, з'являється на першій позиції відсортованого списку.

| **11** | 25 | 12 | 22 | 64 |
| --- | --- | --- | --- | --- |

***Другий прохід:***

Для другої позиції, де присутнє 25, знову послідовно пройдіться по решті масиву.

| 11 | **25** | 12 | 22 | 64 |
| --- | --- | --- | --- | --- |

Після обходу ми виявили, що 12 є другим найменшим значенням у масиві, і воно повинно з'явитися на другому місці у масиві, тому поміняємо ці значення місцями.

| 11 | **12** | 25 | 22 | 64 |
| --- | --- | --- | --- | --- |

***Третій прохід:***

Тепер, для третього місця, де присутнє 25, знову пройдіться по решті масиву і знайдіть третє найменше значення, присутнє в масиві.

| 11 | 12 | **25** | 22 | 64 |
| --- | --- | --- | --- | --- |

Під час обходу 22 виявилося третім найменшим значенням, і воно повинно бути на третьому місці в масиві, тому поміняємо 22 з елементом, що знаходиться на третій позиції.

| 11 | 12 | **22** | 25 | 64 |
| --- | --- | --- | --- | --- |

***Четвертий прохід:***

Аналогічно, для четвертої позиції пройдіться рештою масиву і знайдіть четвертий найменший елемент у масиві. Оскільки 25 є 4-м найменшим значенням, отже, його буде розміщено на четвертій позиції.

| 11 | 12 | 22 | **25** | 64 |
| --- | --- | --- | --- | --- |

***П'ятий прохід:***

Нарешті, найбільше значення, присутнє в масиві, автоматично потрапляє на останню позицію в масиві. Отриманий масив є відсортованим масивом.

| **11** | **12** | **22** | **25** | **64** |
| --- | --- | --- | --- | --- |

**Щоб розв'язати задачу, виконайте наступні кроки:**

* І***ніціалізуйте мінімальне значення (min\_idx) у позиції 0.***
* ***Пройдіться по масиву, щоб знайти мінімальний елемент у масиві.***
* ***Якщо під час обходу буде знайдено елемент, менший за min\_idx, то поміняйте місцями обидва значення.***
* ***Потім збільште min\_idx, щоб вказати на наступний елемент.***
* ***Повторюйте до тих пір, поки масив не буде відсортовано.***

using System;

class GFG

{

static void sort(int []arr)

{

int n = arr.Length;

// One by one move boundary of unsorted subarray

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

{

// Find the minimum element in unsorted array

int min\_idx = i;

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

if (arr[j] < arr[min\_idx])

min\_idx = j;

// Swap the found minimum element with the first

// element

int temp = arr[min\_idx];

arr[min\_idx] = arr[i];

arr[i] = temp;

}

}

// Prints the array

static void printArray(int []arr)

{

int n = arr.Length;

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

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

Console.WriteLine();

}

// Driver code

public static void Main()

{

int []arr = {64,25,12,22,11};

sort(arr);

Console.WriteLine("Sorted array");

printArray(arr);

}

}

# **Алгоритм бульбашкового сортування**

# **(Bubble Sort Algorithm)**

Бульбашкове сортування - це найпростіший алгоритм сортування, який працює шляхом багаторазової заміни сусідніх елементів місцями, якщо вони розташовані в неправильному порядку. Цей алгоритм не підходить для великих наборів даних, оскільки його середня і найгірша часова складність досить висока.

**Як працює бульбашкове сортування?**

**Input:** arr[] = {5, 1, 4, 2, 8}

***Перший прохід:***

Бульбашкове сортування починається з перших двох елементів, порівнюючи їх, щоб визначити, який з них більший.

* ( **5 1** 4 2 8 ) -> ( **1 5** 4 2 8 ), Тут алгоритм порівнює перші два елементи і міняє місцями, оскільки 5 > 1.
* ( 1 **5 4** 2 8 ) -> ( 1 **4 5** 2 8 ), Міняємо місцями, оскільки 5 > 4
* ( 1 **4 5** 2 8 ) -> ( 1 **4 2** 5 8 ), Міняємо місцями, оскільки 5 > 2
* ( 1 4 2 5 8 ) -> ( 1 4 2 5 8 ), Тепер, оскільки ці елементи вже впорядковані (8 > 5), алгоритм не міняє їх місцями.

***Другий прохід:***

Тепер, під час другої ітерації, це має виглядати так:

* ( 1 **4 2** 5 8 ) -> ( 1 **2 4** 5 8 ), поміняти місцями, оскільки 4 > 2
* ( 1 2 4 5 8 ) -> ( 1 2 4 5 8 )

***Третій прохід:***

Тепер масив вже відсортовано, але наш алгоритм не знає, чи завершено його сортування. Алгоритму потрібен один прохід без заміни, щоб дізнатися, що масив відсортовано.

