# Глава 3. Массивы.

## *Методы сортировки и поиска*

Сортировка — это преобразование массива неупорядоченных элементов в массив упорядоченных по какому-либо критерию элементов. В простейшем случае элементами массива могут выступать числа, а сортировка производится для расположения чисел по возрастанию.

Подробно основные методы сортировки и поиска излагаются на лекциях (см. В.А. Антонюк, А.П. Иванов. «Программирование и информатика. Краткий конспект лекций.» – Москва, физический ф-т МГУ, 2015, (https://cmp.phys.msu.ru/sites/default/files/Informatics-2015.pdf, главы 4-5).

### «Пузырьковая» сортировка

Одна из простейших для понимания сортировок. Получила свое название из-за того, что на каждой итерации наибольший элемент оказывается на своей позиции в массиве, «всплывая», как пузырек.

1. Последовательно для каждой очередной пары элементов массива (0; 1), (1; 2), (2; 3), …, (N-2; N-1) производится сравнение. Если в паре элементы расположены не по возрастанию, то производится обмен значений элементов пары. В процессе выполнения первого шага будет «зацеплен» максимальный элемент массива, который далее просто «всплывет» до последней позиции массива.
2. Поэтому далее повторяется первый шаг, но при этом каждый раз уменьшается количество рассматриваемых пар элементов на 1, так как крайний правый элемент уже оказался на своей позиции и его не надо более ни с кем сравнивать. Соответственно, второй повтор первого шага должен будет дойти до N-2 элемента массива, третий – до N-3 и так далее.
3. Алгоритм завершается, когда он выполнит единственное сравнение для первой пары элементов массива, то есть, длина неотсортированной части массива сократится до нуля.

Немного быстрее будет работать модифицированная версия этого алгоритма, когда направления «прогона» пузырька меняются после каждой итерации: на первой итерации «всплывает» максимальный элемент массива, на второй – «тонет» минимальный элемент, н третьей – опять всплывает максимальный из оставшихся и так далее. Таким образом неотсортированная часть будет уменьшаться с обоих концов. Такая модификация называется шейкерной сортировкой.

### Сортировка выбором

На каждой итерации выбирается наименьший элемент массива.

1. Производится поиск наименьшего элемента массива. Как только этот элемент найден, производится обмен этого значения с первым неотсортированным элементом. Тем самым, массив оказывается разбит на две части: отсортированную (пока из одного минимального элемента) и неотсортированную (все остальные элементы, начиная со второго).
2. Повторяется первый шаг для оставшихся элементов неотсортированной части: начиная с элемента 2, на следующей итерации – с элемента 3 и так далее.
3. Алгоритм завершается, как только в неотсортированной части останется только один элемент.

### Сортировка вставкой

Массив делится на две части: уже отсортированную (вначале в ней только один первый элемент массива) и еще не отсортированную (все остальные элементы массива).

На каждой итерации очередной элемент неотсортированной части массива вставляется на нужную позицию в отсортированной части.

1. Взять очередной первый элемент из неотсортированной части массива и найти место для его вставки в отсортированной части (последовательным перебором или методом деления отсортированной части пополам): нужно найти первый элемент отсортированной части, который будет меньше выбранного, тогда позицией для вставки будет позиция следующего за ним элемента.
2. Начиная с найденной позиции весь хвост отсортированной части массива сдвинуть на одну позицию вправо (и тем самым затереть выбранный в п.1 элемент).
3. Сохранить выбранный элемент в найденную позицию (затереть то, что там было ранее). Тем самым, отсортированная часть увеличилась на один элемент, а неотсортированная – уменьшилась на один элемент.
4. Повторить шаги 1-3 для оставшихся элементов входного массива.
5. Алгоритм завершается, когда в неотсортированной части массива не останется ни одного элемента.

