# Поразрядная сортировка

- Сортировка применима к спискам, массивам, содержащим цифровую и текстовую информацию. Целые числа могут быть со знаком и без него, вещественные числа могут быть записаны в форме с фиксированной или плавающей десятичной точкой.

- Рассматриваемый алгоритм имеет следующие особенности.
- не использует сравнений сортируемых элементов.
- ключ, по которому происходит сортировка, необходимо разделить на части, *разряды* ключа. Например, слово можно разделить по буквам, число - по цифрам...
- До сортировки необходимо знать два параметра***: k*** и ***m***, где
- k - количество разрядов в самом длинном ключе
- m - разрядность данных: количество возможных значений разряда ключа
- При сортировки чисел ***m*** = 10 (0.1 . . 9) , ***k*** = наибольшему количеству цифр в числах )
- Эти параметры нельзя изменять в процессе работы алгоритма.

| 0 | 8 | 12 | 56 | 7 | 26 | 44 | 97 | 2 | 37 | 4 | 3 | 3 | 45 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **L0:** | 0 | 10 | | | | | |
| **L1:** | | | | | | | |
| **L2:** | 12 | 2 | | | | | |
| **L3:** | 3 | 3 | | | | | |
| **L4:** | 44 | 4 | | | | | |
| **L5:** | 45 | | | | | | |
| **L6:** | 56 | 26 | | | | | |
| **L7:** | 7 | 97 | 37 | | | | |
| **L8:** | 8 | | | | | | |
| **L9:** | | | | | | | |

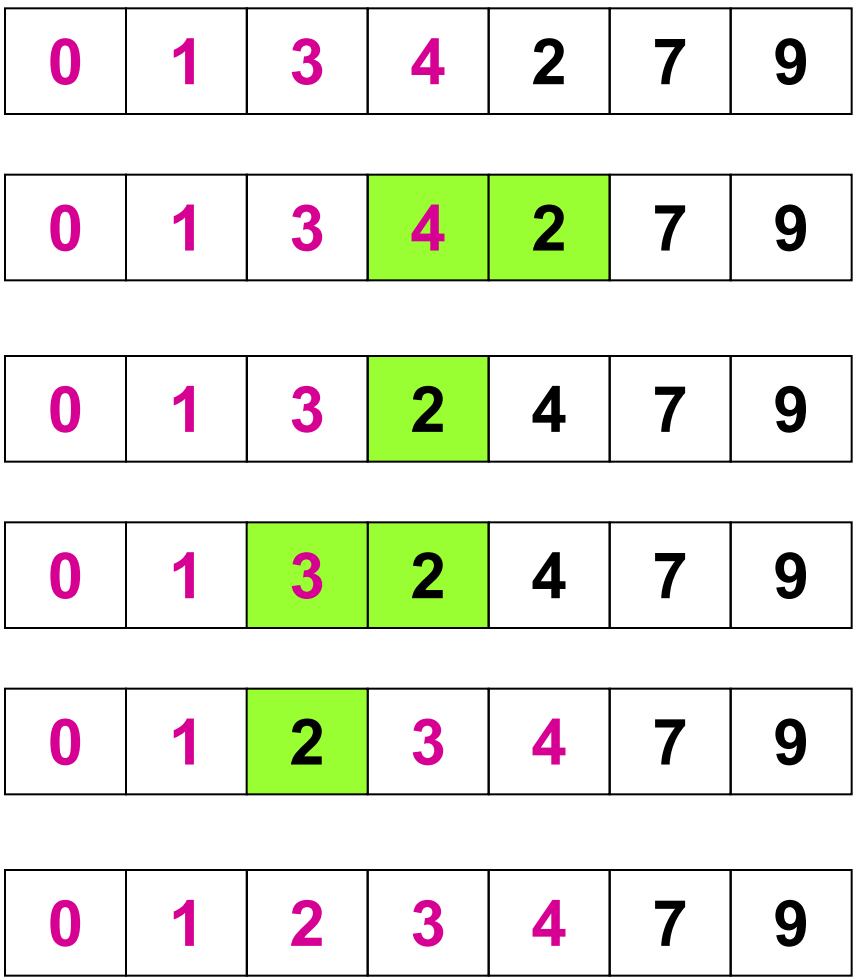
| 0 | 10 | 12 | 2 | 3 | 3 | 44 | 4 | 45 | 56 | 26 | 7 | 97 | 37 | 8 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **L0:** | 0 | 2 | 3 | 3 | 4 | 7 | 8 |
| **L1:** | 10 | 12 | | | | | |
| **L2:** | 26 | | | | | | |
| **L3:** | 37 | | | | | | |
| **L4:** | 44 | 45 | | | | | |
| **L5:** | 56 | | | | | | |
| **L6:** | | | | | | | |
| **L7:** | | | | | | | |
| **L8:** | | | | | | | |
| **L9:** | 97 | | | | | | |

