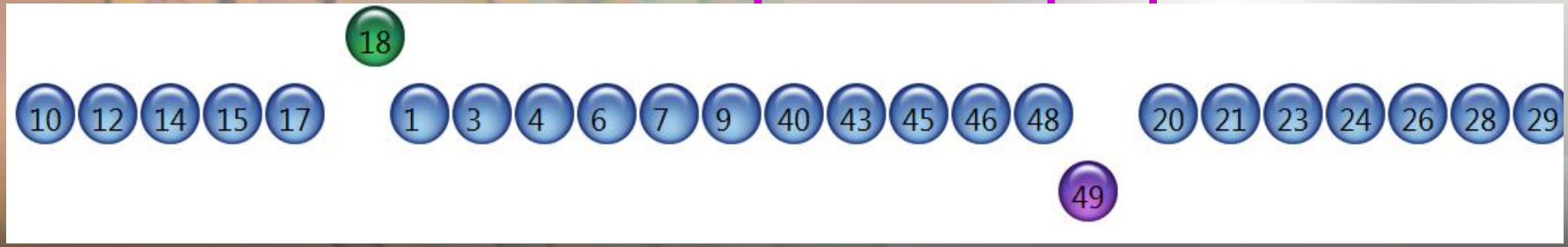
Структуры и алгоритмы обработки данных

Лекция 7

Методы улучшения алгоритмов сортировок



Алгоритм сортировки

-АЛГОРИТМ ДЛЯ УПОРЯДОЧЕНИЯ НЕКОТОРОГО МНОЖЕСТВА ЭЛЕМЕНТОВ ОБЫЧНО ПО ВОЗРАСТАНИЮ ИЛИ УБЫВАНИЮ

Оценка алгоритма сортировки

- ***Время сортировки*** *– основной параметр, характеризующий быстродействие алгоритма*
- ***Память*** *– ряд алгоритмов сортировки требуют выделения дополнительной памяти под временное хранение данных*
- ***Устойчивость*** *– сортировка не меняет взаимного расположения равных элементов*
- ***Естественность поведения*** *– эффективность метода при обработке уже отсортированных, или частично отсортированных данных*

Классификация алгоритмов сортировок

Сортировка

Внутренняя сортировка
или
сортировка массивов

Внешняя сортировка
или
сортировка файлов

Методы внутренней сортировки

Прямые методы

- *вставкой (включением)*
- *выбором (выделением)*
- *обменом («пузырьковая»)*

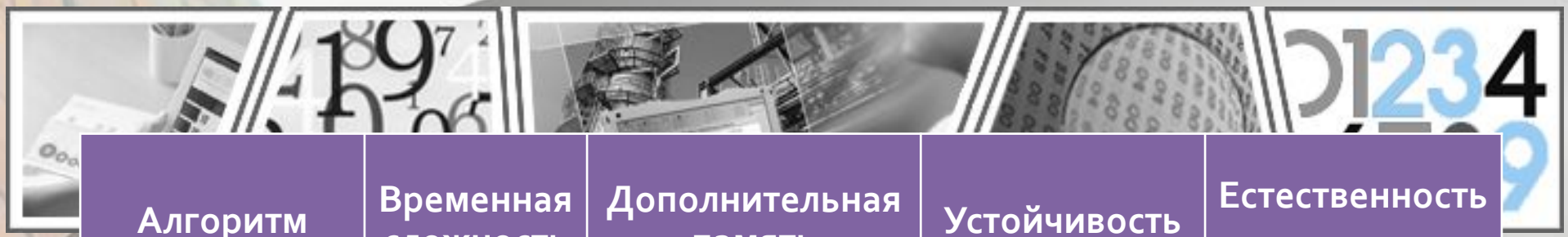
Улучшенные методы

- *быстрая*
- *Шелла*

До

После

Анализ элементарных алгоритмов сортировок



Алгоритм	Временная сложность	Дополнительная память	Устойчивость	Естественность поведения
BubbleSort - метод обмена (пузырьковый)	$\Theta(n^2)$	не требуется	да	нет
SelectSort – метод выбора	$\Theta(n^2)$	не требуется	нет	нет
InsertSort - метод вставок	$\Omega(n)$, $O(n^2)$	не требуется	да	да

Сортировка

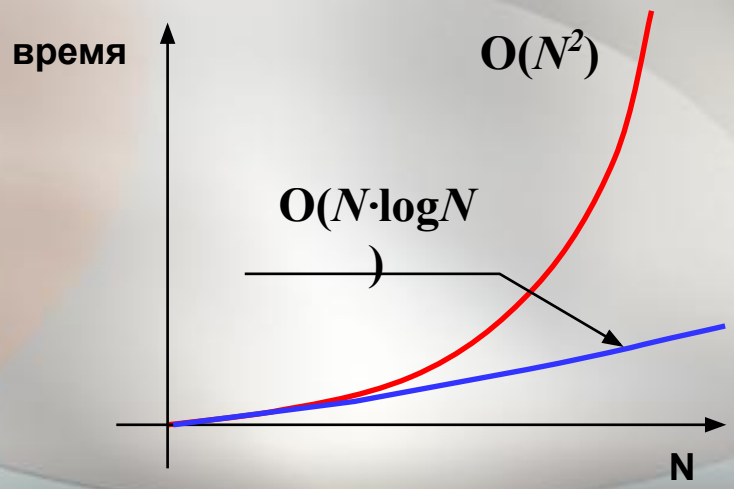
Алгоритмы:

- простые и понятные, но неэффективные для больших массивов
  - **метод пузырька**
  - **метод выбора**
  - **метод прямой вставки**

сложность $O(N^2)$

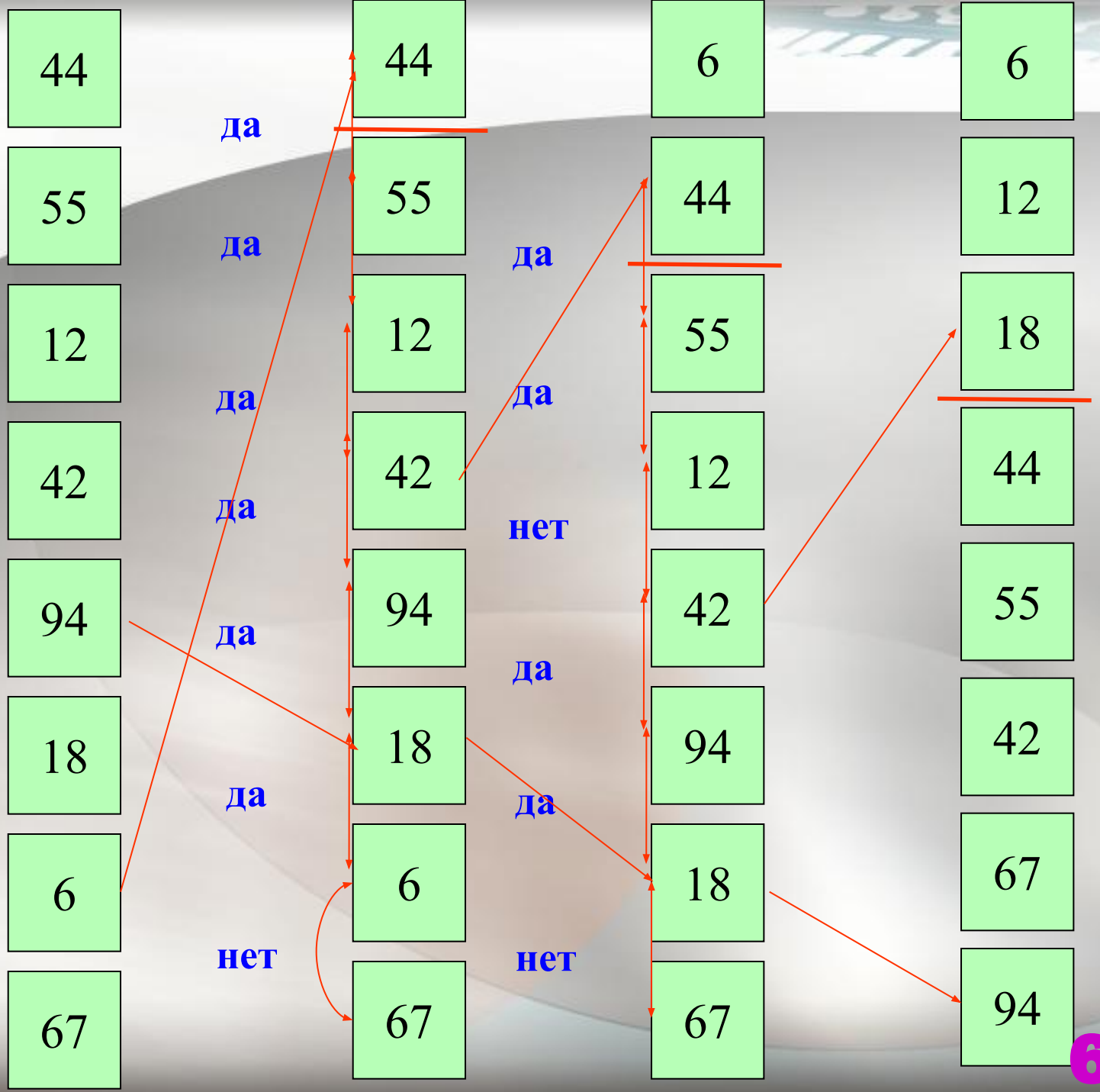
- сложные, но эффективные
  - **«быстрая сортировка» (*Quick Sort*)**
  - **сортировка «кучей» (*Heap Sort*)**
  - **сортировка слиянием**
  - **пирамидальная сортировка**

сложность $O(N \cdot \log N)$



НЕ СУЩЕСТВУЕТ УНИВЕРСАЛЬНОГО, НАИБОЛЕЕ ЭФФЕКТИВНОГО СПОСОБА СОРТИРОВКИ

Сортировка обменом / метод пузырька



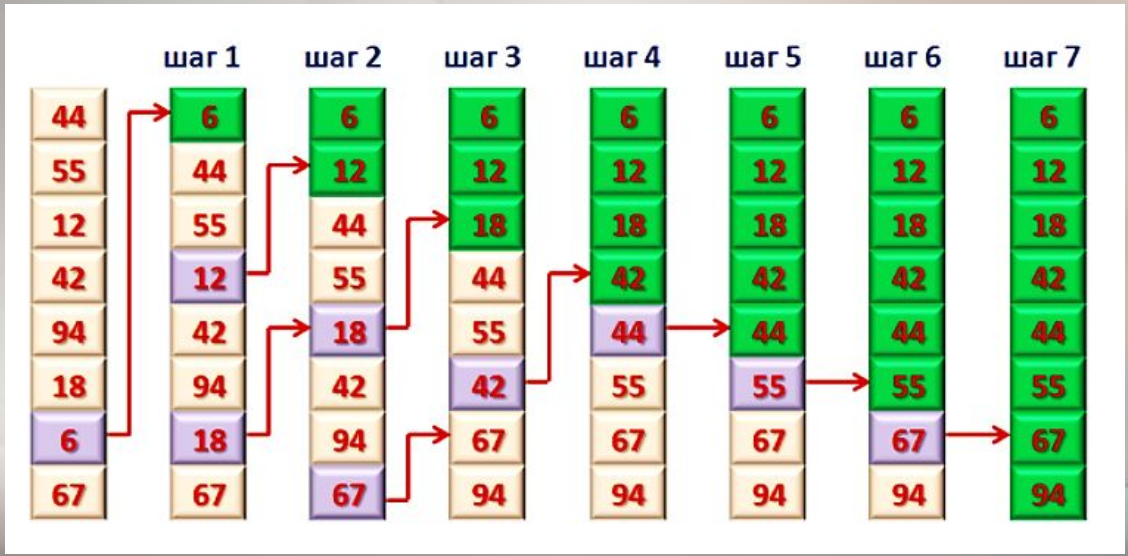
Методы улучшения сортировки обменом

1

На одном из проходов не произошло ни одного обмена

Все пары расположены в правильном порядке, массив уже отсортирован

Продолжать процесс не имеет смысла



Запомнить, производился ли на данном проходе какой-либо обмен!
Если нет – алгоритм заканчивает работу

