Сортировка массивов

Достаточно часто при составлении программ на одном из этапов работы возникает задача сортировки массива: требуется расположить его элементы по возрастанию или по убыванию. Разработано множество алгоритмов её решения. Ниже рассматриваются некоторые из них.

Постановка задачи.

Дан массив *a*, элементы которого пронумерованы от 1 до N включительно. Требуется упорядочить элементы этого массива по возрастанию.

Сортировка с помощью прямого выбора

Идея этого метода сортировки состоит в следующем.

1. Начиная с первого элемента массива, найти наименьший и поменять местами с первым.
2. Начиная со второго элемента массива, найти наименьший и поменять местами со вторым.
3. …
4. Начиная с предпоследнего элемента массива, найти наименьший и поменять местами с предпоследним.

Пример.

3	7	15	5	9	11	2

Наименьший элемент 2 меняем местами с первым.

2	7	15	5	9	11	3

Наименьший элемент теперь 3. Меняем местами со вторым.

2	3	15	5	9	11	7

5 ставим на место третьего элемента, а третий на место пятёрки.

2	3	5	15	9	11	7

Далее

2	3	5	7	9	11	15
2	3	5	7	9	11	15
2	3	5	7	9	11	15

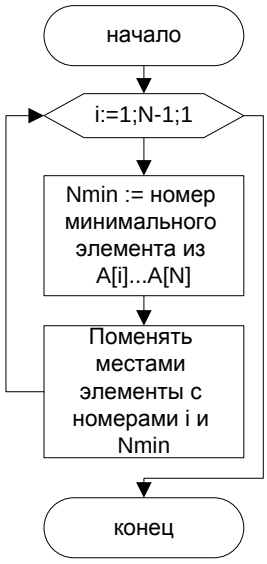

Рисунок 1

Получается следующий набросок алгоритма.
Пока ещё команды этого алгоритма выглядят весьма туманно. Проясним их.

Задачу поиска минимального элемента массива мы уже решали на прошлом занятии. Только теперь нам нужно кроме значения минимального элемента массива запоминать ещё и его номер Nmin. И начинать надо не всегда с первого элемента.

Как поменять местами два элемента массива? Так же, как и значения двух простых переменных *x* и *y*. Или так же, как обменять содержимое двух стаканов: с чаем и с молоком. Можно воспользоваться третьим стаканом.

z:=*x*;
x:=*y*;
y:=*z*

Теперь алгоритм можно записать в окончательной форме. Ввод и вывод массива и его размера не показываем.

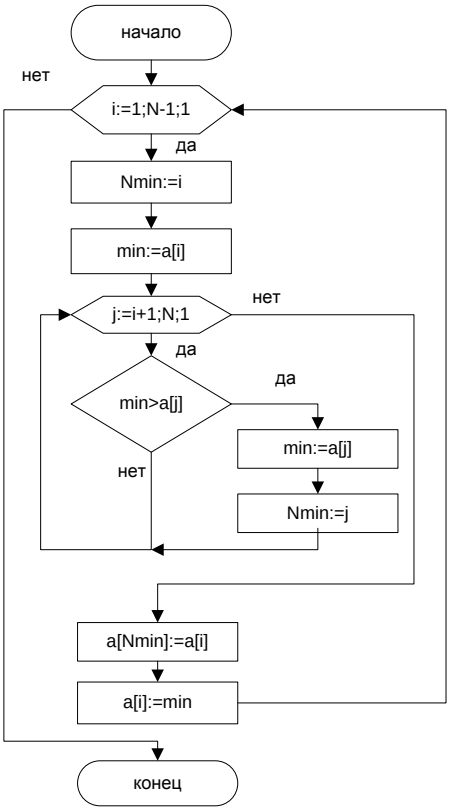

Рисунок 2. Сортировка с помощью прямого выбора

Обратите внимание на метод, при помощи которого составлялся этот алгоритм. Сначала пишется эскиз алгоритма на псевдоязыке. Он может содержать неформальные команды, непонятные исполнителю. Потом такие команды постепенно уточняются, «выкристаллизовываются». Окончательная версия алгоритма будет содержать только допустимые команды исполнителя. Те, которые можно записать на избранном языке программирования. Этот приём называют методом последовательной детализации или пошаговой кристаллизацией[1] алгоритма.

