

L#7

# Basics of Programming. Procedures and functions

Course Basics of Programming Semester 1, FIIT

Mayer Svetlana Fyodorovna

# + and \* operations for arrays

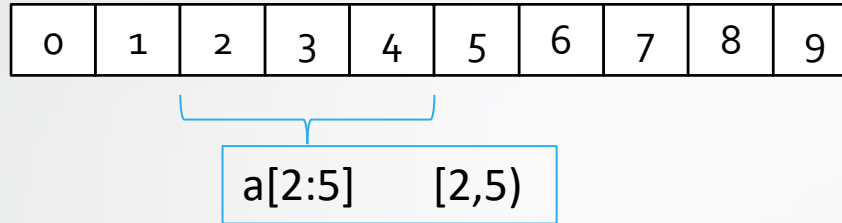
**a + b** – concatenation of two arrays into result array

**a \* N** – concatenation of **N** copies of **a** into result array

```
var a := Arr(1, 2, 3);  
var b := Arr(4, 5, 6);  
  
a + b // 1 2 3 4 5 6  
a * 3 // 1 2 3 1 2 3 1 2 3  
Arr(1) * 10 // 1 1 1 1 1 1 1 1 1 1  
Arr(1)*5 + Arr(2)*5 // 1 1 1 1 1 2 2 2 2 2  
(Arr(1) + Arr(2))*5 // 1 2 1 2 1 2 1 2 1 2
```

# Array slices

```
var a := Arr(0,1,2,3,4,5,6,7,8,9);
```



**Array slice** is a subarray of original array

It has one of two forms: **a[x:y]** or **a[x:y:step]**. Expressions **x** and **y** can be omitted.

```
a[:4] - 0 1 2 3
a[4:] - 4 5 6 7 8 9
a[:a.Length-1] - 0 1 2 3 4 5 6 7 8
a[:] - 0 1 2 3 4 5 6 7 8 9 (copy of a)
a[::2] - 0 2 4 6 8
a[1::2] - 1 3 5 7 9
a[4:1:-1] - 4 3 2
a[::-1] - 9 8 7 6 5 4 3 2 1 0 (reverse of a)
```

# Example

- **To do:** an array  $A$  of size  $N$  and an integer  $K$  ( $1 \leq K \leq N$ ) are given. Print its elements with ordinal numbers (i.e. indexes) that are multiples of  $K$  (i.e. divisible by  $K$ ):
- $A_k, A_{2 * k}, A_{3 * k} \dots$
- Do not use the `if` statement.

```
begin  
var n:=ReadInteger('how many elements'); // 4  
var a:=ReadArrReal(n); // 2 6 7 5  
var k:=ReadInteger('K= '); // 2  
a[k-1 : : k].Print; // 6 5  
end.
```

# Example

- **To do:** An array  $A$  of size  $N$  is given. First, output its elements with even ordinal numbers (in ascending order of ordinal numbers), and then — elements with odd ordinal numbers (also in ascending order of ordinal numbers):

$a_2, a_4, a_6, \dots, a_1, a_3, a_5 \dots$

- Do not use conditional operator.

```
begin  
var n:=ReadInteger('how many elements');  
var a:=arrRandomInteger(n);  
a.Println;  
var slice:=a[1::2]+a[::2];  
slice.Print  
end.
```

```
how many elements 10  
8 96 24 61 80 60 24 40 95 15  
96 61 60 40 15 8 24 80 24 95
```

# Example

- **To do:** An array of size **N** and integers **K** and **L** ( $1 \leq K \leq L \leq N$ ) are given. Find the arithmetic mean (average) of the elements of an array with numbers from **K** to **L** inclusive.

```
begin  
var n:=ReadInteger;  
var a:=arrrandominteger(n);  
a.Println;  
var k:=ReadInteger('K = ');  
var l:=ReadInteger('L = ');  
var slice:=a[k-1:l].Average;  
slice.Print;  
end.
```

```
>> 10  
59 87 0 37 57 69 79 19 100 5  
K = >> 2  
L = >> 4  
41.33333333333333
```

# Example

- **To do:** An array of size **N** is given. Find the minimum element of its even-numbered elements:

$a_2, a_4, a_6, \dots$

```
begin  
var n:=ReadInteger;  
var a:=arrRandomInteger(n);  
a.Println;  
println('slice: ',a[1::2]);  
print(a[1::2].min);  
end.
```

```
>> 10  
96 79 71 87 61 21 51 74 67 89  
slice: [79,87,21,74,89]  
21
```

# Reverse of an array

```
begin  
var a:=new integer[10];  
a:=arrRandomInteger(10);  
print(a); // [41,81,84,63,12,26,88,25,36,72]  
Reverse(a);  
print(a) // [72,36,25,88,26,12,63,84,81,41]  
end.
```

A standard **Reverse(a)** procedure has this algorithm. Thus, we don't need to create it in our program, it's possible just to use it.