### Метод ускоренной сортировки Шелла

В качестве примера программного кода алгоритмов сортировки приведем алгоритм сортировки Шелла, гораздо более быстрый по сравнению с изложенными выше методами:

```
#include <stdlib.h>
#define N 100
….
```

```
int A[N];
int i,j;
for (i = 0; i < N; ++i)
      A[i] = rand(); // Подготовим данные: заполним массив случайными числами
// Сортировка Шелла
int gap;
// Будем сортировать подмассивы с элементами на расстоянии gap друг от друга
for (gap = N/2; gap > 0; gap /= 2) // gap стремится к 1
{
      for (i = gap; i < N; ++i)// Переберём все такие подмассивы
      {
// внутренний цикл – это одна итерация сортировки вставками одного подмассива
            for (j = i - gap; j >= 0 && A[j] > A[j + gap]; j -= gap)
            {
                  int temp = A[j];
                  A[j] = A[j + gap];
                  A[j + gap] = temp;
            }
      }
}
```

**Грубый алгоритм поиска строки в тексте**

Алгоритм проверяет на совпадение с искомой строкой фрагменты текста со всеми возможными смещениями.

1. Сравнить первый символ искомой строки с символом в тексте с текущим смещением относительно начала текста.
2. Если символы совпадают, то проверить следующие по порядку символы. Если совпадения нет – увеличить смещение строки в тексте на 1 и перейти на шаг 1.
3. Если все символы искомой строки совпадают с соответствующими символами в тексте, то поиск завершен успешно, иначе – сдвиги строки по тексту нужно продолжать до тех пор, пока «хвост» строки не выйдет за пределы текста, в этом случае завершение поиска безуспешно.

**Алгоритм Рабина-Карпа для поиска строки в тексте**

Использует контрольную сумму при поиске строки в тексте для того, чтобы сократить количество сравнений элементов строки и текста.

1. Вычислить контрольную сумму всех символов искомой строки.
2. Вычислить контрольную сумму символов подстроки текста начиная с заданного смещения и с длиной, равной длине искомой строки.
3. Если вычисленные суммы совпадают, то перейти к посимвольному сравнению строки и участка текста. Если суммы не совпали или посимвольное сравнение неуспешно – перейти к следующему шагу, иначе – поиск завершен успешно.
4. Из контрольной суммы подстроки текста вычесть значение первого символа текста, увеличить смещение подстроки в строке на 1, для этого к контрольной сумме подстроки текста прибавить последний символ подстроки текста после ее смещения.
5. Перейти к шагу 3 и продолжать до тех пор, пока последний символ текущей подстроки текста не совпадет с последним символом всего текста. В этом случае поиск безуспешен.

**Алгоритм Боуера-Мура (упрощенный) для поиска строки в тексте**

Для каждого символа «алфавита» текста и искомой строки вычислим возможное максимально возможное смещение строки по тексту и сохраним их в таблицу:

- Для символов текста, не входящих в искомую строку это смещение будет определяться значением -1;
- Для символов текста, входящих в искомую строку это смещение будет определяться расстоянием от правого конца искомой строки до первого данного символа в строке справа.

Основной алгоритм поиска:

1. Ставим искомую строку в очередную (начиная с первой) позицию текста и начинаем сравнивать ее с текстом, начиная с последнего символа строки и последовательно перебирая символы, двигаясь к началу строки и текста.
2. Когда будет обнаружено несовпадение очередного символа строки и текста – возьмем код символа текста и используем его в качестве индекса таблицы сдвигов. Достанем из таблицы возможный сдвиг по этому индексу и прибавим его к текущей позиции строки. Если же несовпадения обнаружено не будет (мы дойдем до начала строки) – значит, поиск завершился успешно.
3. Повторим шаги 1-2 до тех пор, пока строка не окажется сдвинутой до конца текста (поиск безуспешен).

Пример программного кода:

```
#include <stdio.h>
```

```
#include <stdlib.h>
#include <string.h>
// функция находящая большее из двух чисел
unsigned int Max(unsigned int i, unsigned int j)
{
	if(i > j)
		return i;
	return j;
}
int main()
{
	unsigned int position = 0;  // Искомая позиция строки в тексте
	char T[] = "text with this string: abrakadabra (for example)";
	unsigned  int N = strlen(T); // Длина текста
	char S[] = "abrakadabra";
	unsigned  int M = strlen(S); // Длина искомой строки

	// Построим таблицу сдвигов:
	int SDVIGI[256];
	for (unsigned i = 0; i < 256; ++i)
		SDVIGI[i] = -1;
	// Для символов, не входящих в строку, сдвиг максимально возможный
	for (unsigned i = 0; i < M; ++i)
		SDVIGI[ S[i] ] = i;
	// Для символов, входящих в строку, сдвиг минимально
	// возможный (при повторах символов он будет уменьшен)

	// Алгоритм поиска Боуера-Мура:
	while(position <= (N - M))
	{
		int j = M - 1;
		while(j >= 0 && S[j] == T[position + j])
			j--; // Пока символы совпадают продвигаемся к началу искомой строки
		if (j < 0)
			break; // Мы нашли строку в тексте в данной позиции, прерываем цикл
		else
			position += Max(1, j - SDVIGI[T[position + j]]);
	}
	if (position <= (N - M))
		printf("String %s found at %u offset.\n", S, position+1);
	// Массив индексируется с нуля, поэтому прибавляем единицу,
	// чтобы выдать расстояние от начала текста
	else
		printf("String %s not found.\n", S);
	return 0;

}
```

