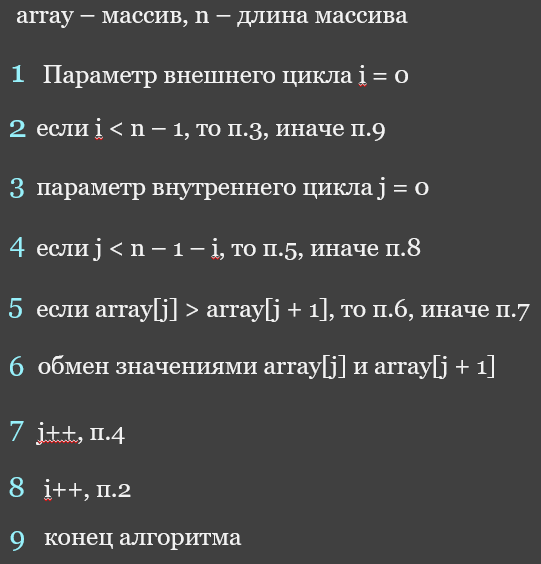
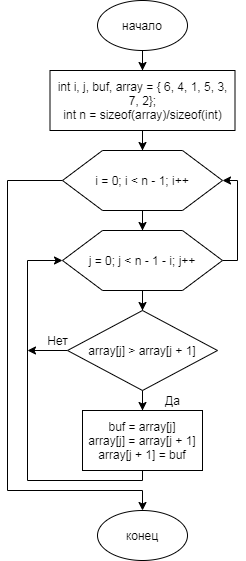
# Пузырек

**Идея алгоритма:**Сортировка пузырьком - это метод сортировки массивов путем последовательного сравнения и обмена соседних элементов, если предшествующий оказывается больше последующего. В процессе выполнения данного алгоритма элементы с большими значениями оказываются в конце массива, а элементы с меньшими значениями постепенно перемещаются по направлению к началу массива.

**Словесное представление:**

Блок-схема:  
   
**Программная реализация:**int main(){

int i, j, buf, arr[] = { 6, 4, 1, 5, 3, 7, 2};

int n = sizeof(arr) / sizeof(int);

for(i = 0; i < n - 1; i++){

for(j = 0; j < n - 1 - i; j++){

if(arr[j] > arr[j + 1]){

buf = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = buf;

