

СОРТИРОВКА

*Многие же первые будут последними
и последние первыми.*

Ев. от Матфея, 19:30

Содержание

2

- Терминология и постановка задачи
- Алгоритмы внутренней сортировки
- Алгоритмы внешней сортировки

Терминология

3

□ *Задача сортировки*

- *Исходные данные* – последовательность элементов данных (a_1, a_2, \dots, a_n) , для которых введено отношение порядка
- *Результат* – последовательность $(\tilde{a}_1, \tilde{a}_2, \dots, \tilde{a}_n)$, состоящая из тех же элементов, идущих в неубывающем порядке:
$$\tilde{a}_1 \leq \tilde{a}_2 \leq \dots \leq \tilde{a}_n$$

□ *Сортируемый элемент*

- type TElement=record
 key: ...; { *Ключ сортировки* }
 OtherData: ...; { *Другие данные* }
end;
- Упрощение: сортируемый элемент \equiv ключ, имеющий целочисленный тип.

Терминология

4

- *Внутренняя сортировка* – упорядочение последовательности элементов, целиком располагающейся в оперативной памяти, без использования внешней памяти (*сортировка массивов*).
- *Внешняя сортировка* – упорядочение последовательности элементов, которая не может быть целиком расположена в оперативной памяти, с использованием оперативной и внешней памяти (*сортировка последовательных файлов*).

Терминология

5

- *Сложность алгоритма сортировки* – количество элементарных операций (пересылка, сравнение) в данном алгоритме.
- Сложность алгоритма сортировки является функцией от n – количества элементов в последовательности.
- Значение сложности зависит от "качества" элементов в последовательности
 - лучший случай – последовательность уже отсортирована
 - худший случай – последовательность отсортирована в обратном порядке
 - средний случай – последовательность содержит элементы со случайными значениями ключей.

Внутренняя сортировка

6

- Большинство алгоритмов внутренней сортировки предполагают упорядочение массива *на месте* – без использования дополнительных массивов для хранения временных данных.
 - ▣ `const N=...;`
`type TArray=array [1..N] of Integer;`
`procedure Sort(var A: TArray);`
- Элементарный шаг внутренней сортировки – обмен местами двух элементов. Различие алгоритмов – в принципе выбора обмениваемых элементов.

Алгоритмы внутренней сортировки

7

- Сортировка подсчетом
- Глупая сортировка
- Пузырьковая сортировка и ее улучшения
- Сортировка простыми и бинарными вставками
- Гномья сортировка
- Сортировка простым выбором
- Сортировка слиянием
- Быстрая сортировка

Сортировка подсчетом

8

- Используется дополнительный (результатирующий) массив.
- Место элемента в результирующем массиве есть $1 +$ количество элементов, меньших, чем данный элемент.

```
var Res: TArray;  
  
for i:=1 to N do begin  
    k:=LessThan(A, i);  
    while Res[k+1]=A[i] do  
    { Повторяющиеся в A }  
        k:=k-1;  
    Res[k+1]:=A[i];  
end;
```


Глупая сортировка

9

- Осуществляется просмотр от начала массива. Текущий элемент сравнивается со следующим: если следующий меньше, то производится обмен и возврат в начало цикла.

```
i := 0;
while i < N do
  if A[i] > A[i+1] then
    begin
      Swap(A[i], A[i+1]);
      i := 0;
    end
  else
    i := i + 1;
```

Пузырьковая сортировка

10

- Алгоритм состоит в повторяющихся проходах по сортируемому массиву. За каждый проход элементы последовательно сравниваются попарно и, если порядок в паре неверный, выполняется обмен элементов.
- При проходе алгоритма, элемент, стоящий не на своем месте, "всплывает" до нужной позиции как пузырек в воде.

Пузырьковая сортировка

11

№ шага	Массив
0	8 23 5 65 44 33 1 6
1	8 23 5 65 44 33 1 6
	8 23 5 65 44 1 33 6
	8 23 5 65 1 44 33 6
	8 23 5 1 65 44 33 6
	8 23 1 5 65 44 33 6
	8 1 23 5 65 44 33 6
	1 8 23 5 65 44 33 6
2	1 8 23 5 65 44 6 33
	1 8 23 5 65 6 44 33
	1 8 23 5 6 65 44 33
	1 8 23 5 6 65 44 33
	1 8 5 23 6 65 44 33
	1 5 8 23 6 65 44 33

№ шага	Массив
3	1 5 8 23 6 65 33 44
	1 5 8 23 6 33 65 44
	1 5 8 23 6 33 65 44
	1 5 8 6 23 33 65 44
4	1 5 6 8 23 33 44 65
	1 5 6 8 23 33 44 65
	1 5 6 8 23 33 44 65
	1 5 6 8 23 33 44 65
5	1 5 6 8 23 33 44 65
	1 5 6 8 23 33 44 65
	1 5 6 8 23 33 44 65
6	1 5 6 8 23 33 44 65
	1 5 6 8 23 33 44 65
7	1 5 6 8 23 33 44 65

Пузырьковая сортировка (анимация)

12



Пузырьковая сортировка

13

```
for i:=1 to N-1 do
  for j:=1 to N-1 do
    if A[j]>A[j+1] then
      Swap(A[j], A[j+1]);
```

Улучшения пузырьковой сортировки

14

- Сокращение длины просмотра:
 - ▣ "всплытие" можно проводить не для всего массива, а для еще не отсортированной части
- Исключение лишних просмотров массива:
 - ▣ если при проведении очередного "всплытия" не было обменов, то массив уже отсортирован.

