Сортировка данных

Коваленко Д.В. о8 ПОВТ(у)

- В широком смысле *сортировкой* называют перестановку элементов множества в определенном порядке.
- Задачей сортировки является преобразование исходной последовательности в последовательность, содержащую те же записи, но в порядке возрастания (или убывания) значений ключа.

Рассматривают две категории сортировки:

Внутренняя сортировка – это тип сортировки, который производится на месте, путем обмена элементами массива. Основным требованием внутренней сортировки является жесткая экономия памяти

Внешняя сортировка — это сортировка данных, расположенных на периферийных устройствах и не вмещающаяся в оперативную память.

Стоит отметить, что внутренняя сортировка значительно эффективней внешней, так как на обращение к оперативной памяти затрачивается намного меньше времени, чем к магнитным дискам, лентам...

Поговорим о некоторые простых видах внутренней сортировки

Сортировка простыми включениями

- Этот метод обычно используют игроки в карты. Элементы (карты) условно разделяют на готовую a[1], ..., a[i-1] и входную последовательности a[i], ..., 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 на подходящее место.

Процесс сортировки простыми включениями показан на примере восьми случайно взятых чисел

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

Алгоритмы сортировки простыми включениями

В графическом виде

На C++

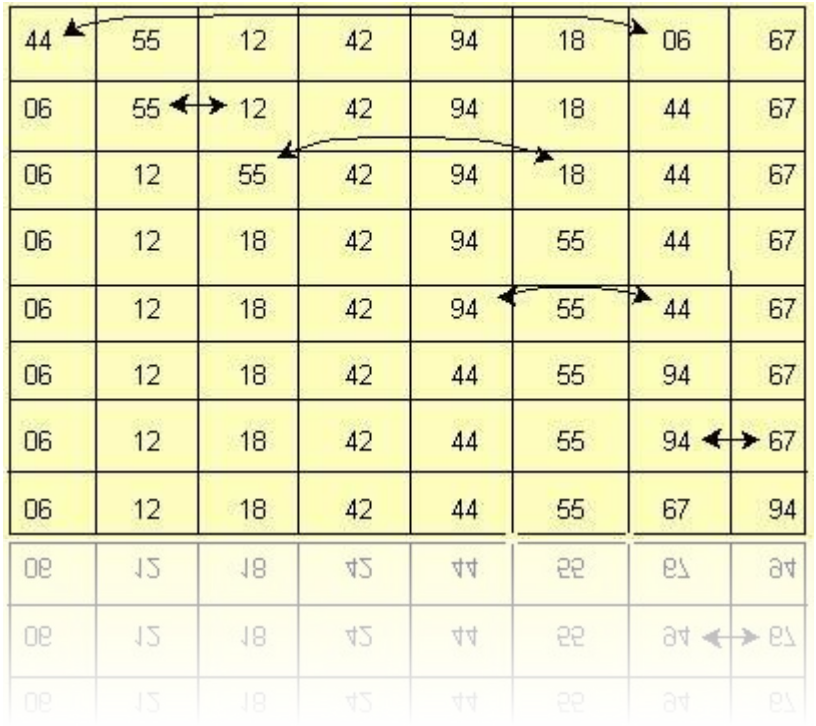
```
#include <algorithm>

template< typename Iterator >
void insertion_sort( Iterator first, Iterator last )
{
    for( Iterator i = first + 1; i < last; ++i )
        for( Iterator j = i; first < j && *j < *(j - 1); --j )
            std::iter_swap( j - 1, j );
}
```

Сортировка простым выбором

Этот метод основан на следующем правиле:

- выбираем (выделяем) элемент с наименьшим ключом,он меняется местами с первым элементом.
- 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 ключах



Алгоритмы сортировки простым выбором

В графическом виде

На C++

```
#include <algorithm>

 template< typename Iterator >
 void selection_sort( Iterator first, Iterator last )
 {
     while( first < --last )
         std::iter_swap( last, std::max_element( first, last + 1 ) );
 }
```

- Данный метод, в некотором смысле противоположен сортировке прямыми включениями; при сортировке простыми включениями на каждом шаге рассматривается только один очередной элемент входной последовательности и все элементы готового массива для нахождения места для включения; при сортировке простым выбором рассматриваются все элементы входного массива для нахождения элемента с наименьшим ключом, и этот один очередной элемент отправляется в готовую последовательность.

