

Федеральное государственное автономное образовательное учреждение  
высшего образования «Пермский национальный исследовательский  
политехнический университет»

Лабораторная работа  
«Простые сортировки»

Выполнил:

студент первого курса

ЭТФ группы РИС-23-36

Акбашева Софья Руслановна

Проверила:

Доцент кафедры ИТАС О. А. Полякова

Пермь, 2024

## Простые сортировки

**Цель:** знакомство с простыми сортировками, в числе которых сортировки: слиянием, подсчетом, быстрая, блочная, Шелла, Хоара.

**Постановка задачи:** реализовать сортировку массива из 25 элементов, с помощью сортировок: слиянием, подсчетом, быстрая, блочная Шелла, Хоара. Необходимо зафиксировать быстродействие каждой сортировки.

### Анализ задачи

1) Программа запрашивает у пользователя, каким способом он хочет создать массив: ввести числа вручную или создать массив случайно. Если необходимо создать массив случайно, то используется датчик случайных чисел, иначе пользователь вводит каждый элемент с клавиатуры с помощью арифметического цикла.

2) Вывод массива осуществляется с помощью арифметического цикла.

3) Программа запрашивает у пользователя, какую сортировку он хочет использовать. После, запускается соответствующая сортировка.

4) В коде используется функция `chrono::steady_clock::now()` для измерения времени выполнения программы. Разница между начальным и конечным временем измеряется с помощью `end_time_recursive - start_time_recursive`. Затем эта разница преобразуется в миллисекунды с помощью `chrono::duration <double, milli>(diff_time).count()`.

5) Сортировка слиянием. Функция `Merge_mas` выполняет слияние двух отсортированных подмассивов массива `mas`. Функция `Merge_Sort` является рекурсивной и выполняет сортировку массива, разделяя его на две части и вызывая `Merge_Sort` для каждой из них, а затем вызывая `Merge_mas` для слияния отсортированных подмассивов. В функции `Merge_mas` создается новый массив `new_mas` для хранения результатов слияния. Среднее арифметическое используется для определения границы между левой и правой частями массива. Затем в арифметическом цикле выполняется слияние двух частей массива. После завершения слияния, результаты записываются обратно

в исходный массив `mas`. Наконец, память, выделенная для `new_mas`, освобождается с помощью `delete[] new_mas`. В функции `Merge_Sort` выполняется рекурсивный вызов `Merge_Sort` для левой и правой частей массива, а затем вызывается `Merge_mas` для слияния отсортированных подмассивов.

6) Быстрая сортировка. Функция `Quick_Sort` принимает массив `mas` и его размер `size` в качестве аргументов. Сначала она определяет начальный и конечный индексы массива, а также центральный элемент. Затем выполняется итерационный цикл, который делит массив на две части: левую, содержащую элементы, меньшие центрального, и правую, содержащую элементы, большие центрального. В итерационном цикле выполняются два вложенных итерационных цикла, которые перемещают указатели `first_index` и `last_index` в соответствующие части массива. Если `first_index` меньше `last_index`, то выполняется обмен элементов между ними. После завершения итерационного цикла, если `last_index` больше нуля, вызывается рекурсивная версия `Quick_Sort` для левой части массива. Аналогично, если `first_index` меньше `size`, вызывается рекурсивная версия `Quick_Sort` для правой части массива.

7) Сортировка подсчетом. Функция `Sort_by_Count` принимает массив `A_mas` и его размер `mas_size` в качестве аргументов. Сначала она определяет максимальный и минимальный элементы массива в арифметическом цикле. Затем она создает новый массив `B_mas` для подсчета количества каждого значения в массиве `A_mas`. В арифметическом цикле выполняется подсчет количества каждого значения в массиве `A_mas` и запись результатов в массив `B_mas`. Затем выполняется арифметический цикл, который записывает значения из массива `B_mas` в массив `A_mas` нужное количество раз. В конце функции `Sort_by_Count` память, выделенная для массива `B_mas`, освобождается с помощью `delete[] B_mas`.

8) Блочная сортировка. Функция `Block_Sort` принимает массив `mas` и его размер `size_mas` в качестве аргументов. В начале кода объявляются две переменные `max_element` и `min_element`, которые используются для

определения максимального и минимального элементов в массиве. Затем определяется переменная `the_numbers_in_the_block`, которая определяет, сколько чисел может поместиться в один блок. `total_blocks` используется для определения количества блоков, которые будут созданы. Затем создается двумерный массив `blocks`, который будет использоваться для хранения блоков. Каждый элемент массива `blocks` — это новый массив, который будет использоваться для хранения чисел в соответствующем блоке. Далее происходит распределение чисел по блокам. Для каждого элемента массива `mas` определяется, в какой блок его следует поместить. Затем этот элемент записывается в соответствующий блок. После этого происходит сортировка каждого блока методом вставки. Наконец, происходит освобождение памяти, выделенной для массива `blocks`. Для каждого элемента массива `blocks` вызывается операция `delete`, которая освобождает память, выделенную для соответствующего блока. Затем вызывается операция `delete`, которая освобождает память, выделенную для массива `blocks`.

9) Сортировка методом Шелла. Алгоритм начинается с инициализации переменной `"step"` равной половине размера массива. Затем выполняется арифметический цикл, который продолжается до тех пор, пока `"step"` больше 0. Внутри этого цикла выполняется другой арифметический цикл, который проходит по массиву с шагом `"step"`. Внутри этого цикла, для каждого элемента массива, выполняется еще один арифметический цикл, который проходит с шагом `"step"`. Если значение текущего элемента меньше значения элемента на `"j - step"`, то они меняются местами. Если значение текущего элемента больше или равно значению элемента на `"j - step"`, то цикл прерывается. После завершения этого цикла, значение текущего элемента присваивается обратно в массив. После каждой итерации `"step"` уменьшается вдвое, что позволяет сортировать массив с уменьшающимся шагом. Это приводит к тому, что после каждой итерации массив становится более упорядоченным, что позволяет алгоритму работать быстрее, чем простая сортировка пузырьком.

