Быстрая сортировка (продолжение), массив из 12 элементов, сортировка в пошаговом режиме.

0	1	2	3	4	5	6	7	8	9	10	11
90	100	20	60	80	110	120	40	10	30	50	70
					1						
30	10	20	60	40	50	70	120	80	100	90	110
		2									
30	10	20	40	50	60	70	120	80	100	90	110
	3										
30	10	20	40	50	60	70	120	80	100	90	110
4											
10	20	30	40	50	60	70	120	80	100	90	110
5		6			7			8			
10	20	30	40	50	60	70	90	80	100	110	120
							9				
10	20	30	40	50	60	70	90	80	100	110	120
							10				
10	20	30	40	50	60	70	80	90	100	110	120
								11			12

Быстрая сортировка (продолжение)

Алгоритм выполняет разбиение, меняет местами опорный элемент с элементами между двумя группами, сортирует одну группу(с использованием многочисленных рекурсивных вызовов), а затем переходит к сортировке большей группы.

На рисунке представлена вся последовательность сортировки.

Горизонтальная линия под массивами показывает, какой подмассив подвергается разбиению на каждом шаге, а числа(красным) – в каком порядке создаются группы.

Перемещение опорного элемента (выделен желтым) на положенное место - это итоговая позиция опорного элемента, изменяться уже не будет.

Горизонтальные линии под отдельной ячейкой (шаги 5,6,7,11,12) обозначают вызовы QuickSort() для базовых ограничений, такие вызовы немедленно возвращают управление.

В некоторых случаях (шаги на 4 и на 10) опорный элемент оказывается в своей исходной позиции на правом крае сортируемого массива.

В подобной ситуации остается отсортировать только один подмассив: тот, что расположен слева от опорного элемента.

Второй (правый) подмассив отсутствует.

Быстрая сортировка (продолжение)

Как и при разбиении множества книг на книжной полке, мы по одному разу сравниваем каждый элемент с опорным и выполняем не более одного обмена для каждого элемента, который сравниваем с опорным.

Поскольку и каждое сравнение, и каждый обмен занимают константное время, общее время работы процедуры *Partition* с n-элементным подмассивом равно $\Theta(n)$.

Каково время работы процедуры Quicksort?

Время работы быстрой сортировки зависит от того, как именно выполняется разбиение.

Математический анализ этого вопроса достаточно сложен, если элементы входного массива располагаются в случайном порядке, то в среднем получаем разделения, достаточно близкие к разбиениям пополам, так что быстрая сортировка имеет при этом время работы $\Theta(n*\ln(n))$.

Наихудший случай по времени – когда массив отсортирован в обратном порядке, тогда $\Theta(n^2)$.

Быстрая сортировка

Массив для сортировки может быть по-разному организован первоначально.

Допустимо не всегда выбирать в качестве опорного последний элемент.

Но тогда процедура *Partition* не будет работать, группы элементов окажутся не на своих местах.

Это не проблема, достаточно перед выполнением процедуры *Partition* поменять местами последний элемент A[r] с некоторым произвольно выбранным элементом из A[p..r]. Теперь опорный элемент выбран случайным образом, так, что далее можно запускать обычную процедуру *Partition.*

Можно вместо случайного выбора одного элемента из A[p..r] выбрать три случайных элемента, далее находится их медиана и меняется местами с A[r].

Быстрая сортировка

Определение медианы по трем точкам

Под медианой трех элементов подразумевается тот элемент, значение которого находится между двумя другими (если два или более из случайно выбранных элементов равны, медиана выбирается произвольно).

Самое компромиссное решение - выбор медианы по первому, последнему и среднему элементам в массиве.

Определение медианы по трем точкам происходит намного быстрее, чем определение по всем элементам массива.

При этом оно успешно избегает выбора наибольшего или наименьшего элемента в тех случаях, когда данные уже отсортированы в прямом или обратном порядке.

Быстрая сортировка

Часто при выборе медианы выполняется операция сортировки трех элементов, использованных в процессе выбора.

После сортировки трех элементов и выбора медианы в качестве опорного значения точно известно, что элемент у левого края подмассива меньше опорного значения (либо равен ему), а элемент у правого края больше опорного значения (или равен ему).

