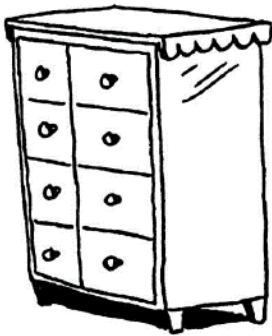
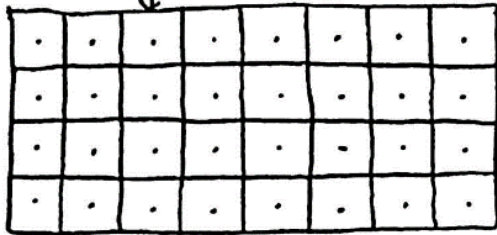


Массивы та зв'язані списки



АДРЕС: fe0fffeeb



Неупорядоченный



Упорядоченный

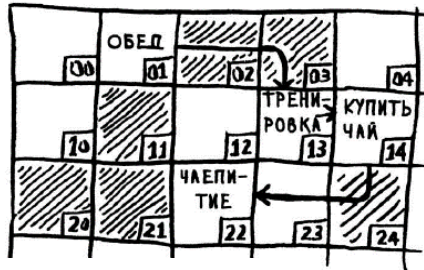
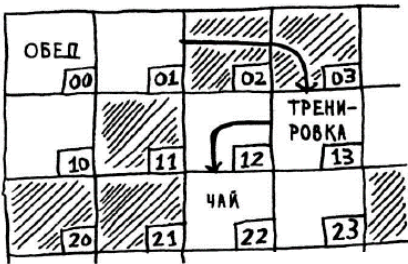
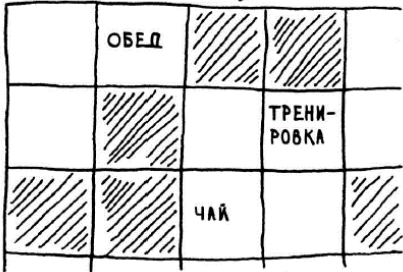
СПИСОК ДЕЛ  
ЭТУ ПАМЯТЬ ИСПОЛЗУЮТ Д



СВОБОДНАЯ ПАМЯТЬ

ЭТУ ПАМЯТЬ ИСПОЛЗУЮТ ДРУГИЕ

СВОБОДНАЯ ПАМЯТЬ



ЭТУ ЗАДАЧУ НЕОБХОДИМО  
ВСТАВИТЬ СЮДА



ПОЭТОМУ ЭТУ ЗАДАЧУ  
ПРИДЕТСЯ СДВИНУТЬ  
ВНИЗ



10 20 30 40

0 1 2 3

Связанные ади  
памяти

МАССИВ ИЗ ПЯТИ ЭЛЕМЕНТОВ



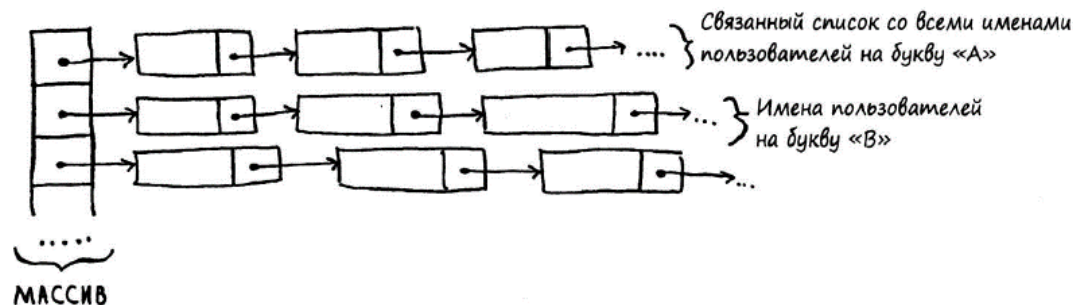
00 01 02 03 04

ПЯТЫЙ  
ЭЛЕМЕНТ

	МАССИВЫ	СПИСКИ
ЧТЕНИЕ	$O(1)$	$O(n)$
ВСТАВКА	$O(n)$	$O(1)$

$O(n)$  = ЛИНЕЙНОЕ ВРЕМЯ  
 $O(1)$  = ПОСТОЯННОЕ ВРЕМЯ

	МАССИВЫ	СПИСКИ
ЧТЕНИЕ	$O(1)$	$O(n)$
ВСТАВКА	$O(n)$	$O(1)$
УДАЛЕНИЕ	$O(n)$	$O(1)$



## Сортировка выбором

~🎵~	СЧЕТЧИК ВОСПРОИЗВЕДЕНИЙ
RADIOHEAD	156
KISHORE KUMAR	141
THE BLACK KEYS	35
NEUTRAL MILK HOTEL	94
BECK	88
THE STROKES	61
WILCO	111

