

# Лекция 12

## Сортировка

1. Сортировка вставкой (простыми включениями)
2. Сортировка выбором (выделением)
3. Сортировка обменом (метод «пузырька»)
4. Сортировка Шелла;
5. Шейкер сортировка
6. Быстрая сортировка (QuickSort).

# Сортировка массивов

**Сортировка** - процесс перестановки заданного множества нумерованных объектов в определенном порядке. Цель сортировки - облегчить последующий поиск элементов в отсортированном множестве.

$$a_1 \leq a_2 \leq a_n$$

$$a_1, a_2, \dots, a_n \Rightarrow a_{k1}, a_{k2}, \dots, a_{kn}$$
$$f(a_{k1}) \leq f(a_{k2}) \leq f(a_{k3}) \leq \dots \leq f(a_{kn})$$

**Две категории методов сортировки:**

- сортировка массивов
- сортировка файлов

## **Базовые операции:**

- пересылки (обмен) –  $M$
- сравнения –  $C$

## **Принципы построения алгоритмов сортировки массивов:**

- сортировка на месте
- произвольный доступ к элементам

## **Базовые алгоритмы сортировки массивов:**

сортировка вставкой (включениями)

сортировка выбором (выделением)

сортировка обменом (метод «пузырька»)

**Улучшенные алгоритмы сортировки массивов:**  
сортировка Шелла;  
Шейкер сортировка  
быстрая сортировка (QuickSort).

```
void f(int *a, int n) {  
    randomize();  
    int k=0;  
    for(int i=0; i<n; i++)  
        *(a+i)=random(100);  
}
```

```
void show(int *a, int n) {  
    for(int i=0; i<n; i++)  
        cout<<a[i]<<"\t";  
    cout<<"\n";  
}
```

```
int _tmain( ) {  
    int *b,n1;  
    cout<<"strok:";  
    cin>>n1;  
    int i,j;  
    b=new int [n1];  
  
    f(b,n1);  
    show(b,n1);  
  
    . . .  
    delete b;  
  
    . . .  
}
```

# Алгоритм сортировки вставкой

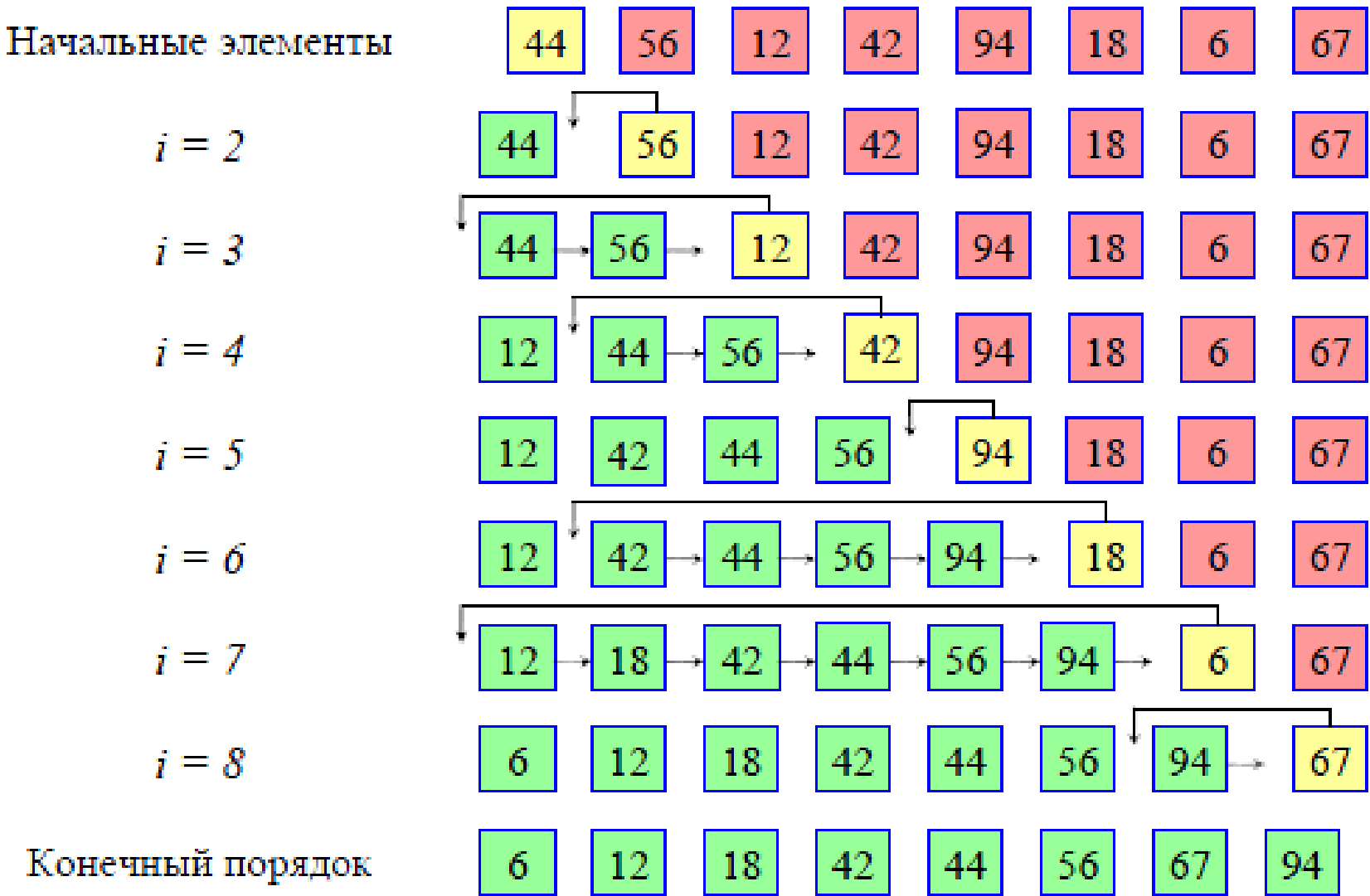





Рис. 1. Пример сортировки простыми включениями:  – отсортированная часть,  – входная часть,  - текущий элемент.

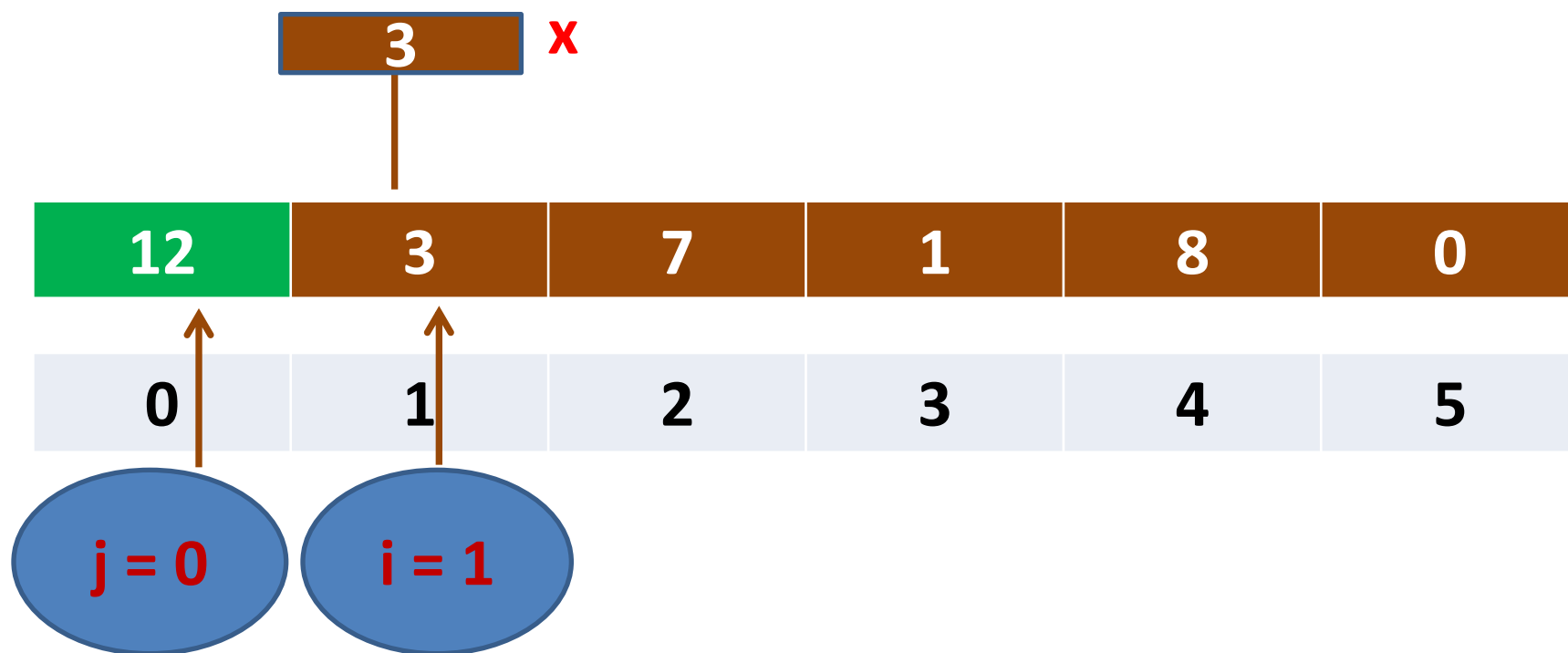
## Алгоритм сортировки вставкой :

1. Для всех  $i$  от 1 до  $n$  выполнить:

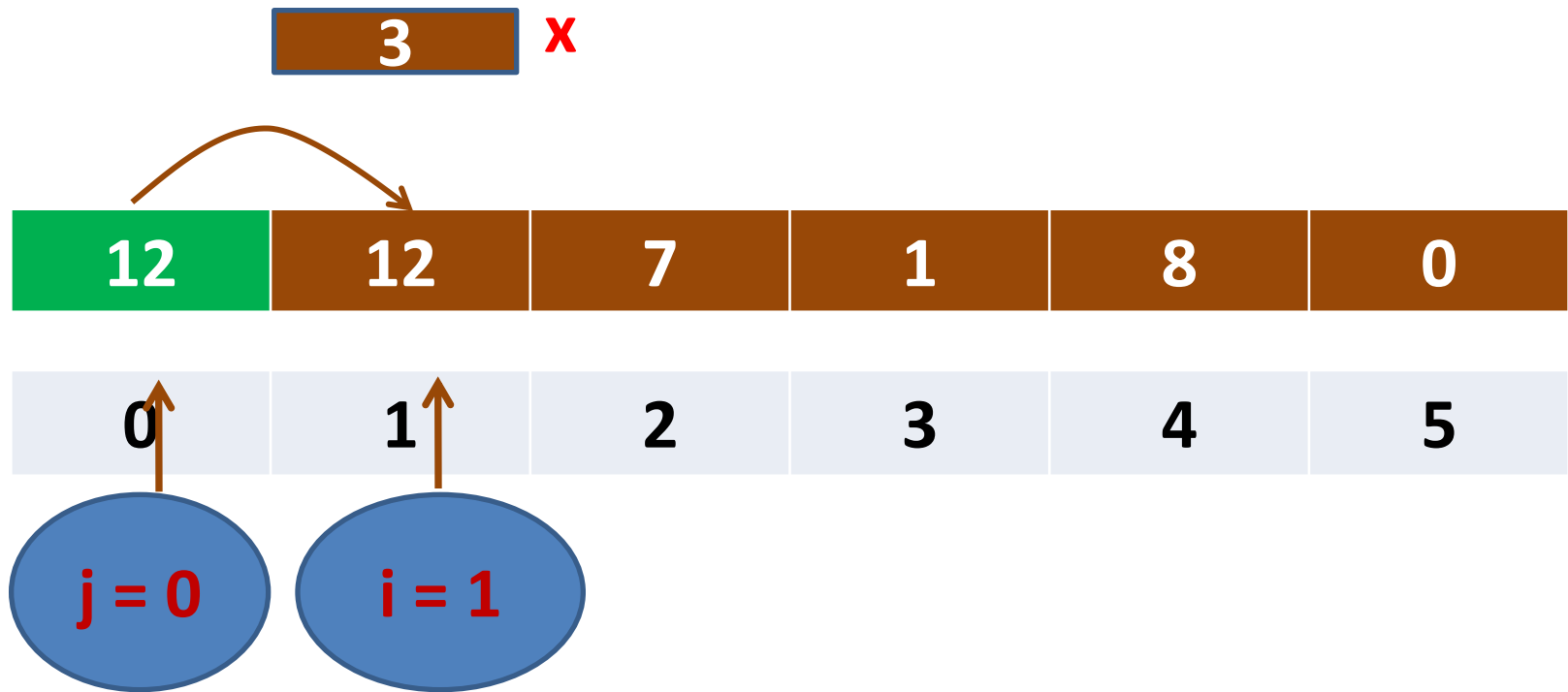
1. 1. Взять очередной  $i$ -й неотсортированный элемент и сохранить его в рабочей переменной  $x$
1. 2. Найти позицию  $j$  в отсортированной  $(0 \dots i-1)$  части массива, в которой присутствие взятого элемента не нарушит упорядоченности элементов
- 1.3. Сдвиг элементов массива от  $i-1$  до  $j-1$  вправо, чтобы освободить найденную позицию вставки
- 1.4. Вставка взятого элемента в найденную  $j$ -ю позицию.



