

# Массивы

При создании использовались материалы К. Полякова  
<http://kpolyakov.narod.ru/>

# Что это?

Массив – упорядоченная последовательность данных одного типа, объединённых одним именем.

## Описание:

ИМЯ

начальный  
индекс

конечный  
индекс

ТИП  
элементов

```
var A: array[1 .. 5] of integer;
```

## Размер через константу

```
const N=5;  
var A: array[1..N] of integer;
```

# Как устроен?



# Действия с массивами

Единственная операция, возможная с массивом — присваивание, но только в случае, когда размерность массивов и тип элементов совпадают.

Var

a,b: array [1..10] of real;

c: array [1..10] of char;

d:array [1..2, 1..5] of real;

Begin

**a:=b;**

~~a:=c;~~ ~~a:=d;~~

Все остальные действия выполняются с элементами массива и зависят от типа данных элементов.

# Заполнение и обработка массивов

Для работы с массивами используется цикл с параметром.

## **Задание:**

```
... Begin
```

```
for i:=1 to N do {ввод данных}
```

```
begin
```

```
  writeln('Введите элемент');
```

```
  readln(a[i]);
```

```
end;
```

```
for i:=1 to N do {вывод данных}
```

```
  write(a[i]:4);
```

# Генерация случайных величин



**Псевдослучайные числа** – обладают свойствами случайных чисел, но каждое следующее число вычисляется по заданной формуле.

**Ф random;** - генерирует вещественное число в диапазоне от 0 до 1.

**Ф random(x);** - генерирует целое число в диапазоне от 0 до x (x – целое число).

Пример:

Begin

```
y:=random(1000);
```



# Многомерные массивы

Многомерные массивы – массивы, размерность которых больше либо равна двум.

Для обработки чаще всего используются вложенные циклы с параметром. Двумерные массивы часто называют **матрицей**.

Var

```
mas: array [1..5,1..10] of integer;
```

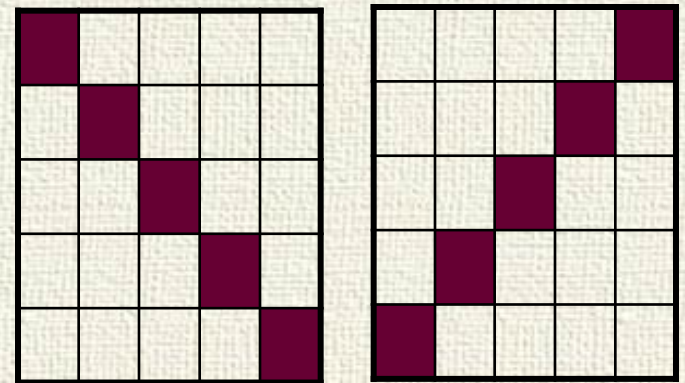
Begin

```
for i:=1 to 5 do {ввод данных}
```

```
for j:=1 to 10 do
```

```
mas[i,j]:=random(100);
```

*Главная и побочная  
диагонали*



# Сортировка элементов массива



# Сортировка

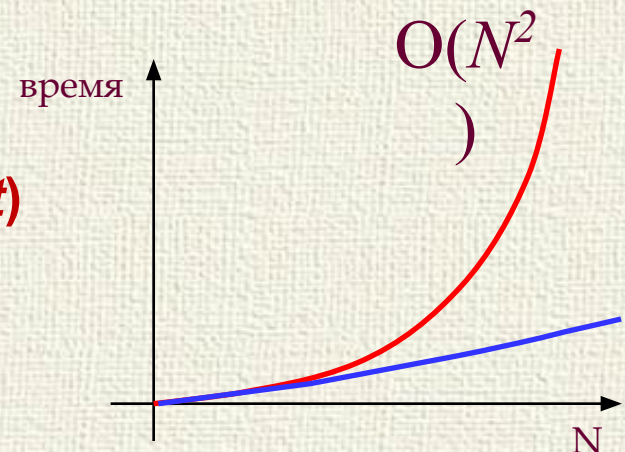
**Сортировка** – это расстановка элементов массива в заданном порядке (по возрастанию, убыванию, последней цифре, сумме делителей, ...).

