ФГБОУ ВПО ЧГУ ИМ. И.Н. УЛЬЯНОВА
ФАКУЛЬТЕТ РАДИОЭЛЕКТРОНИКИ И АВТОМАТИКИ
КАФЕДРА АВТОМАТИКИ И УПРАВЛЕНИЯ В ТЕХНИЧЕСКИХ СИСТЕМАХ

# МАССИВЫ В С++

## Лекция 2.3.

к.п.н., доцент Васильева Л.Н.

# Определение

**Массив** – это группа переменных одного типа, расположенных в памяти рядом (в соседних ячейках) и имеющих общее имя. Каждая ячейка в массиве имеет уникальный номер (индекс).

**Надо**:

- выделять память
- записывать данные в нужную ячейку
- читать данные из ячейки

Резервирование памяти для массива выполняется на этапе компиляции программы.

# Выделение памяти (объявление)

тип имя_массива[размер];

```
int A[5];
double V[8];
bool L[10];
char S[80];
```

число элементов

Элементы нумеруются с нуля!

A[0], A[1], A[2], A[3], A[4]

размер через константу

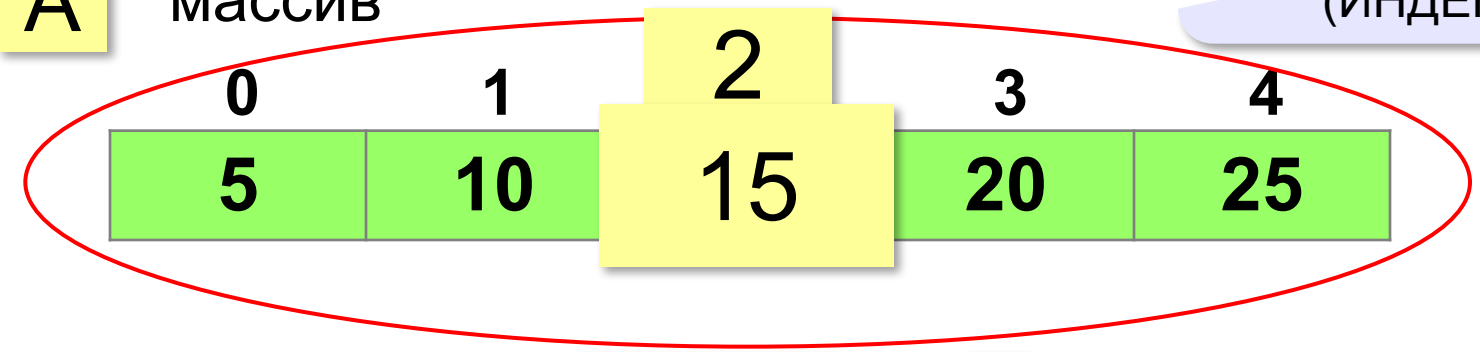
```
const int N=10;
int A[N];
```

Зачем?

# Обращение к элементу массива

НОМЕР
элемента массива
(ИНДЕКС)

A массив




A[0] A[1] ЗНАЧЕНИЕ элемента массива ] A[4]

НОМЕР (ИНДЕКС)
элемента массива: 2

A[2]

ЗНАЧЕНИЕ
элемента массива: 15

# Как обработать все элементы массива?

**Объявление**:

```
const int N = 5;
int A[N];
```

**Обработка**:

```
// обработать A[0]
// обработать A[1]
// обработать A[2]
// обработать A[3]
// обработать A[4]
```

1) если N велико (1000, 1000000)?

2) при изменении N программа не должна меняться!

# Как обработать все элементы массива?

**Обработка с переменной:**

```
i = 0;
// обработать A[i]
i ++;
// обработать A[i]
i ++;
// обработать A[i]
i ++;
// обработать A[i]
i ++;
// обработать A[i]
```

```
i ++;
```

**Обработка в цикле:**

```
i = 0;
while ( i < N )
  {
  // обработать A[i]
  i ++;
  }
```

**Цикл с переменной:**

```
for( i = 0; i < N; i++ )
  {
  // обработать A[i]
  }
```

# Заполнение массива

```
main()
{
   const int N=10;
   int A[N];
   int i;
   for ( i=0; i<N; i++ )
     A[i]=i*i;
}
```

Чему равен **A[9]**?

# Ввод с клавиатуры и вывод на экран

**Объявление:**

```
const int N=10;
int A[N];
```

**Ввод с клавиатуры:**

```
for ( i=0; i<N; i++ )
   {
   cout << "A[" << i << "]=";
   cin >> A[i];
   }
```

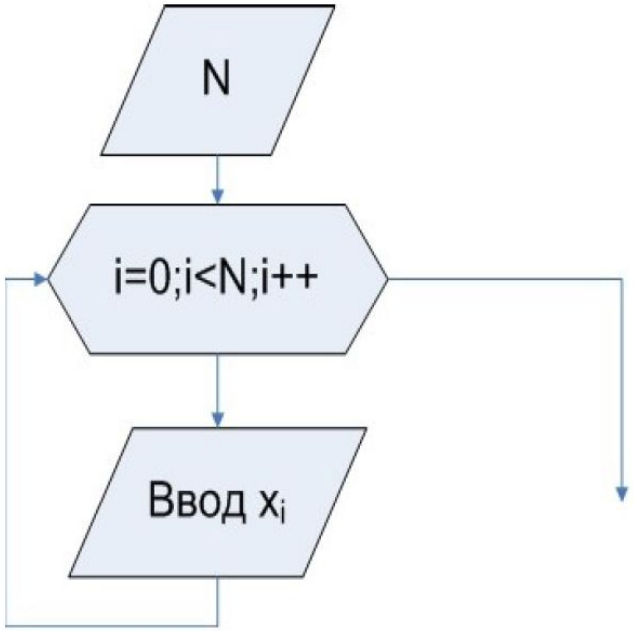
```
A[1]=5
A[2]=12
A[3]=34
A[4]=56
A[5]=13
```

**Вывод на экран:**

```
cout << "Массив A:\n";
for ( i=0; i<N; i++ )
   cout << A[i] << " ";
```

? Зачем пробел?