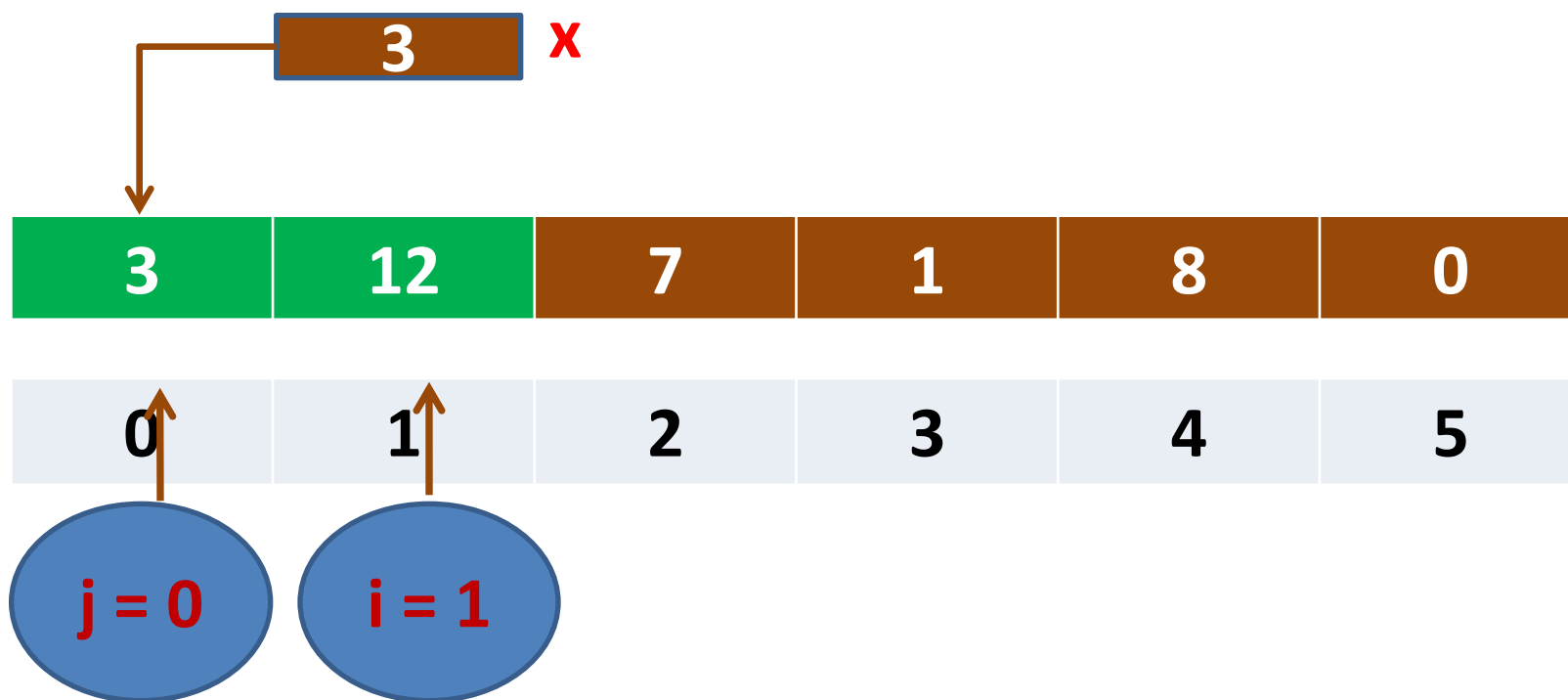
## Сортировка вставкой : 1-й проход



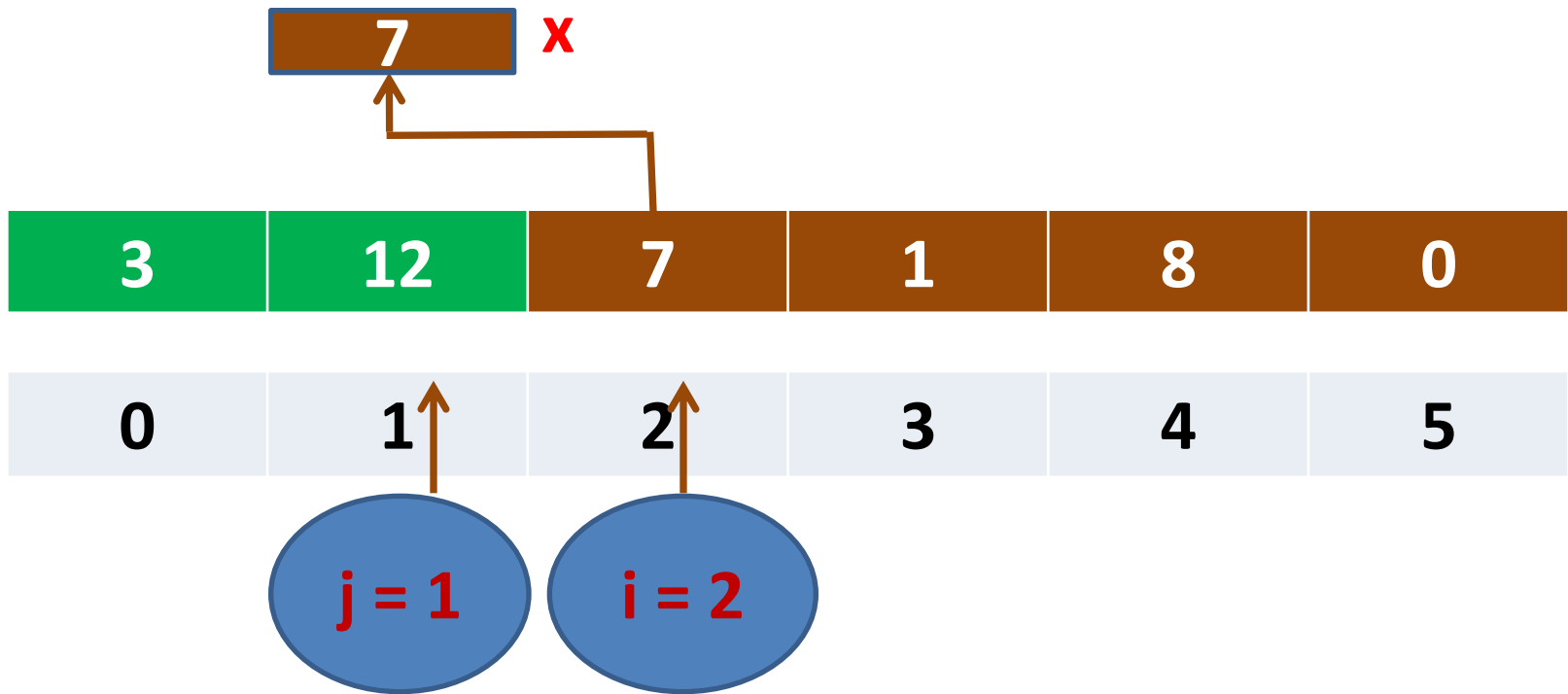
## Сортировка вставкой : 1-й проход



## Сортировка вставкой : 1-й проход

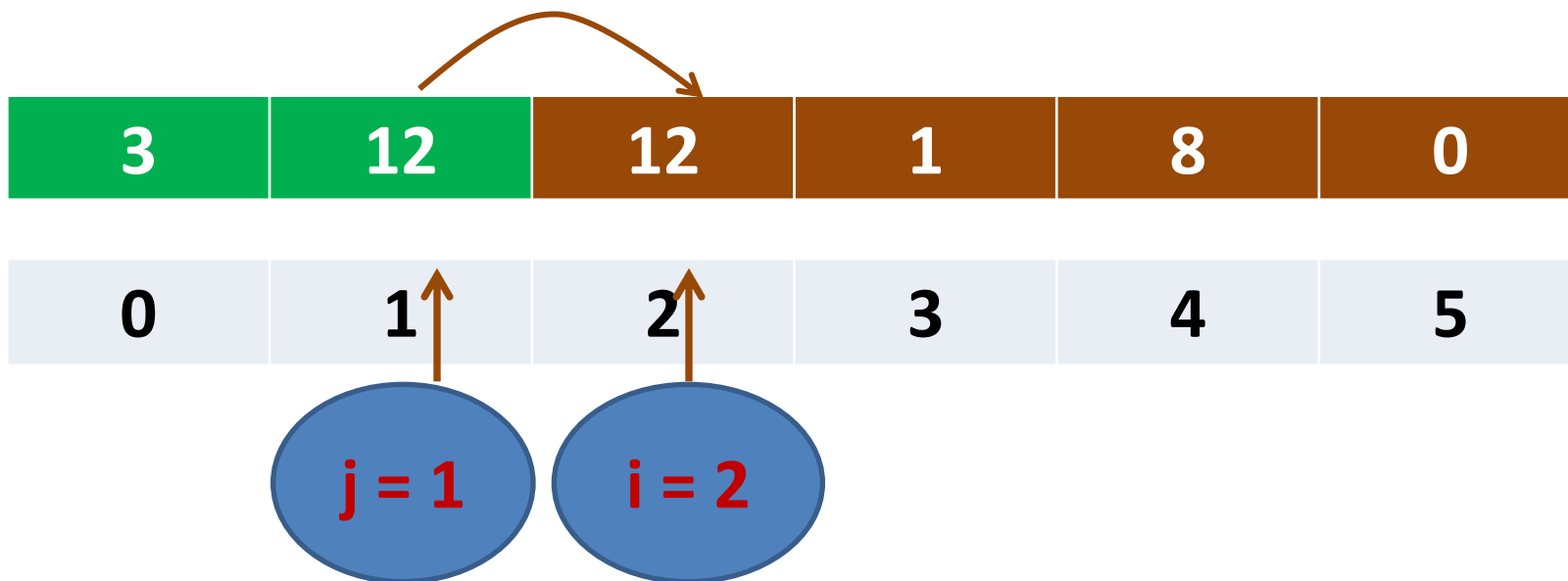


## Сортировка вставкой : 2-й проход

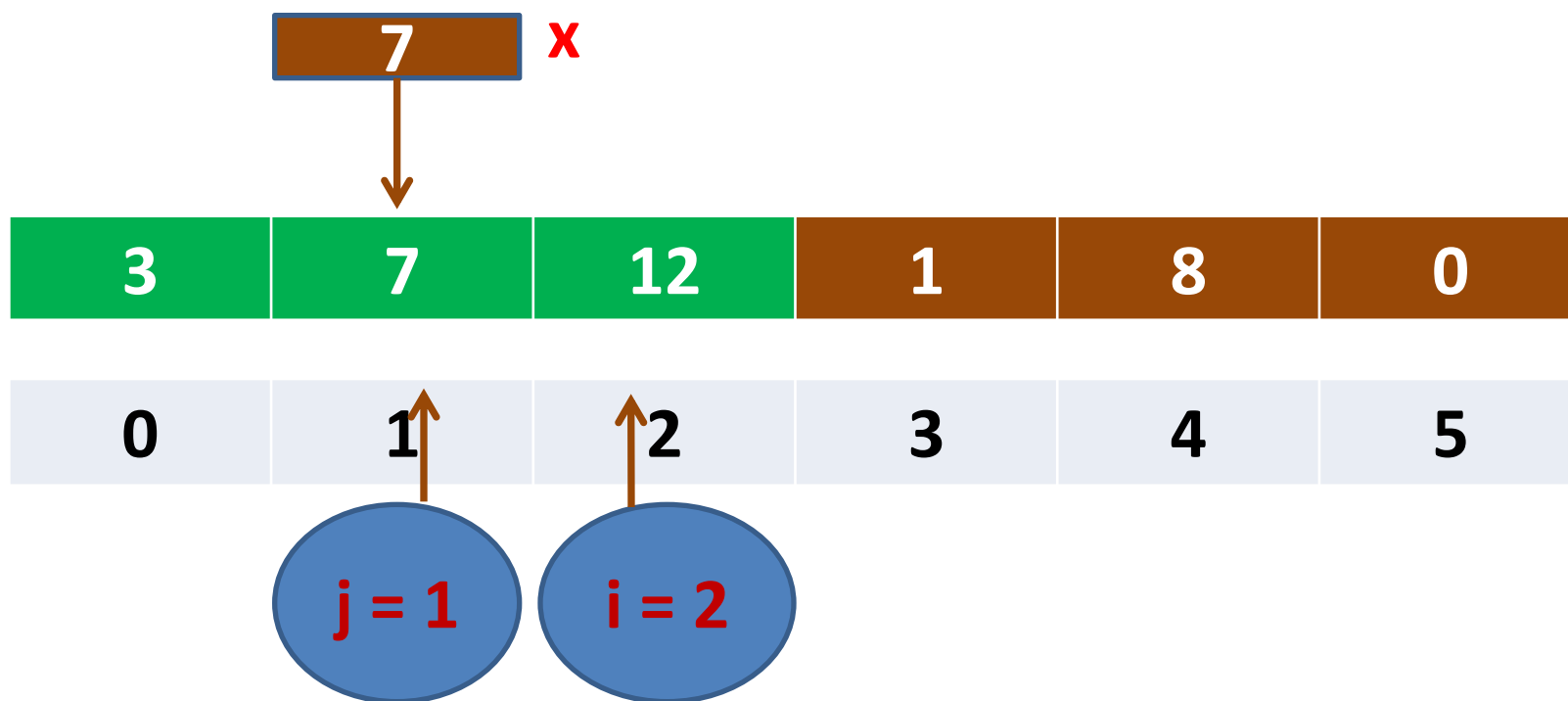


## Сортировка вставкой : 2-й проход

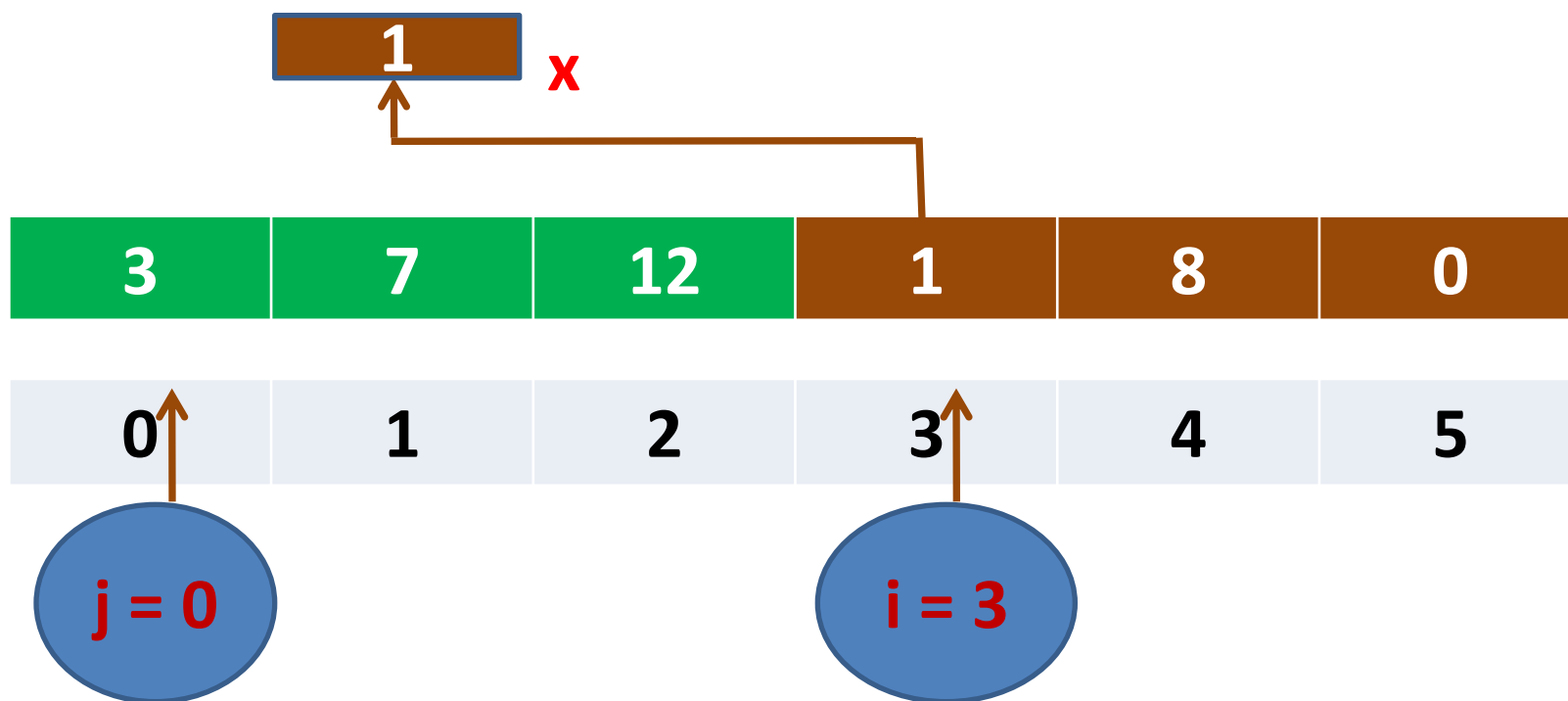
**7** **x**



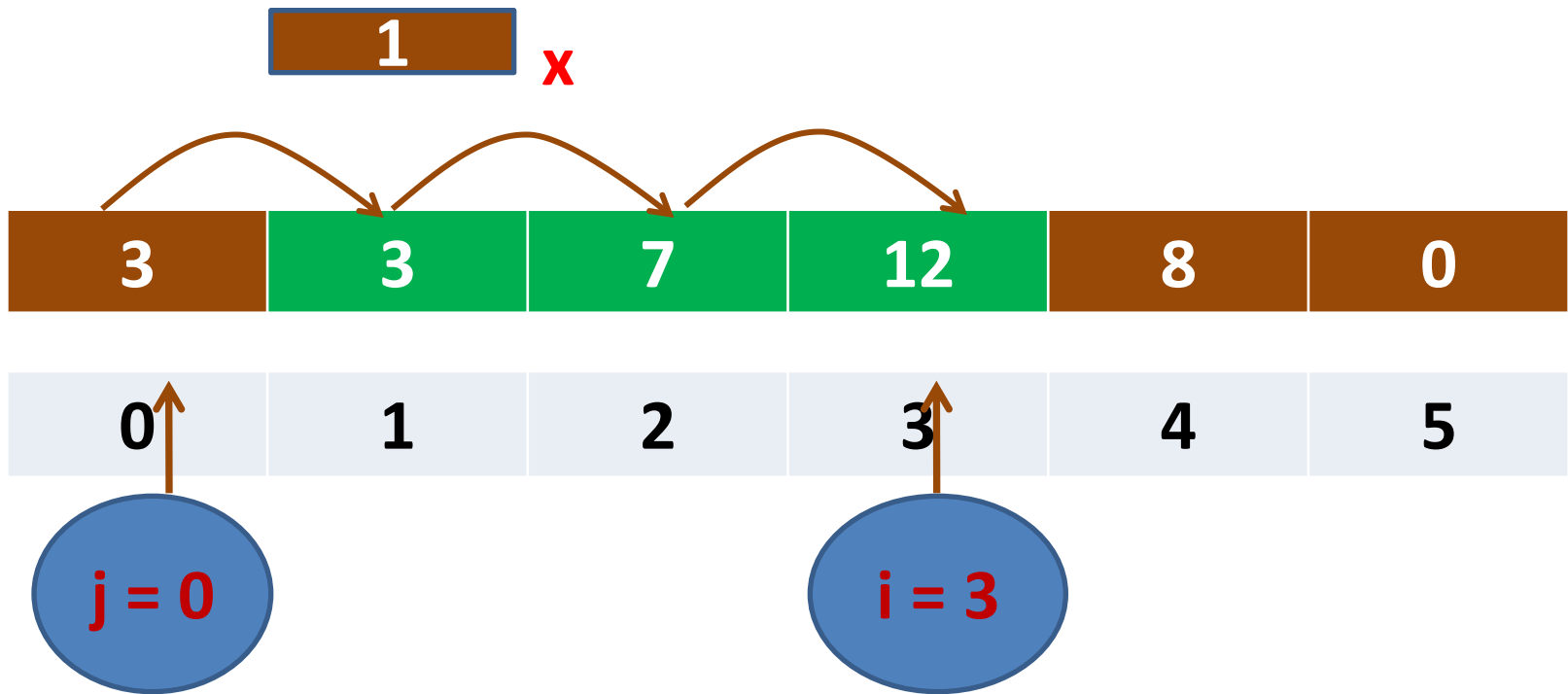
## Сортировка вставкой : 2-й проход



## Сортировка вставкой : 3-й проход

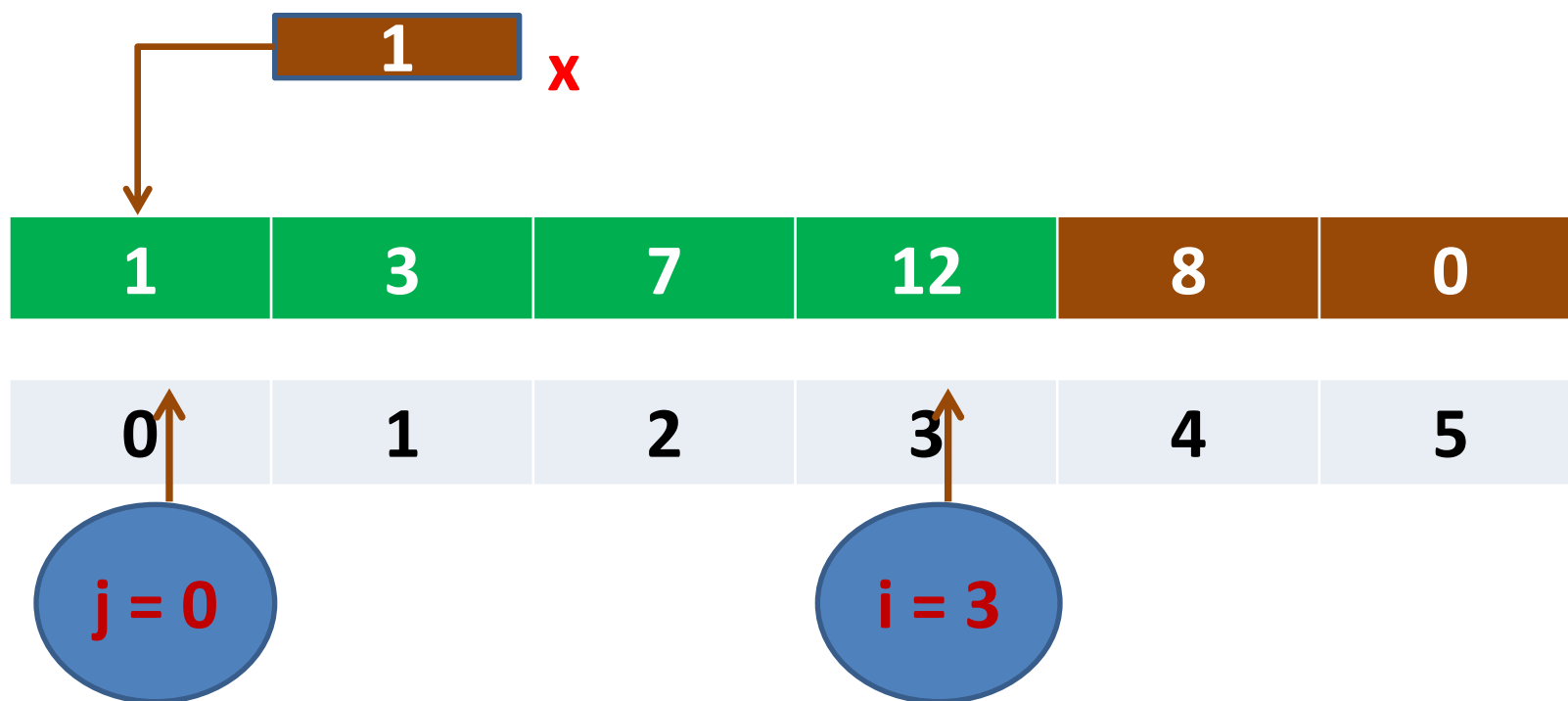


## Сортировка вставкой : 3-й проход





## Сортировка вставкой : 3-й проход



## Простейшая программа сортировки вставкой:

```
void InsertSort (int *a,int size) {  
    //последовательный перебор не отсортированных эл-тов массива  
    for (int i=1;i< size;i++) {  
        int x = a[i]; //взятие очередного элемента  
        int j = i-1;  
        while (x<a[j]) { // повторять пока место вставки не найдено  
            // сдвиг текущего j- го элемента на 1 позицию вправо  
            a[j+1] = a[j];  
            j--;  
            if (j<0) break; //условие выхода при достижении левой границы  
        }  
        a[j+1] = x; // вставка взятого элемента  
    }  
}  
  
... InsertSort(b,n1);
```

$$C_{\min} = n - 1;$$

$$C_{cp} = \frac{1}{4}(n^2 + n - 2);$$

$$C_{\max} = \frac{1}{2}(n^2 + n) - 1;$$

$$M_{\min} = 2(n - 1);$$