We can use slices:  $a := a[::-1]$

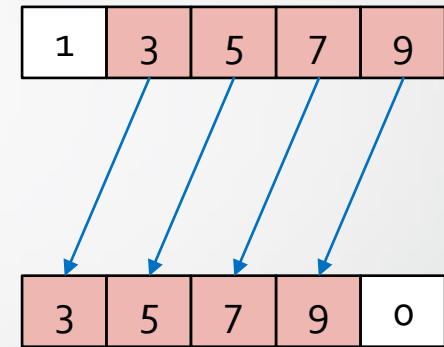


# Shift to the left

**Problem:** Create a user procedure to shift the elements to the left

```
procedure ShiftLeft<T>(a: array of T);
begin
  for var i := 0 to a.Length - 2 do
    a[i] := a[i + 1];
  a[a.Length - 1] := default(T);
end;

begin
  var a := new integer[5];
  a := arrRandomInteger(5); // [56,28,33,57,25]
  shiftLeft(a);
  print(a) // [28,33,57,25,0]
end.
```



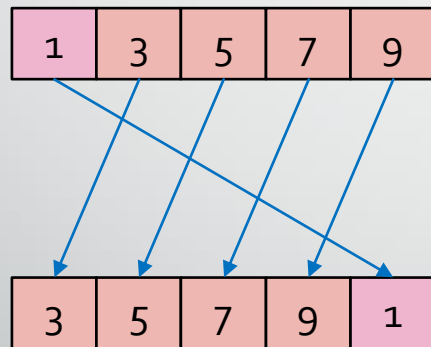
With slices:

`a := a[1:] + Arr(0);`

`[3, 5, 7, 9] + [0]`

# Circular shift left

```
procedure CircularShiftLeft<T>(a: array of T);  
begin  
  var v := a[0];  
  for var i:=0 to a.Length-2 do  
    a[i] := a[i+1];  
  a[a.Length-1] := v;  
end;
```



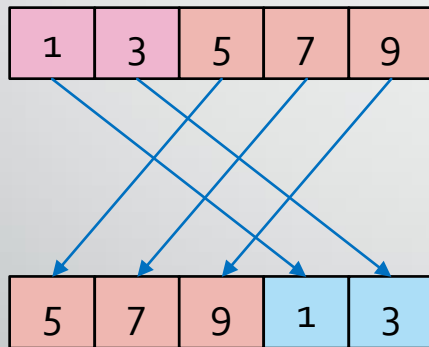
With slices:

$a := a[1:] + a[:1];$

$[3, 5, 7, 9] + [1]$

# Circular shift left by k

1. **loop** k do  
    CircularShiftLeft(a); // ineffective
2. With second array
3. With partial reverse



k = 2

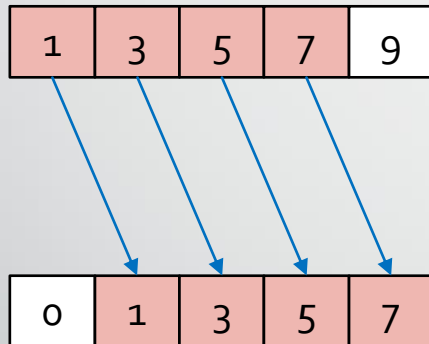
Using slices:

$a := a[k:] + a[:k];$

[5, 7, 9] + [1, 3]

# Shift right

```
procedure ShiftRight<T>(a: array of T);  
begin  
  for var i:=a.Length-1 downto 1 do  
    a[i] := a[i-1];  
  a[0] := default(T);  
end;
```



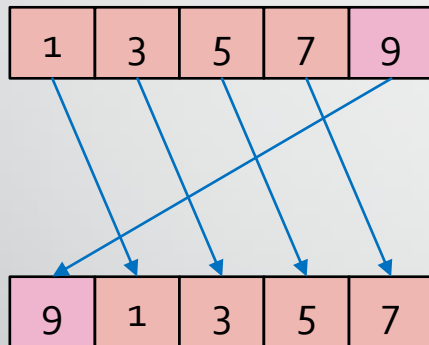
Using slices:

```
a := Arr(0) + a[:a.Length-1];
```

```
[0] + [1, 3, 5, 7]
```

# Circular shift to the right

```
procedure CircularShiftRight<T>(a: array of T);  
begin  
  var v := a[a.Length-1];  
  for var i:=a.Length-1 downto 1 do  
    a[i] := a[i-1];  
  a[0] := v;  
end;
```



Using slices:

```
var m := a.Length-1;  
a := a[m:] + a[:m];
```

```
[9] + [1, 3, 5, 7]
```