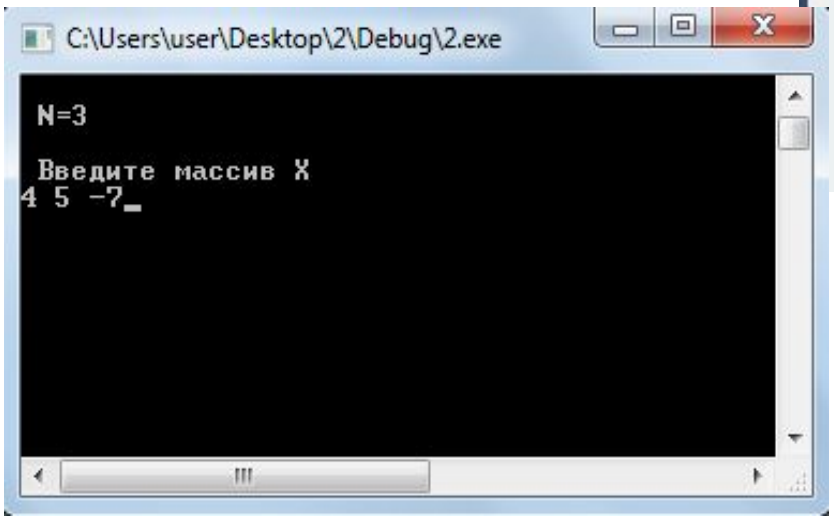
# Ввод с клавиатуры



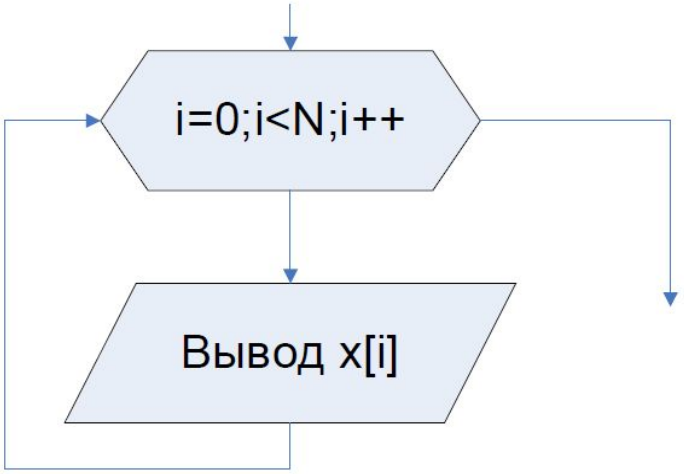

2.cpp

(Глобальная область)

```
#include "stdafx.h"
#include <conio.h>
#include <string>
int _tmain(int argc, _TCHAR* argv[])
{
setlocale(LC_ALL, "rus");
float x[10];
int i,n;
printf("\n N=");
scanf("%d",&n);
printf("\n Введите массив X \n");
for(i=0;i<n;i++)
scanf("%f",&x[i]);
getch();
return 0;
}
```



# Вывод на экран




```
2.cpp
(Глобальная область)
#include "stdafx.h"
#include <conio.h>
#include <string>
#include <iostream>
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{
setlocale(LC_ALL, "rus");
float x[10];
int i,n;
printf("\n N=");
scanf("%d",&n);
printf("\n Введите массив X \n");
for(i=0;i<n;i++)
scanf("%f",&x[i]);
printf("\n Массив X\n\n");
for(i=0;i<n;i++)
printf("%g\t",x[i]);
printf("\n");
getch();
return 0;
}
```

```
C:\Users\user\Desktop\2\Debug\2.exe
 N=5
 Введите массив X
1
2
-9
0
6
 Массив X
1        2        -9       0        6
```

# Заполнение случайными числами

*Задача*. Заполнить массив (псевдо)случайными целыми числами в диапазоне от 20 до 100.

```
for ( i = 0; i < N; i++ )
   {
   A[i] = rand()%81+20;
   cout << A[i] << " ";
   }
```

# Перебор элементов

**Общая схема:**

```
for ( i = 0; i < N; i++ )
   {
   ... // сделать что-то с A[i]
   }
```

**Подсчёт нужных элементов:**

*Задача*. В массиве записаны данные о росте баскетболистов. Сколько из них имеет рост больше 180 см, но меньше 190 см?

```
count = 0;
for ( i = 0; i < N; i++ )
    if ( 180 < A[i] && A[i] < 190 )
       count ++;
```

# Перебор элементов

## Среднее арифметическое:

```
int count, sum;
count = 0;
sum = 0;
for ( i = 0; i < N; i++ )
  if ( 180 < A[i] && A[i] < 190 ) {
    count ++;
    sum += A[i];
    }
cout << (float)sum / count;
```

Зачем `float`?

среднее арифметическое

# АЛГОРИТМЫ ОБРАБОТКИ МАССИВОВ

# Поиск в массиве

## Найти элемент, равный X:

```
i = 0;
while ( A[i] != X )
   i ++;
cout << "A[" << i << "]=" << X;
```

? Что плохо?

```
i = 0;
while ( i < N  &&  A[i] != X )
   i ++;
if ( i < N )
   cout << "A[" << i << "]=" << X;
else
   cout << "Не нашли!";
```

? Что если такого нет?