}

}

}

}

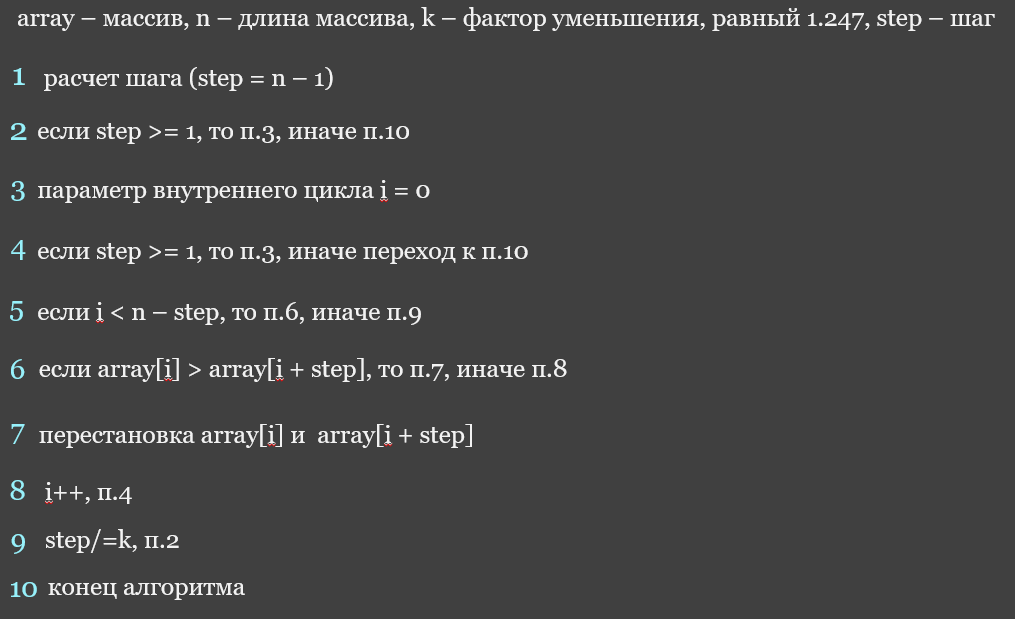
# Расческа

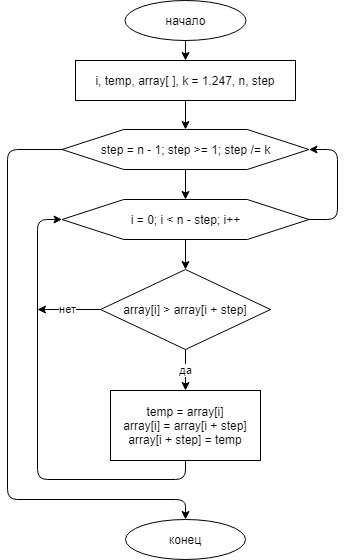
**Идея алгоритма**:

Сортировка расчёской улучшает сортировку пузырьком, и конкурирует с алгоритмами,  
подобными быстрой сортировке. Основная идея — устранить черепах, или маленькие  
значения в конце списка, которые крайне замедляют сортировку пузырьком.

В сортировке пузырьком, когда сравниваются два элемента, промежуток (расстояние друг от друга) равен 1. Основная идея сортировки расчёской в том, что этот промежуток может быть  
гораздо больше, чем единица.

Алгоритм является модификацией «пузырька». Отличие алгоритмов состоит в том, что  
сравниваются не соседние элементы, а отстоящие друг от друга на определённую величину,  
или шаг (назовём его step). Алгоритм реализован с помощью двух циклов. Окончание  
внешнего цикла (и алгоритма) происходит тогда, когда step станет меньше 1. На первой  
итерации расстояние (step) максимально возможное (размер массива – 1), а на последующих  
итерациях оно изменяется по формуле step /= k (дробная часть отбрасывается). k – это фактор  
уменьшения, константа, равная 1.2473309 (при написании программы можно использовать  
примерное значение, равное 1.247). Во внутреннем цикле движение происходит от начала к  
концу, перемещаясь на step. Если значение текущего элемента больше, чем значение  
элемента через step шагов от текущего, то сравниваемые элементы меняются местами.  
Условием продолжения цикла является условие i < n – step (где i – номер текущего элемента).