# Insertion and deletion in an array

## using slices

**Problem 1.** An array of  $N$  integers is given. It's necessary to insert an element  $x$  on  $k$ -th index,  $k \leq N$ .

```
begin
  var a := arr(5, 12, 1, 3, 11, 19);
  var x := ReadInteger ('enter a number to insert');
  var k := ReadInteger ('enter an order number');
  a := a[:k] + Arr(x) + a[k:];
  print(a) // [5,12,3,1,3,11,19]
end.
```

**Problem 2.** An array of  $N$  integers is given. It's necessary to delete an element with index  $k$ ,  $k < N$ .

```
begin
  var a := arr(5, 12, 1, 3, 11, 19);
  var k := ReadInteger ('enter an order number');
  a := a[:k] + a[k+1:];
  print(a) // [5,12,3,11,19]
end.
```

- 
- Tasks 1,2,3,4,5,6,7,8



# Array sorting algorithms



# Selection sort

```
procedure SelectionSort(a: array of integer);  
begin  
  for var i := 0 to a.High-1 do  
    begin  
      var (min, imin) := (a[i], i);  
      for var j := i + 1 to a.High do  
        if a[j] < min then  
          (min, imin) := (a[j], j);  
        Swap(a[imin], a[i]);  
      end;  
    end;  
end;
```



- this algorithm iterates over the array over and over, moving one value to the correct position
- At the next iteration, we will find the minimum in the array after the current element and change it with it, if necessary. Thus, after the i-th iteration, the first i elements will stay in their places.
- it selects the smallest unsorted value and swap it with that which was the smallest
- the sorted portion of the array is at the beginning