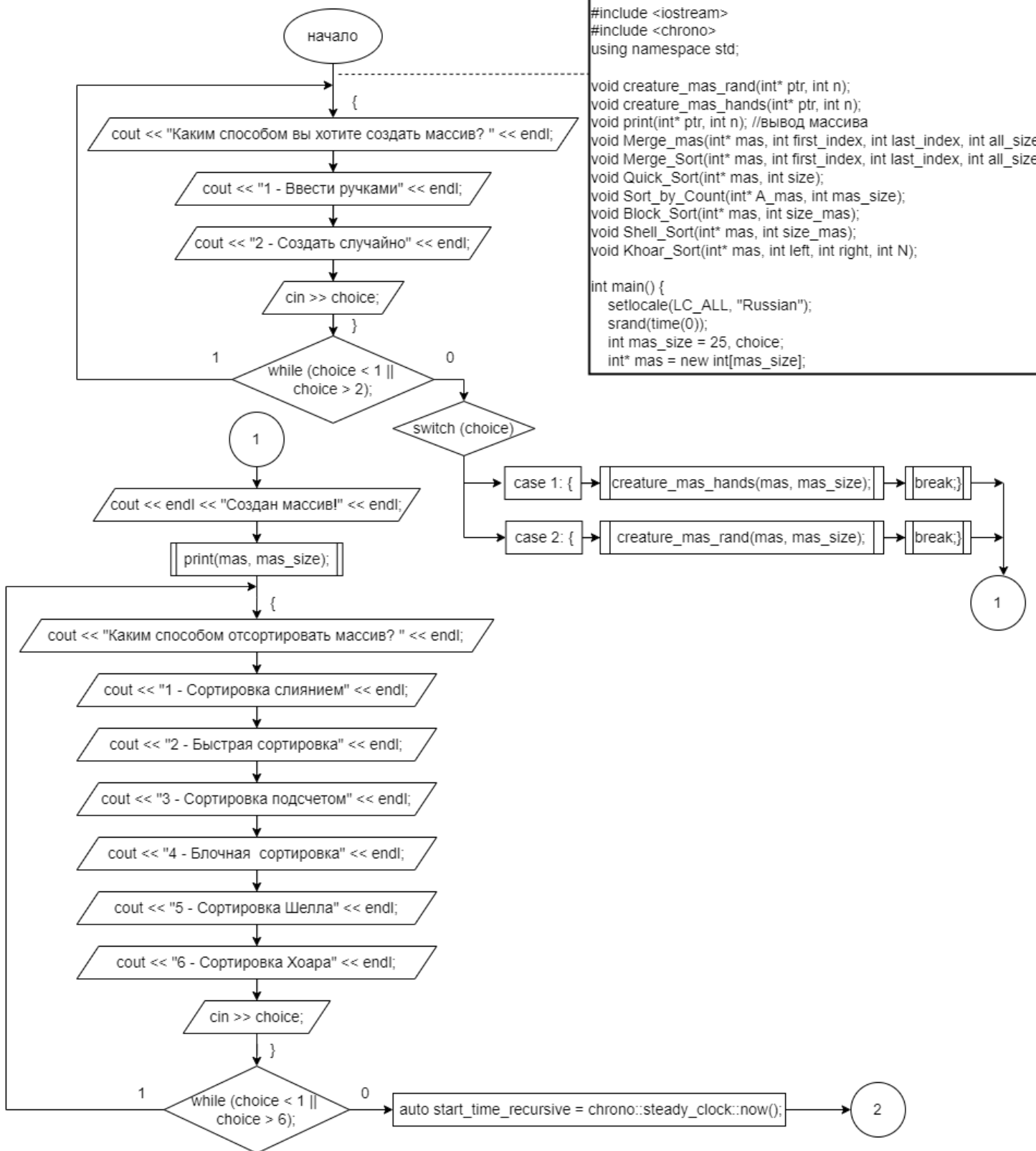
10) Сортировка методом Хоара. Алгоритм начинается с выбора опорного элемента (`support_element`) из середины массива. Затем, с помощью двух вложенных итерационных циклов, код перемещает элементы массива, которые меньше или равны опорному элементу, в левую часть массива, а элементы, которые больше опорного элемента, - в правую часть массива. После этого, если левая и правая границы ( $i$  и  $j$ ) не выходят за пределы массива, происходит обмен элементов на позициях  $i$  и  $j$  с помощью переменной `tmp`. Затем, границы  $i$  и  $j$  увеличиваются и уменьшаются соответственно. После завершения итерационного цикла, код проверяет, не выходят ли границы  $i$  и  $j$  за пределы массива. Если это так, то вызывается рекурсивная функция `Khoar_Sort` для сортировки подмассива.