~🎵~	СЧЕТЧИК ВОСПРОИЗВЕДЕНИЙ
RADIOHEAD	156
KISHORE KUMAR	141
THE BLACK KEYS	35
NEUTRAL MILK HOTEL	94
BECK	88
THE STROKES	61
WILCO	111

1. У RADIOHEAD БОЛЬШЕ ВСЕГО ВОСПРОИЗВЕДЕНИЙ...



«СПИСОК»	СЧЕТЧИК ВОСПРОИЗВЕДЕНИЙ
RADIOHEAD	156

2. ДОБАВЛЯЕМ RADIOHEAD В НОВЫЙ СПИСОК

~🎵~	СЧЕТЧИК ВОСПРОИЗВЕДЕНИЙ
KISHORE KUMAR	141
THE BLACK KEYS	35
NEUTRAL MILK HOTEL	94
BECK	88
THE STROKES	61
WILCO	111

1. СЛЕДУЮЩИЙ ИСПОЛНИТЕЛЬ ПО КОЛИЧЕСТВУ ВОСПРОИЗВЕДЕНИЙ — KISHORE KUMAR



«СПИСОК»	СЧЕТЧИК ВОСПРОИЗВЕДЕНИЙ
RADIOHEAD	156
KISHORE KUMAR	141

2. ПОЭТОМУ ОН СЛЕДУЮЩИМ ДОБАВЛЯЕТСЯ В НОВЫЙ СПИСОК

1. RADIOHEAD
2. KISHORE KUMAR
3. THE BLACK KEYS
4. NEUTRAL MILK HOTEL
5. BECK
6. THE STROKES
7. WILCO

n элементов

Сортировка выбором  $O(n \times n) \Rightarrow O(n^2)$

Быстрая сортировка

$O(n \log n)$

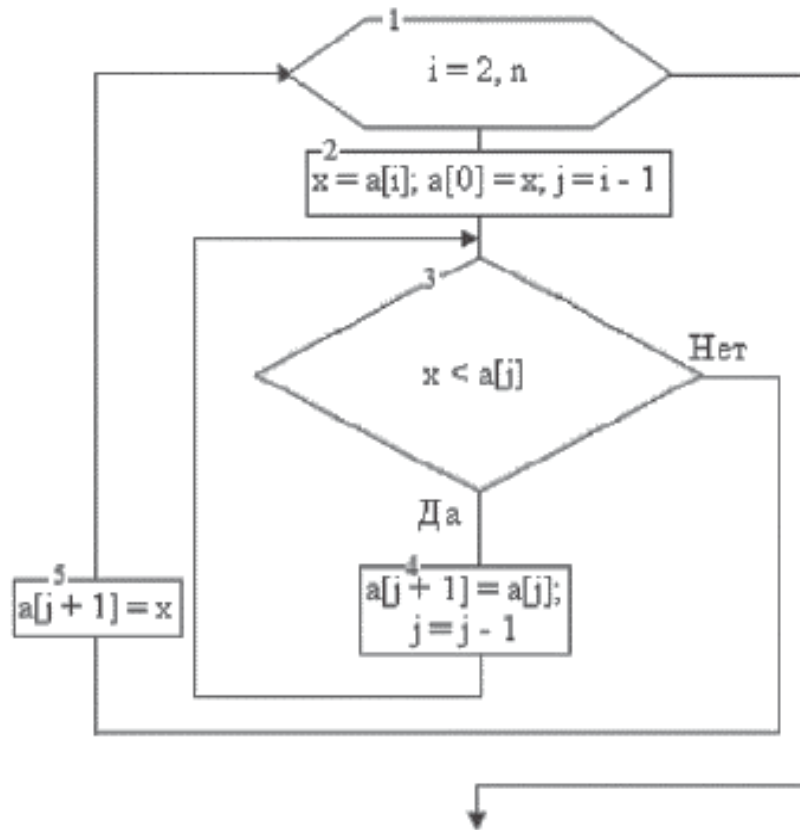
## Алгоритмы сортировки

1. Сортировка включениями (вставкой). 2. Сортировка выбором. 3. Сортировка обменом.

