

Шейкерная сортировка ShakerSort

Можно заметить, что в BubbleSort «легкие» элементы «всплывают» быстро, а «тяжелые» «тонут» медленно.

Пример. СИДОРОВ

ВО

ВР

ВО

ВД

ВИ

ВСИДОРОВ

Шейкерная сортировка ShakerSort

ДВА (!) изменения в алгоритме, которые были предложены **для уменьшения трудоемкости**:

- 1) Изменение **направления просмотра** массива на каждой итерации.
- 2) Установление **границ рабочей части** массива **на место последнего обмена** на каждой итерации.

Шейкерная сортировка ShakerSort

Алгоритм на псевдокоде

L, R – левая и правая границы рабочей части массива,
n – количество элементов в массиве.

```
L := 1, R := n, k := n,  
DO  
  DO ( j := R, R-1, ... L+1)  
    IF (aj < aj-1) aj ↔ aj-1, k := j FI  
  OD  
  L := k  
  DO ( j := L, L+1, ... R-1)  
    IF (aj > aj+1) aj ↔ aj+1, k := j FI  
  OD  
  R := k  
OD (L < R)
```


Два усовершенствования в алгоритме позволяют уменьшить **только количество сравнений**:

$$M_{\text{ср}} = \frac{3(n^2 - n)}{4}$$

$$C < \frac{n^2 - n}{2}$$

$$T = O(n^2), n \rightarrow \infty$$

Метод шейкерной сортировки **сильно зависит** от исходной упорядоченности массива по количеству сравнений.

Метод обеспечивает **устойчивую** сортировку.

Метод	Трудоемкость	Устойчивость	Зависимость от упорядоченности
SelectSort	$O(n^2)$	Не устойчив	Не зависит
BubbleSort	$O(n^2)$	Устойчив	Зависит
ShakerSort	$O(n^2)$	Устойчив	Зависит

Видео: BubbleSort или ShakerSort ?



Метод прямого включения InsertSort

Начиная с $i = 2$ берём очередной i -й элемент массива и включаем его на нужное место среди первых $(i-1)$ элементов, при этом все элементы, которые больше a_i сдвигаются на одну позицию вправо.

Метод прямого включения

Алгоритм на псевдокоде

```
DO ( i := 2, 3, ..., n )
  t := ai, j := i - 1
  DO ( j > 0 и t < aj )
    aj+1 := aj
    j := j - 1
  OD
  aj+1 := t
OD
```

К У Р А П О В А
К У Р А П О В А
К Р У А П О В А
А К Р У П О В А
А К П Р У О В А
А К О П Р У В А
А В К О П Р У А
А А В К О П Р У

Для определения трудоемкости оценим количество операций **для каждого значения i** :

$$C_{\min} = 1 \quad C_{\max} = i - 1$$

$$M_{\min} = 2 \quad M_{\max} = i + 1$$

Тогда **для всех i** :

$$C_{\min} = n - 1 \quad C_{\max} = \frac{n^2 - n}{2}$$

$$M_{\min} = 2(n - 1) \quad M_{\max} = \frac{n^2 - n}{2} + 2n - 2$$

Средняя трудоемкость этого метода имеет квадратичный порядок $T = O(n^2)$, $n \rightarrow \infty$.

Метод прямого включения **устойчивый**.

Метод **сильно зависит** от исходной упорядоченности массива.

Метод	Трудоёмкость	Устойчивость	Зависимость от упорядоченности
SelectSort	$O(n^2)$	Не устойчив	Не зависит
BubbleSort	$O(n^2)$	Устойчив	Зависит
ShakerSort	$O(n^2)$	Устойчив	Зависит
InsertSort	$O(n^2)$	Устойчив	Сильно зависит

Видео: InsertSort



Метод Шелла

ShellSort

Из оценок метода прямого включения *InsertSort* видно, что, чем лучше упорядочен массив, тем меньше операций потребуется для его сортировки.

Основная идея метода Шелла *ShellSort* состоит в **предварительном улучшении порядка элементов** массива, а затем окончательной сортировке его методом прямого включения.

Шелл предложил работать в методе прямого включения с **шагом k** большим, чем 1, что предварительно улучшило бы упорядоченность массива.