Медиана равна 44.

Левый край				Центр				Правый край	
44				86				29	
Левый край				Центр				Правый край	
29				44				86	

Сортировка Шелла

Быстрая сортировка и сортировка методом Шелла относятся к нетривиальным алгоритмам.

Оба работают значительно быстрее простых алгоритмов (пузырьковая, методом выбора, вставок).

Сортировка Шелла выполняется за время $O(n*(\ln(n))^2)$,

быстрая сортировка $O(n*\ln(n))$.

В отличие от сортировки слиянием, ни один из этих алгоритмов не требует значительных затрат памяти.

Сортировка Шелла

- Алгоритм назван в честь Дональда Шелла – специалиста в области компьютерных технологий, который опубликовал описание этого способа сортировки в 1959 году.
- Алгоритм Шелла основан на сортировке методом вставок, но обладает новой особенностью, кардинально улучшающей скорость сортировки.
- Сортировка Шелла хорошо подходит для массивов среднего размера - например, до нескольких тысяч элементов.

Сортировка Шелла

Сортировка методом вставок: слишком много копирования.

В ходе выполнения вставки элементы слева от маркера обладают внутренней упорядоченностью (то есть отсортированы в пределах своего набора), а элементы справа не упорядочены.

Алгоритм извлекает элемент, на который указывает маркер, и сохраняет его во временной переменной.

Затем, начиная с элемента слева от освободившейся ячейки, он сдвигает отсортированные элементы вправо, пока не появится возможность вставки содержимого временной переменной с сохранением порядка сортировки.

И в этом процессе кроется основной недостаток сортировки методом вставок.

Если у правого края массива находится наименьший элемент, это потребует выполнения ***n*** операций копирования всего для одного элемента.

Эффективность сортировки вставками - $O(n^2)$.

Сортировка Шелла

Быстродействие можно улучшить, если бы меньший элемент можно было сдвинуть к левому краю массива без сдвига промежуточных элементов.

Сортировка Шелла выполняет «дальние» сдвиги, сортируя разобщенные элементы посредством сортировки методом вставок.

В методе Шелла выполняется сравнение нескольких элементов, находящихся на заданном расстоянии друг от друга.

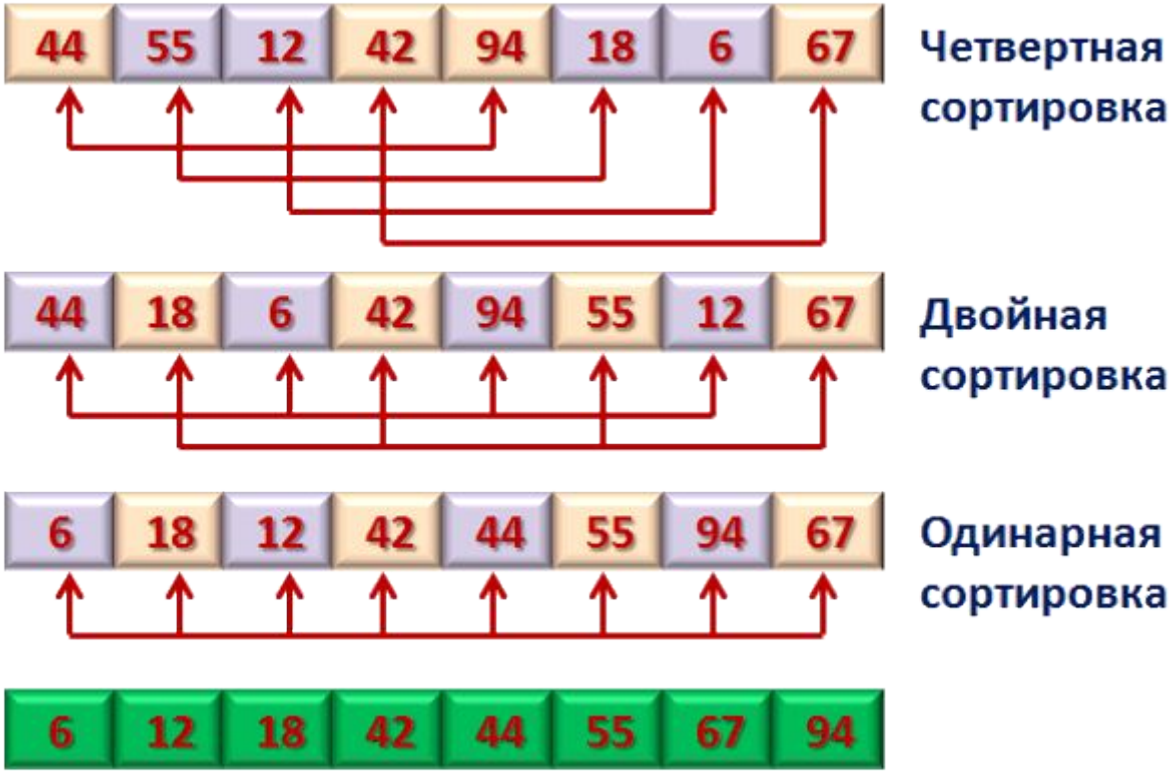
Расстояние между элементами при такой сортировке называется приращением и традиционно обозначается буквой ***h.***