**Словесное представление:**

**Блок-схема:  
**

**Программная реализация:  
int main() {**

**int i, temp, step;**

**int array[] = { 1, 4, 5, 0, 3, 2 };**

**int n = sizeof(array) / sizeof(int);**

**float k = 1.247;**

**for (step = n - 1; step >= 1; step /= k) {**

**for (i = 0; i < n – step; i++) {**

**if (array[i] > array[i + step]) {**

**temp = array[i];**

**array[i] = array[i + step];**

**array[i + step] = temp;**

**}**

**}**

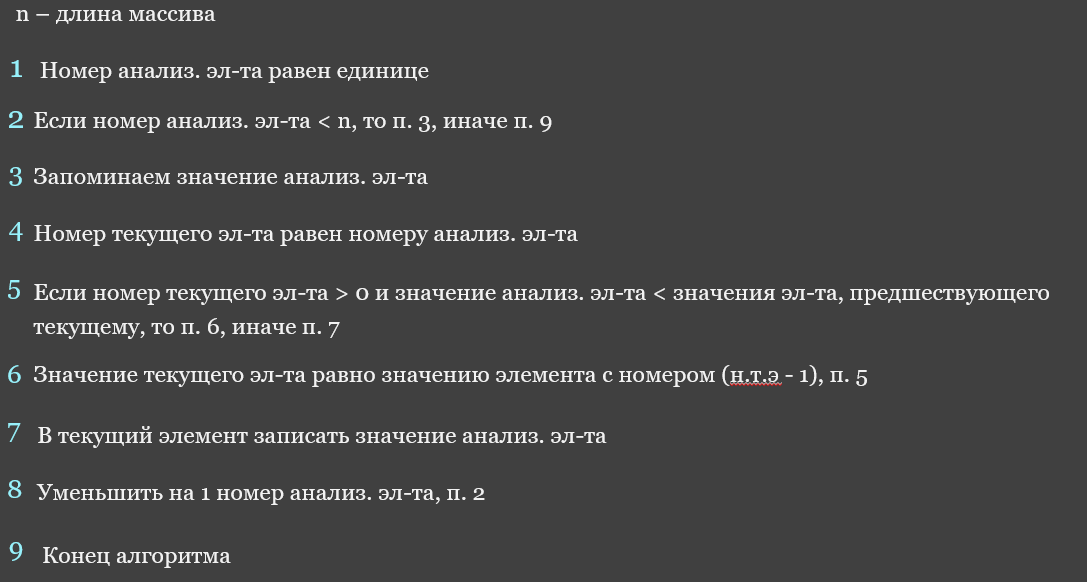
**}**

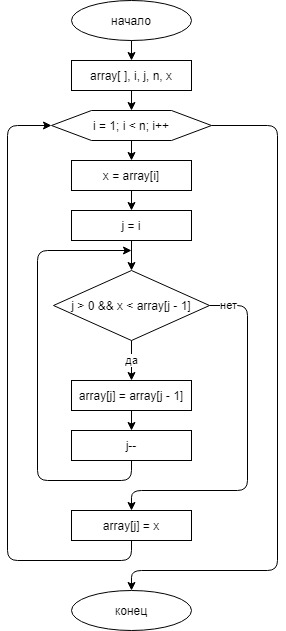
**return 0;**

**}**

# Вставки

**Идея алгоритма:**Сортируемый массив можно разделить на две части — отсортированная часть и  
неотсортированная. В начале сортировки первый элемент массива считается  
отсортированным, все остальные — не отсортированные. Начиная со второго элемента  
массива и заканчивая последним, алгоритм вставляет неотсортированный элемент массива  
в нужную позицию в отсортированной части массива.

**Словесное представление:**

Блок-схема:  
  
**Программная реализация:**int main()

{

int i, j, x;

int array[ ] = { 1, 4, 6, 7, 8, 3};

int n = sizeof(array) / sizeof(int);

for (i = 1; i < n; i++) {

x = array[i];

j = i;

while (j > 0 && x < array[j – 1]) {

array[j] = array[j – 1];

j--;

}

array[j] = x;

}

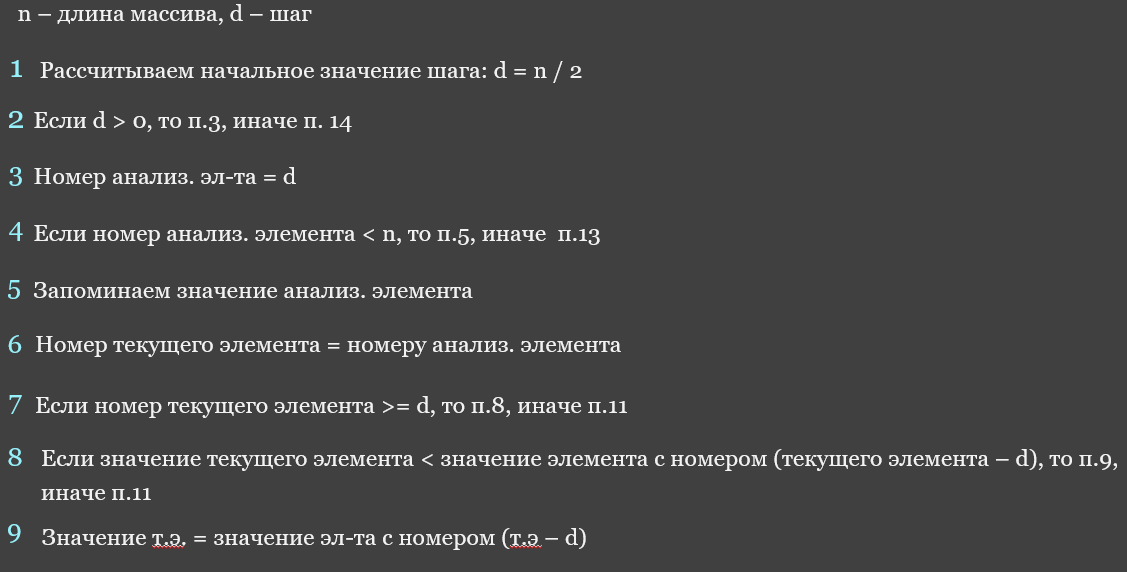
return 0;