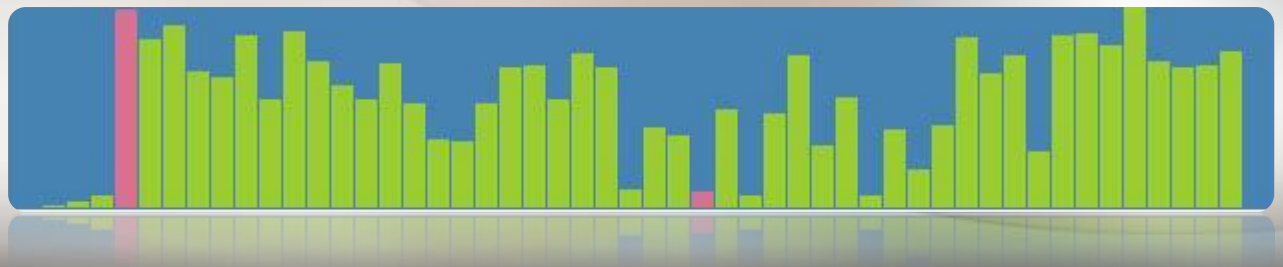
Методы улучшения сортировки обменом

2

Все пары соседних элементов с индексами, меньшими **k**, уже расположены в нужном порядке

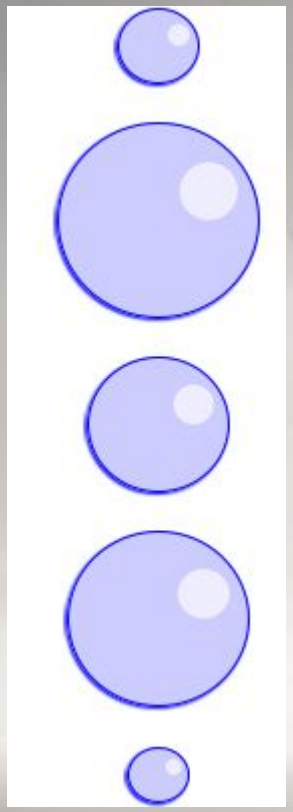
Дальнейшие проходы можно заканчивать на индексе **k**, вместо того чтобы двигаться до установленной заранее верхней границы **i**

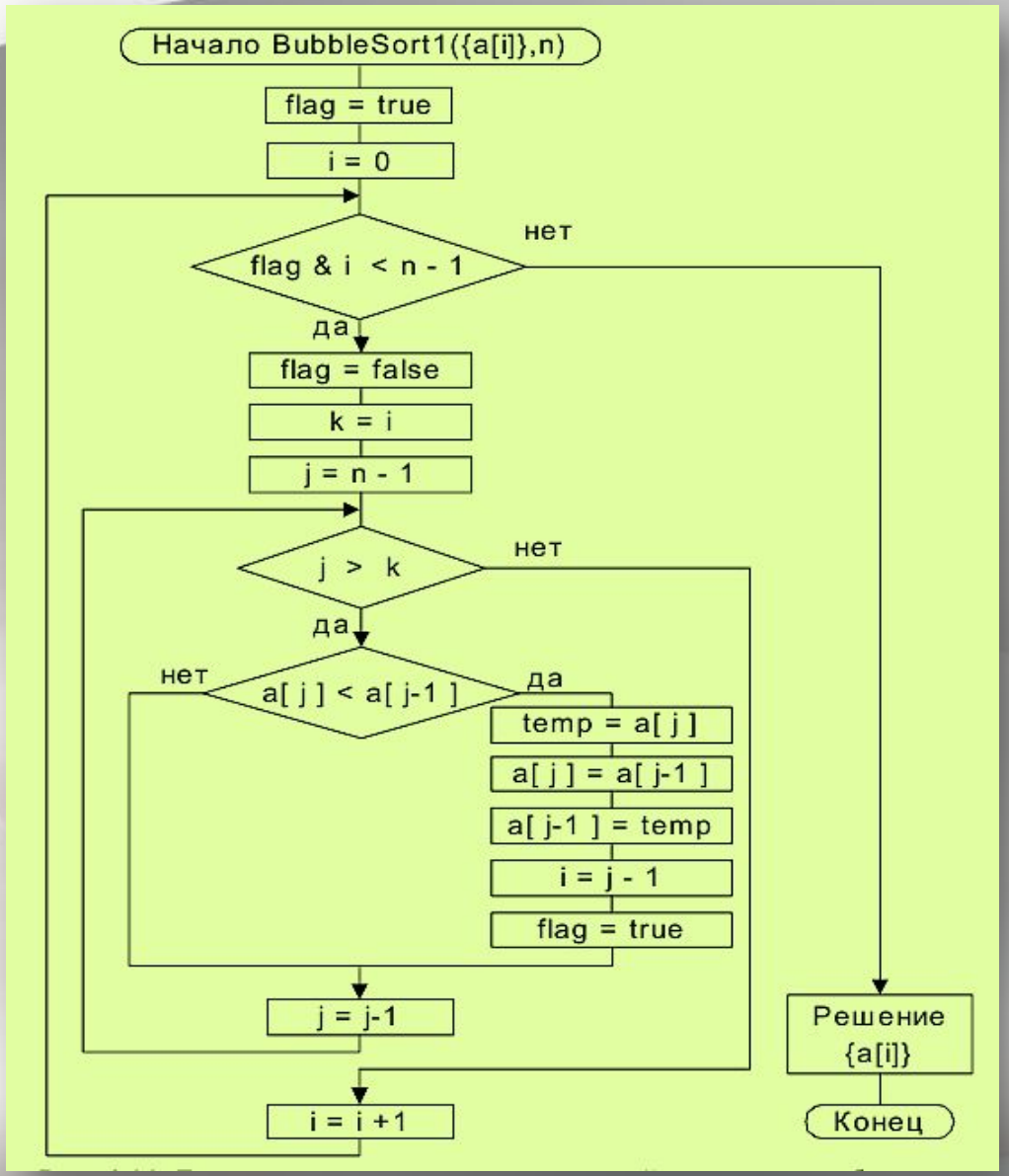
Ввести логическую переменную – признак наличия хотя бы одного обмена в проходе



Методы улучшения сортировки обменом

1+2





Методы улучшения сортировки обменом

3

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

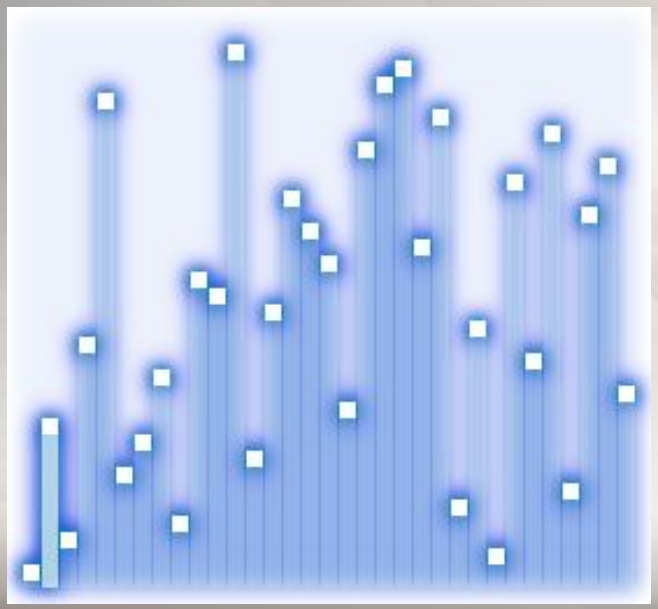
Сортировка последовательности 6 1 2 3 4 5 потребует 5 проходов

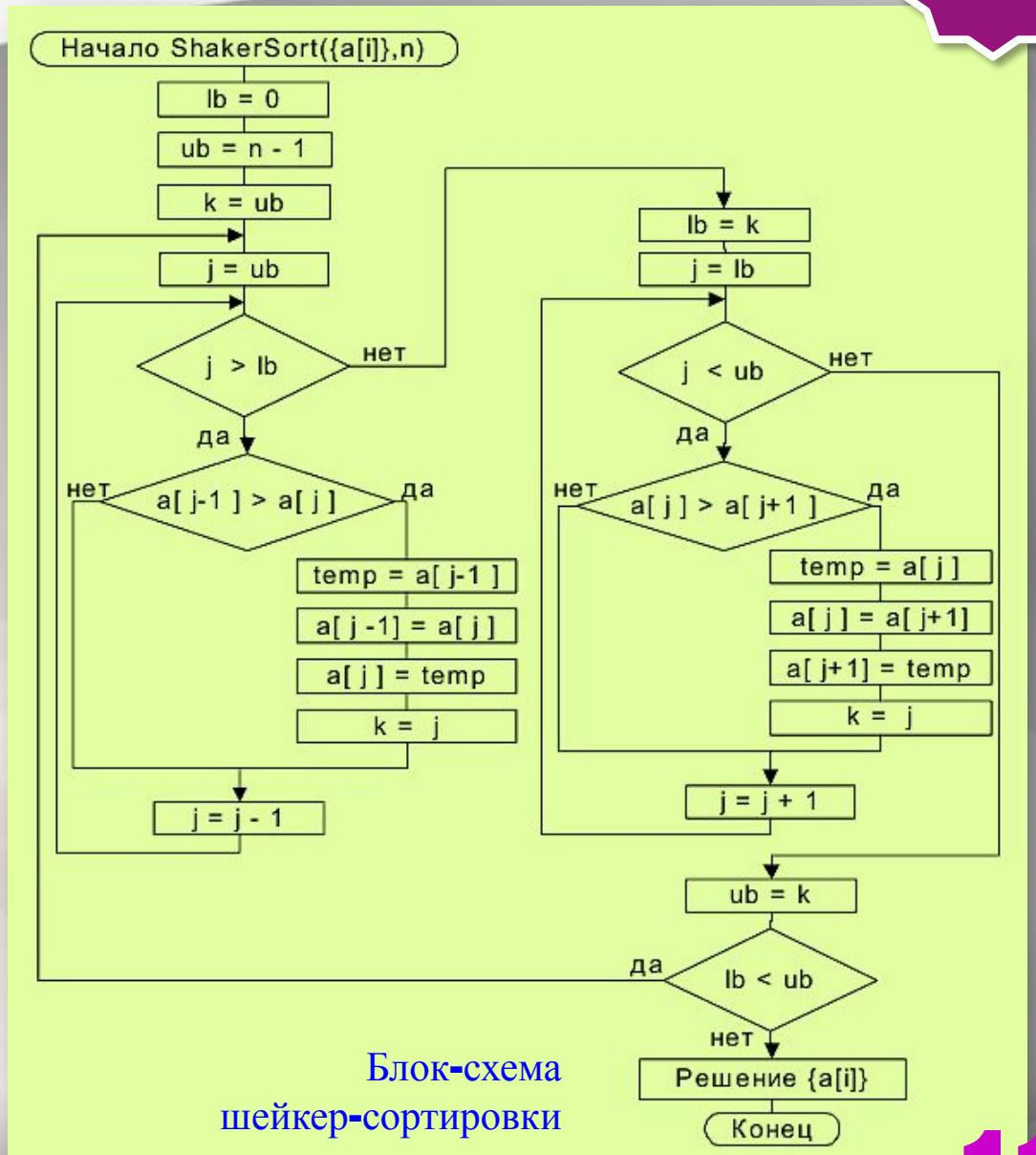
Можно менять направление следующих один за другим проходов - шейкер-сортировка

