«Алгоритмы и структуры данных»

АЛГОРИТМЫ "БЫСТРОЙ" СОРТИРОВКИ

Базируются на принципе «разделяй и властвуй» и операции обмена.

В зависимости от способа деления набора данных на части и различают несколько видов «быстрой» сортировки:

1. Базовый алгоритм «быстрой» сортировки
2. «Быстрая» сортировка с разделением на три части
3. «Быстрая» сортировка вычислением медианы из трех элементов

Базовый алгоритм «быстрой» сортировки

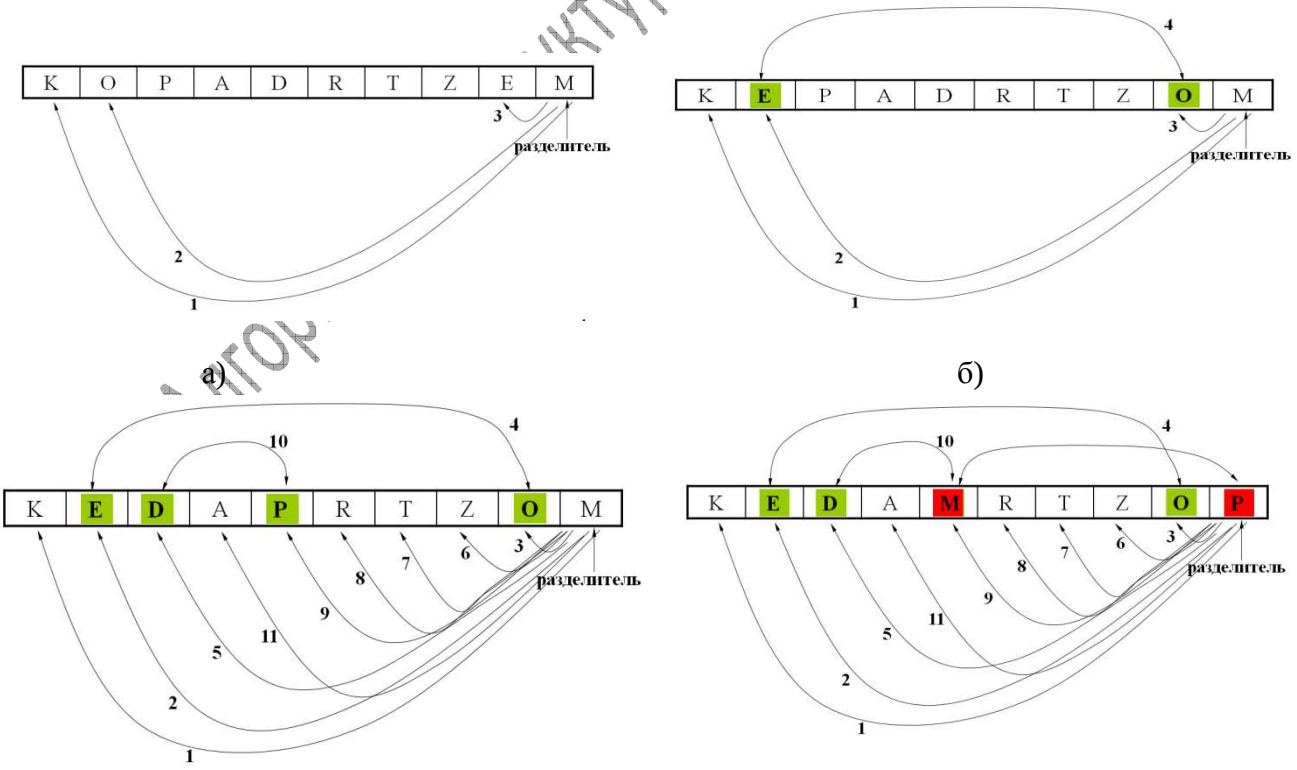
Суть – набор данных делится на две части, которые сортируются независимо друг от друга. Положение точки деления зависит от исходного порядка элементов в наборе.