$$M_{cp} = \frac{1}{4}(n^2 + 9n - 10);$$

$$M_{\max} = \frac{1}{2}(n^2 + 3n - 4);$$

# Сортировка Шелла

**Шаг 0.**  $i = t$  .

**Шаг 1.** Разобьем массив на списки элементов, отстающих друг от друга на  $h_i$ . Таких списков будет  $h_i$ .

**Шаг 2.** Отсортируем элементы каждого списка сортировкой вставками.

**Шаг 3.** Объединим списки обратно в массив.

Уменьшим  $i$ . Если  $i$  неотрицательно — вернемся к шагу 1

```
void ShellSort(int *ms, int k) {  
    int i, j, n;  
    int step; // шаг сортировки  
    int flg; // флаг окончания этапа сортировки  
    for(step = k/2; step > 0; step /= 2)  
        do  
        { flg = 0;  
          for(i = 0, j = step; j < k; i++, j++)  
              if(*(ms+i) > *(ms+j)) // сравниваем отстоящие на step эл-ты  
              { n = *(ms+j);  
                *(ms+j) = *(ms+i);  
                *(ms+i) = n;  
                flg = 1; // есть еще не рассортированные данные  
              }  
        } while (flg); // окончание этапа сортировки  
}
```

41 53 11 37 79 19 7 61 – исходный массив.

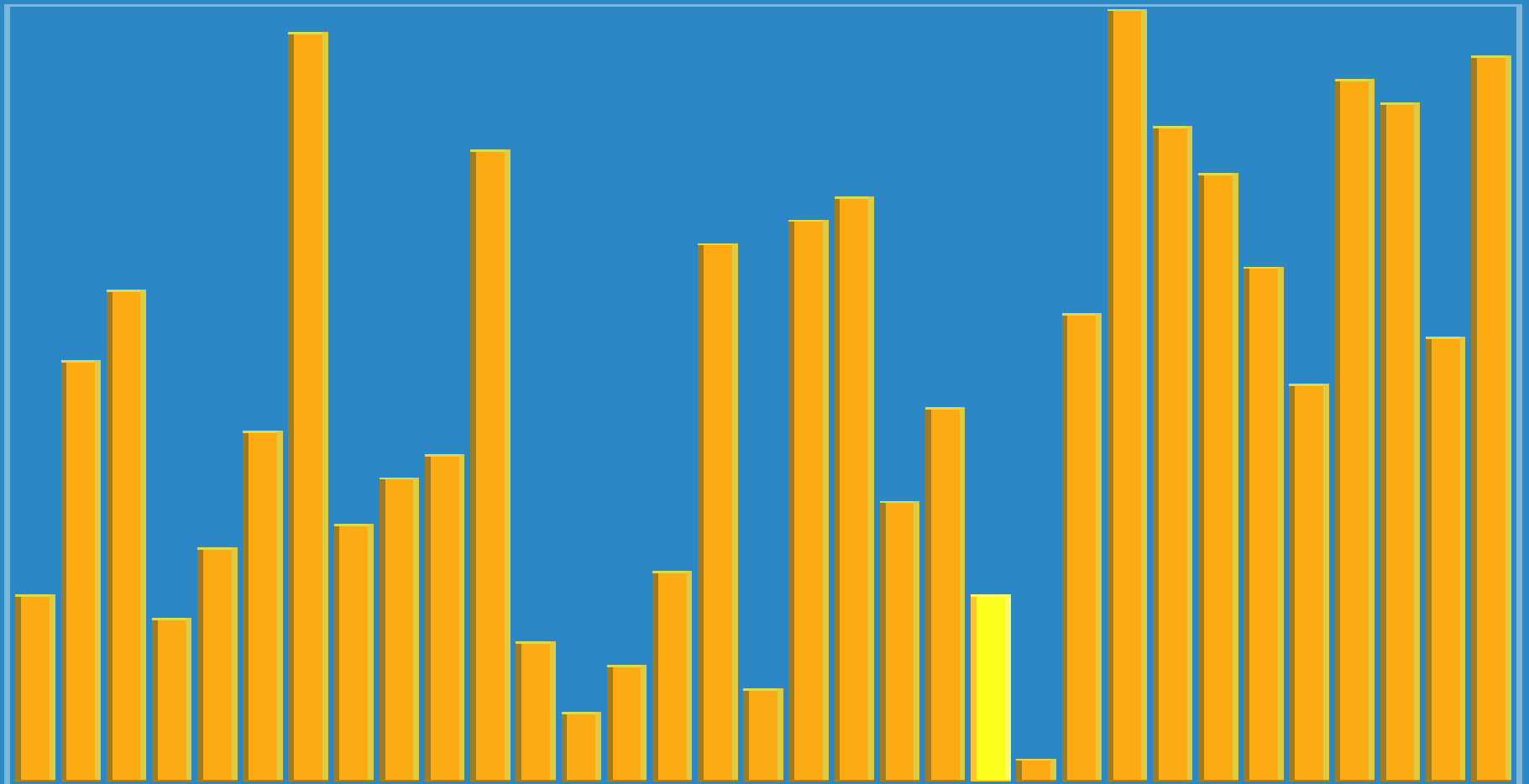
41 19 11 37 79 53 7 61 – (0,4), (1,5) 1-ый цикл

41 19 7 37 79 53 11 61 – (2,6), (3,7)

7 19 41 37 11 53 79 61 – (0,2), (1,3), (2,4), (3,5), (4,6), (5,7) 2-ой цикл

7 19 11 37 41 53 79 61 – (0,2), (1,3), (2,4), (3,5), (4,6), (5,7)

7 11 19 37 41 53 61 79 – сравнивались соседние. (3-ий цикл).



