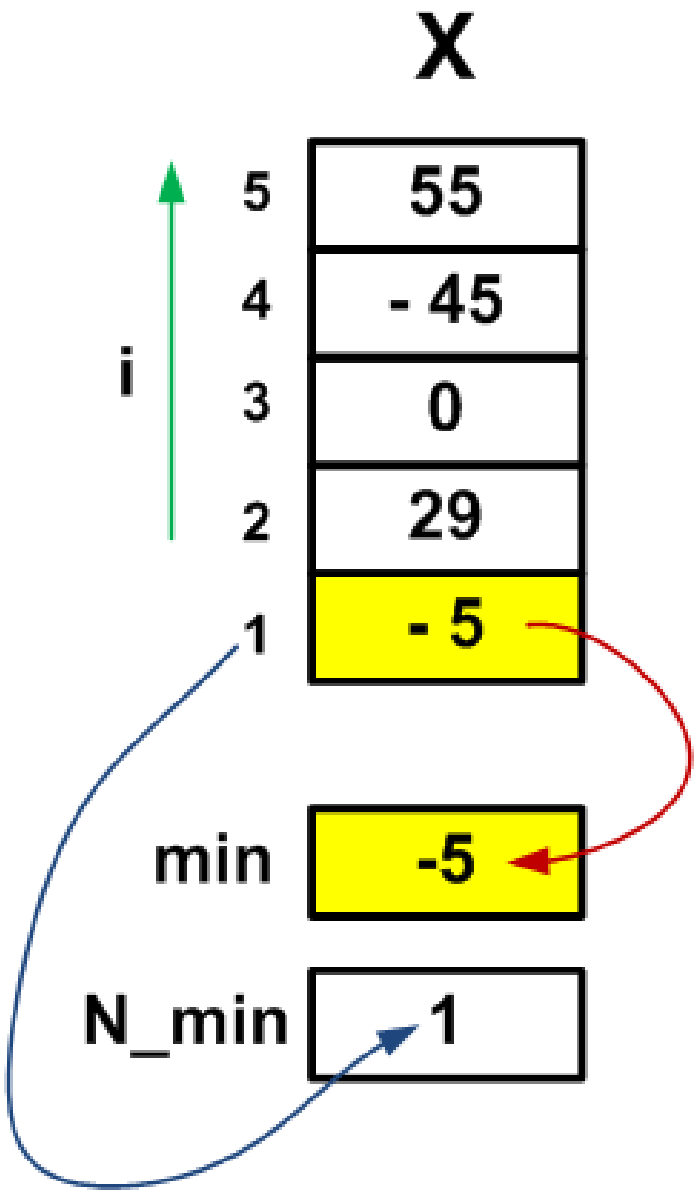
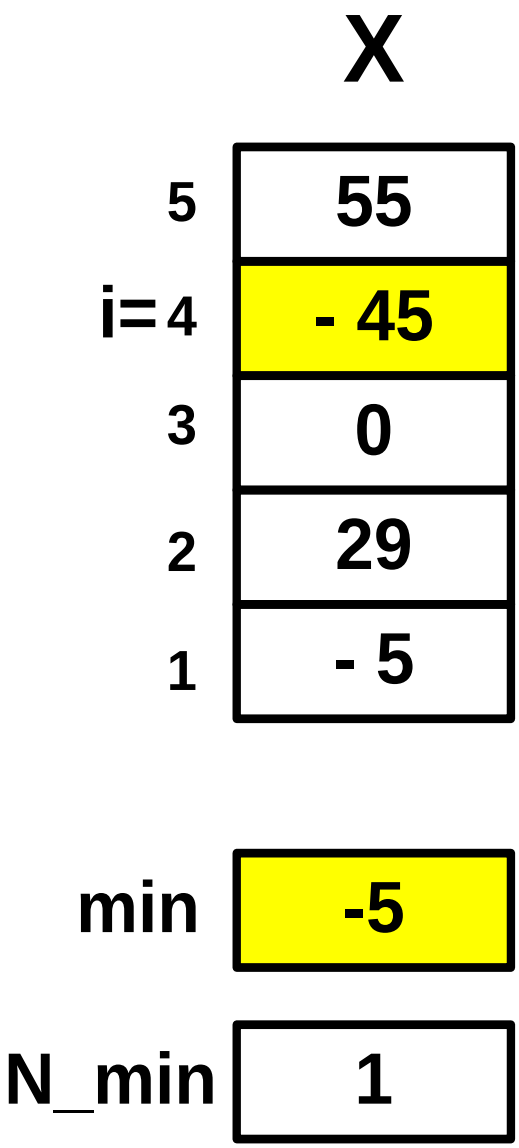
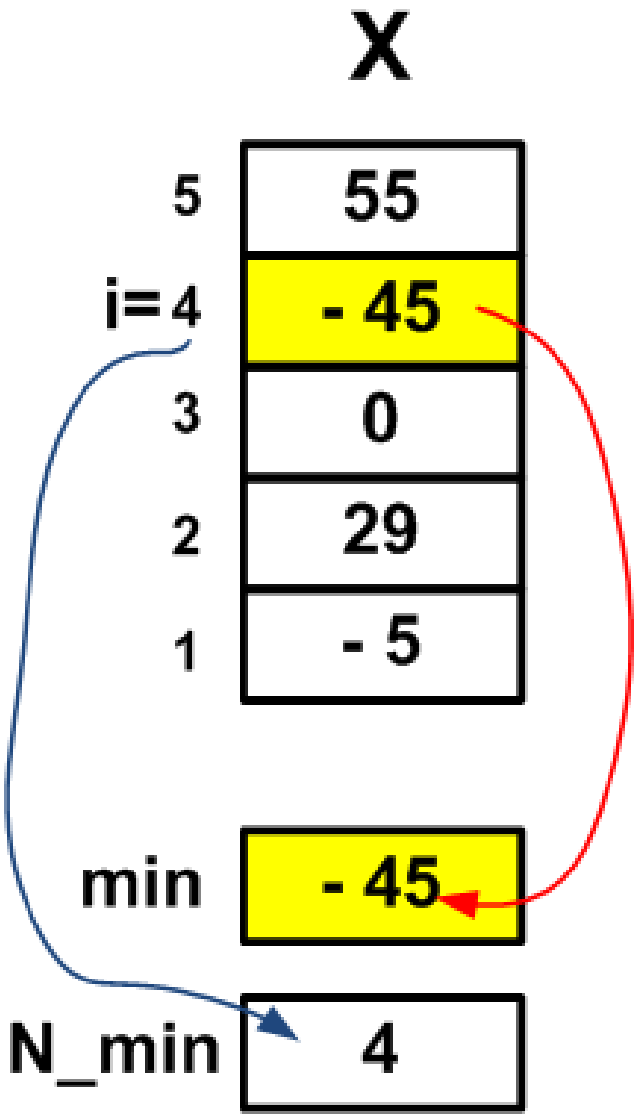
Некоторые алгоритмы для массивов.

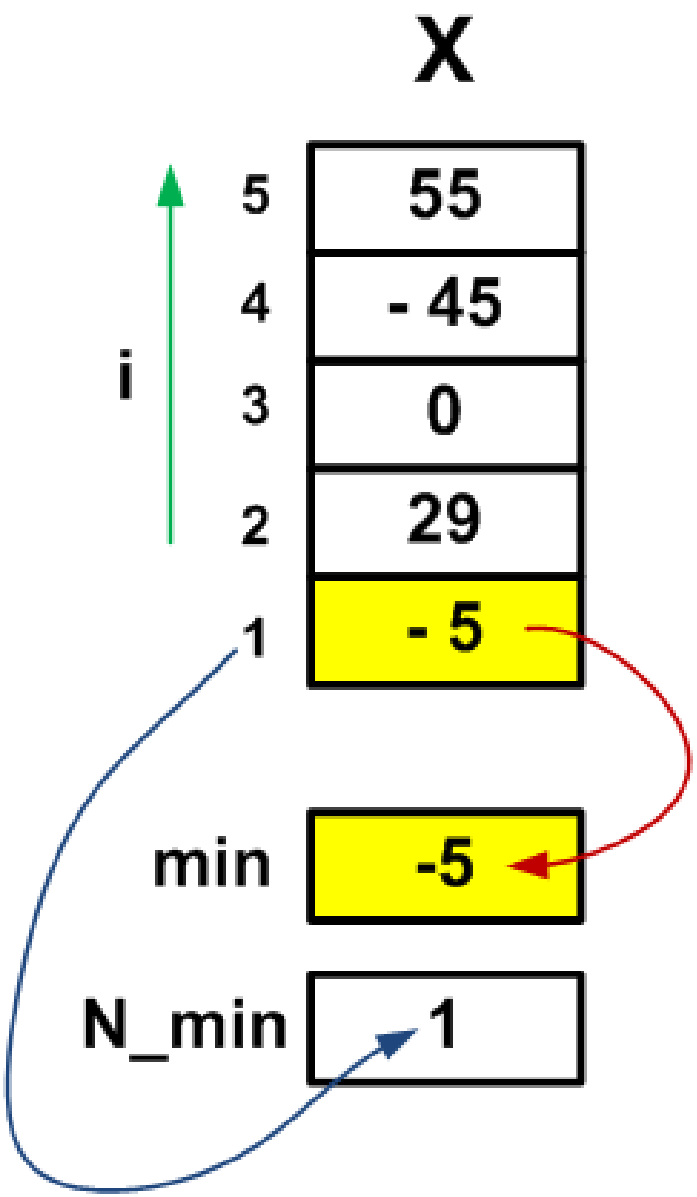
Методы сортировки.

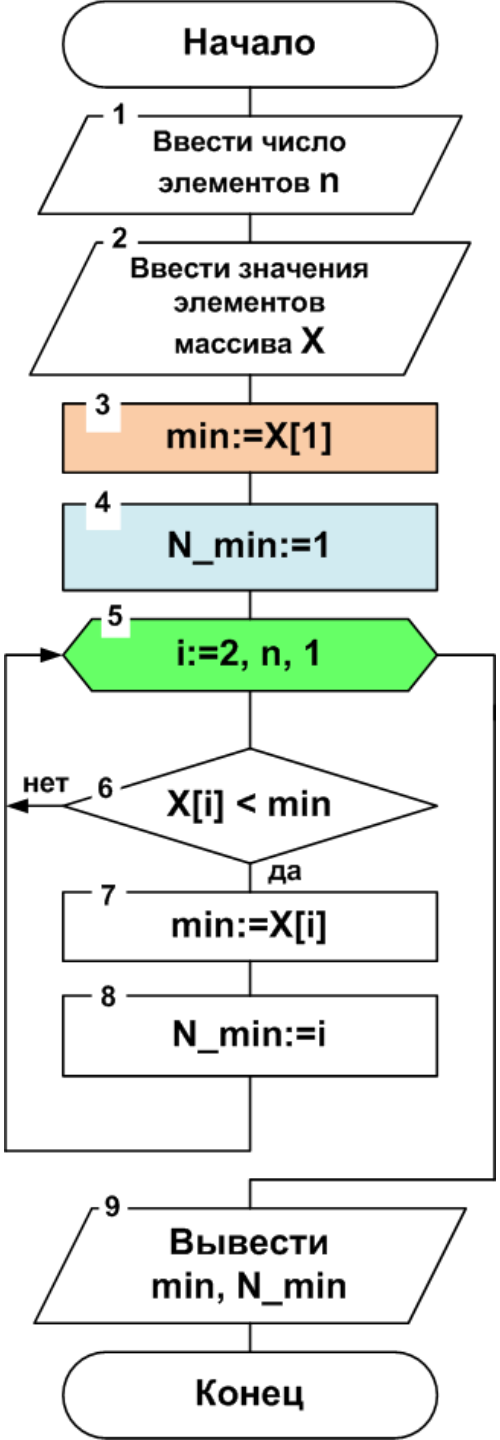
Пример 1. Вывести минимальный элемент массива X и его номер (индекс).

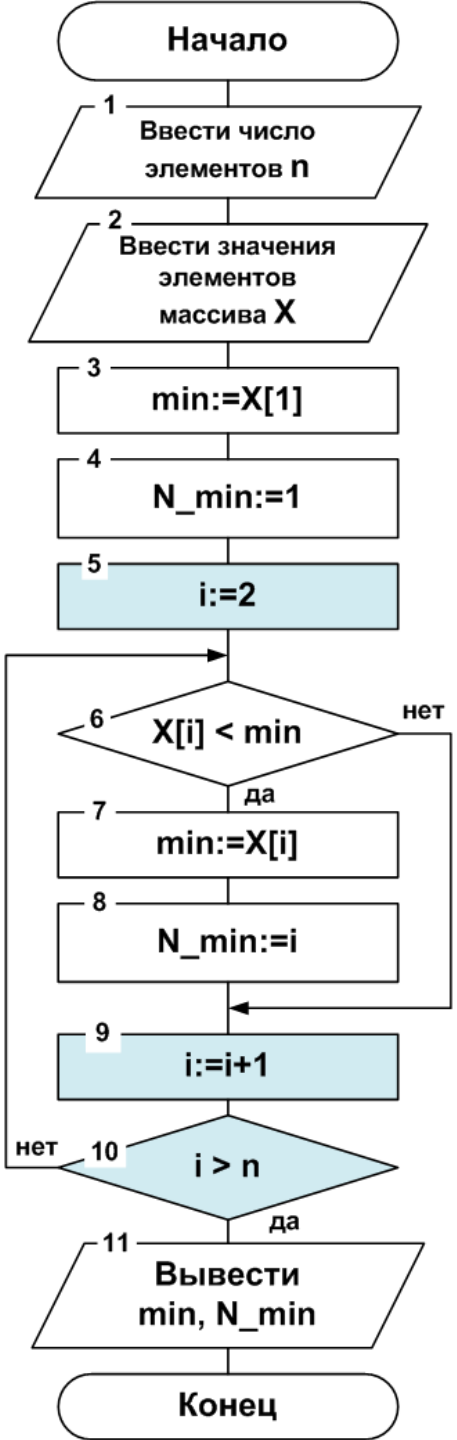


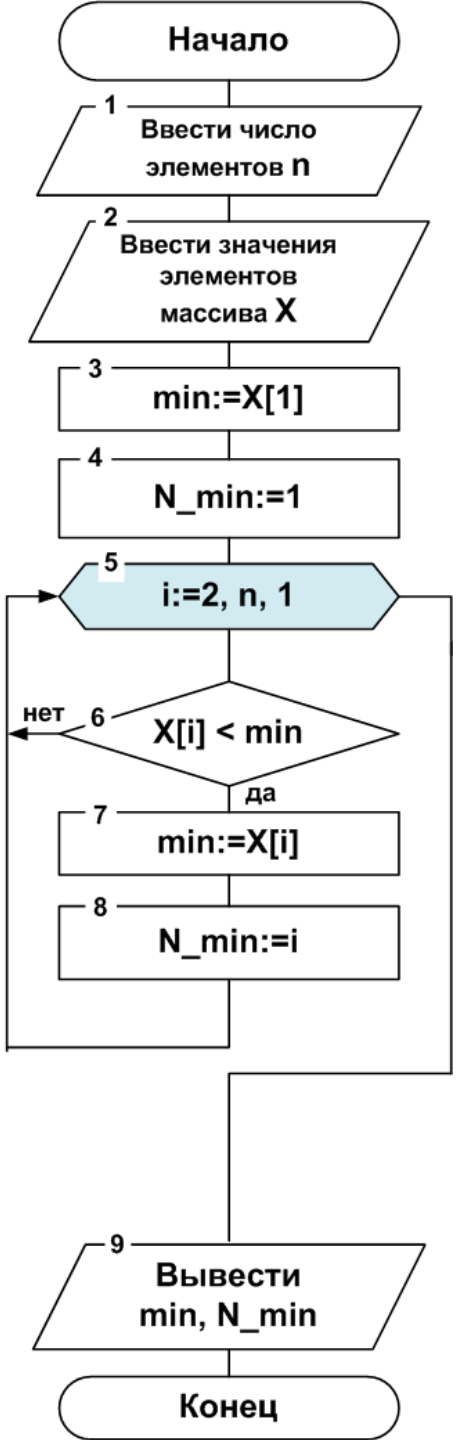

X
5
55
i=4
- 45
3
0
2
29
1
- 5
min
-5
N_min
1


X
5
55
i=4
- 45
3
0
2
29
1
- 5
min
- 45
N_min
4


X
i
5
55
4
- 45
3
0
2
29
1
- 5
min
-5
N_min
1


Начало
1
Ввести число элементов n
2
Ввести значения элементов массива X
3
min:=X[1]
4
N_min:=1
5
i:=2, n, 1
нет
6
X[i] < min
да
7
min:=X[i]
8
N_min:=i
9
Вывести min, N_min
Конец





