Лекция 11

Улучшенные сортировки

Сортировка включениями с убывающим шагом. Метод Шелла

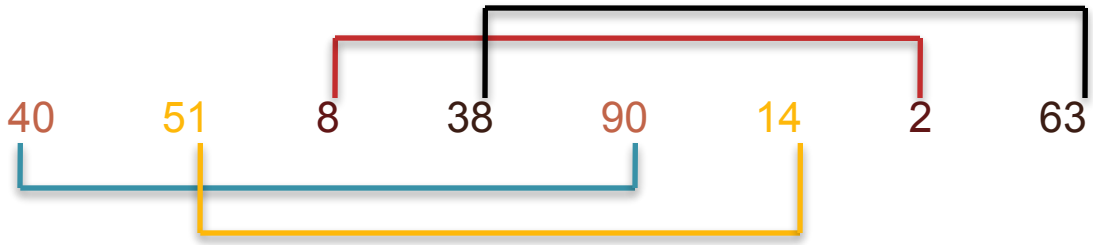
Хоар, Флойд, Шелл: для алгоритмов сортировки, перемещающих в последовательности запись вправо или влево только на одну позицию, среднее время работы будет в лучшем случае пропорционально N^2.

Хотелось бы, чтобы записи перемещались «большими скачками, а не короткими шажками».

Д. Шелл предложил в 1959 г. метод, названный сортировкой *с убывающим шагом.*

Пример работы сортировки Шелла

На первом проходе выделим в подпоследовательности элементы, отстоящие друг от друга на четыре позиции:

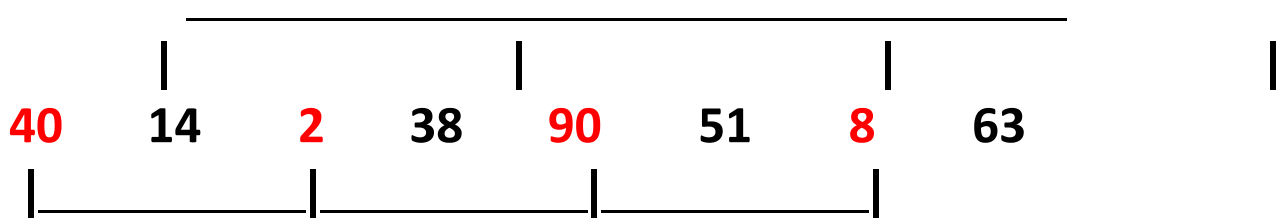


Полученные 4 последовательности отсортируем на месте независимо друг от друга методом простых включений.

Этот процесс называется *4-сортировкой.*

Пример работы сортировки Шелла, продолжение

В результате 4-сортировки получим последовательность:



На следующем шаге элементы, отстоящие друг от друга на две позиции, объединяются в подпоследовательности и сортируются простыми вставками независимо друг от друга.

Этот процесс называется *2-сортировкой.*

После 2-сортировки получим последовательность:

2 14 8 38 40 51 90 63

Ее сортируют методом простых вставок.

К последнему шагу элементы довольно хорошо упорядочены, поэтому требуется мало перемещений.

Данный процесс называется *1-сортировкой.*

Выбор шага в сортировке Шелла

В сортировке методом Шелла можно использовать любую убывающую последовательность шагов

$$h_t,\ h_{t-1},\ \ldots,\ h_1$$

Чтобы выбрать некоторую хорошую последовательность шагов сортировки, нужно проанализировать время работы как функцию от этих шагов.

До сих пор не удалось найти наилучшую возможную последовательность шагов $h_t, h_{t-1}, \ldots, h_1$ для больших *N*.

Выявлен примечательный факт, что элементы последовательностей приращений не должны быть кратны друг другу.

Это позволяет на каждом проходе сортировки перемешивать цепочки, которые ранее никак не взаимодействовали.

Желательно, чтобы взаимодействие между разными цепочками происходило как можно чаще.

Кнут:

..., 121, 40, 13, 4, 1, где $h_{k+1} = 3 \cdot h_k + 1$, $h_1 = 1$

.... 31, 15, 7, 3, 1, где $h_{k-1} = 2 \cdot h_k + 1$, $h_t = 1$

Анализ эффективности метода

Утверждение

Если k-отсортированная последовательность i-сортируется (k > i), то она остается k-отсортированной.