# Сортировка вставками

- Рассмотрим действия на i-м шаге алгоритма. Последовательность к этому моменту разделена на две части: готовую a[0]...a[i] и неупорядоченную a[i+1]...a[n].
- На следующем, (i+1)-м каждом шаге алгоритма берем a[i+1] и вставляем на нужное место в готовую часть массива.
- Поиск подходящего места для очередного элемента входной последовательности осуществляется путем последовательных сравнений с элементом, стоящим перед ним.
- В зависимости от результата сравнения элемент либо остается на текущем месте(вставка завершена) либо они меняются местами и процесс повторяется.

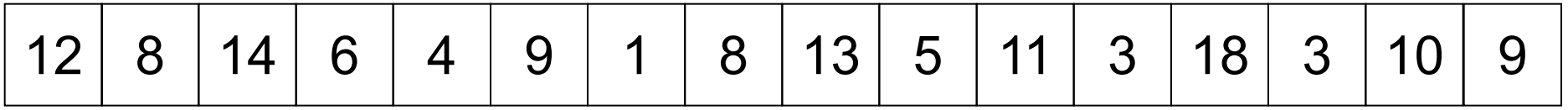
# Сортировка вставками



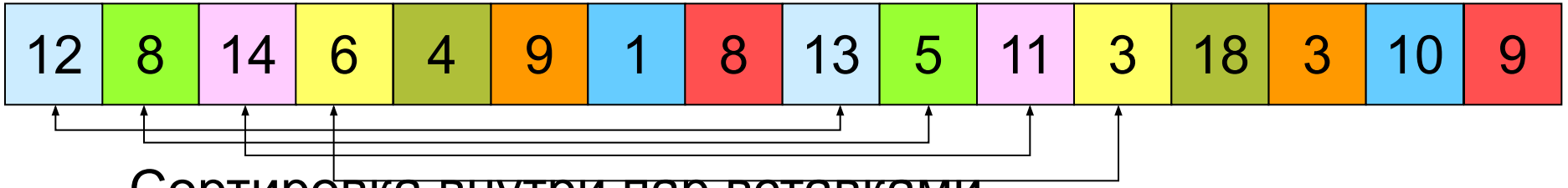
- В процессе вставки "просеиваем" элемент ***a[i]*** к началу массива, останавливаясь в случае, когда
    1. Найден элемент, меньший ***a[i],***
    2. Достигнуто начало последовательности.
- Дополнительная память не используется.
- Хорошим показателем сортировки является весьма естественное поведение: почти отсортированный массив будет досортирован очень быстро.
- Метод устойчив.