# Поиск в массиве

## Вариант с досрочным выходом:

```
nX = -1;
for ( i = 0; i < N; i++ )
   if ( A[i] == X )
      {
      nX = i;
      break;
      }
if ( nX >= 0 )
   cout << "A[" << nX << "]=" << X;
else
   cout << "Не нашли!";
```

досрочный выход из цикла

# Максимальный элемент

```
M=A[0];
for ( i=1; i<N; i++ )
  if ( A[i]>M )
    M=A[i];
cout << M;
```

Как найти его номер?

```
M=A[0]; nMax=0;
for ( i=1; i<N; i++ )
  if ( A[i]>M ) {
    M=A[i];
    nMax=i;
    }
cout << "A[" << nMax << "]=" << M;
```

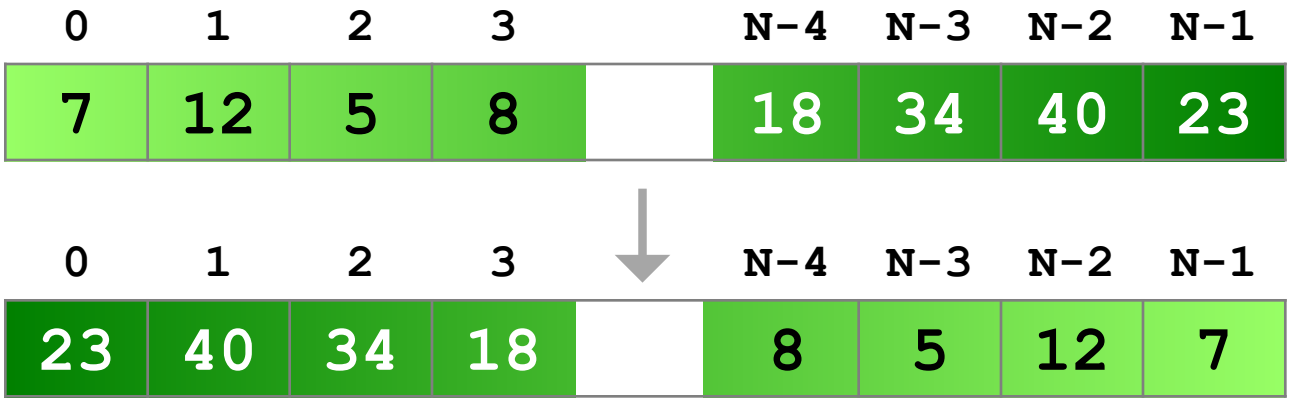
Что можно улучшить?

# Максимальный элемент и его номер

По номеру элемента можно найти значение!

```
nMax = 0;
for ( i = 1; i < N; i++ )
  if ( A[i] > A[nMax] )
    nMax = i;
cout << "A[" << nMax << "]=" << A[nMax];
```

# Реверс массива




**«Простое» решение:**

остановиться на середине!

```
for( i = 0; i < N/2 ; i++ )
   {
   // поменять местами A[i] и A[N+1-i]
   }
```

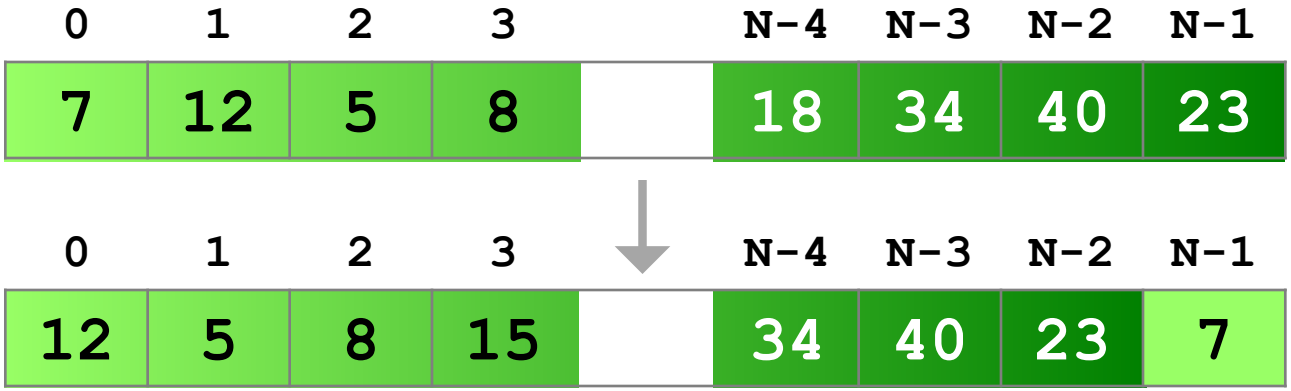
? Что плохо?

# Реверс массива

```
for ( i=0; i<(N/2); i++ )
   {
   c=A[i];
   A[i]=A[N-1-i];
   A[N-1-i]=c;
   }
```

*Как обойтись без переменной `c`?

# Циклический сдвиг элементов




**«Простое» решение:**

```
for ( i = 0; i < N-1; i++ )
  A[i] = A[i+1];
```

? Почему не до **N-1**?

? Что плохо?

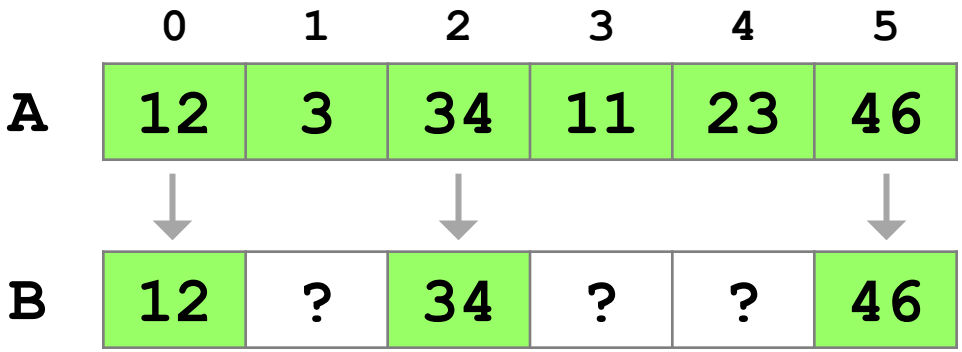
# Отбор нужных элементов

*Задача*. Отобрать элементы массива **A**, удовлетворяющие некоторому условию, в массив **B**.