### Сортировка простой вставкой

Пусть имеется массив  $a[1], a[2], \dots, a[n]$ . Пусть элементы  $a[1], a[2], \dots, a[i - 1]$  уже отсортированы, и пусть имеем входную последовательность  $a[i], a[i + 1], \dots, a[n]$ . На каждом шаге, начиная с  $i = 2$  и увеличивая  $i$  на единицу, берем  $i$ -й элемент входной последовательности и вставляем его на подходящее место в уже отсортированную часть последовательности. Обозначим вставляемый элемент через  $x$ . Пусть начальный массив 32 64 9 30 87 14 2 76.

$i = 2 \quad x = 64$	Ищем для $x$ подходящее место, считая, что $a[1] = 32$ — это уже отсортированная часть последовательности. Получаем 32 64 9 30 87 14 2 76.
$i = 3 \quad x = 9$	Часть последовательности $a[1], a[2]$ уже отсортирована. Ищем для $x$ подходящее место: 9 32 64 30 87 14 2 76.
$i = 4 \quad x = 30.$	Ищем для $x$ подходящее место: 9 <b>30</b> 32 64 87 14 2 76
$i = 5 \quad x = 87$	Ищем для $x$ подходящее место: 9 30 32 64 <b>87</b> 14 2 76.
$i = 6 \quad x = 14$	Ищем для $x$ подходящее место: 9 <b>14</b> 30 32 64 87 2 76
$i = 7 \quad x = 2$	Ищем для $x$ подходящее место: 2 9 14 30 32 64 87 76
$i = 8 \quad x = 76$	Ищем для $x$ подходящее место: 2 9 14 30 32 64 76 87



Блок 1. Начинаем цикл для перебора всех элементов исходного массива.

Блок 2. Присваиваем  $x$  значение очередного элемента массива. Устанавливаем барьер (фиктивный элемент)  $a[0] = x$  и продвигаемся назад по отсортированной части массива.

Блок 3. Начинаем цикл для поиска подходящего места для значения  $x$ .

Блок 4. Пока  $x$  меньше очередного элемента (выход «Да» блока 3), выполняем сдвиг и продвигаемся к началу отсортированной части массива.

Блок 5. По выходу «Нет» блока 3 вставляем элемент  $x$  на нужное место.