<u>Худшее время</u>	зависит от выбранных шагов
<u>Лучшее время</u>	$O(n \cdot k)$ сравнений, $O(k)$ обменов, где $k$ - количество шагов
<u>Среднее время</u>	зависит от выбранных шагов
<u>Затраты памяти</u>	$O(n)$ всего, $O(1)$ дополнительно

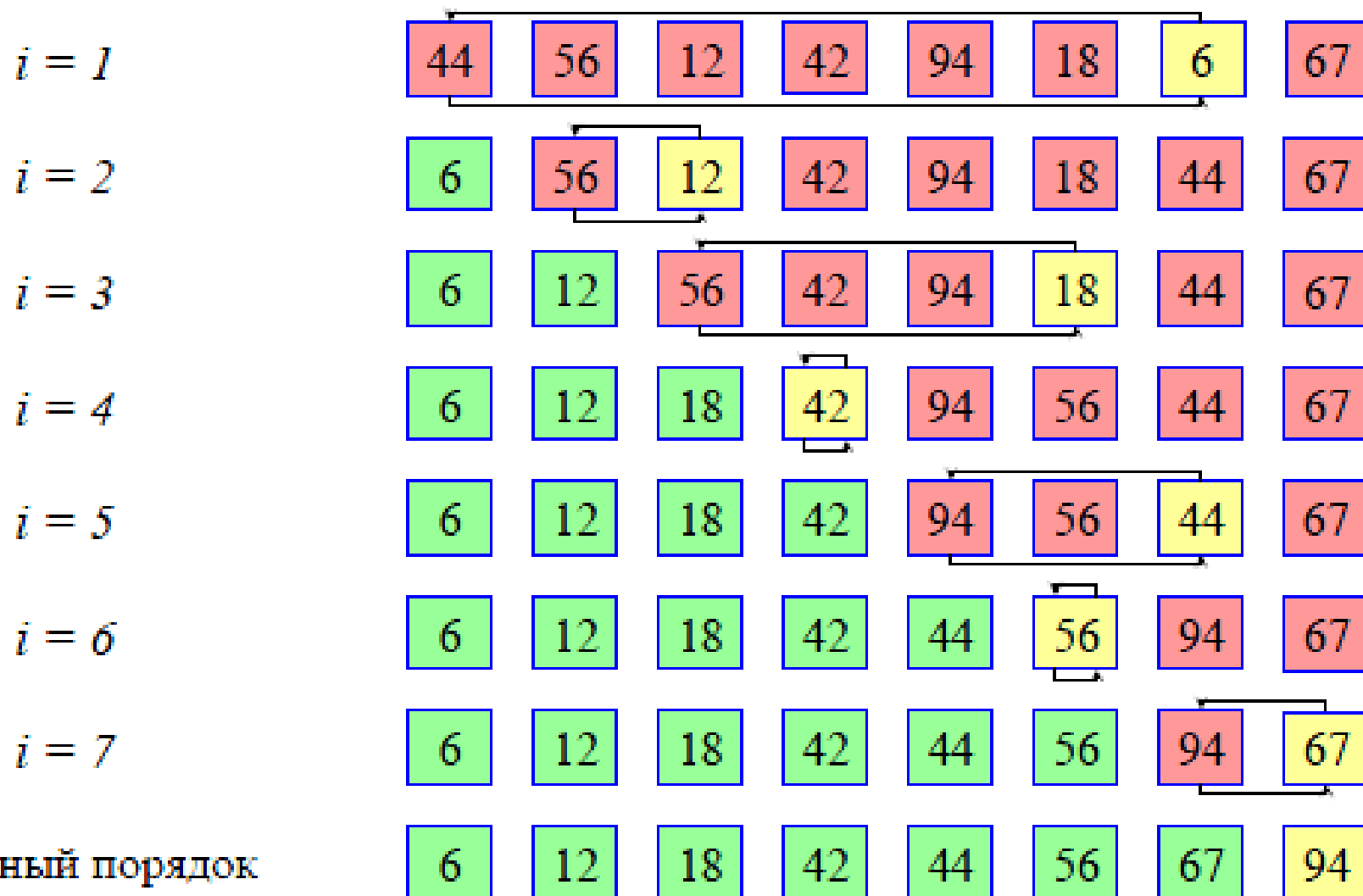
первоначально используемая Шеллом последовательность длин промежутков:

в худшем случае, сложность алгоритма  $O(N^2)$ ;

предложенная Хиббардом последовательность: все значения  $2^i - 1 \leq N, i \in \mathbb{N}$ ;

сложность алгоритма  $O(N^{3/2})$ ;

# Алгоритм сортировки простым выбором



Пример сортировки выбором: ■ – отсортированная часть, ■ – входная часть, ■ – текущий минимум входной (не отсортированной) части



1. Для всех  $i$  от 0 до  $n-2$  выполнить:

1.1. Взять очередной  $i$ -й не отсортированный элемент и сохранить его в рабочей переменной  $min$

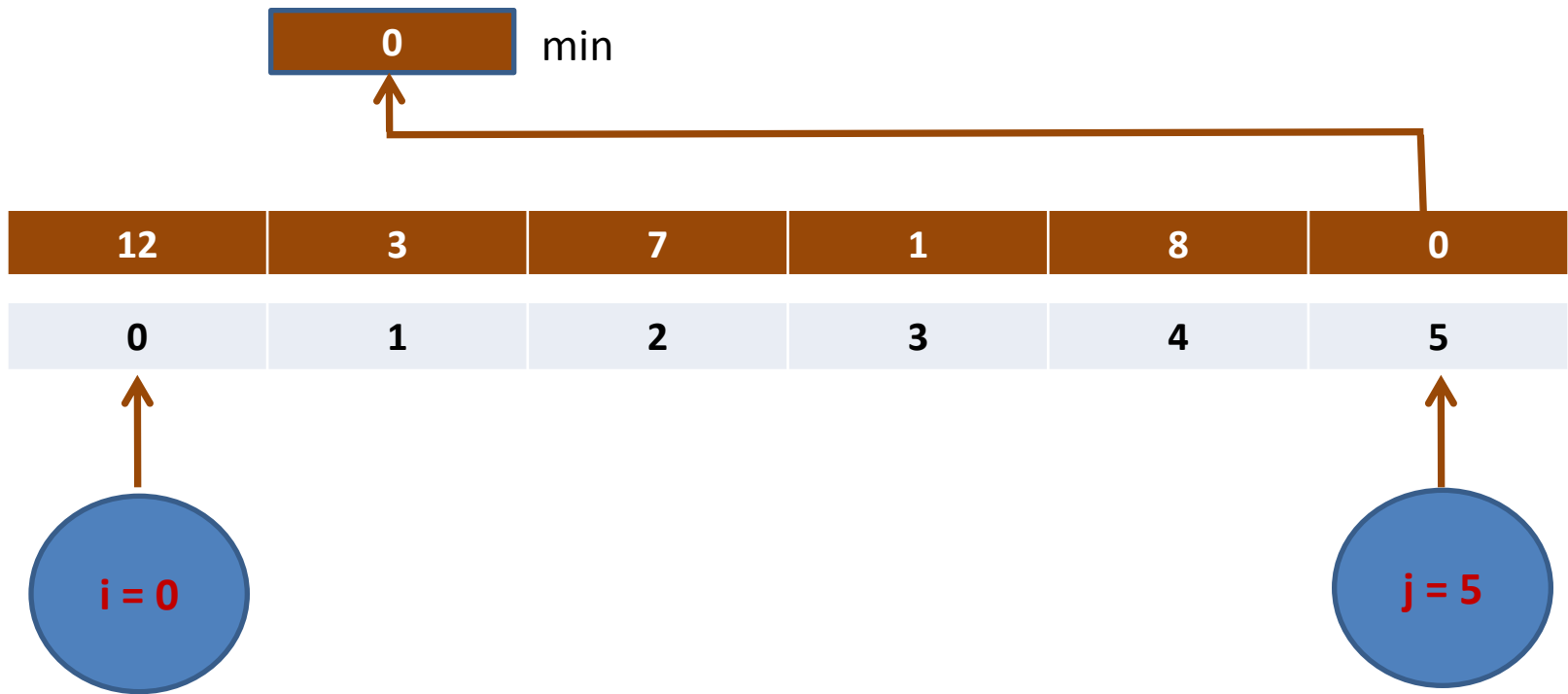
1.2. Присвоить переменной  $index\_min$  значение  $i$ ;

1.3. Найти минимум в части массива от  $a[i+1]$  до  $a[n-1]$  и запомнить его позицию в переменной  $index\_min$ ;

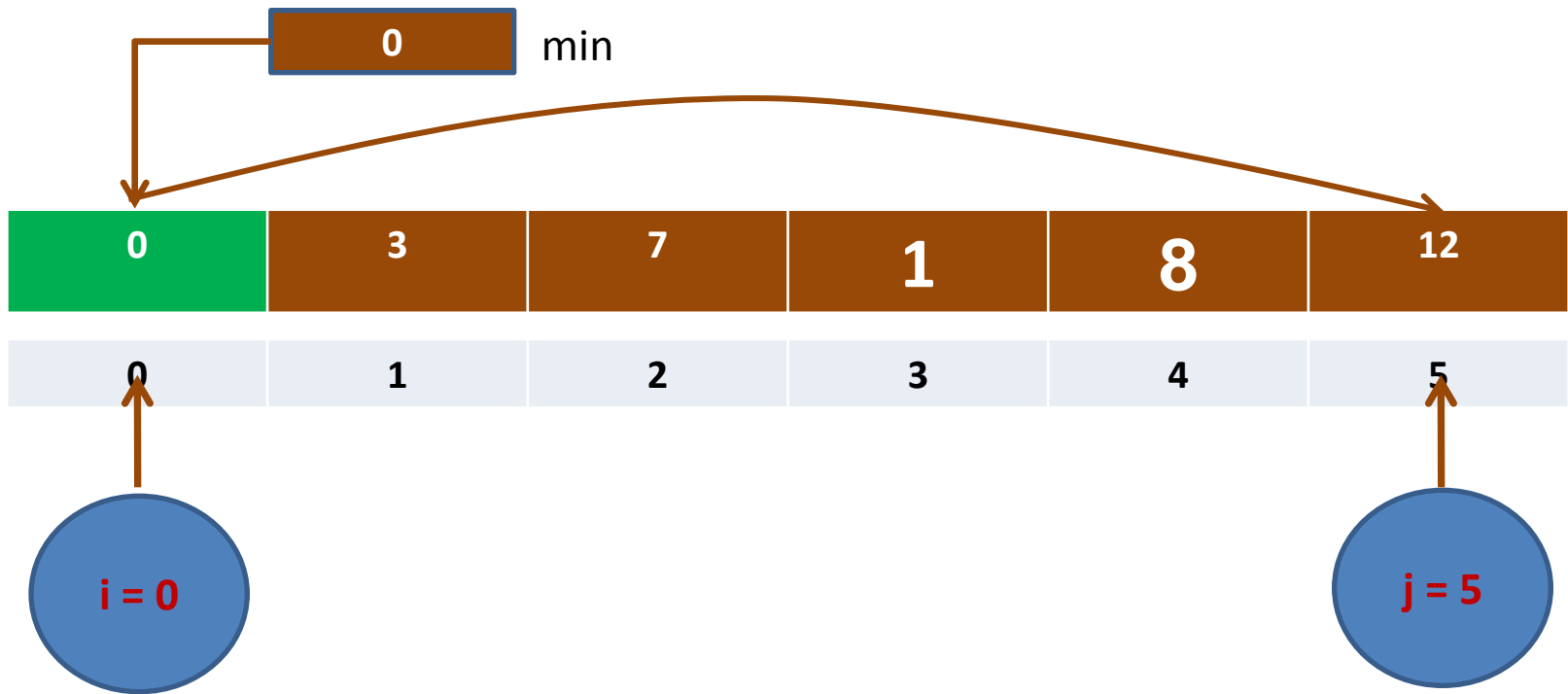
1.4. Поменять местами элементы  $a[i]$  и  $a[index\_min]$ ;

```
void SortSelect (int *a,int size) {  
    //послед-й перебор всех эл-в кроме последнего  
    for (int i=0;i<size-1;i++) {  
        int min = a[i];  
        //присвоение перемен-й минимум текущего элемента  
        int index_min = i;  
        // запоминаем индекс текущего элемента  
        // поиск минимума в части массива от i+1 до конца  
        for (int j=i+1;j< size;j++)  
            if (a[j] < min) { min = a[j];  
                // запоминаем текущий найденный минимум  
                index_min = j; // запоминаем его индекс  
            }  
        //обмен местами текущего элемента и найденного  
        //минимального  
        a[index_min] = a[i]; a[i] = min;  
    } }
```