# Сортировка Шелла

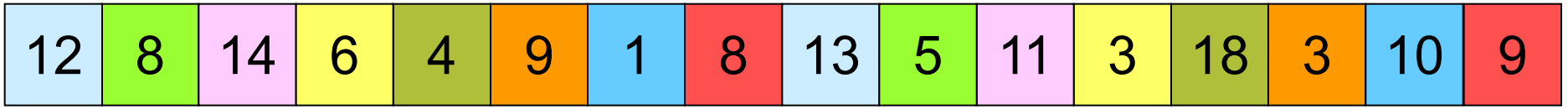
- Сортировка Шелла является модификацией алгоритма сортировки простыми вставками.
- Исходный массив



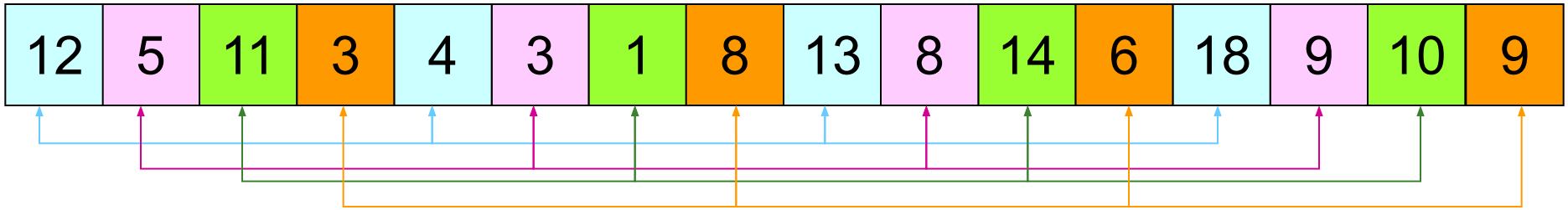

- Объединение парами



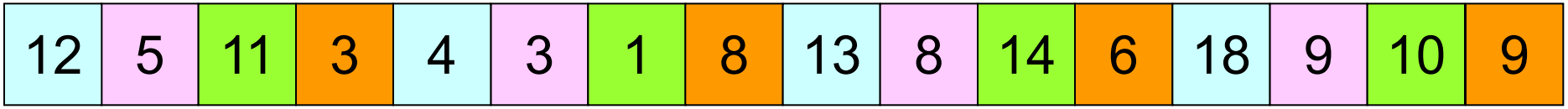

- Сортировка внутри пар вставками



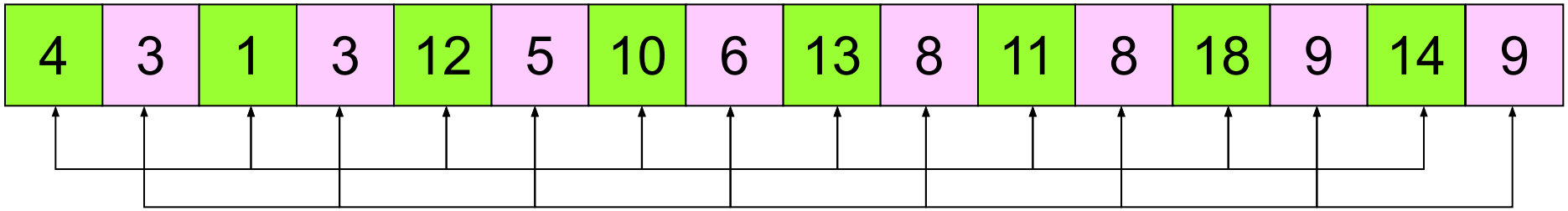
Объединение «четвёрками»