Изображение цикла for на схемах алгоритмов

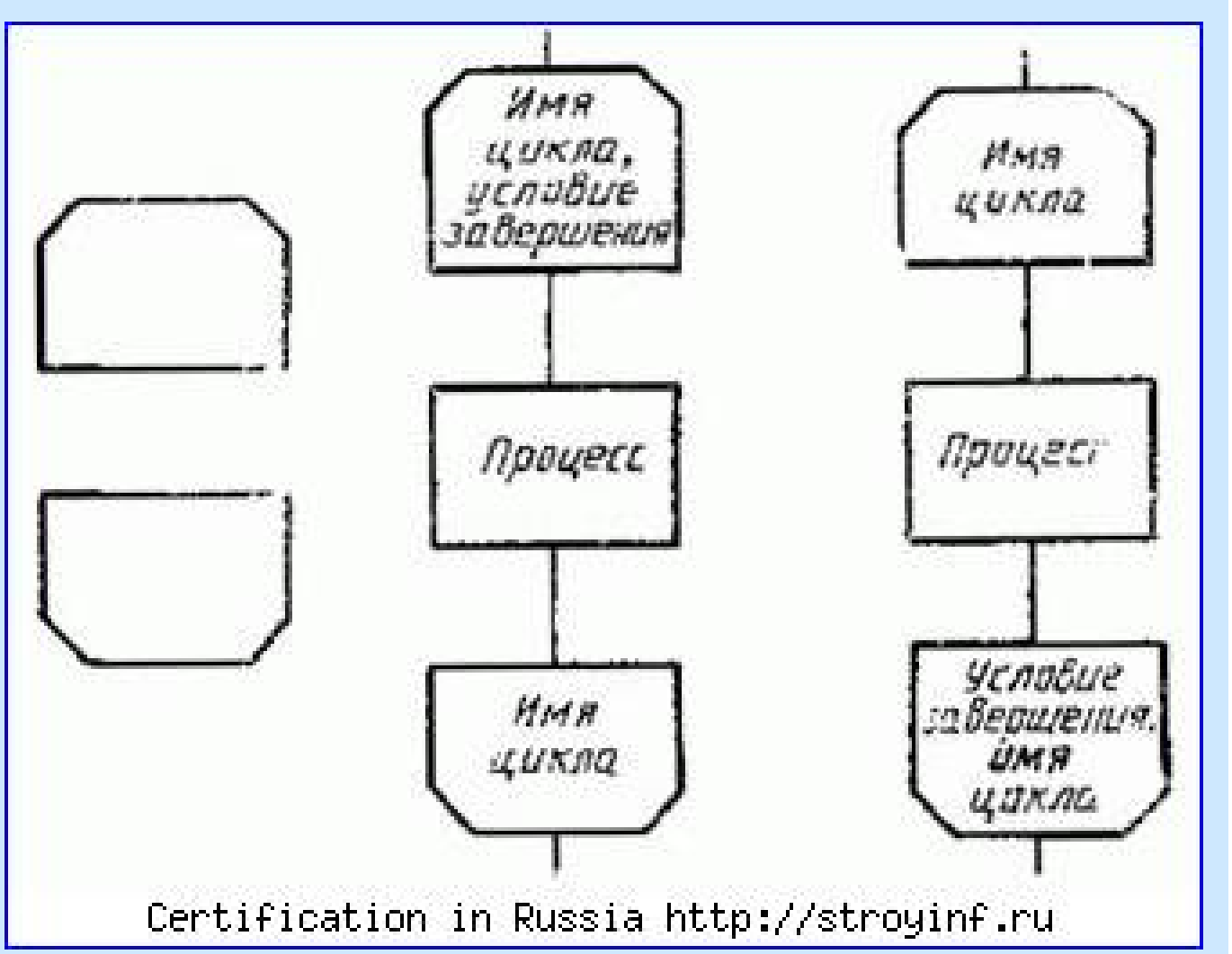
3.2.2.3. Подготовка (ГОСТ)

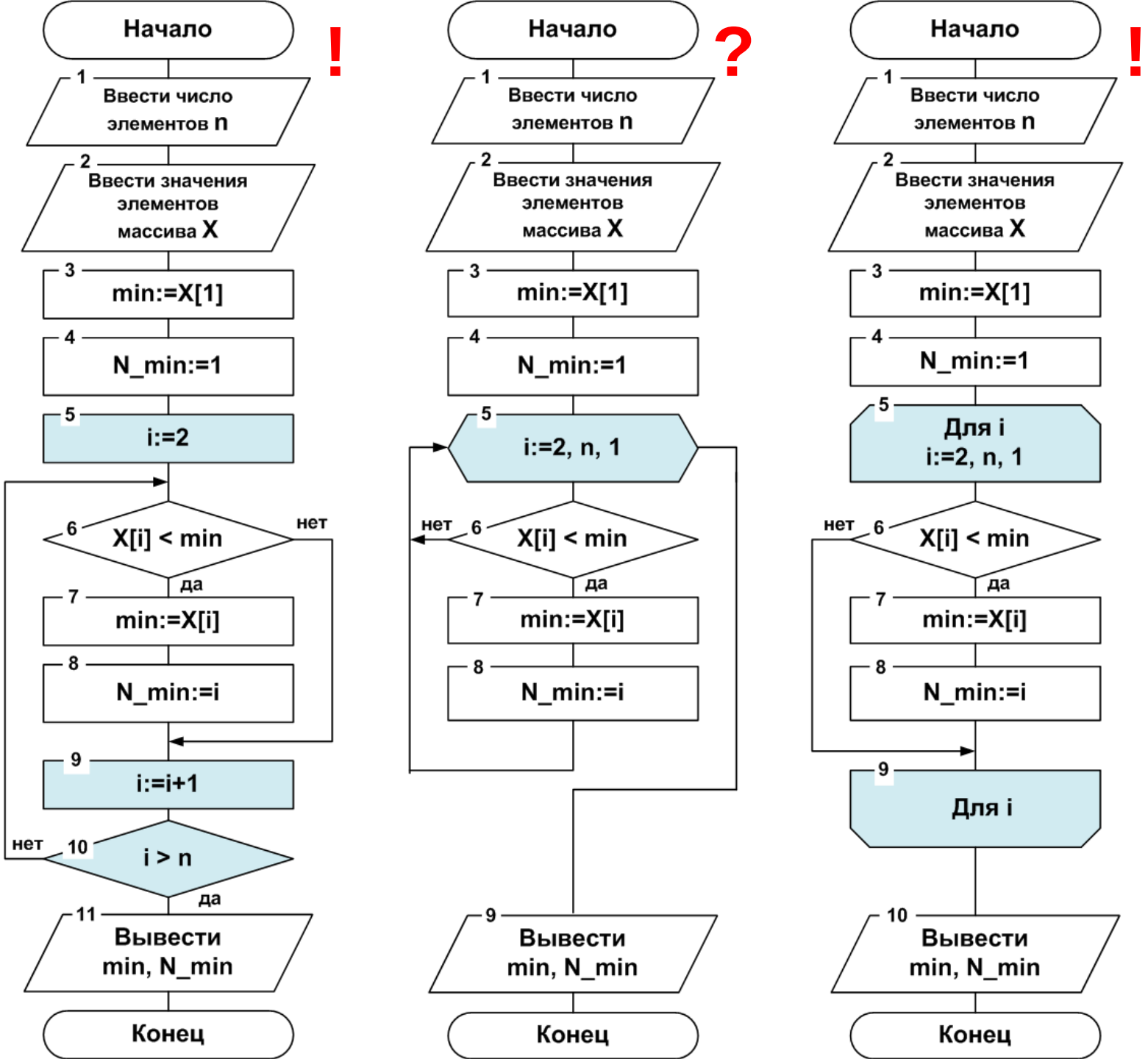
Символ отображает модификацию команды или группы команд с целью воздействия на некоторую последующую функцию (установка переключателя, модификация индексного регистра или инициализация программы).

ГОСТ 19.701-90 (ИСО 5807-85)
Единая система программной документации
СХЕМЫ АЛГОРИТМОВ, ПРОГРАММ, ДАННЫХ И СИСТЕМ
Условные обозначения и правила выполнения

3.2.2.6. Граница цикла

Символ, состоящий из двух частей, отображает начало и конец цикла. Обе части символа имеют один и тот же идентификатор. Условия для инициализации, приращения, завершения и т.д. помещаются внутри символа в начале или в конце в зависимости от расположения операции, проверяющей условие.




Начало
1
Ввести число элементов n
2
Ввести значения элементов массива X
3
min:=X[1]
4
N_min:=1
5
i:=2
6
X[i] < min
нет
да
7
min:=X[i]
8
N_min:=i
9
i:=i+1
нет
10
i > n
да
11
Вывести min, N_min
Конец
!
Начало
1
Ввести число элементов n
2
Ввести значения элементов массива X
3
min:=X[1]
4
N_min:=1
5
i:=2, n, 1
нет
6
X[i] < min
да
7
min:=X[i]
8
N_min:=i
9
Вывести min, N_min
Конец
?
Начало
1
Ввести число элементов n
2
Ввести значения элементов массива X
3
min:=X[1]
4
N_min:=1
5
Для i
i:=2, n, 1
нет
6
X[i] < min
да
7
min:=X[i]
8
N_min:=i
9
Для i
10
Вывести min, N_min
Конец
!

Методы сортировки

Дан массив {A[0], A[1], …, A[n-1]}.
Переставить элементы массива так, чтобы выполнялось условие:
A[0] ≤ A[1] ≤ A[2] ≤… ≤ A[n-1]. (*)

Идея → метод → алгоритм →программа.

Пузырьковая сортировка

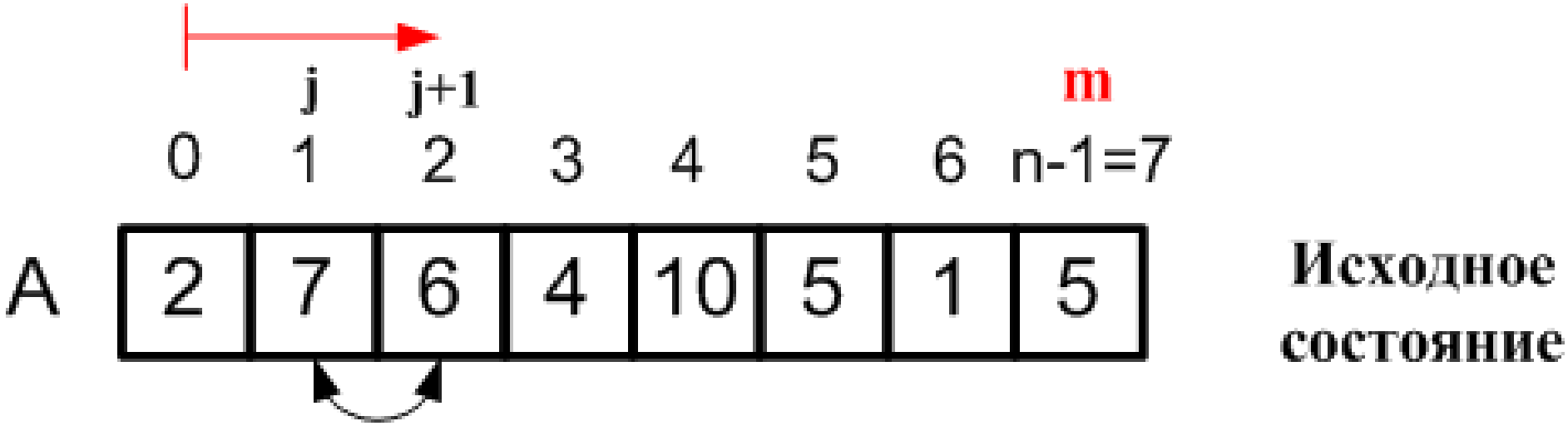
Идея: Анализируем соотношение (*) и замечаем, что в паре соседних элементов левый не больше правого.
Если в паре левый больше правого, их надо переставить.

Пузырьковая сортировка (вариант1)

A – массив, состоящий из n=8 элементов типа integer

Первый просмотр массива A

```
i:=1; { i – номер просмотра}
m:=n-1; {m - граница просмотра}
for j:= 0 to m-1 do { от 0 до = 6 }
```