**Задача:** переставить элементы массива в порядке возрастания.

СЛОЖНОСТЬ  $O(N^2)$

**Алгоритмы:**

- простые и понятные, но неэффективные для больших массивов
  - метод пузырька
  - метод выбора
- сложные, но эффективные
  - «быстрая сортировка» (*Quick Sort*)
  - сортировка «кучей» (*Heap Sort*)
  - сортировка слиянием
  - пирамидальная сортировка



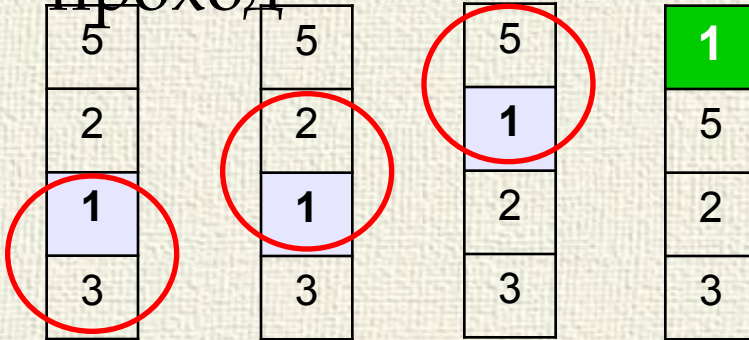
# Метод пузырька

**Идея** – пузырек воздуха в стакане воды поднимается со дна вверх.

**Для массивов** – самый маленький («легкий») элемент перемещается вверх («всплывает»).

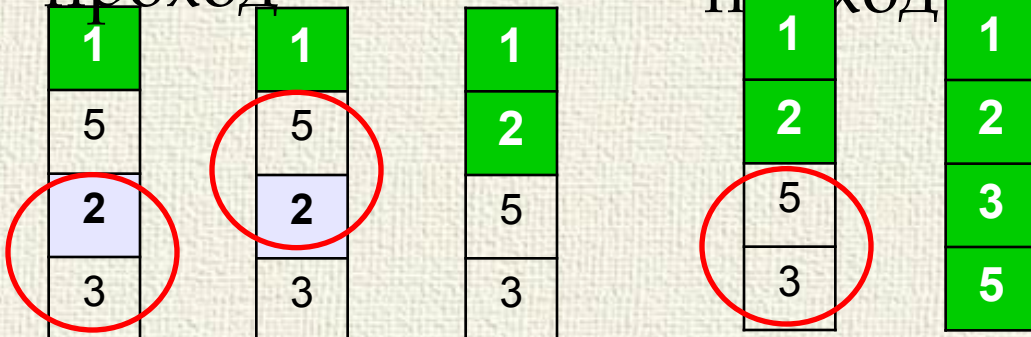
1-ый

проход



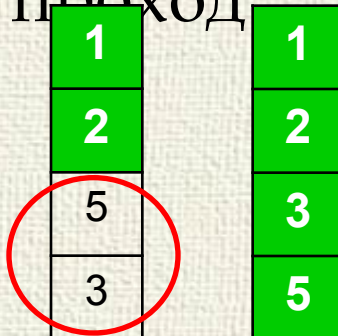
2-ой

проход



3-ий

проход



- начиная снизу, сравниваем два соседних элемента; если они стоят «неправильно», меняем их местами
- за 1 проход по массиву **один** элемент (самый маленький) становится на свое место

Для сортировки массива из N элементов нужен N-1 проход (достаточно поставить на свои места N-1 элементов).