→ С каждым следующим шагом сортировки с убывающим приращением количество отсортированных элементов в последовательности возрастает.

Для последовательности шагов $2^k + 1, \ldots, 9, 5, 3, 1$

количество пересылок пропорционально $N^{1.27}$,

для последовательности $2^k - 1, \ldots, 15, 7, 3, 1$ — $N^{1.26}$,

для последовательности $(3^k - 1)/2, \ldots, 40, 13, 4, 1$ — $N^{1.25}$

Общая оценка: величина порядка $N^{3/2}$

Алгоритм

```
процедура Вставка(b, h)
  // b — номер первого элемента последовательности
  // h – величина шага
начало процедуры
   // Пусть i – номер первого элемента в несортированной части
массива
   i := b + h;
   пока i ≤ N выполнять
      x:= A[i];
      j := i – h;
      пока j ≥ b и A[j]>x выполнять
      // Все элементы из отсортированной части, большие
      // x, сдвинуть на величину шага h вправо,
         A[j+h] := A[j];
      j := j – h;
      конец пока
      // Элемент x поставить на свое место по порядку:
      A[j+h] := x;
      i := i + h;
   конец пока
конец процедуры
```

Алгоритм, продолжение

```
Основная программа:
// Выбор начального шага:
h := 1;
пока h < N/3 выполнять
    h := 3*h + 1;
конец пока
// Сортировка:
пока h ≥ 1 выполнять
    цикл по i от 1 до h с шагом 1 выполнять
        Вставка (i, h);
    конец цикла
    h := (h – 1) / 3;
конец пока
```

Пирамидальная сортировка

При сортировке методом простого выбора на каждом шаге выполняется линейный поиск минимального элемента. Линейная сложность этого поиска делает сложность всей сортировки квадратичной.

Возможно ли найти минимальный элемент за время лучшее линейного?

Оказывается, что это возможно, если использовать на каждом следующем шаге информацию о взаимных отношениях элементов, накопленную на предыдущих шагах.

Идея бинарного выбора может быть эффективно применена, если организовать входные данные в виде так называемой *пирамиды* (или *сбалансированного бинарного дерева поиска*) и поддерживать их в этом виде в процессе сортировки.

Метод сортировки с использованием такой пирамиды был предложен Р. У. Флойдом в 1964 г. под названием *«Heap sort»* — *пирамидальной сортировки* или *сортировки кучей*.

Свойство пирамиды

Пусть дана последовательность $h_1, ..., h_n$.

Элемент h_i *образует пирамиду* в этой последовательности, если выполнены следующие условия:

- если $2 \cdot i \le n$, то $h_i \ge h_{2i}$ и h_{2i} образует пирамиду;
- если $2 \cdot i + 1 \le n$, то $h_i \ge h_{2i+1}$ и h_{2i+1} образует пирамиду.

Элементы $h_{n/2+1}, ..., h_n$ всегда образуют тривиальные пирамиды, поскольку для них приведенные условия имеют ложные посылки.