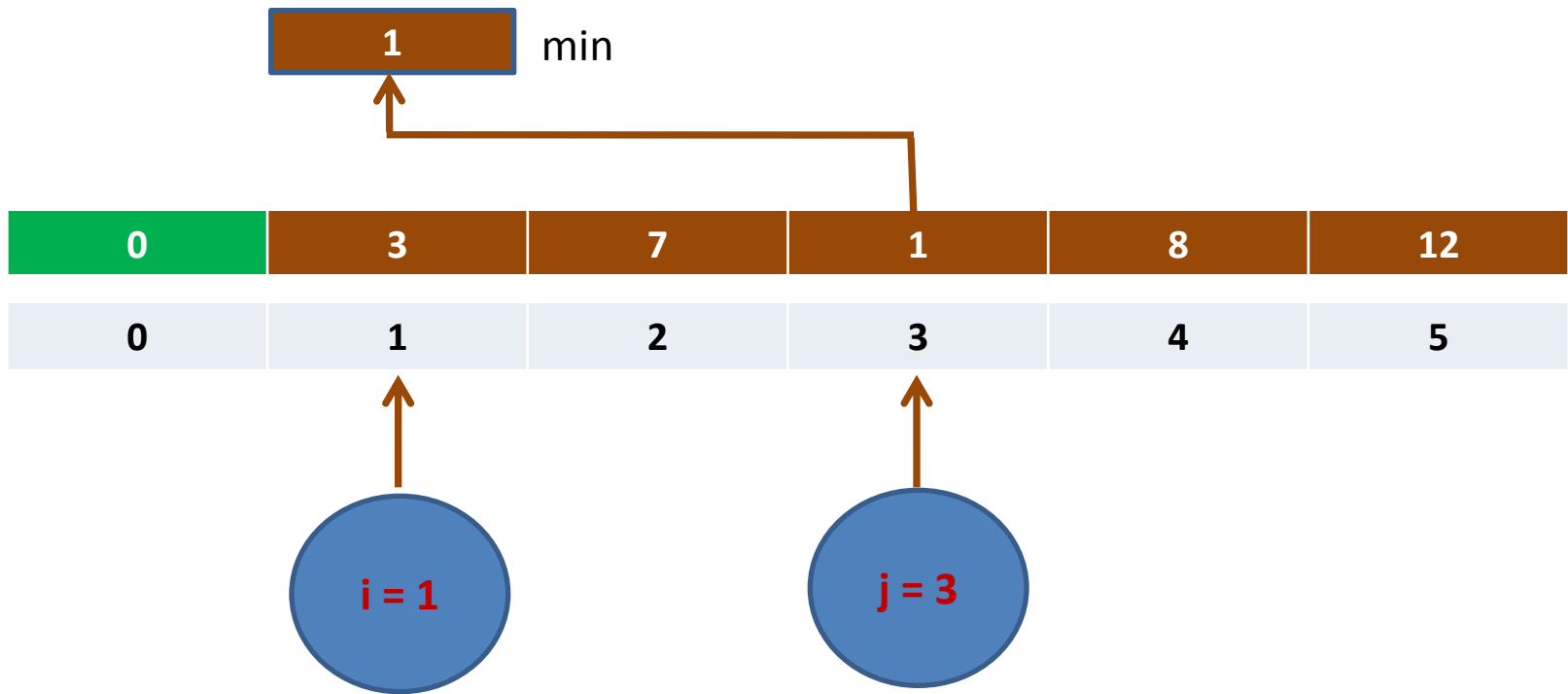
## Сортировка выбором : 1-й проход



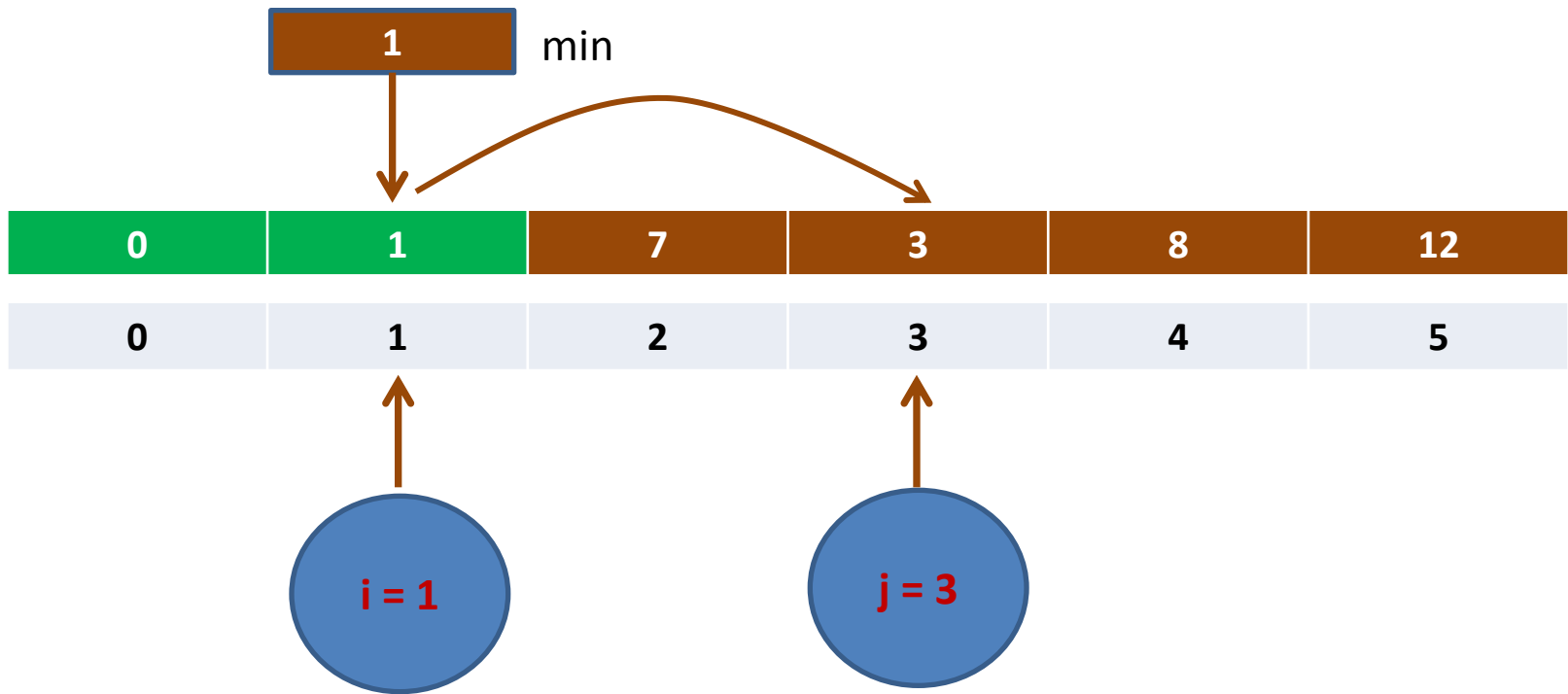
## Сортировка выбором : 1-й проход



## Сортировка выбором : 2-й проход



## Сортировка выбором : 2-й проход



$$C = \frac{1}{2}(n^2 - n)$$

$$M_{\min} = 3(n-1)$$

$$M_{\max} = \left\lceil \frac{n^2}{4} \right\rceil + 3(n-1)$$

$$M_{\varphi} = n(\ln n + \gamma)$$

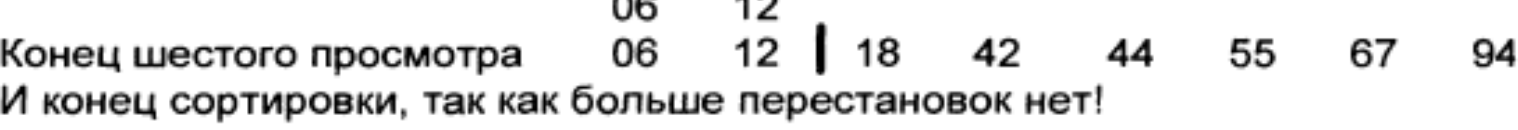
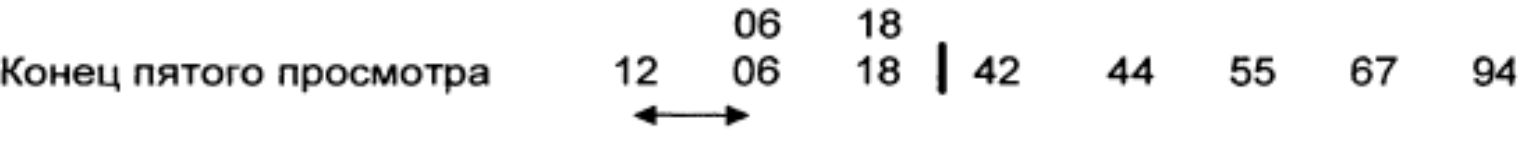
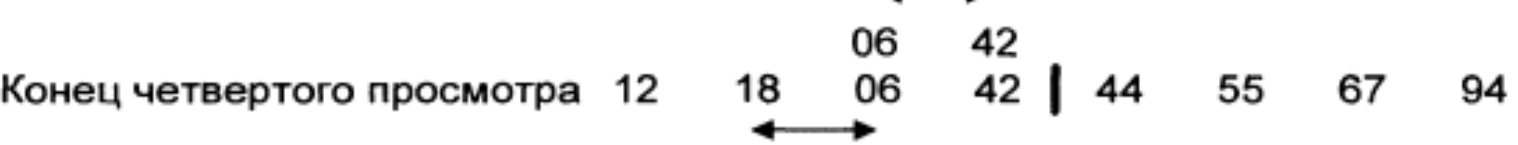
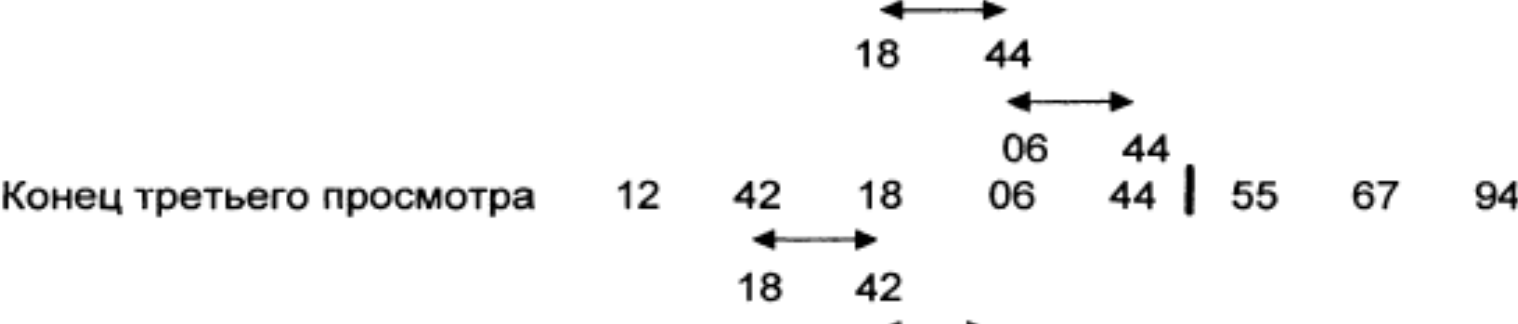
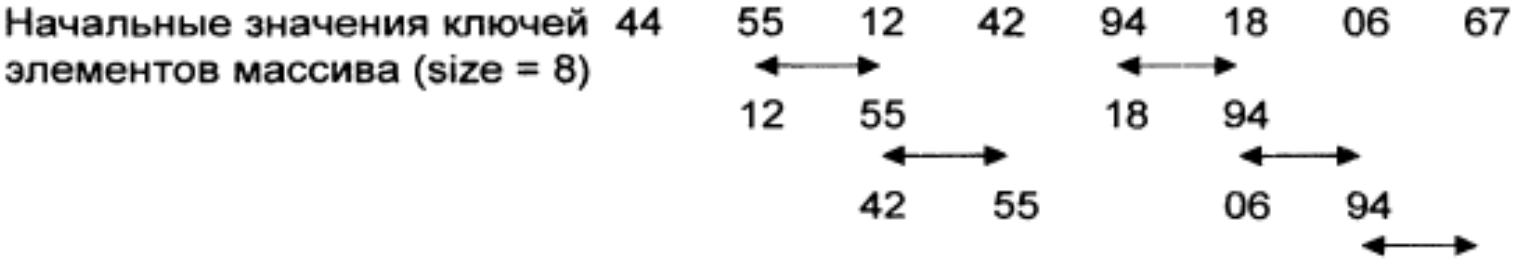
# Алгоритм сортировки обменом (метод «пузырька»)

1. Для всех  $i$  от 1 до  $n-1$  выполнять :

1.1. Слева направо поочередно сравнивать два соседних элемента и если их взаиморасположение не соответствует заданному условию упорядоченности, то менять их местами.







# Алгоритм сортировки обменом (метод «пузырька»)

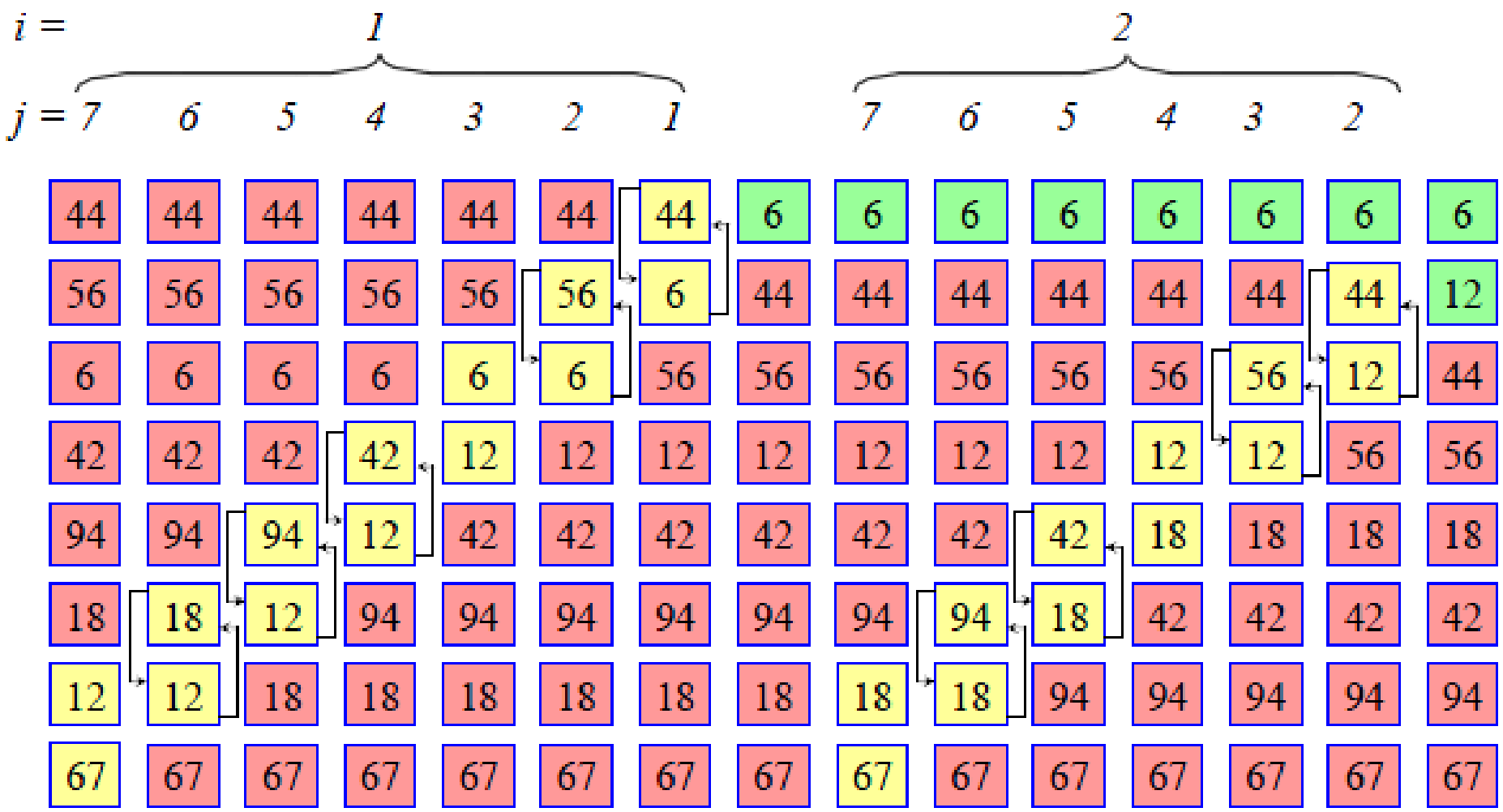


Рис. 3. Пример двух проходов сортировки методом пузырька: ■ – отсортированная часть, ■ – входная часть, ■ – пара сравниваемых элементов.

```
void SortBubble(int *a, int size) {  
    //повторять проходы по массиву n-1 раз  
    for(int i=1;i< size;i++)  
    {  
        //проход с n – 1-го элемента вверх до i-го  
        for (int j= size -1; j >= i; j--)  
            if (a[j-1]>a[j])  
            {  
                // обмен элементов в случае неправильного порядка  
                int x = a[j-1]; a[j-1] = a[j];  
                a[j] = x;  
            }  
    }  
}
```