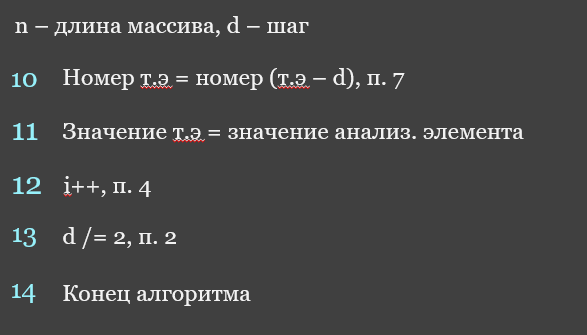
}

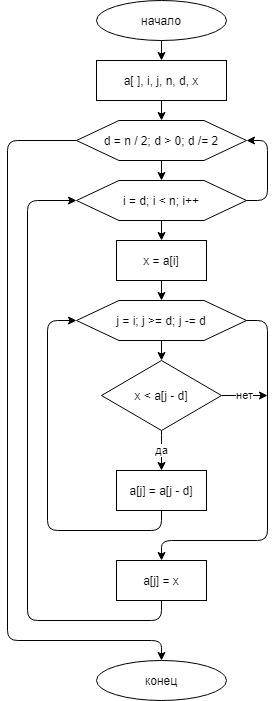
# Шелла

**Идея и понятие алгоритма:  
Сортировка Шелла** (англ. *Shell sort*) — алгоритм сортировки, являющийся  
усовершенствованным вариантом сортировки вставками. Идея метода Шелла состоит в  
сравнении элементов, стоящих не только рядом, но и на определённом расстоянии друг от  
друга. Иными словами — это сортировка вставками с предварительными «грубыми»  
проходами. Аналогичный метод усовершенствования пузырьковой  
сортировки называется сортировка расчёской.

Алгоритм сортирует элементы отстоящие друг от друга на некотором расстоянии.  
Затем сортировка повторяется при меньших значениях шага, и в конце процесс сортировки  
Шелла завершается при шаге, равном 1 (а именно обычной сортировкой вставками). Шелл  
предложил такую последовательность размера шага: N/2, N/4, N/8 …, где N – количество  
элементов в сортируемом массиве.

**Словесное представление:**



Блок-схема:  
  
**Программная реализация:**int main()

{

int i, x, d, j;

int a[ ] = { 1, 4, 6, 7, 8, 3};

int n = sizeof(a) / sizeof(int);

for (d = n / 2; d > 0; d /= 2) {

for (i = d; i < n; i++) {

x = a[i];

for (j = i; j >= d; j -= d) {

if (x < a[j - d])

a[j] = a[j - d];

else break;

}

a[j] = x;

}

}

return 0;