Напишем программу. При этом учтём, что алгоритм от типа данных элементов массива не зависит. Лишь бы для них были определены операции сравнения. Именно с этой целью введён пользовательский тип данных TElem. Переопределяем его, и можно использовать ту же программу для массива уже другого типа.

[1] Ахо А., Хопкрофт Д., Ульман Д. Структуры данных и алгоритмы. — Москва, Санкт-Петербург, Киев: издательский дом «Вильямс», 2001. Стр. 20

```
{$mode delphi}
Program StraightSel;
Const  MAXN = 100;

Type TElem = Integer;

Var a             : array[1..MAXN] of TElem;
    i, j, N, Nmin : Integer;
    min           : TElem;
Begin
  Write('Количество элементов массива (не больше ', MAXN, ') ');
  Read(N);
  If N > MAXN Then
  Begin
     WriteLn('Это слишком много');
     Halt
  End;
  WriteLn('Введите ',N, ' чисел');
  For i:=1 To N Do
  Read(a[i]);

  (* Сортировка прямым выбором *)
  For i:=1 To N-1 Do
  Begin
    Nmin:=i;
    min:=a[i];
    For j:=i+1 To N Do
    If min > a[j] Then
    Begin
      min:=a[j];
      Nmin:=j
    End;
    a[Nmin]:=a[i];
    a[i]:=min
  End;

  (* Вывод отсортированного массива *)
  For i:=1 To N Do
  Write(a[i],' ');

  WriteLn
End.
```

Сортировка с помощью прямого включения

Пусть элементы массива от a_1 до a_{i-1} уже расположены в нужном порядке: $a_1 \le a_2 \le a_3 \le \ldots \le a_{i-1}$.

1. Берём элемент a_i и ищем для него место среди уже упорядоченных. Пусть это место номер j.
2. Сдвигаем все элементы, начиная с a_j и заканчивая a_{i-1}, на одну позицию вправо, а элемент a_i помещаем на место a_j. Теперь упорядочены уже первые i элементов: $a_1 \le a_2 \le \ldots \le a_{i-1} \le a_i$.
3. Берём элемент с номером $i+1$ и повторяем для него ту же последовательность действий.

Вот пример.

3	7	15	5	9	11	2

Первые три элемента расположены в правильном порядке. Место четвёртому ($i = 4$), выделенному красным цветом, между первым и вторым ($j = 2$).

3	5	7	15	9	11	2

Ищем место девятке.

3	5	7	9	15	11	2

Размещаем 11.

3	5	7	9	11	15	2

И, наконец, подыскиваем место последнему элементу, двойке.

2	3	5	7	9	11	15

Теперь массив упорядочен.

А бы если первые элементы не были упорядочены? Но ведь массив из одного элемента обязательно упорядочен! То есть в начале надо взять $i = 2$ и повторять процесс до $i = N$ включительно.

Заметим, что нет необходимости сначала искать место новому элементу, а потом сдвигать элементы с номерами от j до $i-1$. Можно эти процессы проводить одновременно, как показано на блок-схеме рис. 3.

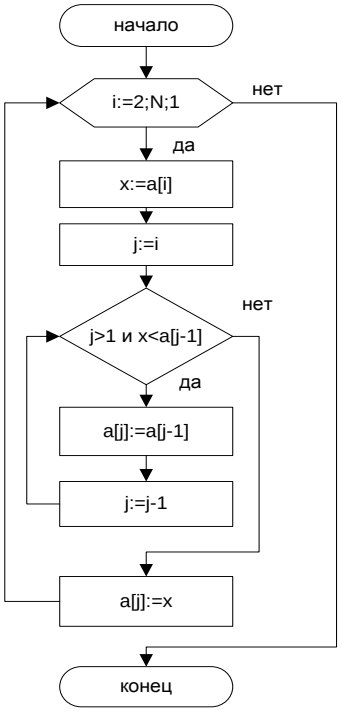

Рисунок 3. Сортировка прямым включением

Здесь новый элемент $x = a_i$ либо остаётся на своём месте, если сразу $x \geq a_{i-1}$, либо x «просеивается» на нужное место в цикле с предусловием.