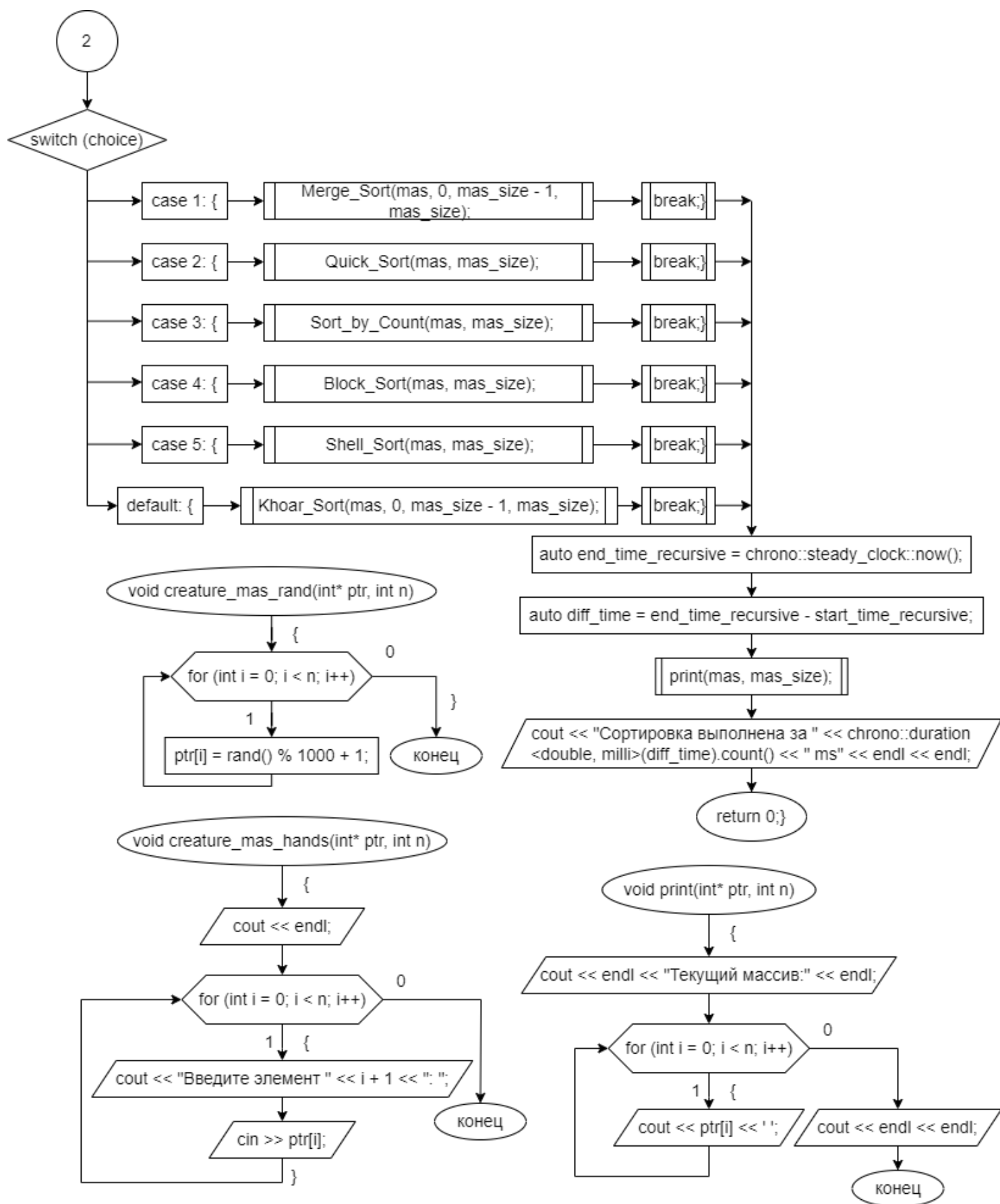
#### **Блок схема**

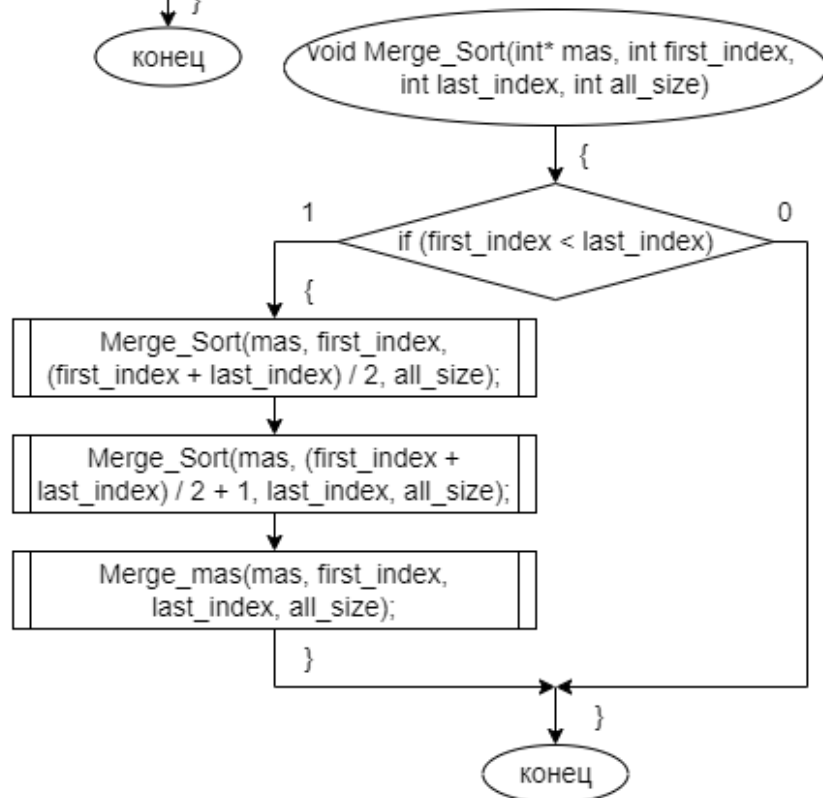
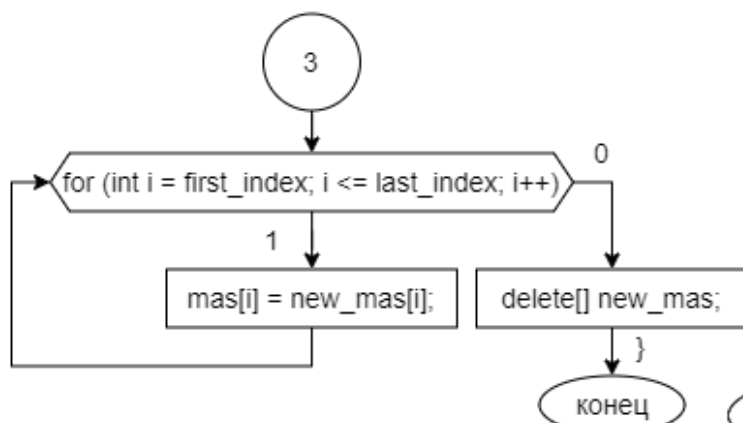
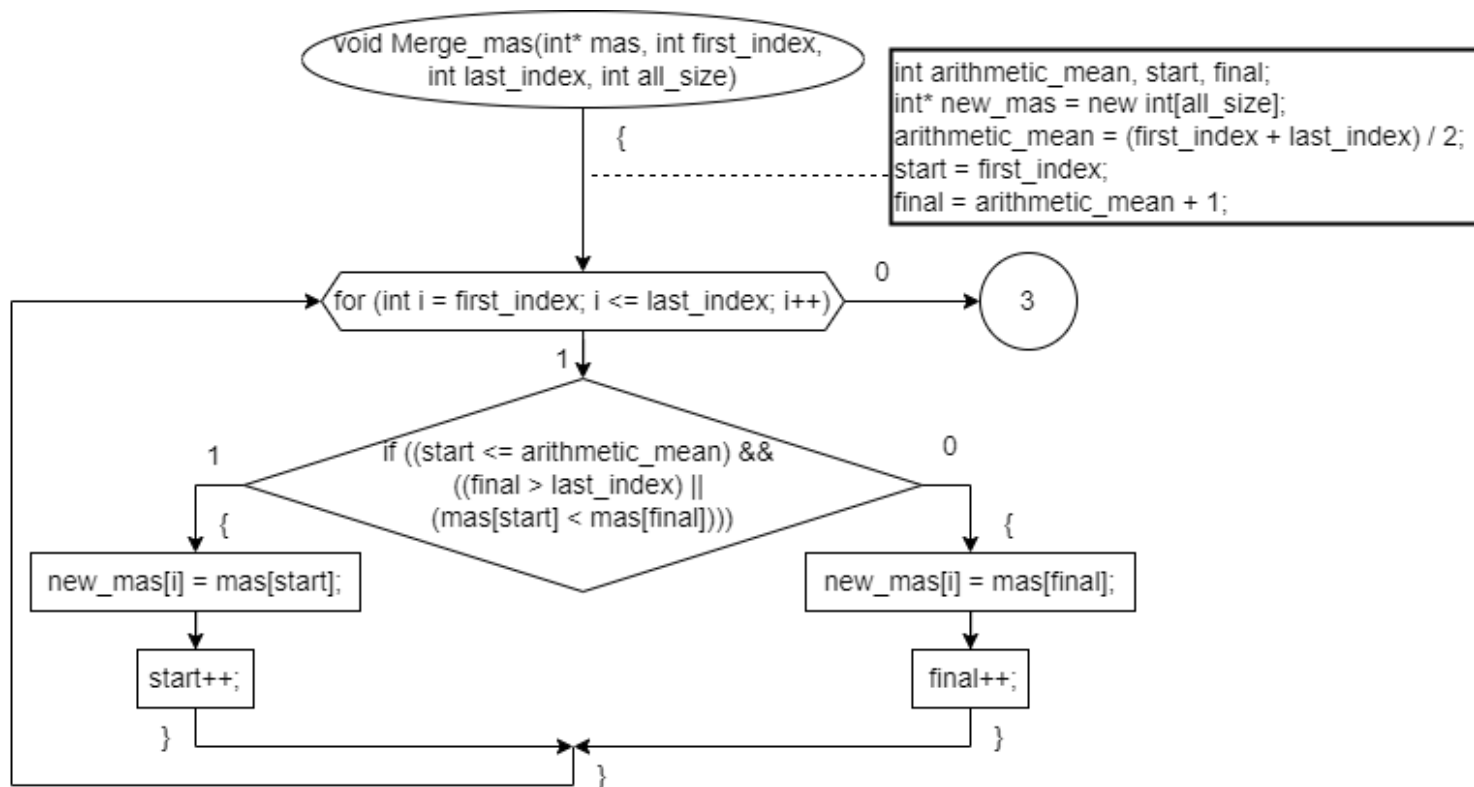
```
#include <iostream>
#include <chrono>
using namespace std;

void creature_mas_rand(int* ptr, int n);