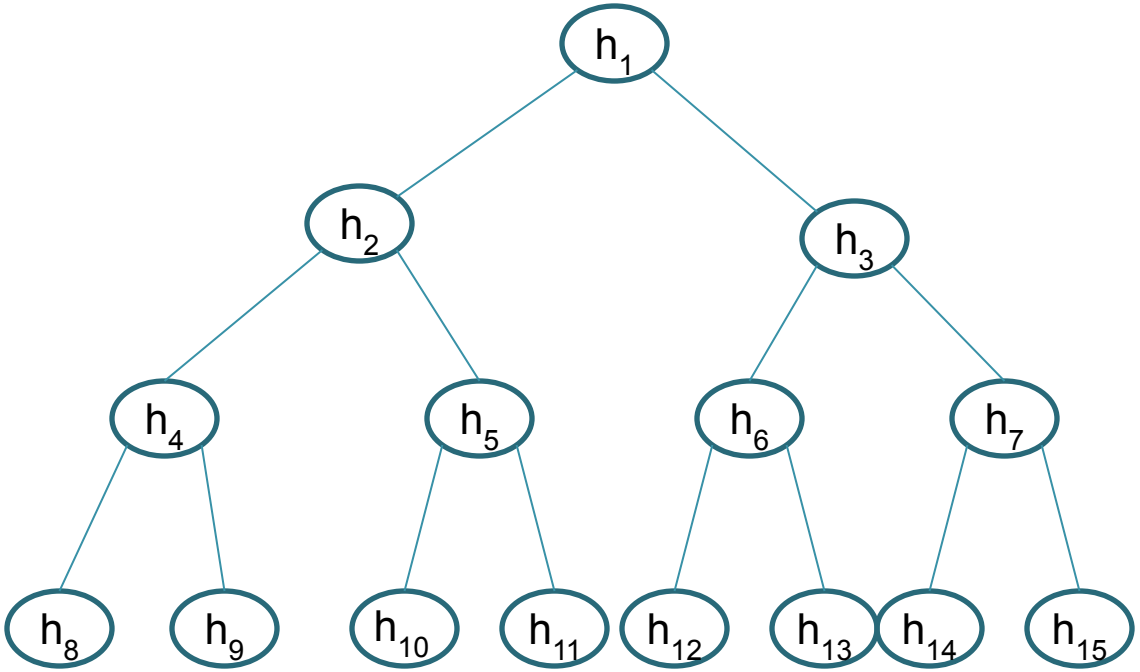
Если элемент h_1 образует пирамиду, то и каждый элемент последовательности образует пирамиду.

В этом случае будем говорить, что вся последовательность является *полной пирамидой.*

Полная пирамида при n=15

Полная пирамида может быть изображена в виде корневого бинарного дерева, в котором элементы h_{2i} и h_{2i+1} являются сыновьями элемента h_i.

Элемент в любом узле численно не меньше всех своих потомков, а вершина полной пирамиды h_1 содержит максимальный элемент всей последовательности.



Пример полной пирамиды при n=12

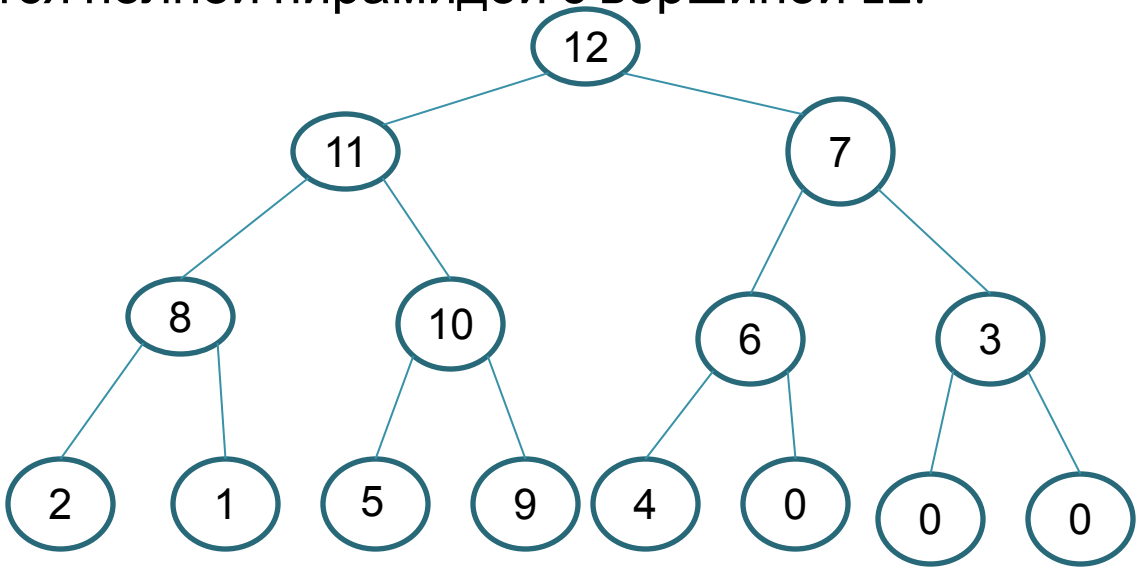
Если число элементов в полной пирамиде не равно $2k - 1$, самый нижний уровень дерева будет неполным: недостающих сыновей можно достроить, добавив в пирамиду несколько заключительных «минимальных» элементов «0», не нарушающих условия пирамиды.

Последовательность, упорядоченная по убыванию, является полной пирамидой.