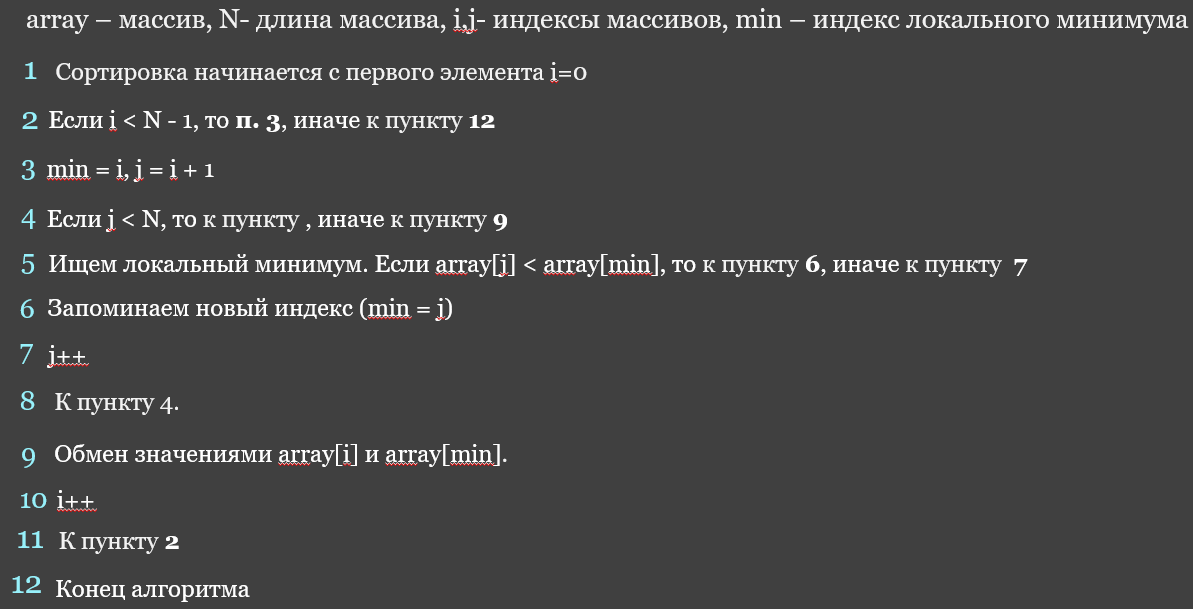
}

# Выбором

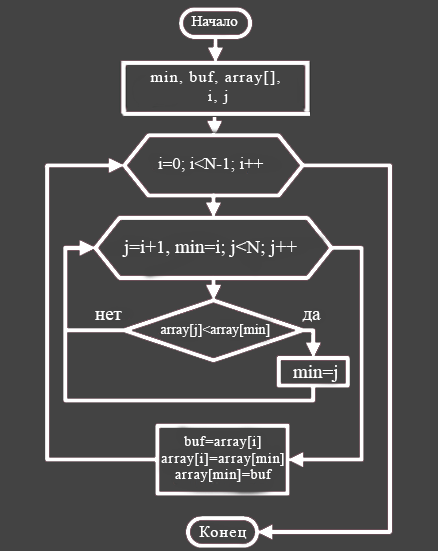
**Идея алгоритма:**Сортировка выбором (Selection sort) — алгоритм сортировки. Может быть как  
устойчивый, так и неустойчивый. На массиве из n элементов имеет время выполнения в  
худшем, среднем и лучшем случае Θ(n2), предполагая, что сравнения делаются за  
постоянное время

Это возможно, самый простой в реализации алгоритм сортировки. Как и в  
большинстве других подобных алгоритмов, в его основе лежит операция сравнения.  
Сравнивая каждый элемент с каждым, и в случае необходимости производя обмен, метод  
приводит последовательность к необходимому упорядоченному виду.

**Пусть имеется массив A размером N, тогда сортировка выбором сводится к  
следующему:**• берем первый элемент последовательности A[i], здесь i – номер элемента, для  
первого i равен 1;  
• находим минимальный (максимальный) элемент последовательности и запоминаем  
его номер;  
• если номер первого элемента и номер найденного элемента не совпадают, тогда два  
этих элемента обмениваются значениями, иначе никаких манипуляций не  
происходит;  
• увеличиваем i на 1 и продолжаем сортировку оставшейся части массива.  
С каждым последующим шагом размер подмассива, с которым работает алгоритм,  
уменьшается.

**Словесное представление:**

Блок-схема:



**Программная реализация:**

**int main() {**

**int i, j, min, buf, N = 7;**

**int arr[] = {6, 4, 1, 5, 3, 7, 2};**

**for (i = 0; i < N - 1; i++) {**

**for (j = i + 1, min = i; j < N; j++){**

**if (arr[j] < arr[min])**

**min = j;**

**}**

**buf = arr[i];**

**arr[i] = arr[min];**

**arr[min] = buf;**

**}**

**return 0;**

**}**

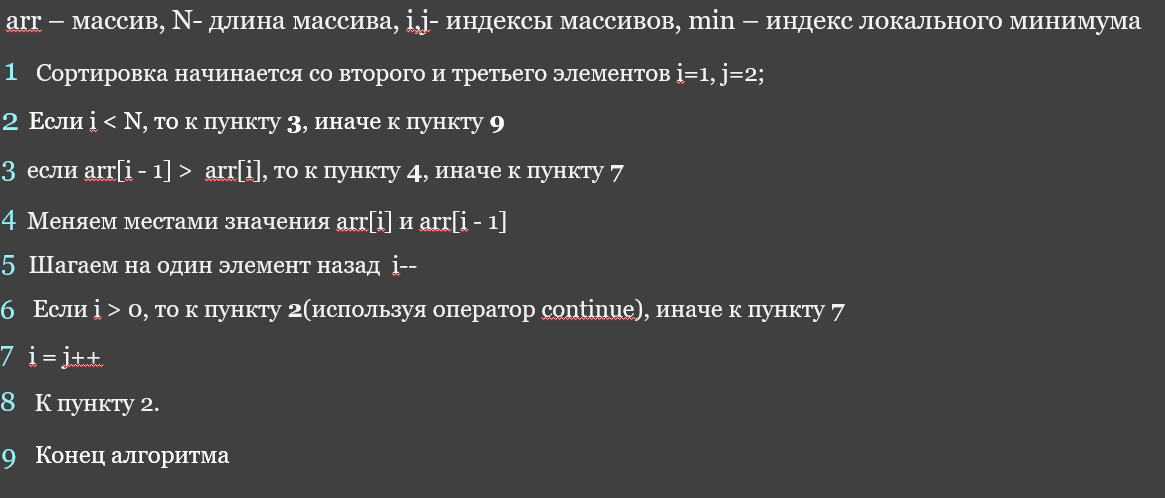
# Гномья

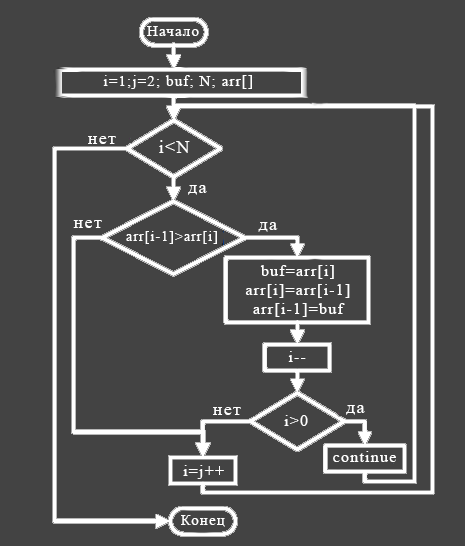
**Идея алгоритма:  
Гномья сортировка** (англ. Gnome sort) — алгоритм сортировки, похожий  
на сортировку вставками, но в отличие от последней перед вставкой на нужное место  
происходит серия обменов, как в сортировке пузырьком. Название происходит от  
предполагаемого поведения садовых гномов при сортировке линии садовых горшков.

Алгоритм концептуально простой, не требует вложенных циклов. Время работы O(n²). На  
практике алгоритм может работать так же быстро, как и сортировка вставками.

Пусть имеется массив A размером N, тогда сортировка выбором сводится к  
следующему:  
• Смотрим на текущий и предыдущий элемент массива:  
• если они в правильном порядке, шагаем на один элемент вперед,  
• иначе меняем их местами и шагаем на один элемент назад.  
• Граничные условия:  
• если нет предыдущего элемента, шагаем вперёд;  
• если нет следующего элемента, стоп.

Это оптимизированная версия с использованием переменной j, чтобы разрешить прыжок  
вперёд туда, где он остановился до движения влево, избегая лишних итераций и сравнений.

**Словесное описание:**

**Блок-схема:**

**Программная реализация:**

#include <iostream>  
using namespace std;  
int main()

{

int N, j=2, i=1, buf;

arr = new int[6, 4, 1, 5, 3, 7, 2];

N=7;

while (i<N) {

if (arr[i-1]>arr[i]) {

buf=arr[i];

arr[i]=arr[i-1];

arr[i-1]=buf;

i--;

if (i>0) continue;

}

i=j++;

}  
 return 0;