Методы улучшения сортировки обменом

Параметры блок-схемы

- **сортируемая последовательность *{a[i]}***
- **индекс ее первого элемента – *lb* (*lower bound* - нижняя граница)**
- **индекс ее последнего элемента - *ub* (*upper bound* - верхняя граница)**





Блок-схема шейкер-сортировки

Методы улучшения сортировки обменом

Алгоритм	Временная сложность	Дополнительная память	Устойчивость	Естественность поведения
BubbleSort - метод обмена (пузырьковый)	$\Theta(n^2)$	не требуется	да	нет
BubbleSort - улучшенные методы обмена	$\Theta(n^2)$	не требуется	Шейкер-сортировка теряет устойчивость	да

В лучшем случае, когда массив уже упорядочен, потребуется всего один проход и ***n-1*** сравнение, что составляет ***O(n).*** Перестановки в этом случае не выполняются

В худшем случае эти виды улучшенной сортировки
не отличаются от исходного алгоритма

Быстрая сортировка

«Разделяй и властвуй"

В 1962 году Ч.А.Р. Хоар (Charles Antony Richard Hoare) предложил *улучшенный вариант обменной сортировки*

***Основная идея улучшения* - обменивать не рядом стоящие элементы массива, а расположенные как можно дальше друг от друга**

СУТЬ СОРТИРОВКИ:

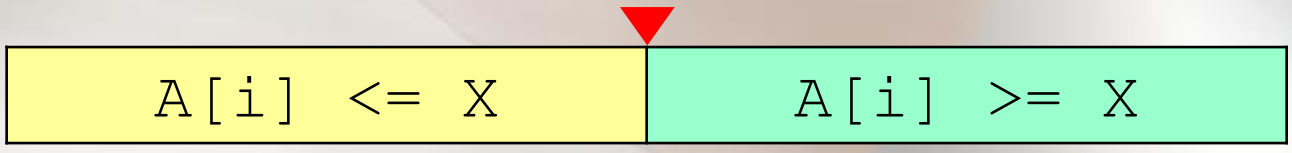
- **Выбирается некоторое значение (x)- барьерный элемент;**
- **Просматриваем массив, двигаясь слева направо, пока не найдется элемент, больший x**
- **Затем просматриваем его справа налево, пока не найдется элемент, меньший x**

Быстрая сортировка

«Разделяй и властвуй"

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

«Разделяй и властвуй"

ПРИМЕР:

1. Выберем один из элементов массива - *барьер*

2. Элементы массива просматриваем с двух концов. При просмотре слева направо (начиная с первого) выбираем элемент, *не меньший* чем барьерный. А во время просмотра справа налево (начиная с последнего) выбирается элемент *не больший* чем барьерный

3. Найденная пара элементов меняется местами, после чего просмотры возобновляются

4. Завершается проход по массиву после того, как индекс просмотра слева, станет больше чем индекс просмотра справа

Пусть a =42 — барьерный элемент

i — индекс элементов при проходе направо: i=1, 2, …

j — индекс элемента при проходе налево: j=N, N-1, N-2, …

Проход слева направо: i=1, x_1 (=44) > a (=42).

Проход справа налево: j=7, x_7 (=6) < a (=42).

Проход слева направо: i=2, x_2 (=55) > a (=42).

Проход справа налево: j=6, x_6 (=18) < a (=42).

Проход слева направо: i=4, Достигнут барьер

Проход справа налево: j=4, Достигнут барьер

Как следует поступить далее?

Можно рекурсивно отсортировать обе части массива

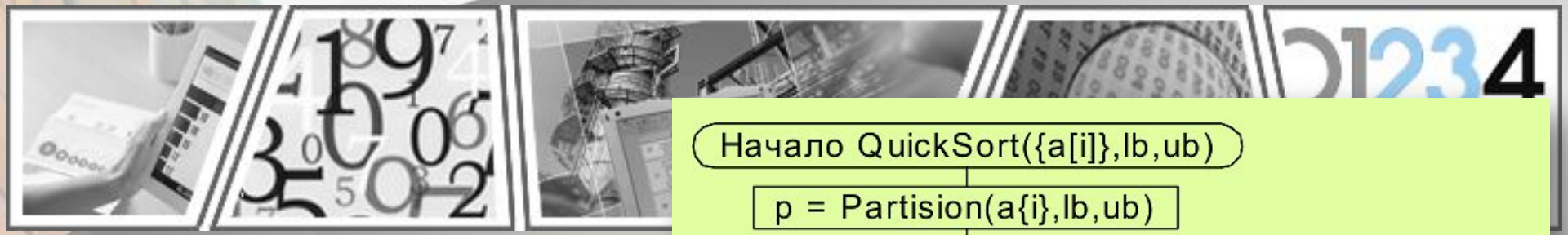
Быстрая сортировка

по возрастанию

Массив отсортирован по возрастанию

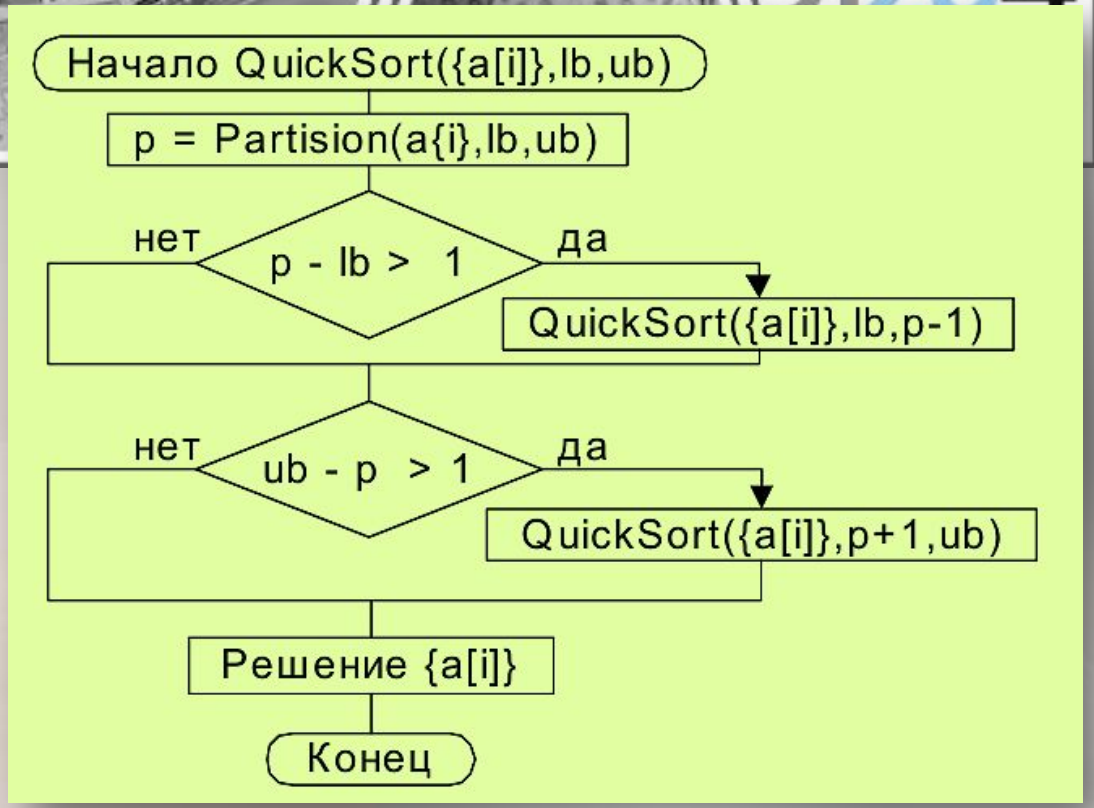
Быстрая сортировка

«Разделяй и властвуй"



Параметры блок-схемы

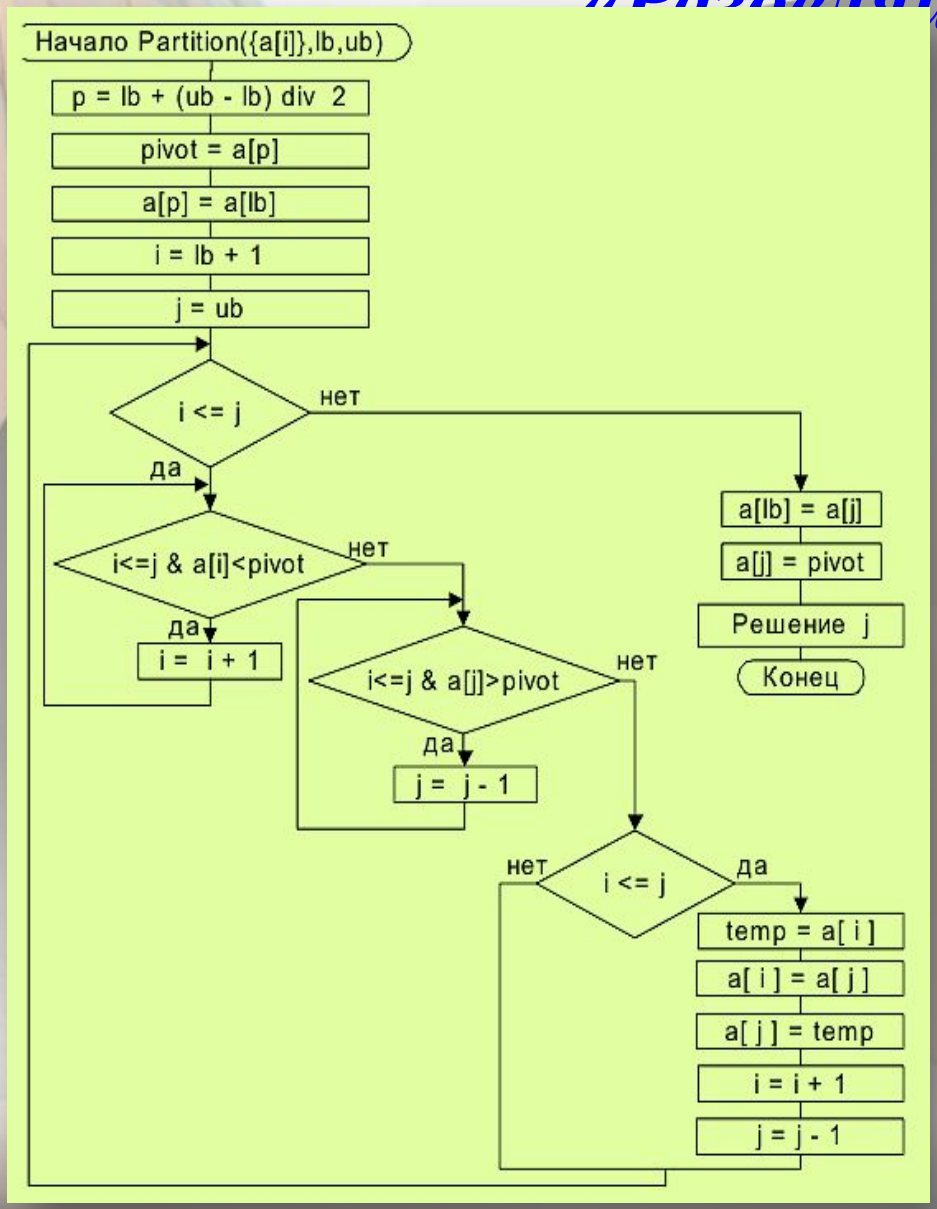
- **сортируемая последовательность *{a[i]}***
- **индекс ее первого элемента – *lb* (*lower bound* - нижняя граница)**
- **индекс ее последнего элемента *ub* (*upper bound* - верхняя граница)**



