

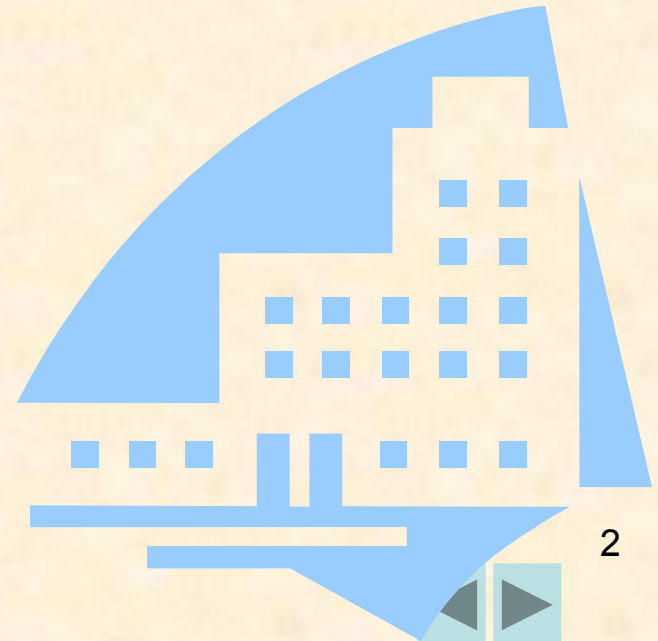
# Методы сортировки данных

дисциплина «Основы алгоритмизации и программирования 2 курс

Соколова Наталья Валентиновна,  
преподаватель спец. Дисциплин  
[Sokolova\\_natali@list.ru](mailto:Sokolova_natali@list.ru)

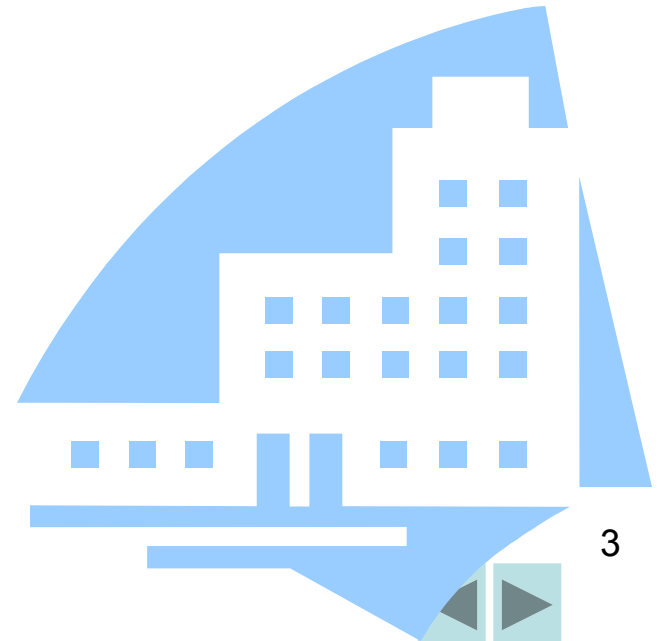


**Сортировка объектов –  
расположение объектов по  
возрастанию или убыванию  
согласно определенному  
линейному отношению порядка**



# Сортировка объектов:

- Внутренняя
- Внешняя



## **Внутренняя сортировка**

**оперирует с массивами, целиком помещающимися в оперативной памяти с произвольным доступом к любой ячейке.**

**Данные обычно сортируются на том же месте, без дополнительных затрат**



# Методы внутренней сортировки

прямые методы

вставкой (включением)

выбором (выделением)

обменом ("пузырьковая")

улучшенные методы

Быстрая

Шелла

# Алгоритм сортировки вставкой

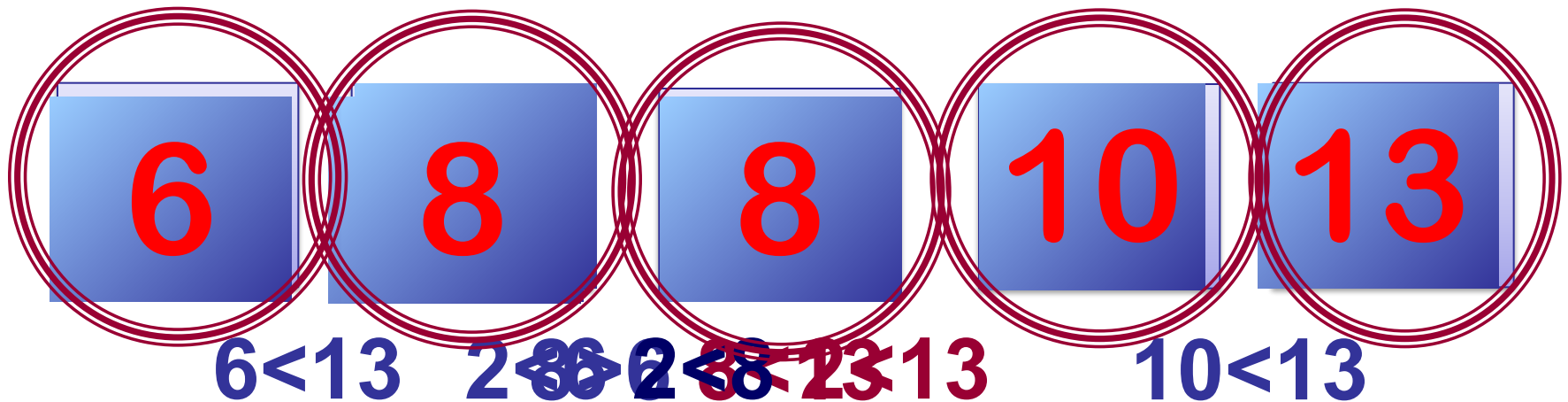


## Суть сортировки:

- ✓ Упорядочиваются два элемента массива
- ✓ Вставка третьего элемента в соответствующее место по отношению к первым двум элементам.
- ✓ Этот процесс повторяется до тех пор, пока все элементы не будут упорядочены.



