Программирование на языке Python

Программирование на языке Python

§ 62. Массивы

Что такое массив?

Как ввести 10000 переменных?

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:

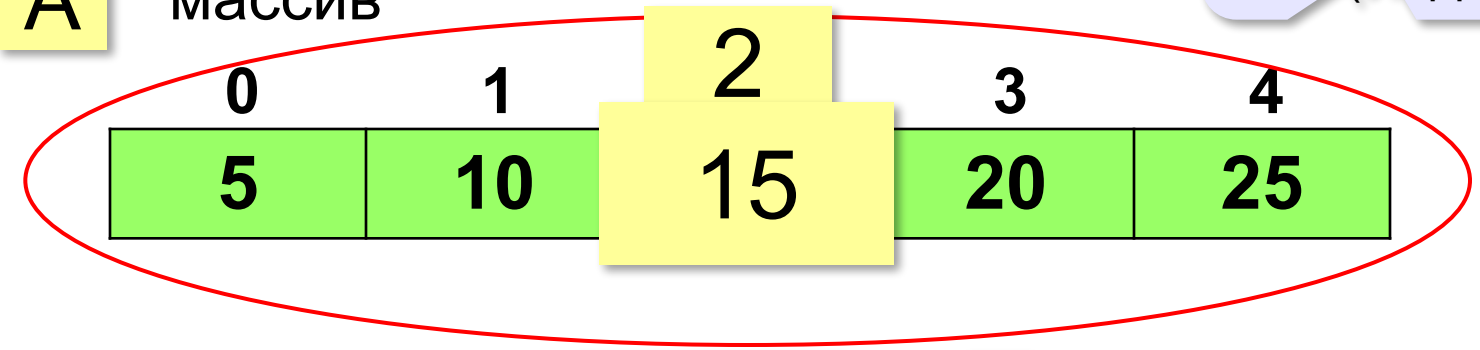
- выделять память
- записывать данные в нужную ячейку
- читать данные из ячейки

Что такое массив?

Массив = **таблица**!

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

Массивы в Python: списки

```
A = [1, 3, 4, 23, 5]
```

```
A = [1, 3] + [4, 23] + [5]
```

```
[1, 3, 4, 23, 5]
```

```
A = [0]*10
```

Что будет?

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

```
A = list( range(10) )
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Генераторы списков

```
A=[ i for i in range(10) ]
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

? Что будет?

```
A=[ i*i for i in range(10) ]
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

```
from random import randint
A=[ randint(20,100)
        for x in range(10)]
```

случайные числа

```
A=[ i for i in range(100)
        if isPrime(i) ]
```

условие отбора

Добавление элементов

```
A = [1, 2, 3]
x = 5
A.append( x+3 ) # [1, 2, 3, 8]
```

в конец списка

Метод – операция, которую можно применить к списку.

```
A = [1, 2, 3]
A.insert( 1, 35 ) # [1, 35, 2, 3]
```

A[1]

? В начало?

```
A.insert ( 0, 90 )
```

```
A = [90] + A
```

Удаление элементов

```
A = [1, 2, 3]
x = A.pop ( 1 )   # x = 2, A = [1, 3]
```

удалить A[1]

```
A = [1, 2, 3]
x = A.pop ()   # x = 3, A = [1, 2]
```

удалить последний

```
A = [11, 29, 37, 45]
A.remove( 37 )   # A = [11, 29, 45]
```

Ввод массива с клавиатуры

Создание массива:

```
N = 10
A = [0]*N
```

Ввод с клавиатуры:

```
for i in range(N):
    print( "A[", i, "]=",
           sep="", end="" )
    A[i] = int( input() )
```

```
A[0] = 5
A[1] = 12
A[2] = 34
A[3] = 56
A[4] = 13
```

`sep = ""`

не разделять элементы

`end = ""`

не переходить на новую строку

Ввод массива с клавиатуры

Ввод без подсказок:

```
A = [ int(input()) for i in range(N) ]
```

Ввод в одной строке:

```
data = input()      # "1 2 3 4 5"
s = data.split()    # ["1","2","3","4","5"]
A = [ int(x) for x in s ]
                    # [1,2,3,4,5]
```

или так:

```
s = input().split()   # ["1","2","3","4","5"]
A = list( map(int, s) )  # [1,2,3,4,5]
```

построить список

применить `int` ко всем элементам `s`

Ввод массива с клавиатуры

Ввод в одной строке:

```
A=[]
N=int(input())
for i in input().split():
    A.append(int(i))
```

Ввод построчно:

```
A=[]
N=int(input())
A = [int(input("Номер %d = " % i))
        for i in range(N)]
```

```
N=int(input('Введите количество чисел ='))
A=[]
for i in range (1,N+1):
    A.append(int(input('Введите число =')))
```

Ввод с клавиатуры и вывод массива

Ввод в одной строке и Вывод массива

```
n=int(input())
A=map(int,input().split(maxsplit = n))
print(n)

for y in A:
    print(y, end = ' ' )
```

Вывод массива на экран

Как список:

```
print ( A )
```

```
[1, 2, 3, 4, 5]
```

В строчку через пробел:

```
for i in range(N):
  print ( A[i], end=" " )
```

```
1 2 3 4 5
```

или так:

```
for x in A:
  print ( x, end=" " )
```

```
1 2 3 4 5
```

или так:

```
print ( *A )
```

⟷

```
print (1, 2, 3, 4, 5)
```

Как обработать все элементы массива?

Создание массива:

```
N = 5
A = [0]*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 += 1
# обработать A[i]
i += 1
# обработать A[i]
i += 1
# обработать A[i]
i += 1
# обработать A[i]
```

```
i += 1
```

Обработка в цикле:

```
i = 0
while  i < N:
   # обработать A[i]
   i += 1
```

Цикл с переменной:

```
for i in range(N):
   # обработать A[i]
```

Заполнение случайными числами

```
from random import randint
N = 10
A = [0]*N
for i in range(N):
    A[i] = randint(20,100)
```

или так:

```
from random import randint
N = 10
A = [ randint(20,100)
            for x in range(N)]
```

случайные числа [20,100]

Перебор элементов

Общая схема (можно изменять A[i]):

```
for i in range(N):
    ... # сделать что-то с A[i]
```

```
for i in range(N):
    A[i] += 1
```

Если не нужно изменять A[i]:

```
for x in A:
    ... # сделать что-то с x
```

x = A[0], A[1], ..., A[N-1]

```
for x in A:
    print ( x )
```

Этапы работы с массивом

```
a=[]
n=int (input ("Введи кол-во элементов
 массива"))
for i in input().split():
    a.append(int (i))
for y in a:
    print (y, end=" ")
print ()
for i in range (n):
 ... # действия с a[i]
print ()
for i in range (n):
     print (a[i], end=' `)
```

Заполнение массива

Вывод созданного массива (1 способ вывода)

Обработка массива

Вывод обработанного массива (2 способ вывода)

```
from random import random
N = 10
a = []
b = []
c = []
for i in range(N):
    n = int(random() * 100)
    a.append(n)

print("Введите числа")
for i in range(N):
    n = int(input())
    b.append(n)

for i in range(N):
    n = a[i] + b[i]
    c.append(n)

print(a)
print(b)
print(c)
```

Три способа создания массива:

Заполнить один массив случайными числами,

другой - введенными с клавиатуры числами,

в ячейки третьего записать суммы соответствующих ячеек первых двух.

Вывести содержимое массивов на экран.

Вывод чисел в обратном порядке

Ввод чисел от A до B в обратном порядке

```
a = int(input())
b = int(input())
a=[i for i in range  (b, a-1, -1)]
for y in a:
    print(y)
```

```
print(*[i for i in range
        (int(input("Введи кол-во чисел")))]
      [::-1])
```

Заполнение случайными числами

```
from random import randint
N = 10
A = [ randint(20,100)
            for x in range(N)]

for y in A:
    print(y,end =' ')
print()

for y in A [::-1]:
    print(y, end =' ‘)
```

[::-1]

Вывод случайных чисел [20,100]

Вывод случайных чисел [20,100] в обратном порядке

Подсчёт количества нужных элементов

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

Как решать?

```
count = 0
for x in A:
    if 180 < x and x < 190:
        count += 1
```

Python:

```
180 < x < 190
```

Сумма элементов массива

```
summa = 0
for x in A:
    if 180 < x < 190:
        summa += x
print ( summa )
```

или так:

```
print ( sum(A) )
```

Перебор элементов

Среднее арифметическое:

```
count = 0
summa = 0
for x in A:
    if 180 < x < 190:
        count += 1
        summa += x
print ( summa/count )
```

среднее арифметическое

или так:

отбираем нужные

```
B = [ x for x in A
            if 180 < x < 190]
print ( sum(B)/len(B) )
```

Задан массив из натуральных чисел. Выведите его в обратном порядке через пробел.

```
def print_rev(arr,k):
    if k<0:
        return
    else:
        print(arr[k],end=' ')
        print_rev(arr,k-1)

a=list(map(int,input().split(' ')))
print_rev(a,len(a)-1)
```

Программирование на языке Python

§ 63. Алгоритмы обработки массивов

Поиск в массиве

Найти элемент, равный X:

```
i = 0
while A[i] != X:
    i += 1
print( "A[", i, "]=", X, sep="" )
```

? Что плохо?

```
i = 0
while i < N and A[i] != X:
    i += 1
if i < N:
    print( "A[", i, "]=", X, sep="" )
else:
    print( "Не нашли!" )
```

? Что если такого нет?

Поиск в массиве

Вариант с досрочным выходом:

номер найденного элемента

```
nX = -1
for i in range ( N ):
  if A[i] == X:
    nX = i
    break
if nX >= 0:
  print ( "A[", nX, "]=", X, sep = "" )
else:
  print ( "Не нашли!" )
```

досрочный выход из цикла

Поиск в массиве

Варианты в стиле Python:

```
for i in range( N ):
  if A[i]==X:
    print( "A[", i, "]=", X, sep="" )
    break
else:
  print( "Не нашли!" )
```

если не было досрочного выхода из цикла

```
if X in A:
  nX=A.index(X)
  print( "A[", nX, "]=", X, sep="" )
else:
  print( "Не нашли!" )
```

Максимальный элемент

```
M=A[0]
for i in range(1,N):
  if A[i]>M:
    M=A[i]
print( M )
```

Если range(N)?

Варианты в стиле Python:

```
M=A[0]
for x in A:
  if x>M:
    M=x
```

Как найти его номер?

```
M=max( A )
```

Максимальный элемент и его номер

```
M = A[0]; nMax = 0
for i in range(1,N):
    if A[i] > M:
        M = A[i]
        nMax = i
print ( "A[", nMax, "]=", M, sep = "" )
```

? Что можно улучшить?

! По номеру элемента можно найти значение!

```
nMax = 0
for i in range(1,N):
    if A[i] > A[nMax]
        nMax = i
print ( "A[", nMax, "]=", A[nMax], sep = "" )
```

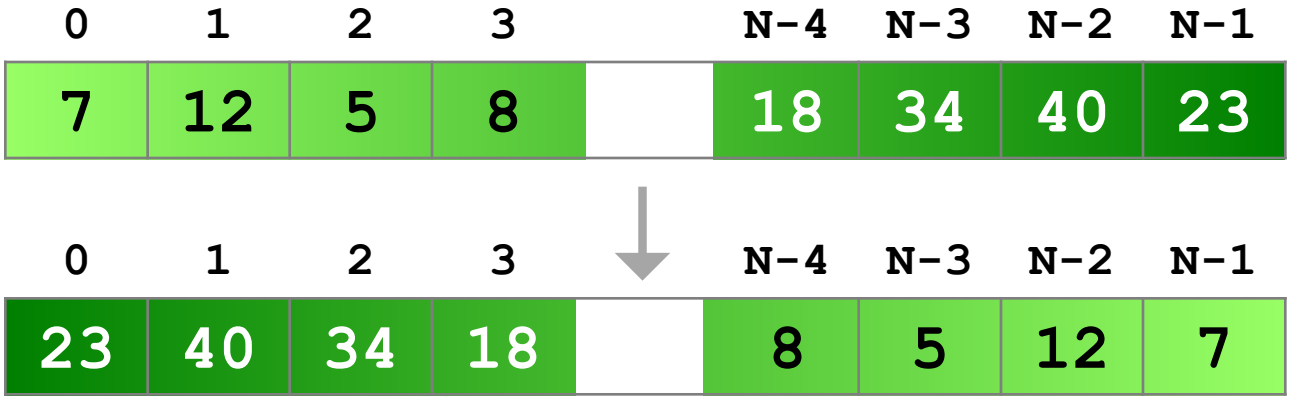
Максимальный элемент и его номер

Вариант в стиле Python:

```
M = max(A)
nMax = A.index(M)
print( "A[", nMax, "]=", M, sep="" )
```

номер заданного элемента (первого из…)

Реверс массива



«Простое» решение:

остановиться на середине!