Сначала этих элементов для сравнения несколько.

Затем величина этого интервала постепенно сокращается, увеличивается количество элементов для сравнения, но и элементы массива уже оказываются почти упорядочены.

На последнем этапе можно выполнить обычную сортировку вставкой, которая очень эффективна на почти отсортированных данных.

Сортировка Шелла



Сортировка Шелла

На слайде 11 величина интервала в начале равна четырем.

Элементы, находящиеся на заданном расстоянии, сравниваются друг с другом, и, в случае необходимости меняются местами.

Затем расстояние сокращается вдвое.

Теперь поочередно сортируются элементы на расстоянии, равном двум.

Этот процесс выполняется до тех пор, пока расстояние не станет равным одному.

Процесс завершается, и на выходе мы получаем отсортированный массив.

Сортировка Шелла

Как выбрать величину интервала?

Исходя из чего делать этот выбор?

Каждый проход в алгоритме характеризуется смещением h_i таким, что сортируются элементы, отстающие друг от друга на h_i позиций.

В примере использован интервал (N – размер массива):

$h_t = N / 2,\ h_{t-1} = h_t / 2,\ \ldots\ldots,\ h_0 = 1.$

Возможны и другие смещения, но $h_0 = 1$ всегда.

Сортировка Шелла

Не всегда размер массива делится на 2.

Например, в массиве из 1000 элементов может быть проведена

364-сортировка, затем 121-сортировка, 40-сортировка,

13-сортировка, 4-сортировка и наконец, 1-сортировка.

Серия чисел, используемых при генерировании интервалов

(в данном примере 364, 121, 40, 13, 4, 1),

называется **интервальной последовательностью**.

Приведенная интервальная последовательность, приписываемая Кнуту, весьма популярна.

Сортировка Шелла

В своей книге *Искусство программирования т.3*, посвященной сортировке, *Д. Кнут* предлагает следующую формулу:

h = 3*h + 1, где **h** — величина интервала.

Начальное значение h=1,

начиная с 1, генерируется по рекурсивной формуле.

Интервальная последовательность Кнута:

h	3*h+1	(h-1)/3
1	4	
4	13	1
13	40	4
40	121	13
121	364	40
364	1093	121
1093	3280	364

Сортировка Шелла

Исходное значение **h** можно определить, выполнив данную операцию в цикле до превышения значения размера массива.

Исследуем, как работает сортировка Шелла с использованием последовательности Кнута.

В алгоритме сортировки сначала в коротком цикле вычисляется исходный интервал:

…***while(h <= length /3)***

h = 3*h +1;

….

В качестве первого значения ***h*** используется 1, а затем по формуле генерируется последовательность 1, 4, 13, 40, 121, 364 и т.д.

Процесс завершается тогда, когда величина интервала превышает размер массива.

Для массива из 1000 элементов число 1093 в последовательности оказывается слишком большим, соответственно процесс сортировки начинается с числа 364.