# Программа



сравниваются пары

$A[N-1]$  и  $A[N]$ ,  $A[N-2]$  и  $A[N-1]$   
...  
 $A[1]$  и  $A[2]$

$A[j]$  и  $A[j+1]$

```
for j:=N-1 downto 1 do
  if A[j] > A[j+1] then begin
    c:=A[j]; A[j]:=A[j+1]; A[j+1]:=c;
  end;
```

**!**  $A[1]$  уже на своем месте!

```
for j:=N-1 downto 2 do
  if A[j] > A[j+1] then begin
    c:=A[j]; A[j]:=A[j+1]; A[j+1]:=c;
  end;
```

```
for j:=N-1 downto i do
  ...
```

# Программа

```
program qq;  
const N = 10;  
var A: array[1..N] of integer;  
    i, j, c: integer;  
begin  
    { заполнить массив }  
    { вывести исходный массив }  
    for i:=1 to N-1 do begin  
        for j:=N-1 downto i do  
            if A[j] > A[j+1] then begin  
                c := A[j];  
                A[j] := A[j+1];  
                A[j+1] := c;  
            end;  
        end;  
    end;  
    { вывести полученный массив }  
end.
```

?

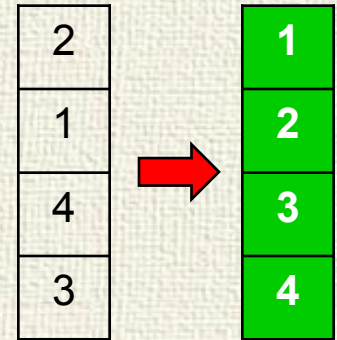
Почему цикл по  $i$  до  $N-1$ ?

элементы выше  $A[i]$   
уже поставлены

# Метод пузырька с флажком

**Идея** – если при выполнении метода пузырька не было обменов, массив уже отсортирован и остальные проходы не нужны.

**Реализация:** переменная-флаг, показывающая, был ли обмен; если она равна **False**, то выход.



```
var flag: boolean;
```

```
repeat
  flag := False; { сбросить флаг }
  for j:=N-1 downto 1 do
    if A[j] > A[j+1] then begin
      c := A[j];
      A[j] := A[j+1];
      A[j+1] := c;
      flag := True; { поднять флаг }
    end;
until not flag; { выход при flag=False }
```



Как улучшить?

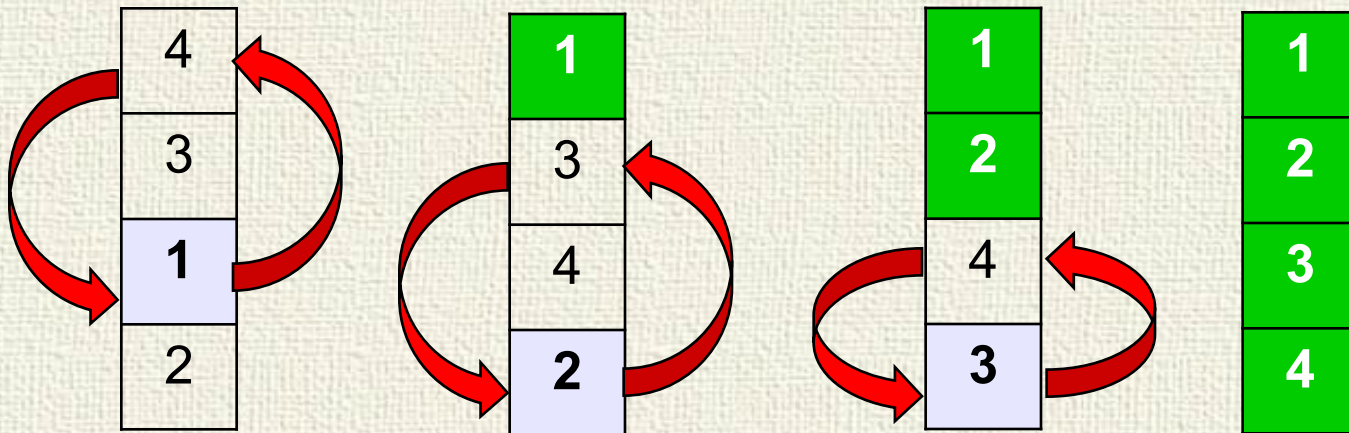
# Метод пузырька с флажком

```
i :=  
0;  
repeat  
  i := i +  
  1;  
  flag := False; { сбросить флаг }  
  for j:=N-1 downto i do  
    if A[j] > A[j+1] then begin  
      c := A[j];  
      A[j] := A[j+1];  
      A[j+1] := c;  
      flag := True; { поднять флаг }  
    end;  
until not flag; { выход при flag=False }
```

# Метод выбора

## Идея:

- найти минимальный элемент и поставить на первое место (поменять местами с  $A[1]$ )
- **из оставшихся** найти минимальный элемент и поставить на второе место (поменять местами с  $A[2]$ ), и т.д.