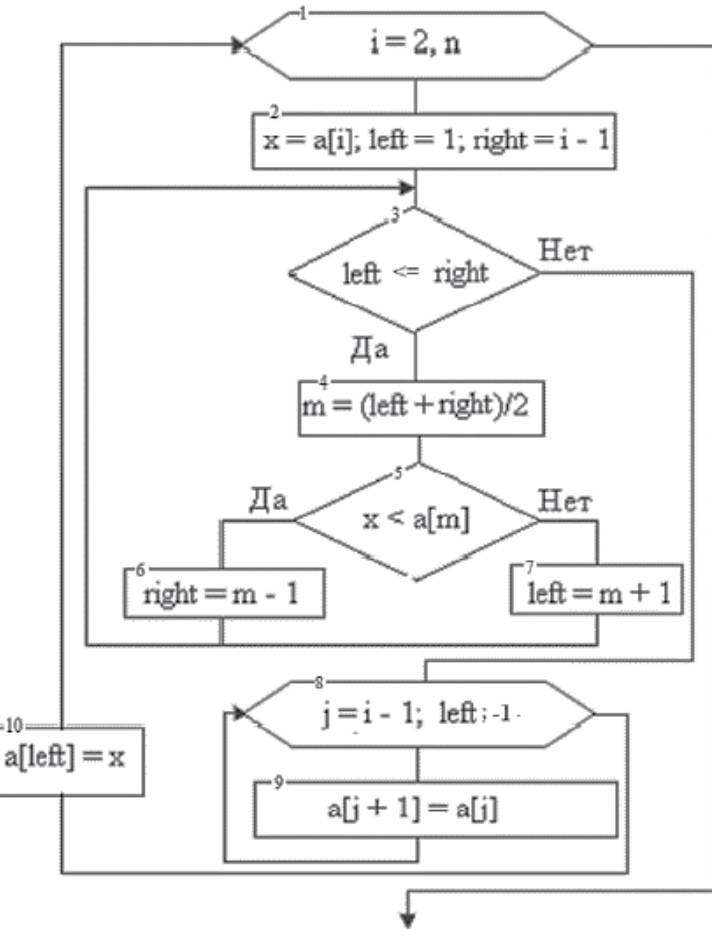
32 64 9 30 87 14 2 76

$i$	$x$	$a[0]$	$j$	$x < a[j]?$	$a[j + 1]$	Массив
2	64	64	1	64 < 32? «Нет»	$a[2] = x = 64$	32 64 9 30 87 14 2 76
3	9	9	2	9 < 64? «Да»	$a[3] = a[2] = 64$	32 64 64 30 87 14 2 76
			1	9 < 32? «Да»	$a[2] = a[1] = 32$	32 32 64 30 87 14 2 76
			0	9 < 9? «Нет»	$a[1] = x = 9$	9 32 64 30 87 14 2 76
4	30	30	3	30 < 64? «Да»	$a[4] = a[3] = 64$	9 32 64 64 87 14 2 76
			2	30 < 32? «Да»	$a[3] = a[2] = 32$	9 32 32 64 87 14 2 76
			1	30 < 9? «Нет»		$a[2] = x = 30$

9 30 32 64 87 14 2 76



Для поиска места для вставки применяют бинарный поиск. Этот метод на каждом шаге сравнивает  $x$  со средним (по положению) элементом отсортированной последовательности до тех пор, пока не будет найдено место включения. Обозначим  $left$  — левая граница отсортированного массива,  $right$  — его правая граница.



## Сортировка бинарными включениями

- Блок 1. Начинаем цикл для перебора всех элементов исходного массива.
- Блок 2. Присваиваем  $x$  значение очередного элемента массива. Устанавливаем левую и правую границы отсортированной части массива.
- Блок 3. Пока левая граница отсортированного массива не превосходит правую, выполняется тело цикла, состоящее из блоков 4–7.
- Блок 4. Находится средний по положению элемент отсортированной части массива.
- Блок 5. Значение  $x$  сравнивается с найденным элементом. По выходу «Да» блока 5 корректируется правая граница отсортированной части массива, по выходу «Нет» — левая. При выходе из цикла с предусловием будет найдено положение вставляемого элемента.
- Блоки 8–9 реализуют сдвиг элементов массива для вставки значения  $x$ .
- Блок 10. Вставка значения в отсортированную часть массива.
- Приведем выполнение алгоритма при сортировке массива 32 64 9 30 87 14 2 76

$i$	$x$	$left$	$right$	$left \leq right?$	$m$	$x < a[m]?$	$j$	$a[j + 1]$	$a[left]$
2	64	1	1	«Да»	1	32 < 64? «Да»			
			0	«Нет»			1	$a[2] = 64$	$a[1] = 32$
3	9	1	2	«Да»	1	9 < 64? «Да»			
			1	«Нет»			2	$a[3] = a[2] = 64$	
							1	$a[2] = a[1] = 32$	
									$a[1] = 9$
4	30	1	3	«Да»	2	30 < 32? «Да»			
			2	«Да»	1	30 < 9? «Нет»			
		2		«Нет»			3	$a[4] = a[3] = 64$	
							2	$a[3] = a[2] = 32$	$a[2] = 30$