Сортировка Шелла

Затем при каждой итерации внешнего цикла метода сортировки интервал сокращается по формуле, обратной по отношению к приведенной:

$$h = (h — 1)/3$$

Полученные числа приведены в третьем столбце таблицы.

По обратной формуле генерируется обратная последовательность

364, 121, 40, 13, 4, 1.

Каждое из этих чисел используется для сортировки массива.

На каждом этапе сравниваются элементы, находящиеся на расстоянии ***h*** (постепенно сдвигаясь вправо).

При необходимости меняются элементы местами.

Выполнение 1-сортировки завершает работу алгоритма.

Сортировка Шелла

Например (фрагмент функции):

```
void shellSort(int *a, int length)
{
    int h;
    int temp;
    int left, right;
    while(h <= length/3)
        h = 3*h + 1;

     //уменьшение расстояния
     for (h; h > 0; h = (h - 1)/3)
          // …. и т.д.
}
```

Сортировка Шелла

Например, в массиве из 1000 элементов может быть проведена сортировка с использованием серии чисел, получаемых при генерировании интервалов (364, 121, 40, 13, 4, 1).

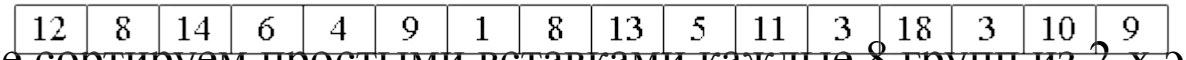
Данная серия называется **интервальной последовательностью**.

Если бы мы делили значение размерности массива пополам:

500, 225, 112, 56, 28, 14, 7, 3, 1.

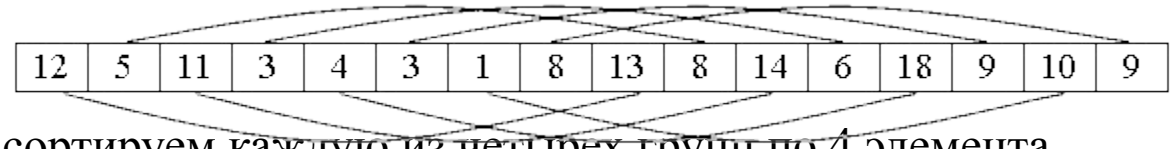
Сортировка Шелла

Рассмотрим следующий алгоритм сортировки массива a[0].. a[15].



1.Вначале сортируем простыми вставками каждые 8 групп из 2-х элементов

(a[0], a[8]), (a[1], a[9]), ... , (a[7], a[15]).



2.Потом сортируем каждую из четырех групп по 4 элемента

(a[0], a[4], a[8], a[12]), ..., (a[3], a[7], a[11], a[15]).

3.Далее сортируем 2 группы по 8 элементов, начиная с

(a[0], a[2], a[4], a[6], a[8], a[10], a[12], a[14]).

4. В конце сортируем вставками все 16 элементов.

Сортировка Шелла

индекс	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	12	8	14	6	4	9	1	8	13	5	11	3	18	3	10	9
шаг	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
8=16/2																
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
результат	12	5	11	3	4	3	1	8	13	8	14	6	18	9	10	9
4=8/2	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
результат	4	3	1	3	12	5	11	6	13	8	10	8	18	9	14	9
2=4/2	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
результат	1	3	4	3	11	5	12	6	10	8	13	8	14	9	18	9
1=2/2																
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
результат	1	3	3	4	5	6	8	8	9	9	10	11	12	13	14	18

Отсортированный массив:

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

Сортировка Шелла

Так, на рисунке величина интервала в начале равна восьми.

Элементы, находящиеся на заданном расстоянии, сравниваются друг с другом, и, в случае необходимости меняются местами.

Затем расстояние сокращается вдвое.

Теперь поочередно сортируются элементы на расстоянии, равном четырем.

Этот процесс выполняется до тех пор, пока расстояние не станет равным одному.

Процесс завершается, и на выходе получаем отсортированный массив.

Сортировка Шелла

Очевидно, лишь последняя сортировка необходима, чтобы расположить все элементы по своим местам.

Так зачем нужны остальные ?

