основы алгоритмизации и программирования

СОРТИРОВКА «БЫСТРАЯ»

ПЛАН ЛЕКЦИИ

Часть 1:

1. Понятие алгоритма
2. Идея алгоритма
3. Визуализация
4. Словесное представление алгоритма

Часть 2:

5. Блок-схема
6. Подробный разбор элементов блок-схемы
7. Программная реализация на языке программирования С

ПОНЯТИЕ АЛГОРИТМА

Быстрая сортировка (англ. *quick sort*, сортировка Хоара) — один из самых известных и широко используемых алгоритмов сортировки. Среднее время работы O(n log n), что является асимптотически оптимальным временем работы для алгоритма, основанного на сравнении. Хотя время работы алгоритма для массива из n элементов в худшем случае может составить Θ(n^2), на практике этот алгоритм является одним из самых быстрых. Для этого алгоритма применяется рекурсивный метод.

Рекурсией называется ситуация, когда программа вызывает сама себя непосредственно или косвенно (через другие функции).

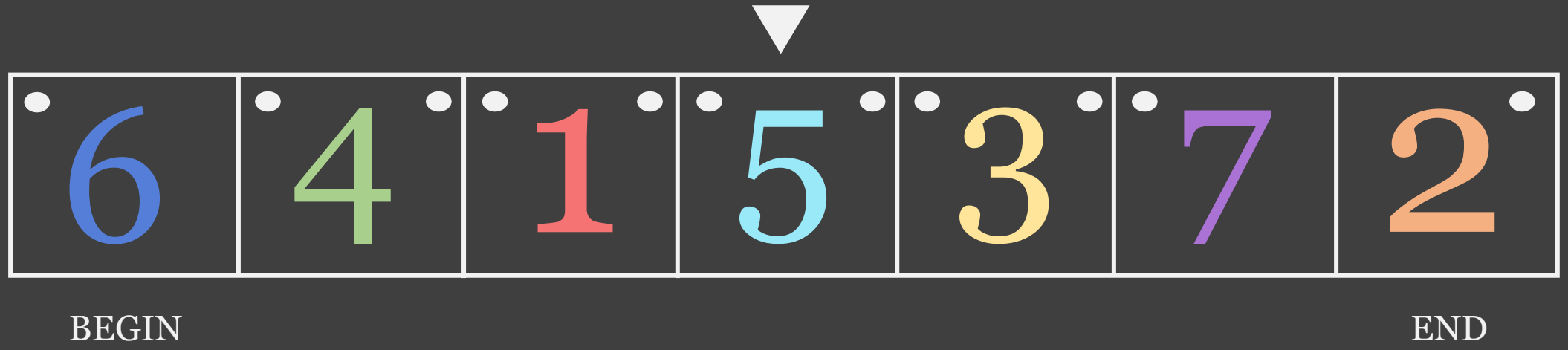
ИДЕЯ АЛГОРИТМА

Выбираем из массива элемент, называемый опорным, и запоминаем его значение. Это может быть любой из элементов массива. От выбора опорного элемента не зависит корректность алгоритма, но в отдельных случаях может сильно зависеть его эффективность.

Далее начинаем двигаться от начала массива по возрастающей, а потом от конца массива по убывающей. Цель: переместить в правую часть элементы больше опорного, а в левую – элементы меньше опорного. Если во время движения по возрастающей находится элемент со значением больше опорного, то мы выходим из цикла, прибавляем единицу к индексу элемента, на котором остановились, и переходим к циклу с движением по убывающей. В этом цикле мы остаемся до тех пор, пока не находится элемент со значением меньше опорного.

ИДЕЯ АЛГОРИТМА

Как только такой элемент найден, мы отнимаем единицу от его индекса, и меняем значение элемента со значением элемента, на котором мы остановились в предыдущем цикле. Делаем так до тех пор, пока индекс левого элемента (найденного в первом цикле) меньше либо равен индексу правого элемента (найденного во втором цикле). В итоге получаем два подмассива (от начала до индекса правого элемента и от индекса левого элемента до конца). С этими подмассивами мы рекурсивно проделываем все то же самое, что и с большим массивом до тех пор, пока все элементы окончательно не отсортируются.