Блок-схема алгоритма быстрой сортировки

Быстрая сортировка

«Разделяй и властвуй"



Параметры блок-схемы

- **сортируемая последовательность *{a[i]}***
- **индекс ее первого элемента – *lb* (*lower bound* - нижняя граница)**
- **индекс ее последнего элемента - *ub* (*upper bound* - верхняя граница)**
- ***pivot* – барьерный элемент**

Блок-схема процедуры разделения массива

Быстрая сортировка

«Разделяй и властвуй"

Довольно сложный анализ эффективности алгоритма быстрой сортировки Ч. Хоара показал:

- **в *оптимальном* случае общее количество сравнений равно $C = N*\log_2 N$, а общее количество пересылок (присваиваний) $M = N*\log_2 N/6$**

- **Оптимальный вариант, при котором достигается самая высокая эффективность и минимальная сложность сортировки, обеспечивается выбором в качестве барьера так называемого *медианного* элемента массива**

Быстрая сортировка

«Разделяй и властвуй"

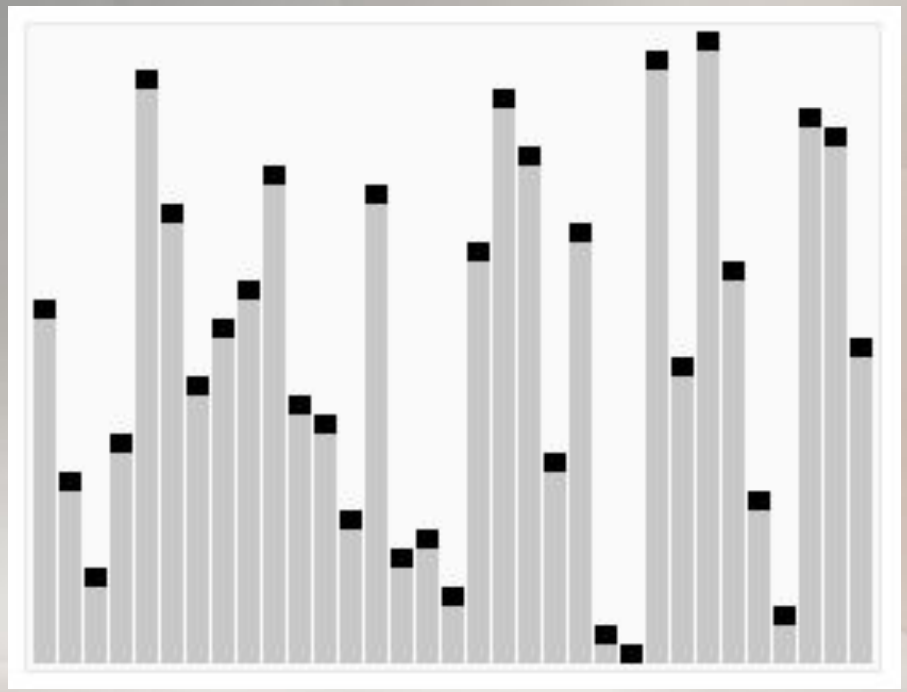
Медианой массива (*медианным элементом*) считается такой его элемент, который не меньше одной половины элементов массива и при этом не больше другой половины (*независимо от их взаимного положения, важно лишь общее количество элементов*)

Так для массива состоящего из 7 элементов, нужно выбрать такой элемент, который окажется *не больше трёх любых элементов* и при этом *не меньше трёх других его элементов*

Например, для массива {16, 12, 90, 84, 18, 67, 10} медианой является элемент $x_5 = 18$, так как он больше $x_1 = 16$, $x_2 = 12$ и $x_7 = 10$, и при этом меньше чем $x_3 = 90$, $x_4 = 84$ и $x_6 = 67$

Быстрая сортировка

«Разделяй и властвуй"



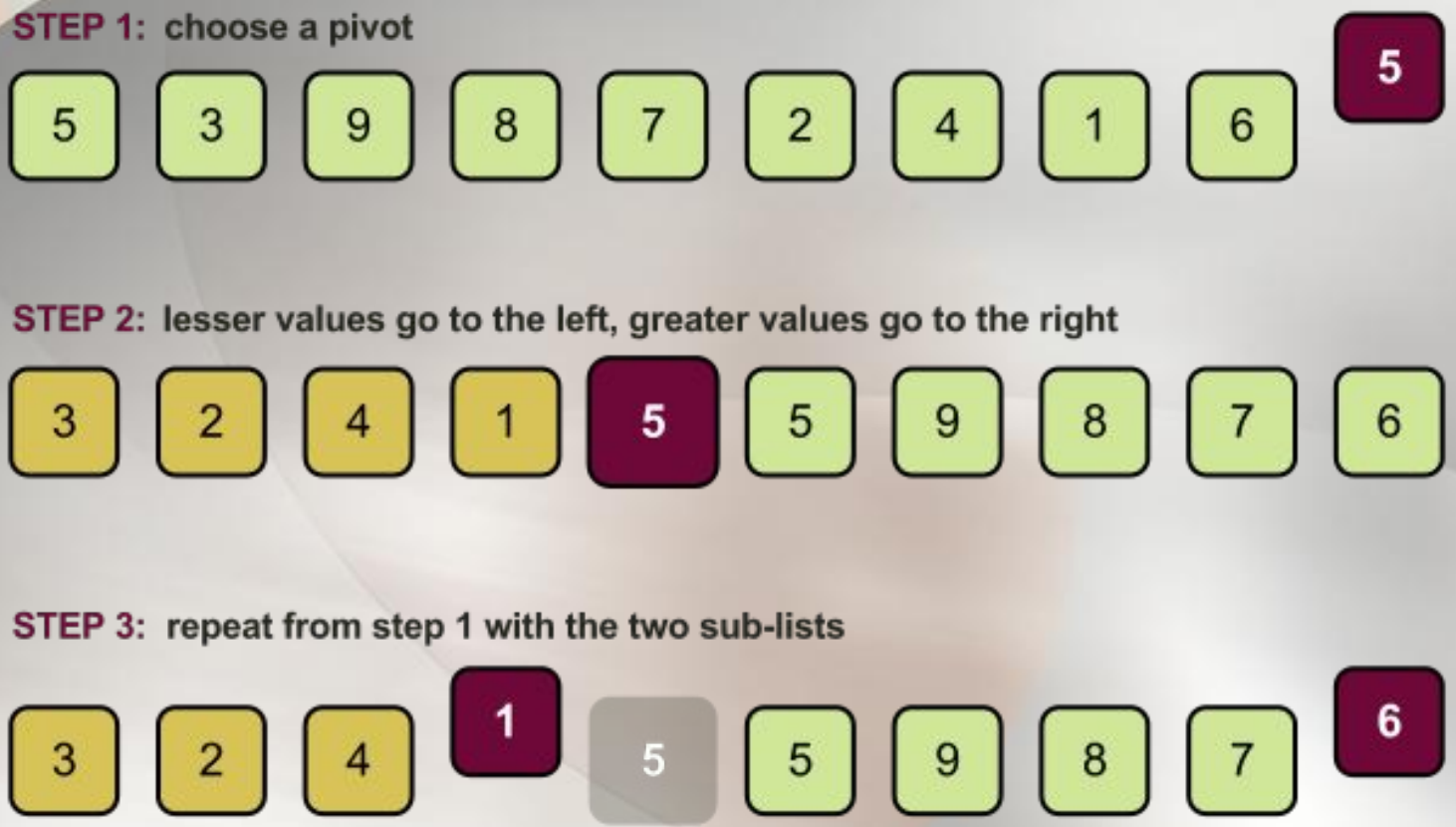
Выбирать *медианный элемент* довольно сложно, во всяком случае такой *выбор* представляет собой *самостоятельную задачу*

Удивительное свойство сортировки Хоара -

её средняя производительность при *случайном* выборе
(в том числе при выборе среднего) барьерного элемента отличается от оптимального всего лишь коэффициентом $2*\log_{10}2$

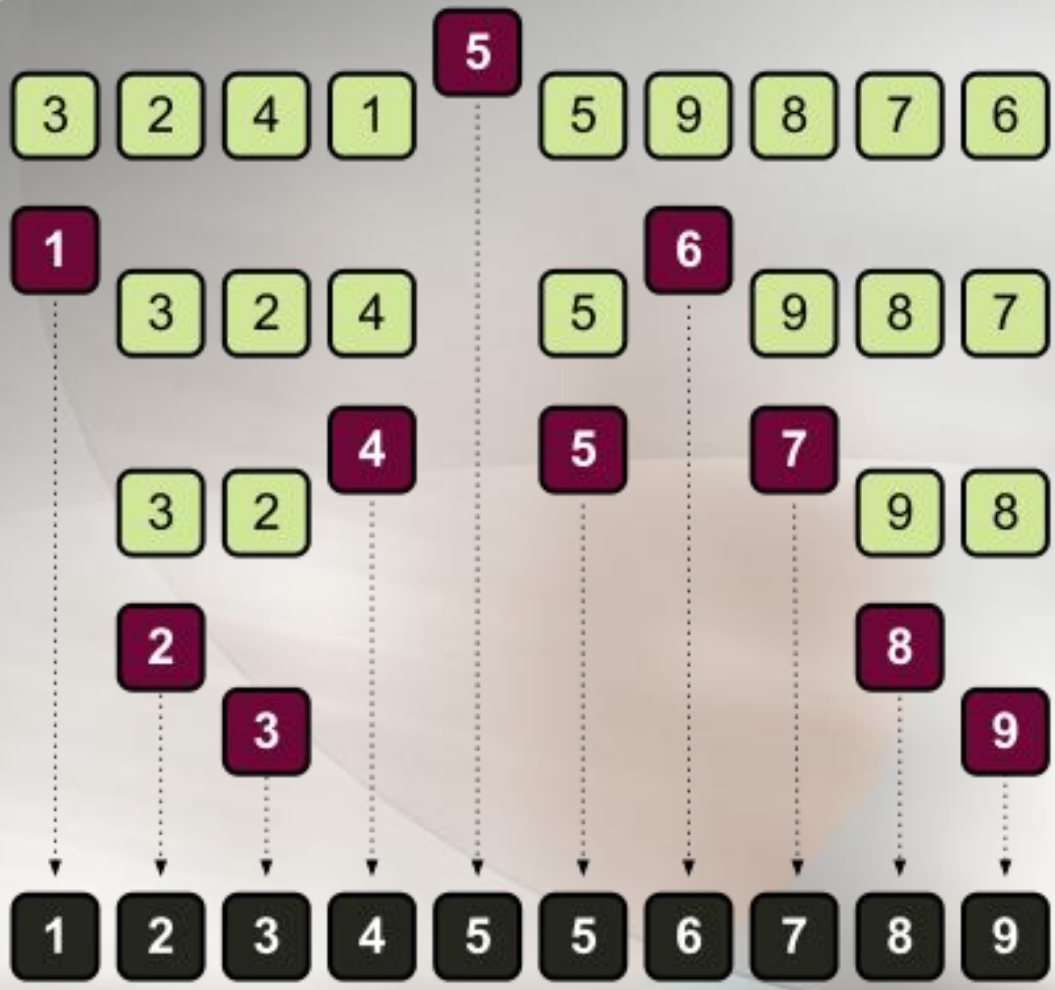
Быстрая сортировка

«Разделяй и властвуй"