Обратите внимание на дополнительное условие $j > 1$ в заголовке этого цикла. Оно позволяет предотвратить выход за пределы массива. Подумайте, что произойдёт в процессе поиска места для последнего элемента массива (двойки) из приведённого выше примера, если исключить из алгоритма проверку $j > 1$. Однако предотвращения выхода за пределы массива можно было бы добиться и без дополнительного сравнения и вычисления логического умножения.

Вот как это можно сделать.

1. Увеличиваем размер массива на один элемент, начиная нумерацию с нуля.
2. На место a_0 всегда помещаем элемент, равный x.

Тогда выражение $x < a_{j-1}$ обязательно будет истинно при некотором значении j из промежутка $1 \leq j \leq i$, даже если x окажется строго меньше любого из элементов от a_{i-1} до a_1. Таким образом,

нулевой элемент массива будет играть роль барьера, предотвращающего выход цикла за пределы массива. За счёт отсутствия лишних сравнения и вычисления значения логического умножения алгоритм окажется более быстрым.

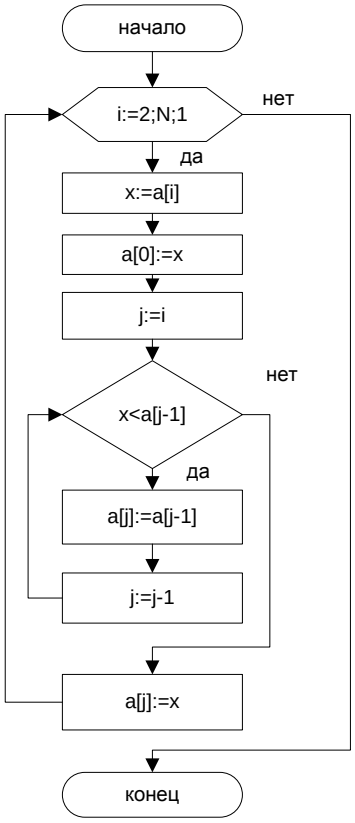

Рисунок 4. Сортировка прямым включением с барьерным элементом

Напишем программу по этому алгоритму.

```
{$mode delphi}
Program StraightIns;
Const  MAXN = 100;

Type TElem = Integer;

Var a             : array[0..MAXN] of TElem;
    i, j, N       : Integer;
    x             : TElem;
Begin
  Write('Количество элементов массива (не больше ', MAXN, ') ');
  Read(N);
  If N > MAXN Then
  Begin
    WriteLn('Это слишком много');
    Halt
  End;
  WriteLn('Введите ', N, ' чисел');

  For i:=1 To N Do
  Read(a[i]);

  (* Сортировка прямым включением *)
  For i:=2 To N Do
  Begin
    x:=a[i];
    a[0]:=x;  //Установка барьера
```

```
    j:=i;
    While x < a[j-1] Do
    Begin
      a[j]:=a[j-1];
      j:=j-1
    End;
    a[j]:=x
  End;

  (* Вывод отсортированного массива *)
  For i:=1 To N Do
  Write(a[i],' ');

  WriteLn
End.
```

Пузырьковая сортировка, или сортировка с помощью прямого обмена

Разобьём массив на пары: первый элемент образует пару со вторым, второй с третьим, третий с четвёртым и т.д. Будем последовательно просматривать каждую пару, начиная с первой, и, если первый элемент пары больше второго, менять их местами. В итоге максимальный элемент окажется в конце массива.

Пример. Исходный массив

3	7	15	5	9	11	2

Первая пара: 3 и 7 — расположена в правильном порядке. Изменений в массив не вносим.
Вторая пара: 7 и 15 — тоже расположена правильно.
Третья пара. 15 > 5. Меняем их местами. Получим

3	7	5	15	9	11	2

Пара 15 и 9. Тоже меняем местами.

3	7	5	9	15	11	2

Теперь обмен в паре 15 и 11.

3	7	5	9	11	15	2

В последней паре 15>2. Опять обмен.

3	7	5	9	11	2	15

Максимальный элемент массива 15 оказался в последней ячейке.