9 30 32 64 87 14 2 76

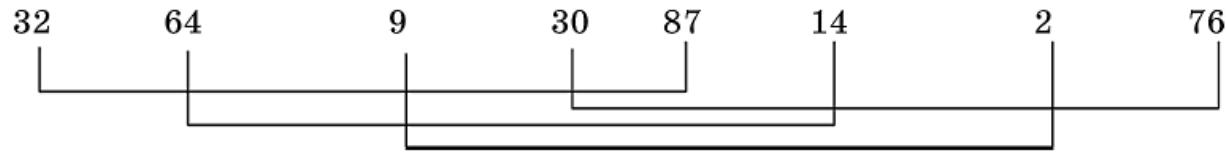
## Сортировка Шелла

Рассмотрим сортировку Шелла на массиве, состоящем из 8 элементов. На первом проходе отдельно группируются и сортируются все элементы, отстоящие друг от друга на четыре позиции. Этот процесс называется 4-сортировкой. В нашем примере из восьми элементов каждая группа содержит ровно два элемента. После этого элементы вновь объединяются в группы с элементами, отстоящими друг от друга на две позиции, и сортируются заново. Этот процесс называется 2-сортировкой. Наконец, на третьем проходе все элементы сортируются обычной сортировкой, или 1-сортировкой. Начальное состояние.

32 64 9 30 87 14 2 76

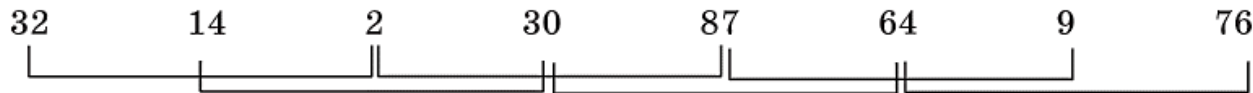
$n = 3.$

$a[1]$  и  $a[5]$ ,  $a[2]$  и  $a[6]$ ,  $a[3]$  и  $a[7]$ ,  $a[4]$  и  $a[8]$



32 64 9 30 87 14 2 76    32 14 2 30 87 64 9 76

$a[1]$  и  $a[3]$ ,  $a[2]$  и  $a[4]$ ,  $a[3]$  и  $a[5]$ ,  $a[4]$  и  $a[6]$ ,  $a[5]$  и  $a[7]$ ,  $a[6]$  и  $a[8]$



2 14 9 30 32 64 87 76



2 9 14 30 32 64 76 87

## Сортировка простым выбором

*В неупорядоченном массиве выбирается и отделяется от остальных элементов наименьший элемент. Наименьший элемент записывается на  $i$ -е место исходного массива, а элемент с  $i$ -го места — на место выбранного. Уже упорядоченные элементы (а они будут расположены начиная с первого места) исключаются из дальнейшей сортировки, поэтому длина оставшегося неупорядоченного массива должна быть на один элемент меньше предыдущего.*

32 64 9 30 87 14 2 76

$i = 1$ , наименьшее значение 2, меняем местами 2 и 32.

2 64 9 30 87 14 32 76

$i = 2$ , наименьшее значение в оставшейся части массива 9, меняем местами 9 и 64.

2 9 64 30 87 14 32 76

$i = 3$ , наименьшее значение в оставшейся части массива 14, меняем местами 14 и 64.

2 9 14 30 87 64 32 76

$i = 4$ , наименьшее значение в оставшейся части массива 30. Обмен не происходит.

2 9 14 30 32 64 87 76

$i = 5$ , наименьшее значение в оставшейся части массива 32, меняем местами 32 и 87.

2 9 14 30 32 64 87 76

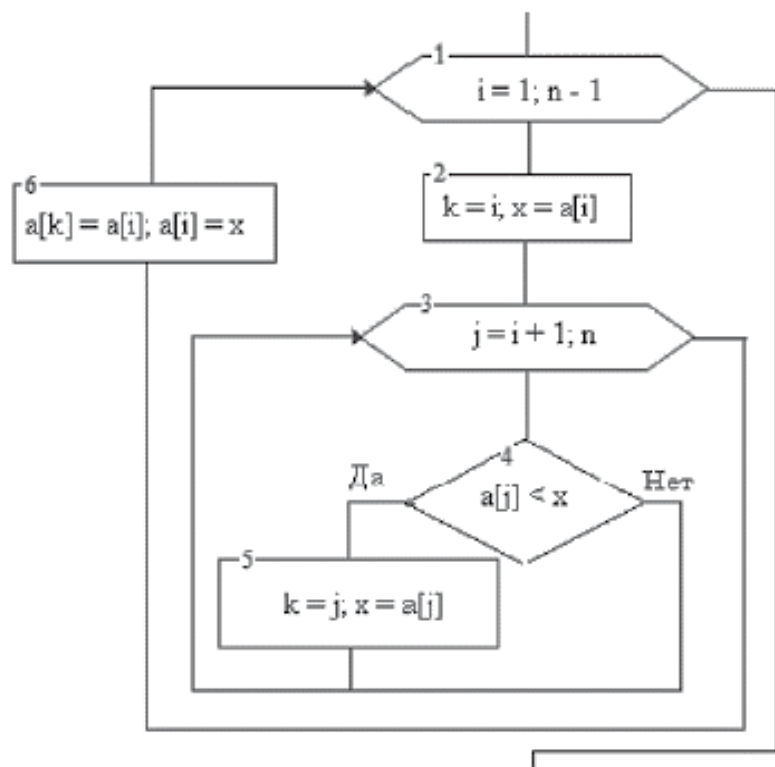
$i = 6$ , наименьшее значение в оставшейся части массива 64. Обмен не происходит.