Например, последовательность из 12 элементов

12, 11, 7, 8, 10, 6, 3, 2, 1, 5, 9, 4

является полной пирамидой с вершиной 12.



Идея метода пирамидальной сортировки

1. Подготовка к сортировке: входная неупорядоченная последовательность перестраивается в пирамиду.
2. Сортировка: входная и готовая последовательности хранятся в одном массиве, причем готовая последовательность формируется *в хвосте* массива, а входная остается в начале массива.

Основой реализации метода является следующая процедура *просеивания*.

Пусть в последовательности $h_1, ..., h_n$ элементы $h_{i+1}, ..., h_n$ уже образуют пирамиды.

Требуется перестроить последовательность так, чтобы пирамиду образовывал элемент h_i.

На каждой итерации цикла наибольший из трех элементов h_i, $h_{2 \cdot i}$ и $h_{2 \cdot i+1}$ путем обмена оказывается в корне текущего поддерева, что обеспечивает истинность условий пирамиды в этом корне.

Если при этом изменяется корень левого или правого

Построение пирамиды

	$a1$	$a2$	$a3$	$a4$	$a5$	$a6$	$a7$	$a8$	$a9$	$a10$
Шаг 1, $i=5$:	52	81	42	23	11	76	91	63	37	20
Шаг 2, $i=4$:	52	81	42	23	20	76	91	63	37	11
Шаг 3, $i=3$:	52	81	42	63	20	76	91	23	37	11
Шаг 4, $i=2$:	52	81	91	63	20	76	42	23	37	11
Шаг 5, $i=1$:	52	81	91	63	20	76	42	23	37	11
Выход:	91	81	76	63	20	52	42	23	37	11

Сортировка

На каждом шаге сортировки первый элемент массива, т. е. максимальный элемент пирамиды, переносится в начало готовой последовательности путем обмена с последним элементом пирамиды, занимающим его место.

Затем остаток входной последовательности вновь перестраивается в пирамиду, обеспечивая корректность следующего шага.

В начале *i*-го шага элементы a_1, .., a_i, по предположению, хранят входную последовательность как пирамиду, а a_{i+1}, .., a_N – упорядоченную по возрастанию готовую последовательность (изначально пустую).

На *i*-м шаге текущий максимальный элемент пирамиды a_1 обменивается с a_i, становясь началом новой готовой последовательности, где он будет новым минимальным элементом. Входная последовательность (пирамида) при этом претерпевает два изменения:

— она теряет последний элемент, что не нарушает условий пирамиды ни в одном узле;

— ее первый элемент становится произвольным, что может нарушать условие пирамиды только в первом узле.