Повторим ту же последовательность шагов для массива из $N-1$ элемента. В результате его максимальный элемент окажется в ячейке под номером $N-1$. Потом всё то же для массива из $N-2, N-3, \ldots, 2$ элементов. Массив из N элементов окажется отсортированным по возрастанию.

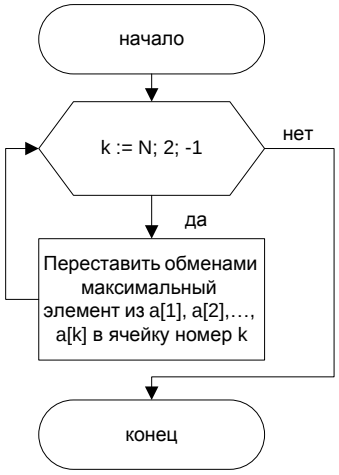

Рисунок 5

То есть в первом приближении алгоритм сортировки выглядит так, как показано на рис. 5.
Осталось конкретизировать перестановку обменами. Здесь надо будет для $i = 1, 2, \ldots, k-1$ сравнивать a_i с a_{i+1} и, в случае $a_i > a_{i+1}$, обменивать их значения через третью переменную. Получаем следующий алгоритм пузырьковой сортировки (рис. 6).

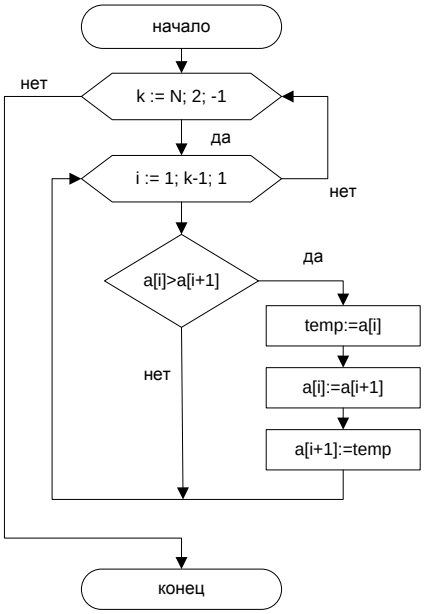

Рисунок 6. Пузырьковая сортировка

Напишем программу.

```
{$mode delphi}
Program BubbleSort;
Const  MAXN = 100;

Type TElem = Integer;

Var a             : array[1..MAXN] of TElem;
    i, k, N       : Integer;
    temp          : TElem;
Begin
  Write('Количество элементов массива (не больше ', MAXN, ') ');
  Read(N);
  If N > MAXN Then
  Begin
     WriteLn('Это слишком много');
     Halt
  End;
  WriteLn('Введите ', N, ' чисел');

  For i:=1 To N Do
  Read(a[i]);

  (* Пузырьковая сортировка *)
  For k:=N DownTo 2 Do
  For i:=1 To k-1 Do
  If a[i] > a[i+1] Then
  Begin
    temp:=a[i];
    a[i]:=a[i+1];
    a[i+1]:=temp
  End;

  (* Вывод отсортированного массива *)
  For i:=1 To N Do
  Write(a[i],' ');
```

```
  WriteLn
End.
```

Почему сортировка названа пузырьковой? Если рассматривать массив как вертикальный столбец, элементы которого нумеруются снизу вверх, а элементы массива как пузырьки воздуха в бочке с водой, то процесс сортировки будет напоминать всплывание этих пузырьков.

Выше были рассмотрены прямые методы сортировки. Все они являются квадратичными. Это значит, что количество действий (сравнений элементов и их перестановок), выполняемых в процессе сортировки, является квадратичной функцией от N (количества элементов массива). Следовательно, при больших значениях N эти алгоритмы становятся неэффективными из-за большого времени их работы.

Чтобы на практике сравнить эффективность написанных программ, требуется запустить их на выполнение при различных исходных данных, в том числе надо использовать массивы достаточно большой размерности. Вводить эти массивы с клавиатуры очень неудобно. Лучше создать текстовые файлы с исходными данными и ввод/вывод производить из них.