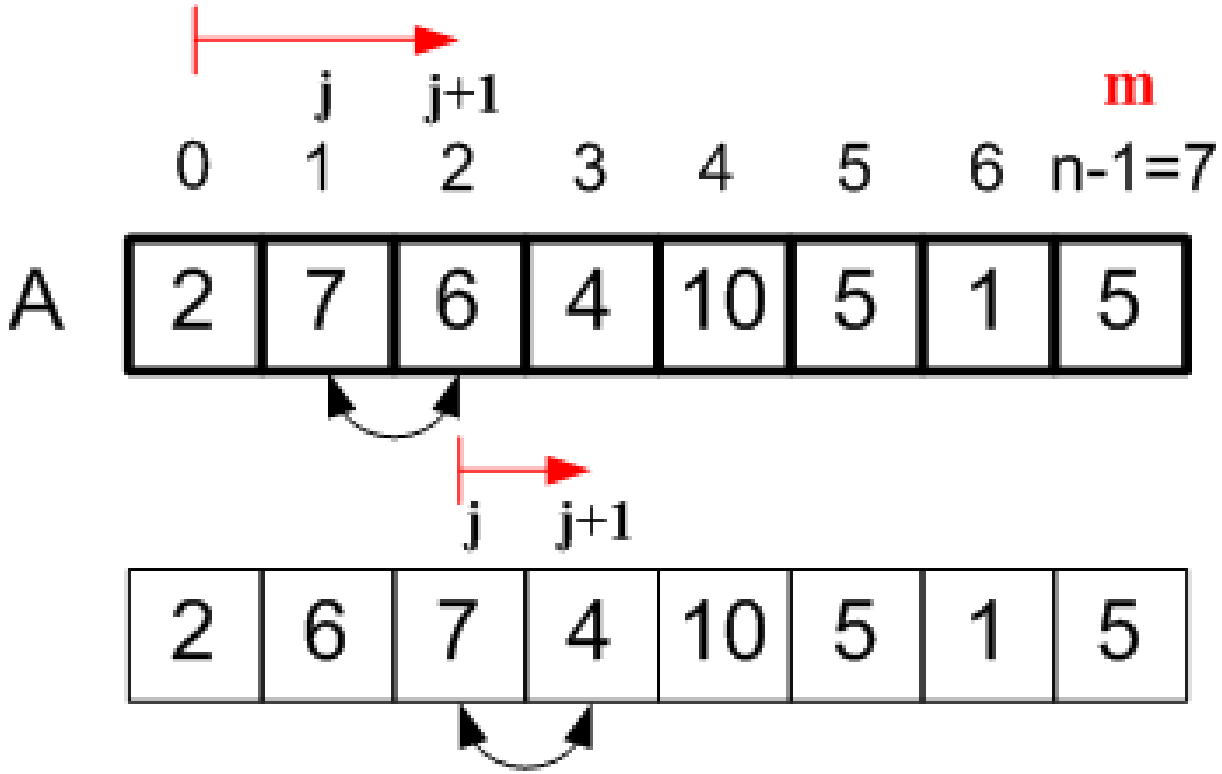


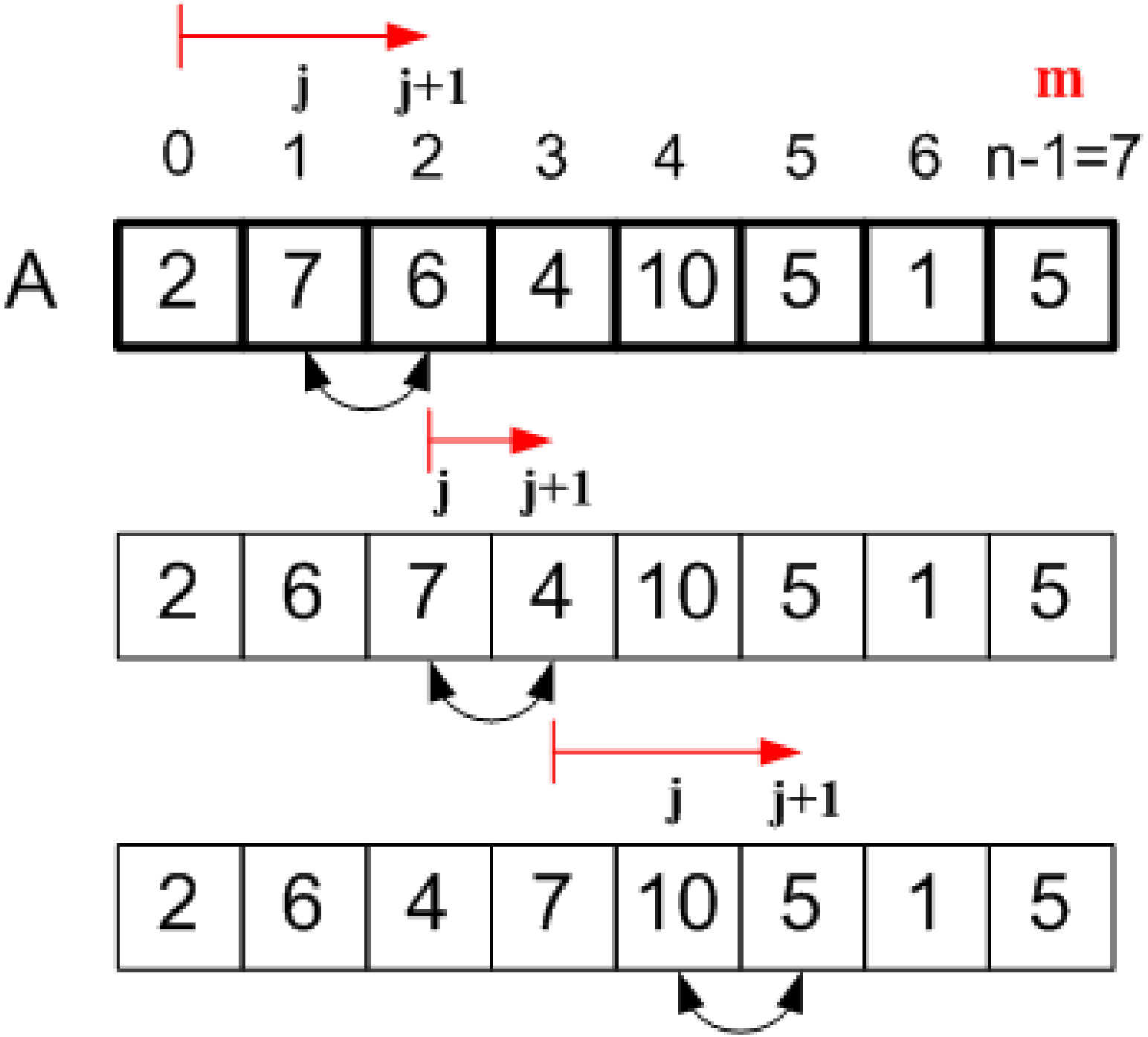
Исходное состояние

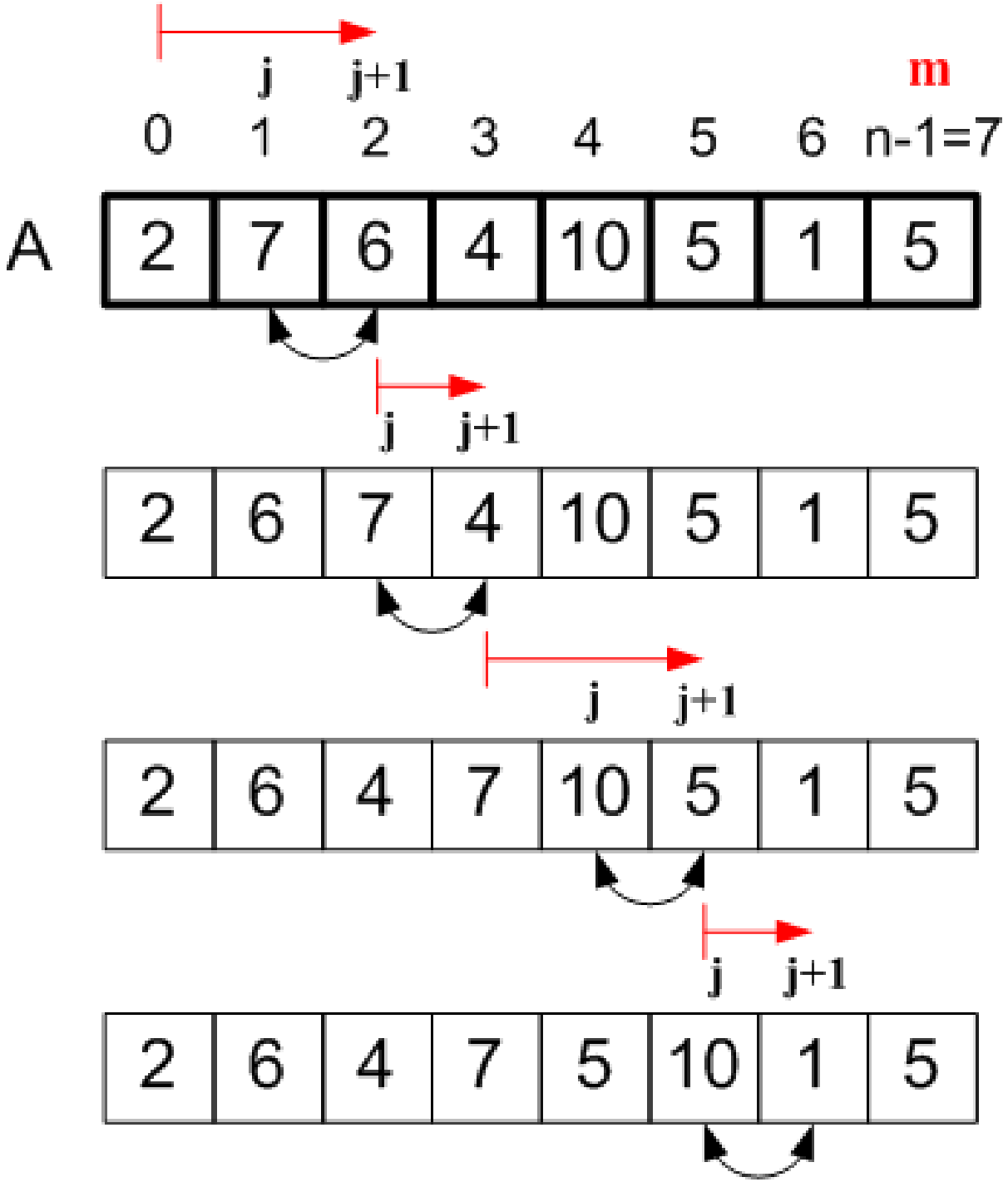
A – массив, состоящий из n=8 элементов типа integer

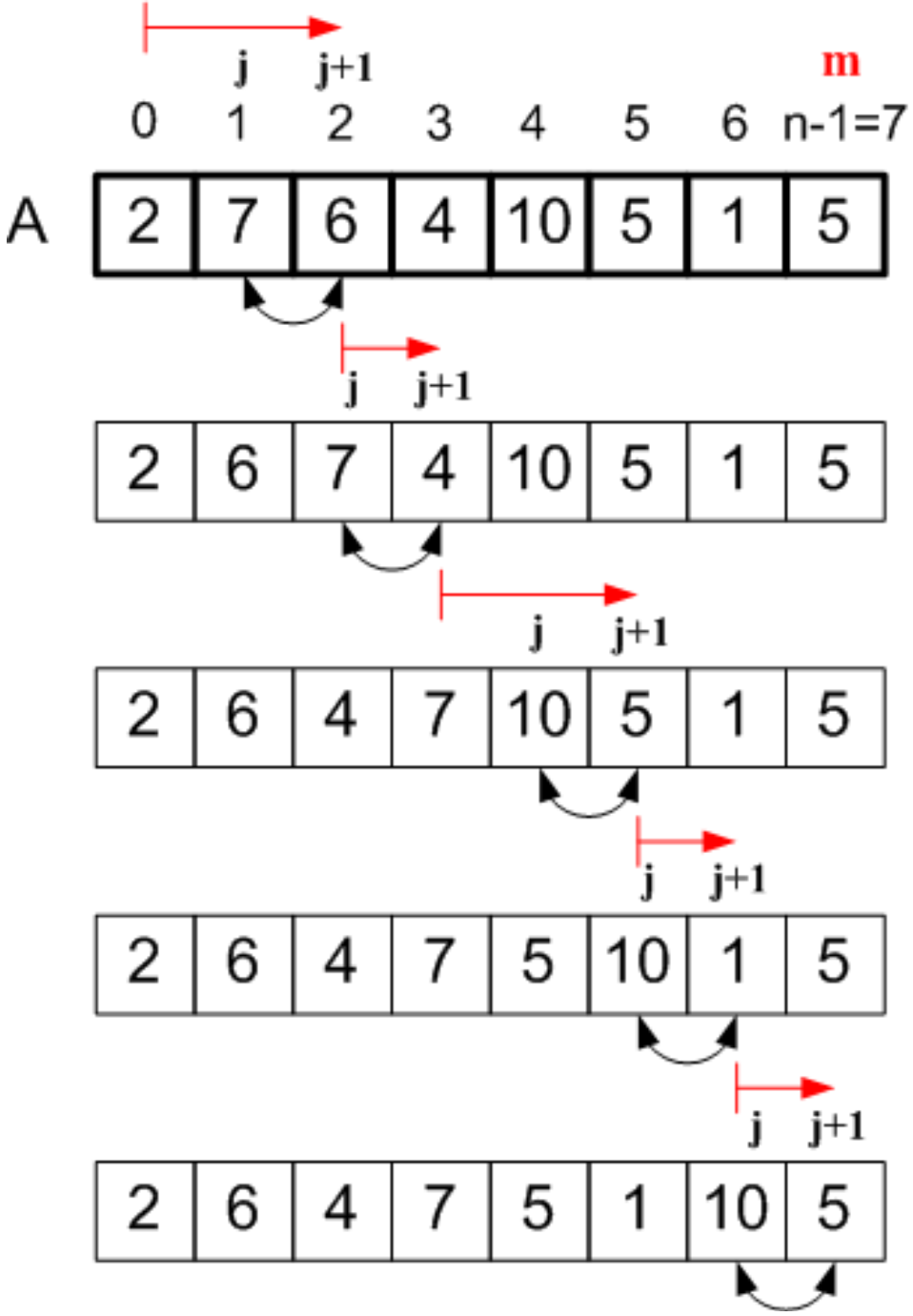
Первый просмотр массива A

```
i:=1; { i – номер просмотра}
m:=n-1; {m - граница просмотра}
for j:= 0 to m-1 do { от 0 до = 6 }
```




j
j+1
m
0
1
2
3
4
5
6
n-1=7
A
2
7
6
4
10
5
1
5
j
j+1
2
6
7
4
10
5
1
5
j
j+1
2
6
4
7
10
5
1
5

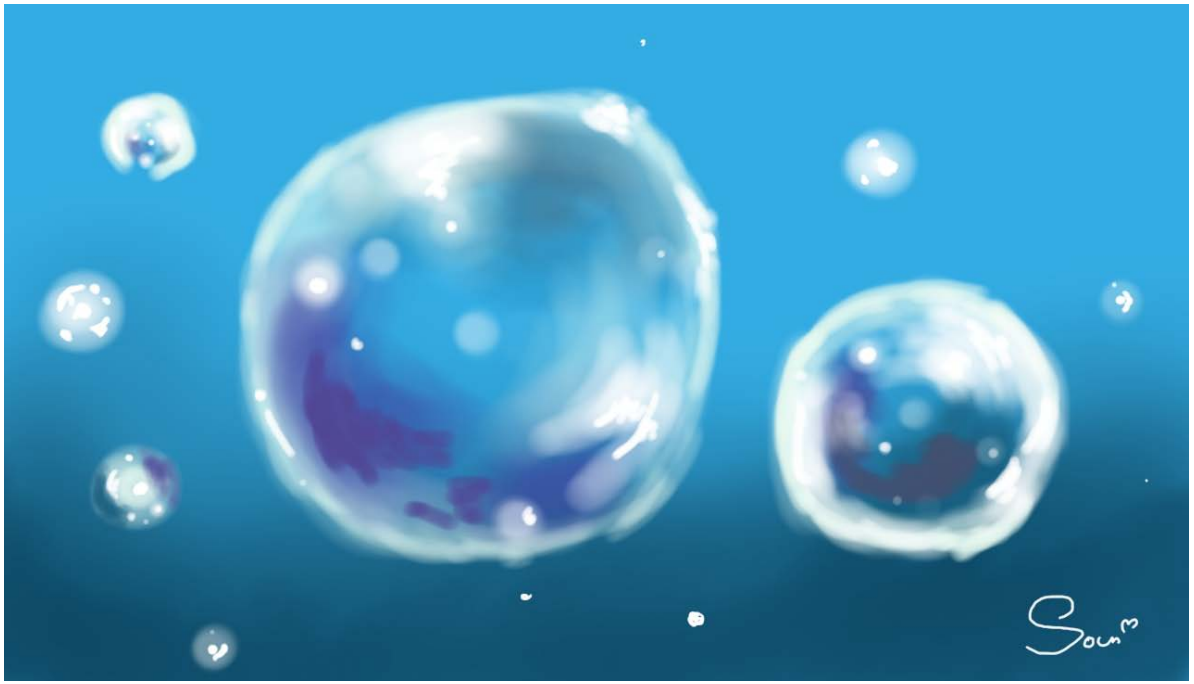

j
j+1
m
0
1
2
3
4
5
6
n-1=7
A
2
7
6
4
10
5
1
5
j
j+1
2
6
7
4
10
5
1
5
j
j+1
2
6
4
7
10
5
1
5
j
j+1
2
6
4
7
5
10
1
5


j
j+1
m
0 1 2 3 4 5 6 n-1=7
A
2 7 6 4 10 5 1 5
j
j+1
2 6 7 4 10 5 1 5
j
j+1
2 6 4 7 10 5 1 5
j
j+1
2 6 4 7 5 10 1 5
j
j+1
2 6 4 7 5 1 10 5

Результат первого просмотра

2	6	4	7	5	1	5	10

Следующий (первый) по возрастанию элемент (пузырек) нашел свое место m (всплыл), значит в очередном просмотре массива он не участвует: m должно стать равным 6

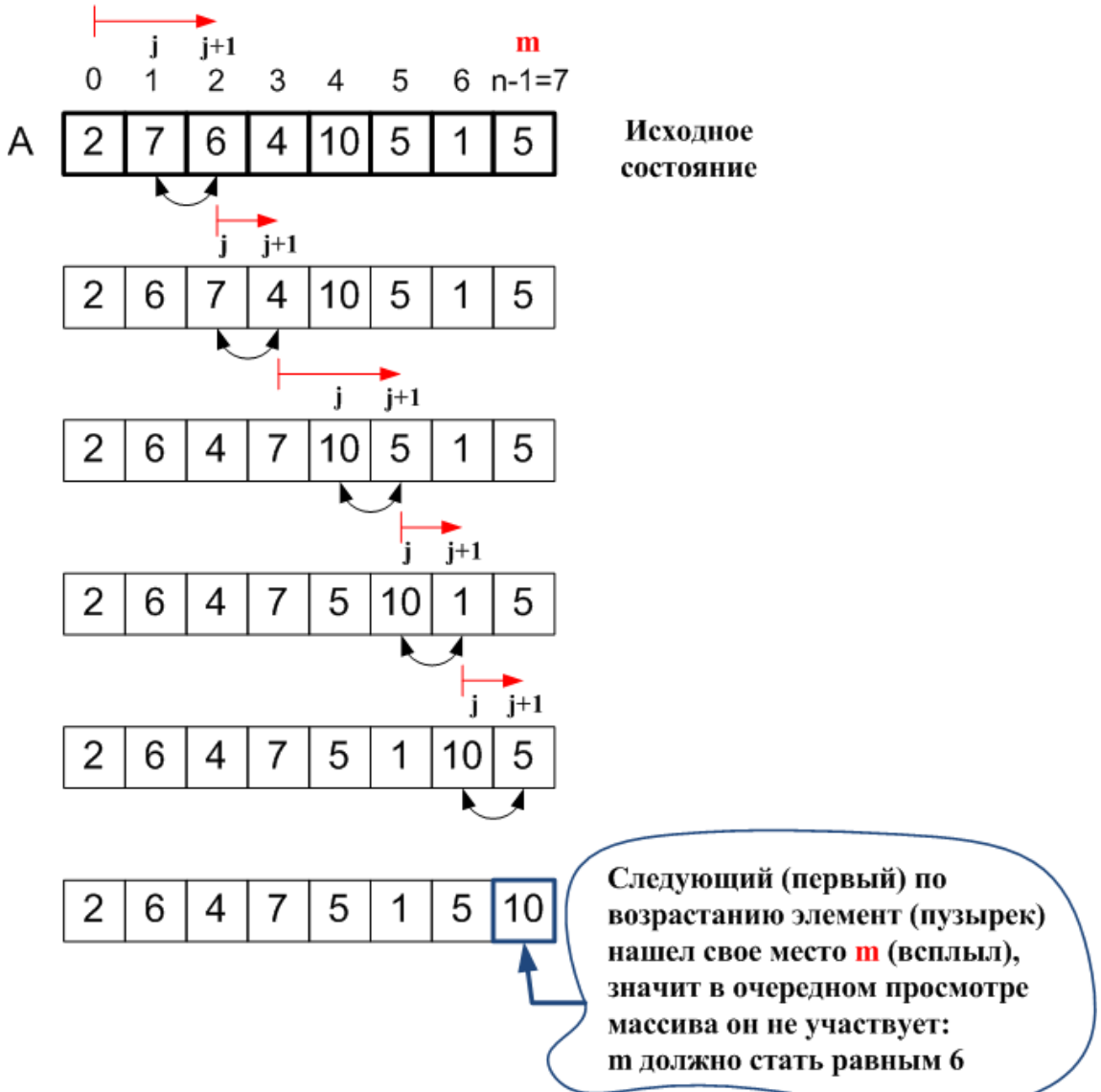


Самый большой пузырек всплыл!