Улучшение пузырьковой сортировки (1)

15

```
R:=N;  
{ Правая граница части A,  
  для которой проводится "всплытие". }  
while R>1 do begin  
  for j:=1 to R-1 do  
    if A[j]>A[j+1] then  
      Swap(A[j], A[j+1]);  
  R:=R-1;  
end;
```

Улучшение пузырьковой сортировки (2)

16

```
R:=N;
WasSwap:=True;
{ Если при проведении "всплытия" не было обменов, то массив
уже отсортирован. }
while (R>1) and WasSwap do begin
    WasSwap:=FALSE;
    for j:=1 to R-1 do
        if A[j]>A[j+1] then begin
            Swap(A[j], A[j+1]);
            WasSwap:=True;
        end;
    R:=R-1;
end;
```


Шейкерная сортировка

17

- Попеременно выполнять улучшенную пузырьковую сортировку
 - справа налево (меньшие элементы передвигаются в начало массива) и
 - слева направо (большие элементы передвигаются в конец массива)
 - одновременно изменяя левую и правую границы просмотров массива.

Шейкерная сортировка

18

№ шага	Массив
0	8 23 5 65 44 33 1 6
1	8 23 5 65 44 33 1 6
	8 23 5 65 44 1 33 6
	8 23 5 65 1 44 33 6
	8 23 5 1 65 44 33 6
	8 23 1 5 65 44 33 6
	8 1 23 5 65 44 33 6
	1 8 23 5 65 44 33 6
2	1 8 23 5 65 44 33 6
	1 8 5 23 65 44 33 6
	1 8 5 23 65 44 33 6
	1 8 5 23 44 65 33 6
	1 8 5 23 44 33 65 6
	1 8 5 23 44 33 6 65

№ шага	Массив
3	1 8 5 23 44 6 33 65
	1 8 5 23 6 44 33 65
	1 8 5 6 23 44 33 65
	1 8 5 6 23 44 33 65
	1 5 8 6 23 44 33 65
4	1 5 6 8 23 44 33 65
	1 5 6 8 23 44 33 65
	1 5 6 8 23 44 33 65
	1 5 6 8 23 33 44 65
5	1 5 6 8 23 33 44 65
	1 5 6 8 23 33 44 65
	1 5 6 8 23 33 44 65

Шейкерная сортировка

19

```
L:=2; { Левая граница проведения "всплытия" }
R:=N; { Правая граница проведения "всплытия" }
k:=N; { Индекс последнего обмена }
repeat
  for j:=R downto L do
    if A[j-1]>A[j] then begin
      Swap(A[j-1], A[j]);
      k:=j;
    end;
  L:=k+1;
  for j:=L to R do
    if A[j-1]>A[j] then begin
      Swap(A[j-1], A[j]);
      k:=j;
    end;
  R:=k-1;
until L>R;
```

Сортировка Шелла

20

- Сравнивать (и в случае неупорядоченности обменивать) не соседние элементы, а отстоящие друг от друга на некотором расстоянии d .
- На каждом шаге сортировки уменьшать d до тех пор, пока d не станет равным 1.

Сортировка Шелла

21

№ шага	Массив
0	8 23 5 65 44 33 1 6
1 (сортируются элементы, отстоящие на $d=4$)	8 23 5 65 44 33 1 6 8 23 5 65 44 33 1 6 8 23 1 65 44 33 5 6 8 23 1 6 44 33 5 65
2 (сортируются элементы, отстоящие на $d=2$)	1 23 8 6 44 33 5 65 1 23 8 6 44 33 5 65 1 23 8 6 5 33 44 65 1 23 5 6 8 33 44 65 1 6 5 23 8 33 44 65 1 6 5 23 8 33 44 65 1 6 5 23 8 33 44 65

№ шага	Массив
3 (сортируются элементы, отстоящие на $d=1$)	1 6 5 23 8 33 44 65 1 5 6 23 8 33 44 65 1 5 6 23 8 33 44 65 1 5 6 8 23 33 44 65 1 5 6 8 23 33 44 65 1 5 6 8 23 33 44 65 1 5 6 8 23 33 44 65

Сортировка Шелла (анимация)

