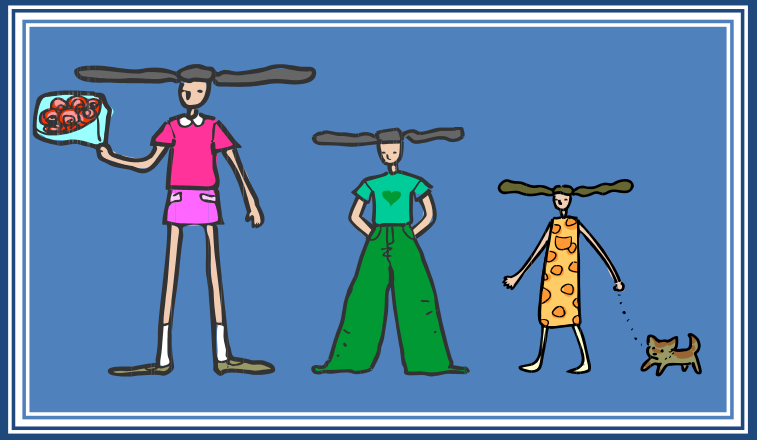


СОРТИРОВКА

Многие же первые будут последними
и последние первыми.

Ев. от Матфея, 19:30

Основы программирования

Содержание

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

Терминология

- *Задача сортировки*
  - *Исходные данные* – последовательность элементов данных $(a_1, a_2, \ldots, a_n)$, для которых введено отношение порядка
  - *Результат* – последовательность $(\tilde{a}_1, \tilde{a}_2, \ldots, \tilde{a}_n)$, состоящая из тех же элементов, идущих в неубывающем порядке: $\tilde{a}_1 \le \tilde{a}_2 \le \ldots \le \tilde{a}_n$
- *Сортируемый элемент*
  - ```
    type TElement=record
           key: …; { Ключ сортировки }
           OtherData: …; { Другие данные }
    end;
    ```
  - Упрощение: сортируемый элемент ≡ ключ, имеющий целочисленный тип.
    ```

Терминология

- *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

- *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

- 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

- 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

- Используется дополнительный (результирующий) 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

- 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

- 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

№ 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 **1 5 6 8 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)

Пузырьковая сортировка


```
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

- 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)


```
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)


```
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

- 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

№ 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


```
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

- 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

№ 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**

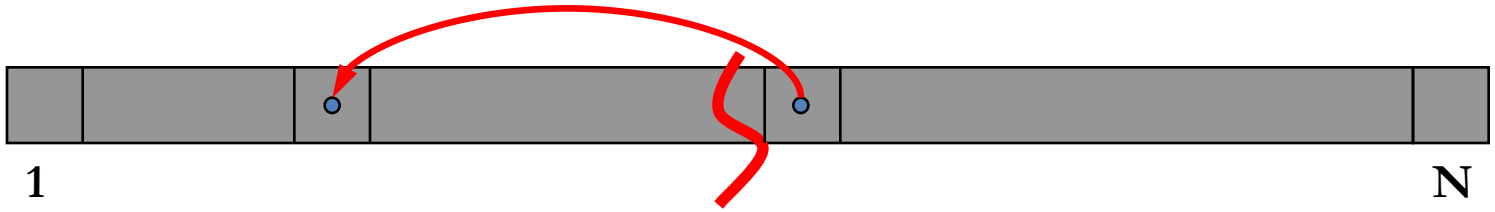
Сортировка Шелла (анимация)

Сортировка Шелла


```
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 простыми вставками

- 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

№ 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)

Сортировка простыми вставками


```
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