На самом деле они продвигают элементы максимально близко к соответствующим позициям, так что в последней стадии число перемещений будет весьма невелико.

Последовательность и так почти отсортирована.

Ускорение подтверждено многочисленными исследованиями и на практике оказывается довольно существенным.

Сортировка методом пузырька (сортировка обменом)

Для сортировки n-элементного массива A методом пузырька требуется до n-1 проходов.

В каждом проходе сравниваются соседние элементы, и если первый из них больше или равен второму, эти элементы меняются местами.

К моменту окончания каждого прохода наименьший элемент поднимается к вершине текущего подсписка, подобно пузырьку воздуха в кипящей воде.

Простейший вариант сортировки обменом приведен в следующей программе:

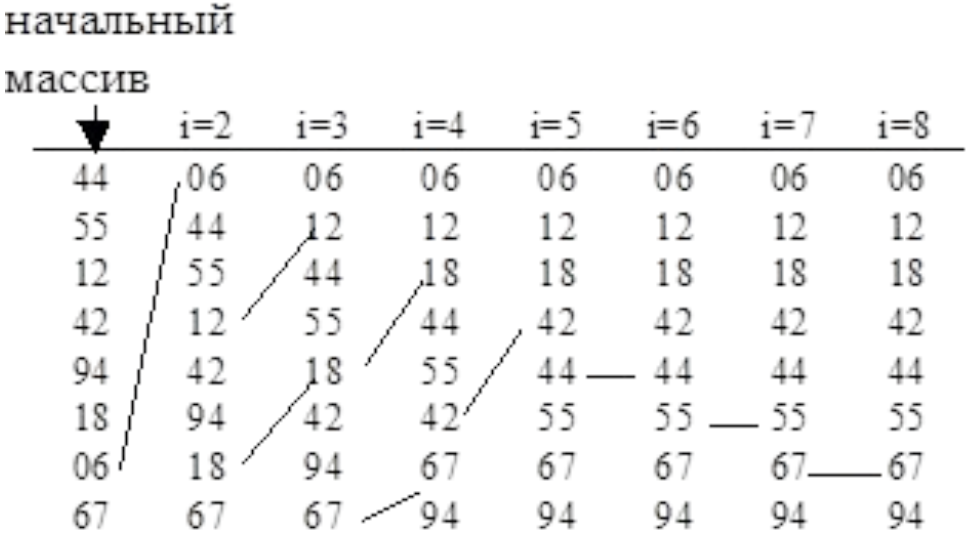
```
for (i=1;i<n;i++)
 for(j=n-1;j>=i;j--)
   if (a[j]<a[j-1]) {temp=a[j];
                     a[j]=a[j-1];
a[j-1]=temp;}
```

Сортировка методом пузырька (сортировка обменом)

Рассмотрим массив, расположенный вертикально.

Каждый проход по массиву приводит к «всплыванию» пузырька на соответствующий его весу уровень.

В данном примере три последних прохода никак не влияют на элементы массива, т.к. они уже отсортированы.



Сортировка методом пузырька

Алгоритм можно улучшить с учетом того, что если в очередном проходе не было обмена, то последовательность упорядочена.

Очевидный способ улучшить данный алгоритм – это запоминать, производился ли на данном проходе какой-либо обмен.
Если нет, то это означает, что алгоритм может закончить работу.
Количество проходов массива для его сортировки зависит от того, в каком месте расположен минимальный или максимальный элемент.

Например, в первом примере сортировки пузырьком, минимальный элемент поднимается наверх за один проход, а максимальный опускается вниз за несколько проходов.

Массив 2 3 4 5 6 1 будет отсортирован за 1 проход,

а сортировка последовательности 6 1 2 3 4 5 потребует 5 проходов.

Можно еще улучшить алгоритм путем смены направлений следующих один за другим проходов.

Этот алгоритм называется ***Шейкер-сортировкой***.

Шейкер-сортировка (сортировка перемешиванием)

Пределы той части массива в которой есть перестановки, сужаются.

Внутренние циклы проходят по массиву то в одну, то в другую сторону, поднимая самый легкий элемент вверх и опуская самый тяжелый элемент в самый низ за одну итерацию внешнего цикла.