**«Простое» решение:**

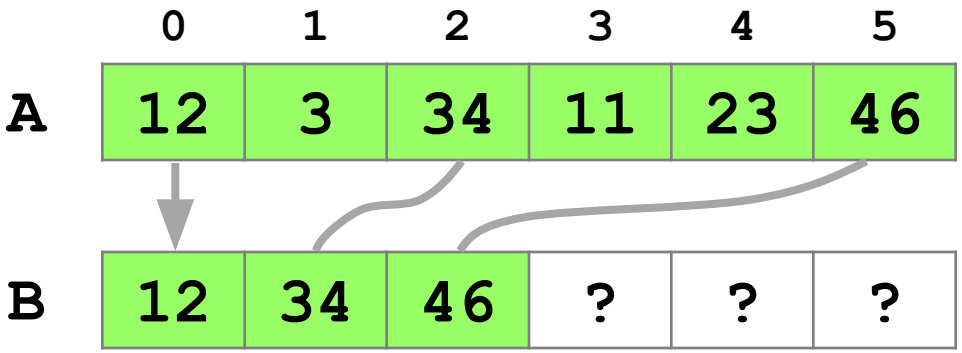
```
сделать для i от 0 до N-1
  если условие выполняется для A[i] то
    B[i]=A[i]
```

? Что плохо?




выбрать чётные элементы

# Отбор нужных элементов




выбрать чётные элементы

```
count = 0;
for ( i = 0; i < N; i++ )
  if ( A[i] % 2 == 0 )
    {
    B[count] = A[i];
    count++;
    }
```

? Как вывести на экран?

```
for ( i = 0; i < count ; i++ )
  printf ( "%d ", B[i] );
```

# СОРТИРОВКА

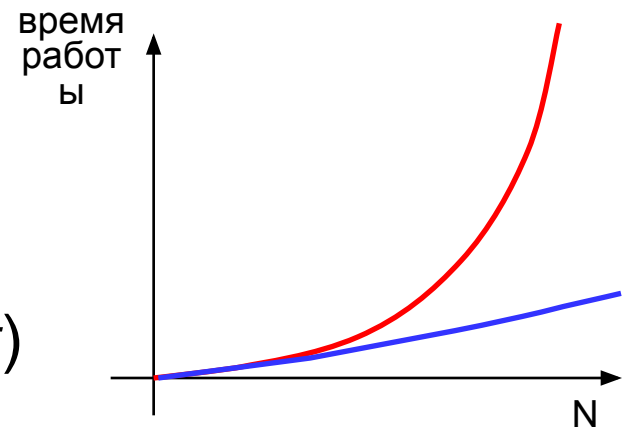
# Что такое сортировка?