Сортировка простым обменом

Классификация методов сортировки не всегда четко определена. Методы простого включения и простого выбора используют в процессе сортировки обмен ключей. Однако теперь мы рассмотрим метод, 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.

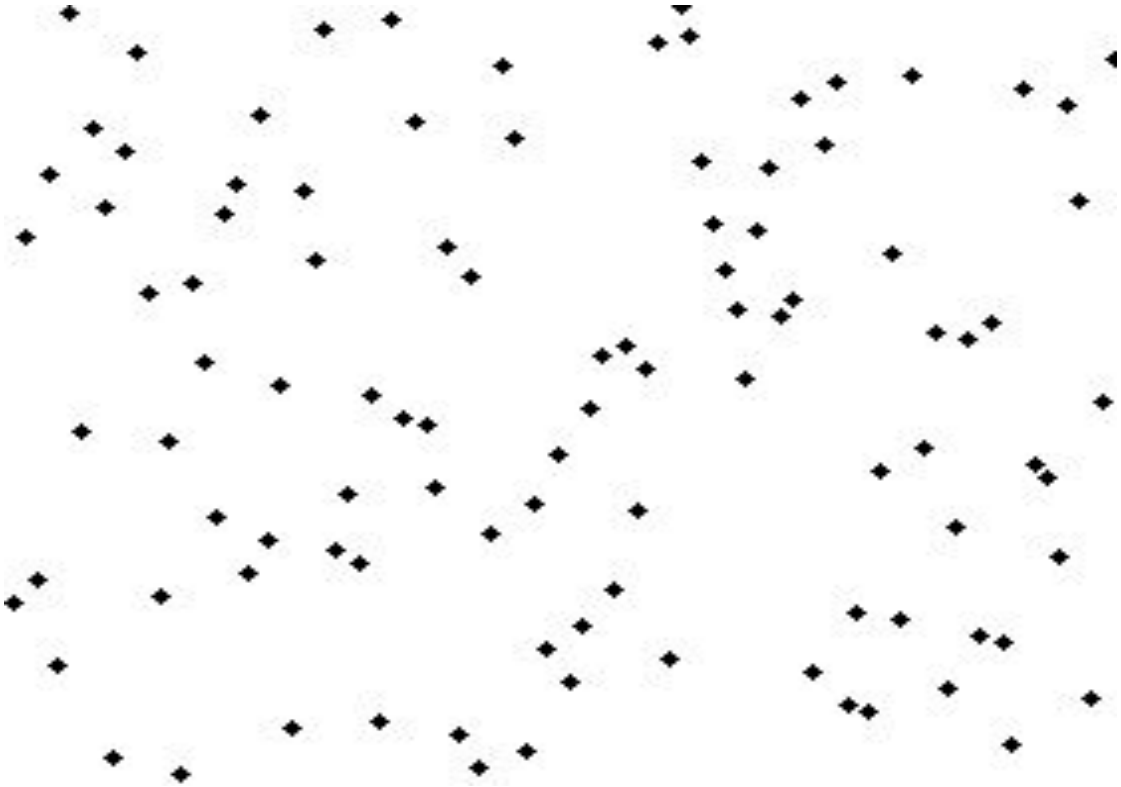
Если мы будем рассматривать массив, расположенный вертикально, а не горизонтально и представим себе элементы пузырьками в резервуаре с водой, обладающими "весами", соответствующими их ключам, то каждый проход по массиву приводит к "всплыванию" пузырька на соответствующий его весу уровень..