2 9 14 30 32 64 87 76

$i = 7$ , наименьшее значение в оставшейся части массива 76, меняем местами 76 и 87.

2 9 14 30 32 64 76 87

$i = 8$ , наименьшее значение в оставшейся части массива 87. Обмен не происходит, так как 87 стоит на своем месте.



Блок 1. Организует внешний цикл для просмотра элементов неупорядоченной части массива.

Блок 2. Запоминание индекса и значения  $i$ -го элемента, которые принимаем за начальные при поиске наименьшего элемента.

Блоки 3–5. Поиск значения наименьшего элемента и его номера в еще неупорядоченной части массива.

Блок 6. Обмен наименьшего и  $i$ -го элементов.

**32 64 9 30 87 14 2 76;  $n = 8$**

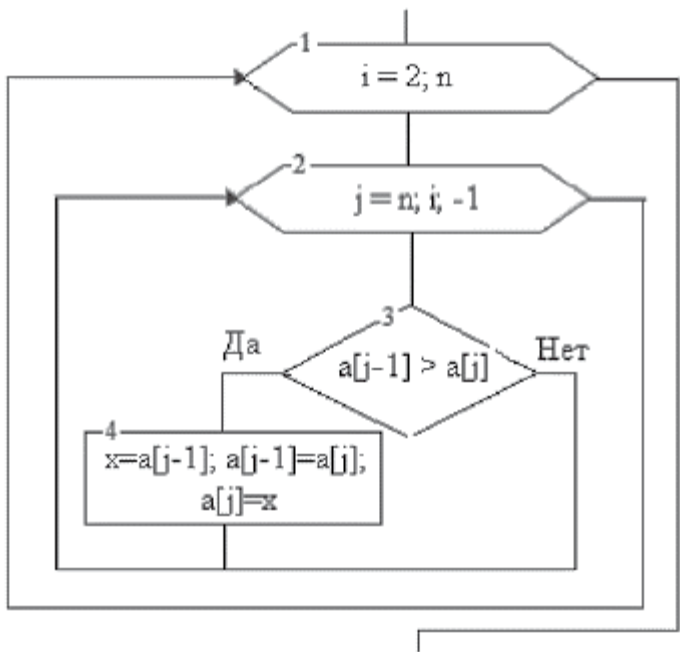
$i$	$k$	$x$	$j$	$a[j] < x?$	$a[k] = a[i]; a[i] = x$
1	1	32	2	$a[2] < 32?$ «Нет»	
	3	9	3	$a[3] < 32?$ «Да»	
			4	$a[4] < 9?$ «Нет»	
			5	$a[5] < 9?$ «Нет»	
			6	$a[6] < 9?$ «Нет»	
	7	2	7	$a[7] < 9?$ «Да»	$A[7] = a[1] = 32; a[1] = 2$
			кц		

**2 64 9 30 87 14 32 76**



# Сортировка простым обменом

Начиная с конца массива, сравниваем два соседних элемента ( $a[n]$  и  $a[n - 1]$ ). Если выполняется условие  $a[n - 1] > a[n]$ , то значения элементов меняются местами. Процесс продолжается для  $a[n - 1]$  и  $a[n - 2]$  и т. д., пока не будет произведено сравнение  $a[2]$  и  $a[1]$ . Понятно, что после этого на месте  $a[1]$  окажется элемент массива с наименьшим значением. На втором шаге процесс повторяется, но последними сравниваются  $a[3]$  и  $a[2]$ . На последнем шаге будут сравниваться только значения  $a[n]$  и  $a[n - 1]$ .