**Сортировка** – это расстановка элементов массива в заданном порядке.

…по возрастанию, убыванию, последней цифре, сумме делителей, по алфавиту, …

**Алгоритмы:**

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

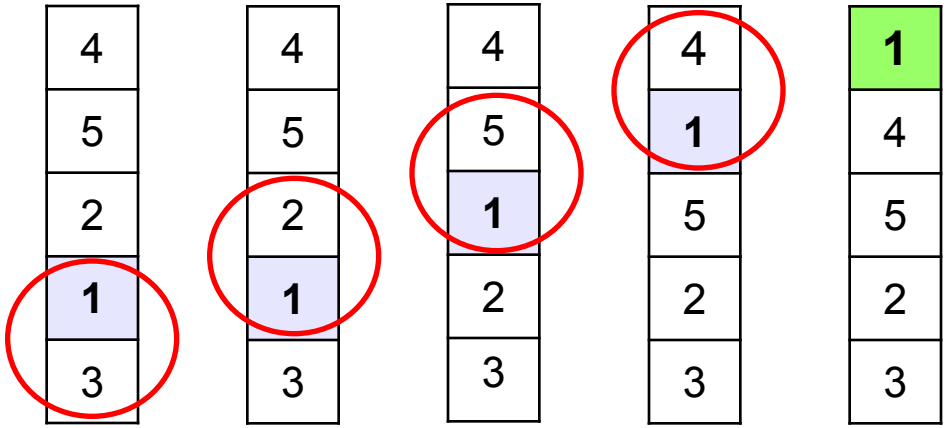


# Метод пузырька (сортировка обменами)

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

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

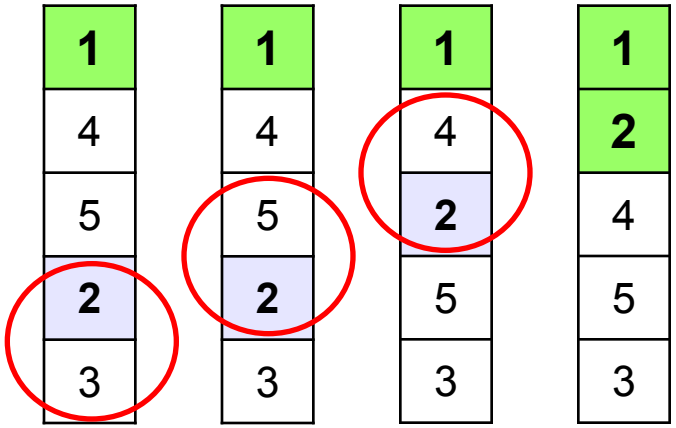
**1-й проход:**



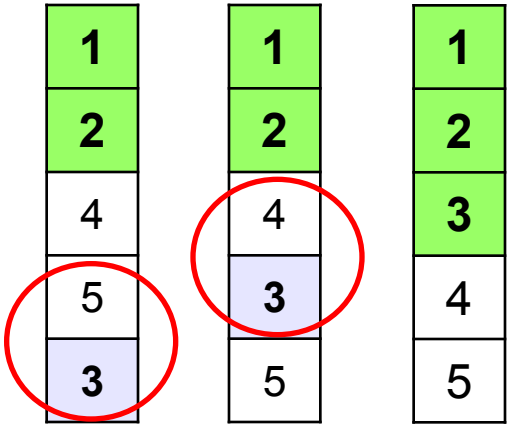

- сравниваем два соседних элемента; если они стоят «неправильно», меняем их местами
- за 1 проход по массиву один элемент (самый маленький) становится на свое место

# Метод пузырька

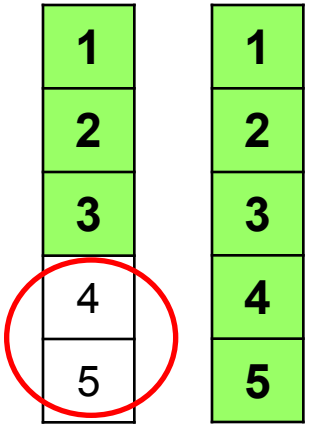
**2-й проход:**




**3-й проход:**




**4-й проход:**




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

# Метод пузырька

**1-й проход:**

```
сделать для j от N-2 до 0 шаг -1
  если A[j+1]<A[j] то
    // поменять местами A[j] и A[j+1]
```

единственное отличие!

**2-й проход:**

```
сделать для j от N-2 до 1 шаг -1
  если A[j+1]<A[j] то
    // поменять местами A[j] и A[j+1]
```

# Метод пузырька

```
for ( i = 0; i < N-1; i++ )
  for ( j = N-2; j >= i ; j-- )
    if ( A[j] > A[j+1] )
      {
      // поменять местами A[j] и A[j+1]
      }
```

# Метод выбора (минимального элемента)

*Идея*: найти минимальный элемент и поставить его на первое место.

```
сделать для i от 0 до N-2
  // найти номер nMin минимального
  // элемента из A[i]..A[N]
  если i != nMin то
    // поменять местами A[i] и A[nMin]
```

# Метод выбора (минимального элемента)

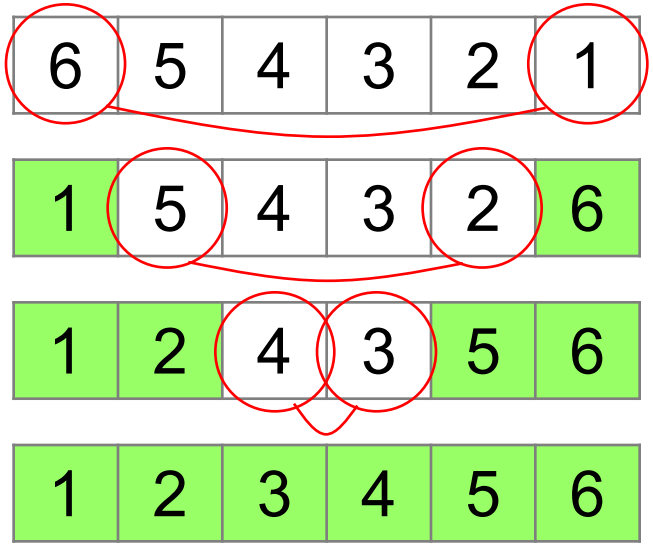
```
for ( i = 0; i < N-1; i++ )
  {
  nMin = i;
  for ( j = i+1; j < N; j++ )
    if ( A[j] < A[nMin] )
      nMin = j;
  if ( i != nMin )
    {
    // поменять местами A[i] и A[nMin]
    }
  }
```

? Как поменять местами два значения?

# Быстрая сортировка (*QuickSort*)

Ч.Э.Хоар

*Идея*: выгоднее переставлять элементы, который находятся дальше друг от друга.

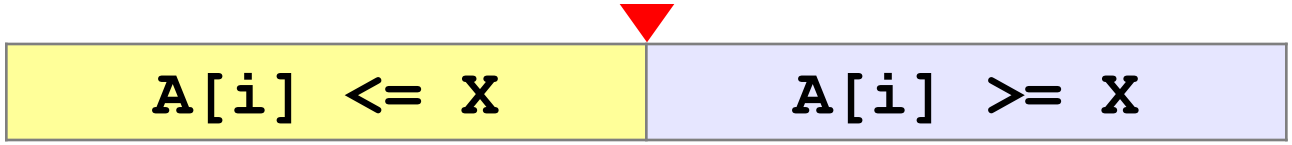


Для массива из **N** элементов нужно всего **N/2** обменов!

# Быстрая сортировка

**Шаг 1**: выбрать некоторый элемент массива X

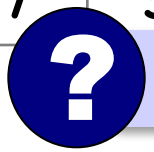
**Шаг 2**: переставить элементы так:




при сортировке элементы не покидают « свою область»!

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

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




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…*).