Пузырьковая сортировка (вариант 1)

A – массив, состоящий из n=8 элементов типа integer

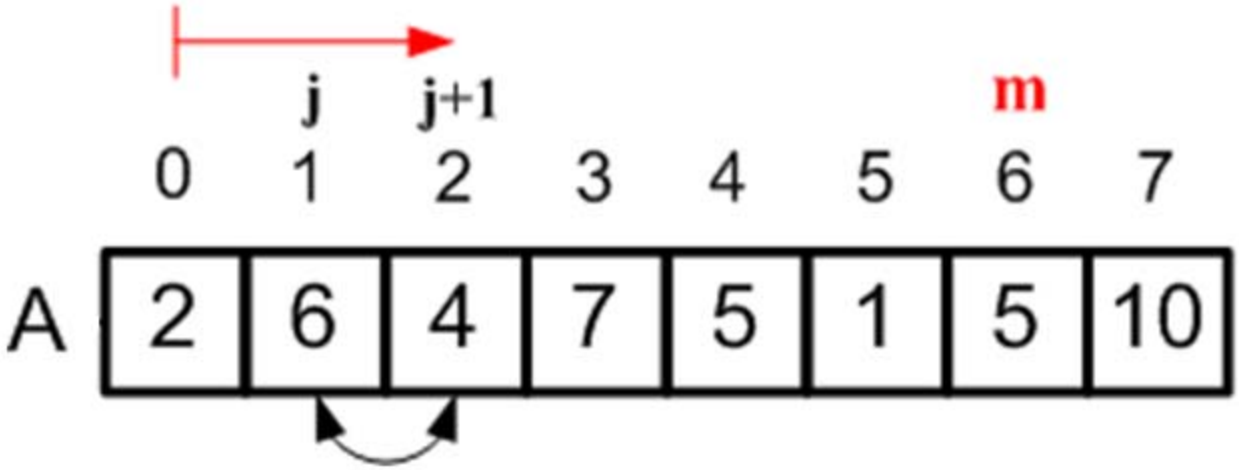
Первый просмотр массива A
i:=1; { i – номер просмотра}
m:=n-1; {m - граница просмотра}
for j:= 0 to m-1 do { от 0 до = 6 }



Итак, первый просмотр (i = 1) окончен, но не все элементы массива A упорядочены. Признак неупорядоченности – были перестановки

Второй просмотр массива A

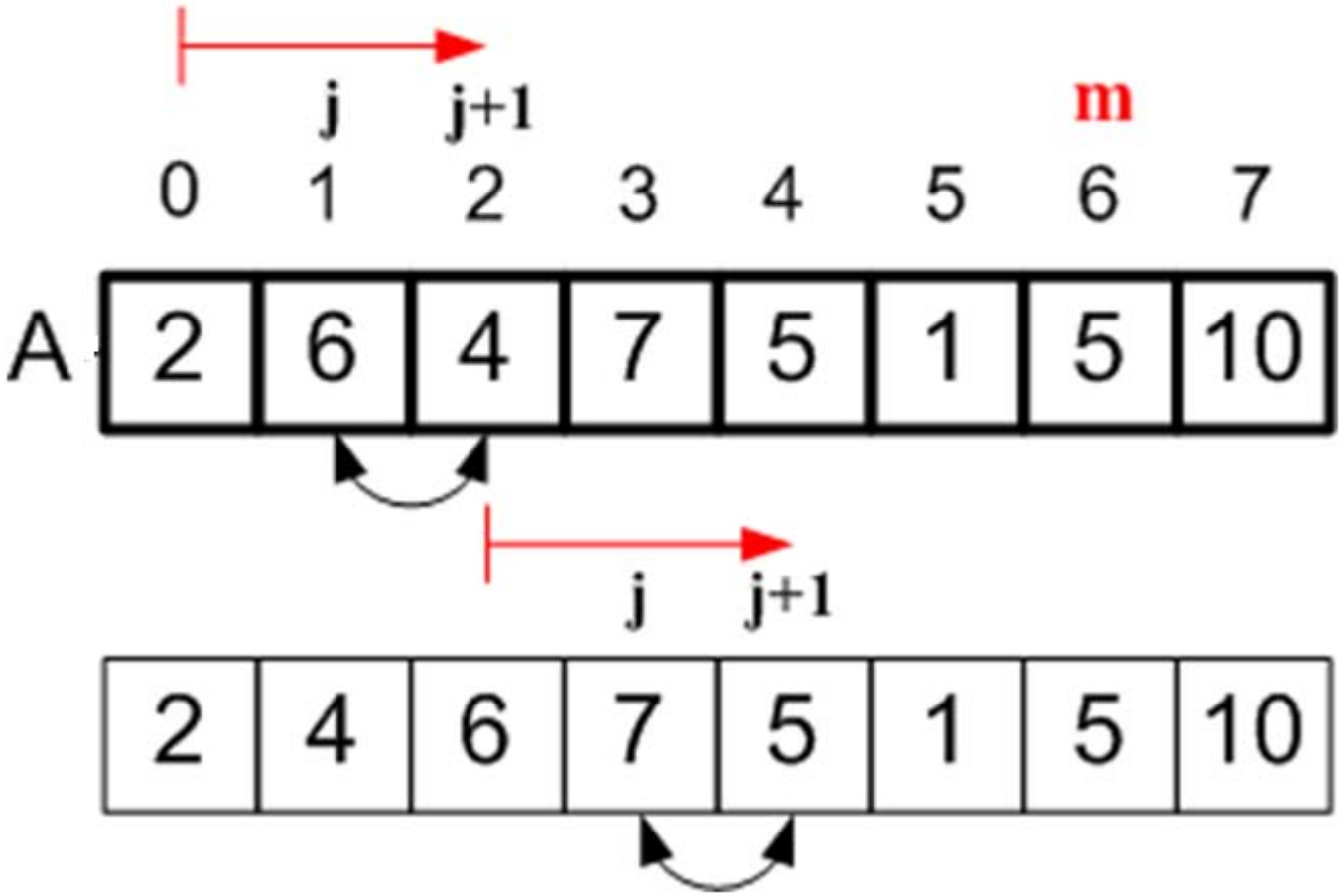
```
i:=2; { i – номер просмотра}
m:=m-1; {граница просмотра m=6}
for j:= 0 to m-1 do  {от 0 до  5}
```

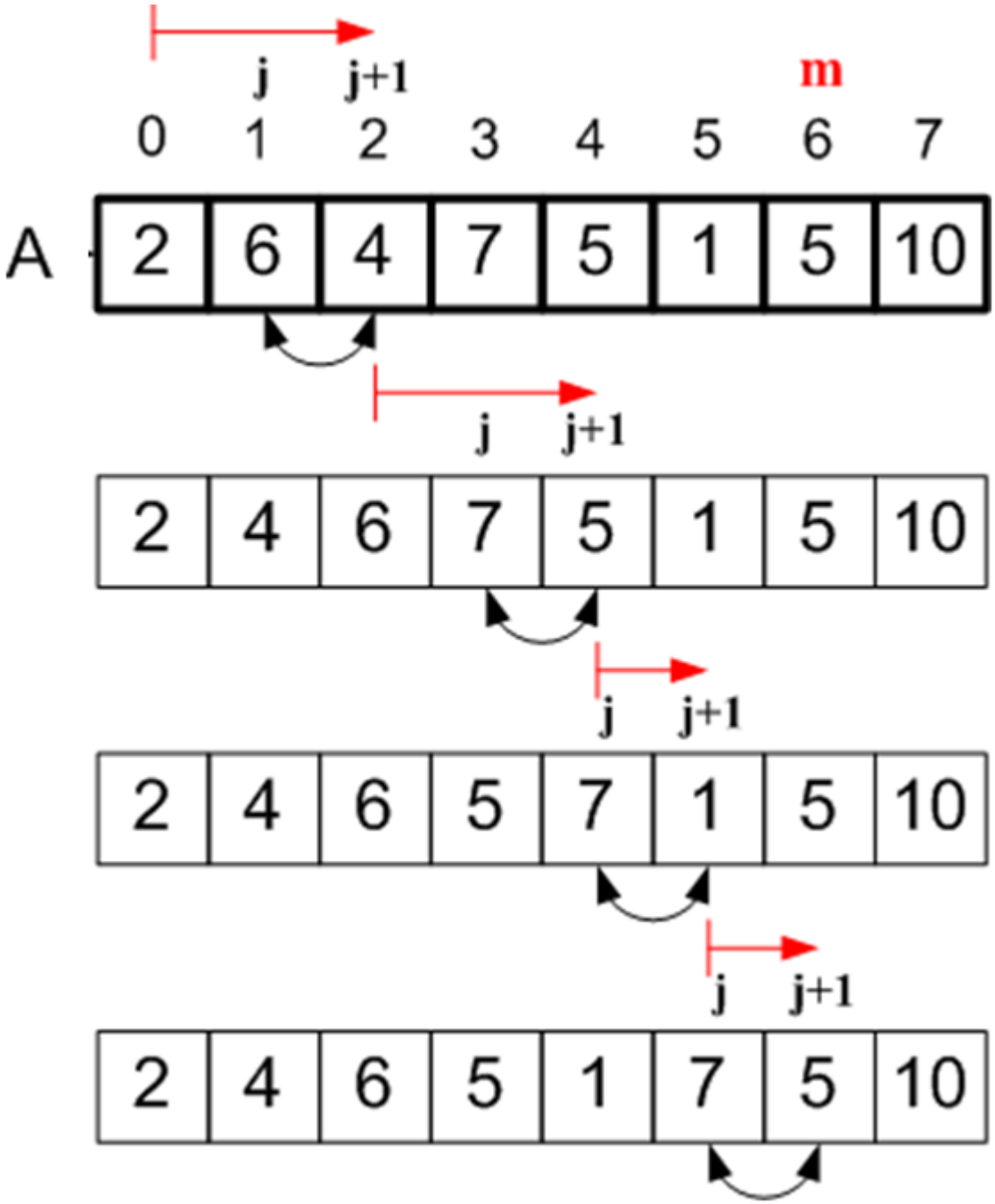


Исходное состояние

Второй просмотр массива A

```
i:=2; { i – номер просмотра}
m:=m-1; {граница просмотра m=6}
for j:= 0 to m-1 do  {от 0 до  5}
```

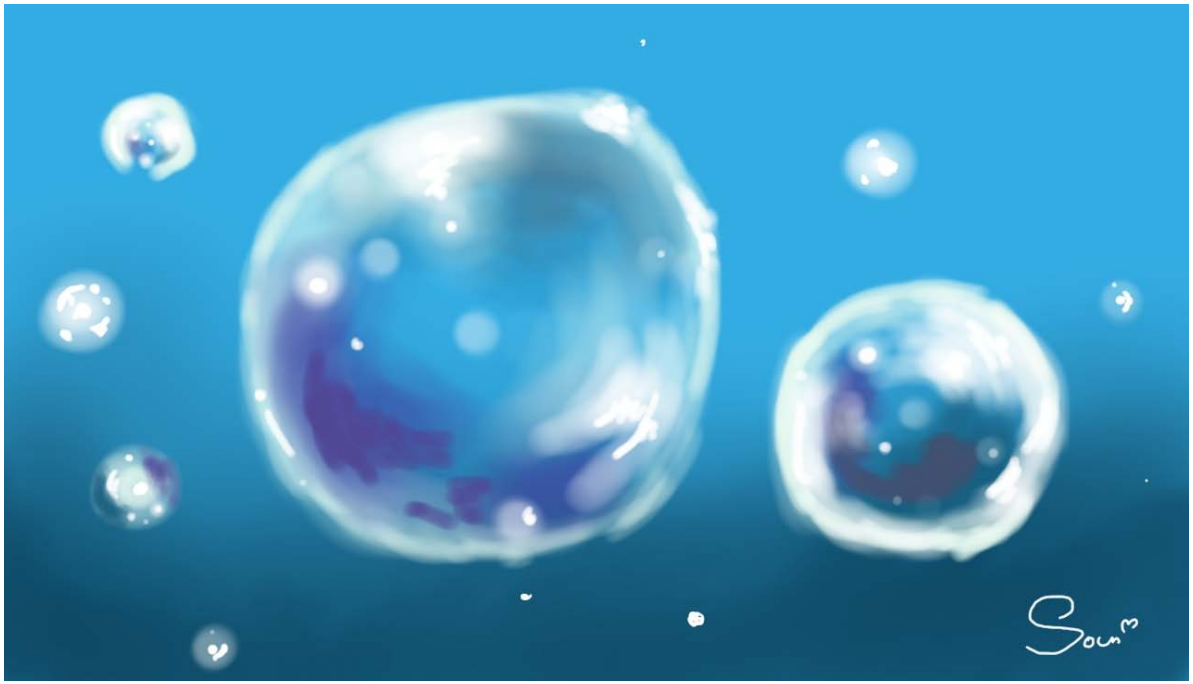



j
j+1
m
0 1 2 3 4 5 6 7
A
2 6 4 7 5 1 5 10
j
j+1
2 4 6 7 5 1 5 10
j
j+1
2 4 6 5 7 1 5 10
j
j+1
2 4 6 5 1 7 5 10

Результат второго просмотра

2	4	6	5	1	5	7	10

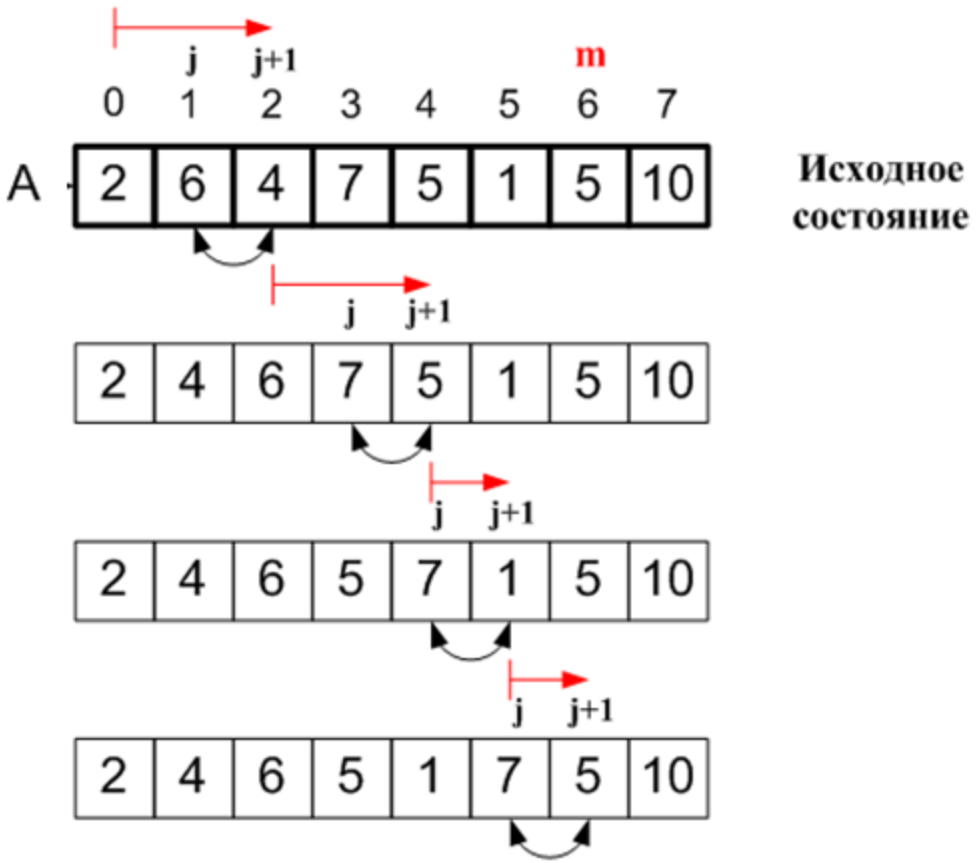
Следующий по возрастанию элемент нашел свое место (m), значит в очередном просмотре массива он не участвует: m должно стать равным 5



Следующий по величине пузырек всплыл!

Второй просмотр массива A

```
i:=2; { i – номер просмотра}
m:=m-1; {граница просмотра m=6}
for j:= 0 to m-1 do  {от 0 до  5}
```



Следующий по возрастанию элемент нашел свое место (m), значит в очередном просмотре массива он не участвует: m должно стать равным 5

Итак, второй просмотр (i = 2) окончен, но не все элементы массива A упорядочены. Признак неупорядоченности – были перестановки

Третий просмотр массива A

i:=3; { i – номер просмотра}

m:=m – 1; {граница просмотра m=5}

for j:=0 to m-1 do {от 0 до 4}

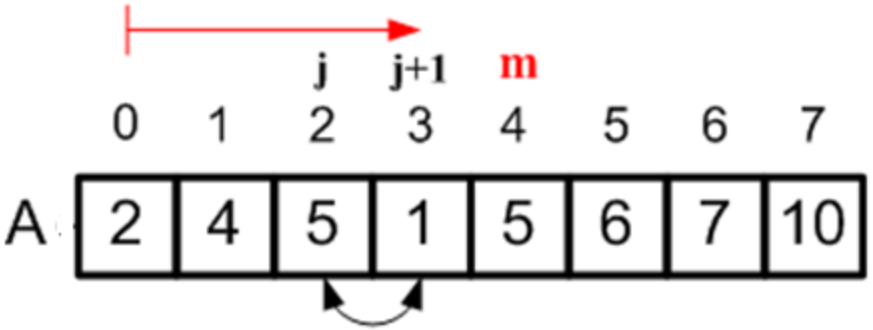
Были перестановки, значит массив не упорядочен – продолжаем просмотр!