}

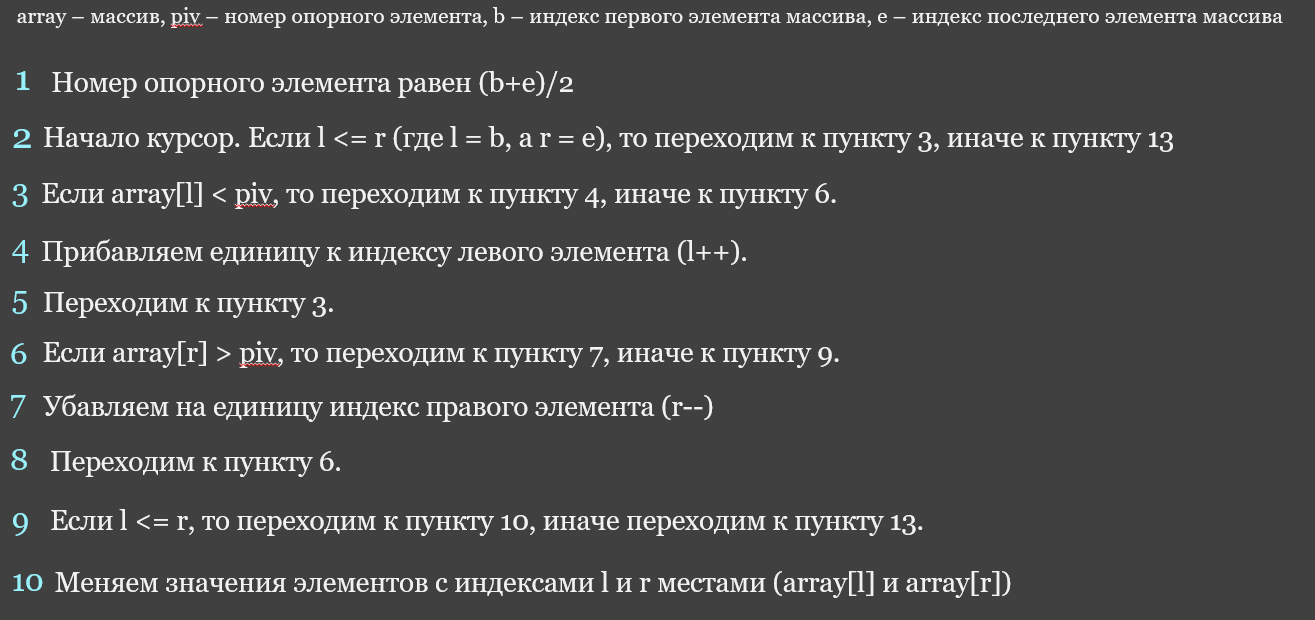
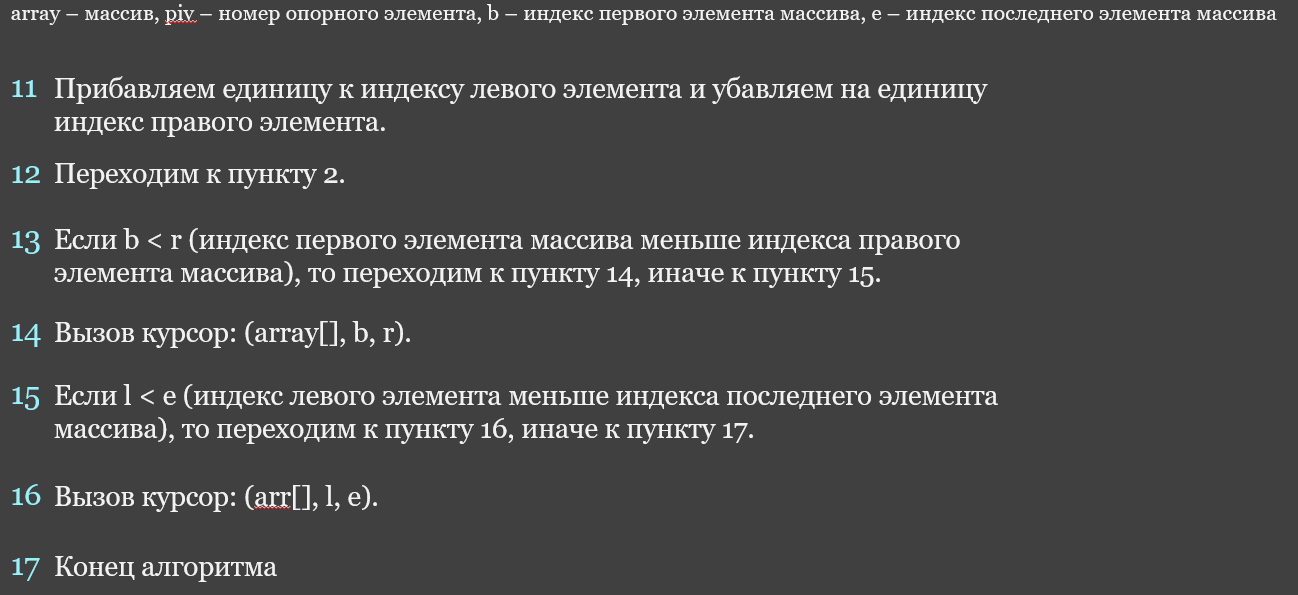
# Быстрая