22



Сортировка Шелла

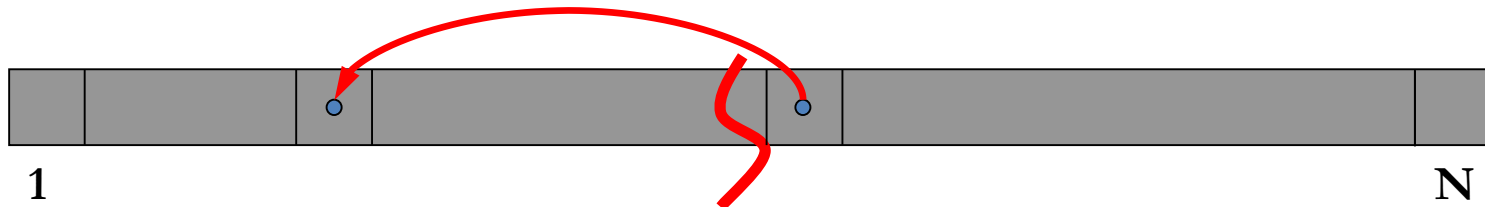
23

```
d:=N div 2;
while d>0 do begin
  for i:=1 to N-d-1 do begin
    j:=i;
    while (j>=1) and (A[j]>A[j+d]) do
      begin
        Swap(A[j],A[j+d]);
        j:=j-1;
      end;
    end;
  d:=d div 2;
end;
```

Сортировка вставками

24

- Массив разбивается на 2 части: левая – упорядоченная, правая – неупорядоченная.
- Шаг алгоритма:
 - взять первый элемент правой части и вставить его в левую часть, сохранив ее упорядоченность;
 - расширить левую (сократить правую) часть на 1 элемент;
 - процесс начинается, когда левая часть состоит из одного элемента a_1 , и завершается, когда правая часть становится пустой.



Сортировка простыми вставками

25

№ шага	Массив
0	8 23 5 65 44 33 1 6
1	8 23 5 65 44 33 1 6
2	8 5 23 65 44 33 1 6 5 8 23 65 44 33 1 6
3	5 8 23 65 44 33 1 6
4	5 8 23 44 65 33 1 6
5	5 8 23 44 33 65 1 6 5 8 23 33 44 65 1 6
6	5 8 23 33 44 1 65 6 5 8 23 33 1 44 65 6 5 8 23 1 33 44 65 6 5 8 1 23 33 44 65 6 5 1 8 23 33 44 65 6 1 5 8 23 33 44 65 6

№ шага	Массив
7	1 5 8 23 33 44 6 65 1 5 8 23 33 6 44 65 1 5 8 23 6 33 44 65 1 5 8 6 23 33 44 65 1 5 6 8 23 33 44 65

Сортировка простыми вставками

(анимация)

26



Сортировка простыми вставками

27

```
for i:=2 to N do begin
  R:=A[i];
  j:=i-1;
  while (R<A[j]) and (j>0) do begin
    { j>0 - "барьер" для j }
    A[j+1]:=A[j];
    j:=j-1;
  end;
  A[j+1]:=R;
end;
```

Сортировка бинарными вставками

28

```
for i:=2 to N do begin
  X:=A[i];
  L:=1; R:=i;
  while L<R do begin
    m:=(L+R) div 2;
    if A[m]<=X then L:=m+1 else R:=m;
  end;
  for j:=i downto R+1 do
    A[j]:=A[j-1];
  A[R]:=X;
end;
```

Гнома сортировка

29

- Садовый гном сортирует линию цветочных горшков.
- Он смотрит на следующий и предыдущий садовые горшки: если они в правильном порядке, он шагает на один горшок вперед, иначе он меняет их местами и шагает на один горшок назад.
- Если нет предыдущего горшка, гном шагает вперед; если нет следующего горшка, гном закончил.



```
Cur:=2; Next:=3;
while Cur<=N do
  if A[Cur-1]>=A[Next] then
    begin
      Cur:=Next;
      Next:=Next+1;
    end
  else begin
    Swap(A[Cur-1],A[Next]);
    Cur:=Cur-1;
    if Cur=1 then begin
      Cur:=Next;
      Next:=Next+1;
    end;
  end;
end;
```

Сортировка простым выбором

30

- Массив разбивается на 2 части: левая – упорядоченная, правая – неупорядоченная.
- Шаг алгоритма:
 - в правой части найти минимальный элемент и поменять местами его и первый элемент;
 - расширить левую (сократить правую) часть на 1 элемент;
 - процесс начинается, когда левая часть пуста, и завершается, когда правая часть состоит из одного элемента \tilde{a}_n .