Ввод/вывод данных из текстовых файлов

Для работы с файлом в программе на Паскале надо

1. описать переменную типа Text:
   var f : Text;
2. связать эту переменную с именем файла вызовом процедуры Assign(f,'filename.txt');
3. открыть файл
   a. для чтения Reset(f). Файл filename.txt должен существовать;
   b. для записи Rewrite(f). При этом создаётся новый файл. Если файл с именем filename.txt уже существует, он будет уничтожен;
   c. вводить или выводить, в зависимости от режима открытия файла, информацию, используя процедуры Read/ReadLn и Write/WriteLn. При этом первым параметром должна быть переменная типа Text. Например, Read(f,x,y) введёт значения переменных x и y из файла filename.txt.
4. После окончания работы с файлом закрыть файл вызовом Close(f).

Не забывайте закрыть файл, открытый для записи, иначе вполне возможна потеря записанной в него информации. После того, как файл закрыт, ту же файловую переменную можно использовать для работы с новым файлом.

Следующая программа записывает в файл random.txt запрошенное с консоли количество случайных целых чисел, равномерно распределённых в диапазоне от нуля включительно до MAX исключительно. Если написать MAX=MAXLONGINT, то MAX будет равно максимальному числу типа LongInt.

```
{$mode delphi}
Program rnd;
Const MAX = 1000000;
      NL  = 7; //Количество чисел в одной строке файла
Var f       : Text;
    i, N, x : Integer;
Begin
  WriteLn('Количество случайных чисел'); //Это стандартный вывод (на экран)
  Read(N); //Стандартный ввод (с клавиатуры)

  Assign(f,'random.txt'); //Связываем f с файлом random.txt
  Rewrite(f);   //Открываем файл для записи
  WriteLn(f,N); //Вывод значения переменной N в файл random.txt
  Randomize;    //Инициализируем генератор псевдослучайных чисел,
                //чтобы при каждом запуске программы
                 //последовательность чисел не повторялась
```

```
  For i:=1 To N Do
  Begin
    x:=Random(MAX);          //Генерируем случайное число 0 <= x < MAX
    Write(f, x:10, ' ');     //Выводим все x в файл
    If i mod NL = 0
    Then WriteLn(f)          //Переходим к новой строке
  End;
  Close(f)   //Закрываем файл. Это обязательно!!!
End.
```

Теперь переделаем программу сортировки методом прямого включения на работу с файлом. Данные будут вводиться из файла, созданного программой rnd, и выводиться в файл output.txt. В конец файла будет выводиться время выполнения сортировки в секундах. Время засекается с помощью процедуры

gettime(часы, минуты, секунды, сотые доли секунды)

Чтобы её использовать, нужно подключить модуль dos:

Uses dos;

Получается такая программа.

```
{$mode delphi}
Program StraightInsF;
uses dos;     //Подключаем модуль для использования gettime
Const  MAXN = 1000000;
       NL   = 7; //Количество чисел в одной строке выходного файла

Type TElem = Integer;

Var a             : array[0..MAXN] of TElem;
    i, j, N       : Integer;
    x             : TElem;
    f             : Text;
    h1,m1,s1,ss1  : Word;     //Часы, минуты, секунды и сотые доли секунды
    h2,m2,s2,ss2  : Word;
    t1,t2         : Integer;  //Время в сотых долях секунды
Begin
  Assign(f,'random.txt');
  Reset(f);  //Открываем файл для чтения

  Read(f,N); //Вводим количество элементов массива
  If N > MAXN Then
  Begin
   WriteLn('N слишком велико');  //Это вывод на экран
   Halt
  End;

  (* Вводим сам массив *)
  For i:=1 To N Do
  Read(f,a[i]);

  Close(f); //Закрываем файл

  (* Засекаем время начала сортировки *)
  gettime(h1,m1,s1,ss1);

  (* Сортировка прямым включением *)
  For i:=2 To N Do
  Begin
    x:=a[i];
    a[0]:=x;  //Установка барьера
    j:=i;
    While x < a[j-1] Do
    Begin
      a[j]:=a[j-1];
      j:=j-1
    End;
    a[j]:=x
  End;
```

```
  (* Засекаем время окончания сортировки *)
  gettime(h2,m2,s2,ss2);

  (* Переводим время в сотые доли секунды *)
  t1:=(s1+m1*60+h1*3600)*100+ss1;
  t2:=(s2+m2*60+h2*3600)*100+ss2;

  (* Вычисляем промежуток времени *)
  t2:=t2 - t1;
  If t2 < 0  //Вдруг кто сортирует в полночь?
  Then t2:=t2+8640000; //86400 секунд в сутках
  (* Теперь в t2 время сортировки в сотых долях секунды *)


  (* Вывод отсортированного массива *)
  Assign(f,'output.txt'); //Теперь f связана с файлом output.txt
  Rewrite(f);             //Файл output.txt открыт для записи
  For i:=1 To N Do
  Begin
    Write(f,a[i]:10,' ');
    If i mod NL = 0
    Then WriteLn(f)
  End;

  WriteLn(f);   //Переходим на новую строку
  WriteLn(f);   //Бросаем пустую строку

  (* Выводим время в секундах с точностью до сотых *)
  Write(f,'time=',(t2/100):0:2,' s');
  Close(f)  //Закрываем файл

End.
```