Этот метод широко известен как сортировка методом пузырька

начальные ключи	i=2	i=3	i=4	i=5	i=6	i=7	i=8
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

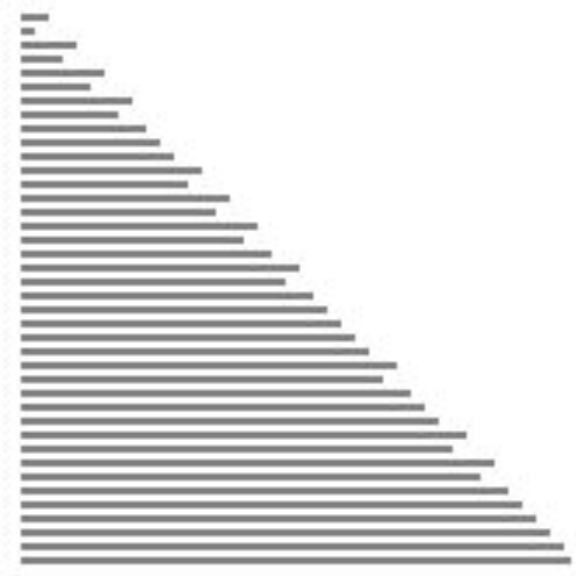
- Данный алгоритм легко оптимизировать. Пример преобразования ключей, приведенный в таблице показывает, что три последних прохода никак не влияют на порядок элементов, поскольку те уже рассортированы. Очевидный способ улучшить алгоритм - это запоминать, производится ли на данном проходе какой-либо обмен. Если нет, то это означает, что алгоритм может закончить работу. Этот процесс улучшения можно продолжить, если запомнить не только сам факт обмена, но и место (индекс) последнего обмена

Пример сортировки пузырьком списка случайных чисел



Алгоритмы сортировки простым обменом (пузырьковая)

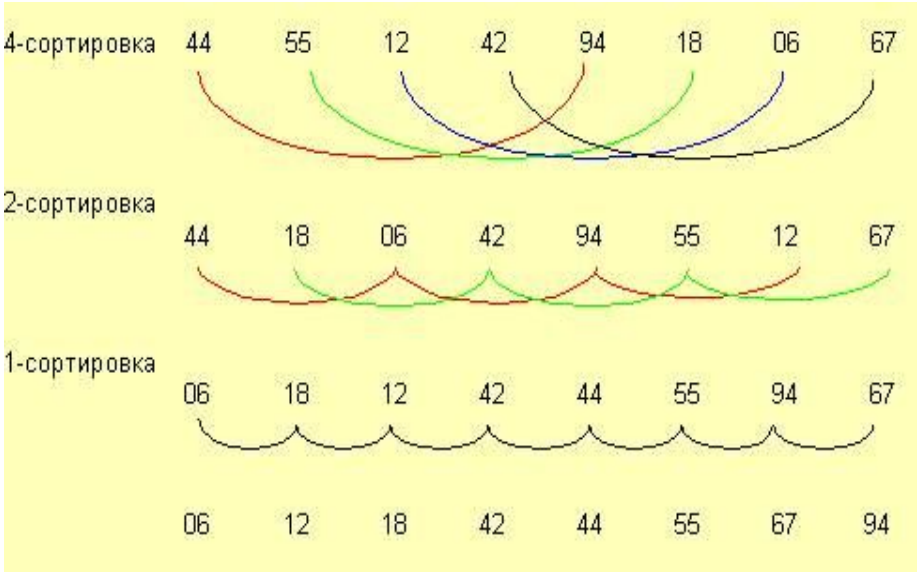
В графическом виде



На C++

```
#include <algorithm>
template< typename Iterator >
void bubble_sort( Iterator First, Iterator Last )
{
    while( First < --Last )
        for( Iterator i = First; i < Last; ++i )
            if ( *(i + 1) < *i )
                std::iter_swap( i, i + 1 );
}
```

Сортировка методом Шелла



Сортировка Шелла получила свое название по имени ее создателя Д.Л.Шелла. Однако, это название можно считать удачным, так как выполняемые при сортировке действия напоминают укладывание морских ракушек друг на друга. ("Ракушка" - одно из значений слова Shell)

Идея алгоритма состоит в сравнении элементов, стоящих не только рядом, но и на расстоянии друг от друга. Иными словами - сортировка вставками с предварительными "грубыми" проходами.