# Быстрая сортировка

**Разделение**:

1)выбрать средний элемент массива (**X=67**)

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

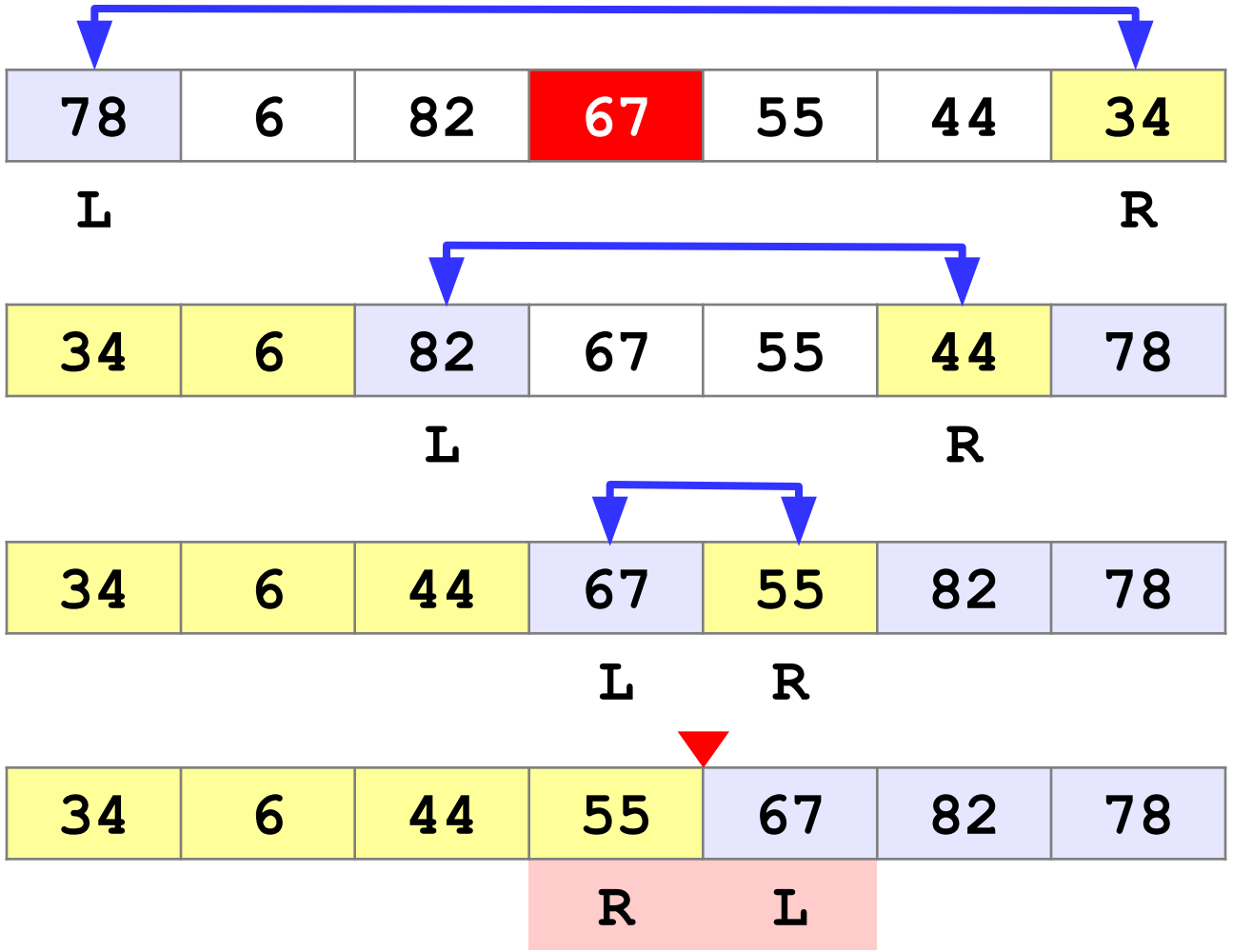
2)установить **L = 1, R = N**

3)увеличивая **L**, найти первый элемент **A[L]**, который **>= X** (должен стоять справа)

4)уменьшая **R**, найти первый элемент **A[R]**, который **<= X** (должен стоять слева)

5)если **L<=R** то поменять местами **A[L]** и **A[R]** и перейти к п. 3
иначе **стоп**.

# Быстрая сортировка




**L > R** : разделение закончено!

# Быстрая сортировка

**Основная программа**:

```
const int N=7;
int A[N];
...
main()
{
  // заполнить массив
  qSort( 0, N-1 ); // сортировка
  // вывести результат
}
```

глобальные данные

процедура сортировки

# Быстрая сортировка

```
void qSort( int nStart, int nEnd )
{
  int L, R, c, X;
  if ( nStart >= nEnd ) return; // готово
  L = nStart; R = nEnd;
  X = A[(L+R)/2]; // или X = A[irand(L,R)];
  while ( L <= R ) {          // разделение
    while ( A[L] < X ) L++;
    while ( A[R] > X ) R--;
    if ( L <= R ) {
      c = A[L]; A[L] = A[R]; A[R] = c;
      L++; R--;
      }
    }
  qSort ( nStart, R );    // рекурсивные вызовы
  qSort ( L, nEnd );
}
```

# Быстрая сортировка

**Передача массива через параметр**:

```
void qSort( int A[], int nStart,
                     int nEnd )
{
   ...
   qSort ( A, nStart, R );
   qSort ( A, L, nEnd );
}
```

```
main()
{ // заполнить массив
   qSort( A, 0, N-1 ); // сортировка
   // вывести результат
}
```