На моём компьютере эта программа сортирует массив из 30000 элементов примерно за 1,6 секунды. Точное время мы определить не сможем. Не удивляйтесь, если при двух запусках этой программы на одних и тех же данных вы будете получать результаты, различающиеся на несколько сотых секунды, а при малых N вообще получите 0. Gettime определяет время не точнее, чем до одной сотой секунды. Следовательно, погрешность измерения промежутка времени может доходить до двух сотых секунды. А разница в значениях двух промежутков — до четырёх сотых секунды. Сам gettime тоже выполняется некоторое, пусть и очень малое, время. И даже если бы не было других источников погрешности, точное время сортировки определить было бы невозможно. Но для сравнения эффективности алгоритмов сортировки нам это и не надо.

Переделать другие программы сортировки на работу с файлами вы сможете сами.

Сортировка Шелла, или сортировка с помощью включения с уменьшающимися расстояниями

В 1959 году Дональд Шелл предложил усовершенствование метода сортировки с помощью прямого включения.
Сортируемый массив разделяется на группы равноотстоящих друг от друга элементов. Например, можно разделить массив на $\frac{N}{2}$ групп по 2 элемента. Расстояние между элементами группы $h = \frac{N}{2}$.

44	55	12	42	94	18	6	67

Здесь группы показаны каждая своим цветом: 44 и 94, 55 и 18, 12 и 6, 42 и 67. Отсортируем каждую из групп методом прямого включения. Это четверная сортировка: расстояние между элементами группы $h = 4$. Получим

44	18	6	42	94	55	12	67

Теперь разделим массив на группы, расстояние между элементами которых меньше. Например, в 2 раза. Получаем следующие группы: {44, 6, 94, 12} и {18, 42, 55, 67}. Снова сортируем каждую из групп. Это двойная сортировка. Получим

6	18	12	42	44	55	94	67

Снова уменьшаем расстояние h между элементами группы в 2 раза. Теперь оно равно единице. Выполняем обычную сортировку прямым включением.

6	12	18	42	44	55	67	94

Задавать начальное распределение в группы и в дальнейшем уменьшать расстояние между его элементами можно было и любым другим способом. Главное, чтобы расстояние между элементами последней группы равнялось единице. Сортировка массива всё равно бы была выполнена. В самом плохом случае всю работу выполнил бы последний проход по массиву, т.е. обычная сортировка прямым включением.

Может показаться, что сортировка Шелла только добавляет работы и будет работать медленней сортировки с помощью прямого включения. Однако это не так. Выигрыш в количестве действий, а значит и во времени, достигается за счёт того, что на каждом этапе либо сортируется относительно мало элементов, либо элементы уже довольно хорошо упорядочены и требуется сравнительно немного перестановок.

В общих чертах сортировку Шелла можно представить следующей схемой (рис. 7). Здесь $h_1 \dots h_t$ — последовательность расстояний между элементами.