1. Определить разделяющий элемент.
2. Поместить его в свою окончательную позицию с выполнением перегруппировка набора данных следующим образом: элементы с меньшим значением помещаются слева от него, элементы с большим значением – справа.
3. Выполнить разделение каждого поднабора на две части отдельно.
4. Повторить п.3 до тех пор, пока в поднаборе не будет один элемент.

Способы выбора разделяющего элемента

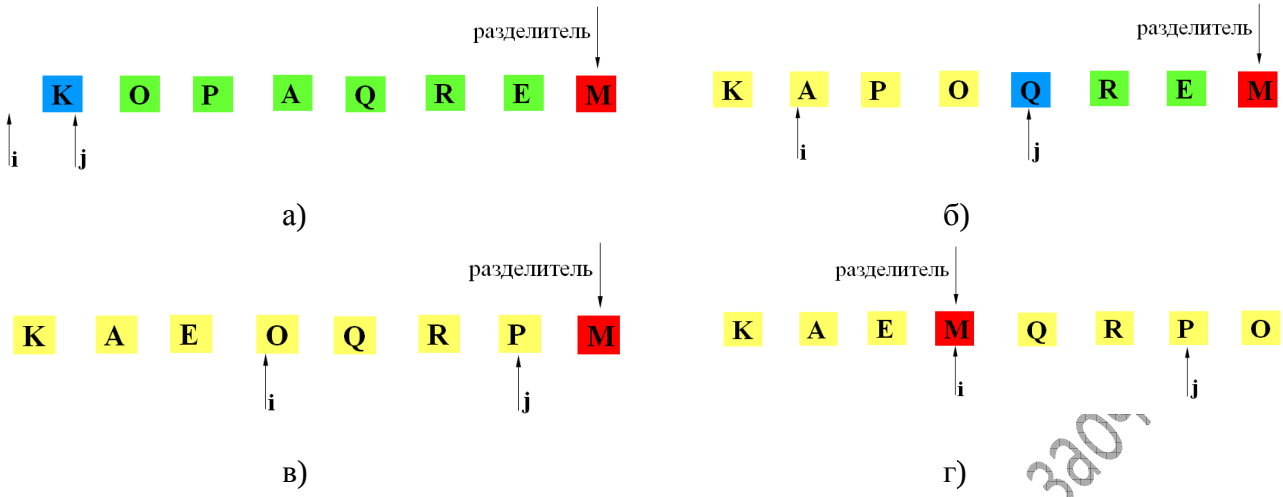
1. Средний элемент;
2. Последний элемент;
3. Первый элемент;
4. Случайный элемент;
5. Средний из трех случайных.

В примере в качестве разделительного элемента выбираем последний. Индексы элементов, что сравниваются, двигаются навстречу друг другу.



Есть другой вариант реализации, когда в качестве разделительного элемента выбирается последний и индексы элементов, что сравниваются, двигаются слева направо (j –

определяет текущий элемент для сравнения с разделителем, i – окончание области элементов, имеющих значения меньше разделителя).



Пример реализации:

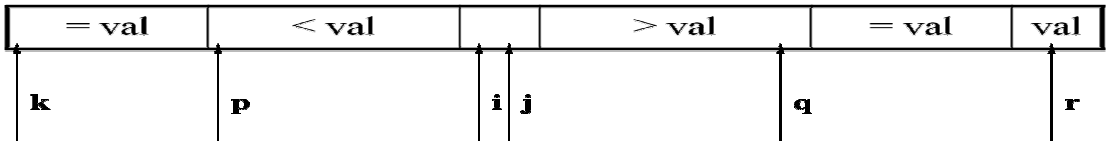
```
Quick_Sort(Array, left, right)
    if  left < right  then
            q ← call  Partition(Array, left, right)
            call   Quick_Sort (Array, left, q−1)
            call   Quick_Sort (Array, q+1, right)

Partition (Array, posN, posK)
    x ← Array[posK]
    i ← posN − 1
    for   j ← posN   to   posK   do
        if   Array[ j ] ≤ x   then
                i ← i +1
                Array[i] ↔ Array[j]
    Array[i +1] ↔ Array[posK]
    return   i +1
```

«Быстрая» сортировка с разделением на три части

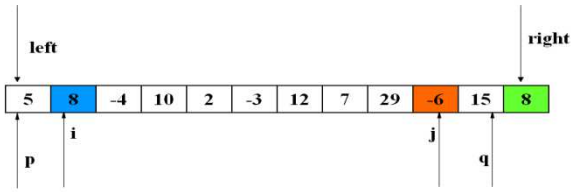
Если в наборе данных присутствуют не уникальные значения.

