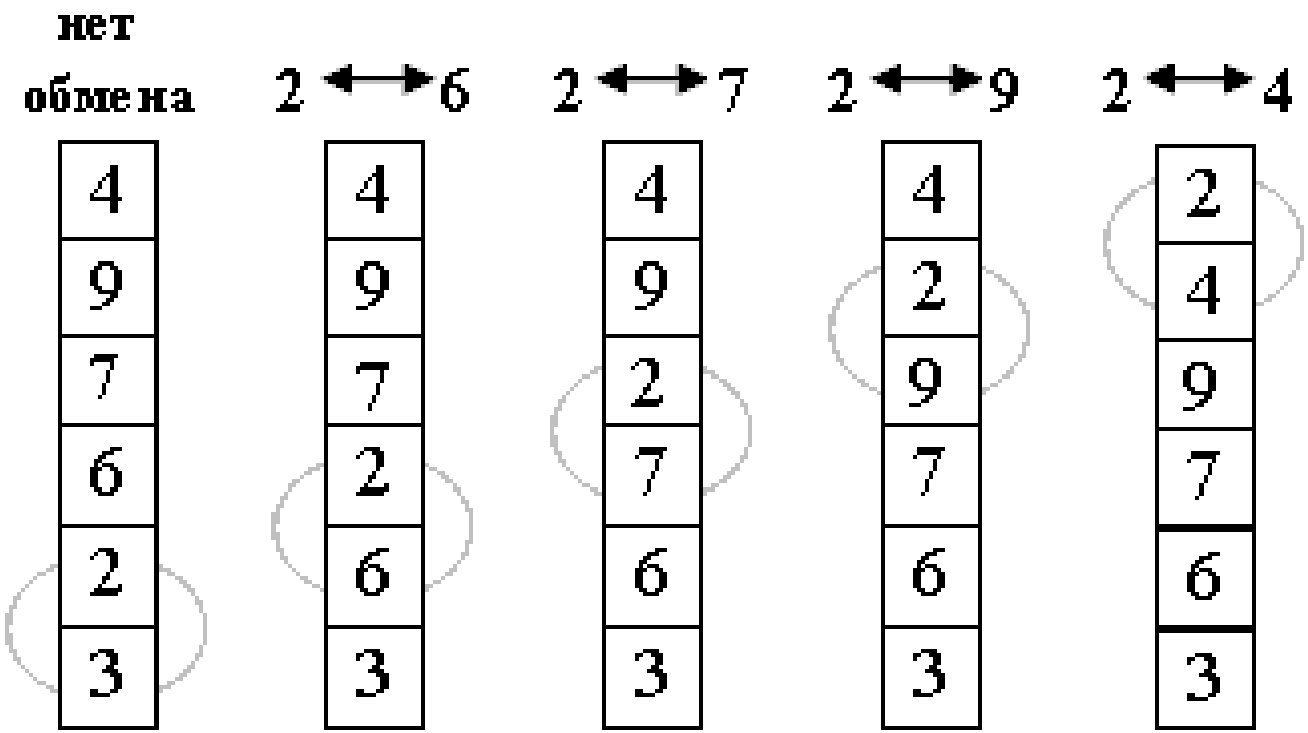
