

### Теоретический материал

**Алгоритм сортировки** — это алгоритм для упорядочивания элементов в массиве.

По некоторым источникам, именно программа сортировки стала первой программой для вычислительных машин. Некоторые конструкторы ЭВМ, в частности разработчики EDVAC, называли задачу сортировки данных наиболее характерной нечисловой задачей для вычислительных машин. В 1945 году Джон фон Нейман для тестирования ряда команд для EDVAC разработал программы сортировки методом слияния. В том же году немецкий инженер Конрад Цузе разработал программу для сортировки методом простой вставки.

В 50-х – 60-х гг XX века было предложено множество различных алгоритмов сортировки: слияние с вставкой, обменная поразрядная сортировка, каскадное слияние и метод Шелла в 1959 году, многофазное слияние и вставки в дерево в 1960 году, осциллирующая сортировка и быстрая сортировка Хоара в 1962 году, пирамидальная сортировка Уильямса и обменная сортировка со слиянием Бэтчера в 1964 году. Появившиеся позже алгоритмы во многом являлись вариациями уже известных методов. Получили распространение адаптивные методы сортировки, ориентированные на более быстрое выполнение в случаях, когда входная последовательность удовлетворяет заранее установленным критериям.

Для реализации алгоритмов сортировки необходимо располагать инструментами, позволяющими произвести обмен значениями двум различным ячейкам массива. *Пример 1 демонстрирует, как можно организовать такой обмен (swap) с помощью средств стандартной библиотеки и с помощью собственной функции.*

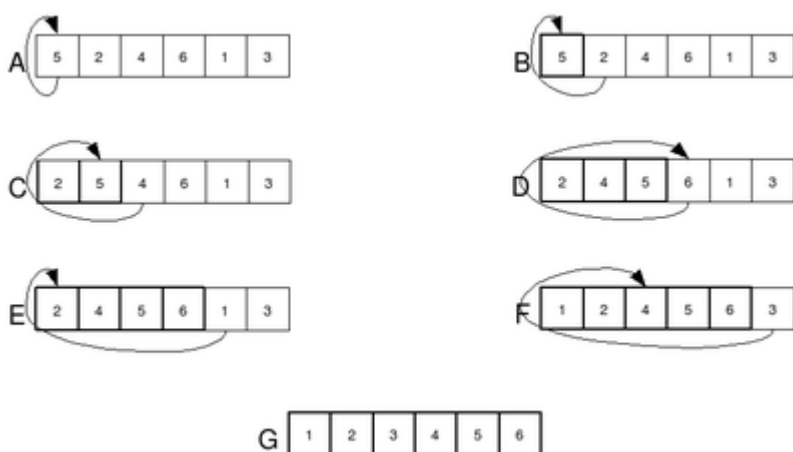
Некоторые алгоритмы сортировки быстрее производят сортировку небольших массивов. Некоторые алгоритмы сортировки напротив более эффективны с большими массивами.

**Сортировка простыми обменами, сортировка пузырьком (англ. bubble sort)** — простой алгоритм сортировки. Алгоритм состоит из повторяющихся проходов по сортируемому массиву. За каждый проход элементы последовательно сравниваются попарно и, если порядок в паре неверный, выполняется перестановка элементов. Проходы по массиву повторяются  $N-1$  раз или до тех пор, пока на очередном проходе не окажется, что обмены больше не нужны, что

означает — массив отсортирован. При каждом проходе алгоритма по внутреннему циклу очередной наибольший элемент массива ставится на своё место в конце массива рядом с предыдущим «наибольшим элементом», а наименьший элемент перемещается на одну позицию к началу массива («всплывает» до нужной позиции, как пузырёк в воде — отсюда и название алгоритма).

**Сортировка перемешиванием, или Шейкерная сортировка, или двунаправленная (англ. Cocktail sort)** — разновидность пузырьковой сортировки. Анализируя метод пузырьковой сортировки, можно отметить два обстоятельства. Во-первых, если при движении по части массива перестановки не происходят, то эта часть массива уже отсортирована и, следовательно, её можно исключить из рассмотрения. Во-вторых, при движении от конца массива к началу минимальный элемент «всплывает» на первую позицию, а максимальный элемент сдвигается только на одну позицию вправо. Эти две идеи приводят к следующим модификациям в методе пузырьковой сортировки. Границы рабочей части массива (то есть части массива, где происходит движение) устанавливаются в месте последнего обмена на каждой итерации. Массив просматривается поочередно справа налево и слева направо.

**Сортировка вставками (англ. Insertion sort)** — алгоритм сортировки, в котором элементы входной последовательности просматриваются по одному, и каждый новый поступивший элемент размещается в подходящее место среди ранее упорядоченных элементов.