# SELECTIONSORT

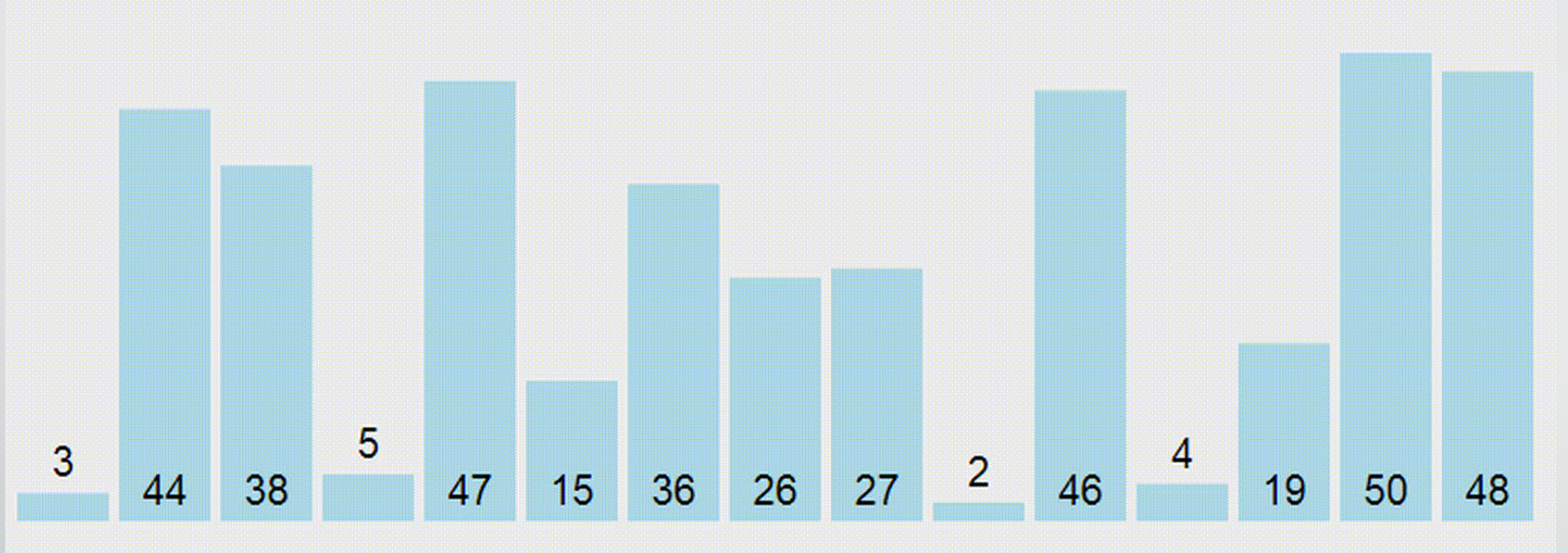
10 randomly ordered elements



INVERSIONS

MAX 45

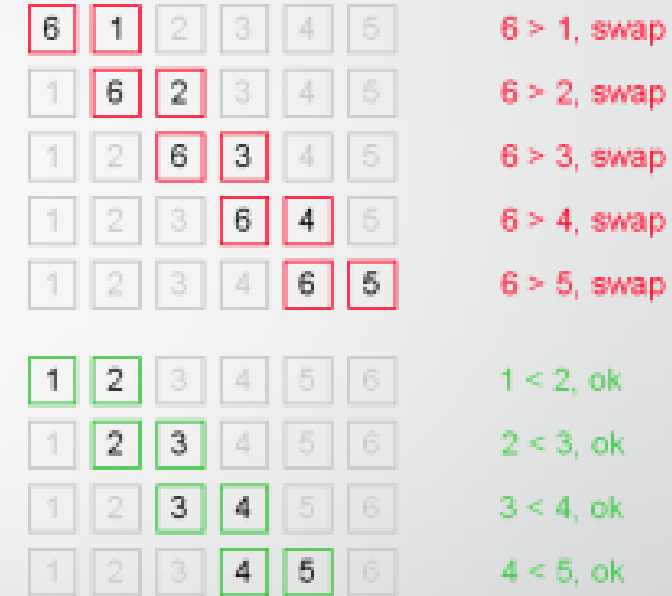




```
procedure SelectionSort(a: array of integer);  
begin  
  for var i := 0 to a.High-1 do  
  begin  
    var (min,imin) := (a[i],i);  
    for var j := i + 1 to a.High do  
      if a[j] < min then  
        (min,imin) := (a[j],j);  
      Swap(a[imin],a[i]);  
    end;  
  end;  
end;
```

# Bubble sort

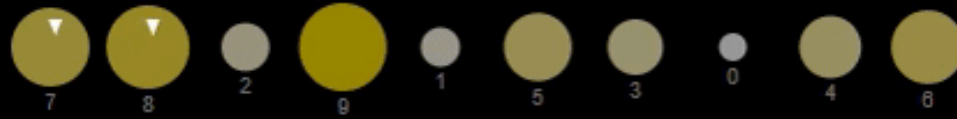
```
procedure BubbleSort(arr: array of integer);  
begin  
  for var i := 1 to arr.High - 1 do  
    for var j := 0 to arr.High - i do  
      if arr[j] > arr[j + 1] then  
        Swap(arr[j], arr[j + 1]);  
end;
```



- Let's iterate over the array from left to right.
- If the current element is greater than the next one, we swap them.
- We do this until the array is sorted.
- Note that after the first iteration, the largest element will be at the end of the array, in the correct place.
- After two iterations, the two largest items will be in the correct place, and so on.