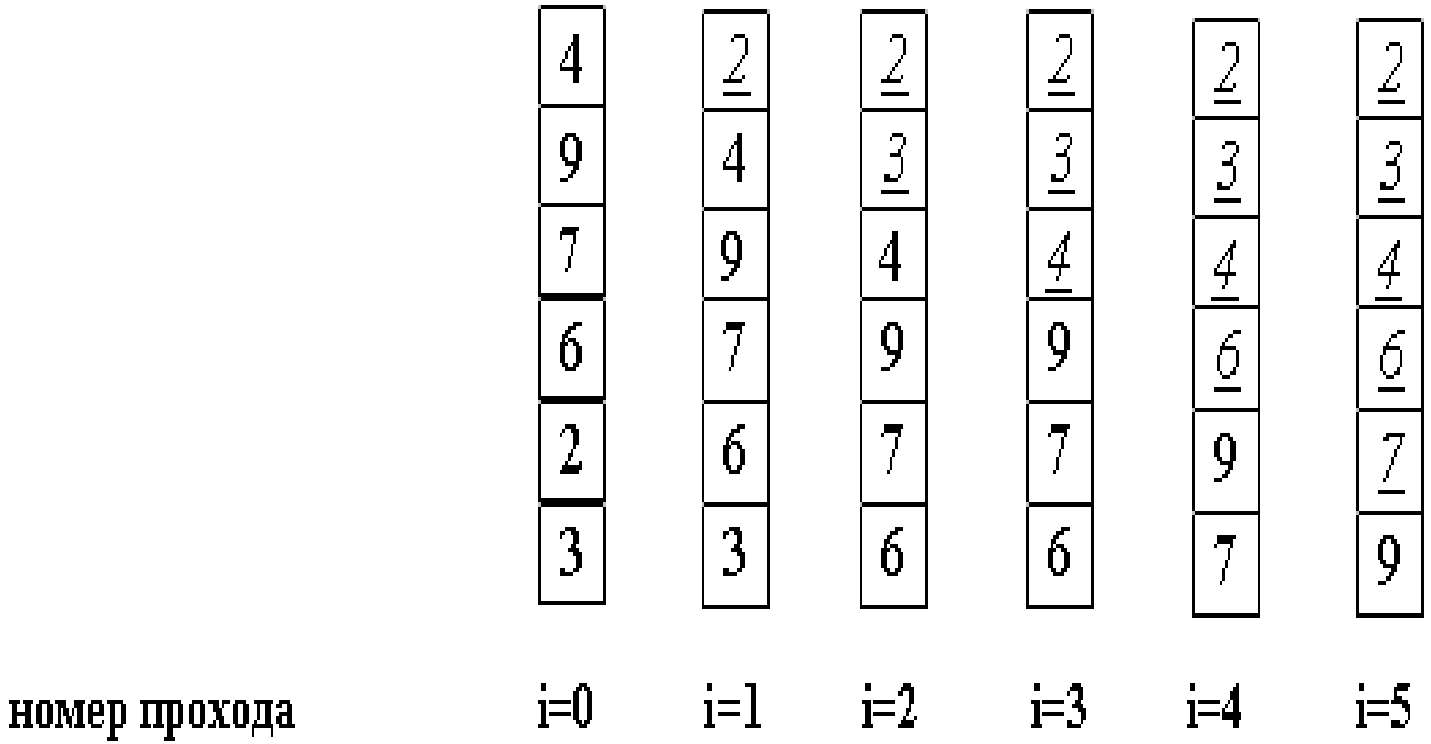
Обменные сортировки:BubbleSort