```
for i in range( N//2 ):
    поменять местами A[i] и A[N-1-i]
```

? Что плохо?

Реверс массива

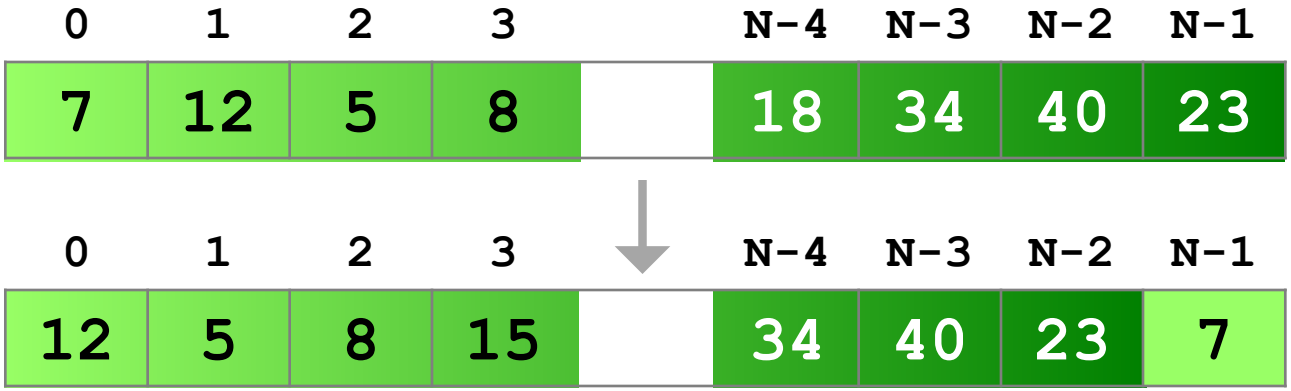
```
for i in range(N//2):
    c=A[i]
    A[i]=A[N-1-i]
    A[N-1-i]=c
```

Варианты в стиле Python:

```
for i in range(N//2):
    A[i],A[N-i-1]=A[N-i-1],A[i]
```

```
A.reverse()
```

Циклический сдвиг элементов



«Простое» решение:

```
for i in range(N-1):
  A[i]=A[i+1]
```

? Почему не до N?

? Что плохо?

Срезы в Python

! Последний элемент не входит в срез!

```
A[1:3] → [12, 5]
A[2:3] → [5]
A[:3] → A[0:3] → [7, 12, 5]
```

с начала

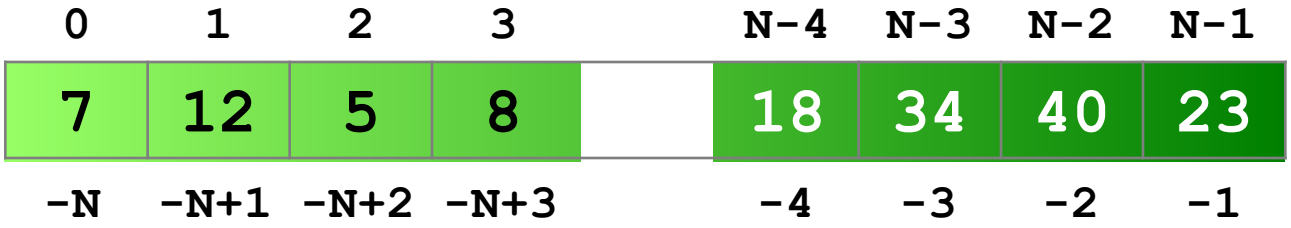
```
A[3:N-2] → [8,…,18,34]
A[3:] → A[3:N] → [8,…,18,34,40,23]
```

до конца

копия массива

```
A[:] → [7,12,5,8,…,18,34,40,23]
```

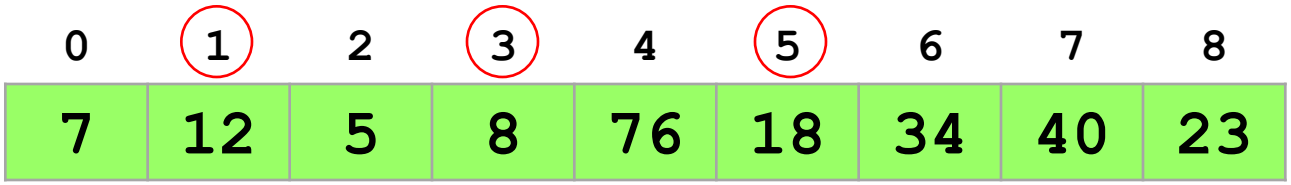
Срезы в Python – отрицательные индексы




```
A[1:-1]
A[1:N-1]   → [12,5,8,…,18,34,40]
```

```
A[-4:-2]
A[N-4:N-2]   → [18, 34]
```

Срезы в Python – шаг



шаг

```
A[1:6:2] → [12, 8, 18]
A[::3] → [7, 8, 34]
A[8:2:-2] → [23, 34, 76]
A[::-1] → [23,40,34,18,76,8,5,12,7]
```

реверс! `A.reverse()`

Отбор нужных элементов

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

Простое решение:

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

```
B = []
for x in A:
  if x % 2 == 0:
    B.append(x)
```

? Какие элементы выбираем?

добавить **x** в конец массива **B**

Отбор нужных элементов

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

Решение в стиле Python:

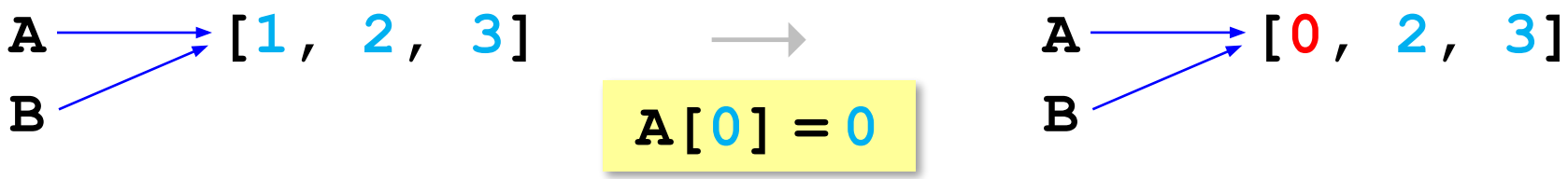
перебрать все элементы A