* (**1** **2** 4 5 8 ) –> ( **1** **2** 4 5 8 )
* ( 1 **2** **4** 5 8 ) –> ( 1 **2** **4** 5 8 )
* ( 1 2 **4** **5** 8 ) –> ( 1 2 **4** **5** 8 )
* ( 1 2 4 **5** **8** ) –> ( 1 2 4 **5** **8** )

Для розв'язання задачі виконайте наступні кроки:

* Запустіть вкладений цикл for для обходу вхідного масиву з використанням двох змінних i та j, таких що 0 ≤ i < n-1 та 0 ≤ j < n-i-1
* Якщо arr[j] більше arr[j+1], то поміняти місцями сусідні елементи, інакше перейти далі
* Вивести відсортований масив

class GFG {

static void bubbleSort(int[] arr)

{

int n = arr.Length;

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

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

if (arr[j] > arr[j + 1]) {

// swap temp and arr[i]

int temp = arr[j];

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

arr[j + 1] = temp;

}

}

/\* Prints the array \*/

static void printArray(int[] arr)

{

int n = arr.Length;

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

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

Console.WriteLine();

}

// Driver method

public static void Main()

{

int[] arr = { 5, 1, 4, 2, 8};

bubbleSort(arr);

Console.WriteLine("Sorted array");

printArray(arr);

}

}

# **Сортування вставок**

# **(Insertion Sort)**

Сортування вставками - це простий алгоритм сортування, який працює подібно до того, як ви сортуєте гральні карти в руках. Масив віртуально розділяється на відсортовану і невідсортовану частини. Значення з невідсортованої частини вибираються і поміщаються в потрібну позицію у відсортованій частині.

Характеристики сортування вставками:

* Цей алгоритм є одним з найпростіших алгоритмів з простою реалізацією
* В основному, сортування вставками ефективне для невеликих значень даних
* Сортування вставками є адаптивним за своєю природою, тобто підходить для наборів даних, які вже частково відсортовані.

**Робота алгоритму сортування вставками:**

Розглянемо приклад: arr[]: {12, 11, 13, 5, 6}

| **12** | **11** | **13** | **5** | **6** |
| --- | --- | --- | --- | --- |

***Перший прохід:***

Спочатку порівнюються перші два елементи масиву у сортуванні вставкою:

| **12** | **11** | 13 | 5 | 6 |
| --- | --- | --- | --- | --- |

* Тут 12 більше за 11, отже, вони не в порядку зростання, і 12 не на своєму місці.
* Отже, поміняємо місцями 11 і 12.
* Отже, поки що 11 зберігається у відсортованому підмасиві.

| **11** | **12** | 13 | 5 | 6 |
| --- | --- | --- | --- | --- |

***Другий прохід:***

Тепер перейдіть до наступних двох елементів і порівняйте їх:

| 11 | **12** | **13** | 5 | 6 |
| --- | --- | --- | --- | --- |

Тут 13 більше, ніж 12, тому обидва елементи знаходяться у порядку зростання, отже, заміни не відбудеться. 12 також зберігається у відсортованому підмасиві разом з 11

***Третій прохід:***

1.Тепер у відсортованому підмасиві присутні два елементи - 11 і 12.

2.Переходимо до наступних двох елементів 13 і 5:

| 11 | 12 | **13** | **5** | 6 |
| --- | --- | --- | --- | --- |

3.Обидві цифри 5 і 13 знаходяться не на своїх місцях, тому поміняйте їх місцями:

| 11 | 12 | **5** | **13** | 6 |
| --- | --- | --- | --- | --- |

4.Після заміни місцями елементи 12 та 5 не відсортовано, тому знову поміняємо місцями:

| 11 | **5** | **12** | 13 | 6 |
| --- | --- | --- | --- | --- |

5.Тут знову 11 і 5 не відсортовані, отже, знову міняємо місцями

| **5** | **11** | 12 | 13 | 6 |
| --- | --- | --- | --- | --- |

6.Тут 5 знаходиться в правильному положенні

| **5** | 11 | 12 | 13 | 6 |
| --- | --- | --- | --- | --- |

***Четвертий прохід:***

Тепер у відсортованому підмасиві присутні елементи 5, 11 і 12

1.Переходимо до наступних двох елементів 13 і 6

| 5 | 11 | 12 | **13** | **6** |
| --- | --- | --- | --- | --- |

2.Очевидно, що вони не відсортовані, тому виконайте заміну між ними

| 5 | 11 | 12 | **6** | **13** |
| --- | --- | --- | --- | --- |

3.Тепер 6 менше ніж 12, отже, знову міняємо місцями

| 5 | 11 | **6** | **12** | 13 |
| --- | --- | --- | --- | --- |

4.Тут також перестановка робить 11 і 6 не відсортованими, отже, знову міняємо місцями

| 5 | **6** | **11** | 12 | 13 |
| --- | --- | --- | --- | --- |

Алгоритм сортування вставками

Відсортувати масив розміром N за зростанням:

* Перебираємо масив від arr[1] до arr[N].
* Порівняти поточний елемент (ключ) з його попередником.
* Якщо ключовий елемент менший за свого попередника, порівняйте його з попередніми елементами. Перемістіть більші елементи на одну позицію вгору, щоб звільнити місце для заміненого елемента.

using System;

class InsertionSort {

// Function to sort array

// using insertion sort

void sort(int[] arr)

{

int n = arr.Length;

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

int key = arr[i];

int j = i - 1;

// Move elements of arr[0..i-1],

// that are greater than key,

// to one position ahead of

// their current position

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

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

j = j - 1;

}

arr[j + 1] = key;

}

}

// A utility function to print

// array of size n

static void printArray(int[] arr)

{

int n = arr.Length;

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

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

Console.Write("\n");

}

// Driver Code

public static void Main()

{

int[] arr = { 12, 11, 13, 5, 6 };

InsertionSort ob = new InsertionSort();

ob.sort(arr);

printArray(arr);

}

}