# Сортировка вставкой по возрастанию



Массив отсортирован  
по возрастанию





# Постановка задачи

Пусть нужно отсортировать массив **A** по возрастанию, в котором **N** элементов методом вставки

Вспомогательные переменные

**j** – номер первого элемента остатка.

**i** – номер перемещаемого элемента.

**f** – условие выхода из цикла (если **f=1**, то выход)

**Val** – промежуточное значение, используемое для перемещения элементов массив



*Начало алгоритма.*

**Шаг 1**  $j:=2$ ,

**Шаг 2** Пока  $j \leq N$  выполнять:

**Шаг 2.1**  $i:=j$ ;  $f:=0$ ,

**Шаг 2.2** Пока  $i \geq 2$  и  $f=0$  выполнять:

**Шаг 2.2.1** Если  $A[i-1] > A[i]$

то  $Val:=A[i-1]$ ;

$A[i-1]:=A[i]$ ;

$A[i]:=Val$ ,

иначе  $f:=1$ ,

**Шаг 2.2.2**  $i:=i-1$ ,

**Шаг 2.3**  $j:=j+1$ .

*Конец алгоритма.*



# Что обозначает данное условие?

*Начало алгоритма.*

**Шаг 1**  $j:=2$ ,

**Шаг 2** Пока  $j \leq N$  выполнять:

**Шаг 2.1**  $i:=j$ ;  $f:=0$ ,

**Шаг 2.2** Пока  $i \geq 2$  и  $f=0$  выполнять:

**Шаг 2.2.1** Если  $A[i-1] > A[i]$

то  $Val:=A[i-1]$ ;

$A[i-1]:=A[i]$ ;

$A[i]:=Val$ ,

иначе  $f:=1$ ,

**Шаг 2.2.2**  $i:=i-1$ ,

**Шаг 2.3**  $j:=j+1$ .

*Конец алгоритма.*



# Почему стартовое значение $j = 2$ ?

*Начало алгоритма.*

**Шаг 1**  $j := 2,$

**Шаг 2** Пока  $j \leq N$  выполнять:

**Шаг 2.1**  $i := j; f := 0,$

**Шаг 2.2** Пока  $i \geq 2$  и  $f = 0$  выполнять:

**Шаг 2.2.1** Если  $A[i-1] > A[i]$

то  $Val := A[i-1];$

$A[i-1] := A[i];$

$A[i] := Val,$

иначе  $f := 1,$

**Шаг 2.2.2**  $i := i - 1,$

**Шаг 2.3**  $j := j + 1.$

*Конец алгоритма.*



# Почему стартовое значение $i = 2$ ?

*Начало алгоритма.*

**Шаг 1**  $j := 2$ ,

**Шаг 2** Пока  $j \leq N$  выполнять:

**Шаг 2.1**  $i := j$ ;  $f := 0$ ,

**Шаг 2.2** Пока  $i \geq 2$  и  $f = 0$  выполнять:

**Шаг 2.2.1** Если  $A[i-1] > A[i]$

то  $Val := A[i-1]$ ;

$A[i-1] := A[i]$ ;

$A[i] := Val$ ,

иначе  $f := 1$ ,

**Шаг 2.2.2**  $i := i - 1$ ,

**Шаг 2.3**  $j := j + 1$ .

*Конец алгоритма.*



**Всегда ли происходит обмен входного  $j$  элемента с отсортированным элементом ?**

*Начало алгоритма.*

**Шаг 1**  $j:=2$ ,

**Шаг 2** Пока  $j \leq N$  выполнять:

**Шаг 2.1**  $i:=j$ ;  $f:=0$ ,

**Шаг 2.2** Пока  $i \geq 2$  и  $f=0$  выполнять:

**Шаг 2.2.1** Если  $A[i-1] > A[i]$

то  $Val:=A[i-1]$ ;

$A[i-1]:=A[i]$ ;

$A[i]:=Val$ ,

иначе  $f:=1$ ,

**Шаг 2.2.2**  $i:=i-1$ ,

**Шаг 2.3**  $j:=j+1$ .

*Конец алгоритма.*



# Возможно ли заменить цикл ПОКА и ЕСЛИ одним циклом ПОКА с условием $i \geq 2$ и $A[i-1] > A[i]$ ?

*Начало алгоритма.*

**Шаг 1**  $j := 2$ ,

**Шаг 2** Пока  $j \leq N$  выполнять:

**Шаг 2.1**  $i := j$ ;  $f := 0$ ,

**Шаг 2.2** Пока  $i \geq 2$  и  $f = 0$  выполнять:

**Шаг 2.2.1** Если  $A[i-1] > A[i]$

то  $Val := A[i-1]$ ;

$A[i-1] := A[i]$ ;

$A[i] := Val$ ,

иначе  $f := 1$ ,

**Шаг 2.2.2**  $i := i - 1$ ,

**Шаг 2.3**  $j := j + 1$ .

*Конец алгоритма.*



# Для чего нужен этот оператор?

*Начало алгоритма.*

**Шаг 1**  $j:=2$ ,

**Шаг 2** Пока  $j \leq N$  выполнять:

**Шаг 2.1**  $i:=j$ ;  $f:=0$ ,

**Шаг 2.2** Пока  $i \geq 2$  и  $f=0$  выполнять:

**Шаг 2.2.1** Если  $A[i-1] > A[i]$

то  $Val:=A[i-1]$ ;

$A[i-1]:=A[i]$ ;