Четвертый просмотр массива A

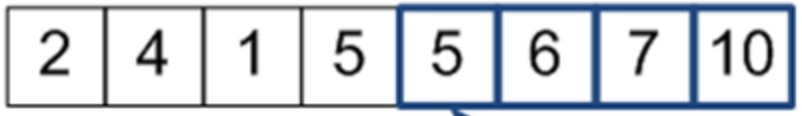
i:=4; { i – номер просмотра}

m:=m – 1; {граница просмотра m=4}

for j:=0 to m-1 do {от 0 до 3}



Исходное состояние



Следующий по возрастанию элемент нашел свое место (m), значит в очередном просмотре массива он не участвует: m должно стать равным 3

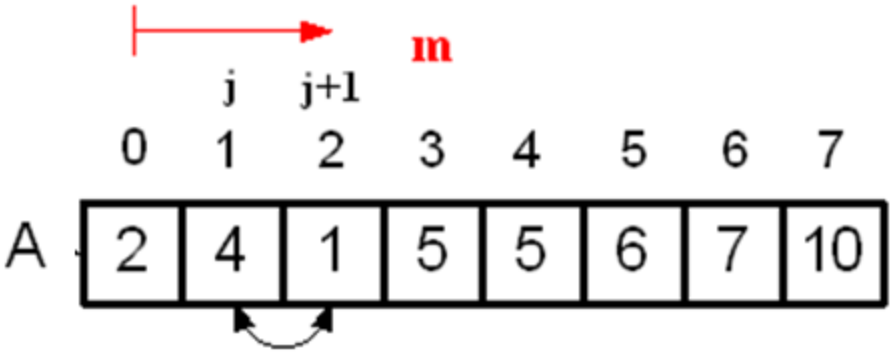
Были перестановки, значит массив не упорядочен – продолжаем просмотр!

Пятый просмотр массива A

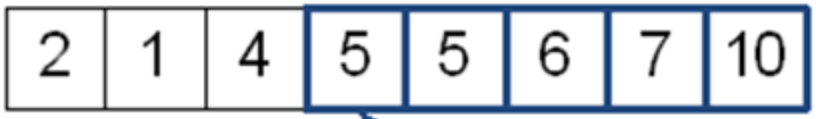
i:=5; { i – номер просмотра}

m:=m – 1; {граница просмотра m=3}

for j:=0 to m-1 do {от 0 до 2}



Исходное состояние



Следующий по возрастанию элемент нашел свое место (m), значит в очередном просмотре массива он не участвует: m должно стать равным 2

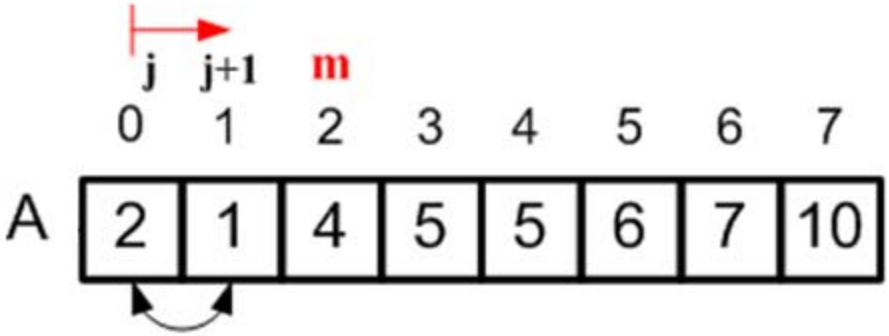
Были перестановки, значит массив не упорядочен – продолжаем просмотр!

Шестой просмотр массива A

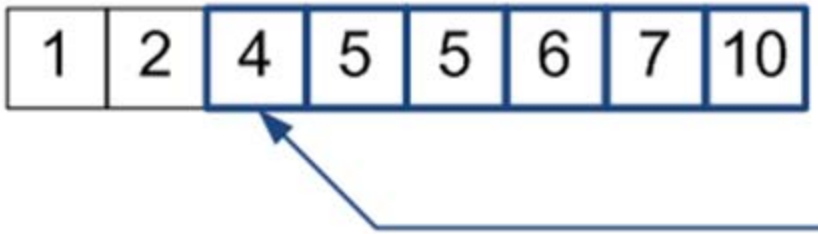
i:=6; { i – номер просмотра}

m:=m – 1; {граница просмотра m=2}

for j:=0 to m-1 do {от 0 до 1}



Исходное состояние



Следующий по возрастанию элемент нашел свое место (m), значит в очередном просмотре массива он не участвует: m должно стать равным 1

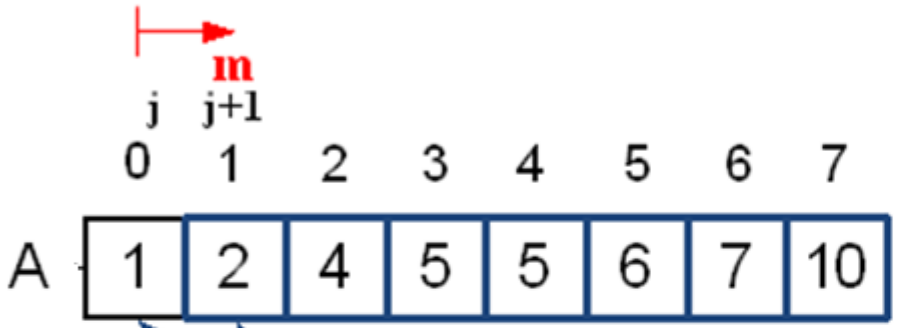
Были перестановки – продолжаем просмотр!

Седьмой просмотр массива A

i:=7; { i – номер просмотра}

m:=m – 1; {граница просмотра m=1}

for j:=0 to m-1 do {от 0 до 0}



Исходное состояние

Предпоследний элемент нашел свое место (а значит и последний (с минимальным значением!) тоже).

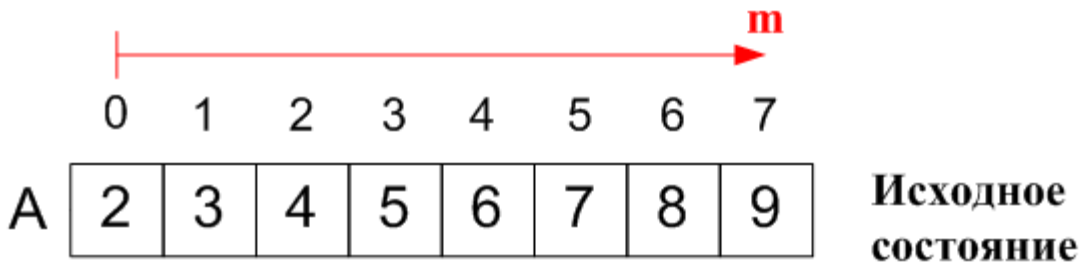
1. На этом просмотре перестановки не потребовались , а это значит, что массив упорядочен.

2. Значение границы m=1 исключает дальнейшие просмотры массива.

Пример 2

Первый просмотр массива A

Второй просмотр массива A

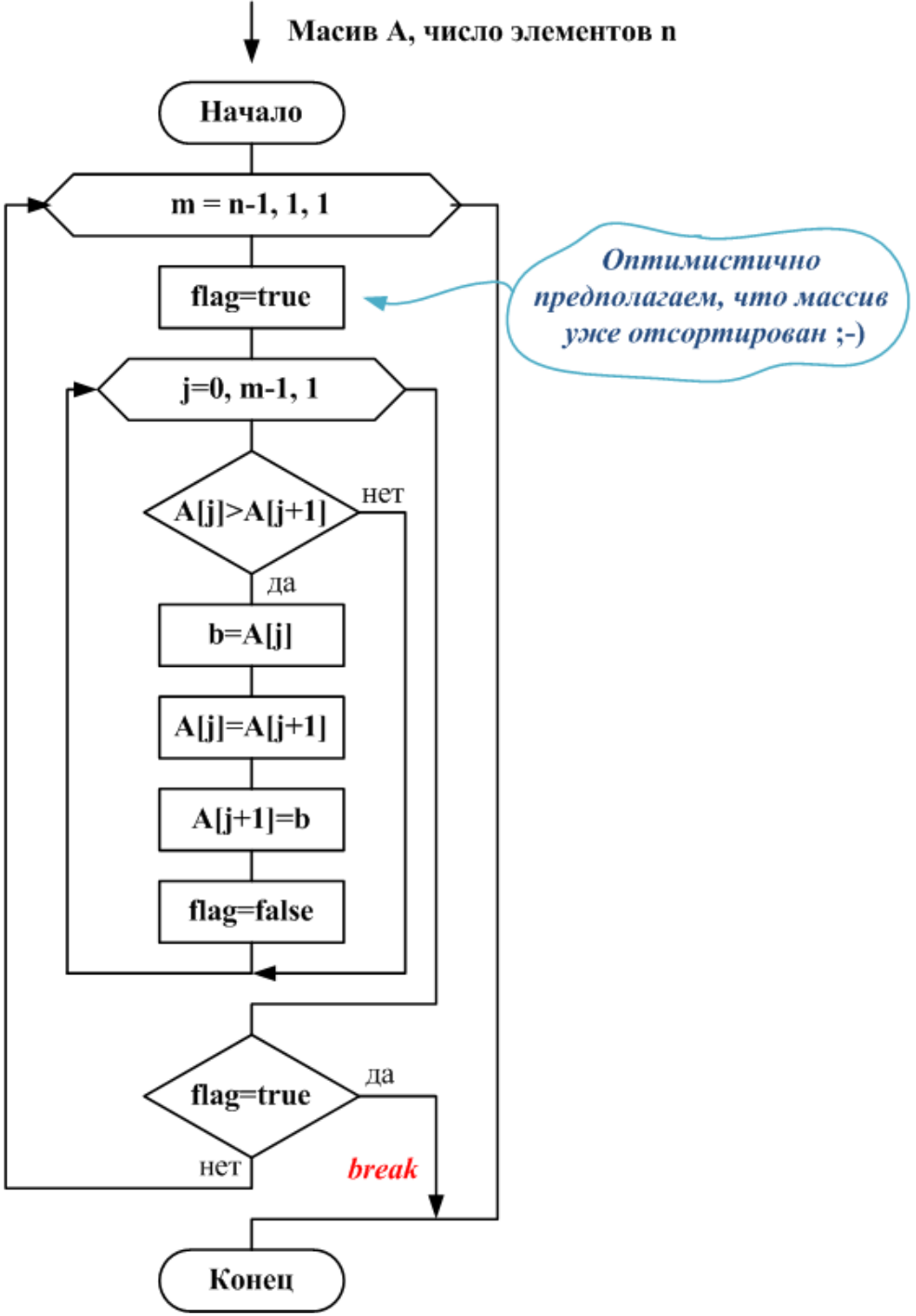


На этом просмотре перестановок не было, следовательно массив A упорядочен, и дальнейшие просмотры не нужны.

Для досрочного завершения просмотров введем переменную flag типа boolean:
если flag=true, то массив упорядочен

Метод:
Двигаясь от начала массива к концу, будем в каждой паре соседних элементов больший располагать справа.
Когда дойдем до конца массива, на последнее место попадет максимальный. Этот элемент уже не будет перемещаться.
Теперь повторяем процесс, но последний элемент уже не рассматриваем и т.д.
Если при очередном просмотре массива перестановок не было, значит массив упорядочен и дальнейшие просмотры не нужны.

Схема алгоритма процедуры упорядочивания элементов массива по возрастанию методом пузырька



Задача. Разработать программу, производящую сортировку двух массивов A и B по возрастанию значений элементов. В программе организовать в виде отдельных процедур ввод элементов массива с терминала, вывод элементов массива на терминал (в одну строку), сортировку массива (методом пузырька).

Головная (вызывающая) программа имеет линейную структуру, все подробности скрыты внутри подпрограмм (процедур).

Это пример хорошего структурирования программы.

```
program sort1;
 type
    tarr1 = array [0..99] of integer;
 var
    A,B: tarr1;
    n,i: integer;
  procedure sort_vial_1(var A: tarr1; n: integer);
   var
      j, m, b: integer;
     flag: boolean;
   begin
      for m := n-1 downto 1 do
        begin
           flag:=true;
           for j:=0 to m-1 do
               if A[ j ] > A[ j+1]
                  then
                     begin
                        b:=A[ j ]; A[ j ]:=A[ j+1]; A[ j+1]:=b;
                        flag:=false;
                     end;
            if  flag  then break;
        end
   end;
```

```
procedure inputarr(var X:tarr1; k:integer);
      var i:integer;
      begin
        for i:=0 to k-1 do
          begin
             write('element ', i ,': ');
             readln(X[i]);
           end
       end;

procedure putarr(var X:tarr1; k:integer);
      var i:integer;
      begin
         for i:=0  to k-1 do  write(X[ i ]:3 ,' ');
         writeln();
      end;
```

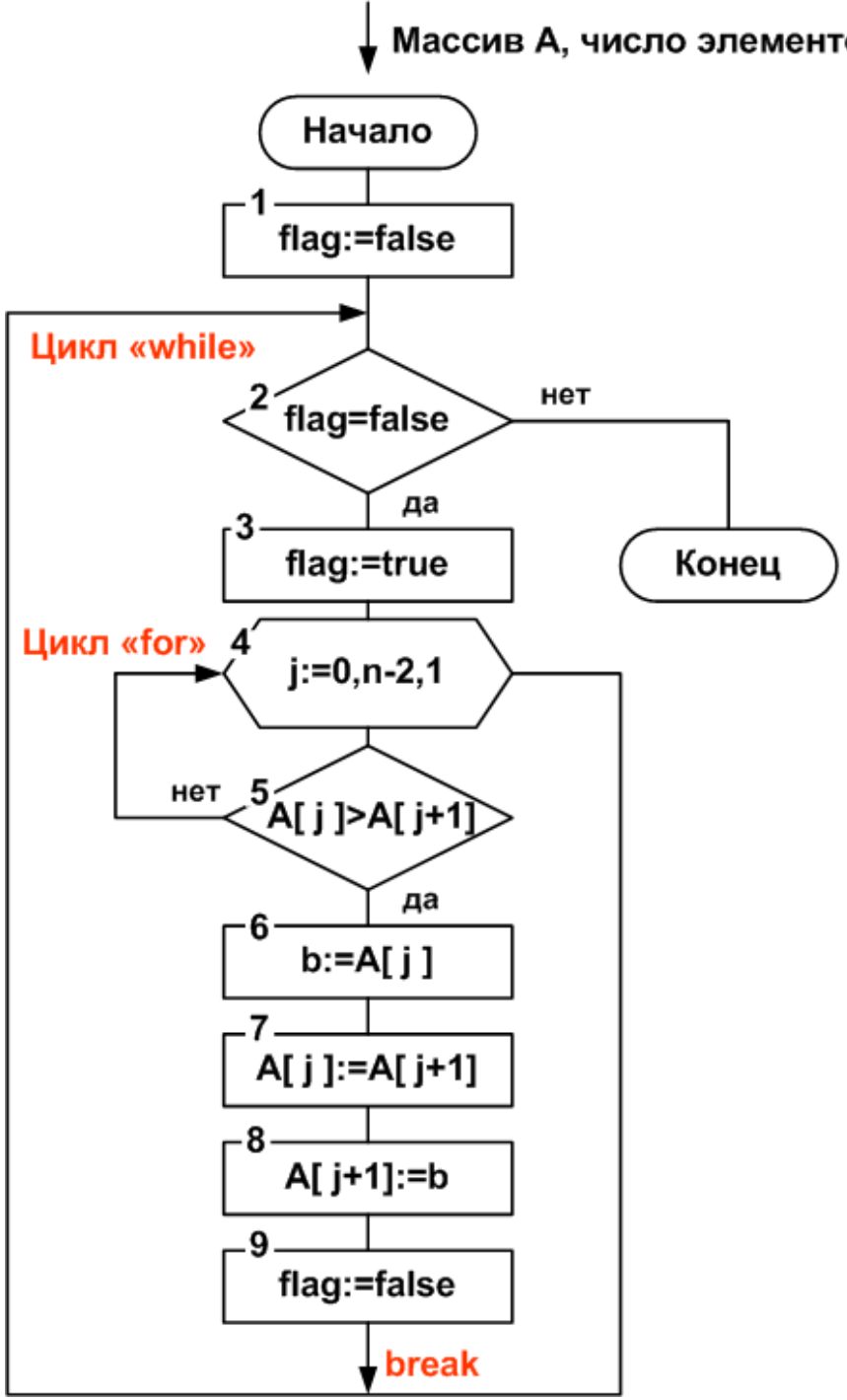
```
begin
  write('Введите число эл. массива n=');
  readln(n);
  writeln('Введите элементы массива A');
  inputarr(A,n);
  writeln('Введите элементы массива B');
  inputarr(B,n);
  writeln('До сортировки');
  putarr(A,n);
  putarr(B,n);
  sort_vial_1(A,n);
  sort_vial_1(B,n);
  writeln('После сортировки');
  putarr(A,n);
  putarr(B,n);
end.
```

Окно вывода

```
Введите число эл. массива n=5
Введите элементы массива A
element 0: 1
element 1: 7
element 2: -3
element 3: 1
element 4: 0
Введите элементы массива B
element 0: -1
element 1: -15
element 2: 9
element 3: 3
element 4: -15
До сортировки
  1   7  -3   1   0
 -1 -15   9   3 -15
После сортировки
 -3   0   1   1   7
-15 -15  -1   3   9
```

Сортировка методом пузырька (вариант2)

Метод: (вариант 2). **С каждой парой связано еще две пары:**
с (A[i], A[i+1])
связаны (A[i-1], A[i]) и (A[i+1], A[i+2]) (кроме первой и последней пар).
Поэтому, если в паре (A[i], A[i+1]) сделана перестановка, на всякий случай надо начать проверку с самого начала.
Если дошли до последней пары, а перестановки не потребовались, значит массив отсортирован.