# Метод выбора

нужно  $N-1$  проходов

```
for i := 1 to N-1 do begin
```

```
  nMin := i;
```

```
  for j := i+1 to N do
```

```
    if A[j] < A[nMin] then nMin := j;
```

```
  if nMin <> i then begin
```

```
    c := A[i];
```

```
    A[i] := A[nMin];
```

```
    A[nMin] := c;
```

```
  end;
```

```
end;
```

ПОИСК МИНИМАЛЬНОГО  
ОТ A[i] ДО A[N]

если нужно,  
переставляем



Можно ли убрать **if**?



# **Эффективные методы сортировки**

# «Быстрая сортировка» (*Quick Sort*)

**Идея** – более эффективно переставлять элементы, расположенные дальше друг от друга.

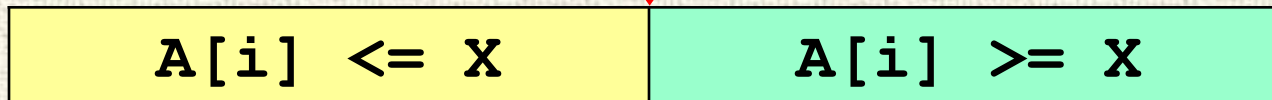


Сколько перестановок нужно, если массив отсортирован по убыванию, а надо – по возрастанию?

$$N \div 2$$

**1 шаг:** выбрать некоторый элемент массива  $X$

**2 шаг:** переставить элементы так:



при сортировке элементы не покидают « свою область »!

**3 шаг:** так же отсортировать две получившиеся области

Разделяй и властвуй (англ. *divide and conquer*)

# «Быстрая сортировка» (*Quick Sort*)

78	6	82	67	55	44	34
----	---	----	----	----	----	----



Как лучше выбрать **X**?

**Медиана** – такое значение  $X$ , что слева и справа от него в отсортированном массиве стоит одинаковое число элементов (*для этого надо отсортировать массив...*).

## Разделение:

1) выбрать средний элемент массива ( $x=67$ )