6
4
1
5
3
7
2
BEGIN
END

СЛОВЕСНОЕ ПРЕДСТАВЛЕНИЕ

array – массив, piv – номер опорного элемента, b – индекс первого элемента массива, e – индекс последнего элемента массива

1. Номер опорного элемента равен (b+e)/2
2. Начало курсор. Если l <= r (где l = b, а r = e), то переходим к пункту 3, иначе к пункту 13
3. Если array[l] < piv, то переходим к пункту 4, иначе к пункту 6.
4. Прибавляем единицу к индексу левого элемента (l++).
5. Переходим к пункту 3.
6. Если array[r] > piv, то переходим к пункту 7, иначе к пункту 9.
7. Убавляем на единицу индекс правого элемента (r--)
8. Переходим к пункту 6.
9. Если l <= r, то переходим к пункту 10, иначе переходим к пункту 13.
10. Меняем значения элементов с индексами l и r местами (array[l] и array[r])

СЛОВЕСНОЕ ПРЕДСТАВЛЕНИЕ

array – массив, piv – номер опорного элемента, b – индекс первого элемента массива, e – индекс последнего элемента массива

11 Прибавляем единицу к индексу левого элемента и убавляем на единицу индекс правого элемента.

12 Переходим к пункту 2.

13 Если b < r (индекс первого элемента массива меньше индекса правого элемента массива), то переходим к пункту 14, иначе к пункту 15.

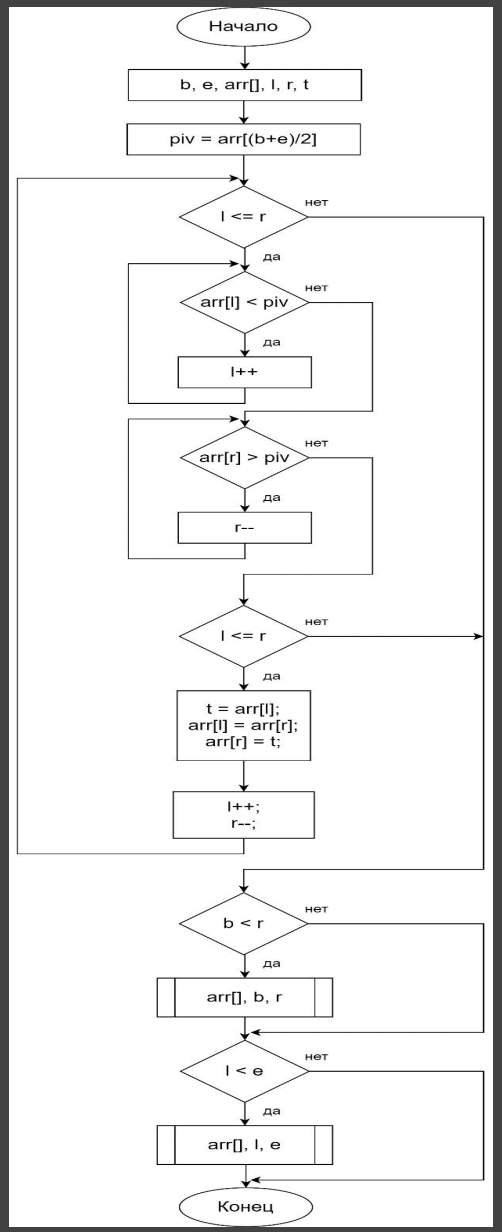
14 Вызов курсор: (array[], b, r).

15 Если l < e (индекс левого элемента меньше индекса последнего элемента массива), то переходим к пункту 16, иначе к пункту 17.

16 Вызов курсор: (arr[], l, e).