void creature_mas_hands(int* ptr, int n);
void print(int* ptr, int n); //вывод массива
void Merge_mas(int* mas, int first_index, int last_index, int all_size);
void Merge_Sort(int* mas, int first_index, int last_index, int all_size);
void Quick_Sort(int* mas, int size);
void Sort_by_Count(int* A_mas, int mas_size);
void Block_Sort(int* mas, int size_mas);
void Shell_Sort(int* mas, int size_mas);
void Khoar_Sort(int* mas, int left, int right, int N);

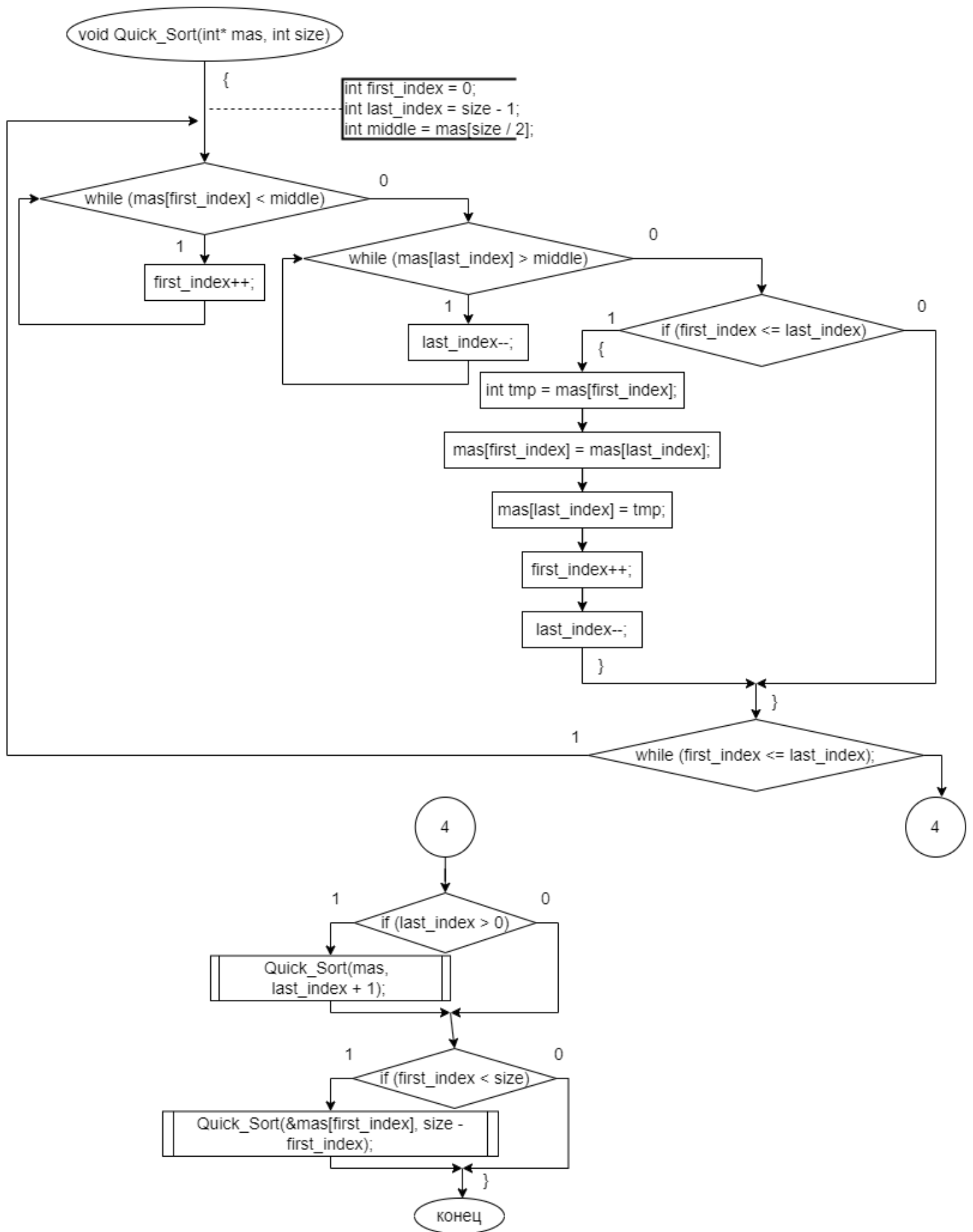
int main() {
    setlocale(LC_ALL, "Russian");
    srand(time(0));
    int mas_size = 25, choice;
    int* mas = new int[mas_size];
```