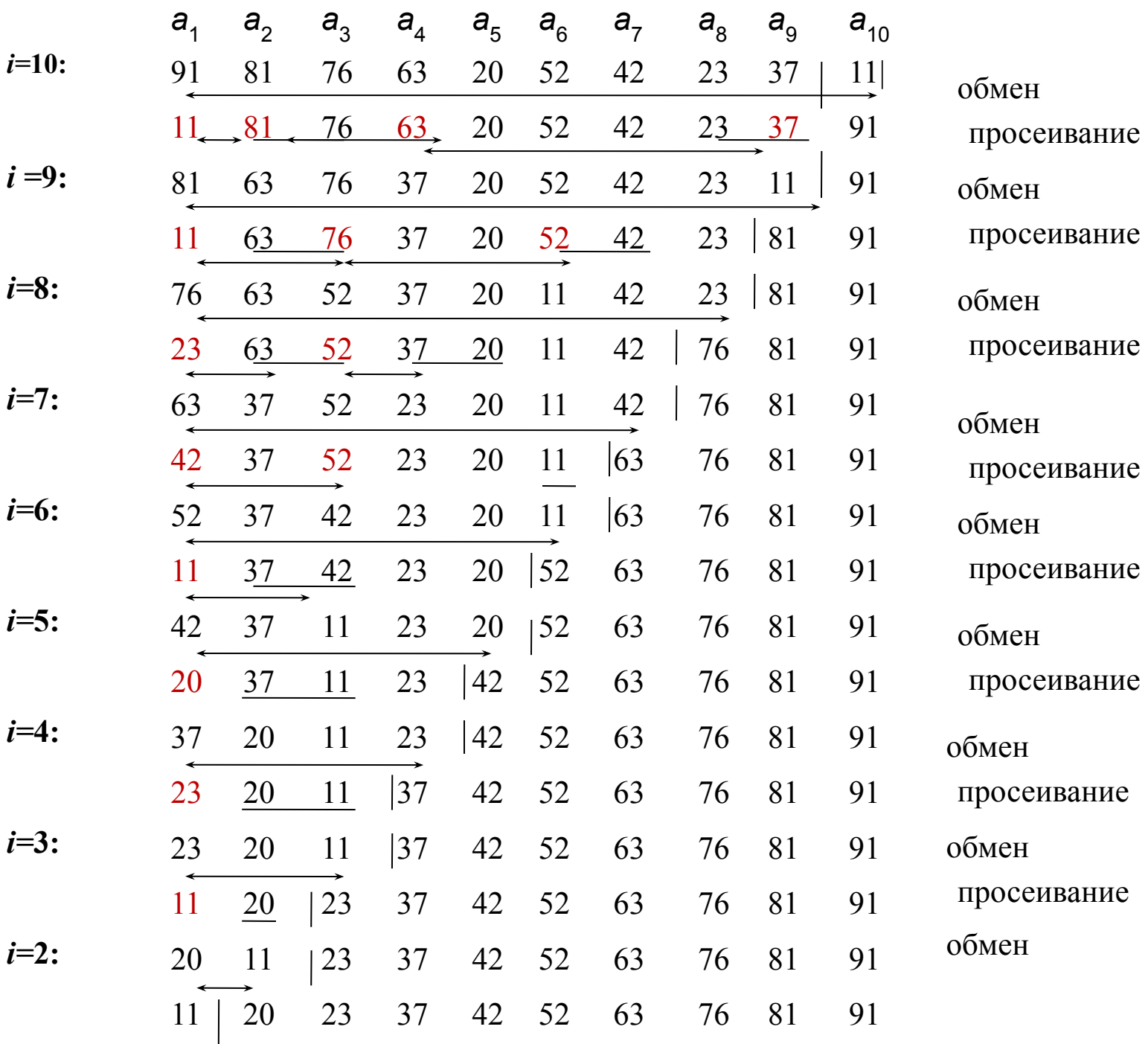
Сортировка, продолжение

Таким образом, для новой входной последовательности

$$a_1, \ldots, a_{i-1}$$

условия пирамиды выполнены для всех элементов, кроме первого.

Применение процедуры просеивания к a_1 восстанавливает полную пирамиду в $a_1, \ldots, a_{i-1}$, что обеспечивает условия осуществимости следующего шага.


a_1 a_2 a_3 a_4 a_5 a_6 a_7 a_8 a_9 a_{10}
i=10:
91 81 76 63 20 52 42 23 37 11
обмен
11 81 76 63 20 52 42 23 37 91
просеивание
i =9:
81 63 76 37 20 52 42 23 11 91
обмен
11 63 76 37 20 52 42 23 81 91
просеивание
i=8:
76 63 52 37 20 11 42 23 81 91
обмен
23 63 52 37 20 11 42 76 81 91
просеивание
i=7:
63 37 52 23 20 11 42 76 81 91
обмен
42 37 52 23 20 11 63 76 81 91
просеивание
i=6:
52 37 42 23 20 11 63 76 81 91
обмен
11 37 42 23 20 52 63 76 81 91
просеивание
i=5:
42 37 11 23 20 52 63 76 81 91
обмен
20 37 11 23 42 52 63 76 81 91
просеивание
i=4:
37 20 11 23 42 52 63 76 81 91
обмен
23 20 11 37 42 52 63 76 81 91
просеивание
i=3:
23 20 11 37 42 52 63 76 81 91
обмен
11 20 23 37 42 52 63 76 81 91
просеивание
i=2:
20 11 23 37 42 52 63 76 81 91
обмен
11 20 23 37 42 52 63 76 81 91

Алгоритм

```
процедура Просеивание (i, n)
    // i – номер элемента, который нужно просеять
        // n – номер последнего элемента массива
начало процедуры
    пока 2*i <= n
          r := 2*i;
           если (r+1 ≤ n) и (A[r] < A[r+1])
        то r := r + 1;
              // i-тый элемент массива ставится на то место,
           // где он удовлетворяет свойству пирамиды:
           // A[i] ≥ max(A[2*i], A[2*i + 1])
           если A[i] < A[r]
           то начало
                  Обмен (i, r);
                  i := r;
                   конец
        иначе выход из процедуры;
        конец пока
конец процедуры
```