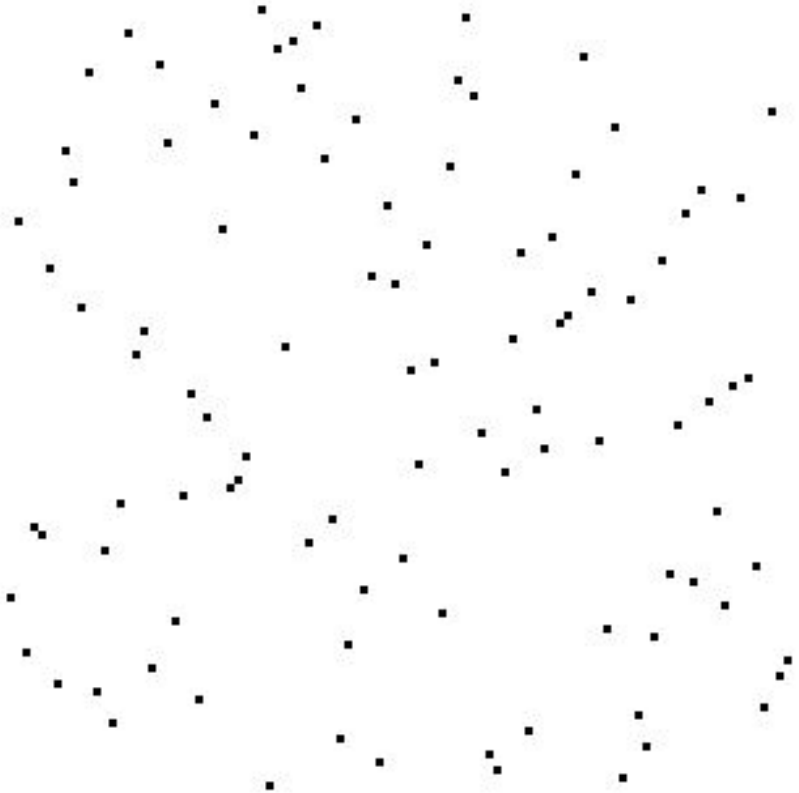
При сортировке Шелла сначала сравниваются и сортируются между собой ключи, отстоящие один от другого на некотором расстоянии d. После этого процедура повторяется для некоторых меньших значений d, а завершается сортировка Шелла упорядочиванием элементов при d = 1 (то есть, обычной сортировкой вставками). Эффективность сортировки Шелла в определённых случаях обеспечивается тем, что элементы «быстрее» встают на свои места (в простых методах сортировки вставками или пузырьком. Каждая перестановка двух элементов уменьшает количество инверсий в списке максимум на 1, при сортировке Шелла же это число может быть больше).

Невзирая на то, что сортировка Шелла во многих случаях медленнее, чем быстрая сортировка, она имеет ряд преимуществ:

- отсутствие потребности в памяти под стек
- отсутствие деградации при неудачных наборах данных

Часто оказывается, что сортировка Шелла есть самый лучший способ сортировки до, примерно, 1000 элементов.

Пример сортировки Шелла списка случайных чисел



Алгоритмы сортировки Шелла

В графическом виде

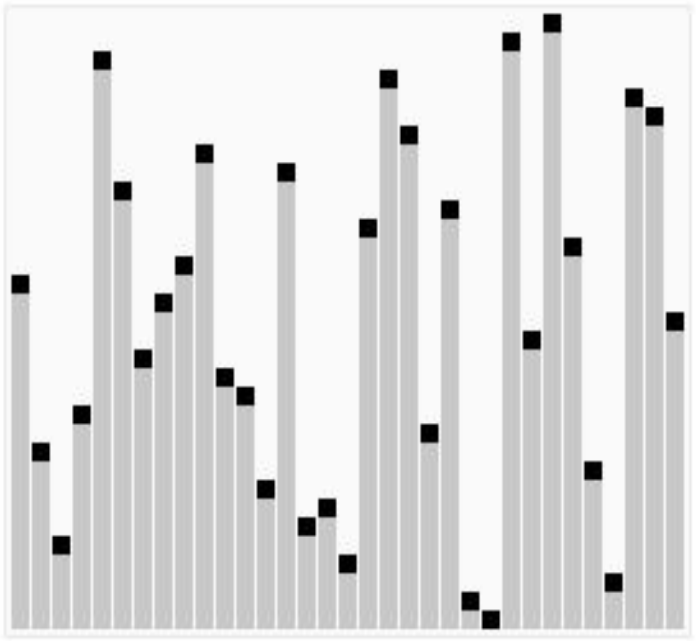
На C++

```
procedure Shell(var item: DataArray; count:integer);
  const
    t = 5;
  var
    i, j, k, s, m: integer;
    h: array[1..t] of integer;
    x: DataItem;
  begin
    h[1]:=9; h[2]:=5; h[3]:=3; h[4]:=2; h[5]:=1;
    for m := 1 to t do
      begin

    k:=h[m];
    s:=-k;
    for i := k+1 to count do
    begin
      x := item[i];
      j := i-k;
      if s=0 then
      begin
        s := -k;
        s := s+1;
        item[s] := x;
      end;
      while (x<item[j]) and (j<count) do
        begin
          item[j+k] := item[j];
          j := j-k;
        end;
        item[j+k] := x;
      end;
    end;
  end;
```