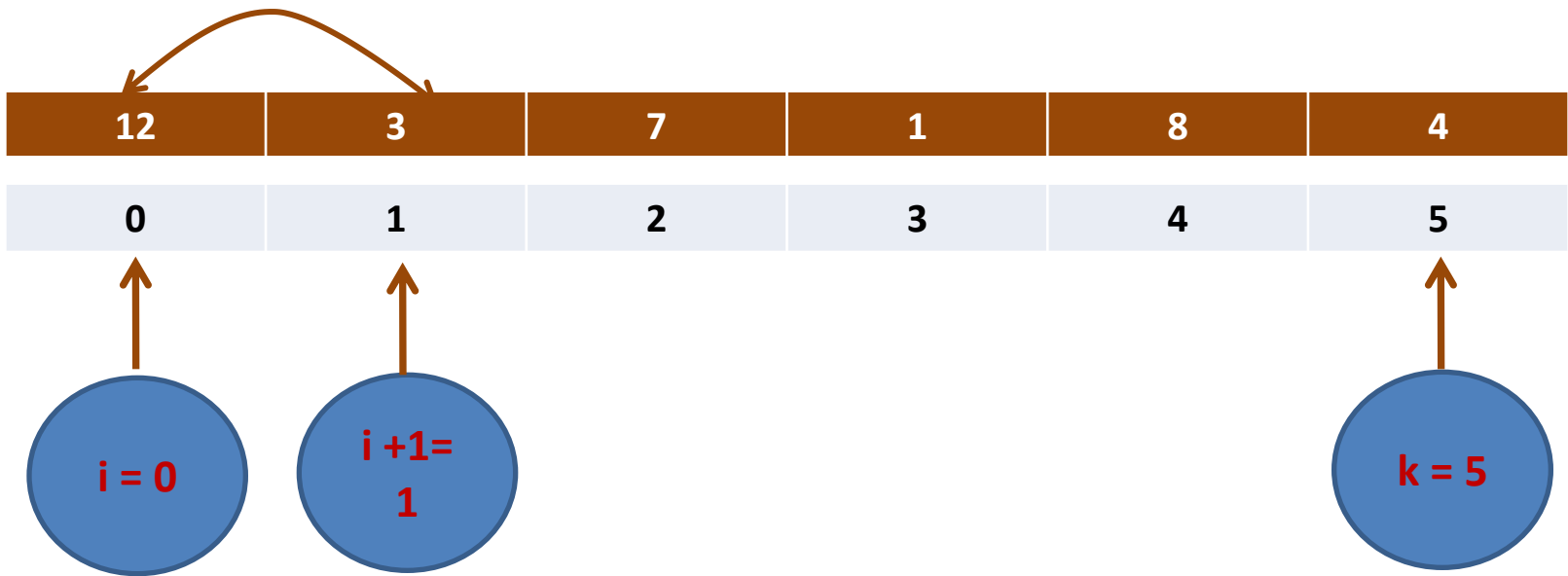
Число сравнений в алгоритме  
простого обмена :

$$C = \frac{1}{2}(n^2 - n)$$

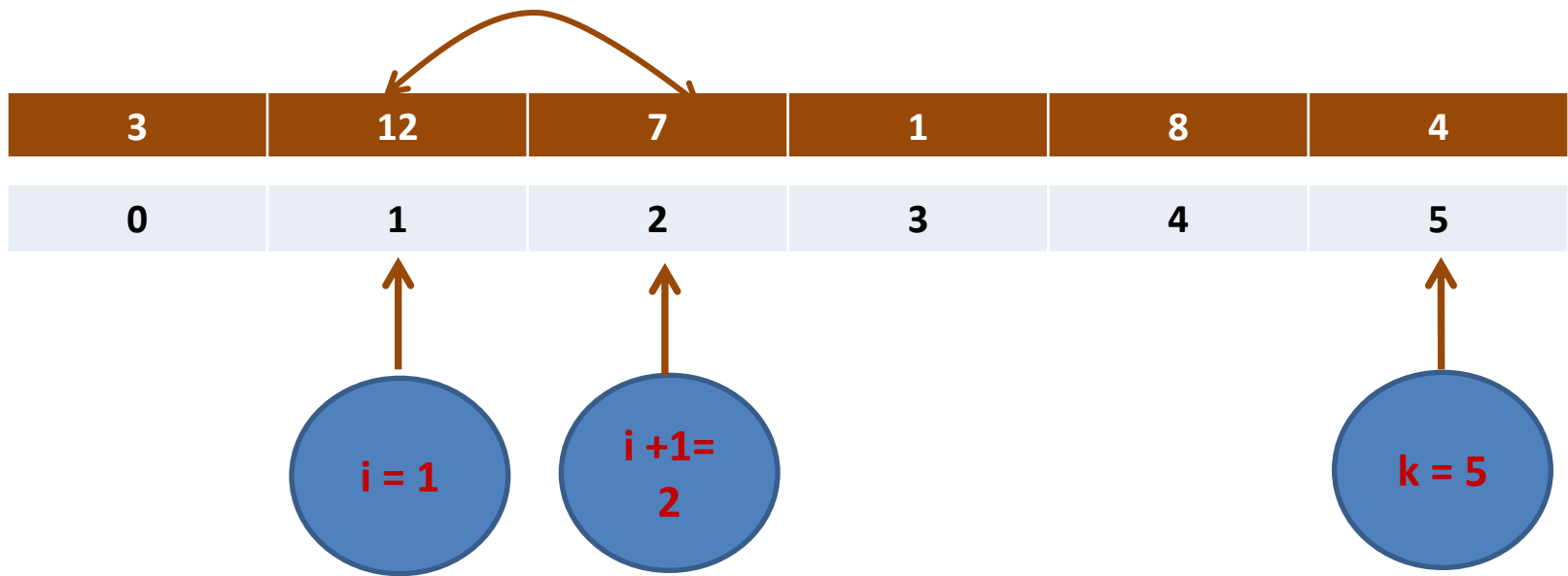
Минимальное, среднее и  
максимальное количества присваиваний :

$$M_{\min} = 0, \quad M_{\text{cp}} = \frac{1}{4}(n^2 - n), \quad M_{\max} = \frac{1}{2}(n^2 - n)$$