Алгоритм, продолжение

```
Основная программа:
Шаг 1. Построение пирамиды:
i := N / 2;
пока i ≥ 1 выполнять
        Просеивание (i, N);
        i := i – 1;
конец пока
Шаг 2. Сортировка на пирамиде:
i := N;
пока i > 1 выполнять
    Обмен (1, i);
    i := i – 1;
    Просеивание (1, i);
конец пока
конец основной программы
```

Анализ

Число итераций цикла в процедуре просеивания не превосходит высоты пирамиды.

Высота полного бинарного дерева из N узлов, каковым является пирамида, равна $[log_2 N]$.

Просеивание имеет логарифмическую сложность.

Итоговая сложность пирамидальной сортировки $\sim N \cdot log_2 N$.

Наилучший случай – обратное упорядочение входной последовательности.

Сортировка с разделением. Быстрая сортировка Ч. Э. Р. Хоар

Это метод сортировки, при котором обмениваются местами пары несоседних элементов, а задача сортировки последовательности рекурсивно сводится к задачам сортировки ее меньших частей.

Допустим сначала, что мы уже переупорядочили некоторым образом элементы входной последовательности, после чего оказалось возможным разделить ее на две непустые части по границе некоторого индекса *m:*

левую (индексы ***1****...m*) и

правую (индексы *m*+1...*N*);

причем значения всех элементов левой части не превосходят значений всех элементов правой части:

$\forall\ i, j:\ \ 1 \le i \le m$ и $m < j \le N$ выполнено: $a_i \le a_j$. (*)

Индекс *m* назовем *медианой.*

Сортировка разделением, идея алгоритма

Отсортируем любым методом обмена отдельно левую часть, не затрагивая элементов правой части, а затем отдельно правую, не трогая левой.

При этом обмениваться могут только пары элементов, находящиеся в одной части, поэтому никакой обмен не нарушает свойство (*).

Значит, оно будет верно и для результирующей последовательности, которая в силу этого оказывается упорядоченной в целом.

```
СортировкаРазделением (l, r)
        /* l, r – границы сортируемой подпоследовательности */
            /* Разделение */
   привести подпоследовательность a_l,…, a_r к условию (*)
   и определить медиану m;
            /* Рекурсивный спуск */
   если l < m то       // части длины 0 и 1 не сортируем
      СортировкаРазделением (l, m);
   если m + 1 < r   то  // части длины 0 и 1 не сортируем
      СортировкаРазделением (m + 1, r);
конец
```

Получается, что в процессе дробления исходной задачи на подзадачи мы приходим к тривиальным подзадачам: сортировке последовательностей длины 0 и 1.

Не приходится ничего делать и для слияния решений подзадач в решение исходной задачи во время возврата из рекурсии: упорядоченные последовательности образуются сами собой по мере упорядочения их частей.

Где же тогда фактически выполняется сортировка?

- На фазе разделения, иллюстрируя, как хорошая подготовка условий для решения зачастую уже и дает решение!

В качестве критерия разделения нам понадобится так называемый *пилотируемый* элемент *x*.

В классической версии алгоритма в качестве *x* выбирается произвольный элемент сортируемой последовательности: первый, последний, расположенный в середине или иначе.

Влияние его выбора на эффективность алгоритма мы обсудим ниже.

В процессе разделения мы соберем в левой части последовательности все элементы $a_i \le x$, а в правой — все элементы $a_j \ge x$.

Условие (*) при этом будет выполнено даже при возможном наличии одинаковых элементов x в обеих частях.

Введем два бегущих индекса-указателя i и j, которые делят разделяемую подпоследовательность на три участка:

левый ($a_l \dots a_{i-1}$),

правый ($a_{j+1} \dots a_r$),

средний ($a_i \dots a_j$).

В левом и правом участках будут накапливаться элементы левой и правой частей, подлежащих затем рекурсивной сортировке, а в среднем находятся остальные, еще не распределенные элементы.