# Быстрая сортировка

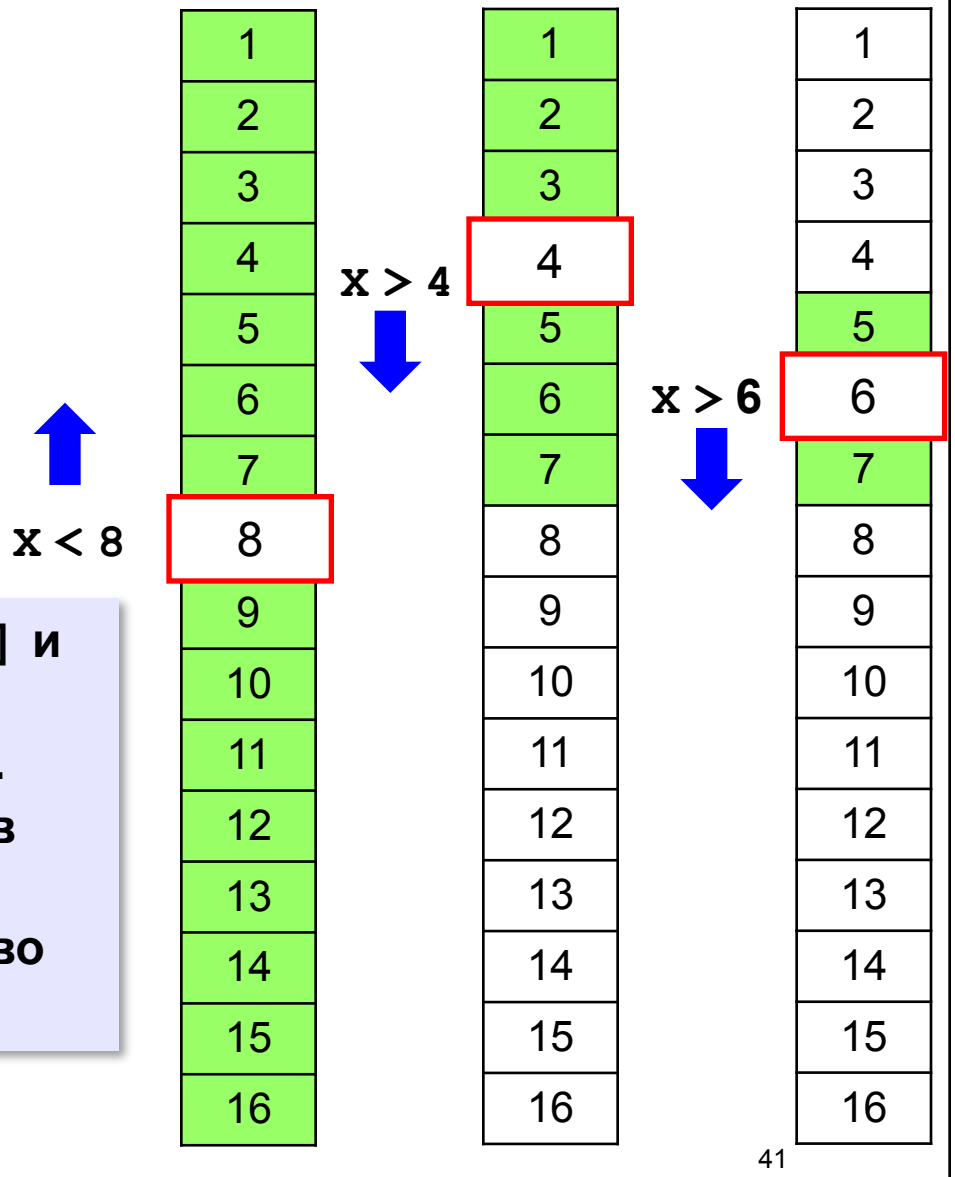
**Сортировка массива случайных значений**:

| N | метод пузырька | метод выбора | быстрая сортировка |
|---|---|---|---|
| 1000 | 0,24 с | 0,12 с | 0,004 с |
| 5000 | 5,3 с | 2,9 с | 0,024 с |
| 15000 | 45 с | 34 с | 0,068 с |

