

ПРОСТЫЕ МЕТОДЫ СОРТИРОВКИ МАССИВОВ

Под *сортировкой* обычно понимают процесс перестановки объектов данного множества в определенном *порядке*. Цель сортировки — облегчить последующий поиск элементов в отсортированном множестве. В этом смысле элементы сортировки присутствуют почти во всех задачах. Упорядоченные объекты содержатся в телефонных книгах, в ведомостях подоходных налогов, в оглавлениях, в библиотеках, в словарях, на складах, да и почти всюду, где их нужно разыскивать. Даже маленьких детей приучают приводить вещи «в порядок», и они сталкиваются с некоторым видом сортировки задолго до того, как узнают что-либо об арифметике.

Следовательно, методы сортировки очень важны, особенно при обработке данных. Казалось бы, что легче рассортировать, чем набор данных? Однако с сортировкой связаны многие фундаментальные приемы построения алгоритмов, которые и будут нас интересовать в первую очередь. Почти все такие приемы встречаются в связи с алгоритмами сортировки. В частности, сортировка является идеальным примером огромного разнообразия алгоритмов, выполняющих одну и ту же задачу, многие из которых в некотором смысле являются оптимальными, а большинство имеет какие-либо преимущества по сравнению с остальными. Поэтому на примере сортировки мы убеждаемся в необходимости сравнительного анализа алгоритмов. Кроме того, здесь мы увидим, как при помощи усложнения алгоритмов можно добиться значительного увеличения эффективности по сравнению с более простыми и очевидными методами.

Основное требование к методам сортировки массивов — экономное использование памяти. Это означает, что переупорядочение элементов нужно выполнять *in situ* (на том же месте) и что методы, которые пересылают элементы из массива a в массив b , не представляют для нас интереса.

Методы, сортирующие элементы *in situ*, можно разбить на три основных класса в зависимости от лежащего в их основе приема:

1. Сортировка включениями.
2. Сортировка выбором.
3. Сортировка обменом.

Сортировка простыми включениями.

Этот метод обычно используют игроки в карты. Элементы (карты) условно разделяются на готовую последовательность a_1, \dots, a_{i-1} и входную последовательность a_i, \dots, a_n . На каждом шаге, начиная с $i = 2$ и увеличивая i на единицу, берут i -й элемент входной последовательности и передают в готовую последовательность, *вставляя* его на подходящее место.

Таблица 1. Пример сортировки простыми включениями

Начальные числа	44	55	12	42	94	18	06	67
$i=2$	44	55	12	42	94	18	06	67
$i=3$	12	44	55	42	94	18	06	67
$i=4$	12	42	44	55	94	18	06	67
$i=5$	12	42	44	55	94	18	06	67
$i=6$	12	18	42	44	55	94	06	67
$i=7$	06	12	18	42	44	55	94	67
$i=8$	06	12	18	42	44	55	67	94

Процесс сортировки включениями показан на примере; восьми случайно взятых чисел (см. табл. 1). Алгоритм сортировки простыми включениями выглядит следующим образом:

for $i := 2$, to n **do**

begin $x := a[i]$;

 «вставить x на подходящее место в $a_1 \dots a_i$ »

end

При поиске подходящего места удобно чередовать сравнения и пересылки, т. е. как бы «просеивать» x , сравнивая его с очередным элементом a_j и либо вставляя x , либо пересылая a_j направо и продвигаясь налево. Заметим, что «просеивание» может закончиться при двух различных условиях:

1. Найден элемент a_j меньший, чем x .
2. Достигнут левый конец готовой последовательности.

Этот типичный пример цикла с двумя условиями окончания дает нам возможность рассмотреть хорошо известный прием фиктивного элемента («барьера»). Его можно легко применить в этом случае, установив барьер $a_0 = x$. (Заметим, что для этого нужно расширить диапазон индексов в описании a до $0, \dots, n$.) Окончательный алгоритм представлен в виде программы 1.

Program *straightinsertion*;

Const $n=100$;

Type *index*= $0..n$;

Var i, j : *index*; x : *integer*;

a : *array*[*index*] of *integer*;

begin

{заполнение массива числами}

.....

{алгоритм сортировки}

for $i := 2$ **to** n **do**

begin $x := a[i]$; $a[0] := x$; $j := i-1$;

while $x < a[j]$ **do**

begin $a[j+1] := a[j]$; $j := j-1$;

end ;

$a[j+1] := x$

end;

{вывод на экран отсортированного массива}

end.