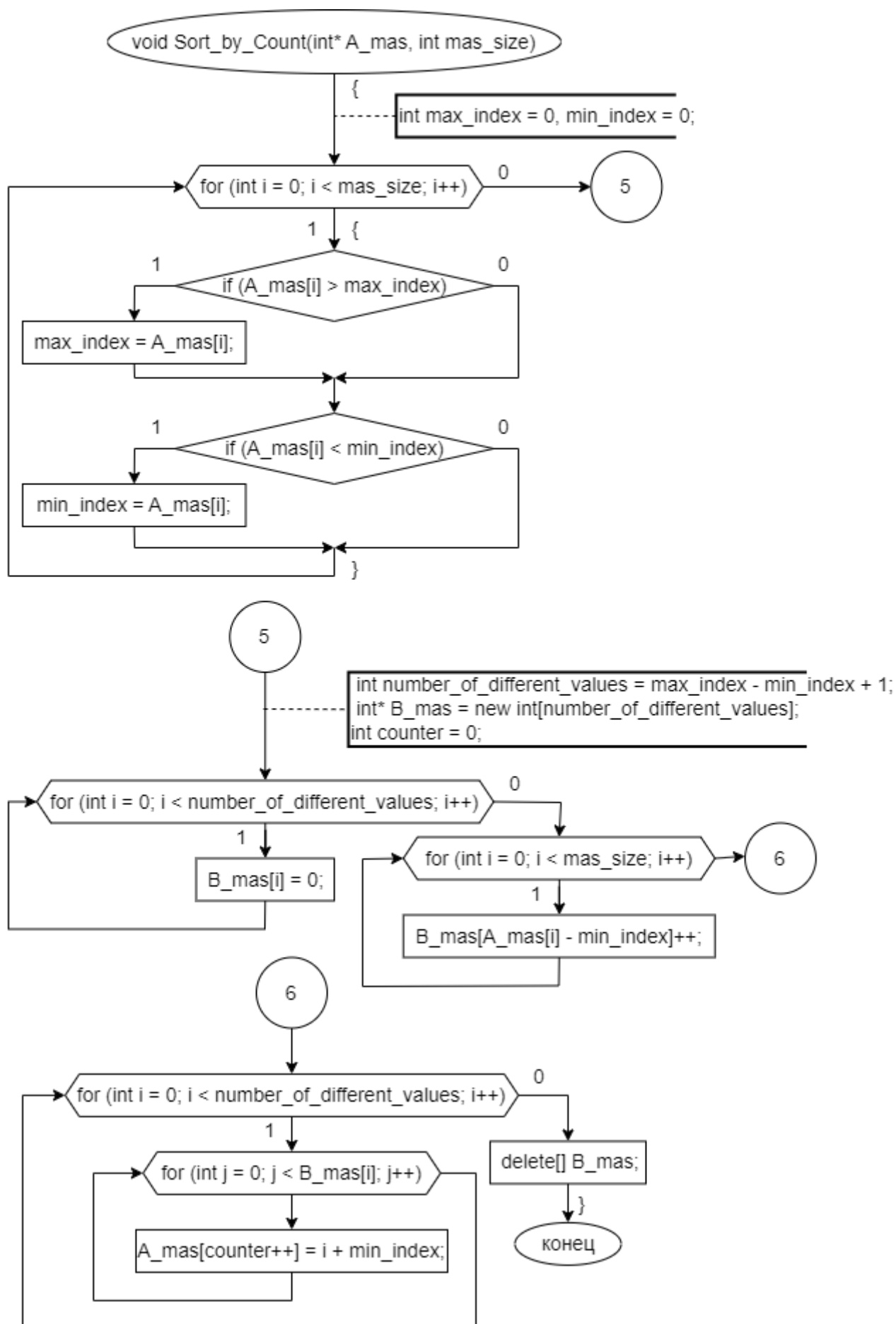












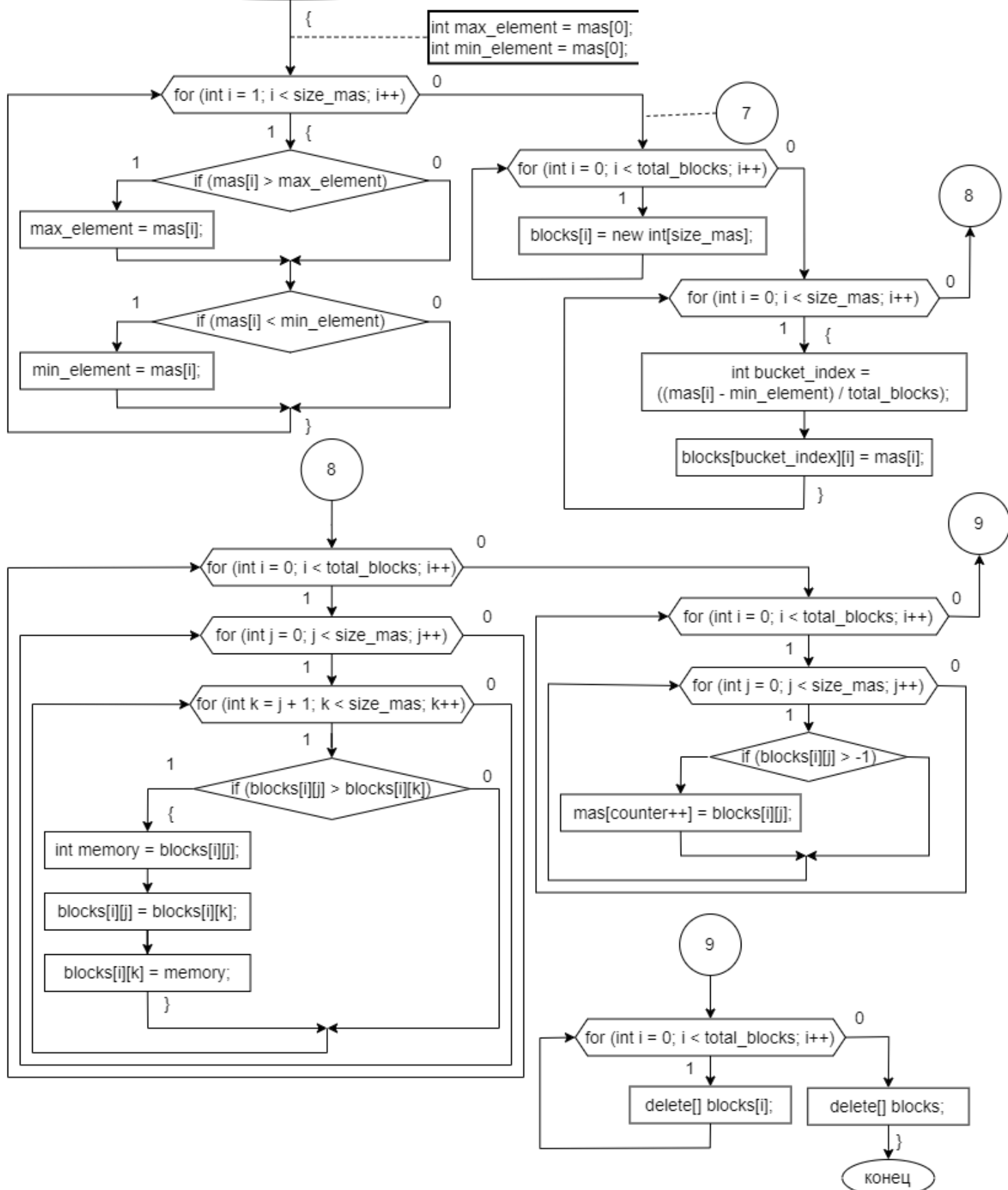
7

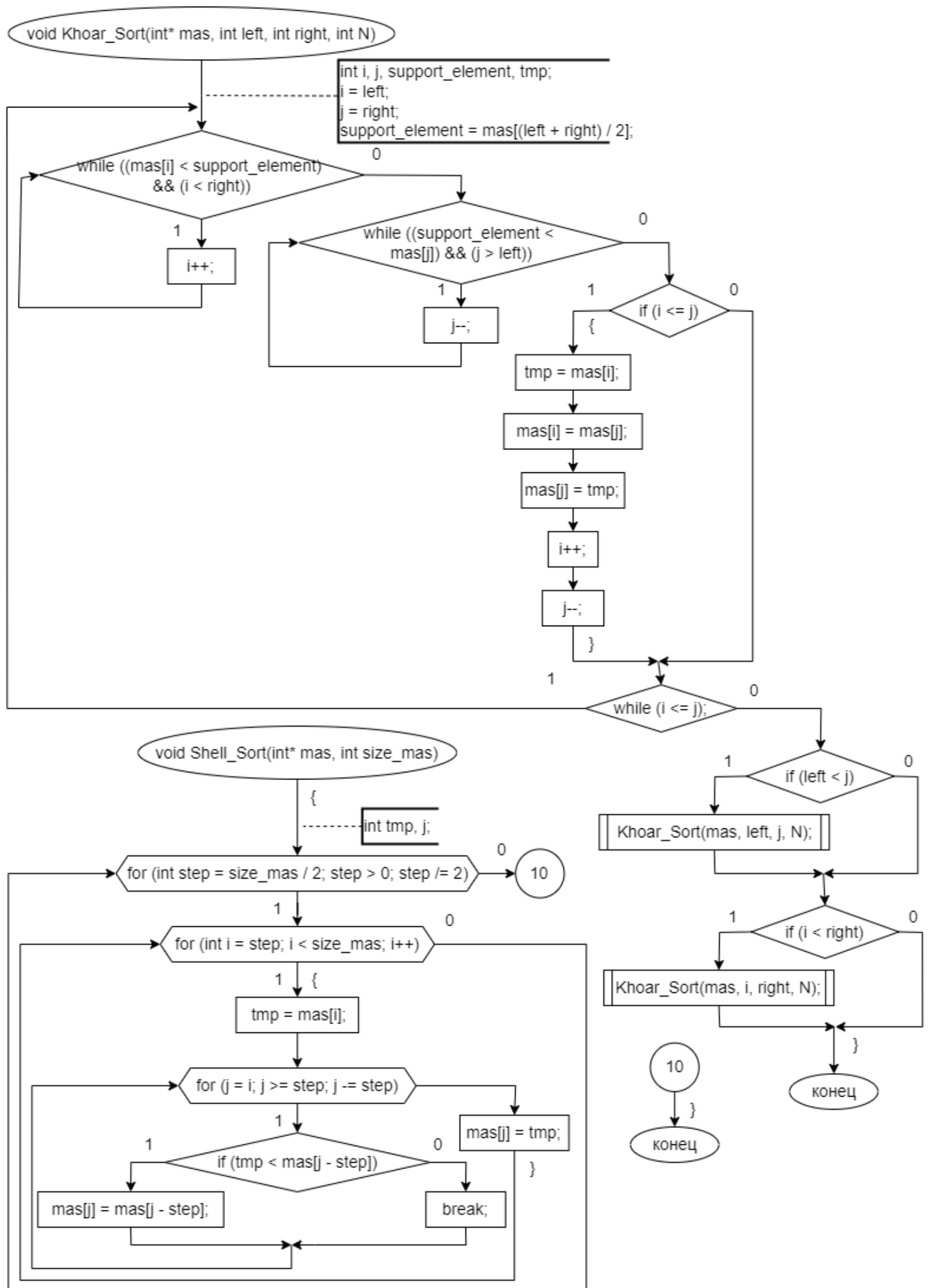
```

int the_numbers_in_the_block = 5;
int total_blocks = ((max_element - min_element) / the_numbers_in_the_block);
int** blocks{ new int* [total_blocks] };
int counter = 0;

```

```
void Block_Sort(int* mas, int size_mas)
```





## Код

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

void creature_mas_rand(int* ptr, int n); //создание массива случайно
void creature_mas_hands(int* ptr, int n); //создание массива с клавиатуры
void print(int* ptr, int n); //вывод массива
void Merge_mas(int* mas, int first_index, int last_index, int all_size); //слияние массивов
void Merge_Sort(int* mas, int first_index, int last_index, int all_size);
//Сортировка слиянием
void Quick_Sort(int* mas, int size); //быстрая Сортировка
void Sort_by_Count(int* A_mas, int mas_size); //сортировка подсчетом
void Block_Sort(int* mas, int size_mas); //блочная сортировка
void Shell_Sort(int* mas, int size_mas); //Сортировка Шелла
void Khoar_Sort(int* mas, int left, int right, int N); //Сортировка Хоара

int main() {
    setlocale(LC_ALL, "Russian");
    srand(time(0));
    int mas_size = 25, choice;
    int* mas = new int[mas_size]; //создание массива

    do {