Процесс разделения

```
i = l;  j = r;
цикл
    пока a_i  < x цикл   /* проверка i < r не нужна: x где-то есть */
          i := i + 1;  /*  в конце a_i ≥ x */
    конец цикла;
    пока x < a_j цикл    /* проверка j > i не нужна: x есть */
        j := j – 1;  /* в конце a_j ≤ x */
    конец цикла;
    если i  ≤  j  то   /* если i = j, a[i = j] = x, нужен сдвиг индексов
                          для выхода из цикла */
        обменять a_i и a_j ;  /*  теперь a_i ≤ x ≤ a_j  */
        i := i + 1;  /*на случай a_i = x : добавить в левую часть */
        j := j – 1;/* на случай a_j = x : добавить в правую часть */
пока i < j;
```

Комментарии

Циклы по встречным индексам переносят из средней части в левую или правую элементы, строго меньшие или большие *x*, которые могут быть добавлены в эти части без перестановки.

После их выполнения процесс разделения либо заканчивается (если $i \geq j$), либо пара a_i и a_j образует инверсию.

В последнем случае их следует обменять и включить в левую и правую части.

Вот здесь и происходят упорядочивающие обмены с уменьшением числа инверсий в последовательности!

Проверка того, что бегущие индексы не выходят за границы *l...r,* строго говоря, необходима, но фактически не нужна:

на первом проходе выход за границы немозможен, так как в массиве есть сам элемент *x* и оба цикла остановятся на нем.

В конце же первого прохода происходит обмен элементов и обе части становятся не пусты, что гарантирует остановку циклов по встречным индексам в пределах интервала *l...r* и на следующих проходах.

Цикл оканчивается при $i \geq j$.

Однако нам еще надо определить медиану.

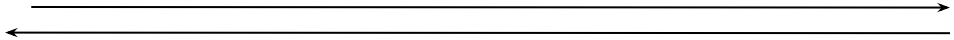
Определенные границы левой части – $l...i-1$,
правой – $j+1...r$, однако интервал $j+1...i-1$ может быть не вырожден и заполнен элементами x (почему?).

Эти элементы останутся на своих местах в процессе сортировки (почему?), поэтому их можно исключить из левой и правой частей.

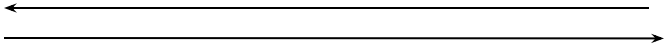
Окончательно границами левой части можно считать $l...j$,
а правой – $i...r$.

Процесс разделения, пример

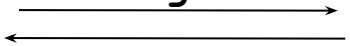
```
 Разделение:   40     51     8     38     89     1     15    63
    Начало:    i                  x                          j
Первый проход:i                                        j
Сближение
```