17 Конец алгоритма

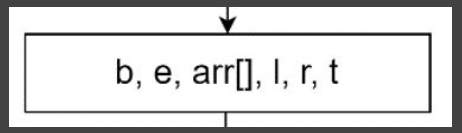
БЛОК-СХЕМА


Начало
b, e, arr[], l, r, t
piv = arr[(b+e)/2]
l <= r
нет
да
arr[l] < piv
нет
да
l++
arr[r] > piv
нет
да
r--
l <= r
нет
да
t = arr[l];
arr[l] = arr[r];
arr[r] = t;
l++;
r--;
b < r
нет
да
arr[], b, r
l < e
нет
да
arr[], l, e
Конец

РАЗБОР ЭЛЕМЕНТОВ БЛОК-СХЕМЫ

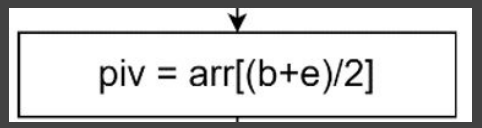
1 Первый элемент («пуск») мы используем для обозначения начала работы алгоритма.

2 Во втором элементе, называемом «процесс», мы задаем переменные, которые в дальнейшем будем применять для работы алгоритма.




```
int b, e, l, r, t, arr[] = { 5, 7, 8, 4, 9 };
```

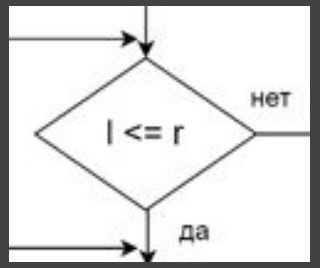
3 Третий элемент («процесс») служит нам для обозначения вычислительных действий, а именно для нахождения номера опорного элемента (piv).




```
int piv = arr[(b + e) / 2];
```

РАЗБОР ЭЛЕМЕНТОВ БЛОК-СХЕМЫ

4 Следующий элемент («решение») используется нами для проверки условия: номер левого элемента меньше либо равен номеру правого элемента массива. От «решения» отходит две ветви: по первой мы идем в случае, если результат проверки условия положительный (элемент, к которому отходит эта ветвь будет рассмотрен нами в пункте **5**), по второй – если отрицательный (элемент, к которому отходит эта ветвь будет рассмотрен нами в пункте **12**).

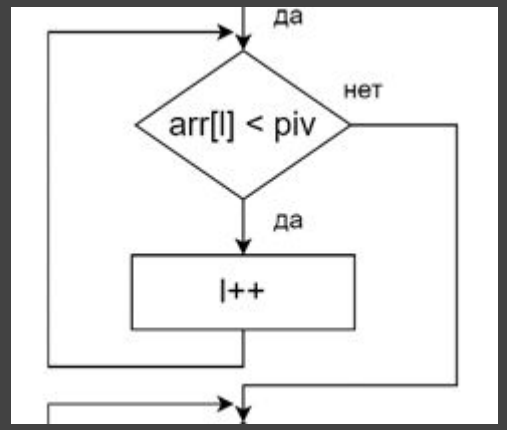



```
while (l <= r) //заголовок внешнего цикла
    {

    }
```

РАЗБОР ЭЛЕМЕНТОВ БЛОК-СХЕМЫ

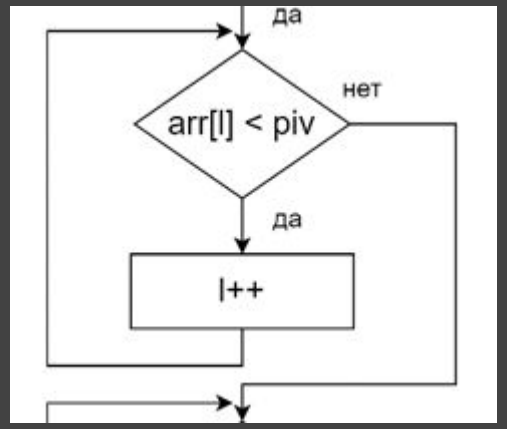
5 Пятым по очереди мы снова применяем элемент «решение» для проверки условия: значение элемента с номером левого элемента массива меньше номера опорного элемента. От «решения» отходит две ветви: по первой мы идем в случае, если результат проверки условия положительный (элемент, к которому отходит эта ветвь будет рассмотрен нами в пункте **6**), по второй – если отрицательный (элемент, к которому отходит эта ветвь будет рассмотрен нами в пункте **7**).




```
while (arr[l] < piv) // первый внутренний цикл
{
    l++; // тело первого внутреннего цикла
}
```