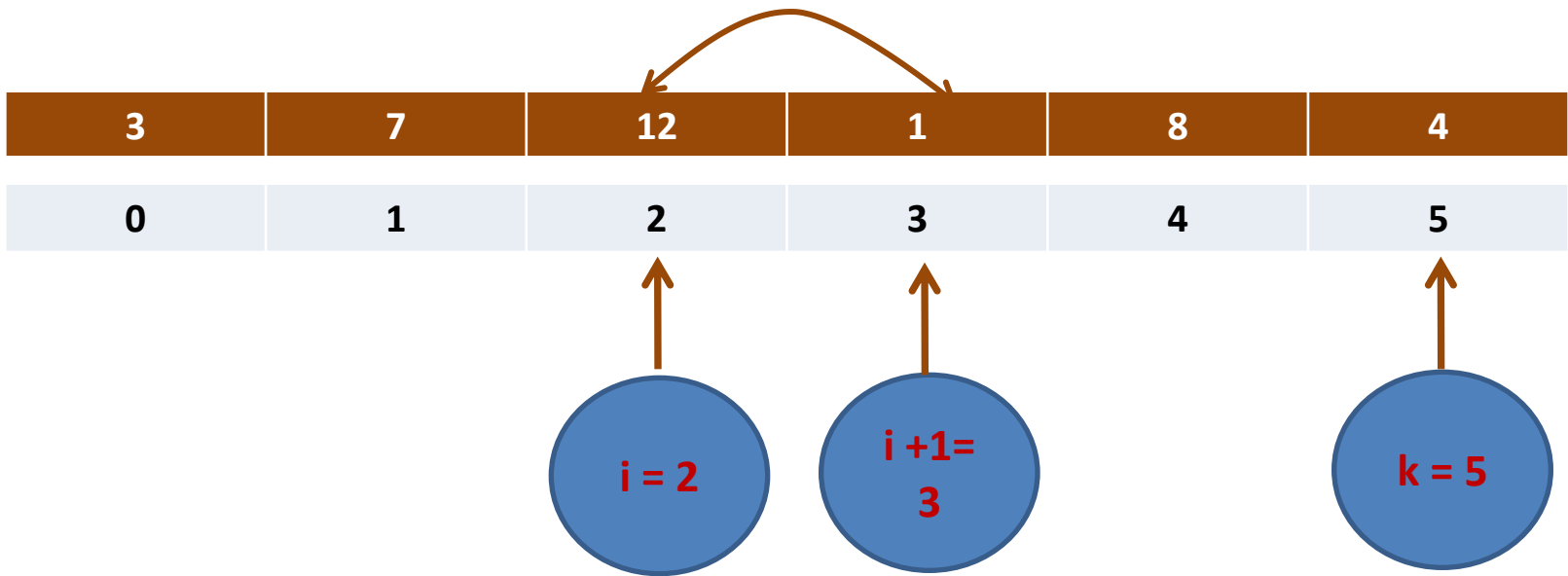
## Сортировка обменом: 1-й проход



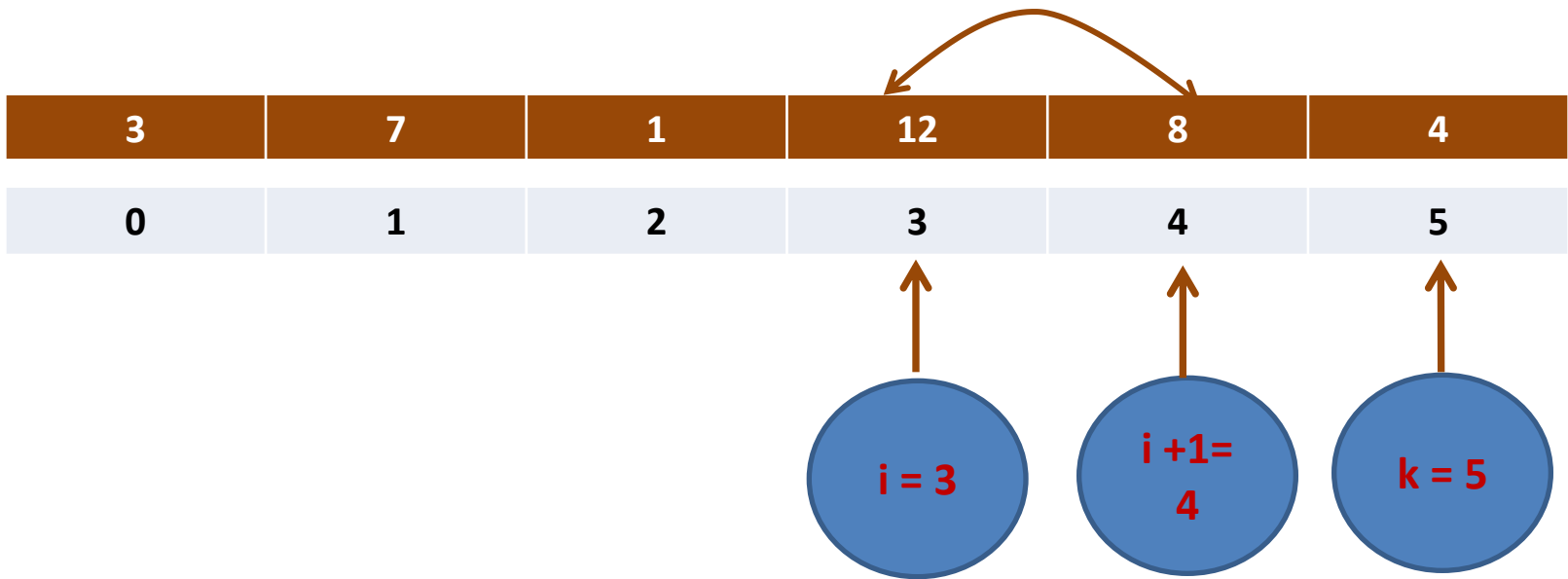
## Сортировка обменом: 1-й проход



## Сортировка обменом: 1-й проход

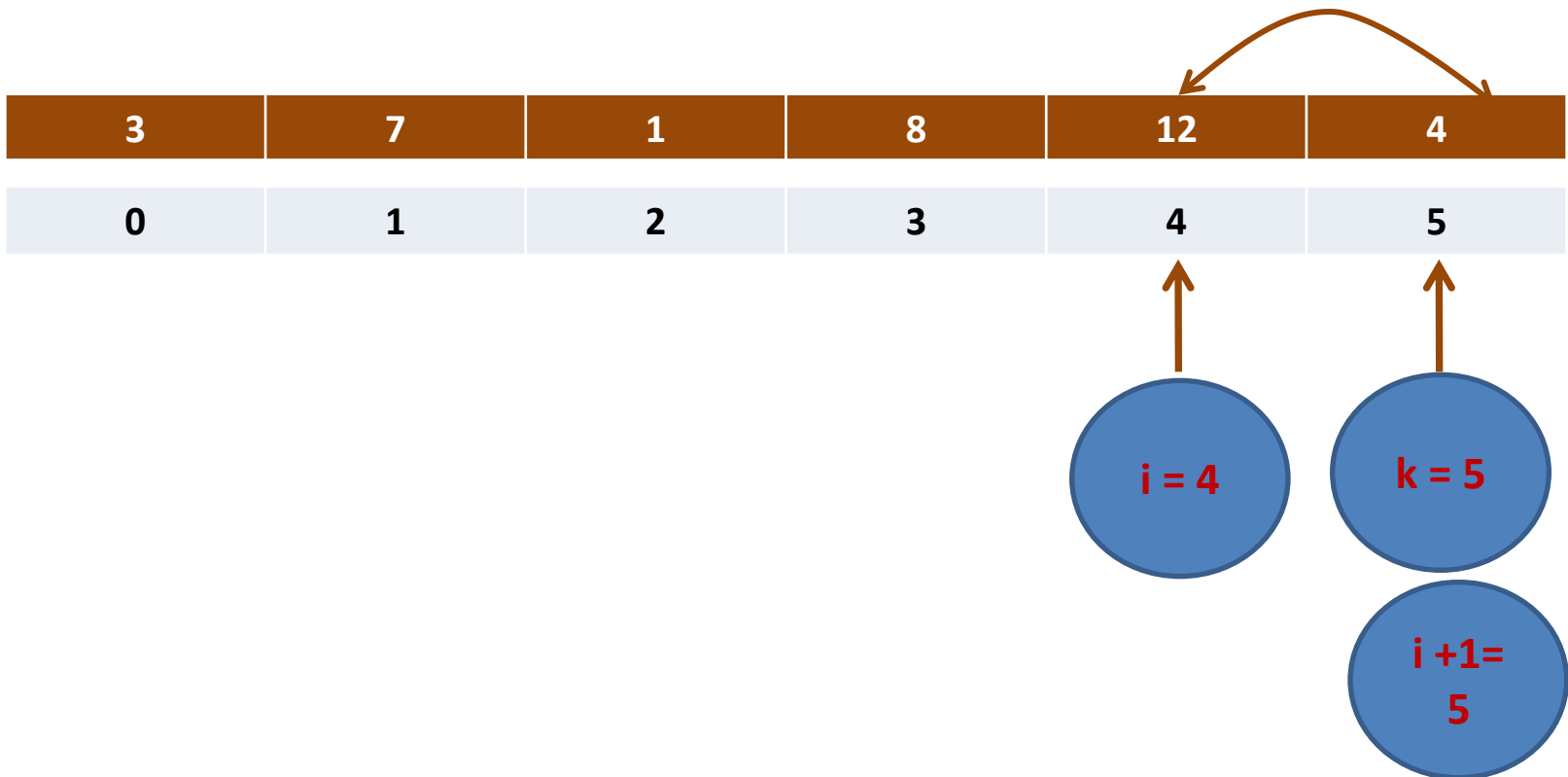


## Сортировка обменом: 1-й проход

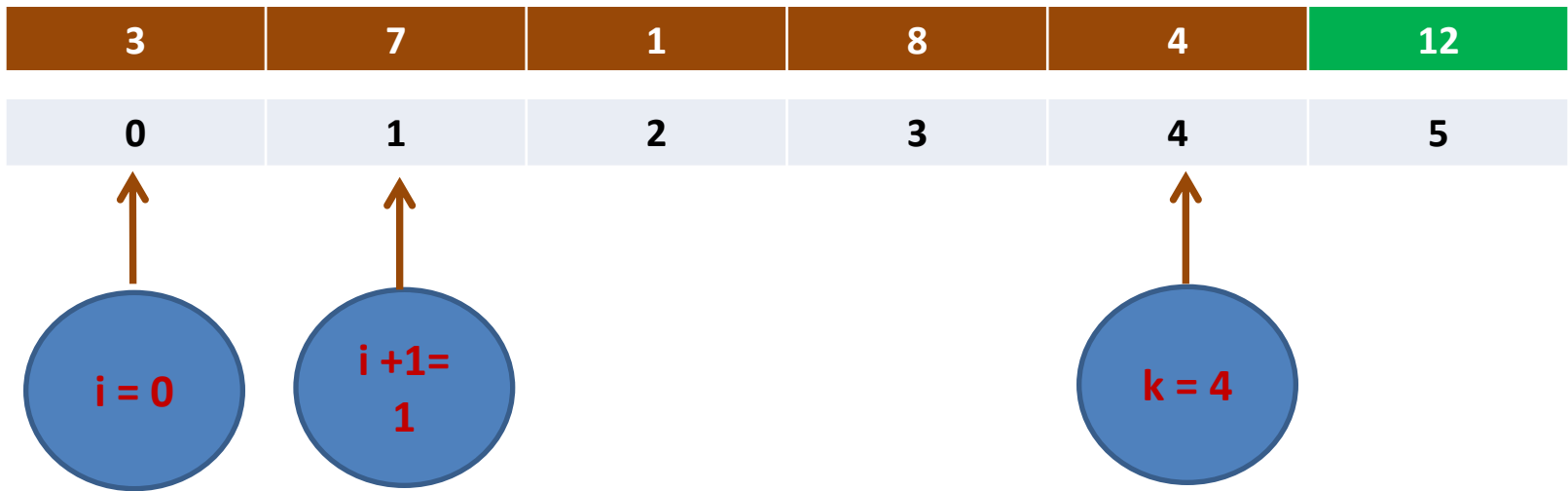




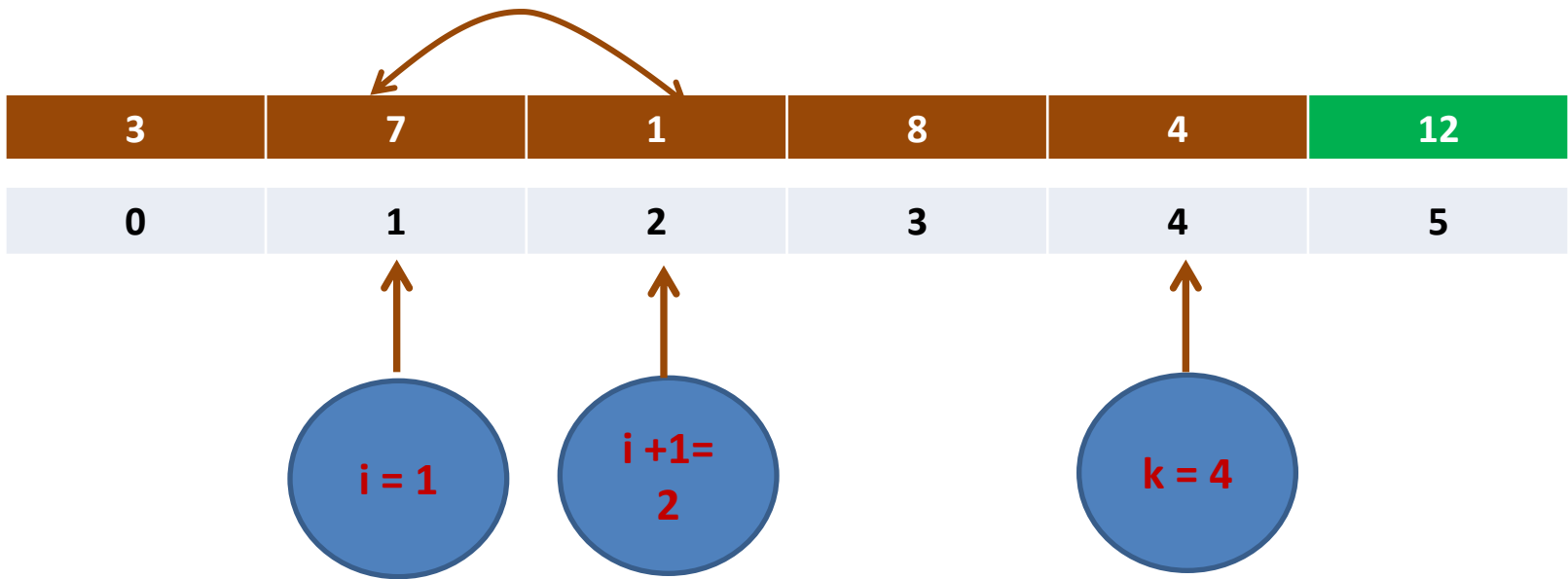
## Сортировка обменом: 1-й проход



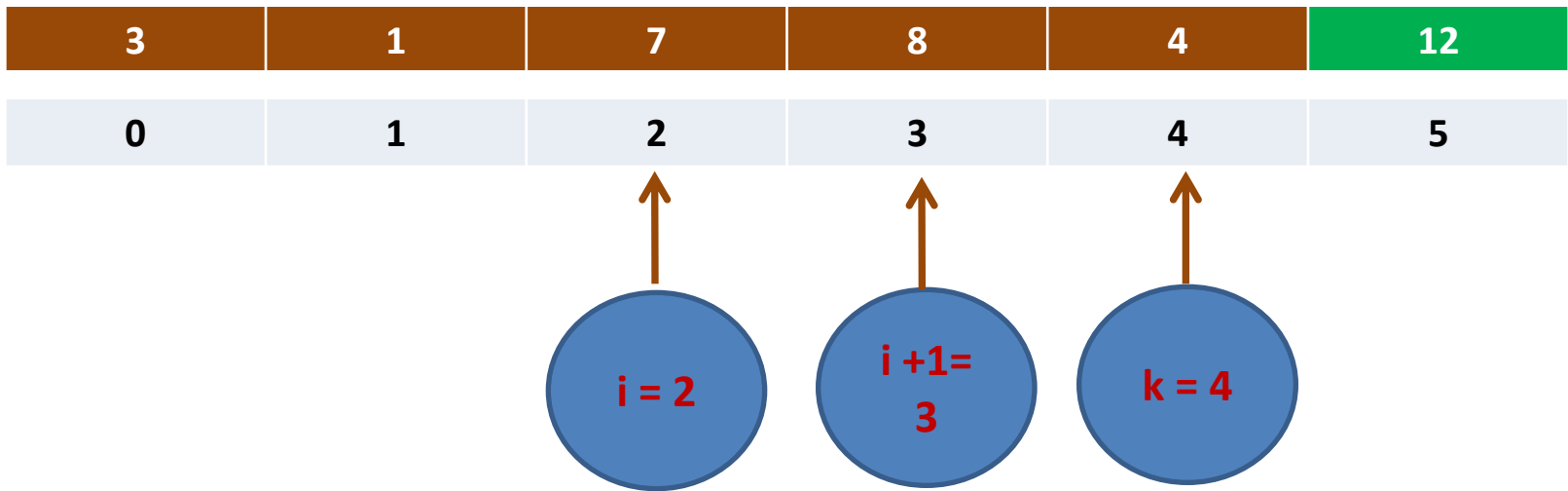
## Сортировка обменом: 2-й проход



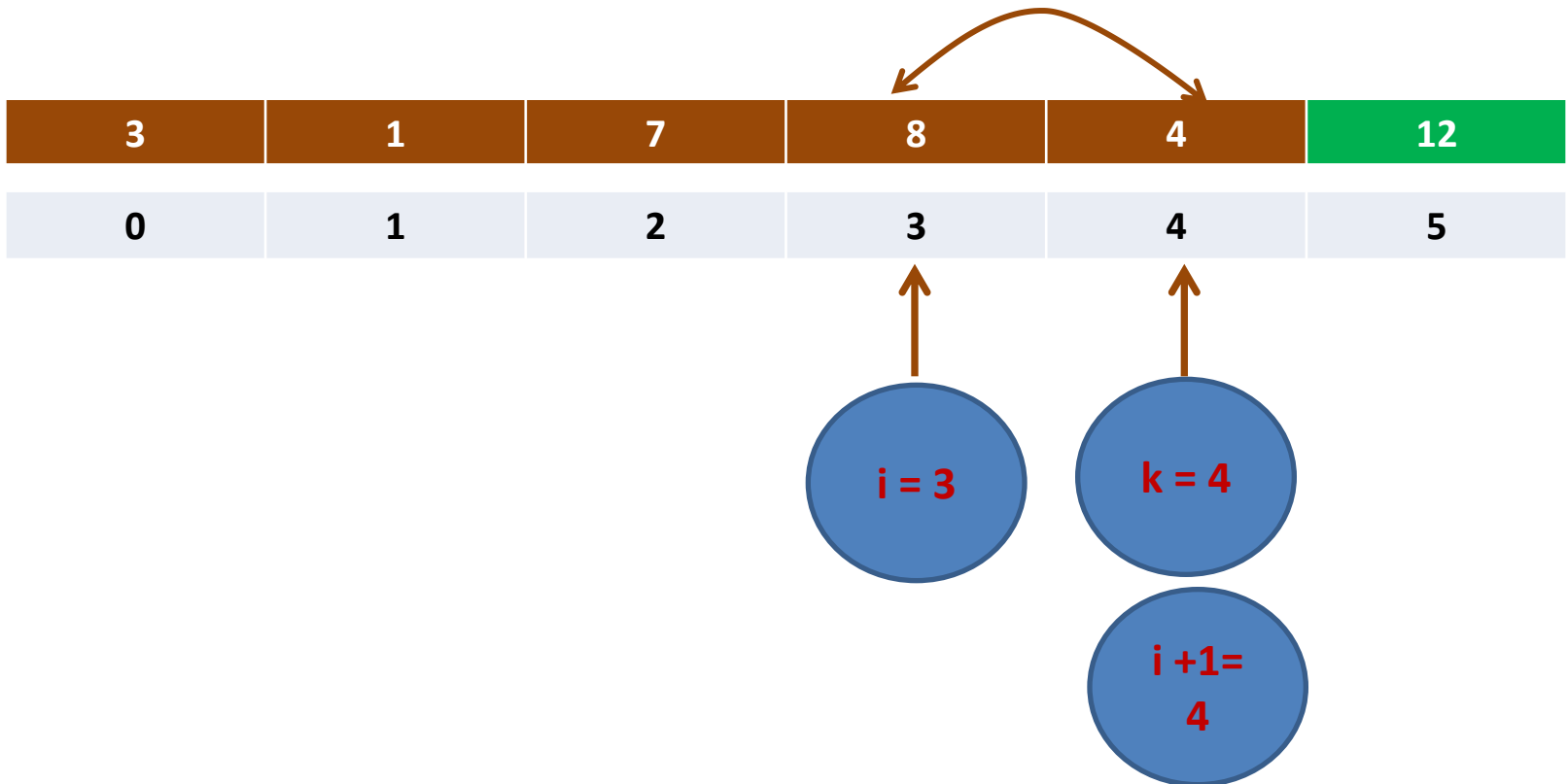
## Сортировка обменом: 2-й проход



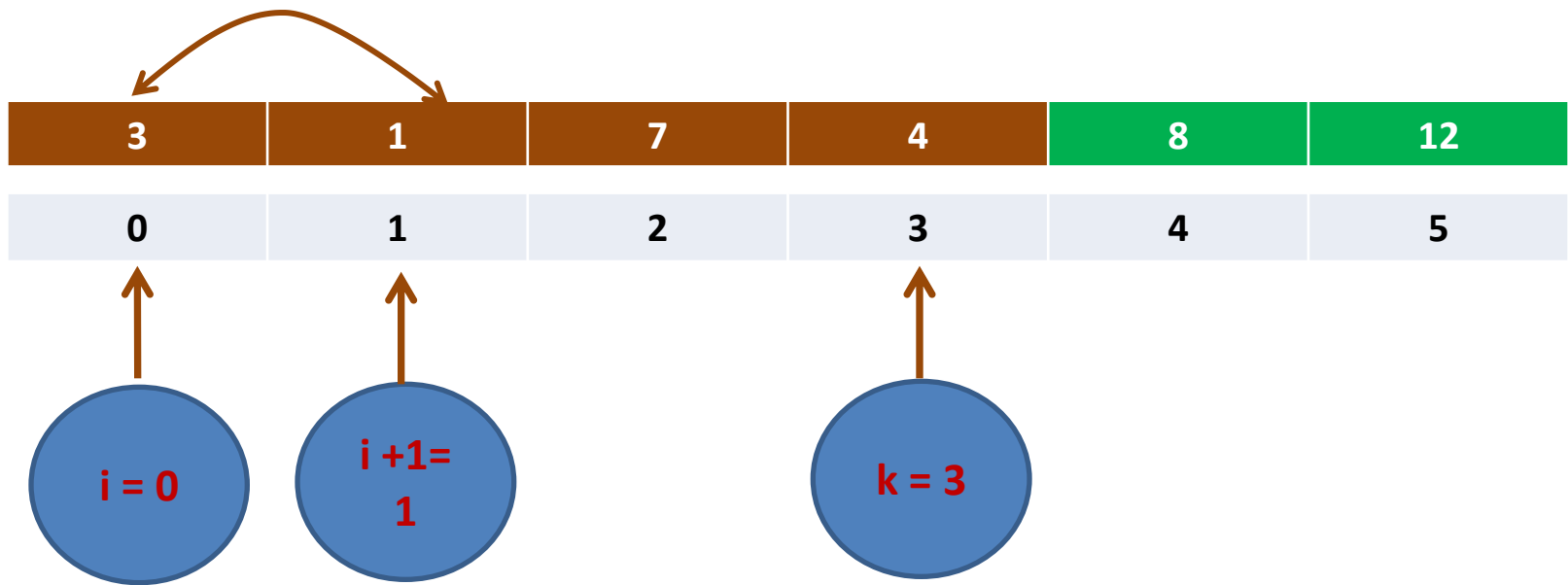
## Сортировка обменом: 2-й проход



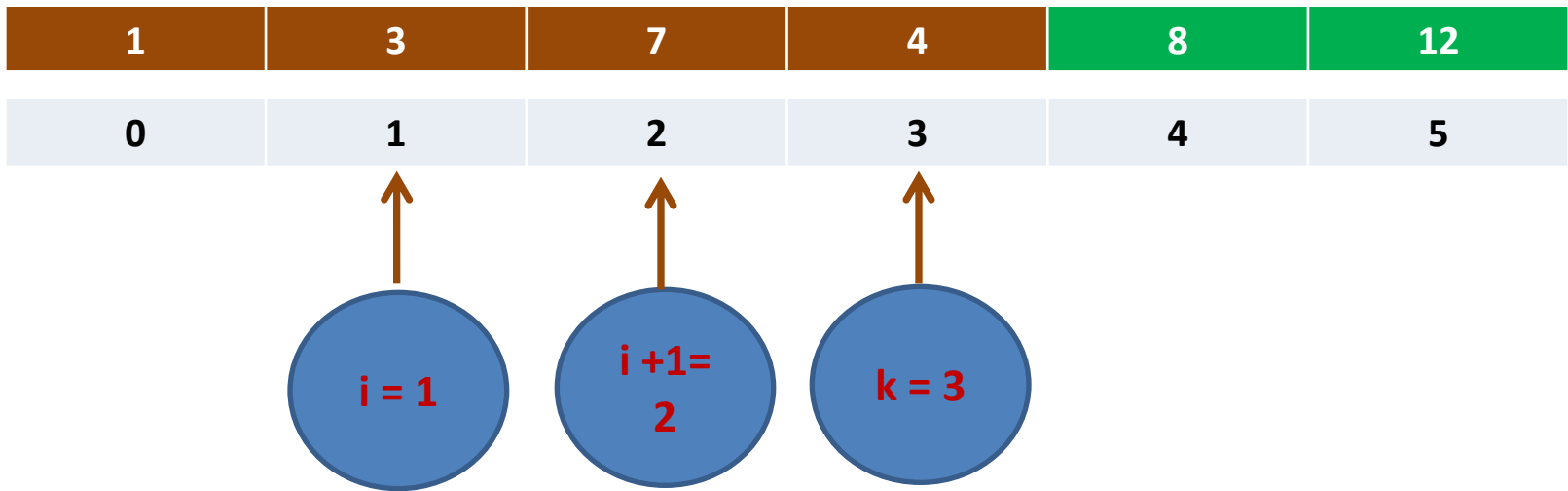
## Сортировка обменом: 2-й проход



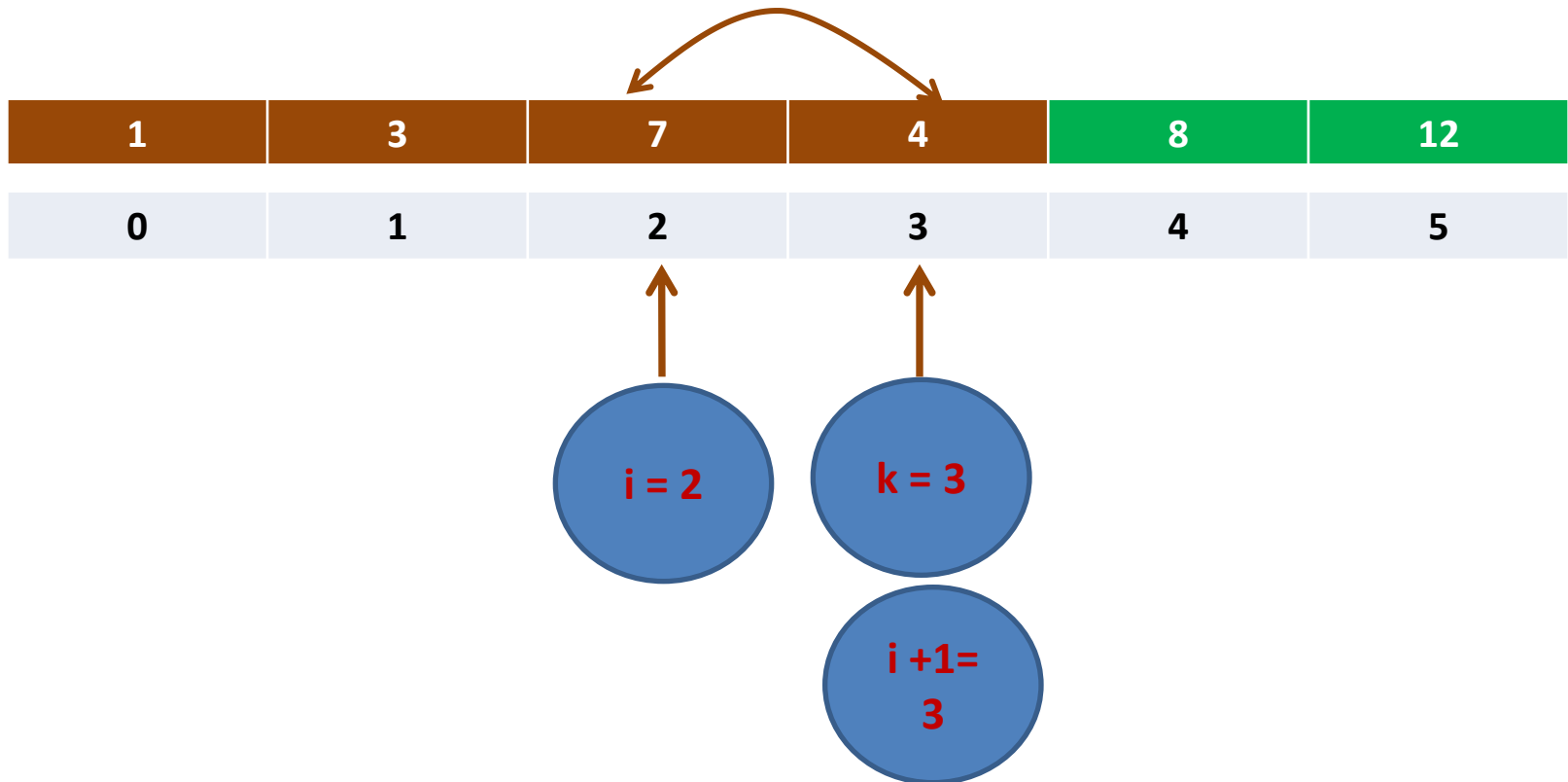
## Сортировка обменом: 3-й проход



## Сортировка обменом: 3-й проход

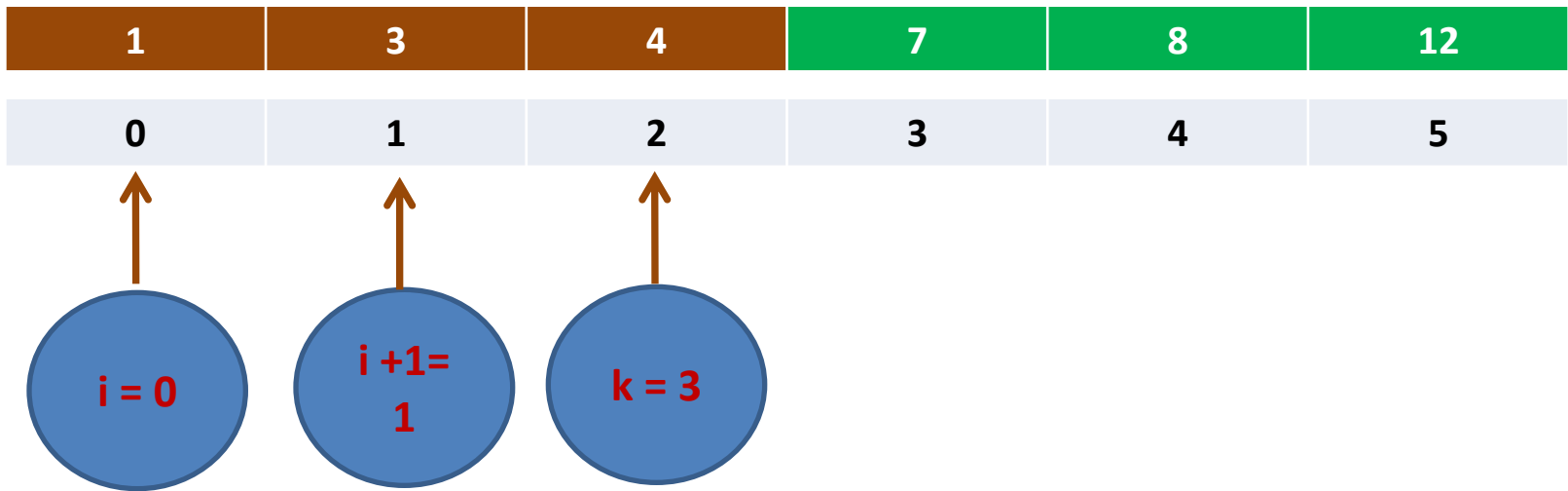


## Сортировка обменом: 3-й проход





Сортировка обменом: 4-й проход ?



# Усовершенствованный метод пузырька – Шейкер сортировка

```
void ShakerSort(int *a, int size) {  
    int left = 1, right = size-1, last = right; //границы участка  
    do {  
        // повторять до тех пор пока границы не сомкнутся  
        for (int j = right; j >= left; j--)  
            // движение справа налево  
            if (a[j-1] > a[j]) // проверка условия обмена пары  
            { // обмен элементов в случае неправильного порядка  
                int temp = a[j-1];  
                a[j-1] = a[j];  
                a[j] = temp;  
                last = j;  
            }  
        // запоминание индекса последнего обмена  
    }
```

```
left = last + 1; //расчет новой левой границы
for (int j = left; j < right+1; j++)
// движение слева направо
if (a[j-1] > a[j])
// проверка условия обмена пары
{ // обмен элементов в случае неправильного порядка
int temp = a[j-1];
a[j-1] = a[j];
a[j] = temp;
last = j; // запоминание индекса последнего обмена
}
right = last - 1; //расчет новой правой границы
} while(left < right);