```
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;
```

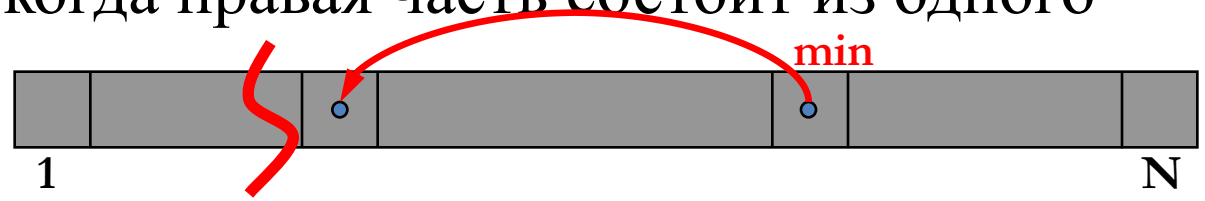
Гномья сортировка

- 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.

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

Сортировка простым выбором

- 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

№ 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)

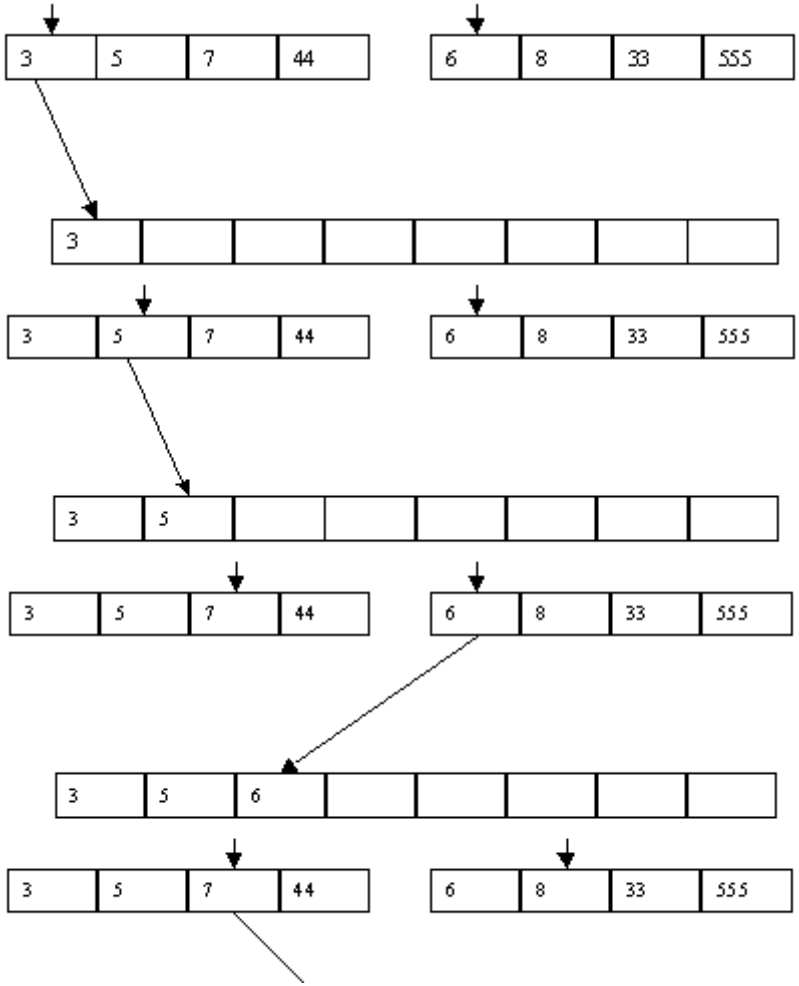
Сортировка простым выбором

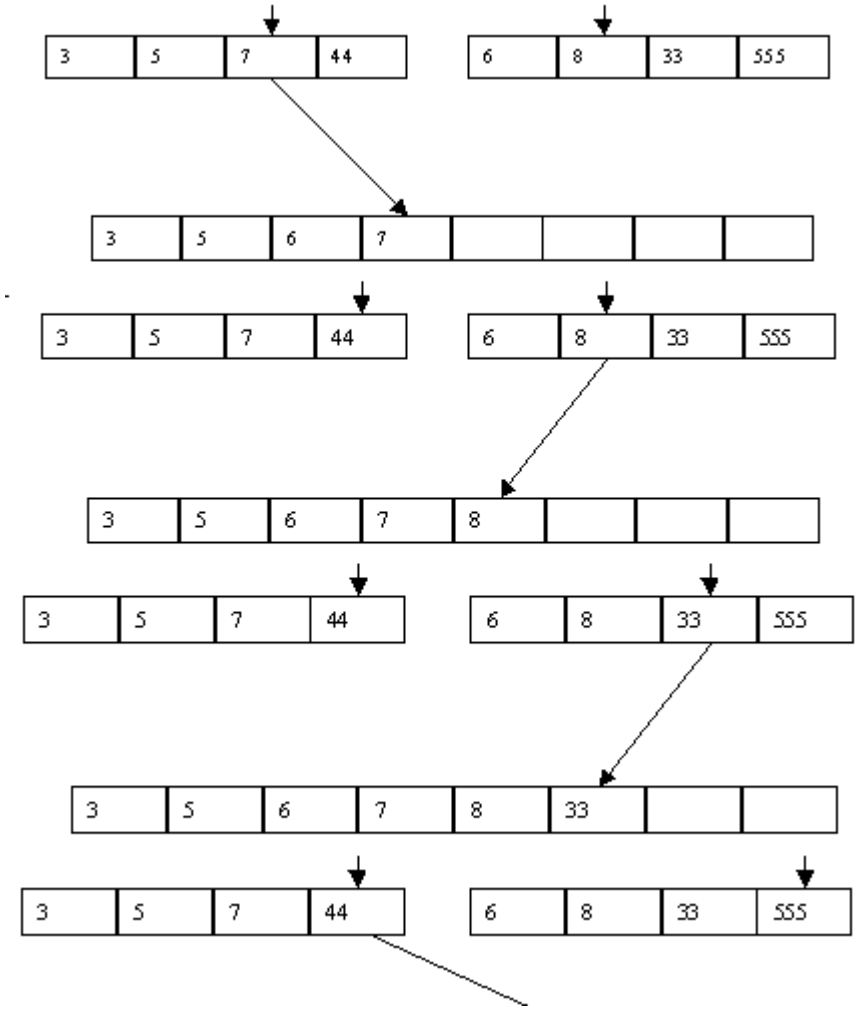