78	6	82	67	55	44	34
----	---	----	----	----	----	----

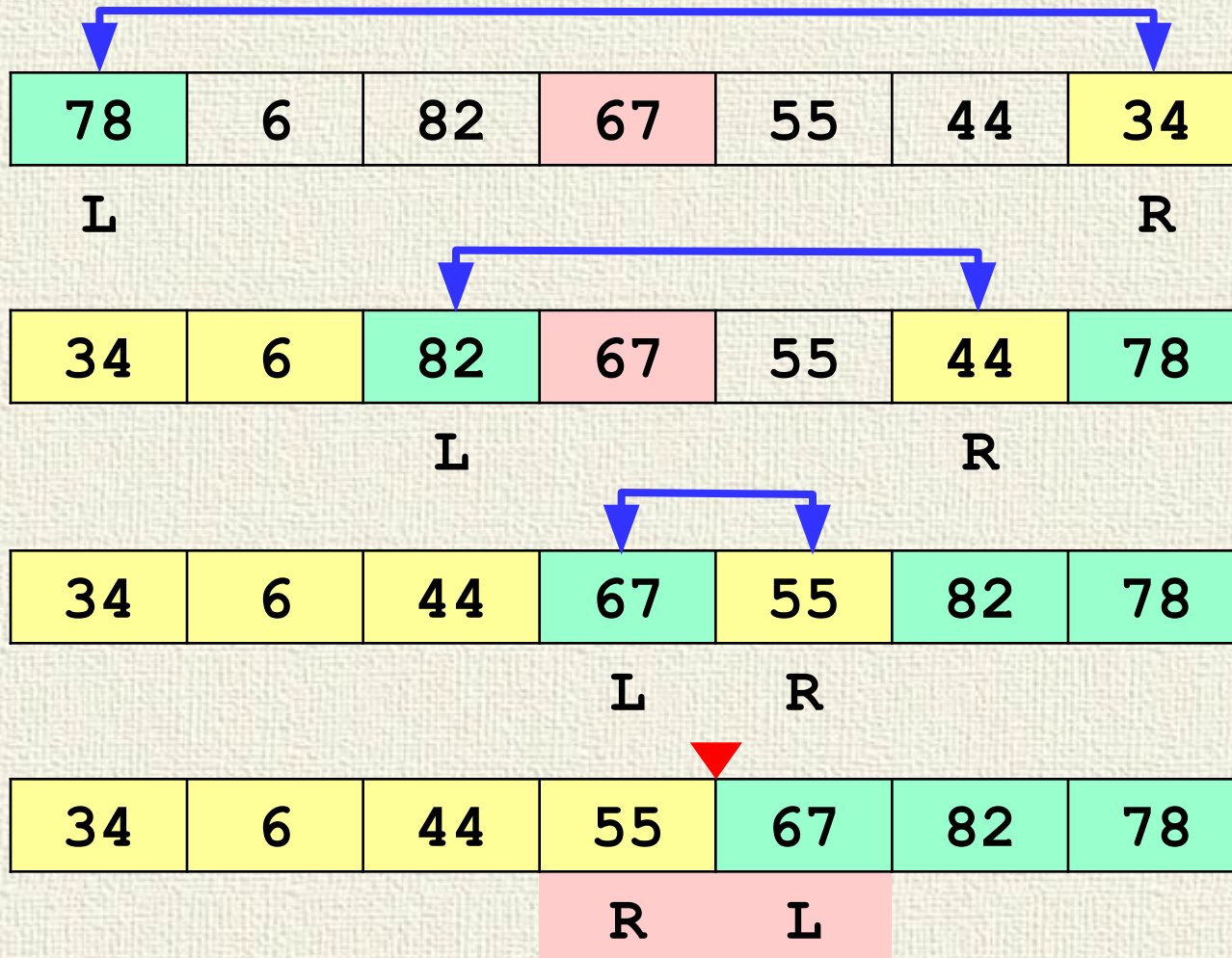
2) установить  $L := 1$ ,  $R := N$


3) увеличивая  $L$ , найти первый элемент  $A[L]$ , который  $\geq X$   
(должен стоять справа)

4) уменьшая  $R$ , найти первый элемент  $A[R]$ , который  $\leq X$   
(должен стоять слева)

5) если  $L \leq R$ , поменять местами  $A[L]$  и  $A[R]$  и перейти к п. 3

# «Быстрая сортировка» (*Quick Sort*)



 **L > R : разделение закончено**

# «Быстрая сортировка» (*Quick Sort*)

```
procedure QSort ( first, last: integer);  
var L, R, c, X: integer;  
begin  
  if first < last then begin  
    X:=A[(first+last) div 2];  
    L:=first; R:=last;  
  
    while L <= R do begin  
      while A[L] < X do L:=L+1;  
      while A[R] > X do R:=R-1;  
      if L <= R then begin  
        c:=A[L]; A[L]:=A[R]; A[R]:=c;  
        L:=L+1; R:=R-1;  
      end;  
    end;  
  end;  
  QSort(first, R);  QSort(L, last);  
end;  
end.
```

ограничение рекурсии

разделение

обмен

двигаемся дальше

сортируем две части

# «Быстрая сортировка» (*Quick Sort*)

```
program qq;  
const N = 10;  
var A: array[1..N] of integer;  
procedure QSort ( first, last: integer);  
...  
begin  
    { заполнить массив }  
    { вывести исходный массив на экран }  
    Qsort ( 1, N ); { сортировка }  
    { вывести результат }  
end.
```



Сложность (в среднем)  $O(N \log N)$  !

# Количество перестановок (случайные данные)

$N$	<i>QuickSort</i> $O(N \log N)$	«пузырек» $O(N^2)$
10	11	24
100	184	2263
200	426	9055
500	1346	63529
1000	3074	248547