$A[i]:=Val$ ,

иначе  $f:=1$ ,

**Шаг 2.2.2**  $i:=i-1$ ,

**Шаг 2.3**  $j:=j+1$ .

*Конец алгоритма.*





# Алгоритм сортировки выбором

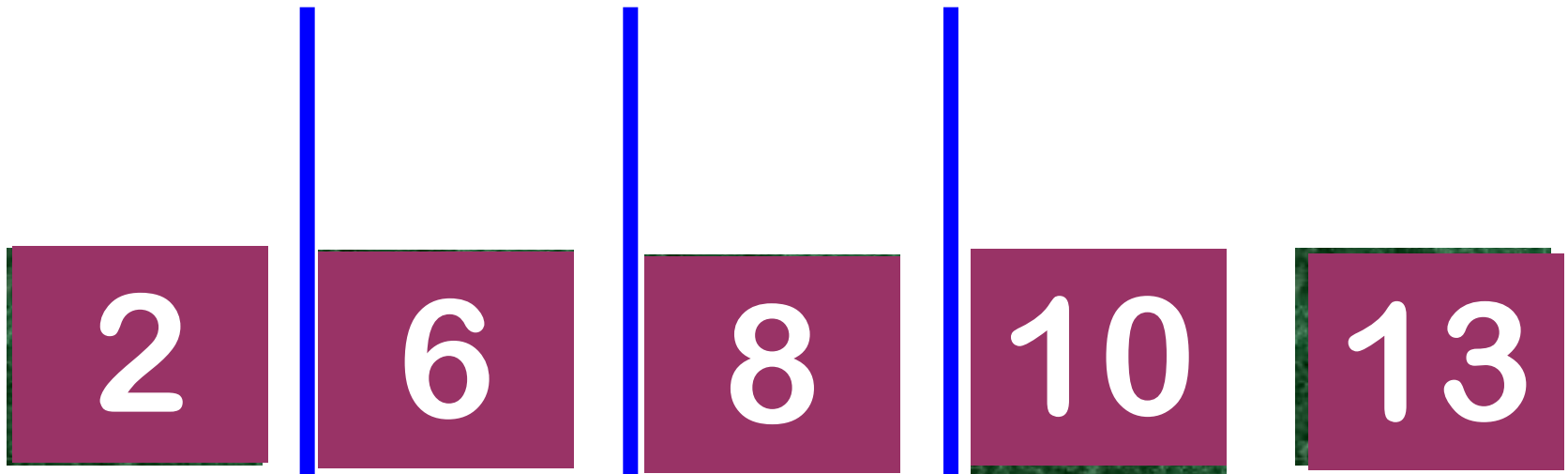


## Суть сортировки:

- ✓ Выбирается элемент с наименьшим значением и делается его обмен с первым элементом массива.
- ✓ Затем находится элемент с наименьшим значением из оставшихся  $n-1$  элементов и делается его обмен со вторым элементом и т.д. до обмена двух последних



# Сортировка выбором по возрастанию



Отсортированная часть массива  
Массив отсортирован по возрастанию



# Постановка задачи

Пусть нужно отсортировать массив **A** по возрастанию, в котором **N** элементов методом выбора.

Вспомогательные переменные

**j** – номер первого элемента остатка.

**i** – номер перемещаемого элемента.

**min** – минимальное число в массиве.

**imin** – номер минимального числа в массиве



## *Начало алгоритма.*

**Шаг 1**  $j:=1$ ,

**Шаг 2** Пока  $j \leq N-1$  выполнять:

**Шаг 2.1**  $\min:=a[j]$ ,  $I_{\min}:=j$ ,  $i:=j+1$

**Шаг 2.2** Пока  $i \leq N$  выполнять:

**Шаг 2.2.1** Если  $A[i] < \min$ ,

то  $\min:=a[i]$ ,  $I_{\min}:=i$

**Шаг 2.2.2**  $i:=i+1$ ,

**Шаг 2.3**  $A[I_{\min}] := A[j]$ ,  $A[j] := \min$

**Шаг 2.4**  $j:=j+1$ .

*Конец алгоритма.*



# Что происходит в выделенном фрагменте?

*Начало алгоритма.*

**Шаг 1**  $j:=1$ ,

**Шаг 2** Пока  $j \leq N-1$  выполнять:

**Шаг 2.1**  $\min:=a[j]$ ,  $I_{\min}:=j$ ,  $i:=j+1$

**Шаг 2.2** Пока  $i \leq N$  выполнять:

**Шаг 2.2.1** Если  $A[i] < \min$ ,  
то  $\min:=a[i]$ ,  $I_{\min}:=i$

**Шаг 2.2.2**  $i:=i+1$ ,

**Шаг 2.3**  $A[I_{\min}] := A[j]$ ,  $A[j] := \min$

**Шаг 2.4**  $j:=j+1$ .

*Конец алгоритма.*



# Что происходит в выделенном фрагменте?

*Начало алгоритма.*

**Шаг 1**  $j:=1$ ,

**Шаг 2** Пока  $j \leq N-1$  выполнять:

**Шаг 2.1**  $\min:=a[j]$ ,  $I_{\min}:=j$ ,  $i:=j+1$

**Шаг 2.2** Пока  $i \leq N$  выполнять:

**Шаг 2.2.1** Если  $A[i] < \min$ ,  
то  $\min:=a[i]$ ,  $I_{\min}:=i$

**Шаг 2.2.2**  $i:=i+1$ ,

**Шаг 2.3**  $A[I_{\min}]:=A[j]$ ,  $A[j]:=\min$

**Шаг 2.4**  $j:=j+1$ .

*Конец алгоритма.*



# **Алгоритм сортировки обменом («пузырьковая»)**



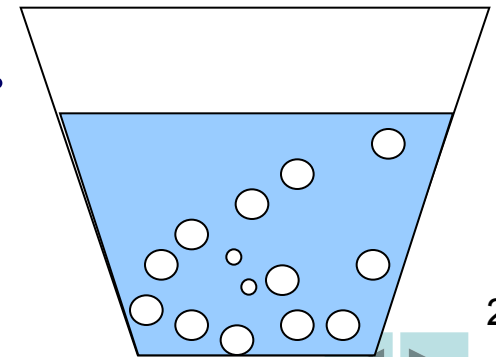


## Суть сортировки:

Последовательно просматривается массив и сравнивается каждая пара элементов между собой.

При этом "неправильное" расположение элементов устраняется путем их перестановки.

Процесс просмотра и сравнения элементов повторяется до просмотра всего массива.