Сортировка простым выбором

31

№ шага	Массив
0	8 23 5 65 44 33 1 6
1	1 23 5 65 44 33 8 6
2	1 5 23 65 44 33 8 6
3	1 5 6 65 44 33 8 23
4	1 5 6 8 44 33 65 23
5	1 5 6 8 33 44 65 23
6	1 5 6 8 23 44 65 33
7	1 5 6 8 23 33 65 44
8	1 5 6 8 23 33 44 65

Сортировка простым выбором

(анимация)

32



Сортировка простым выбором

33

```
for i:=1 to N-1 do begin
  min:=A[i];
  k:=i;
  for j:=i+1 to N
    if A[j]<min then begin
      k:=j;
      min:=A[j];
    end;
  A[k]:=A[i];
  A[i]:=min;
end;
```

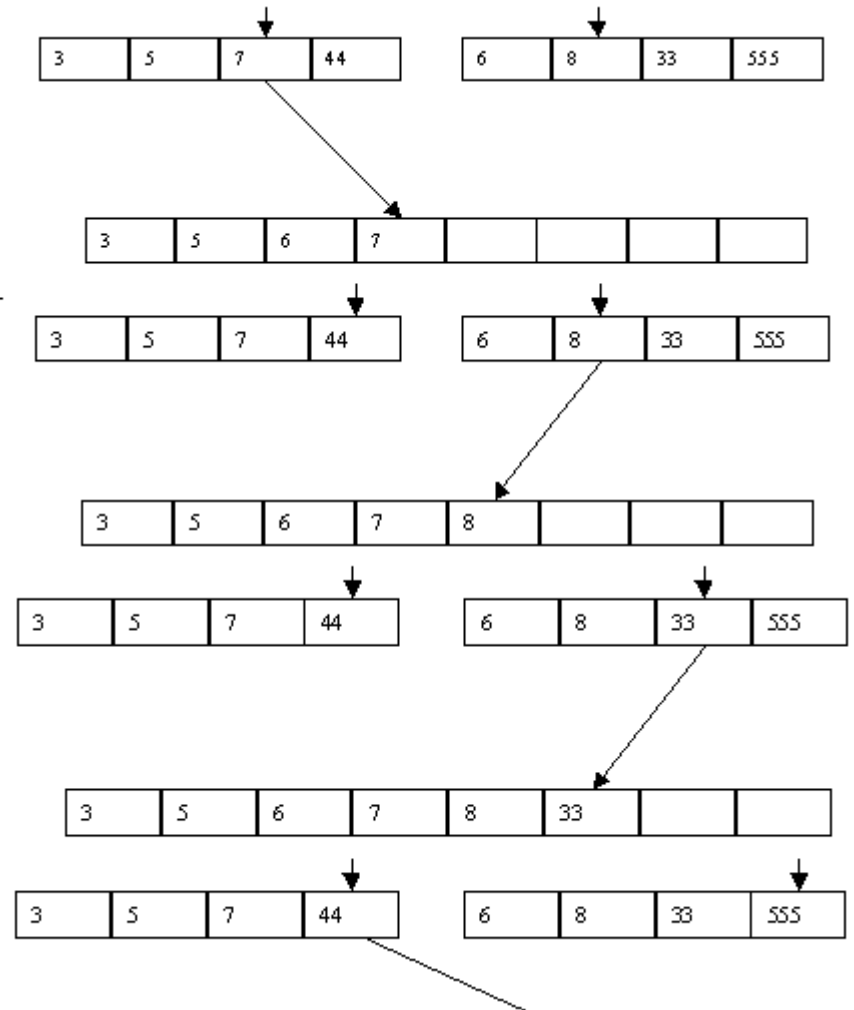
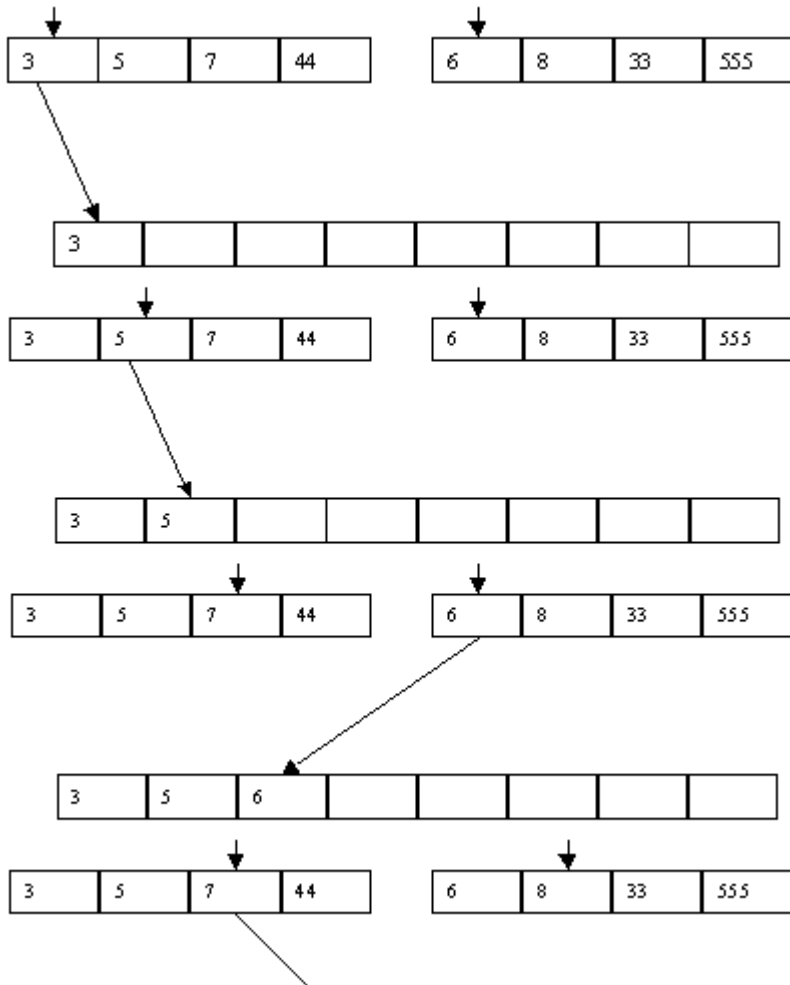
Сортировка слиянием