// повторять до тех пор пока границы не сомкнутся
}
```

Для метода шейкер-сортировки наименьшее число сравнений  $C_{min} = n - 1$ .

Среднее число сравнений пропорционально  $1/2[n^2 - n(k^2 + \ln n)]$ .

усовершенствования не влияют на число обменов, а лишь уменьшают число избыточных повторных проверок.

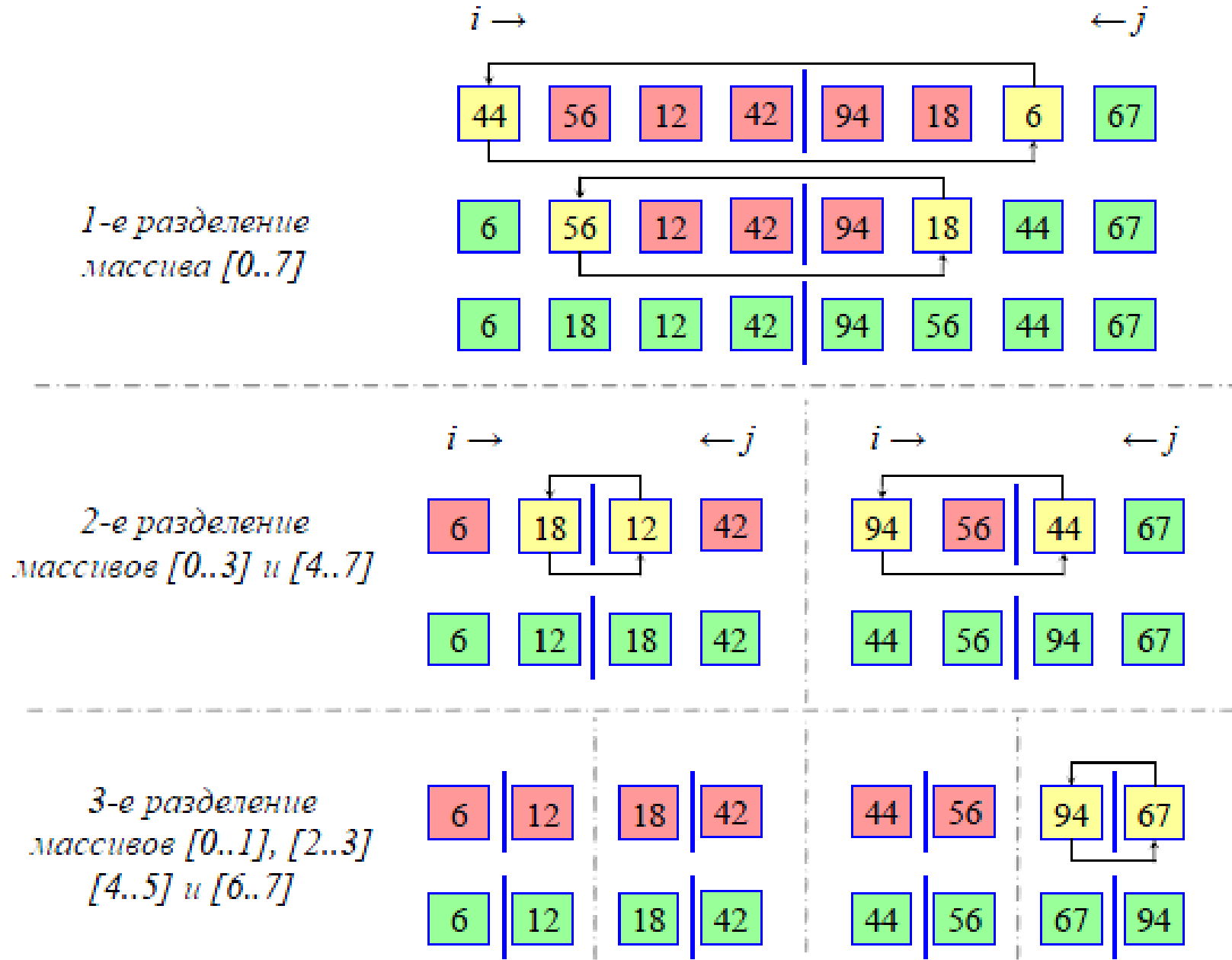


Рис. 4. Пример сортировки разделением:  – текущая отсортированная часть,  – входная часть,  – текущая пара обмениваемых элементов.

# Алгоритм быстрой сортировки

```
void QuickSort(int a[], int left, int right) {  
    //присвоение параметрам счетчиков текущих левой и  
    //правой границ части массива  
    int i = left, j = right;  
    // сохранение в mid значения среднего элемента  
    int mid = a[(left +right)/2];  
    // повторять пока проходы слева и справа не  
    //перехлестнутся
```

```
do {  
    //проход слева до первого попавшегося эл-та больше среднего  
    while (a[i] < mid) i++;  
    //проход справа до первого попавшегося эл-та меньше среднего  
    while (a[j] > mid) j--;  
    if (i <= j) // обмен местами найденных элементов  
    {  
        int temp = a[i];  
        a[i] = a[j];  
        a[j] = temp;  
    }  
    // переход к следующим для каждого прохода значениям  
    i++;  
    j--;  
}  
} while(i < j);
```



```
// повторять пока проходы не перекрестнутся  
// рекурсивное повторение алгоритма для разделенных  
// частей массива
```

```
if ( left < j ) QuickSort(a,left,j);  
if ( i < right) QuickSort(a,i,right);  
  
}
```

```
...  
QuickSort(b,0,n1);
```

Быстрая сортировка (Quicksort), в варианте с минимальными затратами памяти — сложность алгоритма:  $O(n \log n)$  — среднее время,  $O(n^2)$  — худший случай; широко известен как быстрееший из известных для упорядочения больших случайных списков; с разбиением исходного набора данных на две половины так, что любой элемент первой половины упорядочен относительно любого элемента второй половины; затем алгоритм применяется рекурсивно к каждой половине. |