```
B = [ x for x in A
      if x % 2 == 0 ]
```

если **x** – чётное число

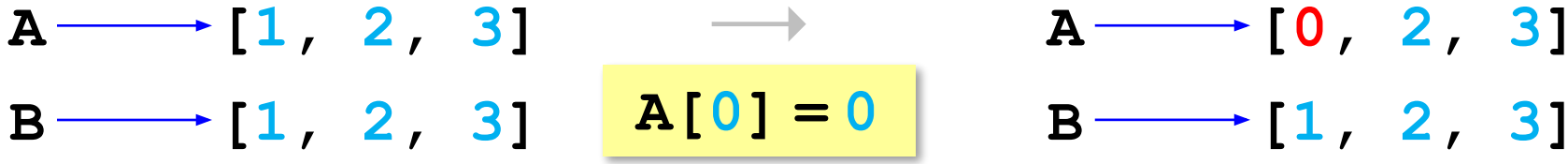
Особенности работы со списками

```
A = [1, 2, 3]
B = A
```




```
A = [1, 2, 3]
B = A[:]
```

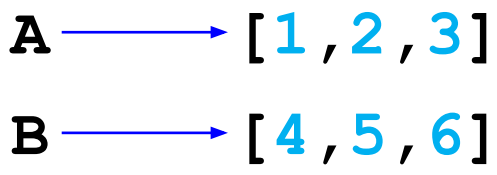
копия массива A



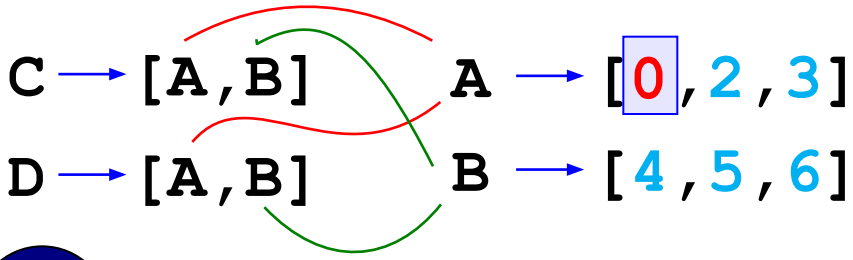
Копирование списков

«Поверхностное» копирование:

```
import copy
A = [1, 2, 3]
B = copy.copy(A)
```

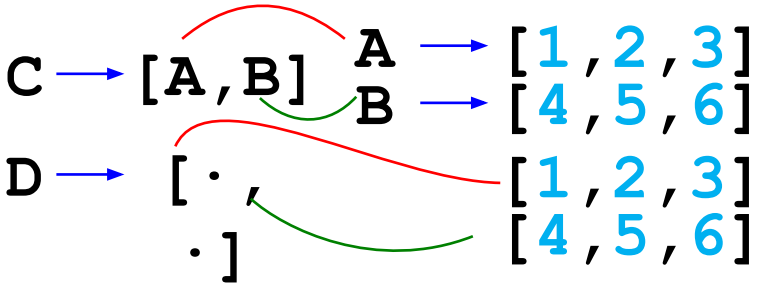



```
A = [1, 2, 3]
B = [4, 5, 6]
C = [A, B]
D = copy.copy(C)
C[0][0] = 0
```



«Глубокое» копирование:

```
D = copy.deepcopy(C)
```



Программирование на языке Python

§ 64. Сортировка

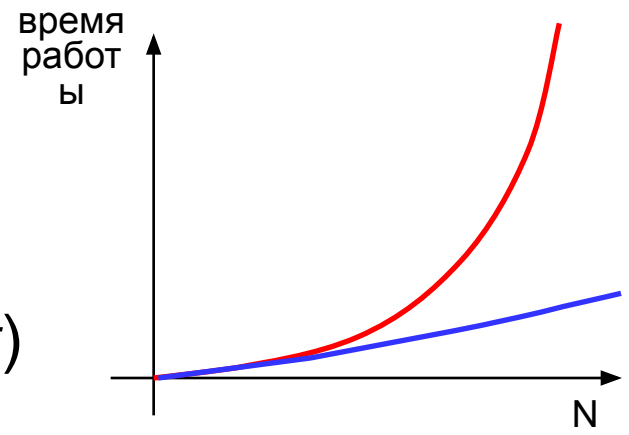
Что такое сортировка?

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

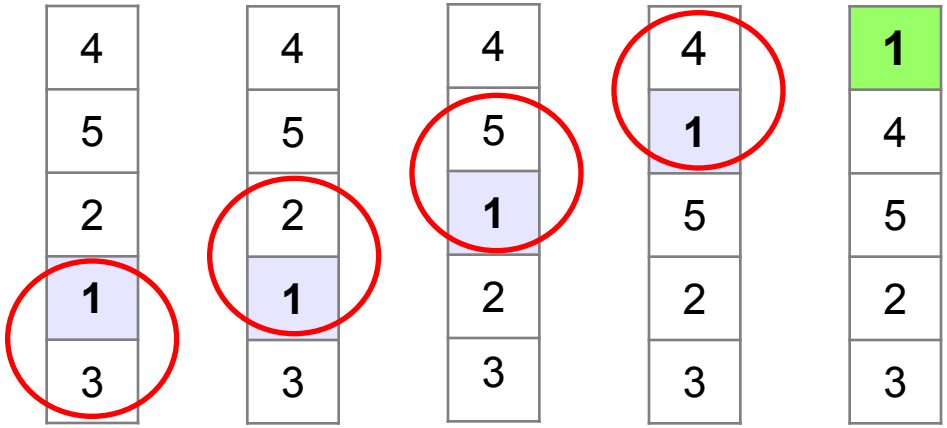


Метод пузырька (сортировка обменами)

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

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

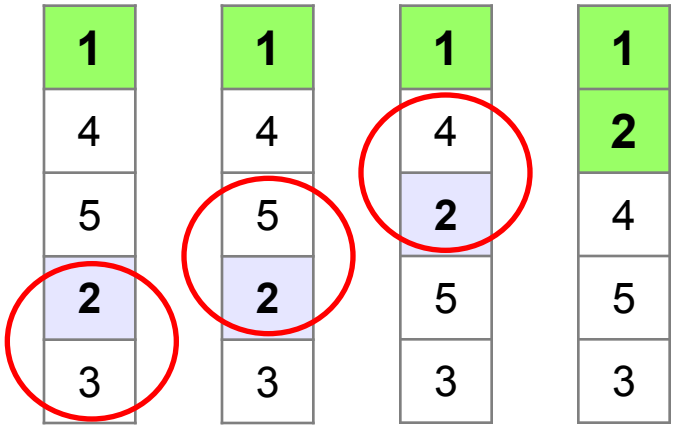
1-й проход:



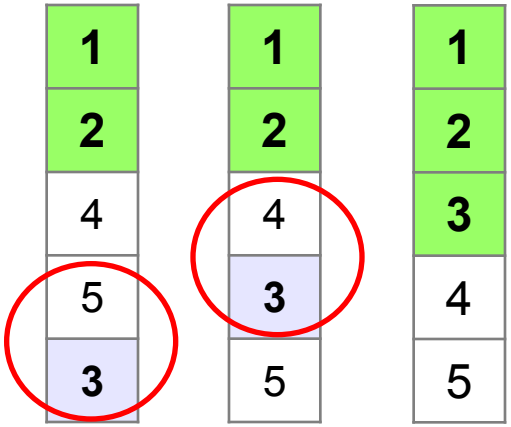
- сравниваем два соседних элемента; если они стоят «неправильно», меняем их местами
- за 1 проход по массиву один элемент (самый маленький) становится на свое место

Метод пузырька

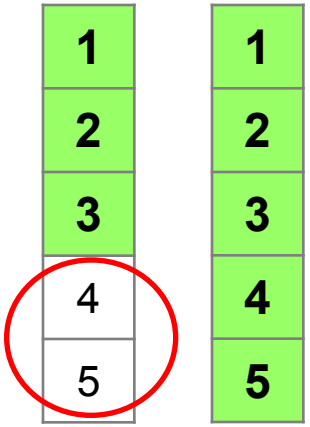
2-й проход:



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]
```