# BUBBLE SORT

10 randomly ordered elements



INVERSIONS

MAX 45

27

0

0

10

20

30

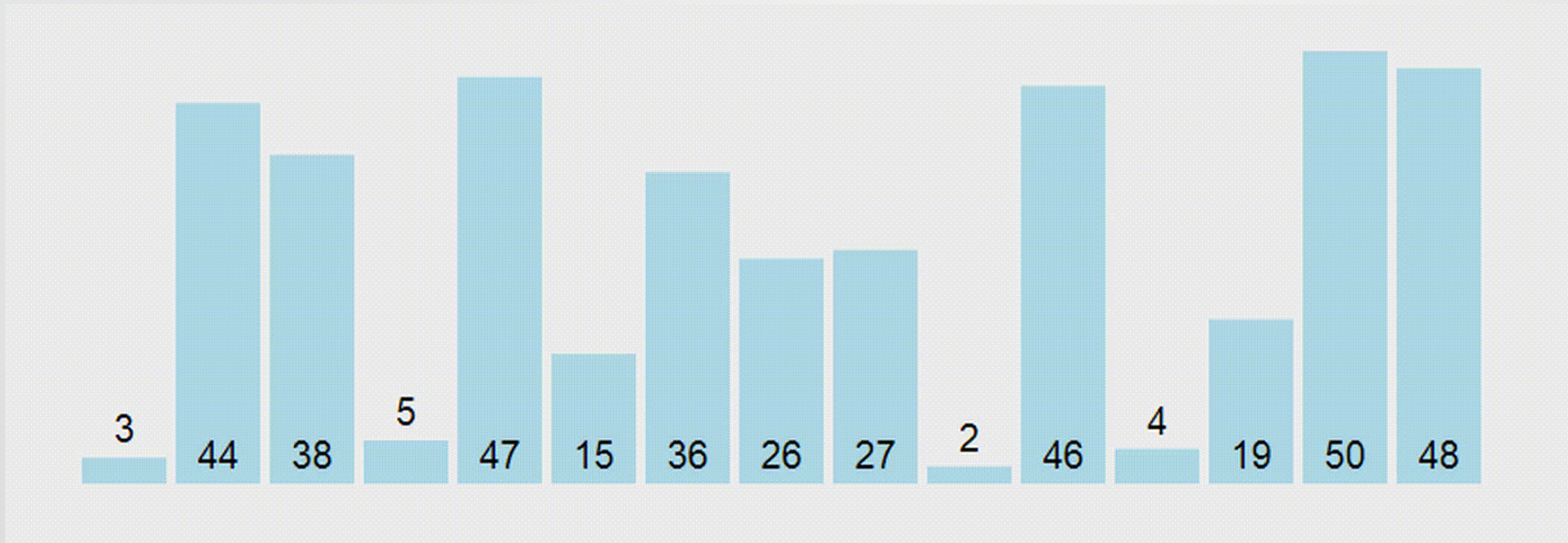
40

50

60

OPERATIONS





```
procedure BubbleSort(arr: array of  
integer);  
begin  
  for var i := 1 to arr.High - 1 do  
    for var j := 0 to arr.High - i do  
      if arr[j] > arr[j + 1] then  
        Swap(arr[j], arr[j + 1]);  
end;
```

# Bubble sort 2

```
procedure BubbleSort2(a: array of integer);  
begin  
  var i := a.High;  
  var q: boolean;  
  repeat  
    q := true;  
    for var j := 0 to i - 1 do  
      if a[j+1] < a[j] then  
        begin  
          Swap(a[j+1], a[j]);  
          q := false;  
        end;  
    i -= 1;  
  until q;  
end;
```

# Insertion sort

```
procedure SortByInsert(a: array of integer);  
begin  
  for var i:=1 to a.High do  
    begin  
      var x := a[i];  
      var j := i - 1;  
      while (j >= 0) and (x < a[j]) do  
        begin  
          a[j+1] := a[j];  
          j -= 1;  
        end;  
        a[j+1] := x;  
      end;  
    end;  
end;
```

After insertion of every element we have sorted array.  
X is a variable to store the value while comparing others

The *Insertion sort* algorithm iterates through the elements of the array one at a time, and places each new taken element in a suitable place among the previously ordered elements.



14	13	11	0	19	13
----	----	----	---	----	----

← 14 храним «в уме» и сравниваем

1	14	14	11	0	19	13
	13	14	11	0	19	13

13	14	11	0	19	13
----	----	----	---	----	----

← 11 храним «в уме»

2	13	14	14	0	19	13
	13	13	14	0	19	13
	11	13	14	0	19	13

0 храним «в уме» и сравниваем

11	13	14	0	19	13
----	----	----	---	----	----

3	11	13	13	14	19	13
	11	13	13	14	19	13
	11	11	13	14	19	13
	0	11	13	14	19	13

0	11	13	14	19	13
---	----	----	----	----	----

← 19 храним «в уме» и сравниваем

4	0	11	13	14	19	13
---	---	----	----	----	----	----

0	11	13	14	19	13
---	----	----	----	----	----

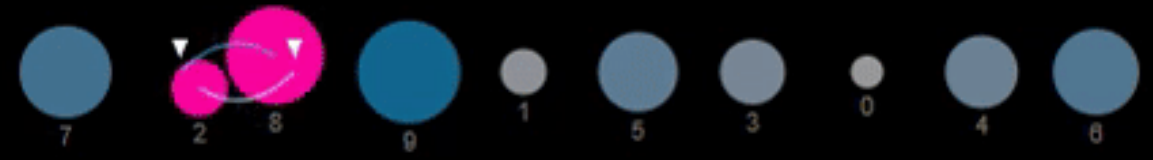
← 13 храним «в уме» и сравниваем

5	0	11	13	14	19	19
	0	11	13	14	14	19
	0	11	13	13	14	19

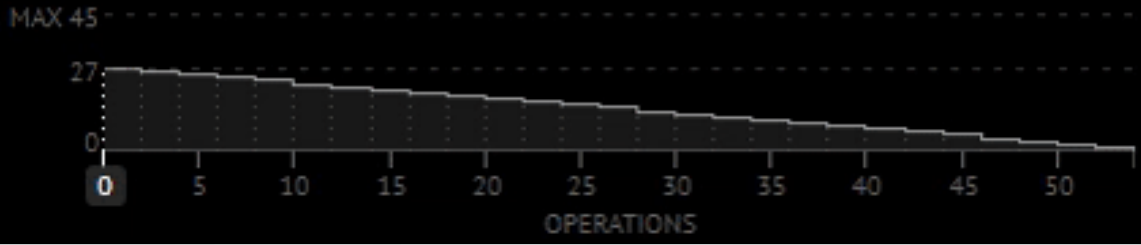
Пока меньше выполняем ...