РАЗБОР ЭЛЕМЕНТОВ БЛОК-СХЕМЫ

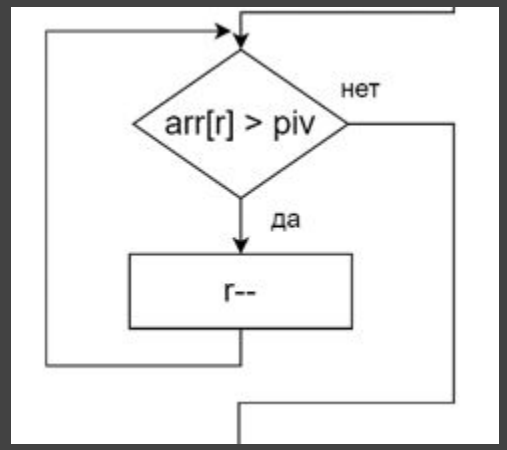
6 Следующим мы используем такой элемент блок-схемы как «процесс» для того, чтобы увеличить номер левого элемента на единицу. На «процессе» завершается один из внутренних циклов нашего алгоритма, поэтому от него отходит зацикливающая ветвь, которая подводит к предыдущему элементу «решение» для очередной проверки условия.




```
i++;
```

РАЗБОР ЭЛЕМЕНТОВ БЛОК-СХЕМЫ

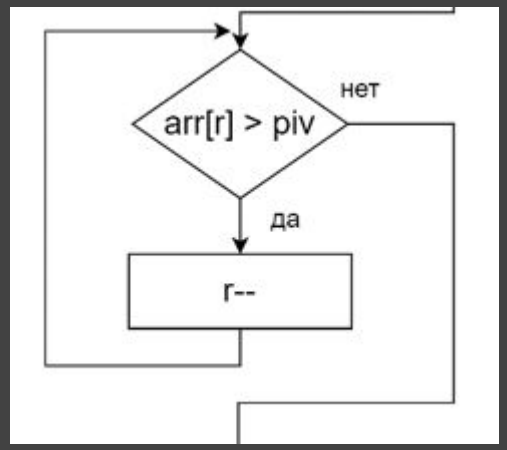
7 Далее мы еще раз используем элемент «решение» для проверки условия: значение элемента с номером правого элемента массива больше номера опорного элемента. От «решения» отходит две ветви: по первой мы идем в случае, если результат проверки условия положительный (элемент, к которому отходит эта ветвь будет рассмотрен нами в пункте **8**), по второй – если отрицательный (элемент, к которому отходит эта ветвь будет рассмотрен нами в пункте **9**).




```
while (arr[r] > piv) // второй внутренний цикл
{
    r--; // тело второго внутреннего цикла
}
```