2-й проход:

единственное отличие!

```
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

от N-2 до 0 шаг -1

1-й проход:

```
for j in range(N-2, -1 ,-1):
  if A[j+1]<A[j]:
    # поменять местами A[j] и A[j+1]
```

единственное отличие!

2-й проход:

```
for j in range(N-2, 0 ,-1):
  if A[j+1]<A[j]:
    # поменять местами A[j] и A[j+1]
```

Метод пузырька

```
for i in range(N-1):
  for j in range(N-2, i- ,-1):
    if A[j+1]<A[j]:   1
      A[j],A[j+1]=A[j+1],A[j]
```

? Как написать метод «камня»?

? Как сделать рекурсивный вариант?

```
a=[]
n=int (input ("Введи кол-во элементов массива"))
for i in input().split():
    a.append(int (i))
for y in a:
    print (y, end=" ")
print ()
for i in range (n-1):
    for j in range (n-2,i-1,-1):
        if a[j+1]<a[j]:
            a[j],a[j+1]=a[j+1],a[j]
print ()
for i in range (n):
     print (a[i], end=' ')
print ()
for y in a:
    print (y, end=" ")
```

Метод выбора (минимального элемента)

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

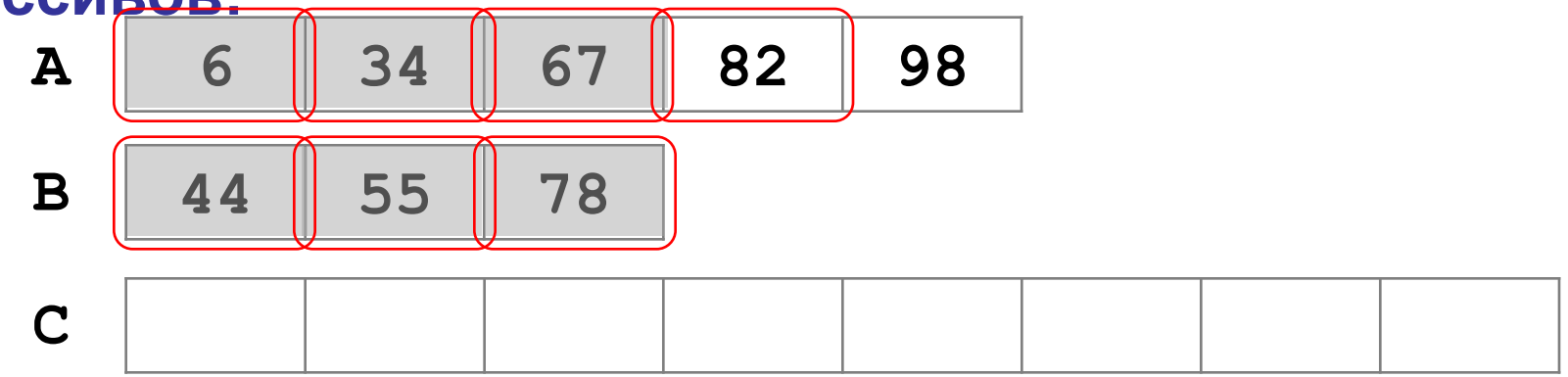
```
for i in range(N-1):
  найти номер nMin минимального
      элемента из A[i]..A[N]
  if i != nMin:
    поменять местами A[i] и A[nMin]
```

Метод выбора (минимального элемента)

```
for i in range(N-1):
    nMin = i
    for j in range(i+1,N):
        if A[j] < A[nMin]:
            nMin = j
    if i != nMin:
        A[i], A[nMin] = A[nMin], A[i]
```

Сортировка слиянием

Слияние отсортированных массивов:




```
Na = len(A); Nb = len(B)
iA = iB = 0; C = []
while iA < Na and iB < Nb:
  if A[iA] <= B[iB]:
    C.append(A[iA]); iA += 1
  else:
    C.append(B[iB]); iB += 1
C = C + A[iA:] + B[iB:]
```

пока оба массива непустые

добавить остаток

Сортировка слиянием