- Алгоритм прямого обмена основывается на сравнении и смене позиций пары соседних элементов.
- Процесс продолжается до тех пор пока не будут упорядочены все элементы

Обменные сортировки:BubbleSort



Обменные сортировки:BubbleSort

- 

Обменные сортировки:BubbleSort

Алгоритм BubbleSort(a, n);

```
for i ← 2 to n do
        Flag ← false;
        for j ← n downto i do
            if a[j-1]>a[j] then
                    Tmp ← a[j-1]; a[j-1] ← a[j];
                    a[j] ← Tmp; Flag ← true;
         if not Flag then return a;
```

Обменные сортировки:BubbleSort

- **Алгоритм имеет среднюю и максимальную временные сложности $O(n^2)$ (два вложенных цикла, зависящих от n линейно).**

- **Введение переменной Flag и прерывание работы в случае отсортированного массива позволяет свести минимальную временную сложность к $O(n)$.**

Обменные сортировки:BubbleSort

- **легкий пузырек снизу поднимется наверх за один проход, тяжелые пузырьки опускаются с минимальной скоростью:один шаг за итерацию.**
- **массив 2 3 4 5 6 1 будет отсортирован за 1 проход,**
- **а сортировка последовательности 6 1 2 3 4 5 потребует 5 проходов.**
- **Чтобы избежать подобного эффекта, можно менять направление следующих один за другим проходов.**
- **Данный алгоритм иногда называют "*шейкер-сортировкой*".**

Обменные сортировки:ShakerSort

- **Алгоритм ShakerSort (a,n);**

```
L ← 2; R ← n; k ← n;
repeat
  for j ← R  downto L   do {слева направо}
     if a[j-1]>a[j] then
        x ← a[j-1]; a[j-1] ← a[j];
        a[j] ← x; k ← j
   L ← k+1;
```

Обменные сортировки:ShakerSort

- **for j ← L to R do {справа налево}**
  if a[j-1] >a[j] then
  x ← a[j-1]; a[j-1] ← a[j];
  a[j] ← x; k ← j
  R ← k-1;

while L< R;

shaker

Обменные сортировки

- **число сравнений строго обменном алгоритме:**

$$(n^2-n)/2$$

- **среднее число перемещений:**

$$3\times(n^2-n)/2$$

Обменные сортировки

минимальное число сравнений

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

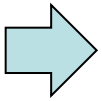
среднее число сравнений пропорционально

$$\frac{1}{2}(n^2 - n(k2 + \ln n))$$

сравнение

Обменные сортировки:QuickSort

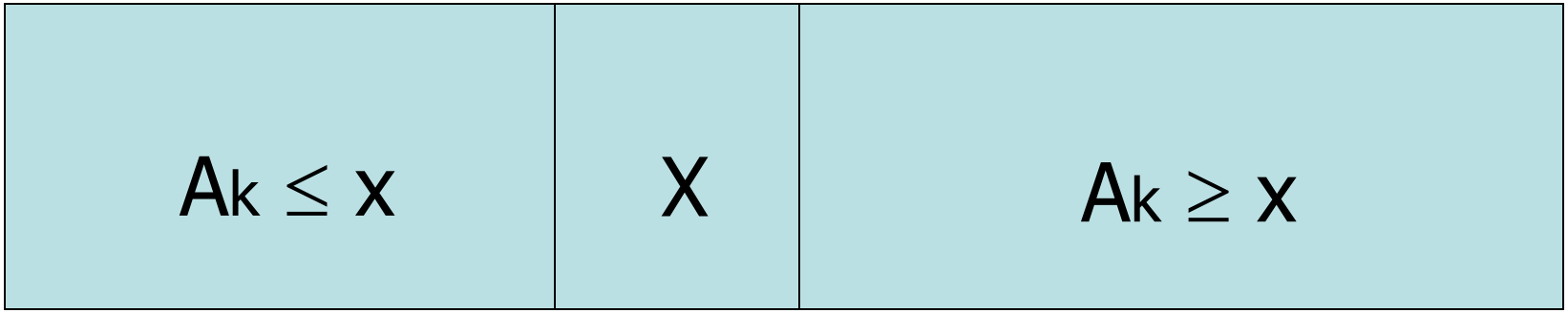
- **Выберем наугад какой-либо элемент массива**



x

- **Просматриваем массив слева направо , пока не обнаружим элемент Ai>x**
- **Просматриваем массив справа налево, пока не встретим Ai<x**
- **Меняем местами эти два элемента**
- **Процесс просмотра и обмена продолжается, пока оба просмотра не встретятся**