# Сортировка обменом

по возрастанию

Первый просмотр



Массив отсортирован по возрастанию



# Постановка задачи

Пусть нужно отсортировать массив **A** по возрастанию, в котором **N** элементов методом обмена

Вспомогательные переменные

**j** – номер первого элемента остатка.

**i** – номер перемещаемого элемента.

**Val** – промежуточное значение, используемое для перемещения элементов массива



# *Начало алгоритма.*

**Шаг 1**  $j := N$ ,

**Шаг 2** Пока  $j \geq 2$  выполнять:

**Шаг 2.1**  $i := 1$ ;

**Шаг 2.2** Пока  $i \leq j - 1$  выполнять:

**Шаг 2.2.1** Если  $A[i] > A[i+1]$

то  $Val := A[i]$ ;

$A[i] := A[i+1]$ ;

$A[i+1] := Val$ ,

**Шаг 2.2.2**  $i = i + 1$ ,

**Шаг 2.3**  $j = j - 1$ .

*Конец алгоритма.*

Сравнение  
соседних  
элементов

Обмен  
соседних  
элементов  
местами, в  
случае если  
левый  
больше  
правого

Формируется  
отсортированная  
часть



Почему условие такое?

*Начало алгоритма.*

**Шаг 1**  $j := N$ ,

**Шаг 2** Пока  $j \geq 2$  выполнять:

**Шаг 2.1**  $i := 1$ ;

**Шаг 2.2** Пока  $i \leq j - 1$  выполнять:

**Шаг 2.2.1** Если  $A[i] > A[i+1]$

то  $Val := A[i]$ ;

$A[i] := A[i+1]$ ;

$A[i+1] := Val$ ,

**Шаг 2.2.2**  $i = i + 1$ ,

**Шаг 2.3**  $j := j - 1$ .

*Конец алгоритма.*



Почему значение  $j$  уменьшается?

Можно ли увеличивать? Что нужно изменить?

*Начало алгоритма.*

**Шаг 1**  $j := N$ ,

**Шаг 2** Пока  $j \geq 2$  выполнять:

**Шаг 2.1**  $i := 1$ ;

**Шаг 2.2** Пока  $i \leq j - 1$  выполнять:

**Шаг 2.2.1** Если  $A[i] > A[i+1]$

то  $Val := A[i]$ ;

$A[i] := A[i+1]$ ;

$A[i+1] := Val$ ,

**Шаг 2.2.2**  $i = i + 1$ ,

**Шаг 2.3**  $j := j - 1$ .

*Конец алгоритма.*



# Алгоритм сортировки Шелла



- ✓ Классифицируется как «**слияние вставкой**»;
- ✓ Называется «**сортировкой с убывающим шагом**»
- ✓ **Общий метод, который использует сортировку вставкой, применяет принцип уменьшения расстояния между сравниваемыми элементами**





## Условия реализации:

**?** Конкретная последовательность шагов может быть другой, но последний шаг должен быть равен 1;

**?** Следует избегать последовательность, которые являются степенями 2 (т.е. нельзя использовать последовательность шагов – 4,2)



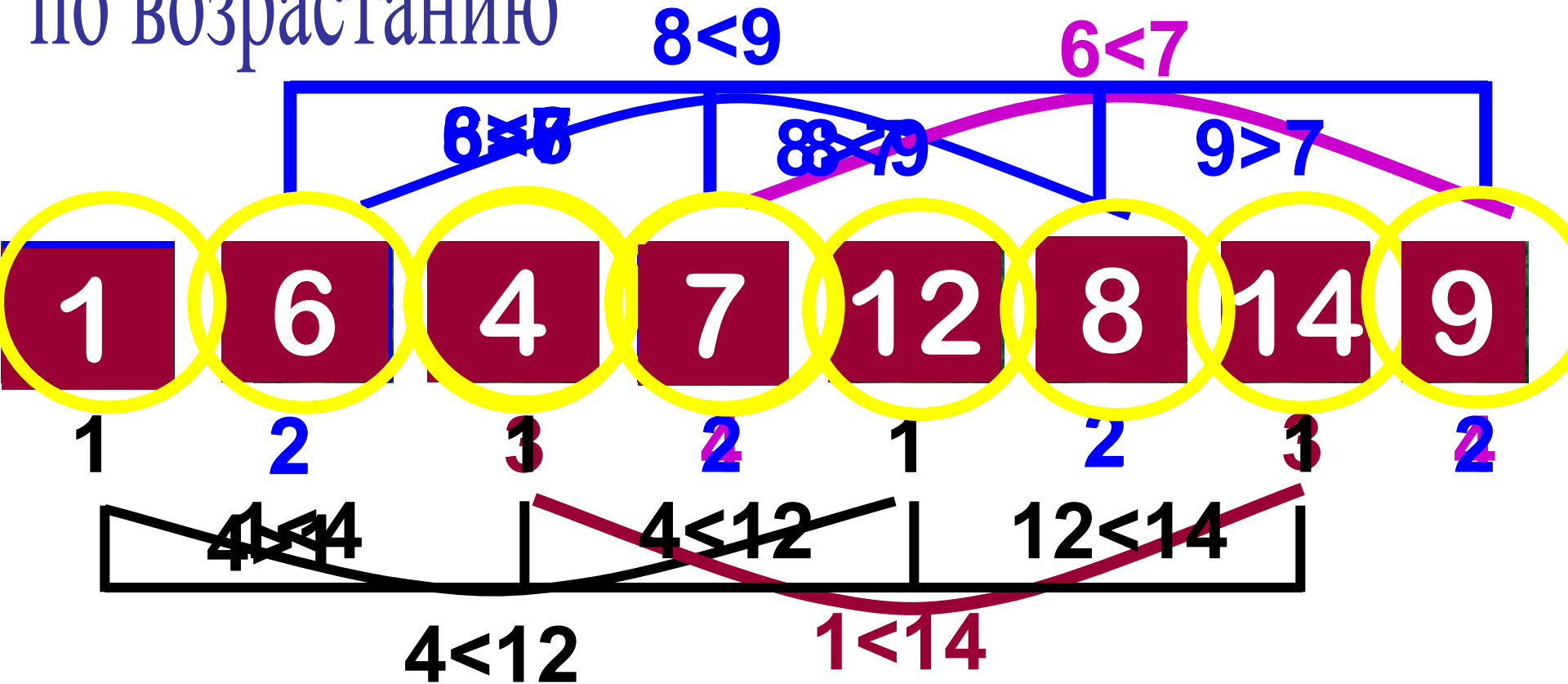
# Суть сортировки:

- ✓ Сначала сортируются все элементы, отстоящие друг от друга на три позиции
- ✓ Затем сортируются элементы, расположенные на расстоянии двух позиций
- ✓ Наконец, сортируются все соседние элементы