Быстрая сортировка

- Быстрая сортировка, часто называемая qsort по имени реализации в стандартной библиотеке языка Си — широко известный алгоритм сортировки, разработанный английским информатиком Чарльзом Хоаром. Один из быстрых известных универсальных алгоритмов сортировки массивов обменов при упорядочении n элементов), хотя и имеющий ряд недостатков.

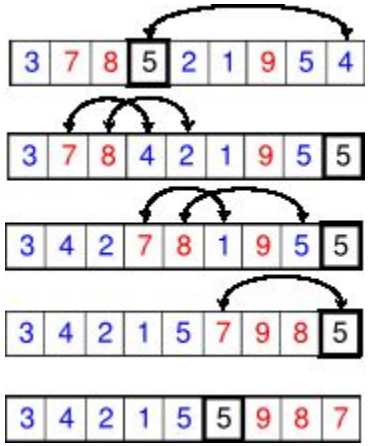
Краткое описание алгоритма



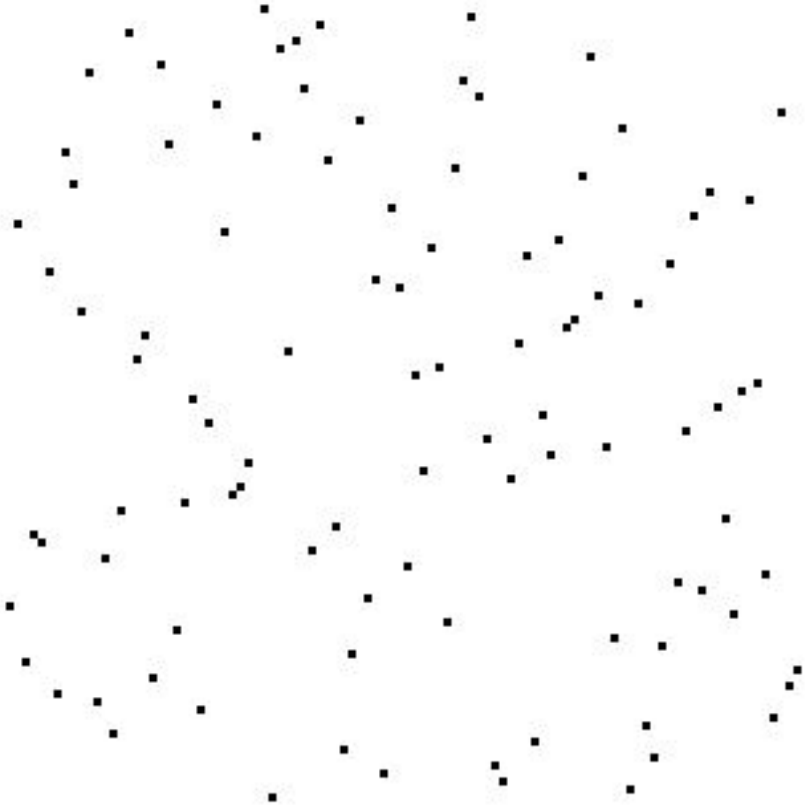
- выбрать элемент, называемый опорным.
- сравнить все остальные элементы с опорным, на основании сравнения разбить множество на три — «меньшие опорного», «равные» и «большие», расположить их в порядке меньшие-равные-большие.
- повторить рекурсивно для «меньших» и «больших».

