

# Сортировка. Алгоритмы сортировки.

---

# Задача сортировки (sorting problem)

---

	Chen	3	A	991-878-4944	308 Blair
	Rohde	2	A	232-343-5555	343 Forbes
	Gazsi	4	B	766-093-9873	101 Brown
item →	<b>Furia</b>	<b>1</b>	<b>A</b>	<b>766-093-9873</b>	<b>101 Brown</b>
	Kanaga	3	B	898-122-9643	22 Brown
	Andrews	3	A	664-480-0023	097 Little
key →	<b>Battle</b>	<b>4</b>	<b>C</b>	<b>874-088-1212</b>	<b>121 Whitman</b>

Andrews	3	A	664-480-0023	097 Little
Battle	4	C	874-088-1212	121 Whitman
Chen	3	A	991-878-4944	308 Blair
Furia	1	A	766-093-9873	101 Brown
Gazsi	4	B	766-093-9873	101 Brown
Kanaga	3	B	898-122-9643	22 Brown
Rohde	2	A	232-343-5555	343 Forbes

# Сортировка выбором. Selection Sort.

## Идея алгоритма

- На каждой итерации  $i$ , найти индекс ( $min$ ) минимального значения
- поменять местами элементы  $a[i]$  и  $a[min]$  - `swap (a[i], a[min])`

7	10	5	3	8	4	2	9	6
$i$						$min$		

2	10	5	3	8	4	7	9	6
$i$		$min$						

2	3	5	10	8	4	7	9	6
	$i$				$min$			

2	3	4	10	8	5	7	9	6
		$i$			$min$			

2	3	4	5	8	10	7	9	6
			$i$		$min$			

(~~min~~ Example5, Project SelectionSort)

# Сортировка выбором. Реализация.

1. перемещаемся по массиву вправо  
 $i++$ ;

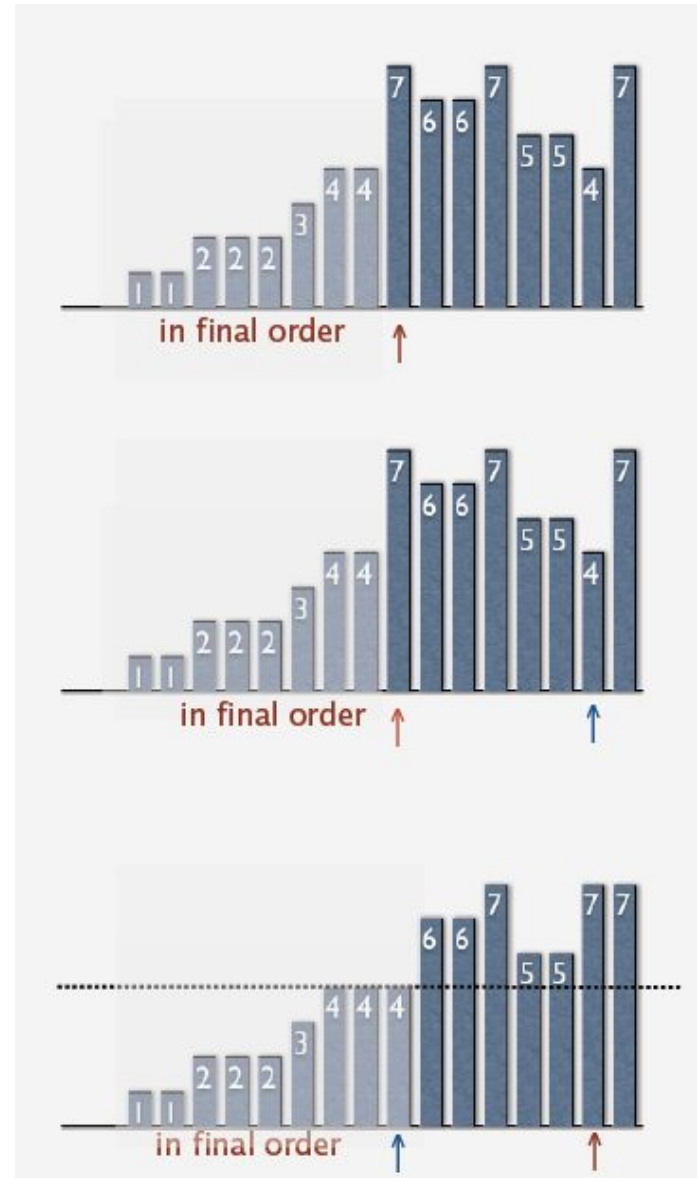
2. находим индекс минимального в  
правой части массива

```
int min=i;
for (int j=i+1; j<n; j++)
    if ( less(a[j], a[min]) )
        min=j;
```

3. меняем местами элементы  $i$ -ый  
и  $min$

```
swap(a, i, min);
```

(см. Example5, Project SelectionSort)



# Сортировка выбором. Анализ.

---

**Сравнений:**  $(N-1)+(N-2)+\dots+1+0 \sim N^2/2$

**Перестановок:**  $N$

**Сложность алгоритма:**  $O(N^2)$

**Время работы алгоритма:** не зависит от порядка расположения исходных данных. Квадратичное, даже если исходный массив отсортирован.