# Сортировка Шелла

2 шаг. 2 группы из 4-х элементов  
по возрастанию



# Сортировка Шелла

3 шаг. 1 группа из 8-ми элементов  
по возрастанию



Массив отсортирован по  
возрастанию



# Постановка задачи

Пусть нужно отсортировать массив **A** по возрастанию, в котором **N** элементов методом Шелла

Вспомогательные переменные

**j** – номер первого элемента остатка.

**i** – номер перемещаемого элемента.

**M**– оптимальный шаг

**P**– промежуточное значение,  
используемое для перемещения элементов массива



*Начало алгоритма.*

**Шаг 1.**  $M = \text{целая часть } N/2$

**Шаг 2.** Пока  $M \neq 0$  выполнять

**Шаг 2.1.**  $i := M + 1$

**Шаг 2.2.** Пока  $i \leq N$  выполнять

**Шаг 2.2.1.**  $P = A[i]$

**Шаг 2.2.2.**  $j = i - M$

**Шаг 2.2.3.** Пока  $j > 0$  и  $P < A[j]$  выполнять

**Шаг 2.2.3.1**  $A[j + M] = A[j]$

**Шаг 2.2.3.2**  $j = j - M$

**Шаг 2.2.4.**  $A[j + M] = P$

**Шаг 2.2.5.**  $i = i + 1$

**Шаг 2.3.**  $M = \text{целая часть } M/2$

*Конец алгоритма.*



# *Начало алгоритма.*

**Шаг 1.**  $M = \text{целая часть } N/2$

Определение максимальной величины шага

**Шаг 2.** Пока  $M \neq 0$  выполнять

**Шаг 2.1.**  $i := M + 1$

**Шаг 2.2.** Пока  $i \leq N$  выполнять

**Шаг 2.2.1.**  $P = A[i]$

**Шаг 2.2.2.**  $j = i - M$

Определение первого и последнего элемента сравнения для текущего шага

**Шаг 2.2.3.** Пока  $j > 0$  и  $P < A[j]$  выполнять

**Шаг 2.2.3.1**  $A[j+M] = A[j]$

**Шаг 2.2.3.2**  $j = j - M$

**Шаг 2.2.4.**  $A[j] = P$

**Шаг 2.2.5.**  $i = i + 1$

Сортировка элементов последовательности

**Шаг 2.3.**  $M = \text{целая часть } M/2$

Определение нового шага сравнения

*Конец алгоритма.*

# Зачем необходимо это действие?

*Начало алгоритма.*

**Шаг 1.**  $M = \text{целая часть } N/2$

**Шаг 2.** **Пока**  $M \neq 0$  **выполнять**

**Шаг 2.1.**  $i := M + 1$

**Шаг 2.2.** **Пока**  $i \leq N$  **выполнять**

**Шаг 2.2.1.**  $P = A[i]$

**Шаг 2.2.2.**  $j = i - M$

**Шаг 2.2.3.** **Пока**  $j > 0$  и  $P < A[j]$  **выполнять**

**Шаг 2.2.3.1**  $A[j+M] = A[j]$

**Шаг 2.2.3.2**  $j = j - M$

**Шаг 2.2.4.**  $A[j] = P$

**Шаг 2.2.5.**  $i = i + 1$

**Шаг 2.3.**  $M = \text{целая часть } M/2$

*Конец алгоритма.*





# Зачем необходимо это действие?

*Начало алгоритма.*

**Шаг 1.**  $M = \text{целая часть } N/2$

**Шаг 2.** Пока  $M \neq 0$  выполнять

**Шаг 2.1.**  $i := M + 1$

**Шаг 2.2.** Пока  $i \leq N$  выполнять

**Шаг 2.2.1.**  $P = A[i]$

**Шаг 2.2.2.**  $j = i - M$

**Шаг 2.2.3.** Пока  $j > 0$  и  $P < A[j]$  выполнять

**Шаг 2.2.3.1**  $A[j+M] = A[j]$

**Шаг 2.2.3.2**  $j = j - M$

**Шаг 2.2.4.**  $A[j] = P$

**Шаг 2.2.5.**  $i = i + 1$

**Шаг 2.3.**  $M = \text{целая часть } M/2$

*Конец алгоритма.*



# Почему условие выхода из цикла такое?

*Начало алгоритма.*

**Шаг 1.**  $M = \text{целая часть } N/2$

**Шаг 2.** Пока  $M \neq 0$  выполнять

**Шаг 2.1.**  $i := M + 1$

**Шаг 2.2.** Пока  $i \leq N$  выполнять

**Шаг 2.2.1.**  $P = A[i]$

**Шаг 2.2.2.**  $j = i - M$

**Шаг 2.2.3.** Пока  $j > 0$  и  $P < A[j]$  выполнять

**Шаг 2.2.3.1**  $A[j+M] = A[j]$

**Шаг 2.2.3.2**  $j = j - M$

**Шаг 2.2.4.**  $A[j] = P$

**Шаг 2.2.5.**  $i = i + 1$

**Шаг 2.3.**  $M = \text{целая часть } M/2$

*Конец алгоритма.*



# Алгоритм быстрой сортировки



- ✓ Придумана Ч.А.Р. Хоаром (Charles Antony Richard Hoare) ;
- ✓ В основе – сортировка обменами ;
- ✓ Основана на делении массива



# Суть сортировки:

- ✓ Выбирается некоторое значение  $(x)$  – **барьерный элемент**, который определяется округлением до целого деления количества сортируемых элементов на 2;
- ✓ Просматриваем массив, двигаясь слева направо, пока не найдется элемент, **больший  $x$**
- ✓ Затем просматриваем его справа налево, пока не найдется элемент, **меньший  $x$**



## Суть сортировки:

- ✓ Меняем найденные элементы местами. В случае, если не найден наибольший или наименьший элементы, местами меняется средний элемент с найденным наибольшим или наименьшим элементом;
- ✓ Дойдя до середины имеем 2 части массива;
- ✓ Процесс продолжается для каждой части, пока массив не будет отсортирован