Суть:

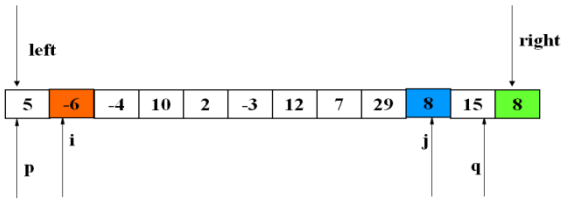
- в процессе разделения элементы со значением равным разделяющему и встретившиеся в левой части перемешаются к левому краю;
- элементы со значением равным разделяющему и встретившиеся в правой части перемещаются к правому краю;
- после установки разделяющего элемента в свою позицию к нему присоединяются равные значения;
- затем левая и правая части сортируются отдельно.



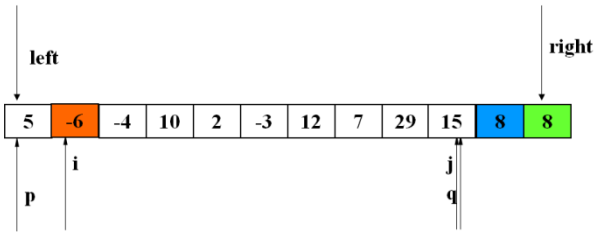
Рассмотрим пример:



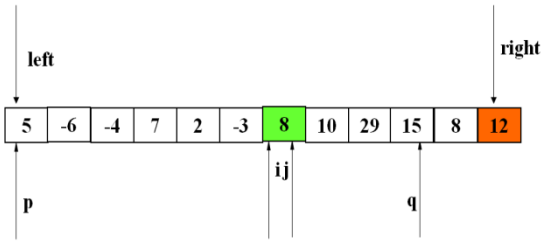
а)



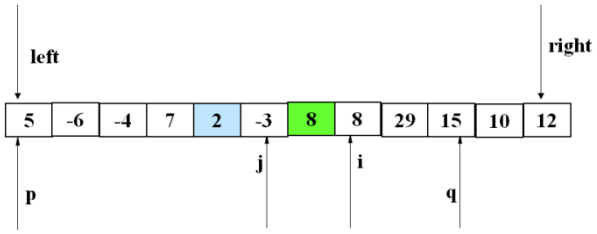
б)



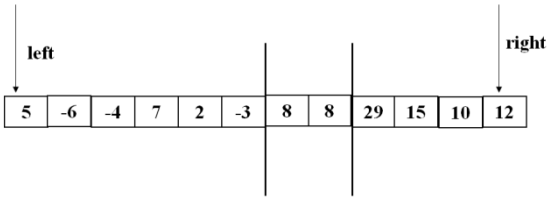
в)



г)



д)



е)

Quick_Part_Sort(*Array*, *left*, *right*)

if *left* ≥ *right* **then return**
value ← *Array*[right]
i ← *left*
j ← *right* - 1
posL ← *left*
posR ← *right* - 1
while TRUE **do**
while *Array*[i] < *value* **do**
i ← *i* + 1
while *Array*[j] > *value* **do**
if *j* = *left* **then** BREAK
j ← *j* - 1
if *i* ≥ *j* **then** BREAK
Array[i] ↔ *Array*[j]
if *Array*[i] = *value* **then**
Array[i] ↔ *Array*[posL]
posL ← *posL* + 1
if *Array*[j] = *value* **then**
Array[j] ↔ *Array*[posR]
posR ← *posR* – 1
Array[i] ↔ *Array*[right]
j ← *i* - 1
i ← *i* + 1
for *n* ← *left* **to** *posL* **do**
Array[n] ↔ *Array*[j]
j ← *j* - 1
for *n* ← *right*-1 **to** *posK* **by** -1 **do**