34

- Разбиение массива на два подмассива. Сортировка каждого из подмассивов.
- Рекурсивное применение алгоритма к каждому из подмассивов, пока размер подмассива не станет равным 1.
- Слияние отсортированных подмассивов в один.
 - Пусть мы имеем две стопки карт, лежащих рубашками вниз так, что в любой момент мы видим верхнюю карту в каждой из этих стопок. Пусть также, карты в каждой из этих стопок идут сверху вниз в неубывающем порядке. Как сделать из этих стопок одну?
 - На каждом шаге мы берем меньшую из двух верхних карт и кладем ее рубашкой вверх в результирующую стопку. Когда одна из оставшихся стопок становится пустой, мы добавляем все оставшиеся карты второй стопки к результирующей стопке.

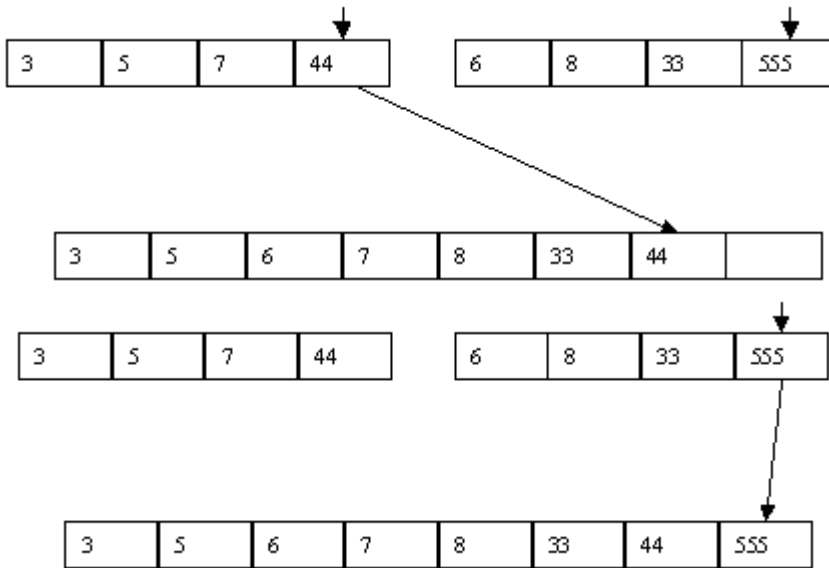
Сортировка слиянием

35



Сортировка слиянием

36



Сортировка слиянием

37

```
procedure MergeSort (var A: TArray;  
                    L, R: Integer)  
var m: Integer;  
begin  
    if L<R then begin  
        m:=(L+R) div 2;  
        MergeSort (A, L, m);  
        MergeSort (A, m+1, R);  
        Merge (A, L, m, m+1, R);  
    end;  
end;
```

№ шага	Массив
0	8 23 5 65 44 33 1 6
1	6 8 1 23 5 33 44 65
2	6 8 44 65 1 5 23 33
3	1 5 6 8 23 33 44 65

Сортировка слиянием (анимация)

38



Сортировка слиянием

39

```
procedure Merge(var A: TArray; L1, R1, L2, R2: Integer);
var
  i, j, k: Integer;
  B: TArray;
begin
  i:=L1; j:=L2; k:=L1;
  while (i<=R1) and (j<=R2) do begin
    if A[i]<=A[j] then begin { Берем из левого подмассива }
      B[k]:=A[i];
      i:=i+1;
    end else begin { Берем из правого подмассива }
      B[k]:=A[j];
      j:=j+1;
    end;
    k:=k+1;
  end;
end;
```