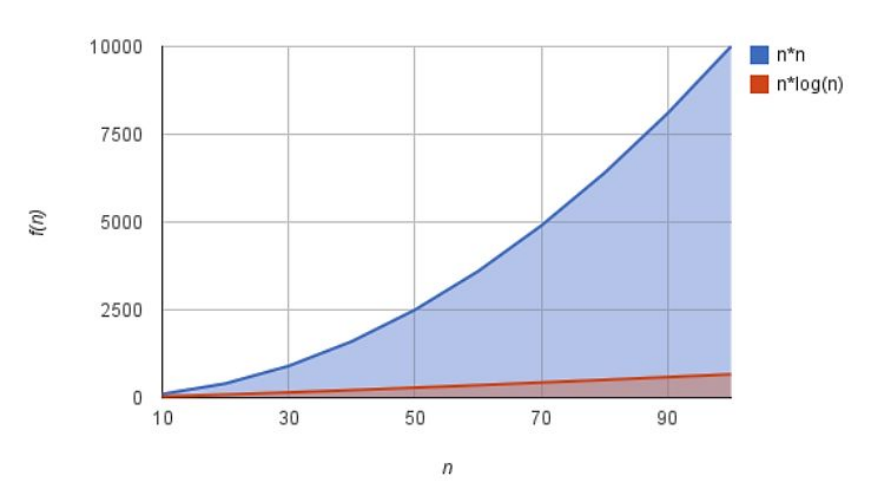
Быстрая сортировка

«Разделяй и властвуй"



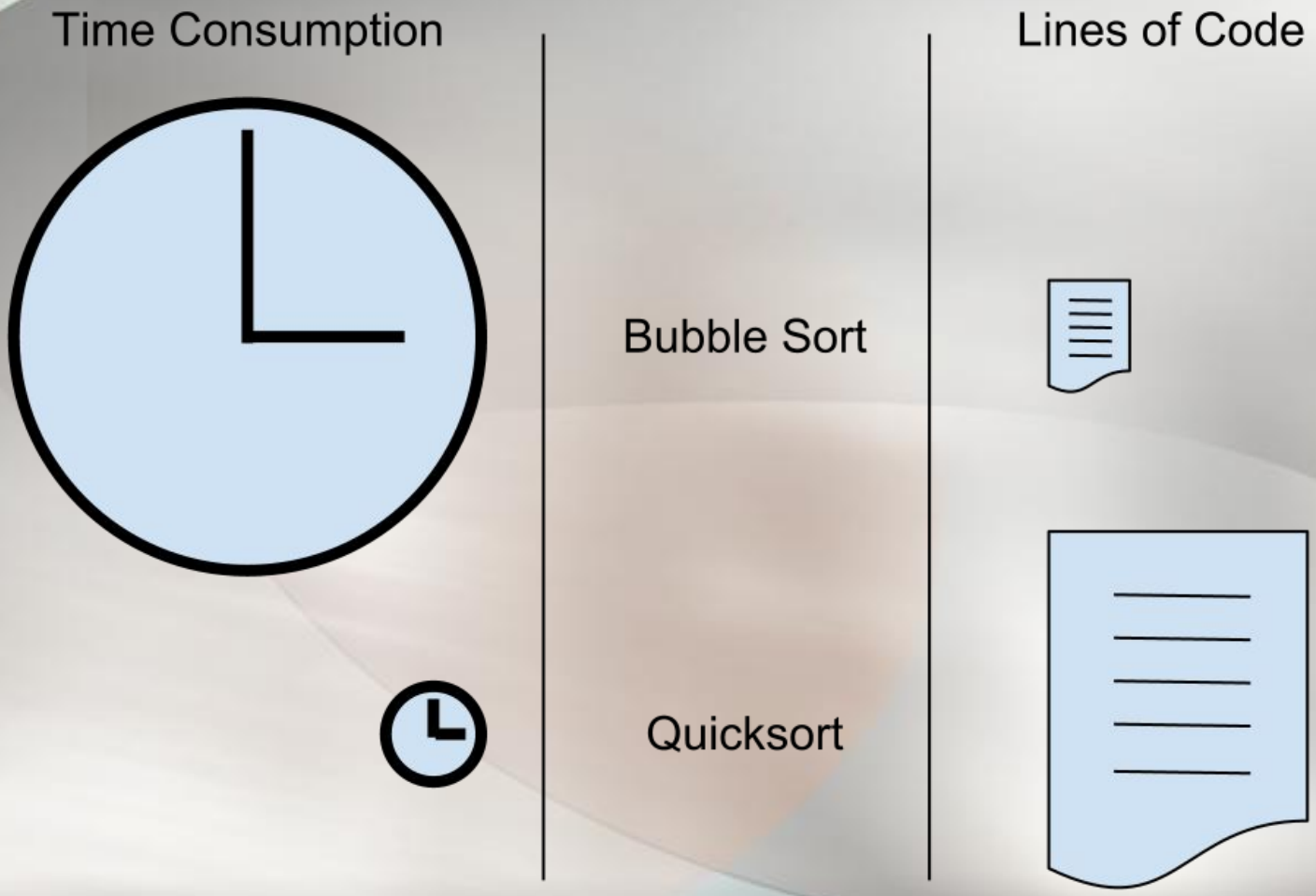
Быстрая сортировка

Алгоритм	Временная сложность	Дополнительная память	Устойчивость	Естественность поведения
BubbleSort - метод обмена (пузырьковый)	$\Theta(n^2)$	не требуется	да	нет
Быстрая сортировка	$O(n \log n)$	использует дополнительную память (рекурсия)	сортировка теряет устойчивость	да



Быстрая сортировка

«Разделяй и властвуй"

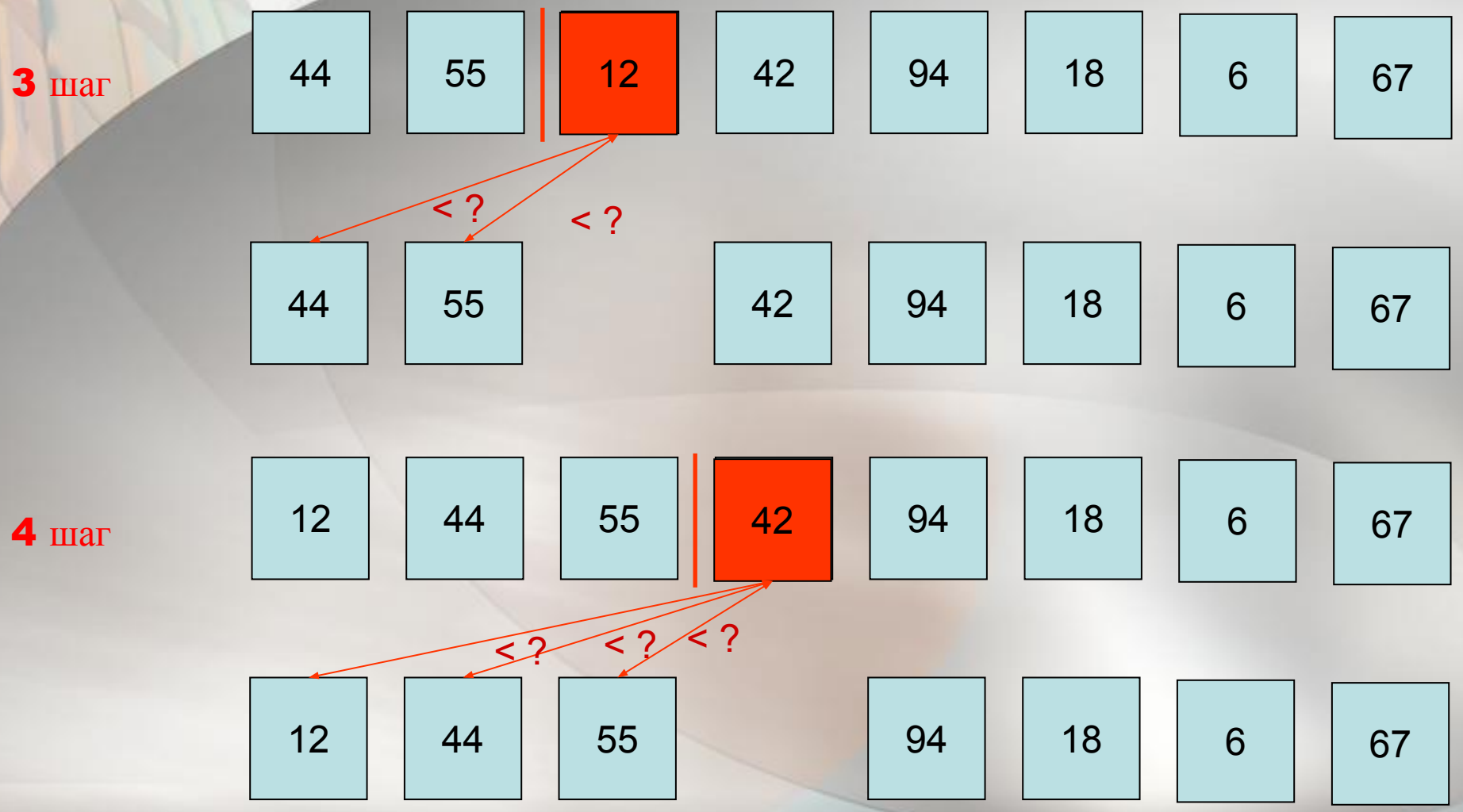


Улучшение сортировки вставками

Сортировка прямыми вставками

Улучшение сортировки вставками

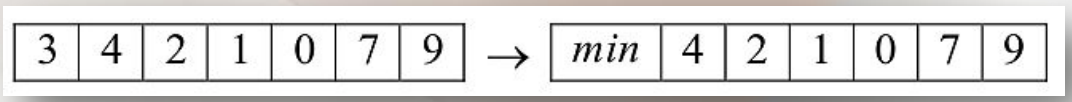
Сортировка прямыми вставками



Улучшение сортировки вставками

Алгоритм сортировки вставками можно слегка улучшить

- На каждом шаге внутреннего цикла проверяются 2 условия
- Можно объединить их в одно, поставив в начало массива специальный сторожевой элемент
- Он должен быть заведомо меньше всех остальных элементов массива



Вставка сторожевого элемента

Улучшение сортировки вставками

Тогда при $j = 0$ будет заведомо верно $a[0] \le x$

Цикл остановится на нулевом элементе, что и было целью условия $j \ge 0$

Сортировка будет происходить правильным образом, а во внутреннем цикле станет на одно сравнение меньше