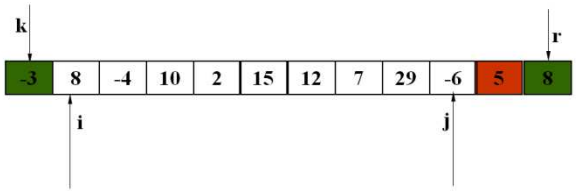
Array[n] ↔ *Array*[i]
i ← *i* + 1
call Quick_Part_Sort(*Array*, *left*, *j*)
call Quick_Part_Sort(*Array*, *i*, *right*)

Алгоритм «быстрой» сортировки вычислением медианы из трех элементов

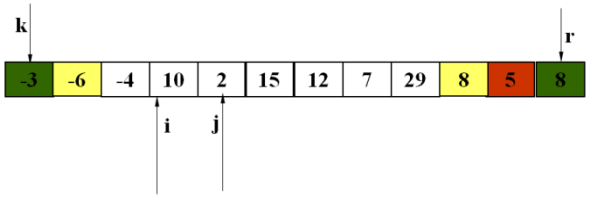
Суть – разделяющий элемент выбирается из трех случайных (в основном из первого, последнего и среднего):

1) Средний меняется местами с предпоследним;
2) Первый, предпоследний и последний элементы упорядочиваются (наименьший – на первое место, наибольший – на последнее);
3) Сортируется набор данных между первым и предпоследним элементами разделением на две части;
4) Окончанием рекурсивной сортировки является проверка условия: если количество элементов в поднаборе меньше некоторого установленного значения, то сортировка в такой части данным алгоритмом игнорируется.

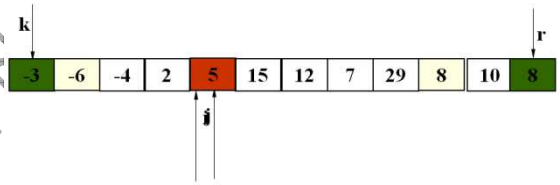
а)



б)



в)



г)

Quick_Median_Sort(*Array*, *left*, *right*)
if *left* < *right* **then**
if (*right* - *left*) = 1 **then**
if *Array*[left] > *Array*[right] **then**
Array[left] ↔ *Array*[right]
else
call Median(*Array*, *left*, *right*)
if (*right* - *left*) ≠ 2 **then**
q ← **call** Partition(*Array*, *left*, *right*-1)
call Quick_Median_Sort (*Array*, *left*, *q*−1)
call Quick_Median_Sort (*Array*, *q*+1, *right*)

Median(*Array*, *left*, *right*)
med ← *left* + (*right* - *left*)/2
if *Array*[left] > *Array*[med] **then**
Array[left] ↔ *Array*[med]
if *Array*[left] > *Array*[right] **then**
Array[left] ↔ *Array*[right]
if *Array*[med] > *Array*[right] **then**
Array[med] ↔ *Array*[right]
Array[med] ↔ *Array*[right-1]

```
Partition(Array, left, right)
    i ← left
    j ← right
    value ← Array[right]
    while TRUE do
        do   i ← i+1
        while Array[i] < value
        do   j ← j-1
        while Array[j] > value
        if i < j then
            Array[i] ↔ Array[j]
        else BREAK
    Array[i] ↔ Array[right]
    return i
```

СОРТИРОВКА СЛИЯНИЕМ

Слияние – это процесс объединения упорядоченных наборов данных в один.

1. Алгоритм двухпутевого слияния
2. Алгоритм нисходящей сортировки слиянием
3. Алгоритм восходящей сортировки слиянием