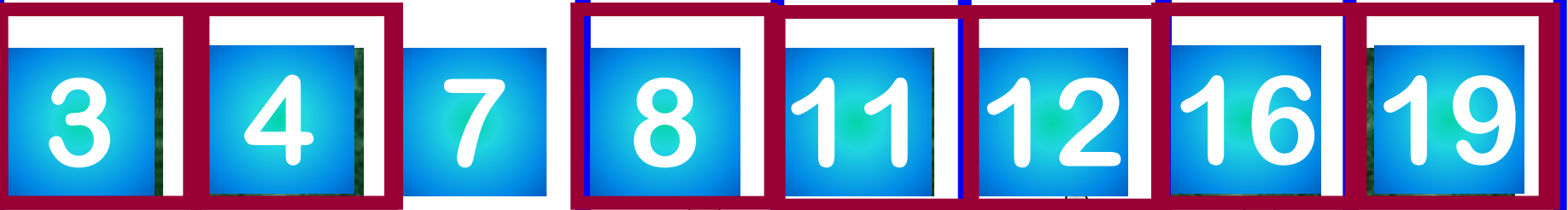
# Быстрая сортировка

по возрастанию

Меньше

Больше 7

равно 7



Отсортированная часть

$19 > 7$

Барьерный элемент

Массив отсортирован по возрастанию

8 2 1 7 1 1 8 19 7 12 1 12 12 12

переносим в правую часть, т.к.  $19 > 7$ .  
 не переносим, т.к.  $12 \leq 7$ .  
 не переносим, т.к.  $12 \leq 7$ .  
 поэтому меняем местами 12 и 19.

# Постановка задачи

Пусть нужно отсортировать массив **A** по возрастанию, в котором **n** элементов быстрым методом

Вспомогательные переменные:

**t** – конечный элемент массива

**m** - начальный элемент массива

**x** – элемент относительно которого перемещаются все остальные элементы.

**w** – промежуточное значение, используемое для перемещения элементов массива





# *Начало алгоритма.*

**Шаг 1**  $i=m$   $j=t$

**Шаг 2**  $x=A[\text{округление до целого}(m+t)/2]$

**Шаг 3** Пока  $i \leq j$  выполнять:

**Шаг 3.1** Если  $A[i] < x$  то  $i:=i+1$ ,

иначе

Если  $A[j] > x$  то  $j:=j-1$

иначе

$w:=A[i]; A[i]:=A[j]; A[j]:=w$   
 $i:=i+1, j:=j-1$

Рекурсивный вызов  
процедуры

**Шаг 4** Если  $m < j$  то *Алгоритм* ( $A, m, j$ );

**Шаг 5** Если  $i < t$  то *Алгоритм* ( $A, i, t$ ).

*Конец алгоритма.*



# Зачем необходимо это действие?

*Начало алгоритма.*

**Шаг 1**  $i=m$   $j=t$

**Шаг 2**  $x=A[\text{округление до целого}(m+t)/2]$

**Шаг 3** Пока  $i \leq j$  выполнять:

**Шаг 3.1** Если  $A[i] < x$  то  $i:=i+1$ ,

иначе

Если  $A[j] > x$  то  $j:=j-1$

иначе

$w:=A[i]; A[i]:=A[j]; A[j]:=w$

$i:=i+1, j:=j-1$

**Шаг 4** Если  $m < j$  то *Алгоритм* ( $A, m, j$ );

**Шаг 5** Если  $i < t$  то *Алгоритм* ( $A, i, t$ ).

*Конец алгоритма.*



# Если исключить условие и просто вызвать процедуру, что может произойти?

*Начало алгоритма.*

**Шаг 1**  $i:=m$   $j:=t$

**Шаг 2**  $x=A[\text{округление до целого}(m+t)/2]$

**Шаг 3** Пока  $i \leq j$  выполнять:

**Шаг 3.1** Если  $A[i] < x$  то  $i:=i+1$ ,

иначе

Если  $A[j] > x$  то  $j:=j-1$

иначе

$w:=A[i]; A[i]:=A[j]; A[j]:=w$

$i:=i+1, j:=j-1$

**Шаг 4** Если  $m < j$  то *Алгоритм* ( $A, m, j$ );

**Шаг 5** Если  $i < t$  то *Алгоритм* ( $A, i, t$ ).

*Конец алгоритма.*



**Если изменить условие цикла на  $i < j$  что произойдет ?**

*Начало алгоритма.*

**Шаг 1**  $i = m$   $j = t$

**Шаг 2**  $x = A[\text{округление до целого}((m+t)/2)]$

**Шаг 3** Пока  $i \leq j$  выполнять:

**Шаг 3.1** Если  $A[i] < x$  то  $i := i + 1$ ,

иначе

Если  $A[j] > x$  то  $j := j - 1$

иначе

$w := A[i]; A[i] := A[j]; A[j] := w$

$i := i + 1, j := j - 1$

**Шаг 4** Если  $m < j$  то *Алгоритм* ( $A, m, j$ );

**Шаг 5** Если  $i < t$  то *Алгоритм* ( $A, i, t$ ).

*Конец алгоритма.*



# Что происходит в выделенном фрагменте?

*Начало алгоритма.*

**Шаг 1**  $i=m$   $j=t$

**Шаг 2**  $x=A[\text{округление до целого}(m+t)/2]$

**Шаг 3** Пока  $i \leq j$  выполнять:

**Шаг 3.1** Если  $A[i] < x$  то  $i:=i+1$ ,

иначе

Если  $A[j] > x$  то  $j:=j-1$

иначе

$w:=A[i]; A[i]:=A[j]; A[j]:=w$

$i:=i+1, j:=j-1$

**Шаг 4** Если  $m < j$  то *Алгоритм* ( $A, m, j$ );

**Шаг 5** Если  $i < t$  то *Алгоритм* ( $A, i, t$ ).

*Конец алгоритма.*



# Что происходит в выделенном фрагменте?

*Начало алгоритма.*

**Шаг 1**  $i=m$   $j=t$

**Шаг 2**  $x=A[\text{округление до целого}(m+t)/2]$

**Шаг 3** Пока  $i \leq j$  выполнять:

**Шаг 3.1** Если  $A[i] < x$  то  $i:=i+1$ ,

иначе

Если  $A[j] > x$  то  $j:=j-1$

иначе

$w:=A[i]; A[i]:=A[j]; A[j]:=w$

$i:=i+1, j:=j-1$

**Шаг 4** Если  $m < j$  то *Алгоритм* ( $A, m, j$ );

**Шаг 5** Если  $i < t$  то *Алгоритм* ( $A, i, t$ ).

*Конец алгоритма.*



# Что происходит с равными элементами?

*Начало алгоритма.*

**Шаг 1**  $i=m$   $j=t$

**Шаг 2**  $x=A[\text{округление до целого}(m+t)/2]$

**Шаг 3** Пока  $i \leq j$  выполнять:

**Шаг 3.1** Если  $A[i] < x$  то  $i:=i+1$ ,

иначе

Если  $A[j] > x$  то  $j:=j-1$

иначе

$w:=A[i]; A[i]:=A[j]; A[j]:=w$

$i:=i+1, j:=j-1$

**Шаг 4** Если  $m < j$  то *Алгоритм* ( $A, m, j$ );

**Шаг 5** Если  $i < t$  то *Алгоритм* ( $A, i, t$ ).

*Конец алгоритма.*



# Оценка эффективности

Основывается:

- количестве необходимых сравнений
- количестве пересылок





# Параметры оценки алгоритмов

- ✓ **Время сортировки** - основной параметр, характеризующий быстродействие алгоритма
- ✓ **Память** – выделяется ли дополнительная память под временное хранение данных



# Параметры оценки алгоритмов



Исходные данные



**Устойчивость** — отсортированный массив не меняет порядок элементов с одинаковыми значениями.

Пример работы неустойчивой сортировки.

Взаимное расположение равных элементов с ключом 1 и

дополнительными полями "a", "b", "c"

**ИЗМЕНИЛОСЬ**



# Параметры оценки алгоритмов

- ✓ **Естественность поведения** - эффективность метода при обработке уже отсортированных, или частично отсортированных данных. Алгоритм ведет себя естественно, если учитывает эту характеристику входной последовательности и работает лучше



# Оценка алгоритма сортировки выбором

- Общее количество сравнений  
 $C = N-1 + N-2 + \dots + 1 = (N^2 - N)/2$
- Общее количество операций  
 $n + (n-1) + (n-2) + (n-3) + \dots + 1 = 1/2 * (n^2 + n) = \text{Theta}(n^2)$
- Число обменов < числа сравнений =  
время сортировки растет квадратично  
относительно количества элементов

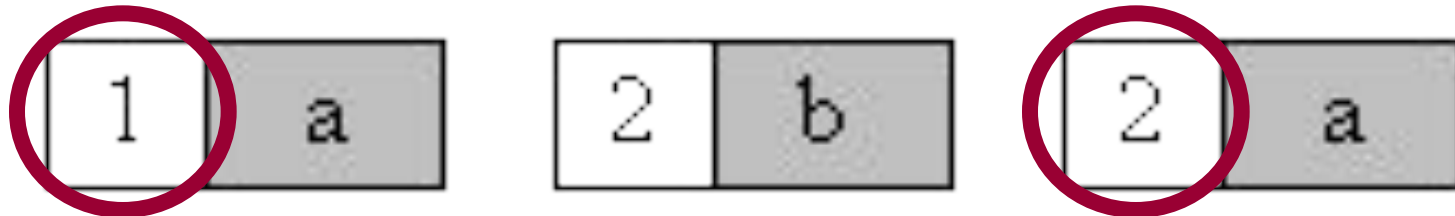


# Устойчив ли этот метод?



исходная последовательность

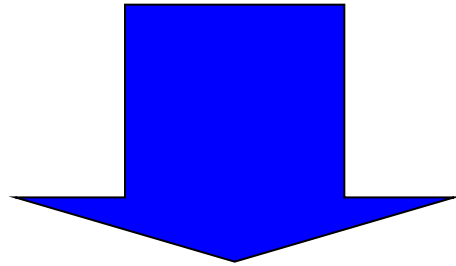
# Не устойчив



шаг 0: 2 ↔ 1



**Если входная последовательность  
почти упорядочена, то сравнений  
будет столько же**



**алгоритм ведет  
себя неестественно**



# Оценка алгоритма сортировки вставкой

Для массива **1 2 3 4 5 6 7 8**  
потребуется  **$N-1$**  сравнение.

Для массива **8 7 6 5 4 3 2 1**  
потребуется  **$(N^2-N)/2$**  сравнение.

Общее количество операций  
 **$\Theta(n^2)$**



# *Устойчив ли этот метод?*



# **УСТОЙЧИВ**

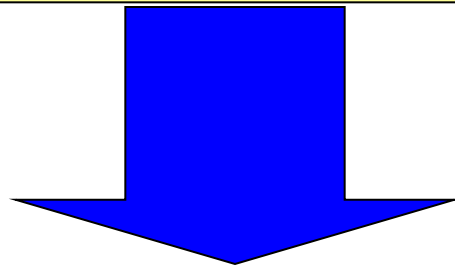


*порядок элементов с одинаковыми  
ключами не изменяется*





**Наименьшие оценки эффективности,  
когда элементы предварительно  
упорядочены, а наибольшие – когда  
элементы расположены в обратном  
порядке**



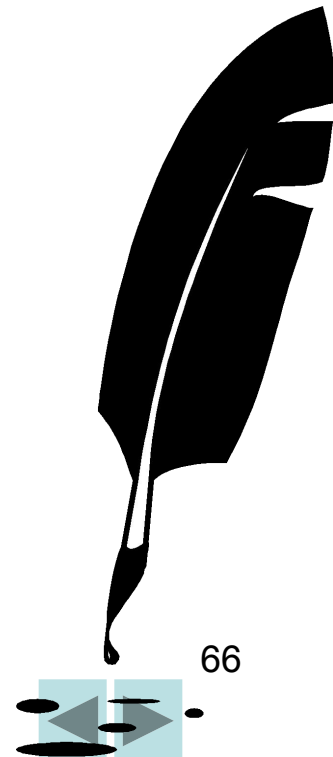
**алгоритм ведет  
себя естественно**





В совокупности устойчивость и естественность поведения алгоритма, делает метод хорошим выбором в соответствующих ситуациях