Сортировка простым выбором

Этот метод основан на следующем правиле:

1. Выбирается наименьший элемент.
2. Он меняется местами с первым элементом a_1

Эти операции затем повторяются с оставшимися $n - 1$ элементами, затем с $n - 2$ элементами, пока не останется только один элемент — наибольший.

Этот метод продемонстрирован на тех же восьми числах в табл. 2.

Таблица 2. Пример сортировки простым выбором

Начальные числа	44	55	12	42	94	18	06	67
	06	55	12	42	94	18	44	67
	06	12	55	42	94	18	44	67
	06	12	18	42	94	55	44	67
	06	12	18	42	94	55	44	67
	06	12	18	42	44	55	94	67
	06	12	18	42	44	55	94	67
	06	12	18	42	44	55	67	94

Программу можно представить следующим образом:

for $i := 1$ to $n-1$ **do**

begin «присвоить k индекс наименьшего элемента из $a[i]...$

 ... $a[n]$ »;

 «поменять местами a_i и a_k »

end

Этот метод, называемый *сортировкой простым выбором*, в некотором смысле противоположен сортировке простыми включениями; при сортировке простыми включениями на каждом шаге рассматривается только *один* очередной элемент *входной последовательности* и все элементы *готового массива* для нахождения места включения; при сортировке простым выбором

рассматриваются *все* элементы *входного массива* для нахождения наименьшего элемента, и этот *один* очередной элемент отправляется в *готовую последовательность*. Весь алгоритм сортировки простым выбором представлен в виде программы 3.

```
Program straightselection;
Const n=100;
Type index=1..n;
Var i,j: index; x: integer;
      a:array[index] of integer;
begin
      {заполнение массива числами}

      .....

      {алгоритм сортировки}
for i := 1 to n—1 do
      begin k := i; x := a[i];
        for j := i+1 to n do
          if a[j] < x then
            begin k := j; x := a[j]
            end ;
          a[k] := a[i]; a[i] := x;
        end;
      {вывод на экран отсортированного массива}
end.
```

Сортировка простым обменом

Классификация методов сортировки не всегда четко определена. Оба представленных ранее метода можно рассматривать как сортировку обменом. Однако в этом разделе мы остановимся на методе, в котором обмен двух элементов является основной характеристикой процесса. Приведенный ниже алгоритм сортировки простым обменом основан на принципе сравнения и

обмена пары соседних элементов до тех пор, пока не будут рассортированы все элементы.

Как и в предыдущих методах простого выбора, мы совершаем повторные проходы по массиву, каждый раз просеивая наименьший элемент оставшегося множества, двигаясь к левому концу массива. Если, для разнообразия, мы будем рассматривать массив, расположенный вертикально, а не горизонтально и — при помощи некоторого воображения — представим себе элементы пузырьками в резервуаре с водой, обладающими «весами», соответствующими их значениям, то каждый проход по массиву приводит к «всплыванию» пузырька на соответствующий его весу уровень (см. табл. 3). Этот метод широко

Таблица 3. Пример сортировки методом пузырька

Начальные числа	2-й	3-й	4-й	5-й	6-й	7-й	8-й
44	06	06	06	06	06	06	06
55	44	12	12	12	12	12	12
12	55	44	18	18	18	18	18
42	12	55	44	42	42	42	42
94	42	18	55	44	44	44	44
18	94	42	42	55	55	55	55
06	18	94	67	67	67	67	67
67	67	67	94	94	94	94	94

известен как *сортировка методом пузырька*. Его простейший вариант приведен в программе 4.

Program bubblesort

Const n=100;

Type index=1..n;

Var i,j: index; x: integer;

```

    a:array[index] of integer;

begin
    {заполнение массива числами}

    .....

    {алгоритм сортировки}
for i := 2 to n do
    begin for j:= n downto i do
        if a[j— 1] > a[j] then
            begin x:= a[j-1]; a[j—1]:= a[j]; a[j] := x
            end;
        end;
    end;
{вывод на экран отсортированного массива}
end. {bubblesort}

```

Программа 4. Сортировка методом пузырька.

Внимательный программист заметит здесь странную асимметрию: один неправильно расположенный «пузырек» в «тяжелом» конце рассортированного массива всплывет на место за один проход, а неправильно расположенный элемент в «легком» конце будет опускаться на правильное место только на один шаг на каждом проходе. Например, массив

12 18 42 44 55 67 94 06

будет рассортирован при помощи метода пузырька за один проход, а сортировка массива

94 06 12 18 42 44 55 67

потребует семи проходов.