**Типовое задание на сортировку:** написать и протестировать программу сортировки массива. Массив заполняется случайными числами при помощи функции **`rand()`**, после чего его нужно вывести на экран. Затем массив сортируется с подсчетом количества операций сравнения и перестановок элементов (отдельными переменными-счетчиками) и снова выводится на экран: отсортированный массив, его размерность, число сравнений, число перестановок.

**Типовое задание на поиск подстроки:** «текст» задается случайным целочисленным массивом. Выводим получившийся массив, чтобы его увидел пользователь. У пользователя запрашивается искомая подстрока, затем запрошенное ищется с подсчетом количества операций сравнения элементов и печатается позиция, в которой подстрока нашлась, а также количество операций сравнения элементов, которые были выполнены.

## Варианты задач для решения

**1. Вариант** Реализовать приближение линейной функции **методом наименьших квадратов**.
Значения хранятся в вещественных массивах **`x`** и **`y`**, размер массивов задаётся с помощью **`#define`**.
Пользователь вводит значения из консоли. Вывести параметры прямой, являющейся наилучшим приближением к заданным точкам на плоскости.

**2. Вариант**
Сортировка массива действительных чисел методом вставки в порядке возрастания.
Массив заполняется случайными значениями с помощью функции rand(). Размер массива задаётся с помощью #define. Необходимо вывести количество перестановок и сравнений, а также входной и выходной массивы.

**3. Вариант**
Сортировка массива действительных чисел **методом вставки** в порядке убывания.
Массив заполняется случайными значениями с помощью функции rand(). Размер массива задаётся с помощью #define. Необходимо вывести количество перестановок и сравнений, а также входной и выходной массивы.

**4. Вариант**
Сортировка массива действительных чисел **методом выбора** в порядке возрастания.
Массив заполняется случайными значениями с помощью функции **`rand()`**. Размер массива задаётся с помощью **`#define`**. Необходимо вывести количество перестановок и сравнений, а также входной и выходной массивы.

**5. Вариант**
Сортировка массива действительных чисел **методом шейкера** в порядке убывания.
Массив заполняется случайными значениями с помощью функции **`rand()`**. Размер массива задаётся с помощью **`#define`**. Необходимо вывести количество перестановок и сравнений, а также входной и выходной массивы.

**6. Вариант**
Сортировка массива действительных чисел **методом выбора** в порядке убывания.
Массив заполняется случайными значениями с помощью функции **`rand()`**. Размер массива задаётся с помощью **`#define`**. Необходимо вывести количество перестановок и сравнений, а также входной и выходной массивы.

**7. Вариант**
Сортировка массива действительных чисел **методом пузырька** в порядке возрастания.
Массив заполняется случайными значениями с помощью функции **`rand()`**. Размер массива задаётся с помощью **`#define`**. Необходимо вывести количество перестановок и сравнений, а также входной и выходной массивы.

**8. Вариант**
Сортировка массива действительных чисел **методом пузырька** в порядке убывания.
Массив заполняется случайными значениями с помощью функции **`rand()`**. Размер массива задаётся с помощью **`#define`**. Необходимо вывести количество перестановок и сравнений, а также входной и выходной массивы.

**9. Вариант**
Выполнить транспонирование матрицы.
Размеры матрицы задаются с помощью **`#define`**. Матрица заполняется случайными значениями с помощью функции **`rand()`**. Вывести исходную и транспонированную матрицы, их размерности.