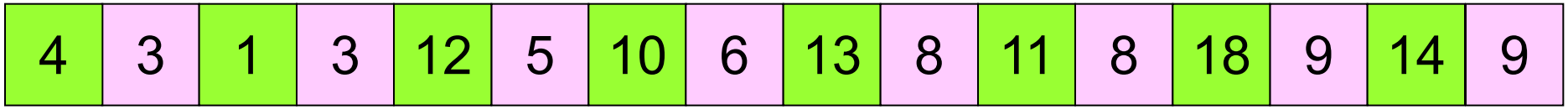

Сортировка внутри «четвёрок» вставками



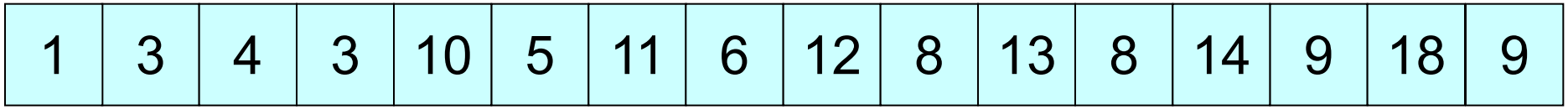
- Объединение «восьмерками»




- Сортировка внутри «восьмерок» вставками




- Объединение заключительное



| 1 | 3 | 4 | 3 | 10 | 5 | 11 | 6 | 12 | 8 | 13 | 8 | 14 | 9 | 18 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

- Сортировка вставками заключительная

| 1 | 3 | 4 | 3 | 10 | 5 | 11 | 6 | 12 | 8 | 13 | 8 | 14 | 9 | 18 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

# Сортировка выбором

- Отсортированная последовательность создается путем присоединения к ней одного элемента за другим в правильном порядке.
- Готовая последовательность строится, начиная с левого конца массива. Алгоритм состоит из n последовательных шагов, начиная от первого и заканчивая (n-1)-м.
- На i-м шаге выбираем наименьший из элементов a[i] ... a[n] и меняем его местами с a[i].

- Шаг-1 (Исходная последовательность )

| 4 | 9 | 7 | 6 | 2 | 3 |
|---|---|---|---|---|---|

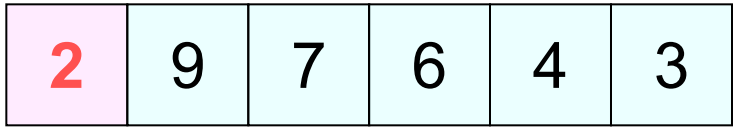
- Нахождение минимального a[m1] элемента из последовательности a[1] ÷ a[n].

| 4 | 9 | 7 | 6 | **2** | 3 |
|---|---|---|---|---|---|

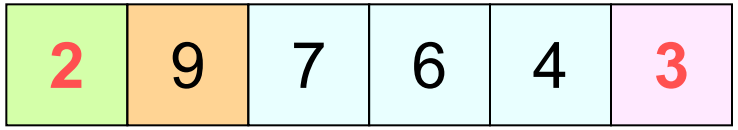
- Меняем местами a[1] и a[m1].

| 4 | 9 | 7 | 6 | **2** | 3 |
|---|---|---|---|---|---|

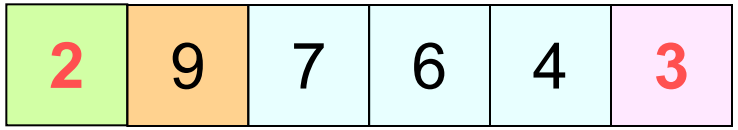
- Шаг-2



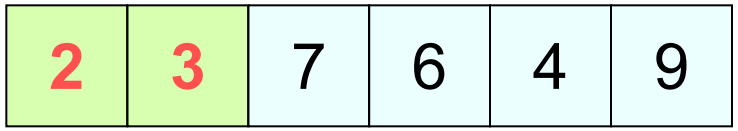

- Нахождение минимального a[m2] элемента из последовательности a[2] ÷ a[n].



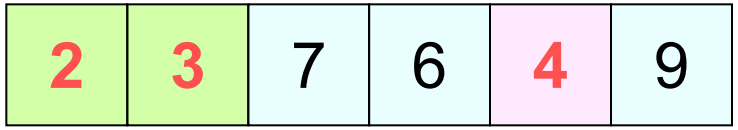

- Меняем местами a[2] и a[m2].