        cout << "Каким способом вы хотите создать массив? " << endl;
        cout << "1 - Ввести ручками" << endl;
        cout << "2 - Создать случайно" << endl;
        cin >> choice;
    } while (choice < 1 || choice > 2);

    switch (choice) { //выбор способа создания массива
    case 1: {
        creature_mas_hands(mas, mas_size);
        break;
    }
    default: {
        creature_mas_rand(mas, mas_size); //создание массива случайно
        break;
    }
    }

    cout << endl << "Создан массив!" << endl;
    print(mas, mas_size); //вывод массива

    do { //выбор сортировки
        cout << "Каким способом отсортировать массив? " << endl;
        cout << "1 - Сортировка слиянием" << endl;
        cout << "2 - Быстрая сортировка" << endl;
        cout << "3 - Сортировка подсчетом" << endl;
        cout << "4 - Блочная сортировка" << endl;
        cout << "5 - Сортировка Шелла" << endl;
        cout << "6 - Сортировка Хоара" << endl;
        cin >> choice;
    } while (choice < 1 || choice > 6);

    auto start_time_recursive = chrono::steady_clock::now(); // начальное время
    switch (choice) { //выбор сортировки
    case 1: {
        Merge_Sort(mas, 0, mas_size - 1, mas_size); //сортировка слиянием
        break;
    }
    case 2: {
        Quick_Sort(mas, mas_size); //быстрая сортировка
        break;
    }
    }
```