12 18 42 44 55 67 94 6 – 1 проход

94 6 12 18 43 44 55 76 – 7 проходов

Блок 1. Арифметический цикл. Значение параметра цикла  $i$  определяет количество сравниваемых элементов во внутреннем цикле.

Блок 2. Задание номеров элементов для сравнения.

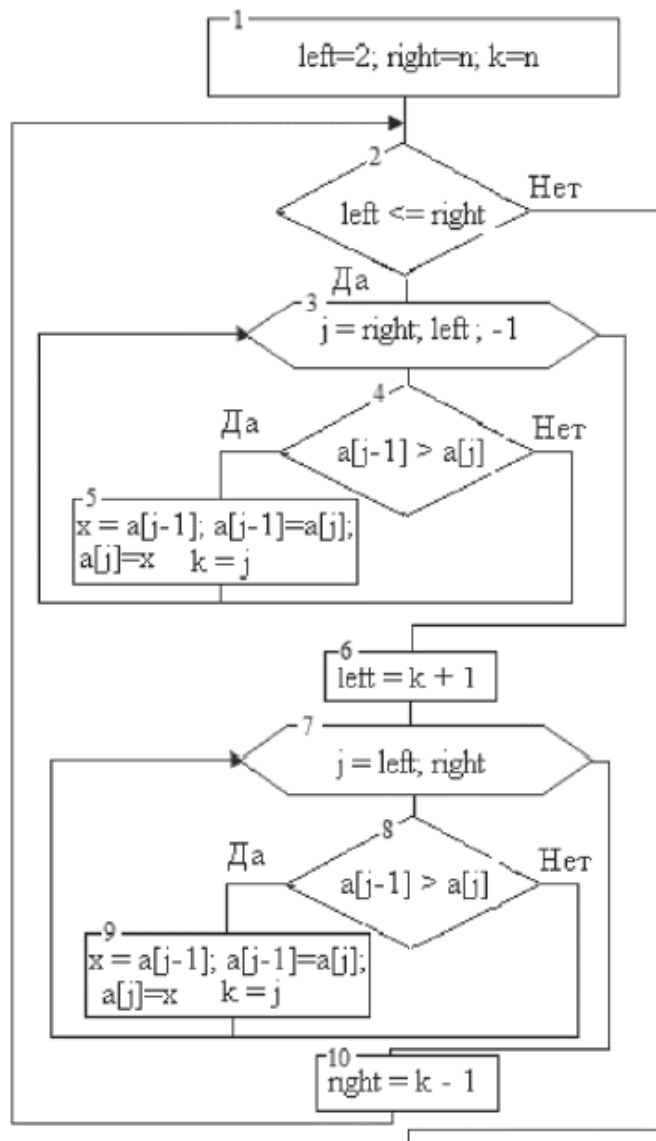
Блок 3. Сравнение двух соседних элементов.

Блок 4. Выполняется обмен элементов массива при выходе «Да» блока 3.

32 64 9 30 87 14 2 76

$i$	$j$	$a[j - 1] > a[j]$ ?	$a[j - 1], a[j]$	Массив
2	8	$2 > 76$ ? «Нет»		32 64 9 30 87 14 2 76
	7	$14 > 2$ ? «Да»	$a[6] = 2, a[7] = 14$	32 64 9 30 87 2 14 76
	6	$87 > 2$ ? «Да»	$a[5] = 2, a[6] = 87$	32 64 9 30 2 87 14 76
	5	$30 > 2$ ? «Да»	$a[4] = 2, a[5] = 30$	32 64 9 2 30 87 14 76
	4	$9 > 2$ ? «Да»	$a[3] = 9, a[4] = 2$	32 64 2 9 30 87 14 76
	3	$64 > 2$ ? «Да»	$a[2] = 2, a[3] = 64$	32 2 64 9 30 87 14 76
	2	$32 > 2$ ? «Да»	$a[1] = 2, a[2] = 32$	2 32 64 9 30 87 14 76

## Шейкер-сортировка



**На каждом следующем шаге меняется направление последовательного просмотра. В результате на одном шаге «всплывает» очередной наиболее легкий элемент, а на другом «тонет» очередной самый тяжелый. Обозначим  $left$  — номер левой границы сортируемой части массива,  $right$  — номер его правой границы**

Блок 1. Установка номеров начальных границ сортируемого массива.