Подробное описание алгоритма

- Быстрая сортировка использует стратегию «разделяй и властвуй». Шаги алгоритма таковы:
- Выбираем в массиве некоторый элемент, который будем называть опорным элементом. С точки зрения корректности алгоритма выбор опорного элемента безразличен. С точки зрения повышения эффективности алгоритма выбираться должна медиана, но без дополнительных сведений о сортируемых данных её обычно невозможно получить. Известные стратегии: выбирать постоянно один и тот же элемент, например, средний или последний по положению; выбирать элемент со случайно выбранным индексом.
- Операция разделения массива: реорганизуем массив таким образом, чтобы все элементы, меньшие или равные опорному элементу, оказались слева от него, а все элементы, большие опорного — справа от него. Обычный алгоритм операции:
- два индекса — l и r, приравниваются к минимальному и максимальному индексу разделяемого массива соответственно;
- вычисляется опорный элемент m;
- индекс l последовательно увеличивается до m или до тех пор, пока l-й элемент не превысит опорный;
- индекс r последовательно уменьшается до m или до тех пор, пока r-й элемент не окажется меньше опорного;
- если r = l — найдена середина массива — операция разделения закончена, оба индекса указывают на опорный элемент;
- 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.
- Рекурсивно упорядочиваем подмассивы, лежащие слева и справа от опорного элемента.
- 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.



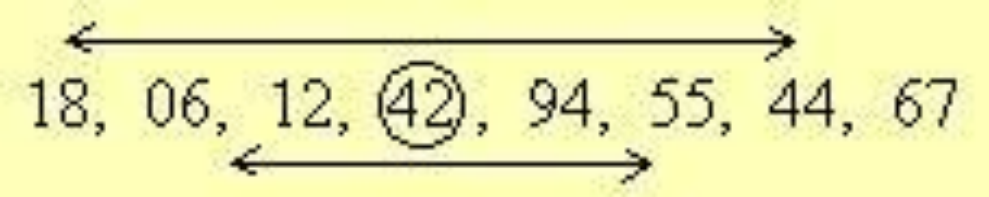
Пример быстрой сортировки списка случайных чисел



Если, например, выбрать средний ключ, равный 42, из массива ключей

44, 55, 12, 42, 94, 06, 18, 67,

то для того, чтобы разделить массив, потребуются два обмена



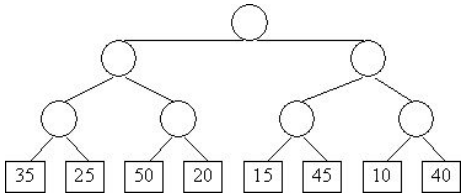
рафическом виде

```
procedure QuickSort(var item: DataArray; count:integer);
    procedure qs(l, r: integer; var it: DataArray);
      var
      i, j: integer;
      x, y: DataItem;
    begin
      i:=l; j:=r;
      x:=it[(l+r) div 2];
      repeat
        while it[i]<x do i := i+1;
        while x<it[j] do j := j-1;
        if y<=j then
        begin
          y := it[i];
          it[i] := it[j];
          it[j] := y;
          i := i+1; j := j-1;
        end;
      until i>j;
      if l<j then qs(l, j, it);
      if l<r then qs(i, r, it)
    end;
  begin
    qs(1, count, item);
  end;
```

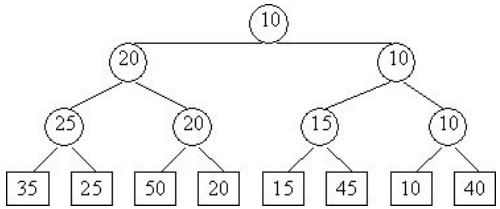
Сортировка выбором с помощью бинарного дерева

- По-другом турнирная сортировка. Бинарные деревья находят применение в качестве деревьев принятия решений. Например, при представлении спортивного турнира, где каждый внутренний узел соответствует победителю встречи между двумя игроками.
- Турнирное дерево может использоваться для сортировки массива из n элементов.