```
for i:=1 to N-1 do begin
  min:=A[i];
  k:=i;
  for j:=i+1 to N
    if A[j]<min then begin
      k:=j;
      min:=A[j];
    end;
  A[k]:=A[i];
  A[i]:=min;
end;
```

Сортировка слиянием

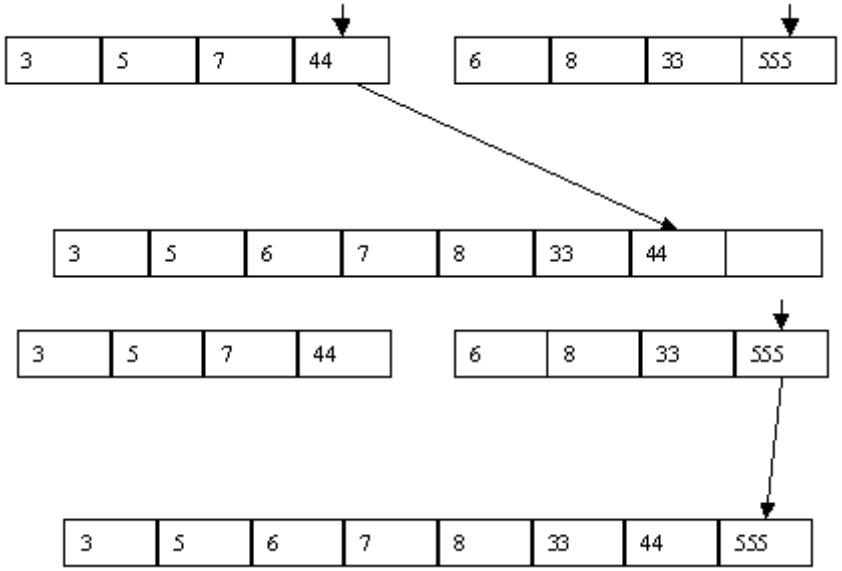
- Разбиение массива на два подмассива. Сортировка каждого из подмассивов.
- Рекурсивное применение алгоритма к каждому из подмассивов, пока размер подмассива не станет равным 1.
- Слияние отсортированных подмассивов в один.
  - Пусть мы имеем две стопки карт, лежащих рубашками вниз так, что в любой момент 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





Сортировка слиянием



Сортировка слиянием


```
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)

Сортировка слиянием


```
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;
```