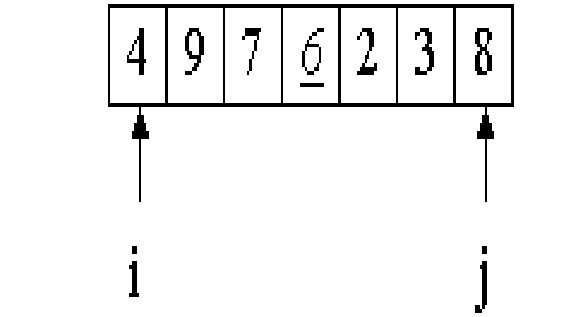
Обменные сортировки:QuickSort



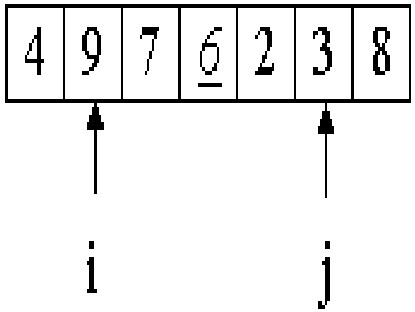
1<k<i: $A_k \le X$

i< k < n: $x \le A_k$

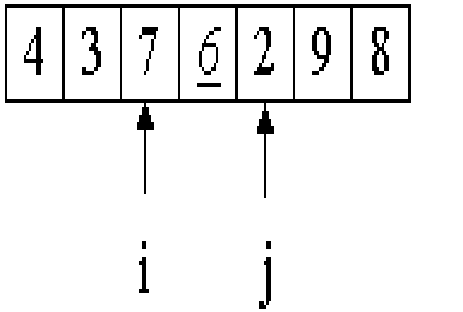
Обменные сортировки:QuickSort



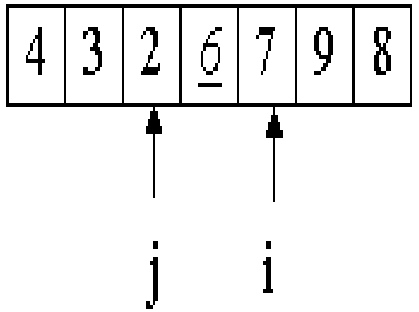
исходное положение указателей



положение первого обмена



положение второго обмена



конец процедуры

Обменные сортировки:QuickSort

- **Алгоритм partition (a,1,n,x);**

```
i ← 1; j ← n;
выбрать x;
repeat
   while a[i] < x do i ← i+1;
   while a[j] >x do j ← j-1;
   if i<=j then   w ← a[i]; a[i] ← a[j];
                  a[j] ← w; i++; j--

while i<j;
```