## Сортировка выбором

Шаги алгоритма:

1. находим номер минимального значения в текущем списке
2. производим обмен этого значения со значением первой неотсортированной позиции (обмен не нужен, если минимальный элемент уже находится на данной позиции)
3. теперь сортируем хвост списка, исключив из рассмотрения уже отсортированные элементы

**Сортировка Шелла (англ. Shell sort)** — алгоритм сортировки, являющийся усовершенствованным вариантом сортировки вставками. Идея метода Шелла состоит в сравнении элементов, стоящих не только рядом, но и на определённом расстоянии друг от друга. Иными словами — это сортировка вставками с предварительными «грубыми» проходами.

При сортировке Шелла сначала сравниваются и сортируются между собой значения, стоящие один от другого на некотором расстоянии  $d$ . После этого процедура повторяется для некоторых меньших значений  $d$ , а завершается сортировка Шелла упорядочиванием элементов при  $d=1$  (то есть обычной сортировкой вставками). Эффективность сортировки Шелла в определённых случаях обеспечивается тем, что элементы «быстрее» встают на свои места (в простых методах сортировки, например, пузырьковой, каждая перестановка двух элементов уменьшает количество инверсий в списке максимум на 1, а при сортировке Шелла это число может быть больше).

Среднее время работы алгоритма зависит от длин промежутков  $d$ , на которых будут находиться сортируемые элементы исходного массива ёмкостью  $N$  на каждом шаге алгоритма. Существует несколько подходов к выбору этих значений, простейший из которых — это первоначально используемая Шеллом последовательность длин промежутков:  $d = N/2$ ,  $d_i = d_{i-1}/2$ ,  $d_k = 1$ . Про другие способы выбора длин промежутков можно прочитать, например, на [https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B0\\_%D0%A8%D0%B5%D0%BB%D0%BB%D0%B0](https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B0_%D0%A8%D0%B5%D0%BB%D0%BB%D0%B0)

**Быстрая сортировка, сортировка Хоара (англ. quicksort)**, часто называемая qsort (по имени в стандартной библиотеке языка Си)

QuickSort является существенно улучшенным вариантом алгоритма сортировки с помощью прямого обмена, известного в том числе своей низкой эффективностью. Принципиальное отличие состоит в том, что в первую очередь производятся перестановки на наибольшем возможном расстоянии и после каждого прохода элементы делятся на две независимые группы (таким образом улучшение самого неэффективного прямого метода сортировки дало в результате один из наиболее эффективных улучшенных методов).

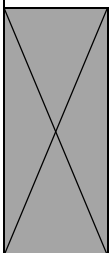
Общая идея алгоритма состоит в следующем:

- Выбрать из массива элемент, называемый опорным. Это может быть любой из элементов массива. От выбора опорного элемента не зависит корректность алгоритма, но в отдельных случаях может сильно зависеть его эффективность (см. ниже).
- Сравнить все остальные элементы с опорным и переставить их в массиве так, чтобы разбить массив на три непрерывных отрезка, следующих друг за другом: «элементы меньше опорного», «равные» и «большие»<sup>[2]</sup>.
- Для отрезков «меньших» и «больших» значений выполнить рекурсивно ту же последовательность операций, если длина отрезка больше единицы.

На практике массив обычно делят не на три, а на две части: например, «меньшие опорного» и «равные и большие»; такой подход в общем случае эффективнее, так как упрощает алгоритм разделения

### Пример 1

#### Задача:



Написать на языке C++ программу, в которой для динамически заданного массива производится перестановка местами двух его произвольных элементов. Реализовать перестановку функцией стандартной библиотеки `std::swap`, а также написать собственную функцию. Сделать замеры времени выполнения функций `swap` для обоих случаев