# ДВОИЧНЫЙ ПОИСК

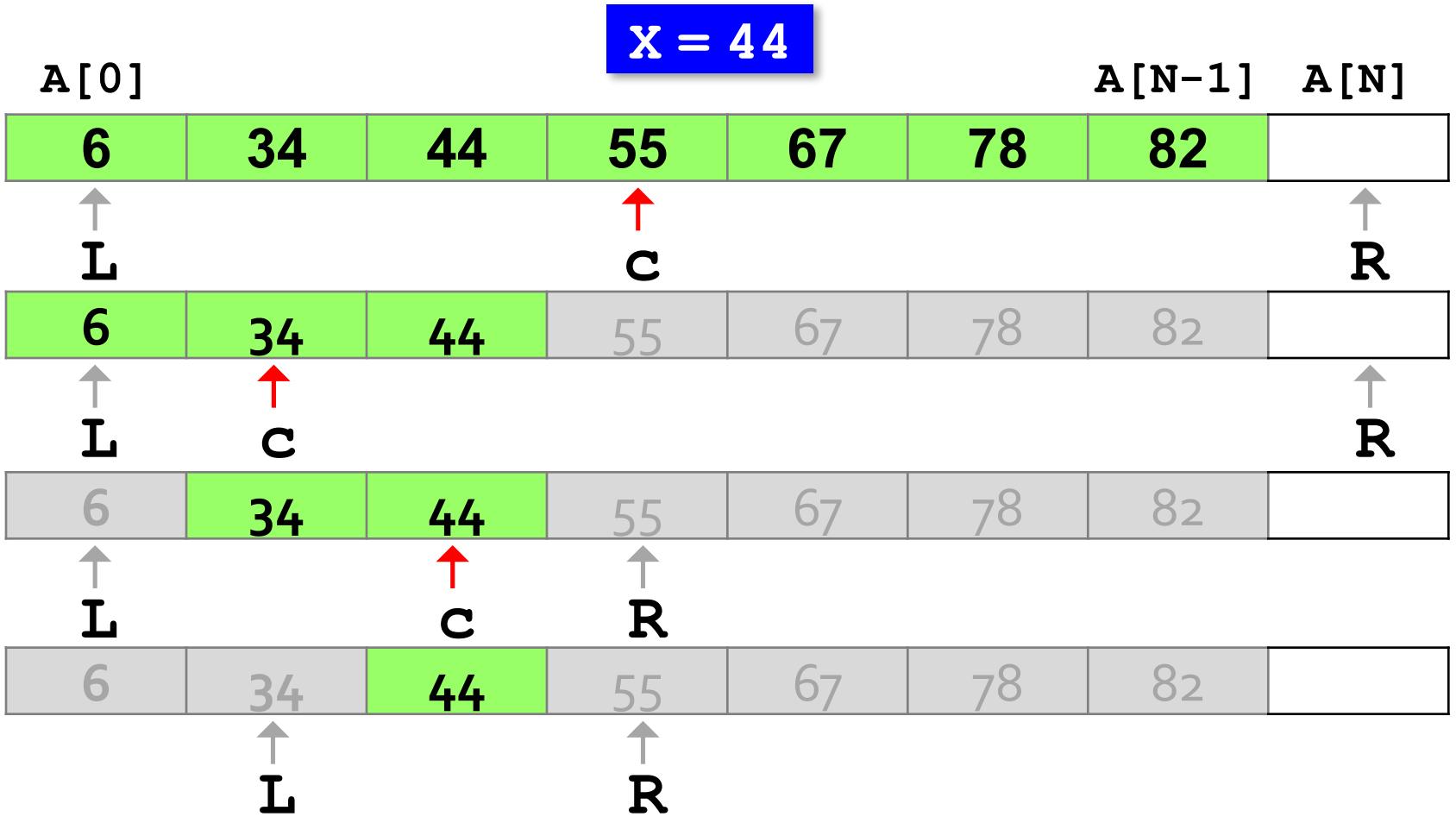
# Двоичный поиск

**X = 7**




1. Выбрать средний элемент `A[c]` и сравнить с X.
2. Если `X = A[c]`, то нашли (стоп).
3. Если `X < A[c]`, искать дальше в первой половине.
4. Если `X > A[c]`, искать дальше во второй половине.

# Двоичный поиск




**!** `L = R-1` : поиск завершен!

# Двоичный поиск

```
int X, L, R, c;
L=0; R=N;          // начальный отрезок
while ( L<R-1 )
   {
   c=(L+R)/2;      // нашли середину
   if ( X<A[c] ) // сжатие отрезка
         R=c;
   else L=c;
   }
if ( A[L]==X )
      printf ( "A[%d]=%d", L, X );
else printf ( "Не нашли!" );
```

# Двоичный поиск

**Число сравнений**:

| N | линейный поиск | двоичный поиск |
|---|---|---|
| 2 | 2 | 2 |
| 16 | 16 | 5 |
| 1024 | 1024 | 11 |
| 1048576 | 1048576 | 21 |

- скорость выше, чем при линейном поиске

- нужна предварительная сортировка

Когда нужно применять?

# МАТРИЦЫ

# Что такое матрица?

| | 0 | 1 | 2 |
|---|---|---|---|
| 0 | -1 | 0 | 1 |
| 1 | -1 | 0 | 1 |
| 2 | 0 | 1 | -1 |

строка 1, столбец 2

**Матрица** — это прямоугольная таблица, составленная из элементов одного типа (чисел, строк и т.д.). Каждый элемент матрицы имеет два индекса – номера строки и столбца.

# Объявление матриц

```
const int N=3, M=4;
int A[N][M];
double X[10][12];
```

строки столбцы

строки столбцы

! Нумерация строк и столбцов с нуля!

Матрицу можно объявить так:

тип имя_массива[n][m];

где n-количество строк, m-количество столбцов.

# Простые алгоритмы

## Заполнение случайными числами:

```
for ( i=0; i<N; i++ ) {
  for ( j=0; j<M; j++ ) {
    A[i][j]=rand()%80;
    cout << width(3);
    cout << A[i][j];
    }
  cout << endl;
  }
```

Вложенный цикл!

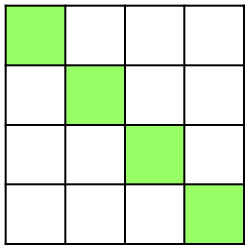
## Суммирование:

```
sum=0;
for ( i=0; i<N; i++ )
  for ( j=0; j<M; j++ )
    sum+=A[i][j];
```

# Перебор элементов матрицы

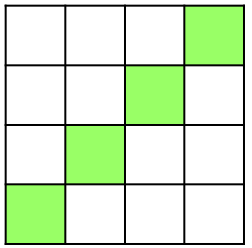
**Главная диагональ:**

```
for ( i = 0; i < N; i++ ) {
   // работаем с A[i][i]
   }
```



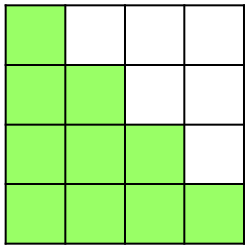
**Побочная диагональ:**

```
for ( i = 0; i < N; i++ ){
   // работаем с A[i][N-1-i]
   }
```



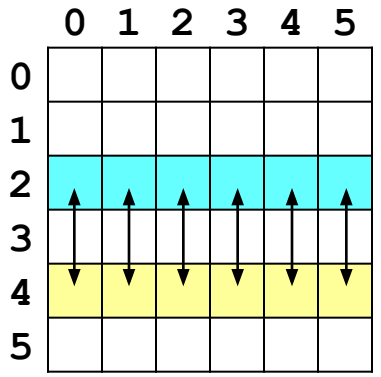
**Главная диагональ и под ней:**

```
for ( i = 0; i < N; i++ )
   for ( j = 0; j <= i ; j++ )
      {
      // работаем с A[i][j]
      }
```



# Перестановка строк

**2-я и 4-я строки:**




```
for ( j=0; j<M; j++ )
   {
   c=A[2][j];
   A[2][j]=A[4][j];
   A[4][j]=c;
   }
```