**10. Вариант**
Грубый алгоритм поиска **первого вхождения подмассива** в массиве.
Массив заполняется случайными значениями с помощью функции **`rand()`**. Размер массива и подмассива задается с помощью **`#define`**. Необходимо вывести количество сравнений, позицию начала вхождения подмассива, сами массив и подмассив.

**11. Вариант**
Грубый алгоритм поиска **последнего вхождения подмассива** в массиве.
Массив заполняется случайными значениями с помощью функции **`rand()`**. Размер массива и подмассива задается с помощью **`#define`**. Необходимо вывести количество сравнений, позицию начала вхождения подмассива, сами массив и подмассив.

**12. Вариант**

Поиск всех вхождений подмассива в массиве методом Рабина-Карпа.
Массив заполняется случайными значениями с помощью функции **`rand()`**. Размер массива и подмассива задается с помощью **`#define`**. Необходимо вывести количество сравнений, позицию начала вхождений подмассива, сами массив и подмассив.

**13. Вариант**

Грубый алгоритм поиска **первого вхождения подмассива** в массиве с одной ошибкой (то есть один элемент найденного вхождения имеет право отличаться от искомого массива).
Массив заполняется случайными значениями с помощью функции **`rand()`**. Размер массива и подмассива задается с помощью **`#define`**. Необходимо вывести количество сравнений, позицию начала вхождения подмассива, сам массив и искомый подмассив.

**14. Вариант**

Грубый алгоритм поиска **первого вхождения подмассива** в массиве с заданным пользователем числом ошибок (то есть указанное количество элементов найденного вхождения имеет право отличаться от искомого массива).
Массив заполняется случайными значениями с помощью функции **`rand()`**. Размер массива и подмассива задаётся с помощью **`#define`**. Необходимо вывести количество сравнений, позицию начала вхождения подмассива, сам массив и искомый подмассив.

**15. Вариант**

Грубый алгоритм поиска **последнего вхождения подмассива** в массиве с одной ошибкой (то есть один элемент найденного вхождения имеет право отличаться от искомого массива).
Массив заполняется случайными значениями с помощью функции **`rand()`**. Размер массива и подмассива задается с помощью **`#define`**. Необходимо вывести количество сравнений, позицию начала вхождения подмассива, сам массив и искомый подмассив.

**16. Вариант**

Поиск первого вхождения подмассива в массиве методом Боуера-Мура.
Массив заполняется случайными значениями с помощью функции **`rand()`**. Размер массива и подмассива задаётся с помощью **`#define`**. Необходимо вывести количество сравнений, позицию начала вхождения подмассива, сам массив и искомый подмассив.

**17. Вариант**

Поиск всех вхождений подмассива в массиве методом Боуера-Мура.
Массив заполняется случайными значениями с помощью функции **`rand()`**. Размер массива и подмассива задаётся с помощью **`#define`**. Необходимо вывести количество сравнений, позиции всех вхождений подмассива, сам массив и искомый подмассив.

**18. Вариант**

Построить **линейную интерполяцию или экстраполяцию** по заданной таблице значений функции. Даны два вещественных массива X и Y, значения в массиве X монотонно возрастают. У пользователя запрашивается значение x, для которого надо найти соответствующий интервал в массиве X: если введенное x значение меньше минимального значения в массиве X или больше максимального – то надо выполнить нахождение значения y для этого x с помощью линейной экстраполяции крайней пары точек. В противном случае нужно найти интервал, в который попадает введенное значение x, после чего нужно найти значение y для этого x с помощью линейной интерполяции.
Вывести исходные массивы, введенное значение x и найденное значение y.

**19. Вариант**

Вывести **строки матрицы** в порядке возрастания суммы их элементов.
Для этого завести отдельный массив сумм элементов по строкам, искать каждый раз минимальный элемент из оставшихся и выводить соответствующую строку матрицы. Матрица заполняется случайными значениями с помощью функции **`rand()`**. Размеры матрицы задаются с помощью **`#define`**. Вывести исходную матрицу, далее преобразованную матрицу, где строки должны идти в указанном порядке (то есть, необходимо поменять местами строки исходной матрицы).