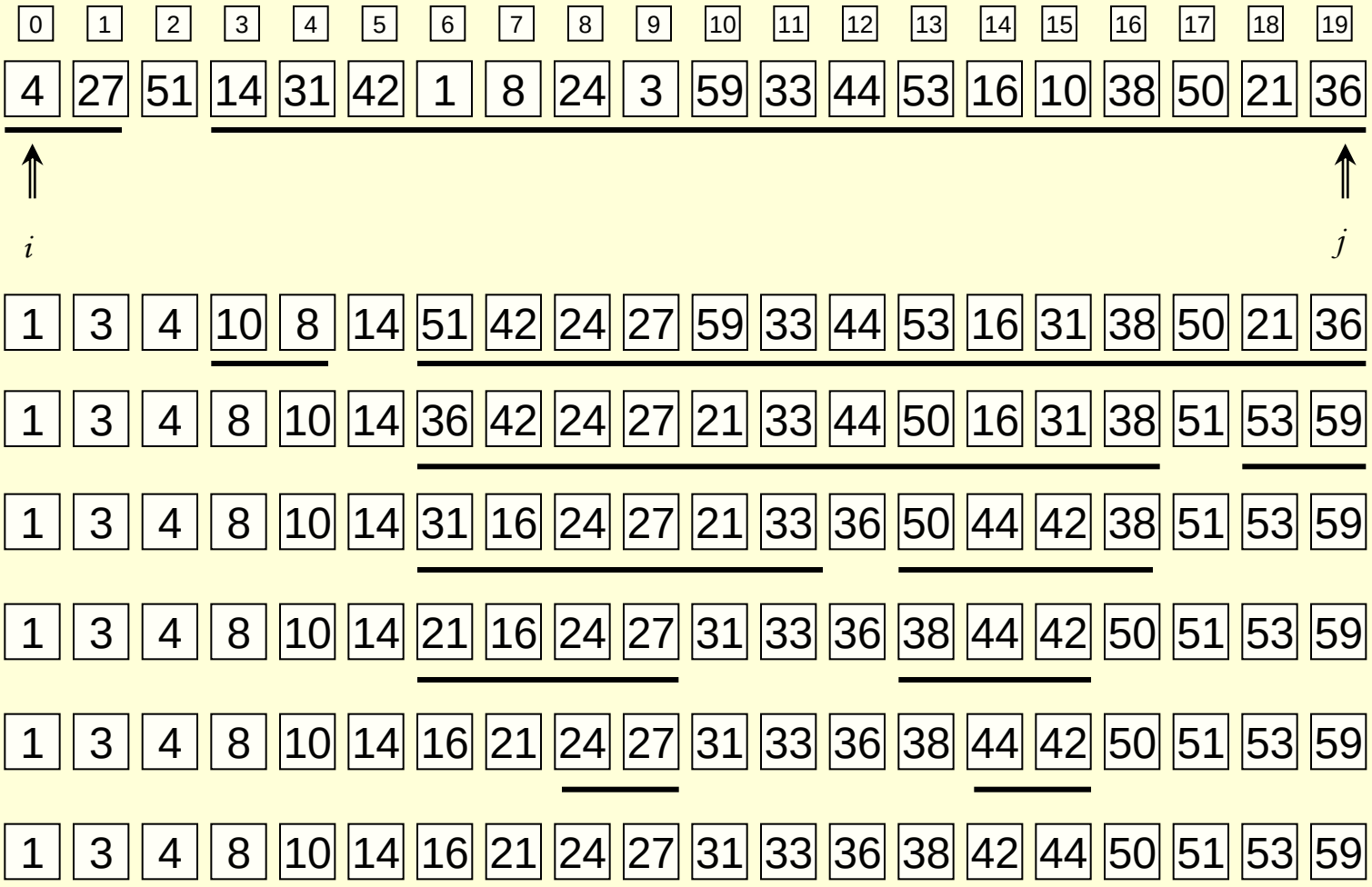
Обменные сортировки:QuickSort

- **Алгоритм Sort(L,R:index);**

```
i ←L; j ← R;
x ← a[(L+R) div 2];
partition (a,L,R,x);
 if j>L then Sort (L,j);
 if i<R then Sort (i,R);
```

Начало рекурсии - Sort (1,n)

Быстрая сортировка



Дополнения и улучшения алгоритма

- Первый элемент в сортируемом куске выбирается случайно и запоминается;
- Участки, меньшие определенного размера, сортируются простыми способами;
- Иногда исключение рекурсивных вызовов приводит к повышению эффективности.

Обменные сортировки:QuickSort

Ожидаемое число обменов:

(n-1)/6

Общее число сравнений:

n*logn

Наихудший случай – для сравнения выбирается наибольшее из всех значений в указанной области, те

левая часть состоит из n-1, а правая из 1, те производительность ~ n^2

Сортировка распределением

- **Пусть каждый элемент массива может принимать M (например, от 0 до M) фиксированных значений.**

- **Введем массив Amount[0..M], первоначально обнулив его.**

- **Затем для каждого i подсчитаем количество элементов массива A, равных i, и занесем это число в Amount[i]:**

Сортировка распределением

```
for i := 0 to M do
                Amount[i] := 0;
 for i := 1 to N do
          Inc(Amount[A[i]]);
```

Сортировка распределением

- Теперь в первые Amount[0] элементов массива A запишем 0,

  в следующие Amount[1] элементов массива A запишем 1и т.д.

  до тех пор, пока не дойдем до конца массива A и массива Amount):

Сортировка распределением

```
 p := 1;
for i := 0 to M do
    for j := 1 to Amount[i] do
       begin
          A[p] := i;
          Inc(p);
       end;
```

Сортировка распределением

- Временную сложность метода можно оценить как O(M+N)

  M появляется в сумме, так как изначально надо обнулить массив Amount, а это требует M действий).