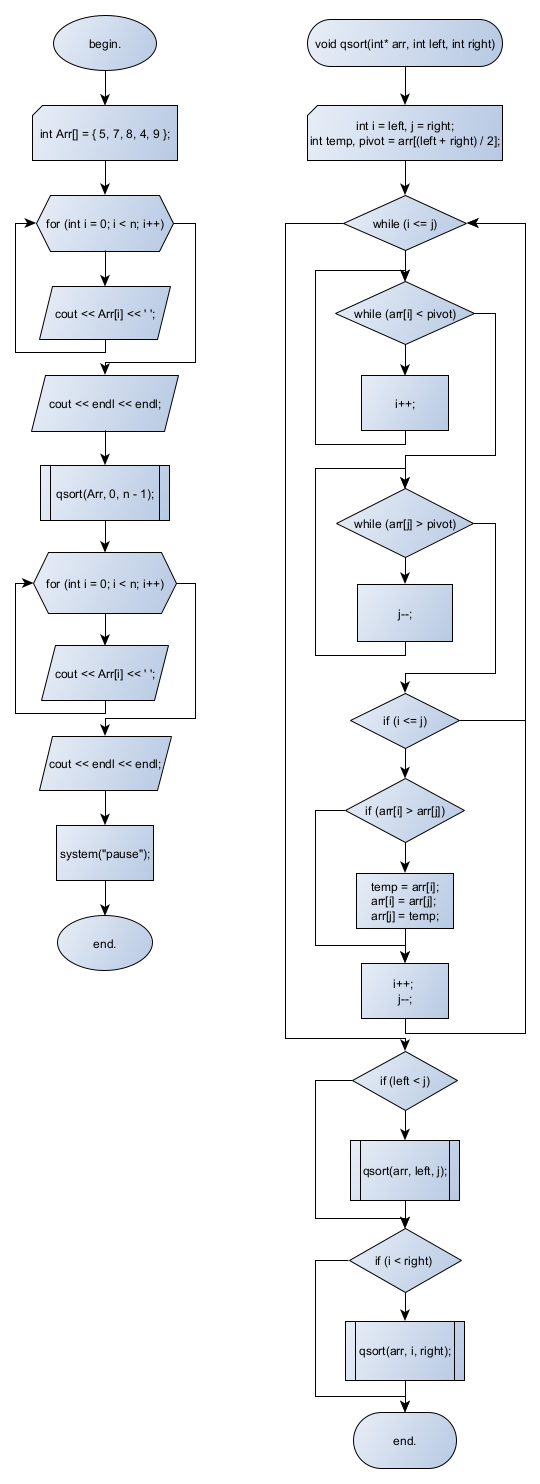
**Идея алгоритма:**

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.

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.

**Словесное представление:**  


**Блок-схема:**

**Программная реализация:**

#include <iostream>//подключение необходимых библиотек

#include <vector>

#include <chrono>

using namespace std;

typedef std::chrono::high\_resolution\_clock Clock;

vector<int> vec = { 4,5,7,6,3,2,1,8,9 };

int q=9, peremen;

void qsort(vector<int>& vec, int left, int right)//объявление подфункции сортировки

{

int i = left, j = right;//объявление переменных крайнего левого положения и крайнего праввого положения

int temp, pivot = vec[(left + right) / 2];//поиск центра(поиск опорного элемента)

while (i <= j)//цикл выполняется, пока границы не сомкнутся

{

while (vec[i] < pivot) //если значение анализируемого элемента из левой части меньше значения опорного элемента

i++;//мы сдвигаем левую границу на одну единицу вправо

while (vec[j] > pivot) //если значение анализируемого элемента из правой части больше значения опорного элемента

j--;//мы сдвигаем правую границу на одну единицу влево

//сдвигаем границы к центру - к опорному элементу

if (i <= j)

{

if (vec[i] > vec[j])//выполнение обмена, если значение текущего элемента в левой части больше значения текущего элемента в правой части

{

temp = vec[i];//обмен значений переменных с использованием буферной переменной temp

vec[i] = vec[j];

vec[j] = temp;

}

i++;

j--;//уменьшнеие границ диапазона

}

}

//if (left < j)

// qsort(vec, left, j);//если обнаружено нарушение порядка в одной из частей, вызывается функция сортировки определенной части

//if (i < right)

// qsort(vec, i, right);//если обнаружено нарушение порядка в одной из частей, вызывается функция сортировки определенной части

}

//-------------------------------------------------------------------

int main()//начало главной функции

{

cout << "Enter the amount of element:";

/\*cin >> q;

for (int k = 0; k < q; k++)

{

peremen = rand() % q + 1;

vec.push\_back(peremen);

}\*/

//vec[] = { 4,7,3,2,1 };

cout << "Array before sort:" << endl;

for (int i = 0; i < q; i++)

cout << vec[i] << ' ';

cout << endl << endl;

auto t1 = Clock::now();

qsort(vec, 0, vec.size() - 1); // вызов функции сортировки

auto t2 = Clock::now();

cout << "Array after sort:" << endl;

for (int i = 0; i < q; i++)

cout << vec[i] << ' ';

cout << endl << endl << "Duration(nanoseconds) is: " << std::chrono::duration\_cast<std::chrono::nanoseconds>(t2 - t1).count() << endl;

system("pause");

return 0;

}