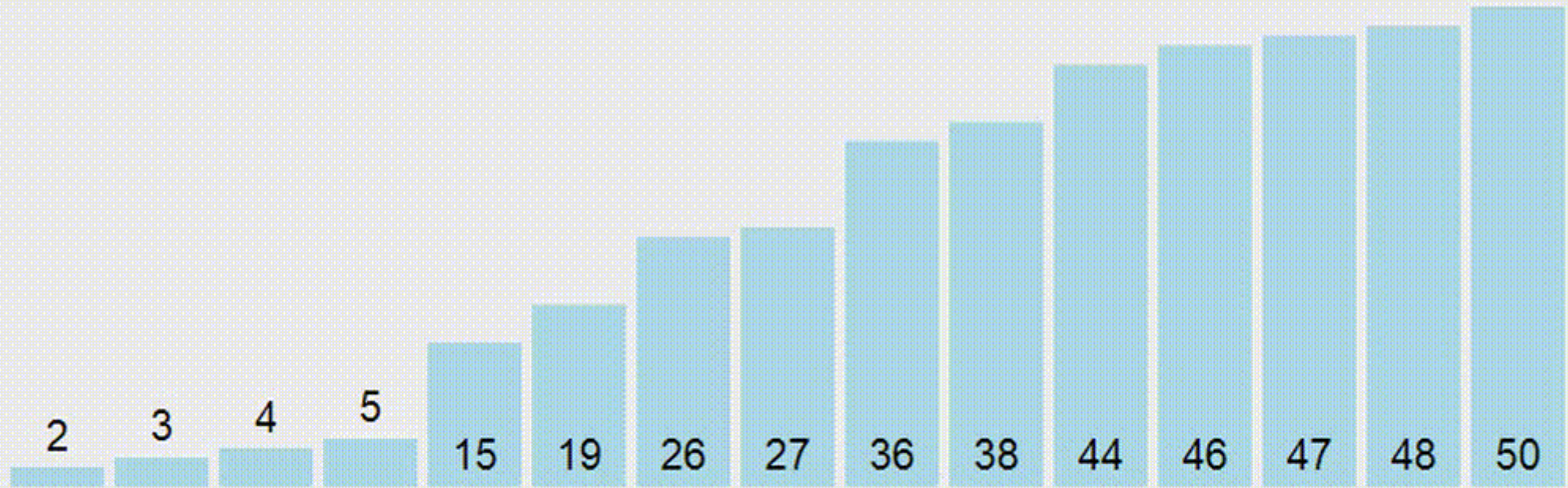
# INSERTIONSORT

10 randomly ordered elements



## INVERSIONS





```
procedure SortByInsert(a: array of
integer);
begin
  for var i:=1 to a.High do
  begin
    var x := a[i];
    var j := i - 1;
    while (j >= 0) and (x < a[j]) do
    begin
      a[j+1] := a[j];
      j -= 1;
    end;
    a[j+1] := x;
  end;
end;
```

# Standard sort

```
Sort (a) ;  
SortByDescending (a) ;
```

# Program execution time

- **To do:** calculate a time to execute the algorithm of Bubble sort

```
procedure MySort(a: array of integer);
begin
  // bubble sort
  for var i := 0 to a.High-1 do
    for var j := a.High downto i+1 do
      if a[j] < a[j-1] then
        Swap(a[j], a[j-1]);
    end;
  begin
    var a:=arrRandomInteger(20000);
    // note the time
    var t:=System.DateTime.Now;
    // run the algorithm
    MySort(a);
    // note the time of algorithm end
    var t1:=System.DateTime.Now;
    println('The time to execute the algorithm: ',t1-t);
    //t:=System.DateTime.Now;
  end.
```

- 
- Task 9



# Merging of two sorted arrays

# Merging of two sorted arrays

**Problem.** Two sorted arrays **a** and **b** are given.

Merge them into third sorted array

**Solution** (bad):

```
var c := a + b;  
Sort(c);
```

```
begin  
  var a:=arr(1,3,11,19);  
  var b:=arr(1,13,21);  
  var c:=a+b; // [1,3,11,19,1,13,21]  
  Sort(c); // [1,1,3,11,13,19,21]  
end.
```

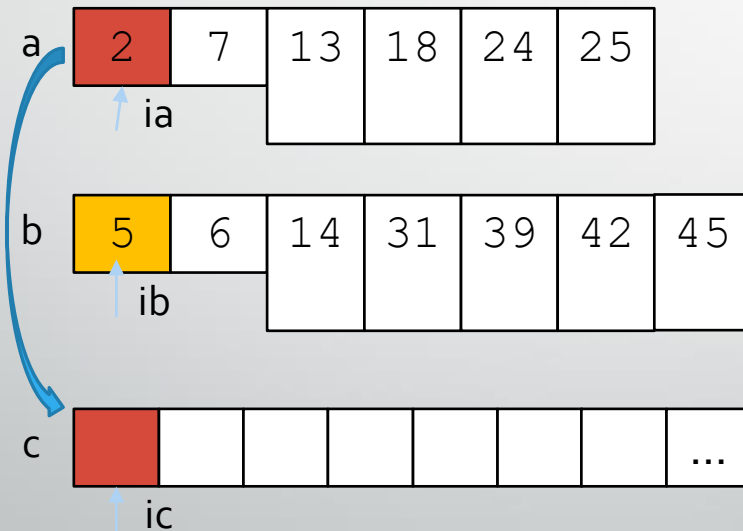


# Merging of two sorted arrays

**Problem.** Two sorted arrays **a** and **b** are given.

Merge them into third sorted array

**Solution** (partial). Let's use the counter **ia** as an index of the first element of **a** and counter **ib** – as the index of the first element of **b**. If **a[ia] < b[ib]**, then copy **a[ia]** to **c** and increase **ia**. Else copy **b[ib]** and increase **ib**.