- Пространственная сложность в этом случае равна O(M), поскольку требуется дополнительная память размером порядка M.

Сортировка слиянием

Divide et impera [дивидэ эт импэра]

Парадигма разработки алгоритмов, заключающаяся в рекурсивном разбиении решаемой задачи на две или более подзадачи того же типа, но меньшего размера, и комбинировании их решений для получения ответа к исходной задаче.

Разбиения выполняются до тех пор, пока все подзадачи не окажутся элементарными.

Сортировка слиянием

“Разделяй и властвуй”

1. **Разделение (декомпозиция)** задачи на несколько подзадач.
2. **Покорение** — рекурсивное решение этих подзадач. Когда объем подзадачи достаточно мал, выделенные подзадачи решаются непосредственно.
3. **Комбинирование** решения исходной задачи из решений вспомогательных задач.

Сортировка слиянием

Алгоритм сортировки слиянием (merge sort)

Разделение: сортируемая последовательность, состоящая из n элементов, разбивается на две меньшие последовательности, каждая из которых содержит n/2 элементов

Покорение: сортировка обеих вспомогательных последовательностей методом слияния

Комбинирование: слияние двух отсортированных последовательностей для получения окончательного результата

Сортировка слиянием

Процедура Merge (A,p, q,r),

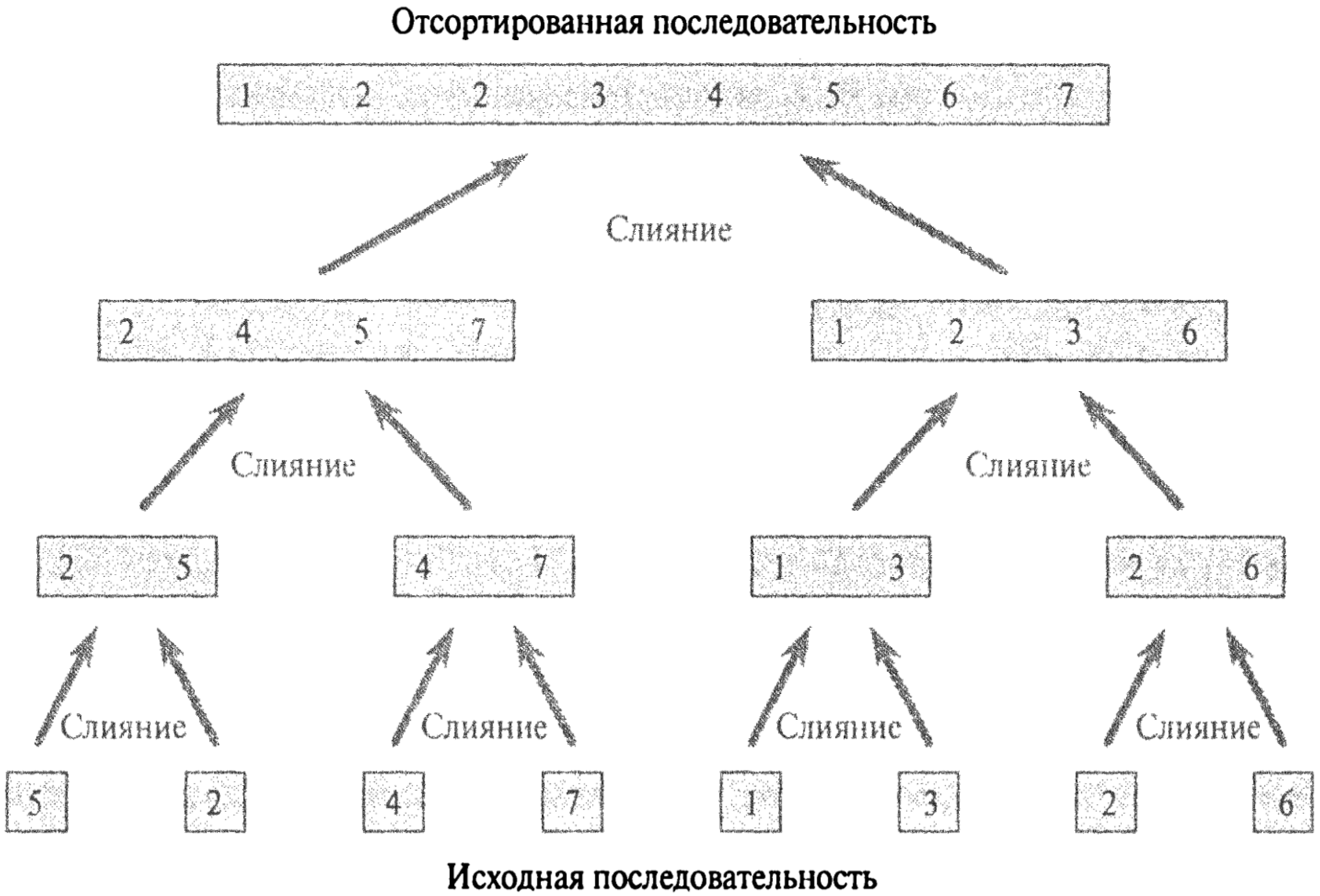
где A — исходный массив, а p, q и r — индексы, нумерующие элементы массива, такие, что $p \le q < r$.