РАЗБОР ЭЛЕМЕНТОВ БЛОК-СХЕМЫ

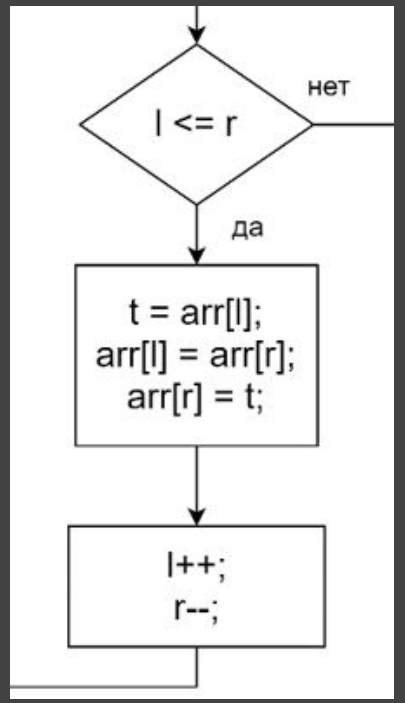
8 Следующим мы вновь применяем элемент - «процесс» для того, чтобы уменьшить номер правого элемента на единицу. На «процессе» завершается второй внутренний цикл алгоритма, поэтому от него отходит зацикливающая ветвь, которая подводит к предыдущему элементу «решение» для очередной проверки условия.




```
r--;
```

РАЗБОР ЭЛЕМЕНТОВ БЛОК-СХЕМЫ

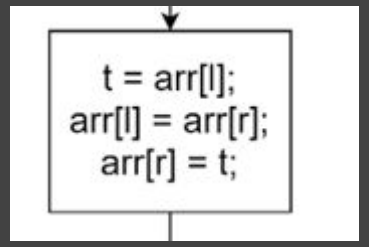
9 Девятым по очереди идет элемент «решение», необходимый нам для проверки следующего условия: номер левого элемента меньше либо равен номеру правого элемента массива. От «решения» отходит две ветви: по первой мы идем в случае, если результат проверки условия положительный (элемент, к которому отходит эта ветвь будет рассмотрен нами в пункте **10**), по второй – если отрицательный (элемент, к которому отходит эта ветвь будет рассмотрен нами в пункте **12**).




```
if (l <= r) // условие
{
    t = arr[l];   // выполняем перестановку
    arr[l] = arr[r];
    arr[r] = t;
    l++; // изменяем значения индексов элементов
    r--;
}// конец внешнего цикла
```

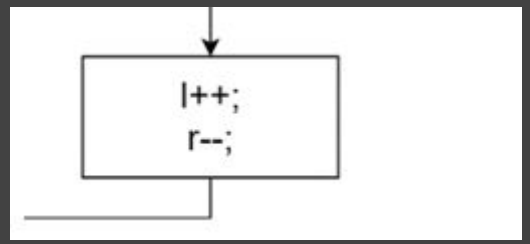
РАЗБОР ЭЛЕМЕНТОВ БЛОК-СХЕМЫ

10 Далее мы используем «процесс» для осуществления перестановки элементов массива.




```
t = arr[l];
arr[l] = arr[r];
arr[r] = t;
```

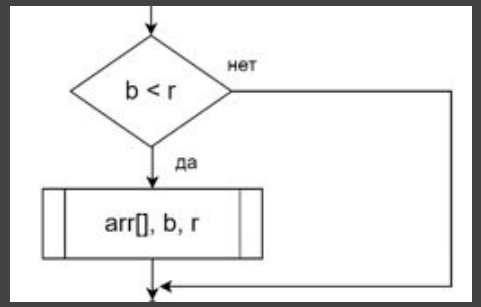
11 Следующим на очереди мы снова применяем элемент «процесс» для того, чтобы увеличить номер левого элемента на единицу и уменьшить номер правого элемента на единицу. На «процессе» завершается внешний цикл нашего алгоритма, поэтому от него отходит зацикливающая ветвь, которая подводит к элементу «решение» (который мы обсуждали в **4** пункте) для очередной проверки условия.




```
l++;
r--;
```