Блок 2. Вход в цикл. Пока левая граница не превосходит правую границу (выход «Да» блока 2) выполняем цикл.

Блок 3. Выполнение прохода массива вниз.

Блок 4. Сравнение соседних элементов.

Блок 5. Если  $a[j - 1] > a[j]$  (выход «Да» блока 4), производим обмен этих элементов и фиксируем номер элемента  $j$ , с которым производился обмен. По окончании прохода вниз (выход блока 3) сдвигаем левую границу массива (блок 6) и выполняем проход вверх (блоки 7–9).

Выполняем проход сверху вниз

<i>left</i>	<i>right</i>	<i>k</i>	<i>left</i> ≤ <i>right</i> ?	<i>j</i>	<i>j</i> > <i>left</i> ?	<i>a</i> [ <i>j</i> - 1] > <i>a</i> [ <i>j</i> ]?	Обмен
2	7	7	«Да»	7	«Да»	2 > 87 «Нет»	
		6		6	«Да»	14 > 2 «Да»	$x = 14; a[5] = 2; a[6] = 14$
		5		5	«Да»	64 > 2 «Да»	$x = 64; a[4] = 2; a[5] = 64$
		4		4	«Да»	30 > 2 «Да»	$x = 30; a[3] = 2; a[4] = 30$
		3		3	«Да»	9 > 2 «Да»	$x = 9; a[2] = 2; a[3] = 9$
		2		2	«Да»	32 > 2 «Да»	$x = 32; a[1] = 2; a[2] = 32$
				1	«Нет»		

**32 9 30 64 14 2 87**

**2 32 9 30 64 14 87**

Выполняем проход снизу вверх

<i>left</i>	<i>right</i>	<i>k</i>	<i>j</i>	<i>j</i> ≤ <i>right</i>	<i>a</i> [ <i>j</i> - 1] > <i>a</i> [ <i>j</i> ]?	
3	7	3	3	«Да»	32 > 9? «Да»	$x = 32; a[2] = 9; a[3] = 32$
		4	4	«Да»	32 > 30? «Да»	$x = 32; a[3] = 30; a[4] = 32$
			5	«Да»	32 > 64? «Нет»	
		6	6	«Да»	64 > 14? «Да»	$x = 64; a[5] = 14; a[6] = 64$
			7	«Да»	64 > 87? «Нет»	
			8	«Нет»		

**2 9 30 32 14 64 87**

## QuickSort



```

1  #include <iostream>
2
3  using namespace std;
4
5  void quick_sort(int[],int,int);
6  int partition(int[],int,int);
7
8  int main()
9  {
10     int a[50],n,i;
11     cout<<"How many elements?";
12     cin>>n;
13     cout<<"\nEnter array
        elements:";
14
15     for(i=0;i<n;i++)
16     |   cin>>a[i];
17
18     quick_sort(a,0,n-1);
19     cout<<"\nArray after sorting:";
20
21     for(i=0;i<n;i++)
22     |   cout<<a[i]<<" ";
23
24     return 0;
25 }

```

```

27 void quick_sort(int a[],int l,int
    u)
28 {
29     int j;
30     if(l<u)
31     {
32         j=partition(a,l,u);
33         quick_sort(a,l,j-1);
34         quick_sort(a,j+1,u);
35     }
36 }

```

```

How many elements?10
Enter array elements:56
76
7
99
65
75675
4
53
7
8

Array after sorting:4 7 7 8 53 56 65 76
99 75675

```

```

37
38 int partition(int a[],int l,int u)
39 {
40     int v,i,j,temp;
41     v=a[l];
42     i=l;
43     j=u+1;
44
45     do
46     {
47         do
48         |   i++;
49         while(a[i]<v&& i<=u);
50
51         do
52         |   j--;
53         while(v<a[j]);
54
55         if(i<j)
56         {
57             temp=a[i];
58             a[i]=a[j];
59             a[j]=temp;
60         }
61     }while(i<j);
62
63     a[l]=a[j];
64     a[j]=v;
65
66     return(j);
67 }
68

```



Веселкова візуалізація алгоритмів сортування

Сервіси для візуалізації алгоритмів