Сортировка слиянием

40

...

```
if i<=R1 then { если в левом подмассиве остались элементы }
  while i<=R1 do begin
    B[k]:=A[i];
    i:=i+1;
    k:=k+1;
  end;
if j<=R2 then { если в правом подмассиве остались элементы }
  while j<=R2 do begin
    B[k]:=A[j];
    j:=j+1;
    k:=k+1;
  end;
for i:=L1 to R2 do A[i]:=B[i];
end;
```


Быстрая сортировка

41

- *Выбрать опорный элемент массива (разделяющий элемент, барьер, pivot).*
- *Разделить массив на подмассивы: сравнить остальные элементы с опорным и разбить массив на три подмассива: "меньшие опорного", "равные" и "большие", расположить их в порядке меньше-равные-большие.*
- *Повторить рекурсивно для подмассивов "меньших" и "больших".*



Сэр Ч.Э.Р. Хоар
(р. 1934)

Разделение массива

42

1. Два индекса l и r , приравниваются к минимальному и максимальному индексу разделяемого массива соответственно
2. Вычисляется опорный элемент m
3. Индекс l последовательно увеличивается до m или до тех пор, пока l -й элемент не окажется больше опорного
4. Индекс r последовательно уменьшается до m или до тех пор, пока r -й элемент не окажется меньше опорного
5. Если $r=l$ (найдена середина массива), операция разделения закончена, оба индекса указывают на опорный элемент
6. Если $l < r$ — найденную пару элементов нужно обменять местами и продолжить операцию разделения с тех значений l и r , которые были достигнуты.

Следует учесть, что если какая-либо граница (l или r) дошла до опорного элемента, то при обмене значение m изменяется на r или l соответственно.

Завершение быстрой сортировки

43

- Базой рекурсии являются наборы, состоящие из одного или двух элементов. Первый возвращается в исходном виде, во втором, при необходимости, сортировка сводится к перестановке двух элементов. Все такие отрезки уже упорядочены в процессе разделения.
- Поскольку в каждой итерации (на каждом следующем уровне рекурсии) длина обрабатываемого отрезка массива уменьшается, по меньшей мере, на единицу, терминальная ветвь рекурсии будет достигнута всегда и обработка гарантированно завершится.

Выбор опорного элемента

44

- С точки зрения корректности алгоритма выбор опорного элемента безразличен.
- С точки зрения повышения эффективности алгоритма выбираться должна медиана – элемент массива, который делит массив на 2 равные части: "не больше" и "не меньше". Без дополнительных сведений о сортируемых данных медиану получить невозможно.
- Известные стратегии выбора опорного элемента:
 - выбирать постоянно один и тот же элемент, например, средний, первый или последний по положению;
 - выбирать элемент со случайно выбранным индексом.