РАЗБОР ЭЛЕМЕНТОВ БЛОК-СХЕМЫ

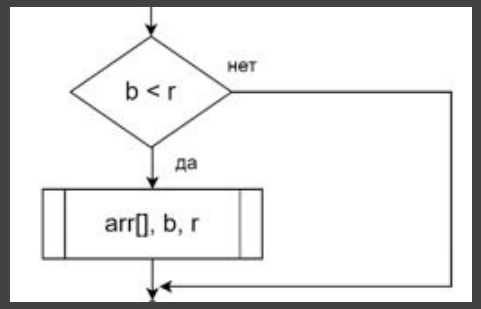
12 В представленном далее элементе – «решение» мы проверяем следующее условие: индекс первого элемента массива меньше индекса правого элемента массива. От «решения» отходит две ветви: по первой мы идем в случае, если результат проверки условия положительный (элемент, к которому отходит эта ветвь будет рассмотрен нами в пункте 13), по второй – если отрицательный (элемент, к которому отходит эта ветвь будет рассмотрен нами в пункте 14).




```
if (b < r) //условие
{
      qsort(arr, b, r);// вызываем функцию Быстрой сортировки
}
```

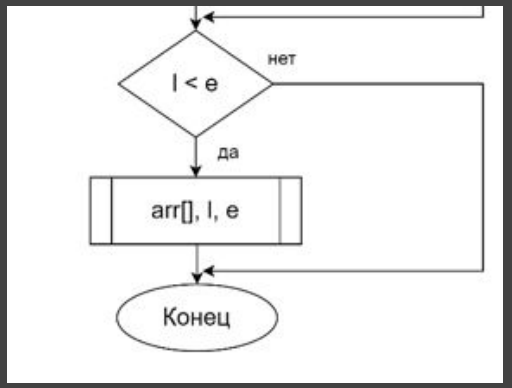
РАЗБОР ЭЛЕМЕНТОВ БЛОК-СХЕМЫ

13 Далее мы обращаемся к такому элементу как «предопределенный процесс» для вызова курсор (функция вызывает сама себя).




```
qsort(arr, b, r);
```

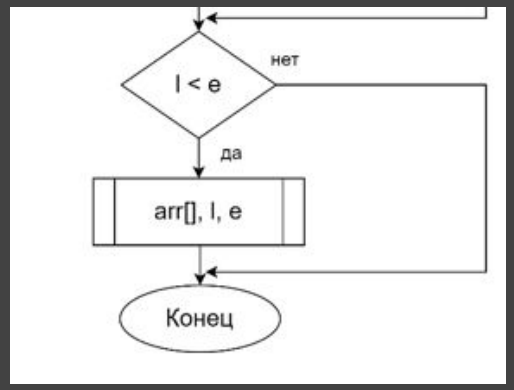
14 Следующим мы еще раз применяем элемент «решение» для проверки условия: индекс левого элемента массива меньше индекса последнего элемента массива. От «решения» отходит две ветви: по первой мы идем в случае, если результат проверки условия положительный (элемент, к которому отходит эта ветвь будет рассмотрен нами в пункте 15), по второй – если отрицательный (элемент, к которому отходит эта ветвь будет рассмотрен нами в пункте 16).




```
if (l<e) //условие
{
    qsort(arr, l, e);
}
```

РАЗБОР ЭЛЕМЕНТОВ БЛОК-СХЕМЫ

15 Следующим мы снова используем элемент «предопределенный процесс» для вызова курсор (функция вызывает сама себя), используя уже другие переменные




```
qsort(arr, l, e);
```

16 Последний элемент («пуск») мы применяем для обозначения конца работы алгоритма.

ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

```
#include <iostream>
using namespace std;

void qsort(int* arr, int b, int e) {
    int piv = arr[(b + e) / 2];
    int l = b;
    int r = e;
    while (l <= r)
    {
        while (arr[l] < piv) l++;
        while (arr[r] > piv) r--;
        if (l <= r) // условие
        {
            int t = arr[l];
            arr[l] = arr[r];
            arr[r] = t;
            l++;
            r--;
        }
    }
}
```

```
//продолжение кода
int main()
{
    setlocale(LC_ALL, "Russian");
    int b, e, l, r, t, arr[] = { 5, 7,
8, 4, 9 };
    if (b < r) {
        qsort(arr, b, r);
    }
    if (l<e)
    {
        qsort(arr, l, e);
    }

}
```