Промоделируйте алгоритм вручную на рассмотренном выше тестовом примере.

Изобразите алгоритм двумя другими (правильными) способами.

```
procedure  sort_vial_2 (var A: tarr1; n: integer);
  var
    j, b: integer;
    flag: boolean;
  begin
    flag:=false;
    while not flag do {пока массив не отсортирован}
      begin
        flag:=true; {вот он - оптимизм}
        for j:=0 to n-2 do {просмотр массива с начала}
          if A[ j ] >A[ j+1]
            then
              begin
                b:=A[ j ];
                A[ j ]:=A[ j+1];
                A[ j+1]:=b; {произошла перестановка}
                flag:=false;
                break; {досрочный выход из for}
              end;
      end
  end;
```

На том же тестовом примере процедура даст те же результаты , что и sort_vial_1 (проверьте!)

Тест – это совокупность входных и соответствующих им выходных данных, на которых проверяется правильность работы программы.

Набор тестов должен быть полным (чтобы проверить все ветви алгоритма), и, желательно, не избыточным.

С чего начать разработку программы?

Сначала рисуем какой-нибудь пример (например матрицу, которую нужно транспонировать) и на нем пытаемся понять логику поставленной задачи (что нужно сделать).

Затем рисуем схему алгоритма (укрупненную, с обобщенными блоками – подпрограммами), т.е. уже в самом начале предполагаем структурирование алгоритма.

Далее строим алгоритмы работы подпрограмм (если подпрограммы простенькие, то можно обойтись и без рисования схемы алгоритма) и набор тестов для каждой подпрограммы и для программы в целом.

Проверяем работу подпрограмм и программы на тестовых примерах. Проводим отладку (можно пользоваться отладчиком или вставлять «отладочные печати»).

В технологии объектного программирования есть свои особенности.

Идея (2). Прямой выбор.

Минимальный из всех элементов должен иметь индекс 0. Минимальный среди оставшихся элементов должен иметь индекс 1, и т.д. Это следует из соотношения

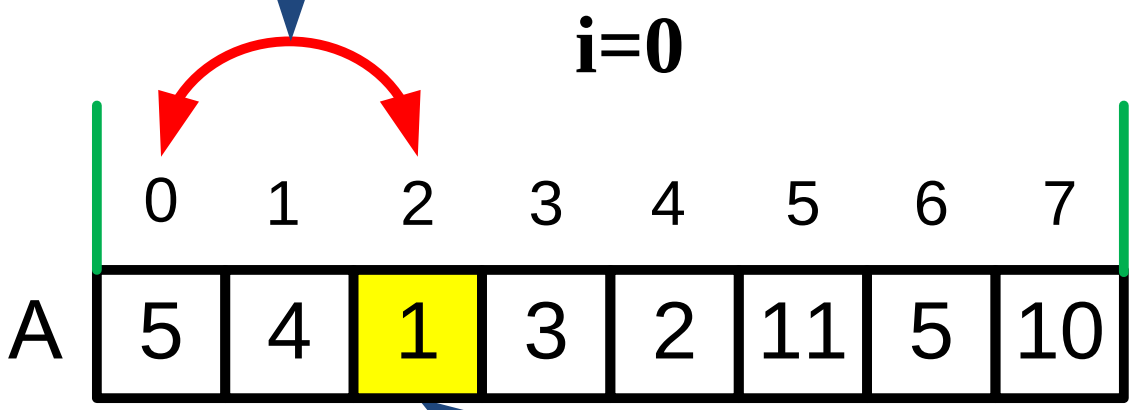
$A[0] \le A[1] \le A[2] \le \ldots \le A[n-1]$. (*).

Метод.

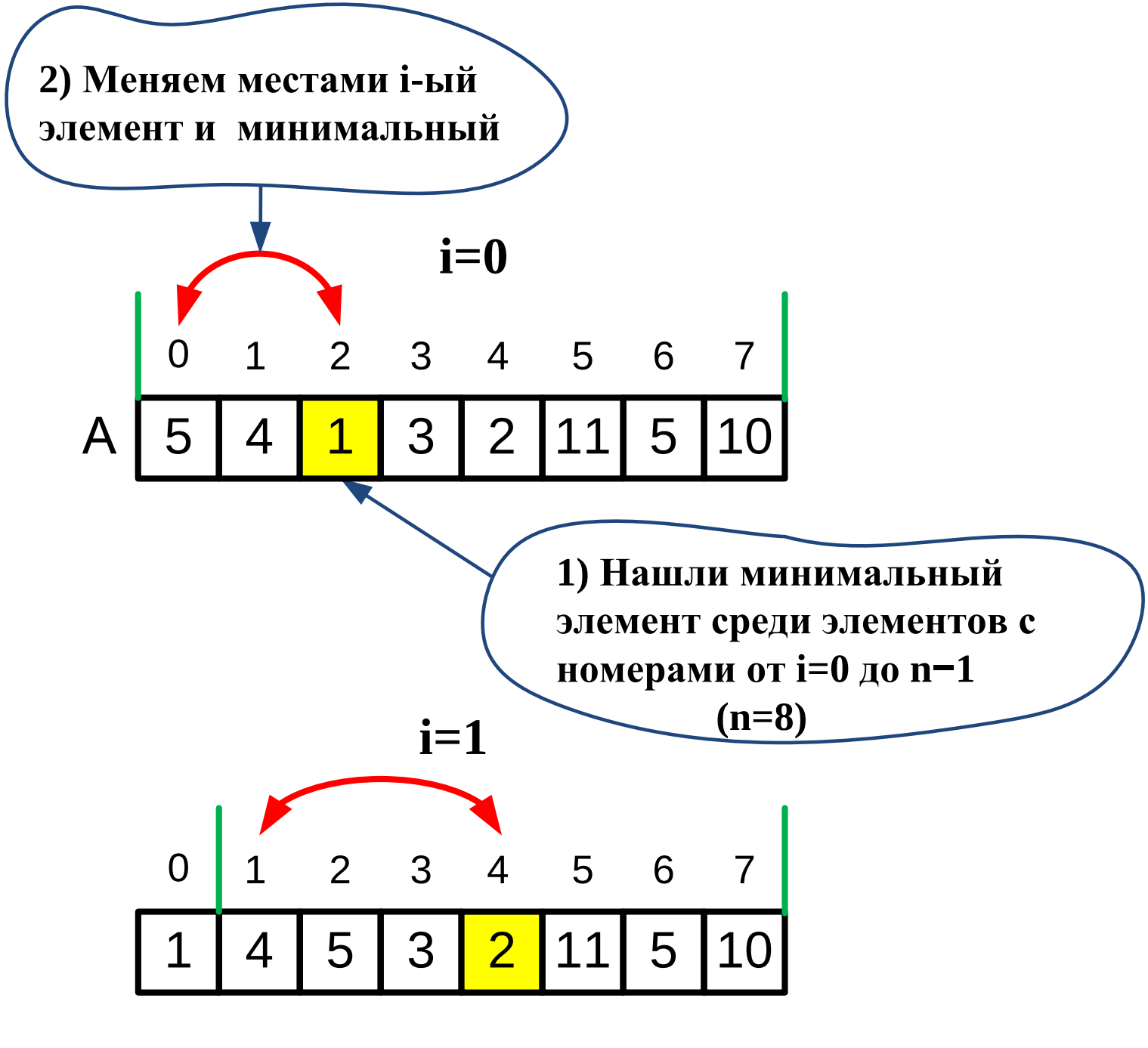
Находим минимальный элемент в массиве и меняем местами с A[0],
находим минимальный среди оставшихся и меняем местами с A[1],
находим минимальный среди оставшихся и меняем местами с A[2],
…,
находим минимальный из A[n-2] и A[n-1] и меняем местами с A[n-2].

Метод прямого выбора

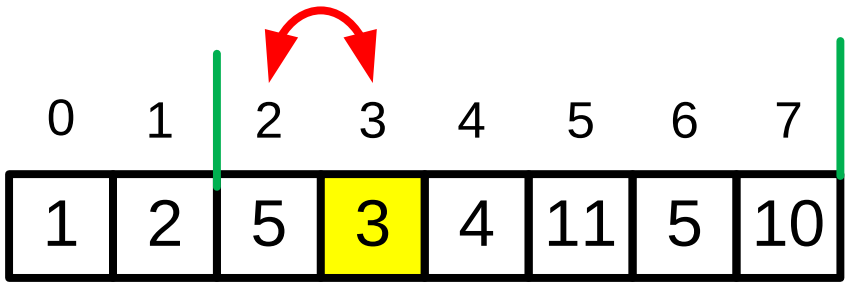
2) Меняем местами i-ый элемент и минимальный



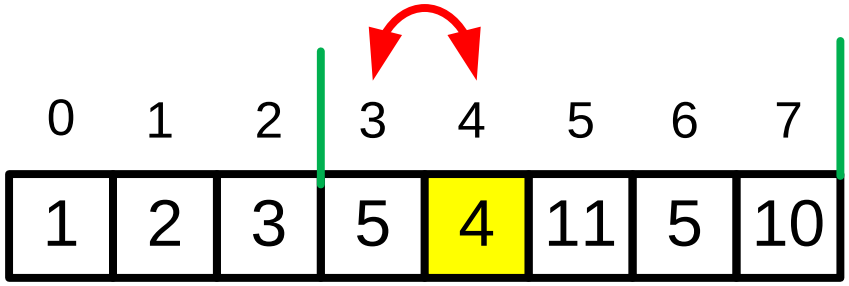
1) Нашли минимальный элемент среди элементов с номерами от i=0 до n−1 (n=8)


2) Меняем местами i-ый элемент и минимальный
i=0
0 1 2 3 4 5 6 7
A 5 4 1 3 2 11 5 10
1) Нашли минимальный элемент среди элементов с номерами от i=0 до n−1 (n=8)
i=1
0 1 2 3 4 5 6 7
1 4 5 3 2 11 5 10

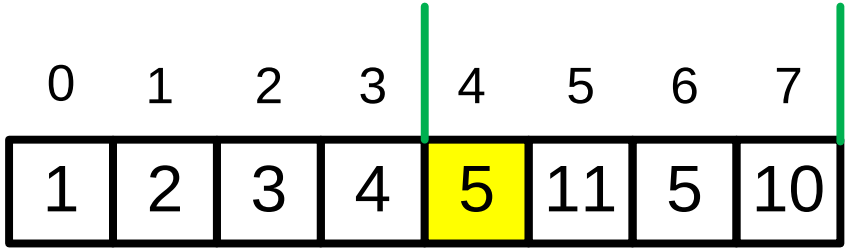
i=2



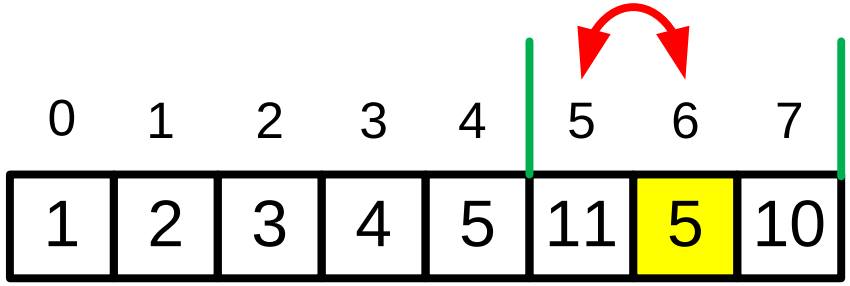
i=3



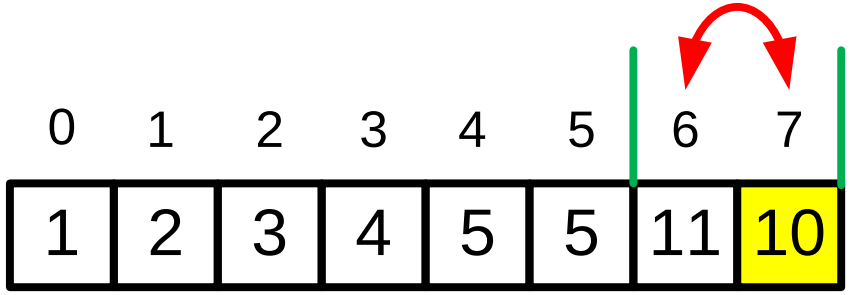
i=4



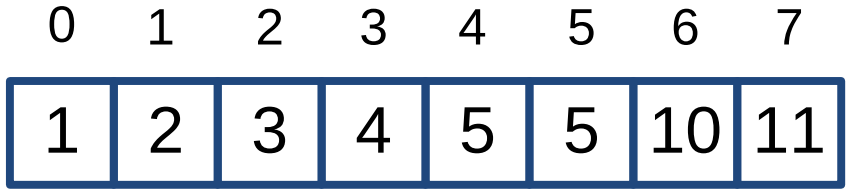
i=5



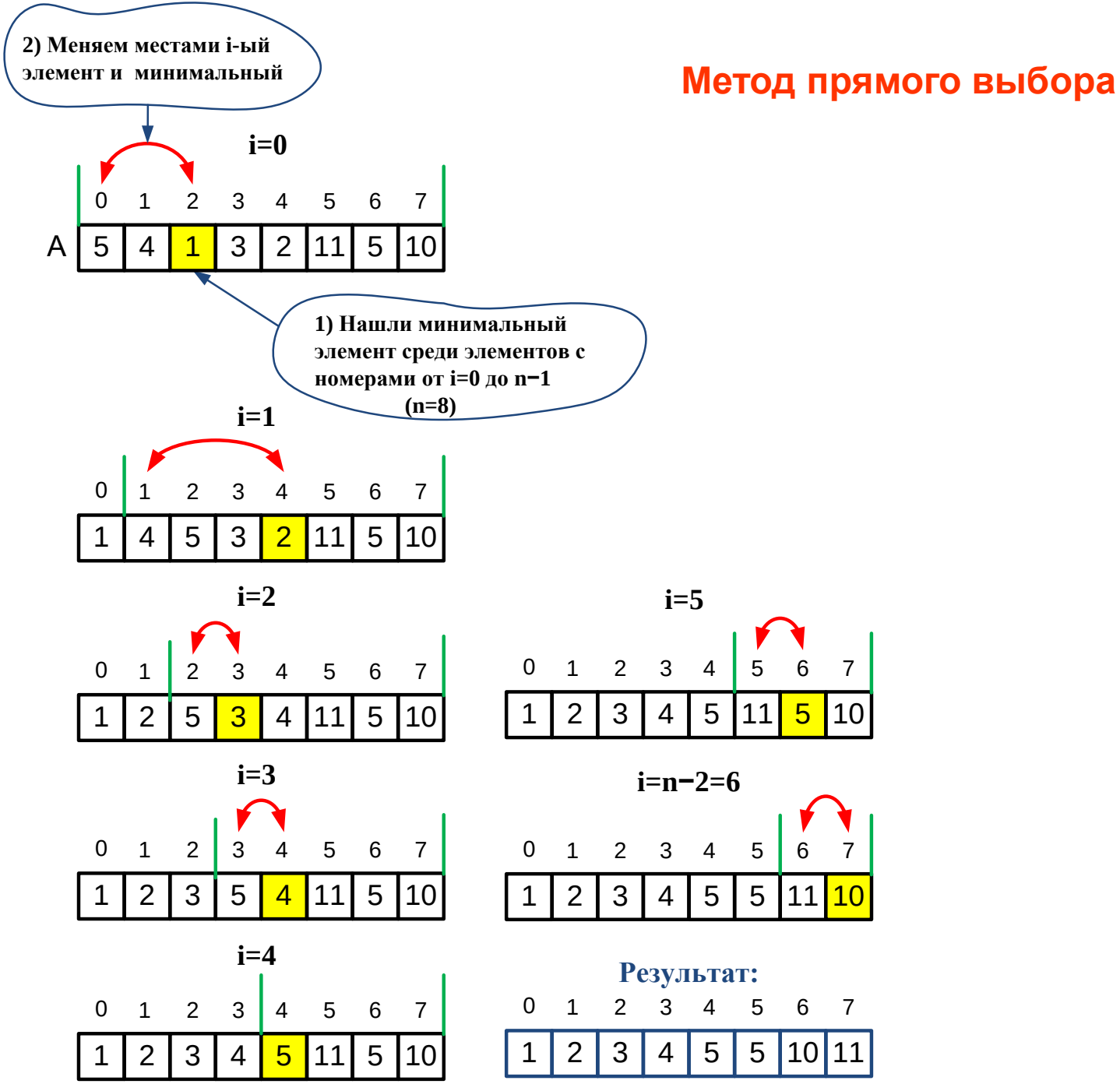
i=n−2=6



Результат:



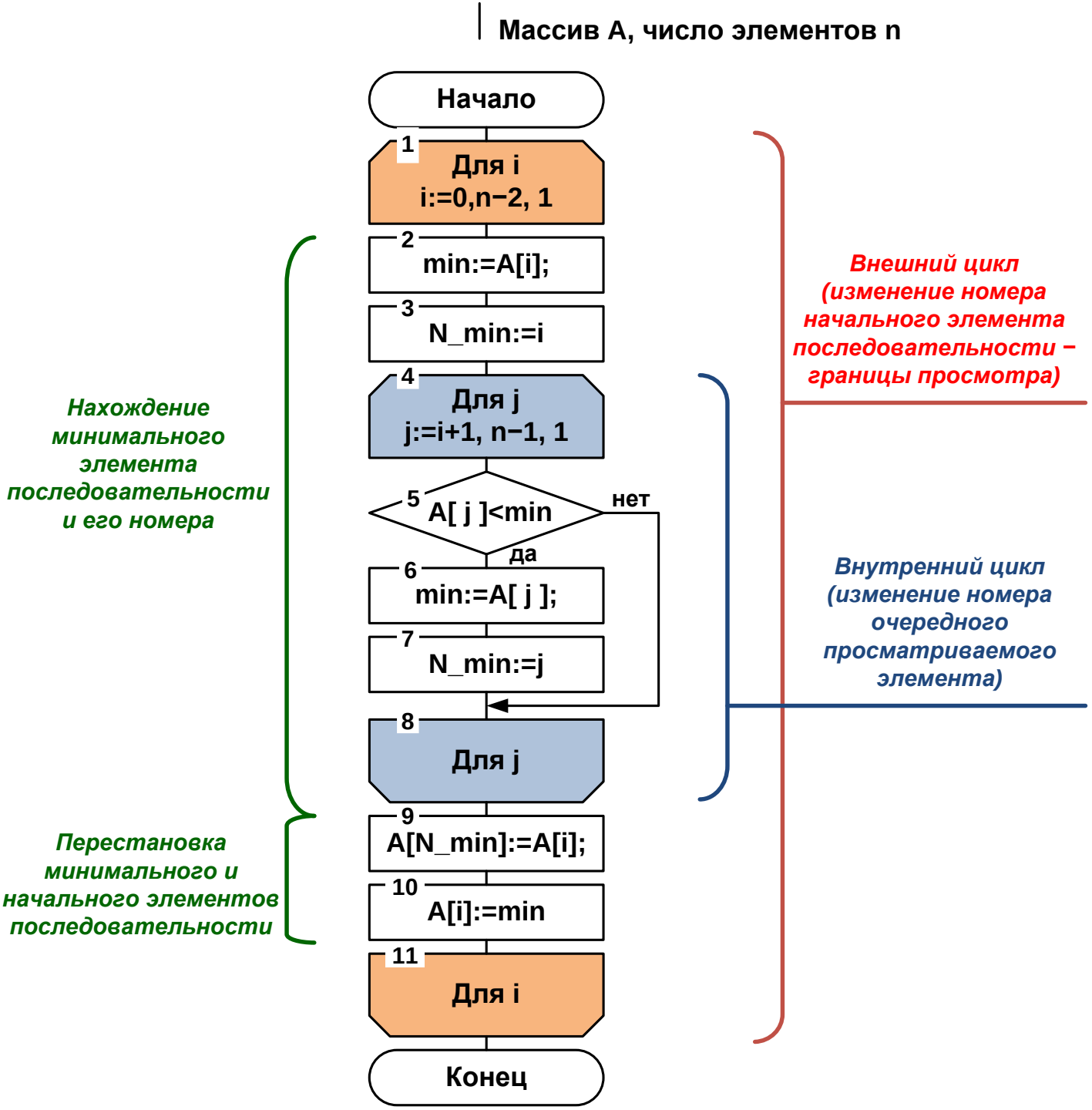
Метод прямого выбора



Обоснование алгоритма. Пусть элементы A[0], A[1], …, A[i-1] - уже на своих местах. Сейчас следует найти минимальный элемент среди элементов последовательности

A[i], A[i+1] , … , A[n-1]

и его номер, и поменять местами с элементом A[i]. Тогда на своих местах будут A[0], A[1], …, A[i-1], A[i]. Количество правильно расположенных элементов увеличилось на 1. Следует это действие выполнить для i = 0, 1, …, n – 2, и получится упорядоченный массив.


Массив A, число элементов n
Начало
1
Для i
i:=0,n−2, 1
2
min:=A[i];
3
N_min:=i
4
Для j
j:=i+1, n−1, 1
5
A[j]<min
нет
да
6
min:=A[j];
7
N_min:=j
8
Для j
9
A[N_min]:=A[i];
10
A[i]:=min
11
Для i
Конец
Нахождение минимального элемента последовательности и его номера
Перестановка минимального и начального элементов последовательности
Внешний цикл (изменение номера начального элемента последовательности − границы просмотра)
Внутренний цикл (изменение номера очередного просматриваемого элемента)

```
procedure sort_direct_choice(var A: tarr1; n: integer);
  var
    min, N_min, i, j: integer;
  begin
    for i:=0 to n-2 do
      begin
        min:=A[i];
        N_min:=i;
        for j:=i+1 to n-1 do
          if A[j]<min
            then
              begin
                min:=A[j];
                N_min:=j;
              end;
        A[N_min]:=A[i];
        A[i]:=min;
      end
  end;
```

Окно вывода

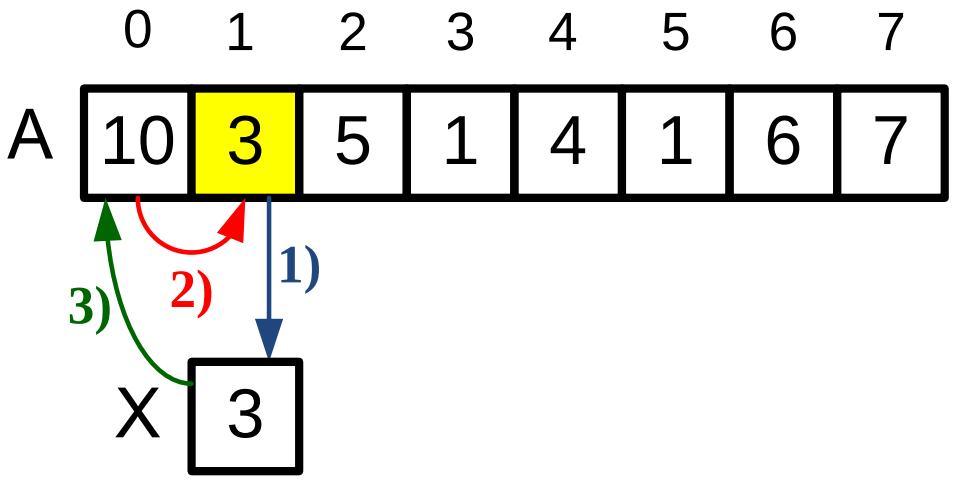
```
Введите число эл. массива n=5
Введите элементы массива A
element 0: -10
element 1: 20
element 2: 99
element 3: 35
element 4: 16
Введите элементы массива B
element 0: -3
element 1: -20
element 2: 10
element 3: -50
element 4: 10
До сортировки
-10  20  99  35  16
 -3 -20  10 -50  10
После сортировки
-10  16  20  35  99
-50 -20  -3  10  10
```

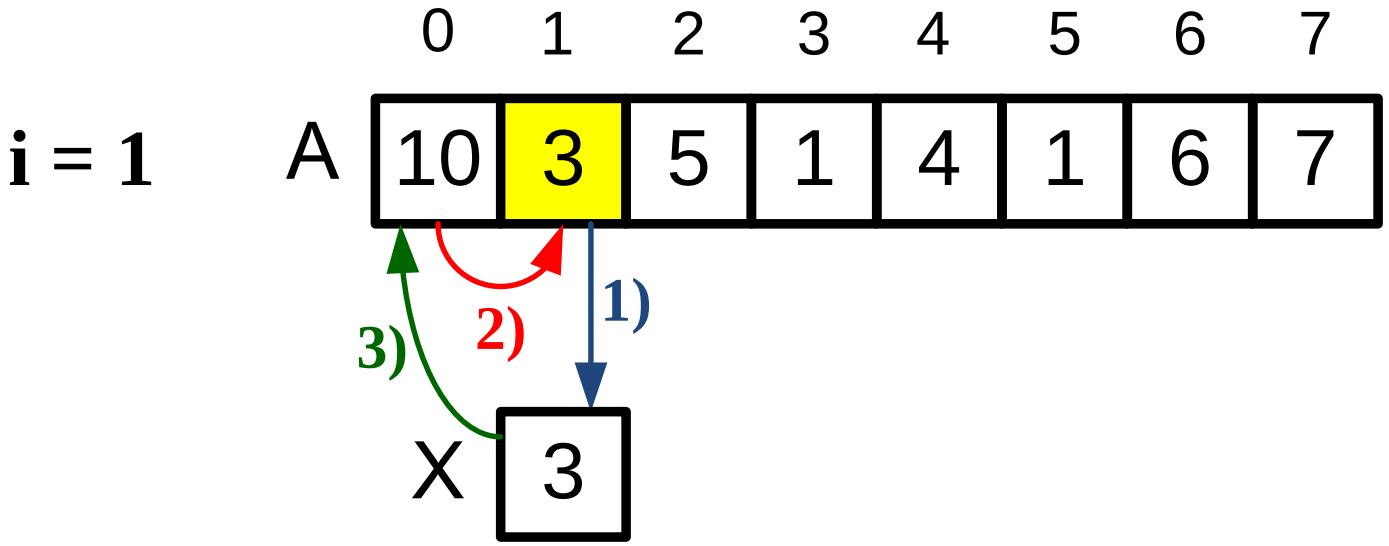
Идея (3). Прямые вставки

Предположим, что у нас есть упорядоченный массив из i элементов:
A[0] ≤ A[1] ≤ A[2] ≤… ≤ A[i-1].
Если бы удалось добавить к этим элементам еще один – X , сохраняя упорядоченность, длина (упорядоченного) массива увеличилась бы на 1.

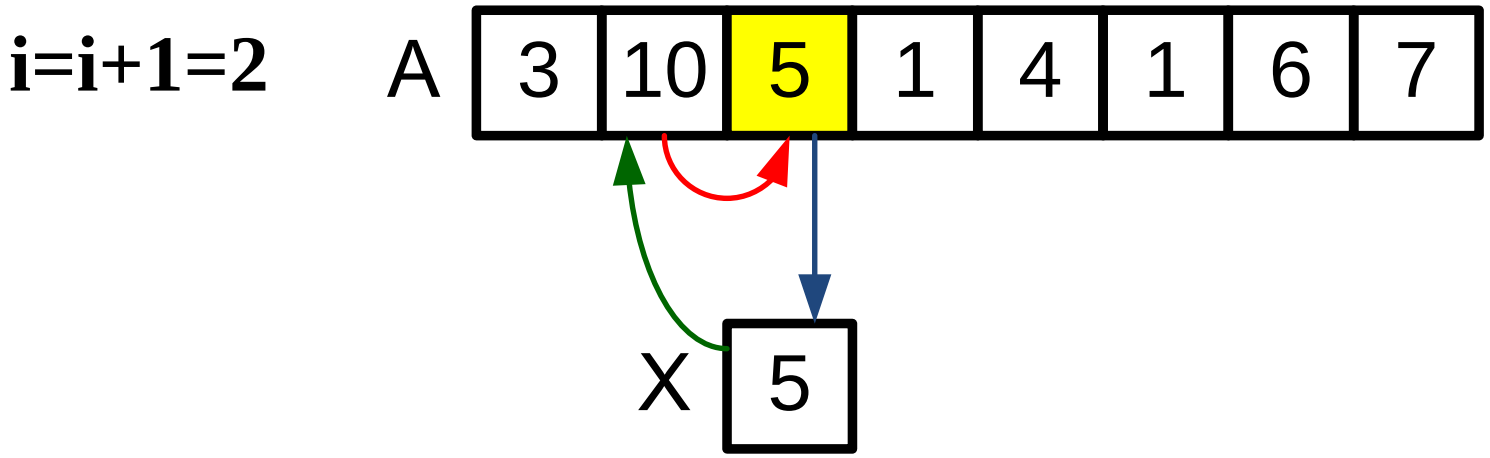
Метод прямых вставок

Начинаем с элемента с номером i = 1 (т.е. с A[1]).
1) Прячем его в карман X.
2) Предыдущие элементы (в данном случае A[0]), которые больше X, сдвигаем на одну позицию вправо.
3) В освободившуюся позицию вставляем X.

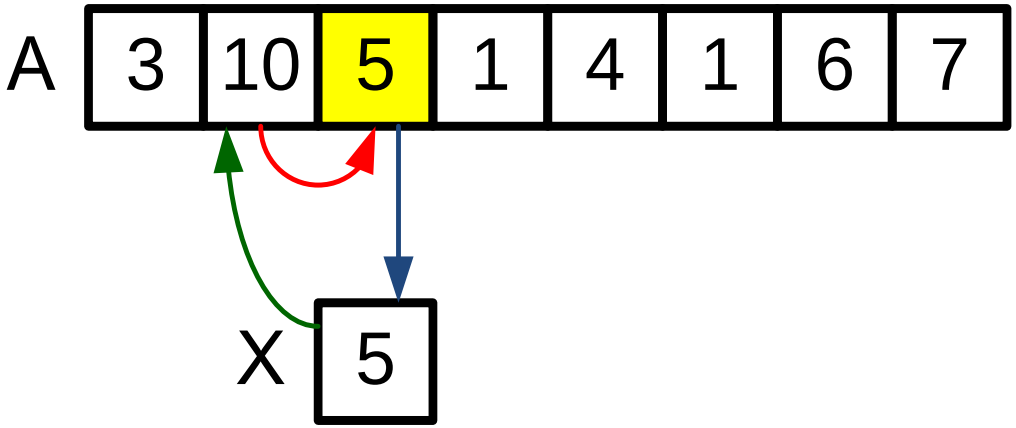




Получаем:

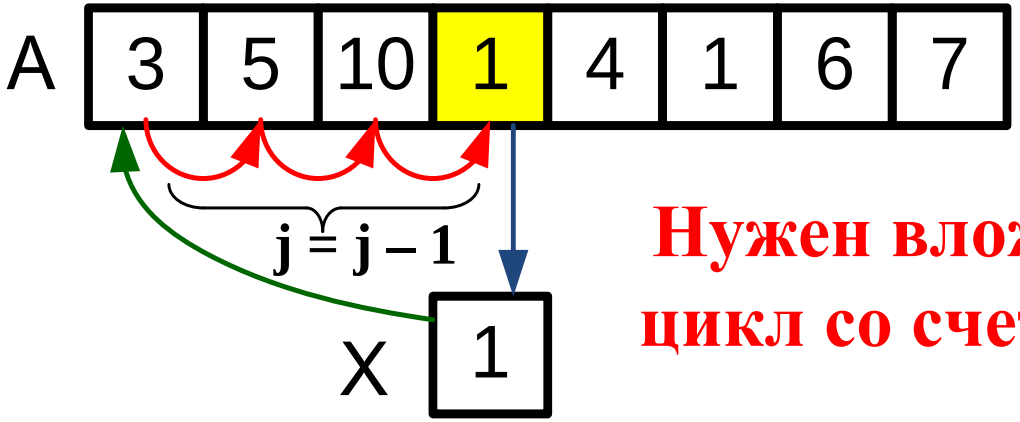


i=i+1=2



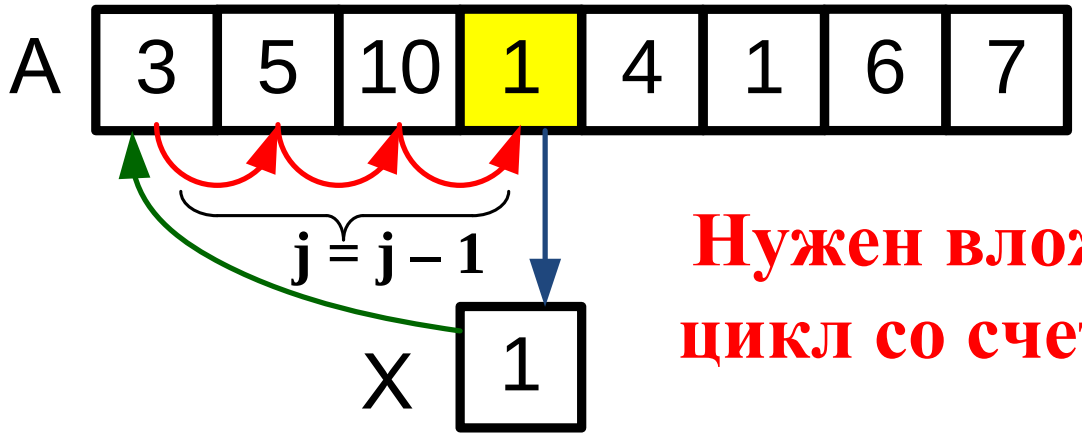
Получаем:

i=i+1=3



Нужен вложенный цикл со счетчиком j

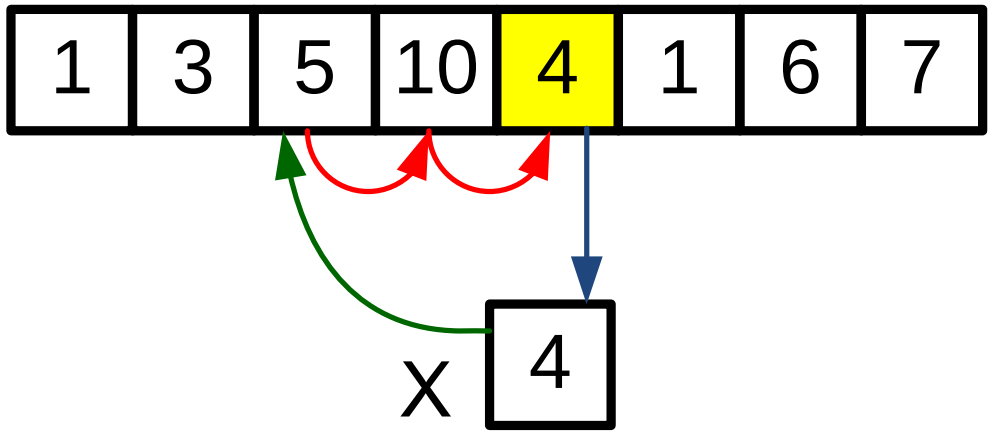
i=i+1=3



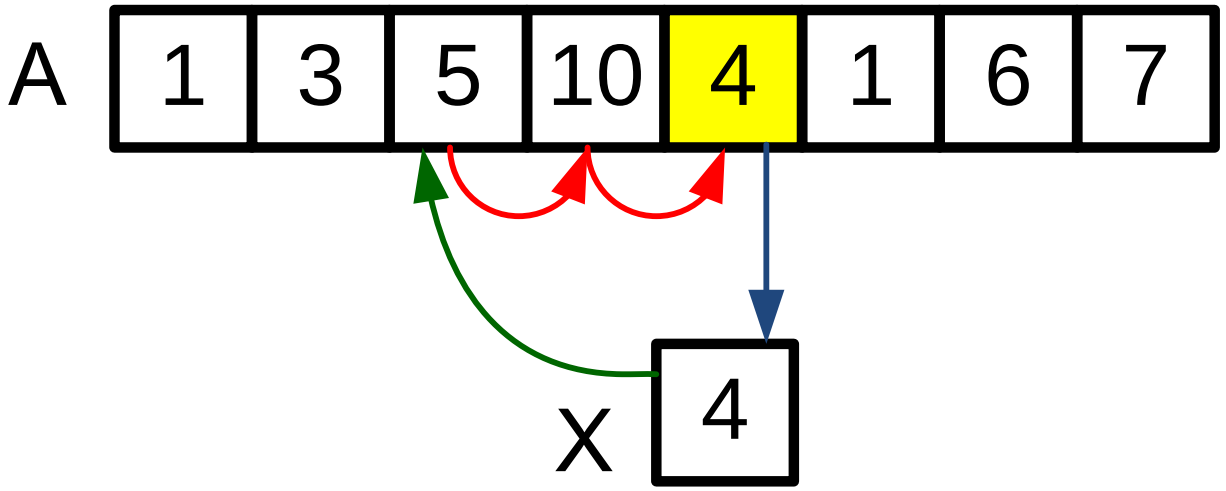
Нужен вложенный цикл со счетчиком j

Получаем:

i=i+1=4

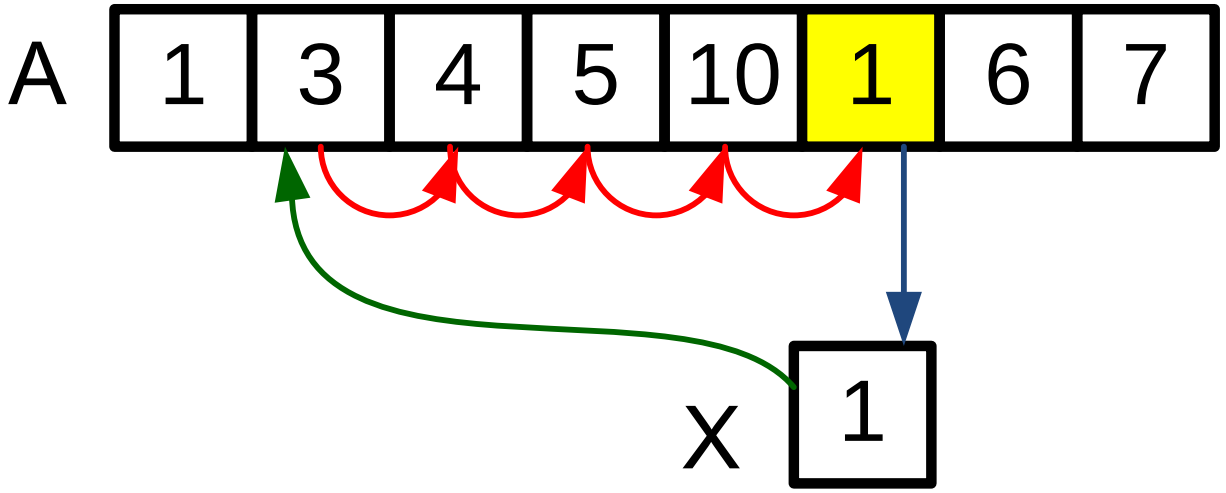


i=i+1=4

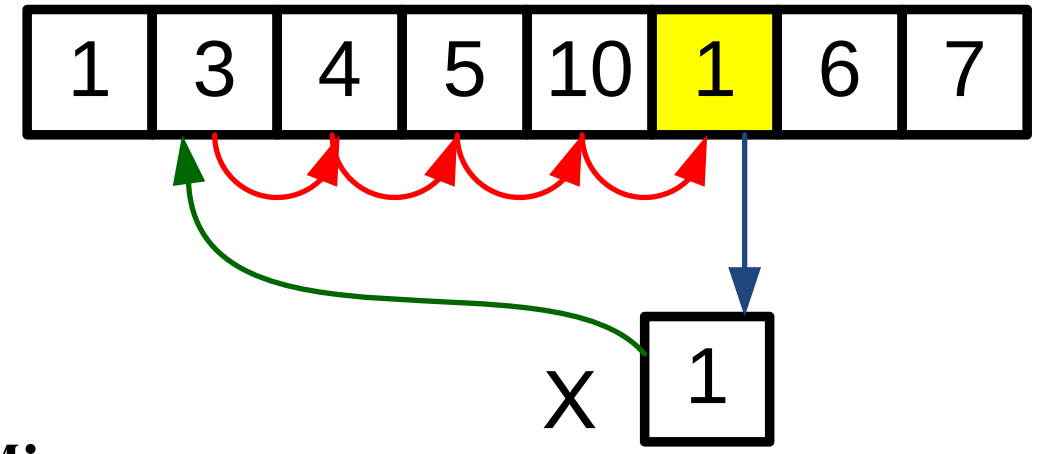


Получаем:

i=i+1=5

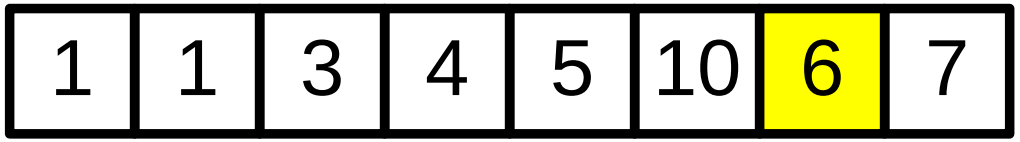


i=i+1=5



Получаем:

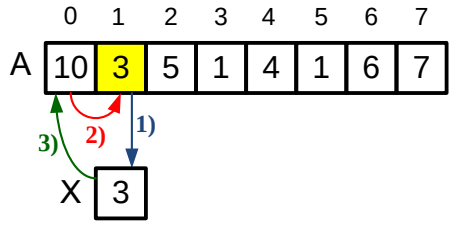
i=i+1=6



и т. д. пока i < n

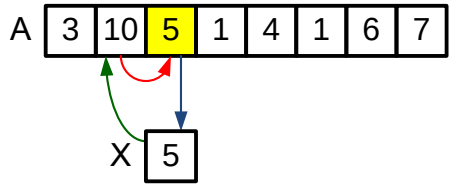
Метод прямых вставок

Начинаем с элемента с номером i = 1 (т.е. с A[1]).
1) Прячем его в карман X.
2) Предыдущие элементы (в данном случае A[0]), которые больше X, сдвигаем на одну позицию вправо.
3) В освободившуюся позицию вставляем X.



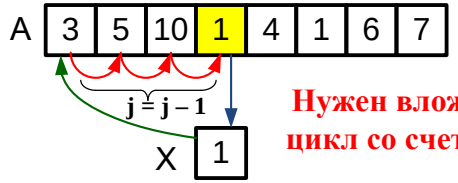
Получаем:

i=i+1=2



Получаем:

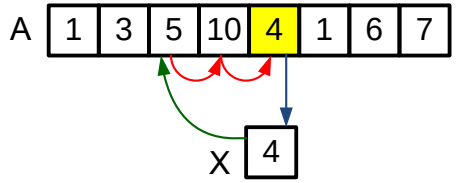
i=i+1=3



Нужен вложенный цикл со счетчиком j

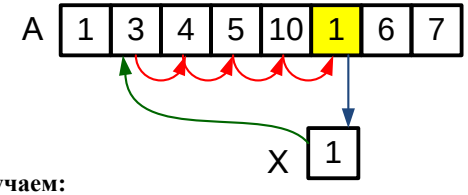
Получаем:

i=i+1=4



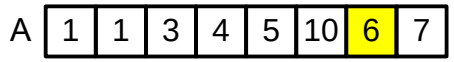
Получаем:

i=i+1=5



Получаем:

i=i+1=6



и т. д. пока i < n

Метод.

Когда потребуется добавить к упорядоченному массиву один элемент (X), следует найти место для этого элемента.

Место определяется соотношением:

A [слева] ≤ X < A[справа].

Можно начать с упорядоченного "массива", содержащего всего один элемент.

Поскольку после одного шага длина массива увеличивается, повторяя шаги, можно упорядочить массив любой длины.

При поиске места для X движемся по массиву влево, пока элемент массива больше X. Как только обнаружен элемент, не больший X, справа от него следует поместить X.

Движение влево возможно, пока номер элемента больше 0.

Алгоритм. Добавляем к упорядоченному массиву элемент A[i]. (Значение i должно изменяться от 1 до n−1 с шагом 1).

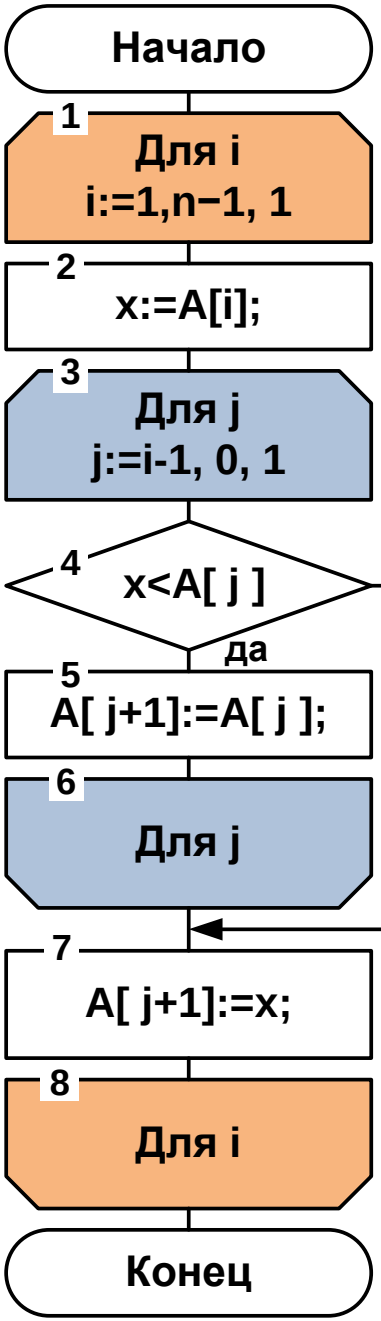
Для этого "прячем" A[i] в карман X, освобождая место i.

Последовательно, начиная с j =i-1 и пока j ≥ 0, сравниваем A[j] с X.

Если X<A[j], элемент A[j] перемещаем в позицию [j+1], уменьшаем j на 1 и повторяем проверку. Иначе (при X ≥ A[j]) в позицию [j+1] помещаем X и проверки заканчиваем.

Если дошли до элемента A[0] и место для X не найдено, его место – позиция 0.

Массив A, число элементов n



Копируем элемент A[i], для которого ищем место, в переменную x

Передвигаем элемент A[j] на одну позицию вправо

Помещаем элемент из переменной x на найденное для него место

Внешний цикл (изменение номера элемента, для которого ищем место, т.е. переход к следующему элементу)

Внутренний цикл (передвижение вправо на одну позицию элементов, находящихся левее элемента, для которого ищем место, и значение которых больше этого элемента)

```
procedure sort_direct_insertion(var A: tarr1; n: integer);
   var
      x, i, j: integer;
   begin
     for i:=1 to n-1 do
       begin
         x:=A[i];
         j:=i-1;
         while j>=0 do
            begin
              if x<A[j]
                 then A[j+1]:=A[j]
                 else break;
              j:=j-1;
            end;
         A[ j+1]:=x;
      end
   end;
```

Окно вывода

```
Введите число эл. массива n=5
Введите элементы массива A
element 0: 7
element 1: 2
element 2: 6
element 3: 4
element 4: 1
Введите элементы массива B
element 0: 10
element 1: -1
element 2: 20
element 3: -50
element 4: 0
До сортировки
  7   2   6   4   1
 10  -1  20 -50   0
После сортировки
  1   2   4   6   7
-50  -1   0  10  20
```

{для Pascal цикл for j:=i-1 downto 0 do... использовать нельзя, т.к.при выходе из цикла значение j остается равным 0, а не -1 (восстанавливается), в других языках все нормально}

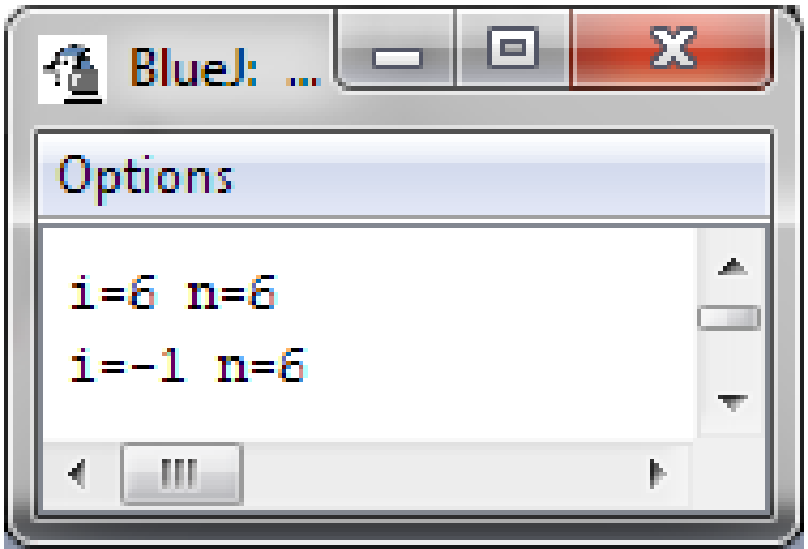
```
program nu_i_nu;{Особенности реализации
                цикла for в языке Pascal}
   var
     j,n: integer;
   begin
     n:=0;
     for j:=5 downto 0 do n:=n+1;
     {как по вашему, чему равно j
      после выхода из цикла}
     writeln ('j=',j,' n=',n);
     n:=0;
     for j:=0 to 5 do n:=n+1;
     {А здесь?}
     writeln ('j=',j,' n=',n);
  end.
```

Окно вывода

```
j=0 n=6
j=5 n=6
```

То же на Java

```
1  public class ForDemo{
2     public static void main (String[] args){
3          int i,n;
4          n=0;
5          for (i=0; i<=5; i++) n=n+1;
6          System.out.println ("i="+i+" n="+n);
7          n=0;
8          for (i=5; i>=0; i--) n=n+1;
9          System.out.println ("i="+i+" n="+n);
10    }
11 }
```



Текст процедуры сортировки на Java

```
public static void вставки (int [ ] A) {
    int x, i, j;
    for(i = 1; i < A.length; i++){
        x=A[ i ];
        for( j = i-1; j >= 0; j--)
            if (x < A[ j ]) A[ j+1] = A[ j ];
            else break;
         A[ j+1] = x;
    }
}
```

Четыре рассмотренных метода сортировки на одном и том же тестовом примере дают одинаковый результат. Проверьте!

Задание: измените рассмотренные методы так, чтобы они делали сортировку по убыванию элементов.

Оценим трудоемкость простых методов сортировки.

Сумма первых n членов арифметической прогрессии:

$S_n=(2a_1+d(n-1))n/2,$

где a_1 – первый член прогрессии;

d – разность прогрессии.

Оценка трудоемкости алгоритма

Худший случай:

Для пузырьковой сортировки (вариант, рассмотренный первым):

$T=(n-1)*t + (n-2)*t + (n-3)*t +…+ 1*t$

(n – число элементов массива, t –среднее время на перестановку двух элементов массива)

$a_1=t;\ d=t;$

$T = S_n=(2t+t(n-1))n/2 =0.5t\ (n^2 +n).$

При больших n главная составляющая – n^2.

Максимальное время выполнения простых методов сортировки – порядка n^2.

Более быстрый метод сортировки, основанный на рекурсии, будет рассмотрен позже.