Сортировка слиянием


```
…
   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

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

Сэр Ч.Э.Р. Хоар (р. 1934)

Разделение массива

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

- 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

- 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.

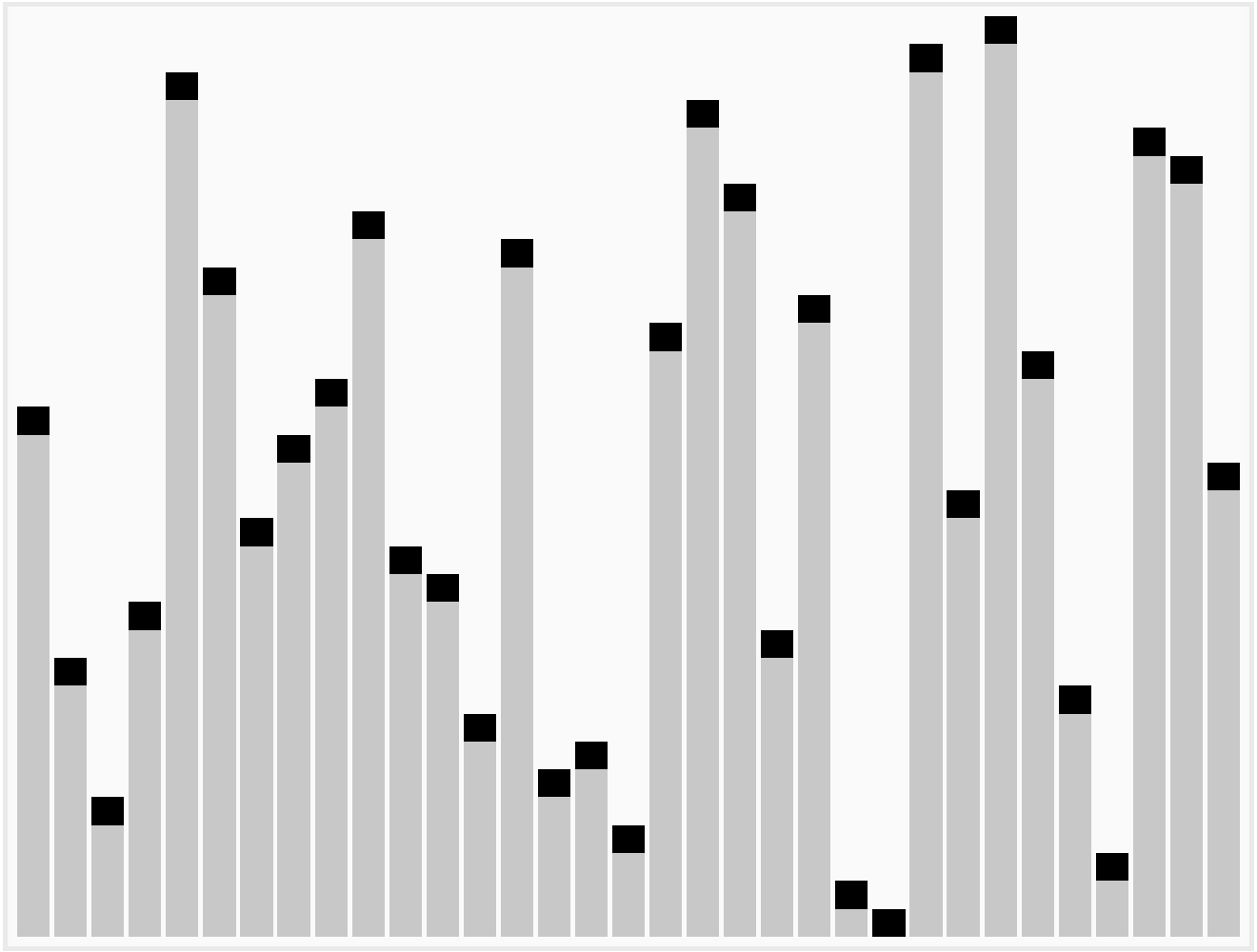
Быстрая сортировка

№ 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-1)

Сортировка

Быстрая сортировка (анимация-2)



Быстрая сортировка


```
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

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

- *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

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

Прямое слияние

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

Серии файла

- *Серия (отрезок)* – упорядоченная часть файла.
- Упорядоченный файл состоит из 1 серии.
- Обратно 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

- 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

№ 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

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

Трехпутевое слияние

№ шага	Файлы	
0	**A:**	8 23 5 65 44 33 1 6
1	**B_1:**	8 23 33
	B_2:	5 65 1 6
	B_3:	44
2	**C_1:**	5 8 23 44 65
	C_2:	1 6
3	**B_1:**	**1 5 6 8 23 33 44 65**

Многофазная сортировка

- 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

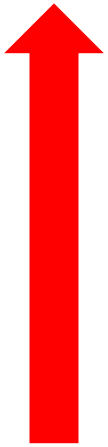
- При произвольном начальном распределении серий по вспомогательным файлам алгоритм может не сойтись, поскольку в единственном непустом файле будет существовать более чем одна серия.
- Пример (для трехфазной сортировки, m=3)

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

- 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: $(B_1, B_2, B_3) \in \{\mathbf{(1,0,0)}, (0,1,0), (0,0,1)\}$.

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



Сортировка

Многофазная сортировка

- 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 *m* вспомогательных файлов условием является описание начального распределения серий между *m*-1 файлами суммами соседних *m*-1, m-2, ..., 1 *чисел Фибоначчи порядка m-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

- Чем более длинные серии содержит файл перед началом применения внешней сортировки, тем меньше потребуется 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.