Сравнение производилось n^2 раз, это – реальное преимущество

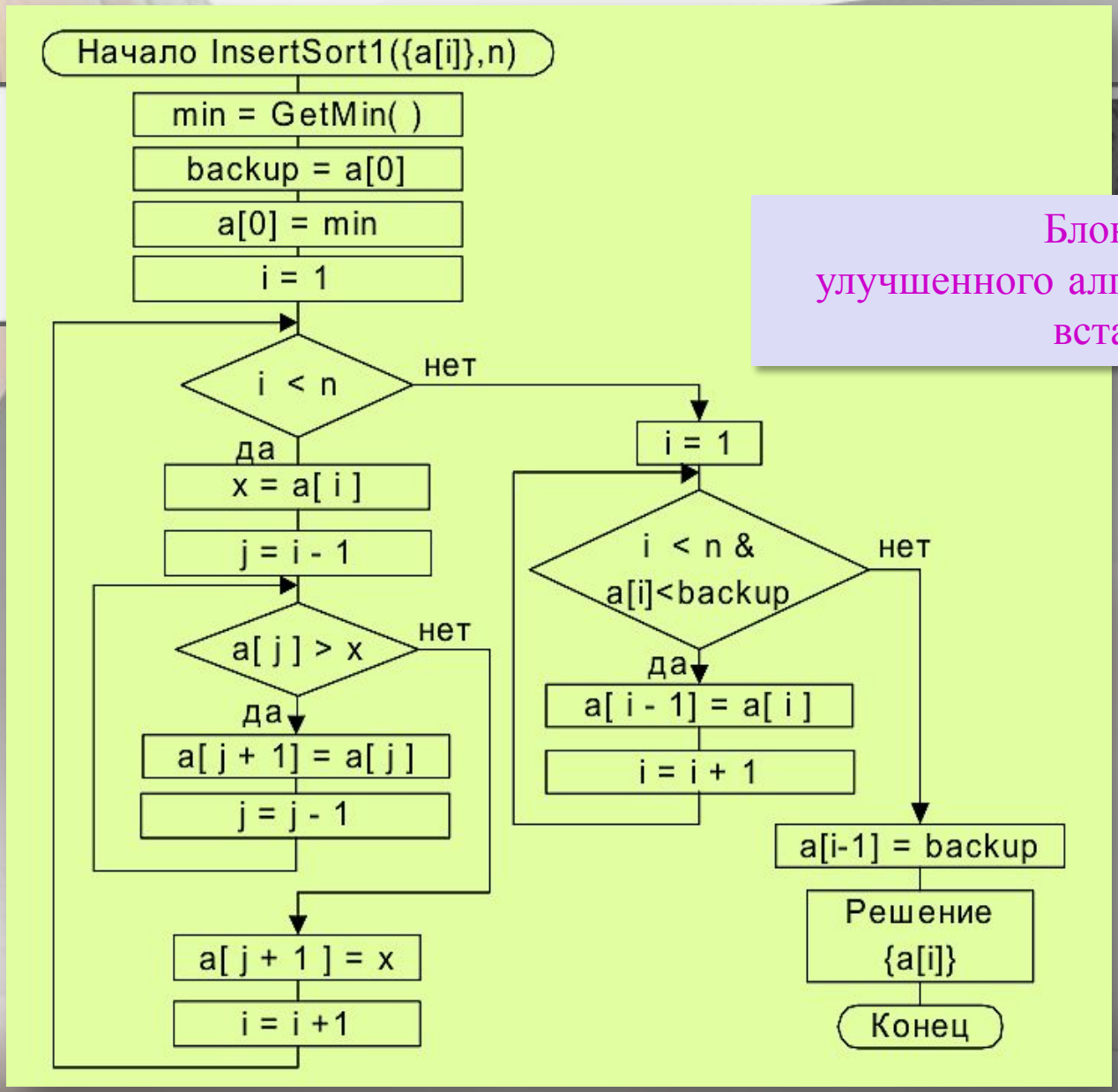
Отсортированный массив будет не полон, так как из него исчезло первое число

Для окончания сортировки это число следует вернуть назад, а затем вставить в отсортированную последовательность a[1]…a[n]

min	0	1	2	4	7	9	→	3	0	1	2	4	7	9	→	0	1	2	3	4	7	9

Замена сторожевого элемента на $a[0]$ и досортировка массива

Улучшение сортировки вставками



Блок-схема
улучшенного алгоритма сортировки
вставками

Функция *GetMin( )* - возвращает элемент, заведомо меньший всех возможных элементов массива, определяется пользователем

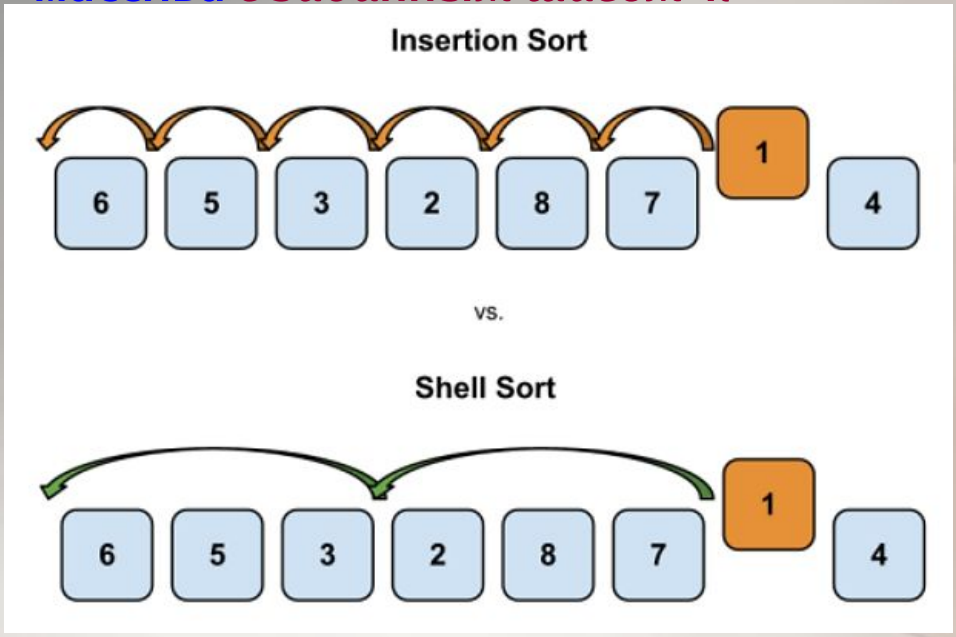
Метод является устойчивым и естественным

Сортировка методом Шелла

Сортировка включениями с уменьшающимся расстоянием

Сортировка Шелла была названа в честь ее изобретателя – Дональда Шелла, который опубликовал этот алгоритм в 1959 году

Основная идея этой сортировки состоит в выполнении нескольких *предварительных* проходов, на каждом из которых методом прямых вставок сортируются некоторые *подпоследовательности* элементов массива *с заданным шагом k*



Такая модификация метода сортировки позволяет быстро переставлять далекие неупорядоченные пары значений (сортировка таких пар обычно требует большого количества перестановок, если используется сравнение только соседних элементов)

В сортировке Шелла элементы каждой из подпоследовательностей упорядочиваются независимо от всех остальных подпоследовательностей

Сортировка методом Шелла

Сортировка включениями с уменьшающимся расстоянием

Д. Шелл предложил выполнить несколько таких проходов с *разными* шагами. Причем, **на последнем проходе шаг обязательно должен быть равным единице**

То есть на последнем шаге необходимо выполнить самую обычную сортировку прямыми вставками

В первоначально предложенном Д. Шеллом варианте сортировка выполнялась за четыре прохода с шагами 9, 5, 3 и 1

До настоящего времени неизвестно, сколько предварительных проходов нужно сделать для получения наилучших результатов и какие шаги должны быть для этого выбраны

Опытным путем подобраны следующие рекомендуемые шаги по проходам:

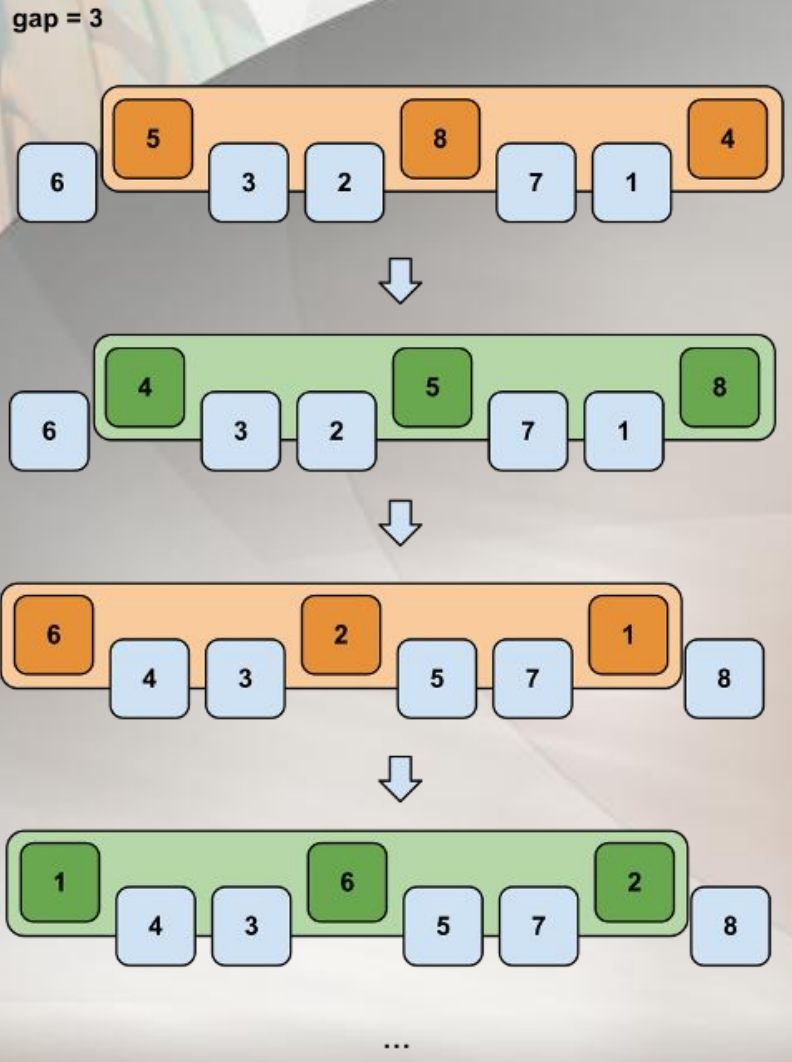
1, 4, 13, 40, 121, 364, 1093, 3280, 9841 . . .
1, 8, 23, 77, 281, 1073, 4193, 16577 . . .