```
def mergeSort( A ):
    if len(A) <= 1: return
    mid = len(A) // 2
    L = A[:mid]
    R = A[mid:]
    mergeSort(L)
    mergeSort(R)
    C = merge(L, R)
    for i in range(len(A)):
        A[i] = C[i]
```

выход из рекурсии

рекурсивные вызовы

слияние

копируем результат в массив **A**

? Почему нельзя **A = C**?

Сортировка слиянием

Процедура слияния:

```
def merge( A, B ):
  Na = len(A); Nb = len(B)
  iA = iB = 0; C = []
  while iA < Na and iB < Nb:
    if A[iA] <= B[iB]:
      C.append(A[iA]); iA += 1
    else:
      C.append(B[iB]); iB += 1
  C = C + A[iA:] + B[iB:]
  return C
```

Сортировка слиянием

- работает быстро

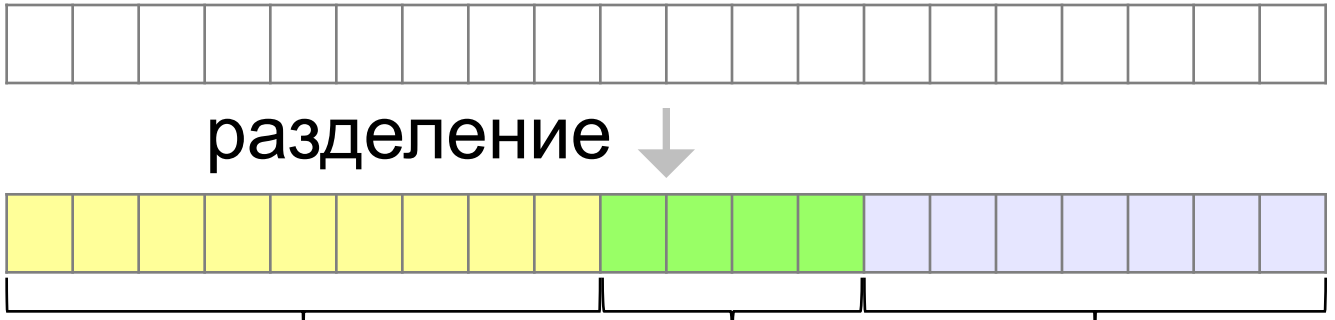
- нужен дополнительный массив

«Разделяй и властвуй» (*divide and conquer*):

1) задача разбивается на несколько подзадач меньшего размера;
2) эти подзадачи решаются с помощью рекурсивных вызовов того же (или другого) алгоритма;
3) решения подзадач объединяются, и получается решение исходной задачи.

Быстрая сортировка (*QuickSort*)

Ч.Э.Хоар

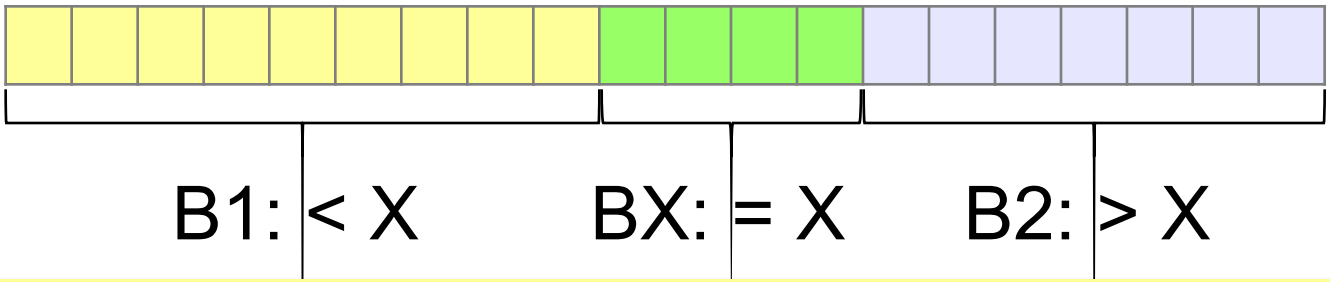


! Эти части нужно так же отсортировать!

? Как лучше выбирать X?

Медиана – такое значение **X**, что слева и справа от него в отсортированном массиве стоит одинаковое число элементов (*долго искать …*).

Быстрая сортировка (*QuickSort*)