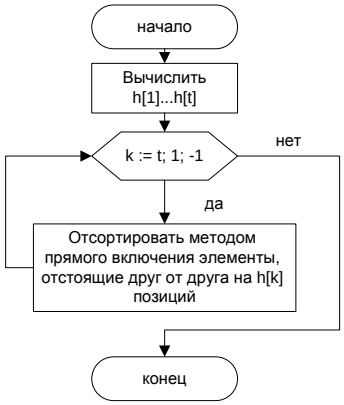

Рисунок 7

Для сортировки методом прямого включения будем использовать слегка изменённую блок-схему рисунка 3. Внешний цикл будем начинать не с 2, а с $h_k + 1$, а во внутреннем цикле всюду вместо $j - 1$ будем использовать $j - h_k$. Условие $j > 1$ заменим на $j > h_k$. Получим блок-схему рис. 8.

Эффективность алгоритма зависит от того, каким образом массив делится на группы. Как это сделать наилучшим образом, не выяснено до сих пор. Однако было доказано, что описанный выше способ выбора расстояний недостаточно эффективен, хотя сам Шелл первоначально предлагал именно его. Дональд Кнут пишет, что если $N < 1000$ самое простое правило наподобие

Принять $h_1 = 1$, $h_{k+1} = 3h_k + 1$ *и остановиться на* h_{t-1}, *когда* $3h_t \geq N$

оказывается не хуже любого другого.

Вот несколько первых членов этой последовательности: 1, 4, 13, 40, 121.

Понятно, что применять полученные расстояния надо будет в порядке, обратном их получению. Если мы будем знать наибольшее значение h, то все остальные вычислим целочисленным делением на 3:

```
h:=h div 3
```

Это следствие записанного выше рекуррентного соотношения.

Количество действий для такого выбора расстояний в худшем

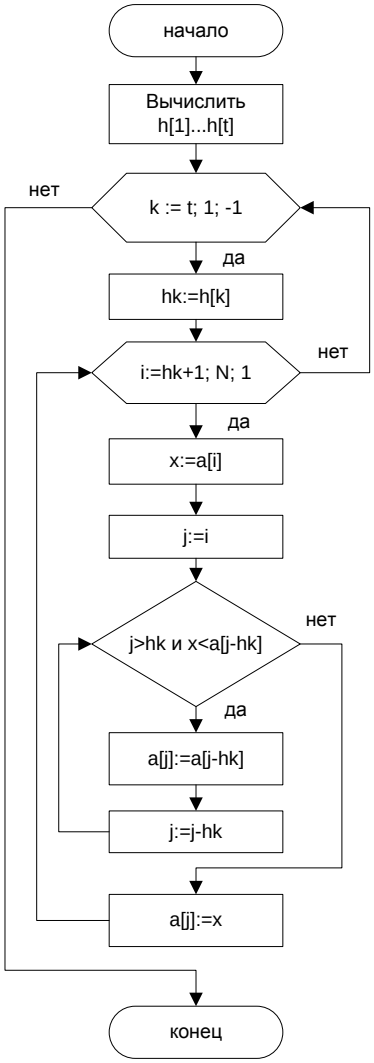

Рисунок 8. Сортировка Шелла

случае будет пропорционально $N^{\frac{3}{2}}$, что лучше, чем для рассмотренных нами ранее прямых методов.