- Рассмотрим алгоритм турнирной сортировки на следующем примере: Пусть имеется массив из 8-ми элементов A[8]={35, 25, 50, 20, 15, 45, 10, 40}. Необходимо его отсортировать по возрастанию.
- 1. Элементы массива запоминаются в бинарном дереве на уровне k, где ; 2 в степени k >=n; n – это количество элементов массива. В данном случае элементы массива A запоминаются на уровне 3



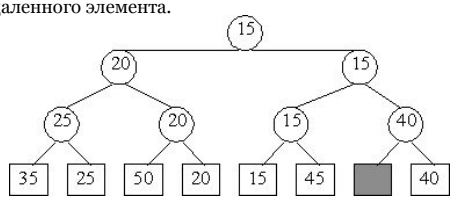
- 2. В родительские узлы помещаются наименьшие значения в парах. В результате последнего сравнения наименьший элемент массива попадает в корень дерева на уровне 0.



3. Наименьший элемент удаляется со своего старого места на дереве и копируется во вспомогательный массив.

10							

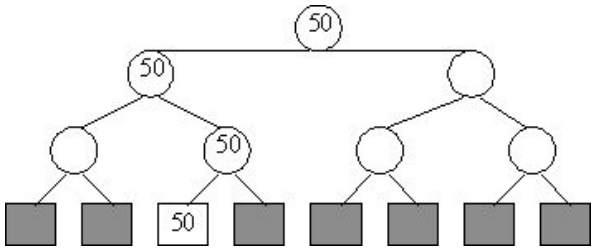
Так как один элемент был удален, то п.2 выполняется заново, но уже без удаленного элемента.



4. Аналогичные действия выполняются с числом 15:

10	15						

- Процесс продолжается до тех пор, пока все листья не будут удалены. Последний (наибольший) узел играет серию матчей, в которых побеждает всех по умолчанию:



  - В массиве, содержащем n = 2 k элементов, для выявления наименьшего элемента требуется n-1 сравнений. Общее число матчей равно $2^{k-1} + 2^{k-2} + \ldots + 2^1 + 1 = n - 1$
  - Общее число сравнений равно

$$(n-1) + (k-1)(n-1) = (n-1) + (n-1)(\log_2 n - 1) = (n-1)\log_2 n$$

Пирамидальная сортировка

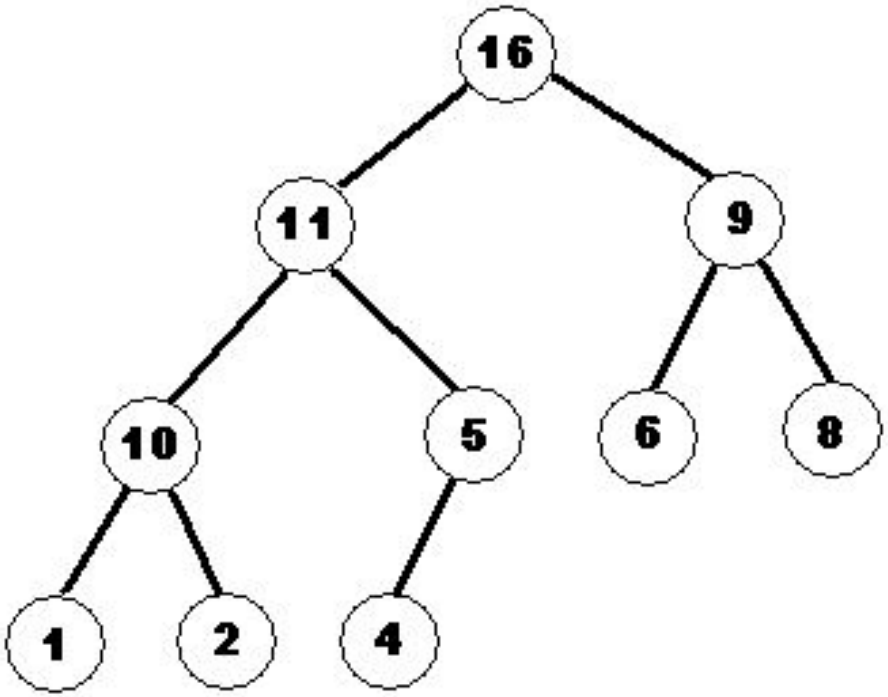
Является усовершенствованным методом простого выбора и входит в число наиболее эффективных методов внутренней сортировки

Простейший алгоритм

Сортировка пирамидой использует сортирующее дерево. Сортирующее дерево — это такое двоичное дерево, у которого выполнены условия:

1. Каждый лист имеет глубину либо d либо d – 1, d — максимальная глубина дерева.
2. Значение в любой вершине больше, чем значения её потомков.