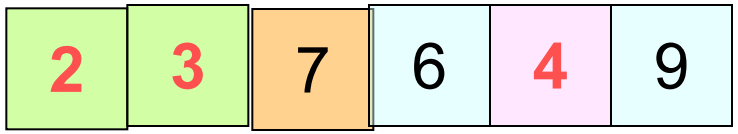
- Шаг-3



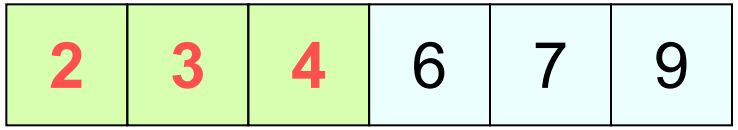

- Нахождение минимального a[m3] элемента из последовательности a[3] ÷ a[n].




- Меняем местами a[3] и a[m3].

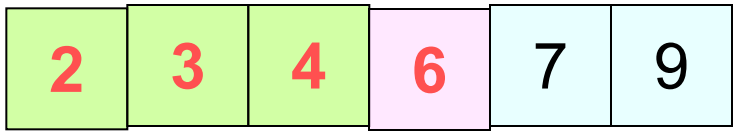


- Шаг-4




- Нахождение минимального a[m4] элемента из последовательности a[4] ÷ a[n].


- Меняем местами a[4] и a[m4]. (в данном случае элемент не смещается)



- Вне зависимости от номера текущего шага i, последовательность a[1]...a[i] является упорядоченной. Таким образом, на (n-1)-м шаге вся последовательность, кроме a[n] оказывается отсортированной, а a[n] стоит на последнем месте по праву: все меньшие элементы уже ушли влево.
- Для нахождения наименьшего элемента из n рассматриваемых алгоритм совершает n сравнений. С учетом того, что количество рассматриваемых на очередном шаге элементов уменьшается на единицу, общее количество операций:

  $$n + (n-1) + (n-2) + (n-3) + \ldots + 1 = 1/2 * ( n2+n )$$

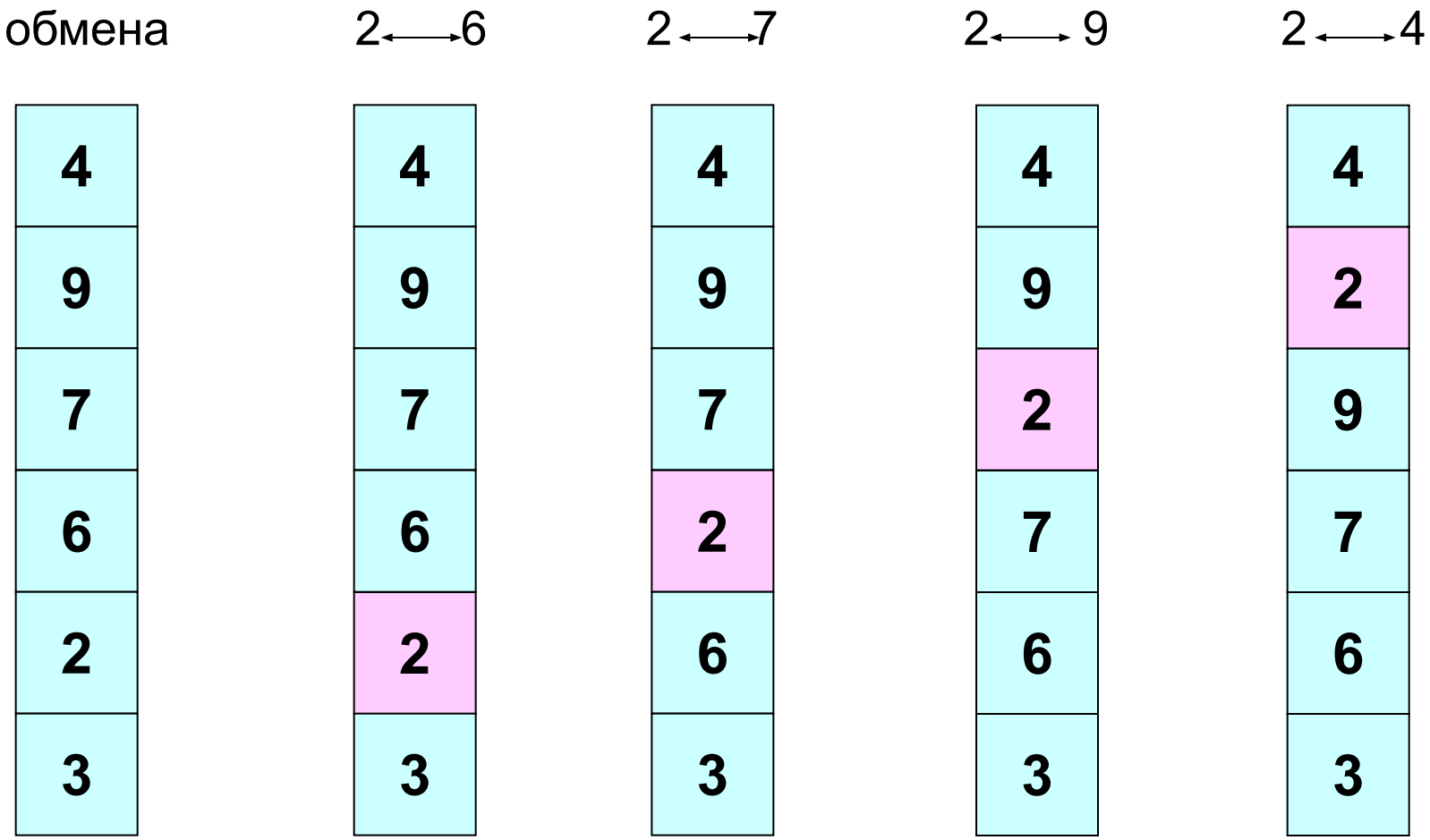
  Таким образом, так как число обменов всегда будет меньше числа сравнений, время сортировки растет квадратично относительно количества элементов.

- Алгоритм **не использует дополнительной памяти**: все операции происходят "на месте".
- Метод **неустойчив**.
- Если входная последовательность почти упорядочена, то сравнений будет столько же, значит алгоритм ведет себя **неестественно**.