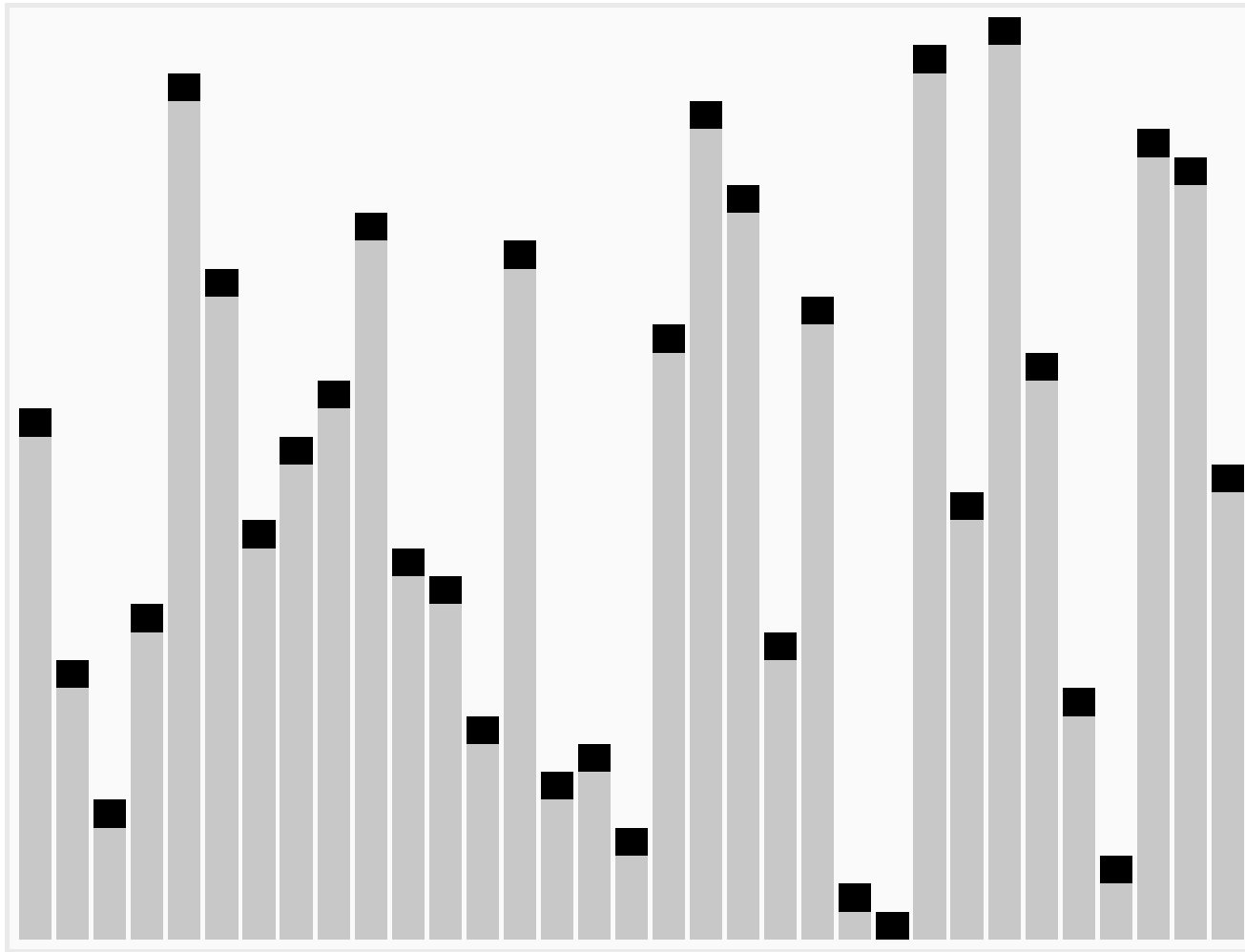
Быстрая сортировка

45

№ шага	Массив
0	8 23 5 65 44 33 1 6
1	8 23 5 6 44 33 1 65 8 23 5 6 1 33 44 65
2	8 23 5 6 1 33 44 65 1 23 5 6 8 33 44 65 1 5 23 6 8 33 44 65
3	1 5 23 6 8 33 44 65 1 5 8 6 23 33 44 65

Быстрая сортировка (анимация)

46



Быстрая сортировка

47

```
procedure Sort (var A: TArray;  
                L, R: Integer);  
var i, j, Bar: Integer;  
begin  
  i:=L; j:=R;  
  Bar:=A[(L+R) div 2];  
  { Bar:=A[Random(R-L+1)+L]; }  
  repeat  
    while A[i]<Bar do i:=i+1;  
    while Bar<A[j] do j:=j-1;  
    if i<=j then begin  
      if A[i]>A[j] then  
        Swap(A[i], A[j]);  
      i:=i+1;  
      j:=j-1;  
    end;  
  until i>j;  
  if L<j then Sort(A, L, j);  
  if i<R then Sort(A, i, R);  
end;
```

```
procedure QuickSort  
(var A: TArray);  
begin  
  { Randomize; }  
  Sort(A, 1, N);  
end;
```

Сравнение алгоритмов сортировки

48

Алгоритм	Мин	Сред	Макс
Простые вставки	$C=n-1$ $M=2(n-1)$	$C=(n^2+n-2)/4$ $M=(n^2-9n-10)/4$	$C=(n^2-n)/2-1$ $M=(n^2-3n-4)/2$
Простой выбор	$C=(n^2-n)/2$ $M=3(n-1)$	$C=(n^2-n)/2$ $C=n(\ln(n) + 0.57)$	$C=(n^2 - n)/2$ $M=n^2/4+3(n-1)$
Пузырьковая сортировка	$C=(n^2-n)/2$ $M=0$	$C=(n^2- n)/2$ $M=3(n^2-n)/4$	$C=(n^2-n)/2$ $M=3(n^2-n)/2$

C – количество операций сравнения

M – количество операций обмена

Внешняя сортировка

49

- *Внешняя сортировка* – упорядочение последовательности элементов, которая не может быть целиком расположена в оперативной памяти, с использованием оперативной и внешней памяти (*сортировка файлов*).
- Файл допускает только последовательный доступ к элементам. Любой алгоритм сортировки файла основан на изъятии элементов файла и включении их в файл в новые места.
Возможные операции:
 - *Распределение* – переписывание части файла в новый файл (или файлы).
 - *Слияние* – объединение разделенных файлов.

Прямое слияние

50

- При распределении в два дополнительных файла попеременно записываются последовательности из $2^{\text{шаг}-1}$ (1, 2, 4, 8, 16, ...) элементов исходного файла.
- При слиянии дополнительных файлов сливаются две последовательности из $2^{\text{шаг}-1}$ элементов, и результирующая последовательность записывается в исходный файл.