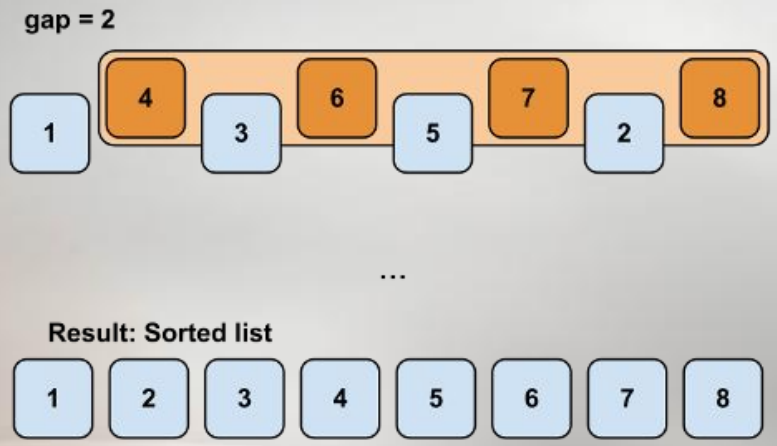
1, 3 5, 9, 37, …

1, 3, 7, 15, 31, …

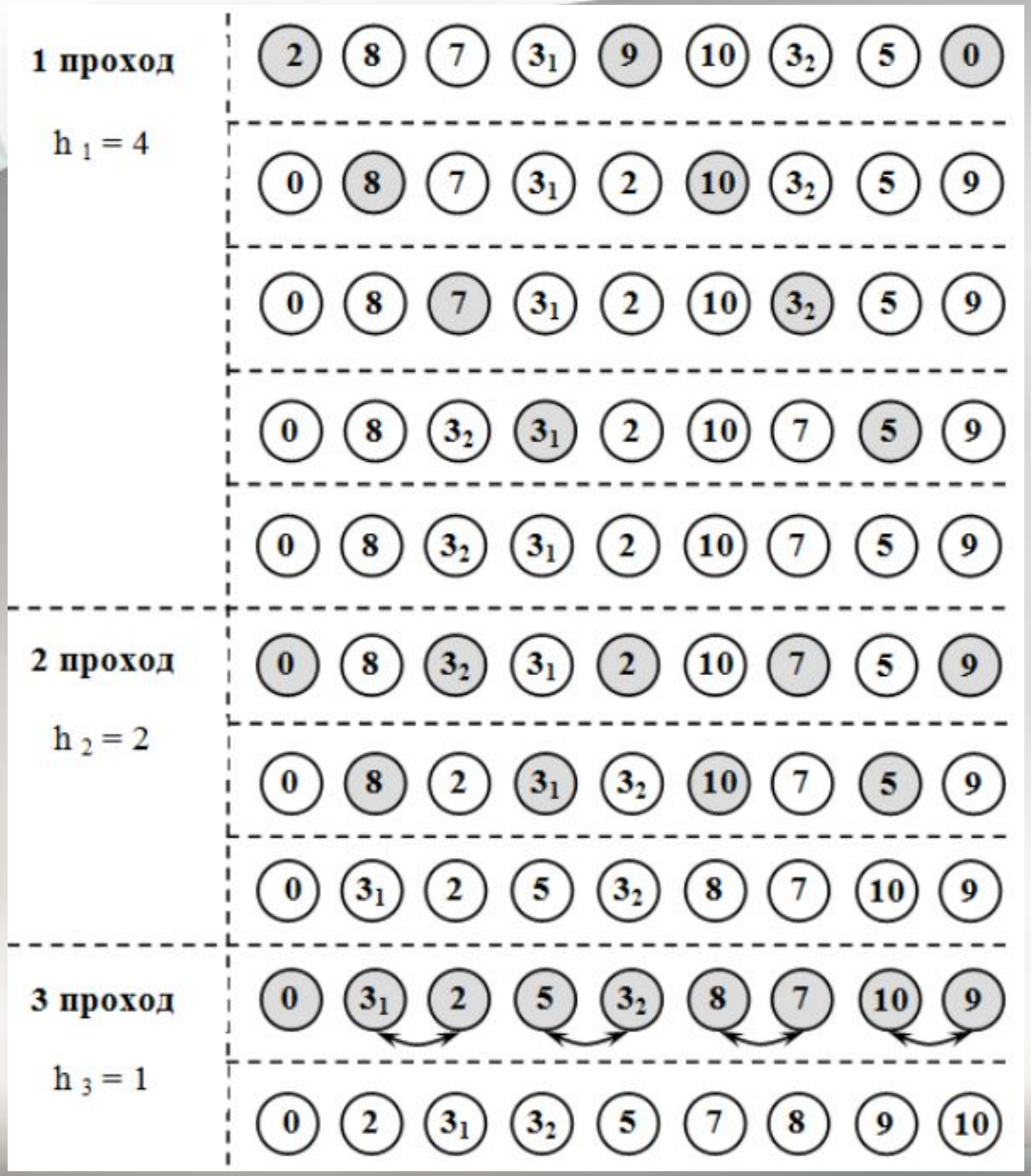
Сортировка методом Шелла

Пример 1





Сортировка методом Шелла

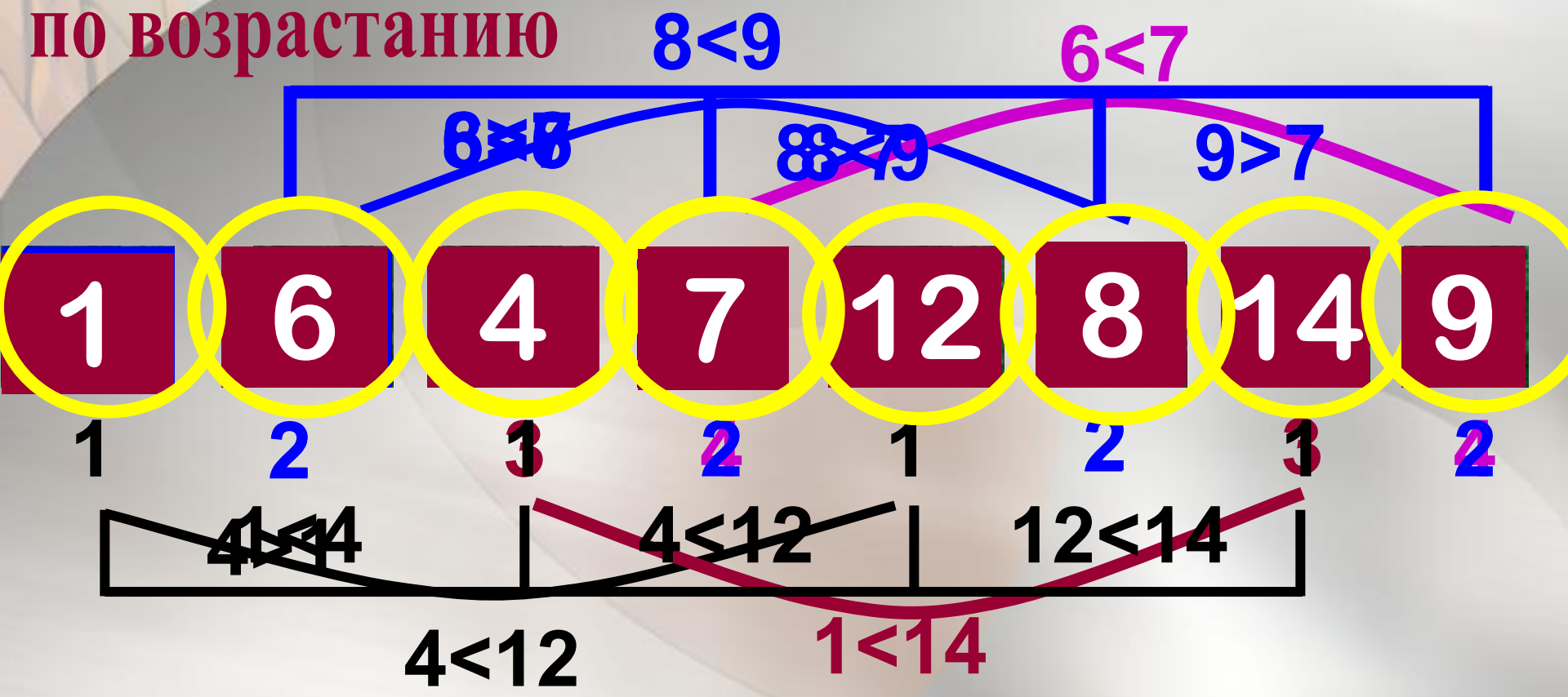


Пример 2

Сортировка методом Шелла

Пример 3

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



Сортировка методом Шелла

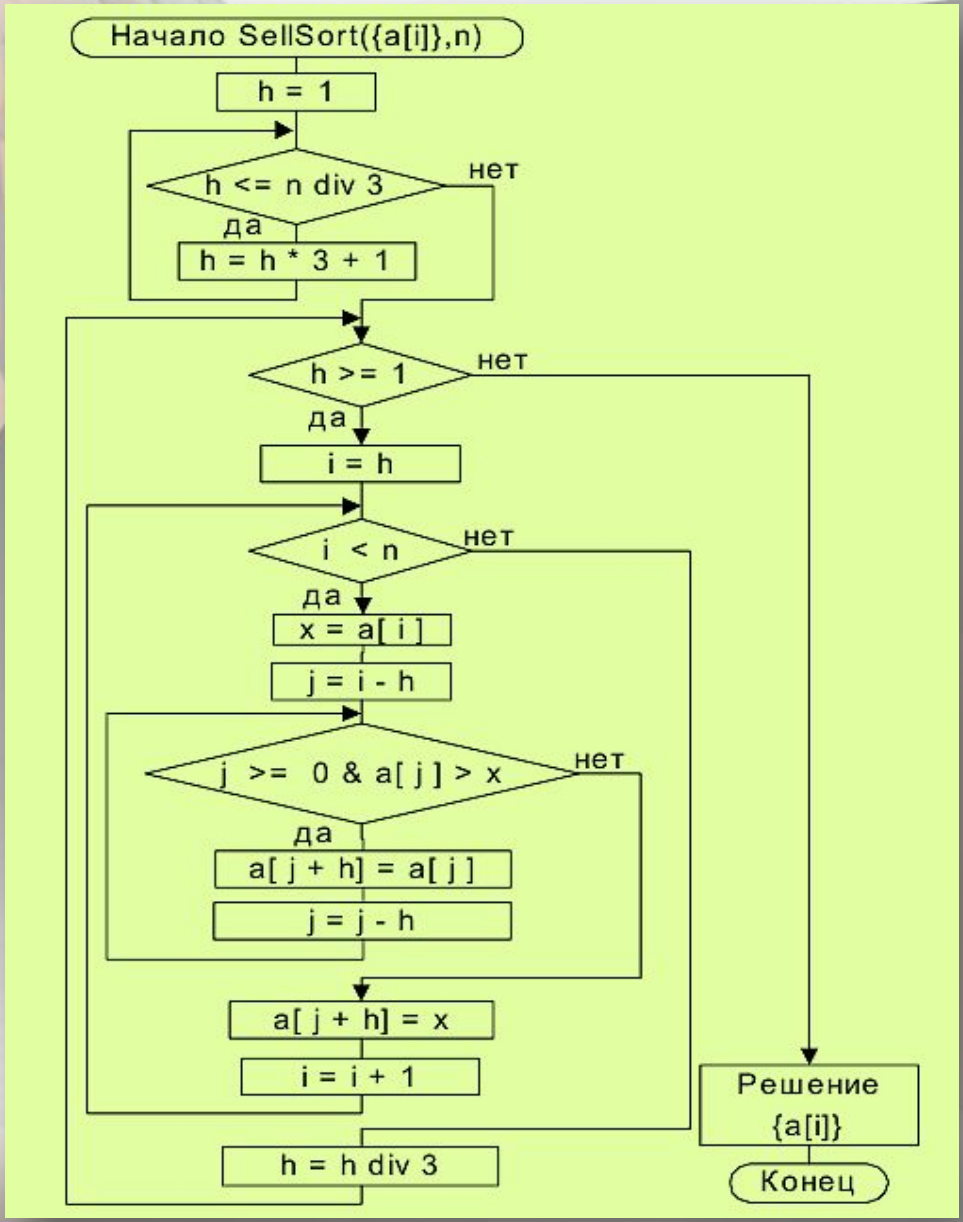
Пример 3

3 шаг. 1 группа из 8-ми элементов

по возрастанию

Массив отсортирован по возрастанию

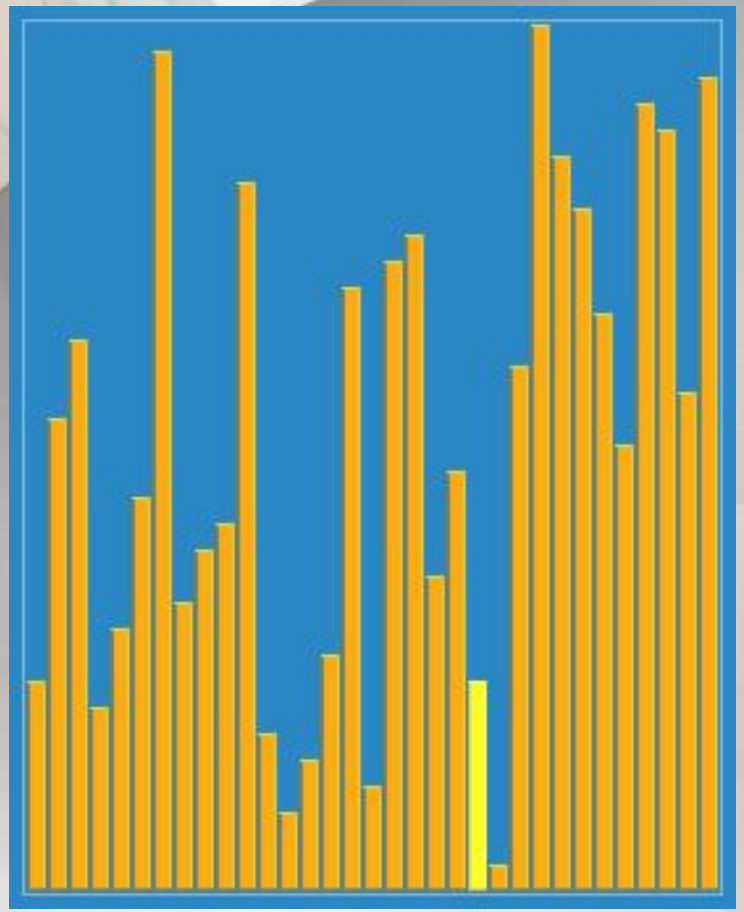
Сортировка методом Шелла

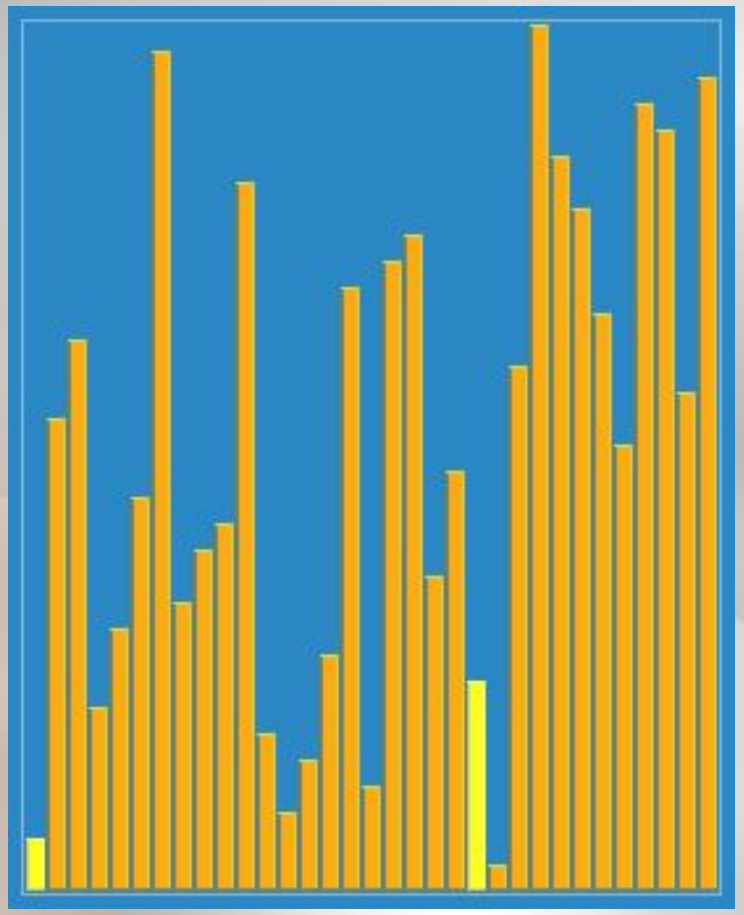


Блок-схема алгоритма сортировки Шелла
с выбором шагов, предложенных Кнутом:

..., 121, 40, 13, 4, 1

Сортировка методом Шелла

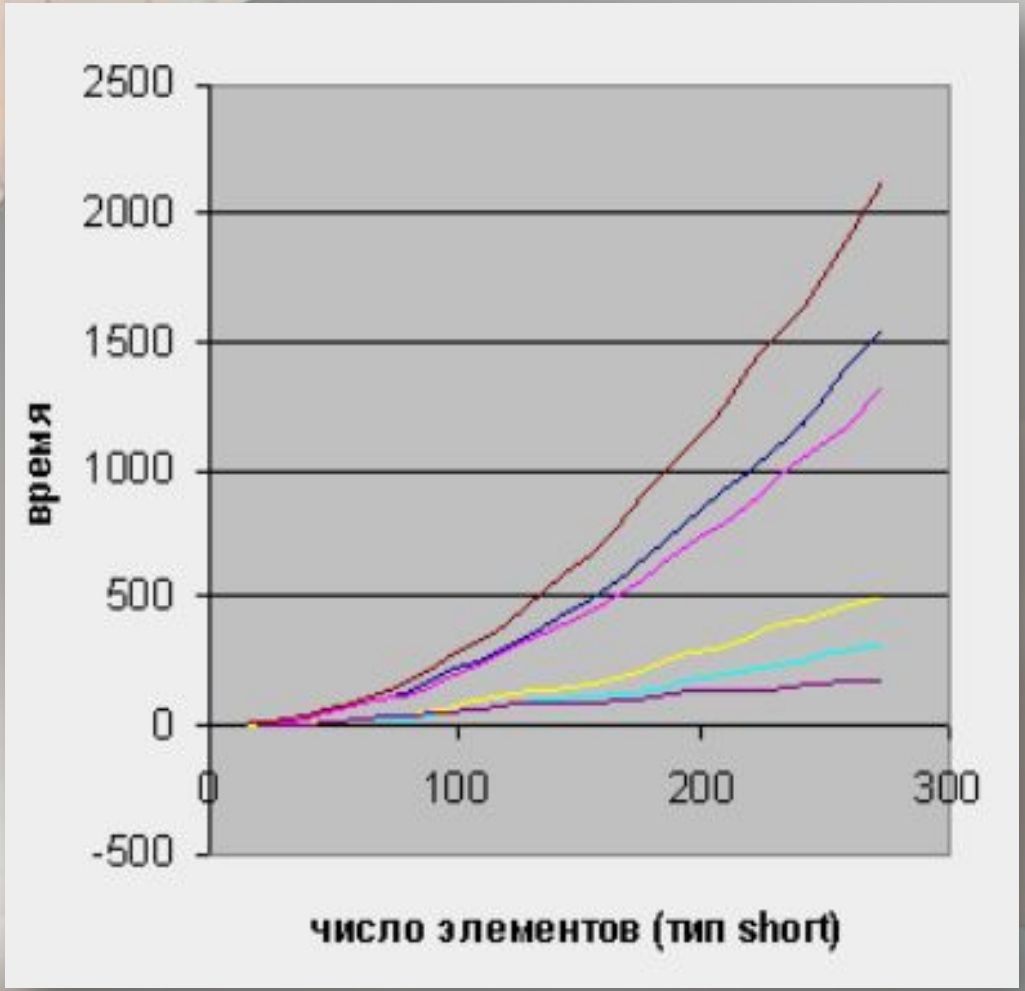




Сортировка методом Шелла

Алгоритм	Временная сложность	Дополнит память	Устойчивость	Естественность поведения
InsertSort - метод вставок	$\Omega(n), O(n^2)$	не требуется	да	да
Метод Шелла	$O(n^{1.25})$ $O(n^{1.5})$ – *наихудший случай*	не требуется	сортировка теряет устойчивость	да

Эффективность изученных алгоритмов



сортировка пузырьком

шейкер-сортировка

сортировка выбором

сортировка вставками

сортировка вставками со сторожевым элементом

сортировка Шелла