# Сортировка пузырьком

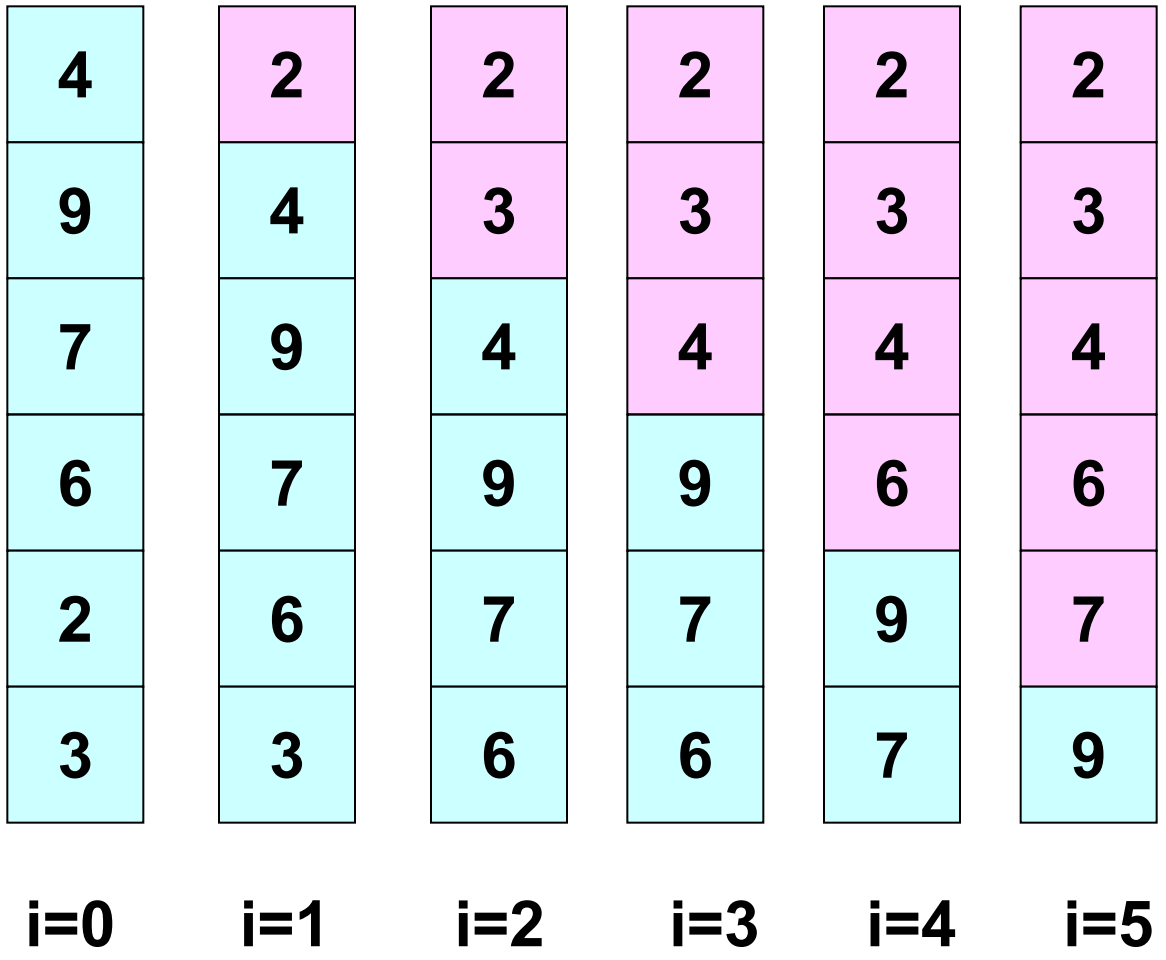
- Расположим массив сверху вниз, от нулевого элемента - к последнему.
- Идея метода: шаг сортировки состоит в проходе снизу вверх по массиву. По пути просматриваются пары соседних элементов. Если элементы некоторой пары находятся в неправильном порядке, то меняем их местами.

- Нулевой проход



- После нулевого прохода по массиву "вверху" оказывается самый "легкий" элемент - отсюда аналогия с пузырьком.
- Следующий проход делается до второго сверху элемента, таким образом второй по величине элемент поднимается на правильную позицию…
- Делаем проходы по все уменьшающейся нижней части массива до тех пор, пока в ней не останется только один элемент. На этом сортировка заканчивается, так как последовательность упорядочена по возрастанию.