Прямое слияние

51

№ шага	Файлы
0	А: 8 23 5 65 44 33 1 6 В: С:
1	В: 8 5 44 1 С: 23 65 33 6 А: 8 23 5 65 33 44 1 6
2	В: 8 23 33 44 С: 5 65 1 6 А: 5 8 23 65 1 6 33 44
3	В: 5 8 23 65 С: 1 6 33 44 А: 1 5 6 8 23 33 44 65

Серии файла

52

- *Серия (отрезок)* – упорядоченная часть файла.
- Упорядоченный файл состоит из 1 серии.
- Обрато упорядоченный файл состоит из n серий (n – количество элементов в файле).
- Пример:
11 14 ▽ 7 9 10 ▽ 3 ▽ 0 14 17 22 71 ▽ 65
- Сортировка – преобразование файла из n ($n > 1$) серий в файл из 1 серии.

Естественное слияние

53

- *При распределении* в два дополнительных файла попеременно записываются серии исходного файла.
- *При слиянии* дополнительных файлов сливаются две очередные серии, и результирующая серия записывается в исходный файл.
- Если просмотр одного дополнительного файла заканчивается раньше, чем просмотр другого, то остаток второго файла целиком переписывается в конец результирующего файла.
- Процесс завершается, когда в результирующем файле остается только одна серия.

Естественное слияние

54

№ шага	Файлы
0	A: 8 23 5 65 44 33 1 6 B: C:
1	B: 8 23 44 1 6 C: 5 65 33 A: 5 8 23 44 65 1 6 33
2	B: 5 8 23 44 65 C: 1 6 33 A: 1 5 6 8 23 33 44 65

Многопутевое слияние

55

- Распределение серий исходного файла A по m вспомогательным файлам B_1, B_2, \dots, B_m и их слияние в m вспомогательных файлов C_1, C_2, \dots, C_m .
- Слияние файлов C_1, C_2, \dots, C_m в файлы B_1, B_2, \dots, B_m и т.д., пока в B_1 или C_1 не образуется одна серия.

Трехпутевое слияние

56

№ шага	Файлы
0	A: 8 23 5 65 44 33 1 6
1	B₁: 8 23 33
	B₂: 5 65 1 6
	B₃: 44
2	C₁: 5 8 23 33 44 65
	C₂: 1 6
3	B₁: 1 5 6 8 23 33 44 65

Многофазная сортировка

57

- В многопутевой сортировке по мере увеличения длины серий часть вспомогательных файлов с перестает использоваться, поскольку этим файлам "не достается" ни одной серии.
- В многофазной сортировке из имеющихся m вспомогательных файлов $m-1$ файл служит для ввода сливаемых последовательностей, а один – для вывода образуемых серий. Как только один из файлов ввода становится пустым, его начинают использовать для вывода серий, получаемых при слиянии серий нового набора $m-1$ файлов.

Многофазная сортировка

58

- При произвольном начальном распределении серий по вспомогательным файлам алгоритм может не сойтись, поскольку в единственном непустом файле будет существовать более чем одна серия.
- Пример (для трехфазной сортировки, $m=3$)

Количество серий в файле		
B_1	B_2	B_3
10	6	0
4	0	6
0	4	2
2	2	0
0	0	2

Многофазная сортировка

59

- Каким должно быть начальное распределение серий алгоритма трехфазной сортировки?
 - ▣ Алгоритм должен завершаться.
 - ▣ На каждом шаге должно осуществляться слияние максимального количества серий
- Рассмотрим распределение серий в обратном порядке работы алгоритма, начиная с одного из желаемых: $(B_1, B_2, B_3) \in \{(1,0,0), (0,1,0), (0,0,1)\}$.

Количество серий в файле		
B_1	B_2	B_3
13	8	0
5	0	8
0	3	5
3	0	2
1	2	0
1	0	0



Многофазная сортировка

60

- Метод трехфазной внешней сортировки работает, если начальное распределение серий между вспомогательными файлами описывается *соседними числами Фибоначчи*: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...
- В общем виде при использовании t вспомогательных файлов условием является описание начального распределения серий между $t-1$ файлами суммами соседних $t-1, t-2, \dots, 1$ чисел Фибоначчи порядка $t-2$.
 - Последовательность чисел Фибоначчи порядка p начинается с p нулей, $p+1$ элемент равен 1, а каждый следующий равняется сумме предыдущих $p+1$ элементов.
 - Последовательность чисел Фибоначчи порядка 4:
0, 0, 0, 0, 1, 1, 2, 4, 8, 16, 31, 61, ...
- Количество серий в исходном файле может не обеспечивать возможность такого распределения серий. Тогда в файл добавляются пустые серии, которые в дальнейшем как можно более равномерно распределяются между промежуточными файлами и опознаются при последующих слияниях.

Применение алгоритмов сортировки массивов в сортировке файлов

61

- Чем более длинные серии содержит файл перед началом применения внешней сортировки, тем меньше потребуются слияний и тем быстрее закончится сортировка.
- До применения любого из методов внешней сортировки исходный файл частями считывается в основную память, к каждой части применяется один из наиболее эффективных алгоритмов внутренней сортировки (обычно Quicksort) и отсортированные части (серии) записываются в новый файл, который будет сортироваться вместо исходного.