Чем больше величина шага, тем меньше операций сравнения и пересыпки приходится

Определение Предварительная сортировка массива методом прямого включения с шагом $k > 1$ называется **к - сортировкой**.

Метод Шелла заключается в проведении сначала **нескольких к-сортировок**, а затем **окончательной сортировке массива с шагом $k=1$** .

Обозначим

$H = (h_1, h_2, \dots, h_m)$ – последовательность из m возрастающих шагов, где $h_1 = 1, h_1 < h_2 < h_3 < \dots < h_m$.

Производя последовательно **к-сортировки** с шагами h_m, h_{m-1}, \dots, h_1 , получим упорядоченный массив. Это гарантируется тем, что последний шаг $h_1 = 1$.

Метод Шелла (ShellSort)

Алгоритм на псевдокоде

<вычисление массива шагов h>

```
DO ( k := hm, hm-1, ... 1 )
    DO ( i := k+1, k+2, ... n )
        t := ai, j := i - k
        DO ( j > 0 и t < aj )
            aj+k := aj
            j := j - k
        OD
        aj+k := t
    OD
OD
```

К_ У Р □ А П О В А

К У_ Р □ А П О В А

К_ А Р_ У □ О В А

К А П_ У Р □ О В А

К А_ П О_ Р_ У □ В А

В_ А_ К_ О_ П_ У_ Р_ □ А

В А_ К А_ П_ О_ Р_ У



В А К А П О Р У
А В К А П О Р У
А В К А П О Р У
А А В К П О Р У
А А В К П О Р У
А А В К П Р У
А А В К О П Р У

The image shows a 7x8 grid of Cyrillic letters. The letters are arranged in rows and columns. The first row is В, А, К, А, П, О, Р, У. The second row is А, В, К, А, П, О, Р, У. The third row is А, В, К, А, П, О, Р, У. The fourth row is А, А, В, К, П, О, Р, У. The fifth row is А, А, В, К, П, О, Р, У. The sixth row is А, А, В, К, П, Р, У. The seventh row is А, А, В, К, О, П, Р, У. Annotations include: underlines under В (row 1), А (row 2), А (row 3), А (row 4), А (row 5), А (row 6), А (row 7); boxes around А (row 1), К (row 2), А (row 3), П (row 4), О (row 5), Р (row 6), У (row 7); and arrows pointing from В (row 1) to В (row 2), from А (row 3) to В (row 4), from К (row 3) to К (row 4), from О (row 5) to П (row 6), and from П (row 6) to П (row 7).

Эффективность метода Шелла по времени работы зависит от **выбора значений шагов**.

Последовательность значений шагов, которая дает наилучшую трудоемкость, пока неизвестна, но существует и часто используется следующая **последовательность шагов, предложенная**

Д.Кнудом:

$$h_1 = 1, \quad h_i = 2h_{i-1} + 1, \quad i = 2, \dots, m, \quad m = \lfloor \log_2 n \rfloor - 1$$

Пример

$$n = 100, \quad m = \lfloor \log_2 100 \rfloor - 1 = 5,$$

$$h_1 = 1, \quad h_2 = 2h_1 + 1 = 3,$$

$$h_3 = 2h_2 + 1 = 7, \quad h_4 = 2h_3 + 1 = 15, \quad h_5 = 2h_4 + 1 = 31.$$

При использовании **последовательности шагов, предложенной Д.Кнутом**, метод имеет порядок трудоёмкости $O(n^{1.2})$, $n \rightarrow \infty$.

Метод Шелла **существенно быстрее** методов с квадратичной трудоёмкостью.

В примере: Insert – 46 операций,
Shell – 34 операции.

Метод Шелла **не устойчив**.

Метод **зависит** от исходной упорядоченности массива.

Метод	Трудоемкость	Устойчивость	Зависимость от упорядоченности
SelectSort	$O(n^2)$	Не устойчив	Не зависит
BubbleSort	$O(n^2)$	Устойчив	Зависит
ShakerSort	$O(n^2)$	Устойчив	Зависит
InsertSort	$O(n^2)$	Устойчив	Сильно зависит
ShellSort	$O(n^{1,2})$	Не устойчив	Зависит

Видео: ShellSort