В этой процедуре предполагается,

что элементы подмассивов A [p..q] и A[q+1..r] упорядочены.

Она сливает эти два подмассива в один отсортированный, элементы которого заменяют текущие элементы подмассива A [p..r].

Сортировка слиянием



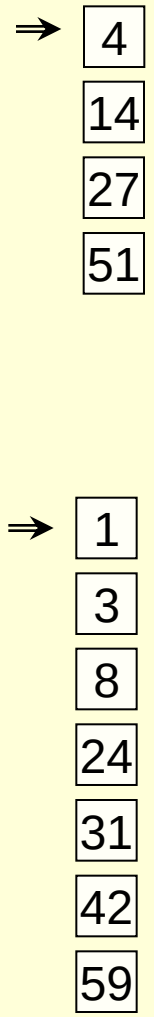
Сортировка слиянием

```
MERGE(A, p, q, r)
 1  n1 ← q − p + 1
 2  n2 ← r − q
 3  Создаем массивы L[1..n1 + 1] и R[1..n2 + 1]
 4  for i ← 1 to n1
 5       do L[i] ← A[p + i − 1]
 6  for j ← 1 to n2
 7       do R[j] ← A[q + j]
 8  L[n1 + 1] ← ∞
 9  R[n2 + 1] ← ∞
10  i ← 1
11  j ← 1
12  for k ← p to r
13       do if L[i] ⩽ R[j]
14             then A[k] ← L[i]
15                  i ← i + 1
16             else A[k] ← R[j]
17                  j ← j + 1
```

Сортировка слиянием

```
MERGE_SORT(A, p, r)
1   if p < r
2       then q ← ⌊(p + r)/2⌋
3            MERGE_SORT(A, p, q)
4            MERGE_SORT(A, q + 1, r)
5            MERGE(A, p, q, r)
```

Алгоритм слияния упорядоченных массивов




```
public static int[] merge(int[] a, int[] b) {
    int na = a.length,
        nb = b.length,
        nc;
    int[] c = new int[nc = na + nb];
    int ia = 0,
        ib = 0,
        ic = 0;
    while (ia < na && ib < nb) {
        if (a[ia] < b[ib])
            c[ic++] = a[ia++];
        else
            c[ic++] = b[ib++];
    }
    while (ia < na) c[ic++] = a[ia++];
    while (ib < nb) c[ic++] = b[ib++];
    return c;
}
```

Сортировка фон Неймана (слиянием)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
4	27	51	14	31	42	1	8	24	3	59	33	44	53	16	10	38	50	21	36