**20. Вариант**

Вывести **строки матрицы** в порядке убывания суммы их элементов.
Для этого завести отдельный массив сумм элементов по строкам, искать каждый раз максимальный элемент из оставшихся и выводить соответствующую строку матрицы. Матрица заполняется случайными значениями с помощью функции **`rand()`**. Размеры матрицы задаются с помощью **`#define`**. Вывести исходную матрицу, далее преобразованную матрицу, где строки должны идти в указанном порядке (то есть, необходимо поменять местами строки исходной матрицы).

**21. Вариант**

Подсчитать **среднее значение и дисперсию** в каждом столбце матрицы.
Матрица заполняется случайными значениями с помощью функции **`rand()`**. Размеры матрицы задаются с помощью **`#define`**.

**22. Вариант**

Реализовать **сложение** трех десятичных чисел в столбик. Числа представлены массивами десятичных цифр и запрашиваются у пользователя.
Размеры массивов, представляющих слагаемые, задаются с помощью **`#define`**. Выводить слагаемые и результат. Учтите, что результат может быть длиннее каждого из слагаемых.

**23. Вариант**

Реализовать **вычитание** двух шестнадцатеричных чисел в столбик. Числа представлены массивами шестнадцатеричных цифр и запрашиваются у пользователя.
Размеры массивов, представляющих числа, задаются с помощью **`#define`**. Выводить слагаемые и результат.

**24. Вариант**

Построить и вывести **вертикальную гистограмму** (количество элементов каждого значения) значений целочисленного массива. Каждая строка гистограммы содержит столько символов «**`*`**», сколько раз встретилось уникальное значение входного массива, соответствующее данной строке гистограммы. Строки гистограммы должны выводиться от меньшего значения исходного массива к большему.
Размеры исходного массива и количество разных значений в гистограмме задать через **`#define`**.

**25. Вариант**

Построить **график функции**, фигурировавшей в уравнении, которое решалось на предыдущей задаче практикума, в окрестности найденного корня уравнения. График должен изображаться в виде матрицы символов, в которой значения функции помечаются символом «**`*`**», а также помечаются оси координат, прочие символы должны быть пробелами. Необходимо сделать преобразование масштаба, чтобы окрестность корня изображалась максимально информативно.
Размерность матрицы задается с помощью **`#define`** (не более 80 столбцов и 25 строк).

**26. Вариант**

Сортировка массива действительных чисел **методом Шелла** в порядке убывания.
Массив заполняется случайными значениями с помощью функции **`rand()`**. Размер массива задаётся с помощью **`#define`**. Необходимо вывести количество перестановок и сравнений, а также входной и выходной массивы. Если останется время – сравнить количество перестановок и сравнений с методом сортировки вставками.

**27. Вариант**

Написать программу, которая объединяет два упорядоченных по возрастанию массива в один, также упорядоченный массив и проверяет, есть ли в выходном массиве пара соседних элементов, сумма которых равна заданному числу.
Размеры массивов задаются с помощью **`#define`**. Нужно распечатать исходные массивы и результат работы программы.

**28. Вариант**

Написать программу, удаляющую из вещественного массива все элементы, которые отклоняются от среднего значения более, чем на три стандартных отклонения (правило трех сигма).
Размер исходного массива задаётся с помощью **`#define`**. Результат сохраняется в этом же массиве, его уменьшенная длина запоминается в переменной N. Нужно распечатать исходный массив, среднее значение, стандартное отклонение и получившийся прореженный массив.

**29. Вариант**

Написать программу, которая приводит заданную квадратную матрицу к треугольному виду по методу Гаусса (домножение строк на множитель и сложение двух соседних строк с заменой второй строки на результат сложения).
Матрица заполняется случайными значениями с помощью функции **`rand()`**. Размеры матрицы задаются с помощью **`#define`**.

**30. Вариант**

В исходном случайном целочисленном массиве оставить только уникальные значения. Для эффективного решения этой задачи его следует отсортировать любым способом, после чего просканировать для удаления рядом стоящих повторов и подсчета количества этих повторов.
Размер исходного массива задаётся с помощью **`#define`**. Результат сохраняется в этом же массиве, его уменьшенная длина запоминается в переменной N. Нужно распечатать исходный массив, и результат работы программы: для каждого уникального значения выходного массива распечатать количество его повторов.