Напишем программу.

```
{$mode delphi}

(* Сортировка Шелла с использованием расстояний h[k+1]=3*h[k]+1 *)

Program Shell;
uses dos;     //Подключаем модуль для использования gettime
Const  MAXN = 1000000;
       NL   = 7; //Количество чисел в одной строке выходного файла

Type TElem = Integer;

Var a             : array[1..MAXN] of TElem;
    i, j, N       : Integer;
    h             : Integer;
    x             : TElem;
    f             : Text;
    h1,m1,s1,ss1  : Word;     //Часы, минуты, секунды и сотые доли секунды
    h2,m2,s2,ss2  : Word;
    t1,t2         : Integer;  //Время в сотых долях секунды
Begin
  Assign(f,'random.txt');
  Reset(f);  //Открываем файл для чтения

  Read(f,N); //Вводим количество элементов массива
  If N > MAXN Then
  Begin
    WriteLn('N слишком велико');  //Это вывод на экран
    Halt
  End;

  (* Вводим сам массив *)
  For i:=1 To N Do
  Read(f,a[i]);

  Close(f); //Закрываем файл

  (* Засекаем время начала сортировки *)
  gettime(h1,m1,s1,ss1);

  (* Сортировка Шелла *)

  (* Вычисляем максимальное h *)
  h:=1;
  If N > 9 Then //Иначе h будет равно 1 или 0
  Begin
    While (3*h) < N Do
      h:=3*h+1;
    h:=h div 3;
  End;

  While h>=1 Do
  Begin
    For i:=h+1 To N Do
    Begin
      x:=a[i];
      j:=i;
      While (j > h) and (x < a[j-h]) Do
      Begin
```

```
        a[j]:=a[j-h];
        j:=j-h
      End;
      a[j]:=x
    End;
    h := h div 3
  End;


  (* Засекаем время окончания сортировки *)
  gettime(h2,m2,s2,ss2);

  (* Переводим время в сотые доли секунды *)
  t1:=(s1+m1*60+h1*3600)*100+ss1;
  t2:=(s2+m2*60+h2*3600)*100+ss2;

  (* Вычисляем промежуток времени *)
  t2:=t2- t1;
  If t2 < 0  //Вдруг кто сортирует в полночь?
  Then t2:=t2+8640000; //86400 секунд в сутках
  (* Теперь в t2 время сортировки в сотых долях секунды *)


  (* Вывод отсортированного массива *)
  Assign(f,'output.txt'); //Теперь f связана с файлом output.txt
  Rewrite(f);             //Файл output.txt открыт для записи
  For i:=1 To N Do
  Begin
    Write(f,a[i]:10,' ');
    If i mod NL = 0
    Then WriteLn(f)
  End;

  WriteLn(f);   //Переходим на новую строку
  WriteLn(f);   //Бросаем пустую строку

  (* Выводим время в секундах с точностью до сотых *)
  Write(f,'time=',(t2/100):0:2,' s');
  Close(f)  //Закрываем файл

End.
```

Попробуйте сравнить её быстродействие с быстродействием программы сортировки прямым включением. На достаточно больших N различие очень заметно. Однако ответ для случайных массивов и уже предварительно упорядоченных будет разным.

Для больших N Кнут рекомендует использовать последовательность, предложенную Седжвиком

$$h_s = \begin{cases} 9 \cdot 2^s - 9 \cdot 2^{\frac{s}{2}} + 1, \text{если } s \text{ чётно} \\ 8 \cdot 2^s - 6 \cdot 2^{\frac{s+1}{2}} + 1, \text{если } s \text{ нечётно} \end{cases}$$

Остановиться следует на h_{t-1}, если $3h_t \geq N$.

Эффективность алгоритма в этом случае ещё выше. Количество действий будет в среднем пропорционально $N^{\frac{7}{6}}$, а в худшем случае $N^{\frac{4}{3}}$.

Задания

1. Предложите способы сортировки массива по убыванию.
2. Одно из возможных усовершенствований пузырьковой сортировки состоит в следующем. Если «пузырьки перестали всплывать» (внутренний цикл не сделал ни одной перестановки), то сортировку можно прекращать. Внесите соответствующие изменения в программу сортировки.

3. Напишите программу сортировки Шелла с использованием последовательности Седжвика.
4. Переделайте все программы сортировки, о которых говорилось на этом занятии, для работы с файлами. Сравните их быстродействие при различных значениях N: маленьких, средних, больших. Доведите размер массива до 100000 и более элементов. Какие из изученных методов сортировки дают наилучший результат для больших массивов? (Однако имейте в виду, что есть и ещё более быстрые методы.) Какой метод наихудший? Используйте различные виды исходных массивов: случайные, уже упорядоченные в прямом и обратном порядке. Свои выводы сделайте отдельно для каждого из них.
5. Составьте программу следующего метода сортировки. Сравниваются два соседних элемента a_i и a_{i+1}. Если $a_i \leq a_{i+1}$, то продвигаются на один элемент вперёд. Если $a_i > a_{i+1}$, то производится перестановка и сдвигаются на один элемент назад, если это возможно. И так пока не будет пройден весь массив.

Литература

1. Д. Кнут. Искусство программирования для ЭВМ. Том 3. Сортировка и поиск. Издание 3. — Издательский дом «Вильямс», 2005.
2. Н. Вирт. Алгоритмы и структуры данных. — СПб.: Невский диалект, 2008.