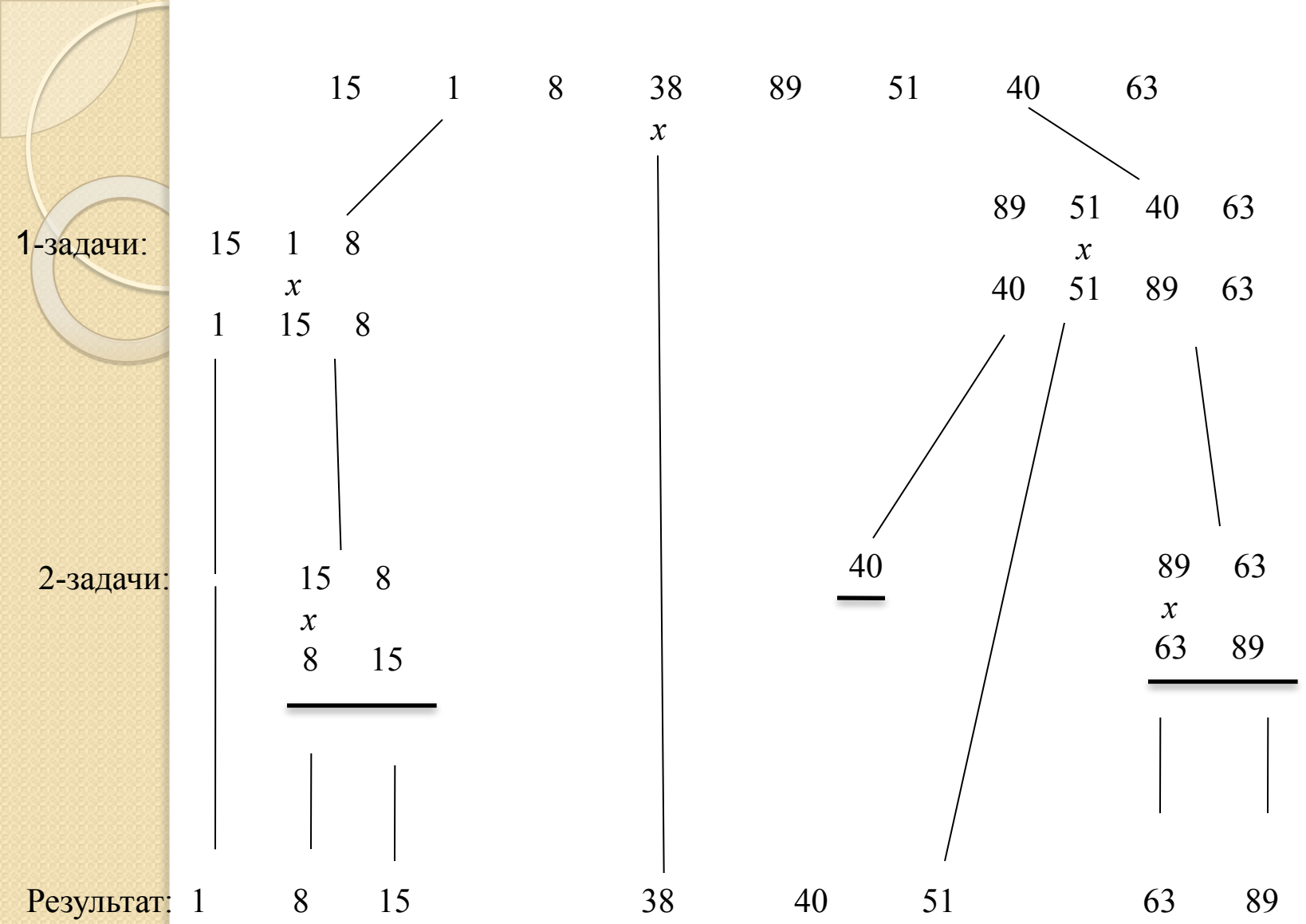
```
Обмен
           15     51     8     38     89     1     40     63
Второй проход:       i                          j
Сближение
```



```
Обмен
                i                          j
               15      1     8     38     89     51    40     63
Третий проход:          i           j
Сближение                        ij
```



```
Обмен
               15     1      8     38     89     51    40     63
Четвертый проход:       j           i
Сближение
```


15 1 8 38 89 51 40 63
x
1-задачи:
15 1 8
x
1 15 8
89 51 40 63
x
40 51 89 63
2-задачи:
15 8
x
8 15
40
89 63
x
63 89
Результат: 1 8 15 38 40 51 63 89

Разделение

Анализ

Процессу разбиения подвергается весь массив, следовательно выполняется N сравнений.

Число обменов?

Пусть после разделения x будет занимать в массиве позицию k.

Число требующихся обменов равно числу элементов в левой части массива (k - 1), умноженному на вероятность того, что элемент нужно обменять.

Элемент обменивается, если он не меньше, чем x.

Вероятность этого равна (N - (k - 1))/N.

Анализ, продолжение

Просуммируем всевозможные варианты выбора медианы и разделим эту сумму на *N,* в результате получим ожидаемое число обменов:

$$M = \frac{1}{N} \cdot \left[\sum_{i=1}^{N} (x-1) \cdot \frac{(N-x+1)}{N} \right] = \frac{N}{6} - \frac{1}{6 \cdot N}.$$

Ожидаемое число обменов равно приблизительно *N*/6.

В лучшем случае каждое разделение разбивает массив на две равные части, а число проходов, необходимых для сортировки, равно $log_2 N$.

Тогда общее число сравнений равно $N \log_2 N$.

Анализ, продолжение

Однако в худшем случае сортировка становится «медленной».

Например, когда в качестве пилотируемого элемента всегда выбирается наибольшее значение. Тогда в результате разбиения в левой части оказывается N - 1 элемент, т. е. массив разбивается на подмассивы из одного элемента и из N - 1 элемента.

В этом случае вместо $\log_2 N$ разбиений необходимо сделать ~ N разбиений.

В результате в худшем случае оценка оказывается ~ N^2, что гораздо хуже пирамидальной сортировки.