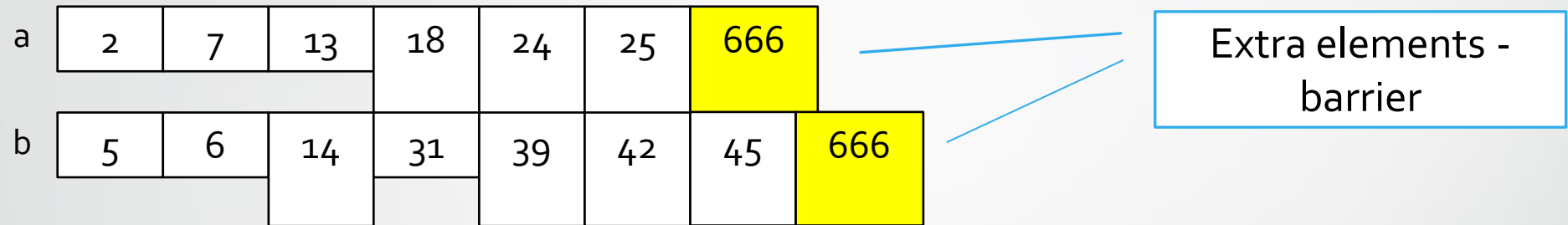


It doesn't work! Why?

```
var (ia,ib) := (0,0);
for var ic:=0 to c.Length-1 do
  if a[ia]<b[ib] then
  begin
    c[ic] := a[ia];
    ia += 1
  end
  else
  begin
    c[ic] := b[ib];
    ib += 1
  end
end;
```

# Merging of two sorted arrays

**Solution (complete).** Let's add the barrier element to the end of every array^



Previous code will be work correctly!

# Merging of two sorted arrays

```
function Merge(a, b: array of integer; n, m: integer): array of real;  
begin  
  Assert((0 < n) and (n < a.Length));  
  Assert((0 < m) and (m < b.Length));  
  a[n] := integer.MaxValue; // barrier  
  b[m] := integer.MaxValue; // barrier  
  SetLength(Result, m + n);  
  var (ia, ib) := (0, 0);  
  for var ic := 0 to n + m - 1 do  
    if a[ia] < b[ib] then  
      begin  
        Result[ic] := a[ia];  
        ia += 1;  
      end  
    else  
      begin  
        Result[ic] := b[ib];  
        ib += 1;  
      end;  
  end;
```

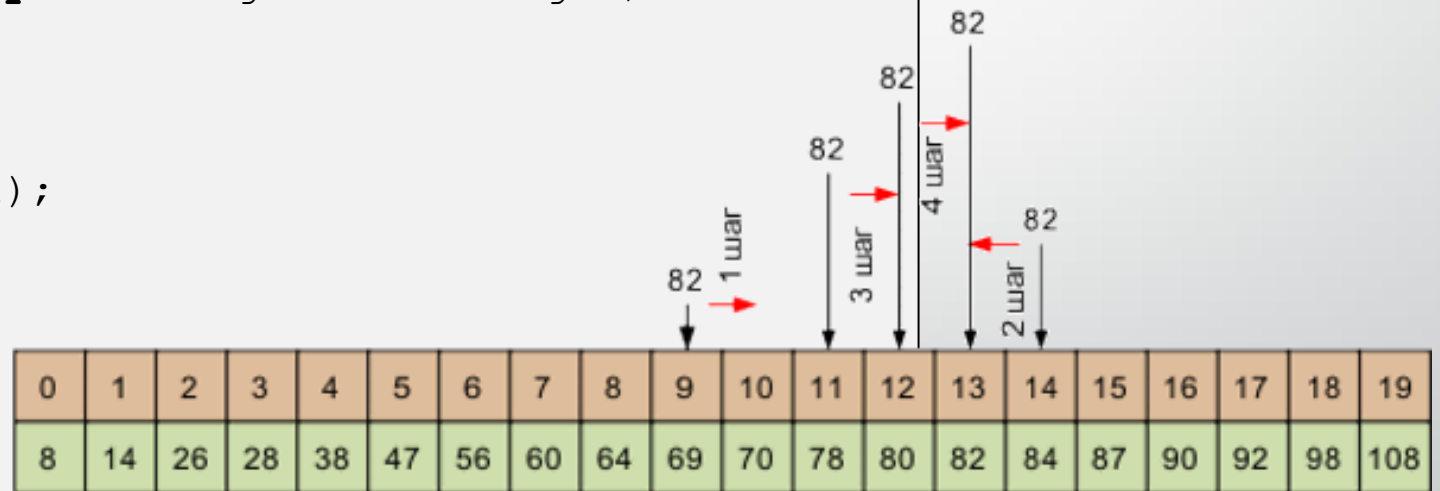
```
begin  
  var a := arr(1, 3, 11, 19);  
  var b := arr(1, 13, 21);  
  setLength(a, 5); // for extra element  
  setLength(b, 4); // for extra element  
  print(Merge(a, b, 4, 3)) // [1,1,3,11,13,19,21]  
end.
```