Алгоритм двухпутевого слияния

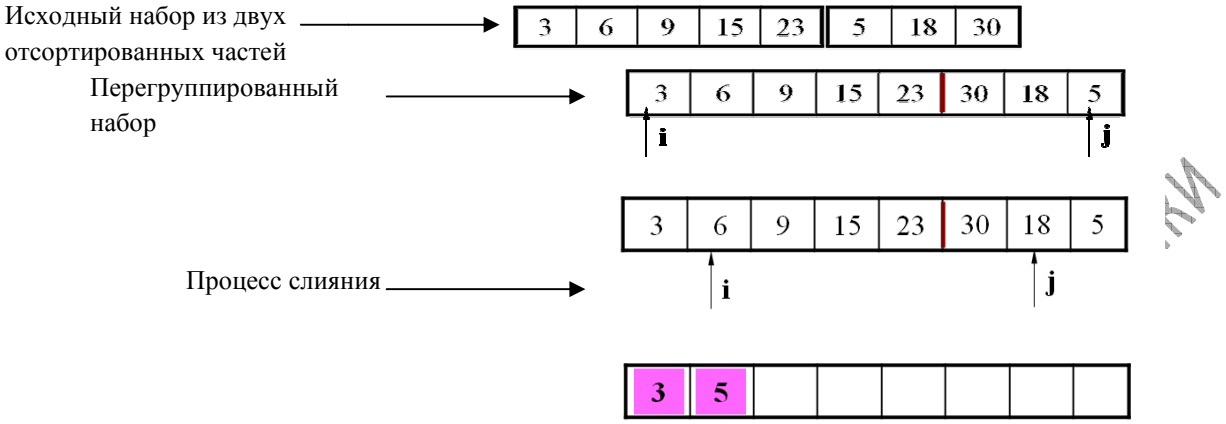
Суть – из двух отсортированных наборов данных последовательно выбираются наименьшие (наибольшие) и перемещаются в третий (i-й элемент из одного набора сравнивается с j-й элементом из второго набора и наименьший (наибольший) записывается в третий).

Для выполнения такой сортировки требуется дополнительная память объемом не менее, чем суммарный объем объединяемых данных.

```
Merge(Array_1, Array_2)
    ► Создание массива Aux[] размерностью length(Array_1)+length(Array_2)
    i ← 0
    j ← 0
    for k ← 0 to length(Array_1)+length(Array_2) do
        if i ≥ length(Array_1) then
            Aux[k] ← Array_2[j]
            j ← j + 1
        else if j ≥ length(Array_2) then
            Aux[k] ← Array_1[i]
            i ← i + 1
        else if Array_1[i] ≤ Array_2[j] then
                Aux[k] ← Array_1[i]
                i ← i + 1
            else
                Aux[k] ← Array_2[j]
                j ← j + 1
    return Aux
```

Алгоритм обменного слияния

Альтернатива прямому слиянию – это обменное слияние: набор данных разделен на две упорядоченные части, которые копируются в дополнительный набор в порядке роста значений элементов навстречу друг другу, затем производится сравнение элементов объединенного набора данных с левого и правого конца и производится перемещение наименьшего (наибольшего) в результирующий набор данных.



Merge(*Array, left, med, right*)

► Создание массива Aux[] размерностью (right – left + 1)

for *i* ← *med* **to** *left*+1 **by** -1 **do**

Aux[i] ← *Array*[i]

for *j* ← *med*+1 **to** *right*+1 **do**

Aux[right + med + 1 - j] ← *Array*[j]

for *k* ← *left* **to** *right*+1 **do**

if *Aux*[i+1] < *Aux*[j-1] **then**

Array[k] ← *Aux*[i+1]

i ← *i* + 1

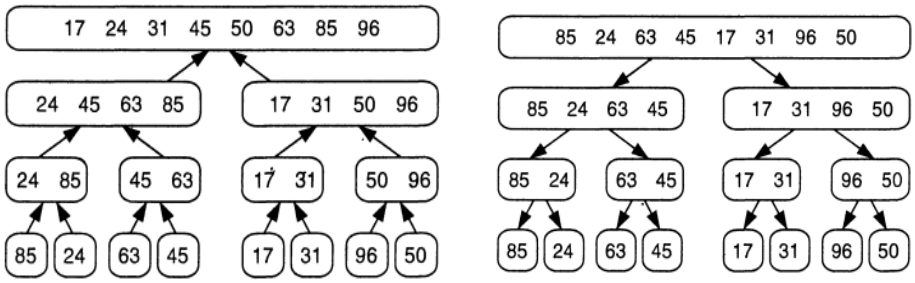
else

Array[k] ← *Aux*[j-1]

j ← *j* - 1

Алгоритм нисходящей сортировки слиянием

Суть – набор данных делится на две части и выполняется рекурсивная сортировка обеих частей с последующим их слиянием.