Не эффективный метод, так как включение элемента связано со сдвигом всех предшествующих элементов на одну позицию, а эта операция неэкономна



# Оценка алгоритма сортировки обменом

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

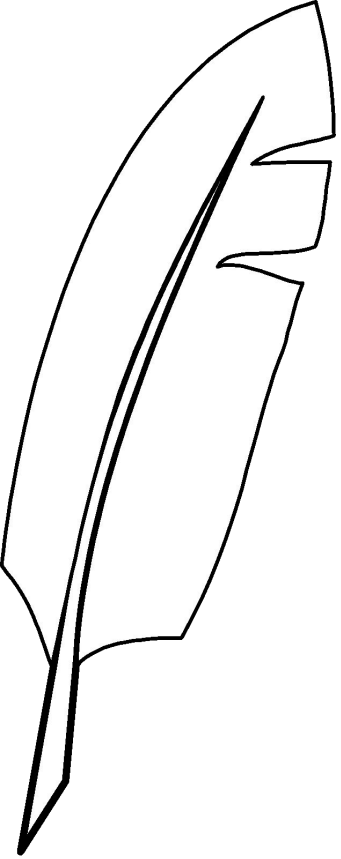
Общее количество операций  
 $\Theta(n^2)$



# Ответьте на следующие вопросы:

- *Устойчив ли этот метод?*
- *Естественное ли поведение этого алгоритма?*

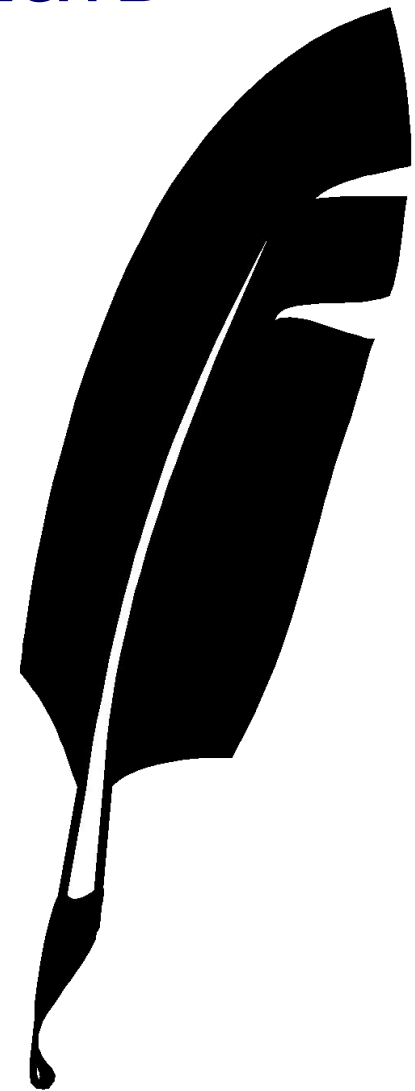




Прост, и его можно улучшить

Очень медленен и  
малоэффективен.

На практике, даже с  
улучшениями, работает,  
слишком медленно, поэтому  
почти не применяется.



# Сравнение методов простой сортировки

|                   | Минимум                         | Максимум                                |
|-------------------|---------------------------------|---|
| Простые включения | $C = n-1$<br>$M = 2(n-1)$       | $C = (n^2-n)/2$<br>$M = (n^2+3n-4)/2$   |
| Простой обмен     | $C = (n^2-n)/2$<br>$M = 3(n-1)$ | $C = (n^2-n)/2$<br>$M = n^2/4 + 3(n-1)$ |
| Простой выбор     | $C = (n^2-n)/2$<br>$M = ?$      | $C = (n^2-n)/2$<br>$M = (n^2-n) * 1,5$  |

**N** – количество элементов,  
**M** – кол-во пересылок,  
**C** – кол-во сравнений

Чему будет равно количество пересылок



# Выбор метода сортировки

- При сортировке маленьких массивов (менее 100 элементов) лучше использовать метод «Всплывающего пузырька»;
- Если известно, что список уже почти отсортирован, то подойдет любой метод;



# Оценка алгоритма Шелла

Время выполнения пропорционально  $n^{1.2}$ , т. к. при каждом проходе используется небольшое число элементов или элементы массива уже находятся в относительном порядке, а упорядоченность увеличивается при каждом новом просмотре данных





# Оценка алгоритма быстрой сортировки

Если размер массива равен числу, являющемуся степенью двойки ( $N=2^g$ ), и при каждом разделении элемент  $X$  находится точно в середине массива, тогда при первом просмотре выполняется  $N$  сравнений и массив разделится на две части размерами  $N/2$ . Для каждой из этих частей  $N/2$  сравнений и т. д. Следовательно

$$C=N+2*(N/2)+4*(N/4)+...+N*(N/N).$$

Если  $N$  не является степенью двойки, то оценка будет иметь тот же порядок



Общее количество операций **Theta(n)**.

Количество шагов деления (глубина рекурсии) составляет приблизительно **log n**, если массив делится на более-менее равные части. Таким образом, общее быстроедействие: **O(n log n)**

Если каждый раз в качестве центрального элемента выбирается максимум или минимум входной последовательности, тогда быстроедействие **O(n<sup>2</sup>)**



- Метод неустойчив.
- Поведение довольно естественно, если учесть, что при частичной упорядоченности повышаются шансы разделения массива на более равные части
- Сортировка использует дополнительную память



## Итоги:

- Предпочтительным является метод прямого включения;
- Сортировка методом простого обмена является наихудшей;
- Быстрая сортировка превосходит все остальные методы сортировки;



# Контрольные вопросы

- ? Что такое «сортировка»?
- ? В чем заключается метод сортировки отбором?
- ? В чем заключается метод сортировки вставками?
- ? В чем заключается метод пузырьковой сортировки?
- ? В чем заключается метод быстрой сортировки?
- ? В чем заключается метод сортировки Шелла?



# Контрольные вопросы

- ? Какой алгоритм сортировки считается самым простым?
- ? Какой алгоритм сортировки считается самым эффективным?
- ? Сколько существует групп алгоритмов сортировки?
- ? По каким признакам характеризуются алгоритмы сортировки?
- ? Что нужно учитывать при выборе алгоритма сортировки?