**Плюсы:** Количество перестановок минимально

**Минусы:** Очень высокая вычислительная сложность  $O(N^2)$

# Сортировка вставками. **Insertion Sort.**

---

Идея

алгоритма

- движемся по массиву элементов слева на право
- на каждой итерации  $i$  меняем местами  $a[i]$  с каждым элементом слева от  $a[i]$  и большим его

(см. Example5, Project InsertionSort)

# Сортировка вставками. Демо.

7	10	5	3	8	4	2	9	6
---	----	---	---	---	---	---	---	---

$i$   $j$

7	10	5	3	8	4	2	9	6
---	----	---	---	---	---	---	---	---

$i$   $j$

7	10	5	3	8	4	2	9	6
---	----	---	---	---	---	---	---	---

$i$   $j$

7	5	10	3	8	4	7	9	6
---	---	----	---	---	---	---	---	---

$j$   $i$

5	7	10	3	8	4	7	9	6
---	---	----	---	---	---	---	---	---

$j$   $i$

5	7	10	3	8	4	7	9	6
---	---	----	---	---	---	---	---	---

$j$   $i$

5	7	3	10	8	4	7	9	6
---	---	---	----	---	---	---	---	---

$j$   $i$

5	3	7	10	8	4	7	9	6
---	---	---	----	---	---	---	---	---

$j$   $i$

# Сортировка вставками. Реализация.

1. перемещаемся по массиву слева направо

```
i++;
```



2. двигаемся справа-налево от  $i$ -го элемента и меняем местами с каждым, большим  $a[i]$

```
for (int j=i; j>0; j--)  
    if ( less(a[j], a[j-1]) )  
        swap(a, j, j-1);  
else break;
```



# Сортировка вставками. Анализ.

---

**Сравнений:**  $\sim 1/4 N^2$  в среднем

**Перестановок:**  $\sim 1/4 N^2$  в среднем

**Сложность алгоритма:**  $O(N^2)$

**Наилучший случай:** если массив отсортирован, то алгоритм выполняет  $N-1$  сравнение и 0 перестановок.

**Наихудший случай:** если массив отсортирован в обратном порядке, то алгоритм выполняет  $\sim 1/2 N^2$  сравнений и  $\sim 1/2 N^2$  перестановок

## **Плюсы:**

- эффективен на небольших наборах данных (до десятков элементов)
- эффективен на частично-отсортированных наборах данных

**Минусы:** Очень высокая вычислительная сложность  $O(N^2)$

# Сортировка простыми обменами. **Bubble Sort.**

---

## Идея алгоритма

- Алгоритм состоит из повторяющихся проходов по массиву
- За каждый проход элементы сравниваются попарно.
- Если порядок в паре неверный, то выполняется обмен элементов.
- Проходы выполняются  $n-1$  раз или до тех пор, пока на очередном шаге окажется, что обмены больше не нужны, т.е массив отсортирован.

(см. Example5, Project BubbleSort)

# Bubble sort. Demo.

---

7	10	5	3	8	4	2	9	6
---	----	---	---	---	---	---	---	---

Swap (6,9)

7	10	5	3	8	4	2	6	9
---	----	---	---	---	---	---	---	---

Don't swap

7	10	5	3	8	4	2	6	9
---	----	---	---	---	---	---	---	---

Swap (2,4)

7	10	5	3	8	2	4	6	9
---	----	---	---	---	---	---	---	---

Swap (2,8)

			...					
--	--	--	-----	--	--	--	--	--

2	7	10	5	3	8	4	6	9
---	---	----	---	---	---	---	---	---

Конец 1-го прохода

2	3	7	10	5	4	8	6	9
---	---	---	----	---	---	---	---	---

Конец 2-го прохода

			...					
--	--	--	-----	--	--	--	--	--

# Bubble Sort. Реализация.

---

1. Выполняем  $i$ -ый проход, перестановок не было

```
int i=0; bool swapped=false;
```

2. Сравниваем все пары соседних элементов  $a[j]$ ,  $a[j-1]$ . Если порядок нарушен, выполняем перестановку

```
for (int j=n-1; j>i; j--)  
    if ( less(a[j], a[j-1]) )  
        {  
            swap(a, j, j-1);  
            swapped=true;  
        }
```

3. Если перестановок не было, то завершаем проходы, иначе следующий проход ( $i++$ )

```
if (!swapped) break;
```

# Bubble sort. Анализ.

---

**Сравнений:**  $(N-1)*N$

**Перестановок:**  $(N-1)*N/2$

**Сложность алгоритма:**  $O(N^2)$

**Наилучший случай:** если массив отсортирован, то алгоритм выполняет  $N*(N-1)$  сравнений и 0 перестановок.

**Наихудший случай:** если массив отсортирован в обратном порядке, то алгоритм выполняет  $N*(N-1)$  сравнений  $(N-1)*N/2$

**Плюсы:**

**Минусы:** Очень высокая вычислительная сложность  $O(N^2)$  для любых случаев расположения исходных данных