```
import random
def qSort ( A ):
   if len(A) <= 1: return A
   X = random.choice(A)
   B1 = [ b for b in A if b < X ]
   BX = [ b for b in A if b == X ]
   B2 = [ b for b in A if b > X ]
   return qSort(B1) + BX + qSort(B2)
```

? Где рекурсия?

расход памяти!

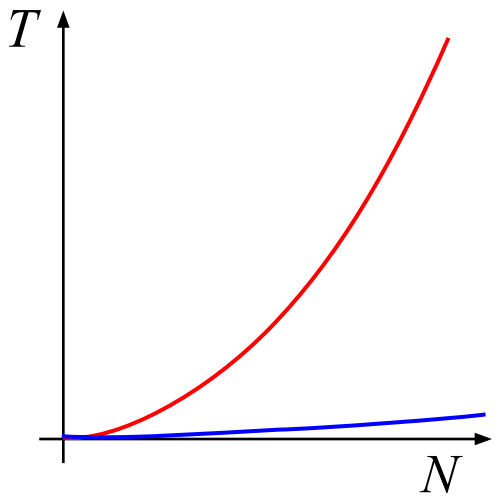
```
Asort = qSort( A )
```

? Что плохо?

Сравнение алгоритмов сортировки

N	Метод пузырька	Метод выбора	Сортировка слиянием	Быстрая сортировка
1000	0,08 с	0,05 с	0,006 с	0,002 с
5000	1,8 с	1,3 с	0,033 с	0,006 с
15000	17,3 с	11,2 с	0,108 с	0,019 с

$\sim N^2$

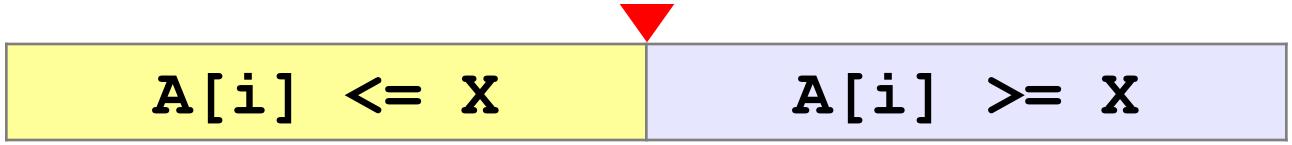


$\sim N \log N$

Быстрая сортировка «на месте»

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*)

78	6	82	67	55	44	34

? Как лучше выбрать X?

Быстрая сортировка

Разделение:

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

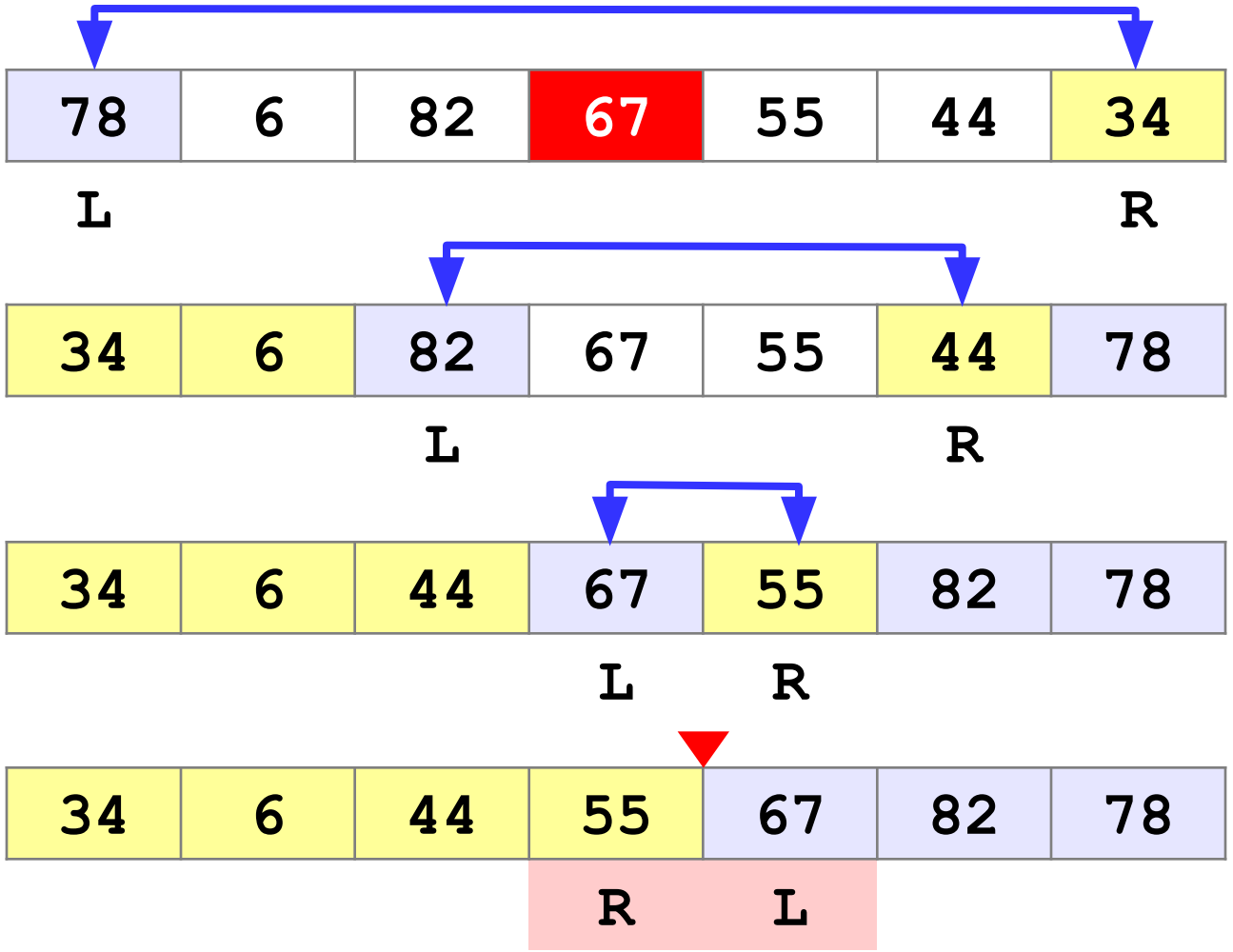
2)установить **L = 0, R = N-1**

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:

```
N = 7
A = [0]*N
# заполнить массив
# массив  # начало  # конец

qSort( A, 0, N-1 )  # сортировка
# вывести результат
```

Быстрая сортировка

массив начало конец

```
def qSort ( A, nStart, nEnd ):
  if nStart >= nEnd: return
  L = nStart; R = nEnd
  X = A[(L+R)//2]
  while L <= R:
    while A[L] < X: L += 1
    while A[R] > X: R -= 1
    if L <= R:
      A[L], A[R] = A[R], A[L]
      L += 1; R -= 1
  qSort ( A, nStart, R )
  qSort ( A, L, nEnd )
```

разделение на 2 части

меняем местами

рекурсивные вызовы

Быстрая сортировка

Случайный выбор элемента-разделителя:

```
from random import randint
def qSort ( A, nStart, nEnd ):
    ...
    X=A[randint(L,R)]
    ...
```

или так:

```
from random import choice
def qSort ( A, nStart, nEnd ):
    ...
    X=choice ( A[L:R+1] )
    ...
```

Сортировка в Python

По возрастанию:

```
B = sorted( A )
```

алгоритм *Timsort*

По убыванию:

```
B = sorted( A, reverse = True )
```

По последней цифре:

```
def lastDigit ( n ):
    return n % 10
B = sorted( A, key = lastDigit )
```

или так:

```
B = sorted( A, key = lambda x: x % 10 )
```

«лямбда»-функция (функция без имени)

Сортировка в Python – на месте

По возрастанию:

```
A.sort()
```

По убыванию:

```
A.sort ( reverse = True )
```

По последней цифре:

```
def lastDigit ( n ):
    return n % 10
A.sort ( key = lastDigit )
```

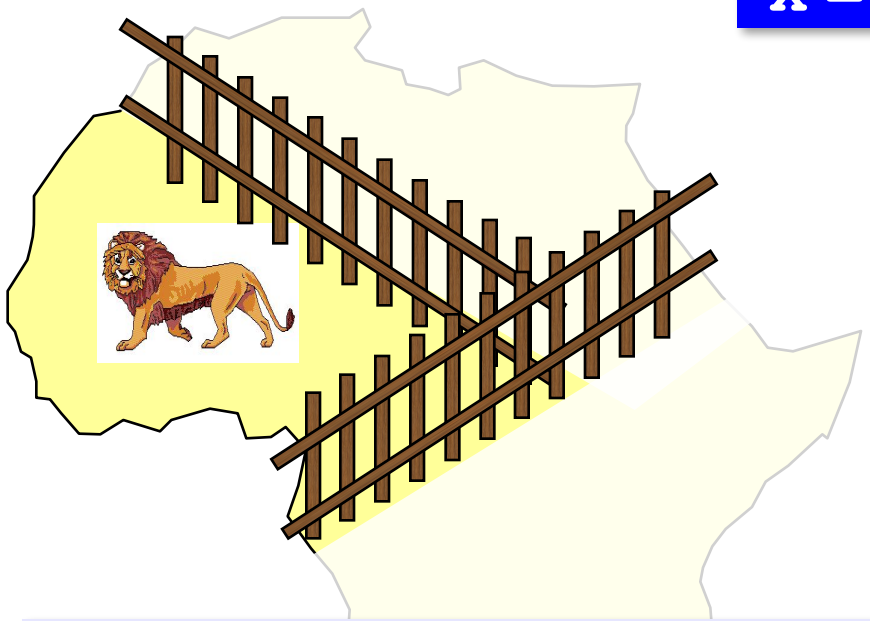
или так:

```
A.sort( key = lambda x: x % 10 )
```

Программирование на языке Python

§ 65. Двоичный поиск

Двоичный поиск



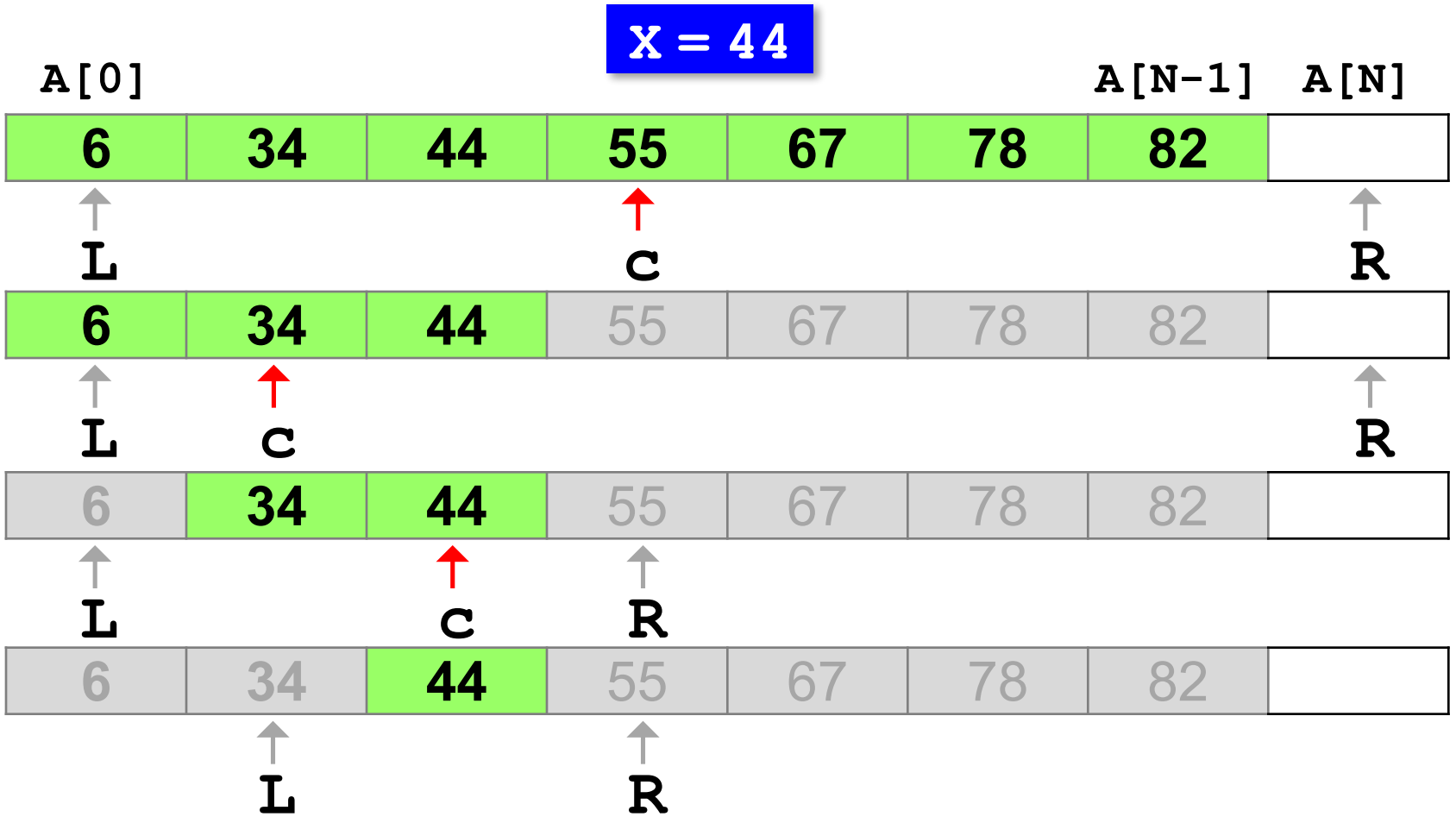
X = 7

1	1	1	1	
2	2	2	2	
3	3	3	3	
4	4	4	4	
5	5	5	5	
6	6	6	6	
7	7	7	7	
8	8	8	8	
9	9	9	9	
10	10	10	10	
11	11	11	11	
12	12	12	12	
13	13	13	13	
14	14	14	14	
15	15	15	15	
16	16	16	16	

X < 8 **X > 4** **X > 6**

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

```
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:
   print ( "A[", L, "]=", X, sep = "" )
else:
   print ( "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?