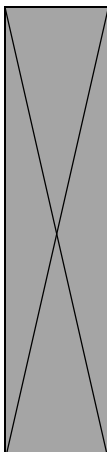
#### Решение:

```

1  #include <iostream>
2  #include <chrono> //Для оценки времени работы функции
3
4  using namespace std;
5
6  void my_swap(int& x, int& y); //Собственная функция swap()
7
8  int main()
9  {
10     setlocale(LC_ALL, "Russian");
11     srand((unsigned)time(NULL)); //Для генератора случайных чисел
12     srand((unsigned)rand()); //Для генератора случайных чисел
13     int count = 0; //Переменная для хранения размера массива
14     cout << "Введите размер массива (больше 10): ";
15     cin >> count;
16
17     int* num = new int[count]; //Массив
18     for (int i = 0; i < count; i++)
19     {
20         num[i] = rand() % 31; //заполнение массива числами от 0 до 30
21     }
22
23     // Вывод массива
24     cout << "Ваш массив:" << endl;
25     for (int i = 0; i < count; i++)
26     {
27         cout << num[i] << " ";
28     }
29     cout << endl;
30     cout << endl;
31
32     //Перестановка 2 и 5 элементов с помощью std::swap
33     auto begin = std::chrono::steady_clock::now(); //Старт замера работы функции
34     std::swap(num[2], num[5]);
35     auto end = std::chrono::steady_clock::now(); //Окончание замера работы функции
36     cout << "Массив с переставленными 2 и 5 элементами" << endl;
37     for (int i = 0; i < count; i++)
38     {
39         cout << num[i] << " ";
40     }
41     cout << endl;
42
43     auto elapsed_ms = std::chrono::duration_cast<std::chrono::nanoseconds>(end - begin); // Расчет времени
44     std::cout << "Время работы стандартной функции swap: " << elapsed_ms.count() << " ns\n";
45     cout << endl;
46
47     //Перестановка обратно 2 и 5 элементов с помощью std::swap
48     begin = std::chrono::steady_clock::now();
49     my_swap(num[2], num[5]);
50     end = std::chrono::steady_clock::now();
51     cout << "Массив с обратно переставленными 2 и 5 элементами" << endl;
52     for (int i = 0; i < count; i++)
53     {
54         cout << num[i] << " ";
55     }
56     cout << endl;
57
58     elapsed_ms = std::chrono::duration_cast<std::chrono::nanoseconds>(end - begin);
59     std::cout << "Время работы собственной функции swap: " << elapsed_ms.count() << " ns\n";
60
61     delete[] num;
62 }
63
64 void my_swap(int& x, int& y)
65 {
66     int temp;
67     temp = x;
68     x = y;
69     y = temp;
70 }
71

```

**Ответ:**

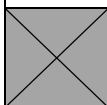


Консоль отладки Microsoft Visual Studio

Введите размер массива (больше 10): 15  
Ваш массив:  
8 6 16 22 3 15 29 12 19 0 19 23 9 1 2  
  
Массив с переставленными 2 и 5 элементами  
8 6 15 22 3 16 29 12 19 0 19 23 9 1 2  
Время работы стандартной функции swap: 300 ns  
  
Массив с обратно переставленными 2 и 5 элементами  
8 6 16 22 3 15 29 12 19 0 19 23 9 1 2  
Время работы собственной функции swap: 200 ns

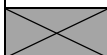
### Задание 1

#### Задача:



Написать программу для сортировки массива пузырьковым алгоритмом. Здесь и далее во всех задачах массив сортируется по возрастанию.

#### Решение:

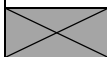


#### Ответ:



### Задание 2\*

#### Задача:



Написать программу для сортировки массива шейкерным алгоритмом

#### Решение:

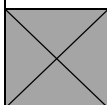


#### Ответ:



### Задание 3

#### Задача:



Написать программу для сортировки массива алгоритмом сортировки вставками

#### Решение:

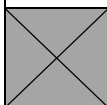


#### Ответ:



### Задание 4

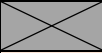
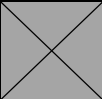
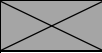

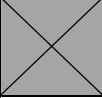

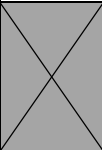
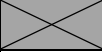

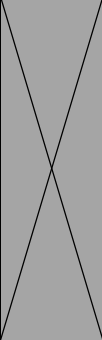

#### Задача:



Написать программу для сортировки массива алгоритмом сортировки выбором

#### Решение:



<b>Ответ:</b>	
	
<b>Задание 5*</b>	
<b>Задача:</b>	
	Написать программу для сортировки массива алгоритмом сортировки Шелла
<b>Решение:</b>	
	
<b>Ответ:</b>	
	
<b>Задание 6*</b>	
<b>Задача:</b>	
	Написать программу для сортировки массива алгоритмом быстрой сортировки
<b>Решение:</b>	
	
<b>Ответ:</b>	
<b>Задание 7*</b>	
<b>Задача:</b>	
	Оформить алгоритмы сортировки из заданий 1-6 в виде функций. Сделать замеры времени выполнения этих функций для одного и того же исходного массива (маленького и большого размера).
<b>Решение:</b>	
	
<b>Ответ:</b>	
	
<b>Задание 8*</b>	
<b>Задача:</b>	
	Алгоритм вставки допускает простое и очень эффективное улучшение. Ключевой момент вставки – это поиск правильной позиции. Заметим, что если речь идет о поиске нового места для элемента с номером $L$ , то, значит, массив с 1 по $L-1$ элемент уже упорядочен. Поиск правильной позиции можно легко выполнить методом половинного деления. Напишите реализацию алгоритма вставки, в которой простой перебор заменен методом половинного деления. Сравните время работы алгоритмов.
<b>Решение:</b>	
	
<b>Ответ:</b>	