# Binary search in a sorted array

## Standard method `a.BinarySearch(x)`

Code of the method (не рассматривать).

```
function BinarySearch(a: array of integer; x: integer):  
integer;  
begin  
  var k: integer;  
  var (l,r) := (0, a.Length-1);  
  repeat  
    k := (l+r) div 2;  
    if x>a[k] then  
      l := k+1  
    else r := k-1;  
  until (a[k]=x) or (l>r);  
  Result := a[k]=x ? k : -1;  
end;
```



- 
- Task 10, 11



List

# List<T>

**List** is a kind of dynamic array with dynamic change of its size during program execution.

**List definition:**

real, array of integer, ...

```
var l := new List<integer>; // l.Count = 0
```

**Short definition with initialization:**

```
var L := Lst(25, -23, 47, 100, 0, 14);
```

Or:

```
var L2 := Lst(Arr(14, 172, -5, 0, 39)); // [14,172,-5,0,39]
var L3 := Lst(ArrRandom(5, -99, 99)); // [0,-6,2,-81,10]
var L4 := Lst(L3); // there is one element there in L4 list,
it is list [[0,-6,2,-81,10]]
```

# List<T>

**Adding an element to the end of a list:**

```
var L := Lst(Arr(14, 172, -5, 0, 39));  
L.Add(5); // List is expanded  
L.Add(3);  
L.Add(4);  
L += 8; // a synonym of l.Add(8)  
println(L); // [14,172,-5,0,39,5,3,4,8]
```

**Iterating over the list:**

```
for var i:=0 to L.Count-1 do  
    Print(L[i]);  
foreach var x in L do  
    Print(x);
```



# List operations and methods

Operations with list:

```
var L := Lst(5,2,3);  
Print(2 in L); // True  
Print(2 * L); // [5,2,3,5,2,3]  
L.Print; // 5 2 3  
Print(L + Lst(7,6,8)); // 5 2 3 7 6 8
```

**Methods of list:**

```
L.Insert(ind,x); // insert x by index ind  
L.RemoveAt(ind); // delete element by index ind  
L.RemoveRange(ind,count) // delete a diapason of elements  
L.RemoveAll(x -> x.IsOdd); // delete elements by condition  
L.IndexOf(3); // index of a first element or -1  
L.FindIndex(x -> x > 4); // index of a first element or -1  
L.Clear;  
L.Reverse;  
L.Sort;
```

# Examples

**Problem.** An array of **N** integers is given. Insert all even elements of the array into **L1**, and all odd elements into **L2**

**Solution.**

```
begin
  var a := arrrandominteger(10, 5, 25);
  println(a); // [17,25,8,17,21,9,19,22,19,24]
  var L1 := new List<integer>;
  var L2 := new List<integer>;

  foreach var x in a do
    if x.IsEven then
      L1 += x
    else L2 += x;
  L1.println; // 8 22 24
  L2.println; // 17 25 17 21 9 19 19
end.
```

# Insertion and deletion in an array using lists methods

**Problem 1.** An array of  $N$  integers is given. It's necessary to insert an element  $x$  on  $k$ -th index,  $k \leq N$ .


```
begin
  var a := arr(5, 12, 1, 3, 11, 19);
  var x := ReadInteger ('enter a number to insert');
  var k := ReadInteger ('enter an order number');
  a := a[:k] + Arr(x) + a[k:];
  print(a) // [5,12,3,1,3,11,19]
end.
```

**With lists:**  
**L.Insert**(k, x);

**Problem 2.** An array of  $N$  integers is given. It's necessary to delete an element with index  $k$ ,  $k < N$ .

```
begin
  var a := arr(5, 12, 1, 3, 11, 19);
  var k := ReadInteger ('enter an order number');
  a := a[:k] + a[k+1:];
  print(a) // [5,12,3,11,19]
end.
```

**With lists:**  
**L.RemoveAt**(k);

- 
- Task 12, 13, 14



Q & A