И так далее…

Алгоритм сортировки фон Неймана

```
public static void mergeSort(int[] data) {
    int n = data.length;          // Длина массива
    int[] altData = new int[n];   // Дополнительный массив
    int[] from = data,            // Указатели "откуда" и "куда" происходит слияние
          to = altData;
    int len = 1;                  // Длина сливаемого фрагмента
    do {
        int start = 0;
        while (start < n) {
            // Сливаем участки from[start:(start+len-1)] и from[(start+len):(start+2*len-1)]
            // в to[start:(start+2*len-1)]
            mergeSection(
                    from, start, Math.min(start+len, n),
                    from, Math.min(start+len, n), Math.min(start+(len<<1), n),
                    to, start);
            start += (len << 1);
        }
        // Меняем направление слияния
        int[] interm = from; from = to; to = interm;
    } while ((len <<= 1) < n);
    // Если после последнего слияния результат оказался "не там",
    // то переносим результат "куда надо"
    if (from != data) {
        mergeSection(from, 0, n, from, n, n, data, 0);
    }
}
```

Сортировка слиянием

- Пусть k - положительное целое число.
- Разобьем массив A[1]..A[n] на отрезки длины k.

  (Первый - A[1]..A[k], затем A[k+1]..A[2k] и т.д.)
- Последний отрезок будет неполным, если n не делится нацело на k.
- Назовем массив k-упорядоченным, если каждый из этих отрезков длины k упорядочен.

Сортировка слиянием

- любой массив 1-упорядочен, так как его подмассивы длиной 1 можно считать упорядоченными.

- если массив k-упорядочен и n<=k, то он упорядочен.

Сортировка слиянием

- Процедура преобразования k-упорядоченного массива в 2k-упорядоченный:

1. Сгруппировать все подмассивы длины k в пары подмассивов.
2. Пару упорядоченных подмассивов объединить в один упорядоченный подмассив.
3. Проделав это со всеми парами, мы получим 2k-упорядоченный массив:

Сортировка слиянием

```
{группировка   подмассивов длины K в пары}
t:=1;
while t + k < n do
  begin
    p := t; {индекс 1 эл-та 1-ого подмасс.}
   q := t+k; {индекс 1 эл-та 2-ого подмасс.}
   ...
   r := min(t+2*k, n);
   ...
слияние подмассивов A[p..q-1] и A[q..r-1]
   t := r;
end;
```

Сортировка слиянием

- Пусть p0 и q0 - номера последних элементов участков, подвергшихся слиянию,
- s0 - последний записанный в массив B элемент.
- На каждом шаге слияния производится одно из двух действий:

1. B[s0+1]:=A[p0+1];
   Inc(p0); Inc(s0);
   или
2. B[s0+1]:=A[q0+1];{
   Inc(q0); Inc(s0);

Сортировка слиянием

- Первое действие может производиться при двух условиях:

1. первый отрезок не кончился (p0 < q);
2. второй отрезок кончился (q0 = r) или не кончился, но элемент в нем не меньше (q0 < r) и (A[p0+1] <= A[q0+1])

Сортировка слиянием

- Окончательный текст

```
k := 1;
while k < N do
  begin t := 1;
   while t+k < N do
    begin p := t; q := t+k;
     if t+2*k>N
     then r := N
     else r := t+2*k;
     p0 := p; q0 := q; s0 := p;
```

Сортировка слиянием

```
while (p0<>q) or (q0<>r) do
 begin
  if (p0<q) and ((q0=r)or((q0<r)and
                  (A[p0+1]<=A[q0+1])))
  then begin
         B[s0+1] := A[p0+1];
         Inc(p0);
       end
  else begin B[s0+1] := A[q0+1];
              Inc(q0);
       end;
  Inc(s0);
end;
```

Сортировка слиянием

```
    t := r;
end {цикла t+k};
k := k *2;
A := B
End {цикла k};
```

example

Сортировка слиянием

- **Временная сложность**

O(N*log(N))

(так как преобразование k-упорядоченного массива в 2k-упорядоченный требует порядка N действий и внешний цикл по k совершает порядка log(N) итераций).

сравнение