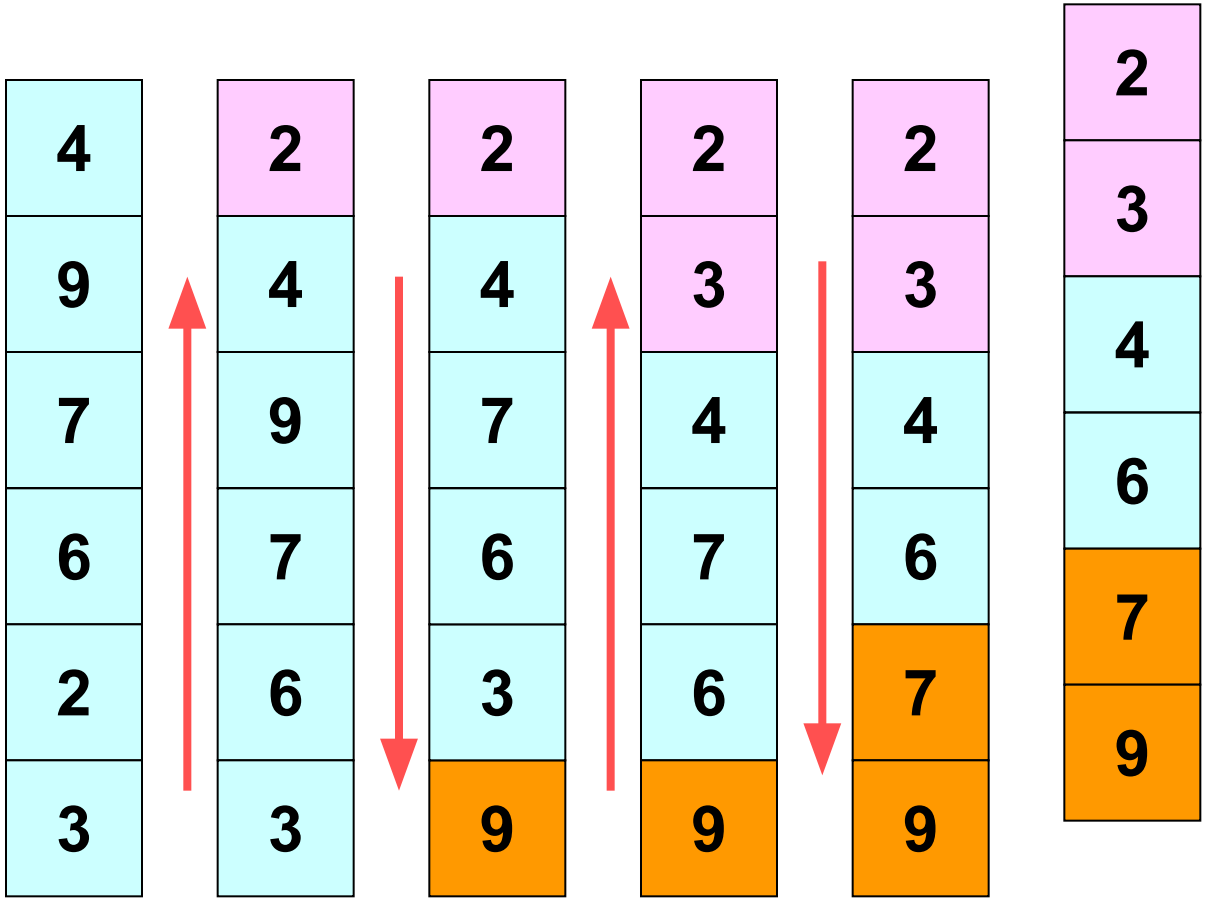
- Номер прохода



- **Дополнительная памятьне требуется**.
- Поведение усовершенствованного (но не начального) метода довольно **естественное,** почти отсортированный массив будет отсортирован намного быстрее случайного.
- Сортировка пузырьком **устойчива,** однако шейкер-сортировка утрачивает это качество.

# Шейкер-сортировка

- Улучшение алгоритма можно получить из следующего наблюдения. Легкий пузырек снизу поднимется наверх за один проход, тяжелые пузырьки опускаются со минимальной скоростью: один шаг за итерацию.
- Чтобы избежать подобного эффекта, можно менять направление следующих один за другим проходов. Получившийся алгоритм иногда называют "*шейкер-сортировкой*".




Номер прохода

i=0 i=1 i=2 i=3 i=4 i=5

# Сравнительная таблица

| | Быстро действие | Доп. память | Устойчивость | Естественность |
|---|---|---|---|---|
| Быстрая | **1** | да | нет | да |
| Слиянием | **2** | да | да | да |
| Вставками | **4** | нет | да | нет |
| Шелла | **3** | нет | да | нет |
| Выбором | **5** | нет | нет | нет |
| Пузырьковая | **7** | нет | да | да/нет |
| Шейкер | **6** | нет | нет | нет |