Эффективность изученных алгоритмов

По результатам замеров производительности методов можно сделать следующие выводы:

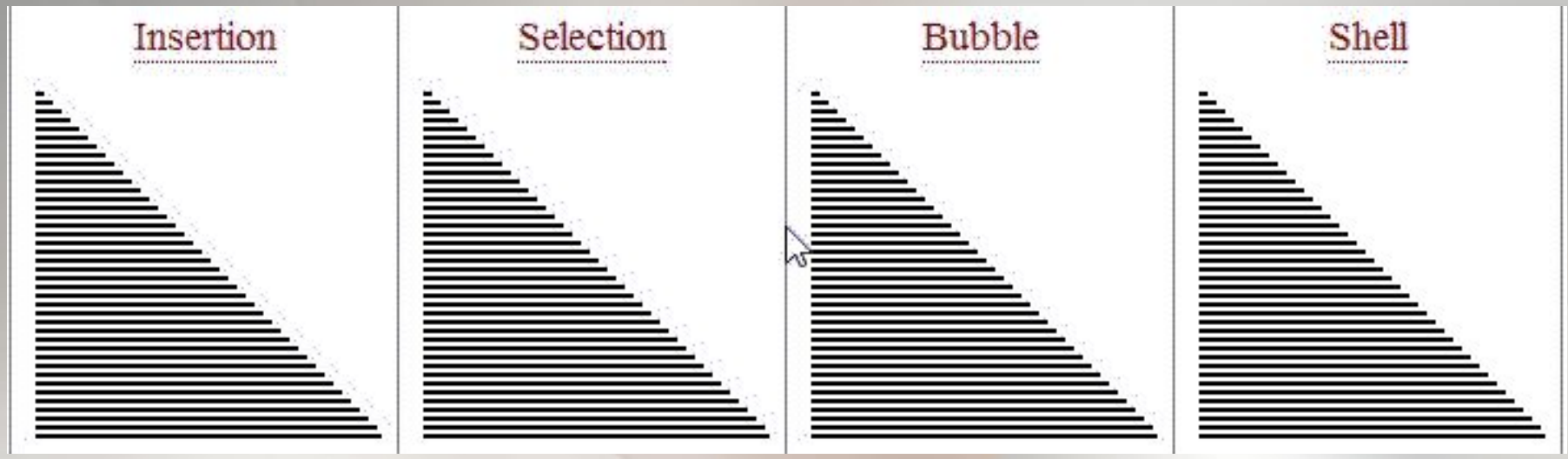
- Наиболее универсальным методом, является метод быстрой сортировки («QuickSort»), он показывает стабильно высокие результаты на любых размерах массивов. На втором месте находится метод Шелла. Его использование может быть обосновано более простым алгоритмом с точки зрения программиста
- Метод вставки эффективен, при условии большого времени выполнения операций перестановки, так как он является абсолютным лидером по количеству перестановок, проигрывая при этом по количеству сравнений

Эффективность изученных алгоритмов

По результатам замеров производительности методов можно сделать следующие выводы:

- При использовании небольших массивов данных нет большой разницы по скорости между методами сортировки, поэтому целесообразнее применять метод пузырька или метод вставок
- Исследование проводилось на массивах с большой степенью неупорядоченности. Для массивов, которые уже являются почти отсортированными, наиболее применим метод сортировки вставками

Эффективность изученных алгоритмов



вставка *выбор* *обмен* *метод Шелла*


Консультации
понедельник – 5 пара
пятница – 4 пара