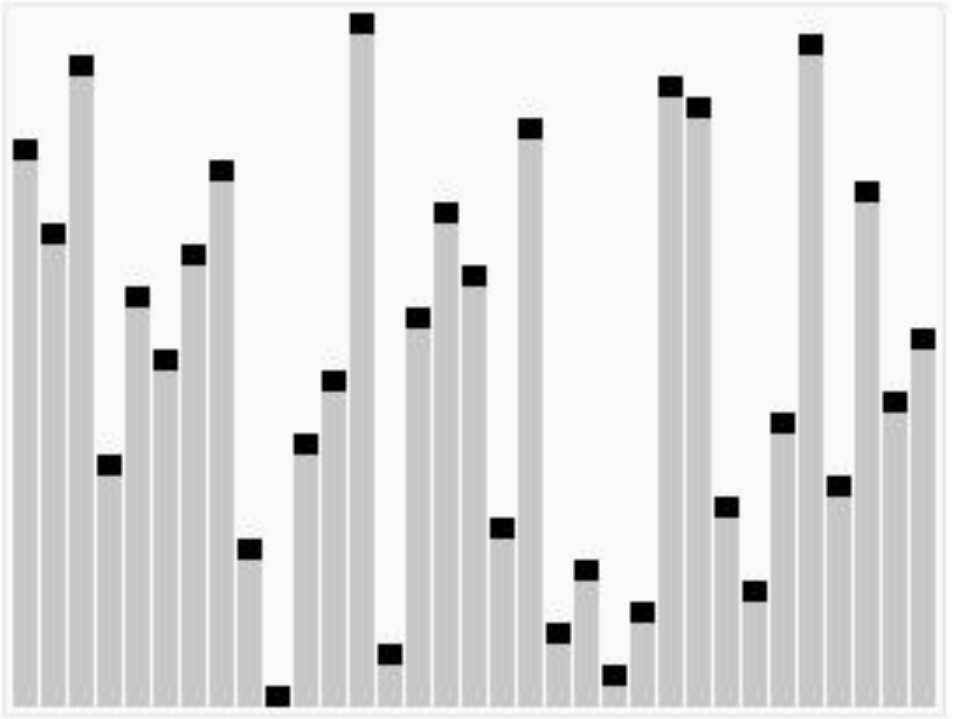
Пример сортирующего дерева



Алгоритмы пирамидальной сортировки на C++

```
#include <iterator>

template< typename Iterator >
void adjust_heap( Iterator first
                , typename std::iterator_traits< Iterator >::difference_type current
                , typename std::iterator_traits< Iterator >::difference_type size
                , typename std::iterator_traits< Iterator >::value_type tmp )
{
   typedef typename std::iterator_traits< Iterator >::difference_type diff_t;

   diff_t top = current, next = 2 * current + 2;

   for ( ; next < size; current = next, next = 2 * next + 2 )
   {
      if ( *(first + next) < *(first + next - 1) )
         --next;
      *(first + current) = *(first + next);
   }

   if ( next == size )
      *(first + current) = *(first + size - 1), current = size - 1;

   for ( next = (current - 1) / 2;
         top > current && *(first + next) < tmp;
         current = next, next = (current - 1) / 2 )
   {
      *(first + current) = *(first + next);
   }
   *(first + current) = tmp;
```

Анимированная схема алгоритма



В Заключении

- следует отметить, что имеются простые и усовершенствованные методы внутренней сортировки. Простые методы обеспечивают достаточную скорость для небольших массивов и входят как составная часть в усовершенствованные методы.