Merge_Sort(*Array, left, right*)

if *left* < *right* **then** ►Условие окончания деления на части

q ← (*left* + *right*)/2 ► Деление на две части

► Повторный вызов для левой части

Merge_Sort(*Array, left, q*)

► Повторный вызов для правой части

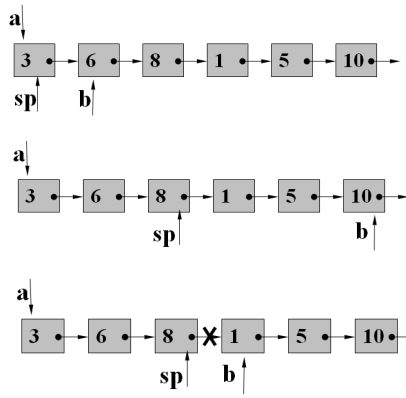
Merge_Sort(*Array, q*+1, *right*)

►Обменное слияние двух частей

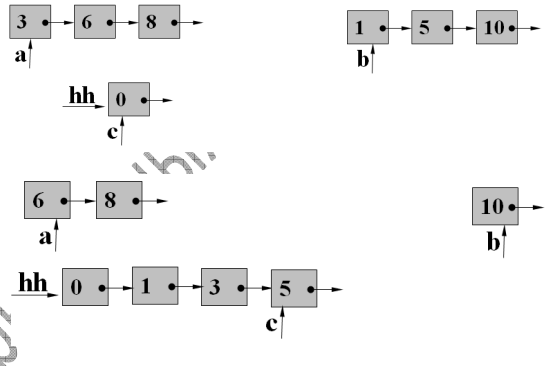
Merge(*Array, left, q, right*)

Пример применения нисходящей сортировки слиянием для списка:

```
public Elem sort(Elem sp)  {
        if ((sp == null) || (sp.next == null))
                return sp;
        Elem a = sp,
             b = sp.next;

        while ((b != null) && (b.next != null))  {
                        sp = sp.next;
                        b = b.next.next;
        }
        b = sp.next;
        sp.next = null;
        return con(sort(a), sort(b));
    }
```



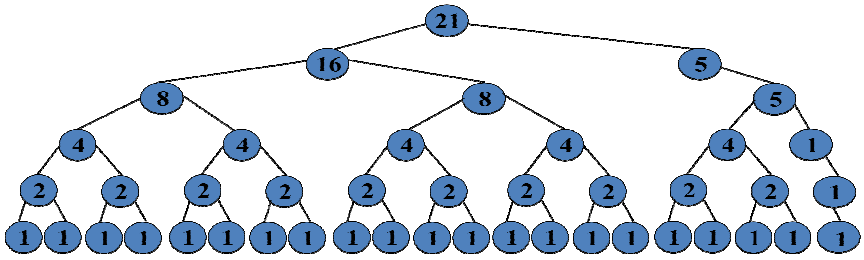

```
public Elem con(Elem a, Elem b) {
        Elem c = new Elem(0),
             hh = c;
        while ((a != null) && (b != null))
                if (a.d < b.d) {
                        c.next = a;
                        c = a;
                        a = a.next;
                }
                else
                        c.next = b;
                        c = b;   b =
                        b.next;
                }
        c.next = (a == null) ? b : a;
        return hh.next;
 }
```



Алгоритм восходящей сортировки слиянием

Суть – нерекурсивная сортировка, выполнение которой предполагает использование принципа «объединяй и властвуй» (сначала выполняется слияние соседних элементов, затем соседних пар, затем соседних двойных пар и т.д.

Пример выполнения для массива из 21 элемента



Merge_Sort_1 (*Array*, *left*, *right*)

for *m* ← 1 **to** *right*+l **by** *m* ← 2**m* **do**

for *i* ← 0 **to** *right* - *m* **by** *i* ← *i* + 2**m* **do**

Merge(*Array*, *i*, *i*+*m*-1, min(*i*+2**m*-1, *right*));