```

    case 3: {
        Sort_by_Count(mas, mas_size); //сортировка подсчетом
        break;
    }
    case 4: {
        Block_Sort(mas, mas_size); //блочная сортировка
        break;
    }
    case 5: {
        Shell_Sort(mas, mas_size); //сортировка шелла
        break;
    }
    default: {
        Khoar_Sort(mas, 0, mas_size - 1, mas_size); //сортировка хоара
        break;
    }
}

auto end_time_recursive = chrono::steady_clock::now(); // конечное время
auto diff_time = end_time_recursive - start_time_recursive; //разница между
начальным и конечным временем
print(mas, mas_size); //вывод массива

cout << "Сортировка выполнена за " << chrono::duration <double,
milli>(diff_time).count() << " ms" << endl << endl;

return 0;
}

void creature_mas_rand(int* ptr, int n) { //создаю массив через датчик случайных
чисел
    for (int i = 0; i < n; i++) {
        ptr[i] = rand() % 1000 + 1; //случайное число от 1 до 1000
    }
}

void creature_mas_hands(int* ptr, int n) { //создание массива с клавиатуры
    for (int i = 0; i < n; i++) {
        cout << "Введите элемент " << i + 1 << ": ";
        cin >> ptr[i];
    }
}

void print(int* ptr, int n) { //вывод текущего массива
    cout << endl << "Текущий массив:" << endl;
    for (int i = 0; i < n; i++) { //прохожу по массиву
        cout << ptr[i] << ' ';
    }
    cout << endl << endl;
}

void Merge_mas(int* mas, int first_index, int last_index, int all_size) {
//слияние массивов
    int arithmetic_mean, start, final;
    int* new_mas = new int[all_size];

    arithmetic_mean = (first_index + last_index) / 2; //вычисление среднего
арифметического
    start = first_index; //начало левой части

    final = arithmetic_mean + 1; //начало правой части
    for (int i = first_index; i <= last_index; i++) { //иду от начала до конца
        if ((start <= arithmetic_mean) && ((final > last_index) || (mas[start] <
mas[final]))) {
            new_mas[i] = mas[start]; //запись начала
            start++; //меняю начало
        }
        else {
            new_mas[i] = mas[final]; //запись конца
            final++; //меняю конец
        }
    }
}

```

```

    }
    for (int i = first_index; i <= last_index; i++) { //запись в основной массив
        mas[i] = new_mas[i]; //запись
    }
    delete[] new_mas; //освобождаю память
}

void Merge_Sort(int* mas, int first_index, int last_index, int all_size) {
    //рекурсивная процедура сортировки
    if (first_index < last_index) {
        Merge_Sort(mas, first_index, (first_index + last_index) / 2, all_size);
        //сортировка левой части
        Merge_Sort(mas, (first_index + last_index) / 2 + 1, last_index, all_size);
        //сортировка правой части
        Merge_mas(mas, first_index, last_index, all_size); //слияние двух частей
    }
}

void Quick_Sort(int* mas, int size) { //быстрая Сортировка
    int first_index = 0; //указатель в начало массива
    int last_index = size - 1; //указатель в конец массива
    int middle = mas[size / 2]; //центральный элемент массива
    do { //деление массива
        while (mas[first_index] < middle) { //в левой части оставляем элементы
            //которые меньше центрального
            first_index++;
        }
        while (mas[last_index] > middle) { //в правой части оставляем элементы
            //которые больше центрального
            last_index--;
        }
        //меняю элементы местами
        if (first_index <= last_index) { //если индекс первого элемента меньше
            //индекса последнего
            int tmp = mas[first_index];
            mas[first_index] = mas[last_index];
            mas[last_index] = tmp;
            first_index++; //увеличиваю индекс начала
            last_index--; //уменьшаю индекс конца
        }
    } while (first_index <= last_index);

    //если осталось что сортировать - рекурсия
    if (last_index > 0) {
        Quick_Sort(mas, last_index + 1); //левая часть
    }
    if (first_index < size) {
        Quick_Sort(&mas[first_index], size - first_index); //правая часть
    }
}

void Sort_by_Count(int* A_mas, int mas_size) { //сортировка подсчетом
    int max_index = 0, min_index = 0;
    for (int i = 0; i < mas_size; i++) { //нахожу максимальный и минимальный
        //элементы
        if (A_mas[i] > max_index)
            max_index = A_mas[i]; //индекс максимального элемента
        if (A_mas[i] < min_index)
            min_index = A_mas[i]; //индекс минимального элемента
    }
    int number_of_different_values = max_index - min_index + 1; //максимально
    //возможное количество разных значений в массиве
    int* B_mas = new int[number_of_different_values]; //массив для подсчета
    //количества разных значений в основном массиве
    for (int i = 0; i < number_of_different_values; i++) { //заполняю 0 новый
        //массив
        B_mas[i] = 0;
    }
}

```

```

    }
    for (int i = 0; i < mas_size; i++) { //подсчет количества каждого значения
        B_mas[A_mas[i] - min_index]++;
    }
    int counter = 0; //счетчик индекса для основного массива
    for (int i = 0; i < number_of_different_values; i++) { //проход по массиву
значений
        for (int j = 0; j < B_mas[i]; j++) { //запись значения необходимое число
раз
            A_mas[counter++] = i + min_index; //запись значения в основной массив
        }
    }
    delete[] B_mas; //очищаю память
}

void Block_Sort(int* mas, int size_mas) { //блочная сортировка

    int max_element = mas[0]; //максимальный элемент
    int min_element = mas[0]; //минимальный элемент

    for (int i = 1; i < size_mas; i++) {
        if (mas[i] > max_element) { //нахожу максимальный элемент
            max_element = mas[i];
        }
        if (mas[i] < min_element) { //нахожу минимальный элемент
            min_element = mas[i];
        }
    }

    int the_numbers_in_the_block = 5; //сколько чисел максимум войдет в один блок
    int total_blocks = ((max_element - min_element) / the_numbers_in_the_block);
//количество блоков
    int** blocks{ new int* [total_blocks] }; //двумерный массив блоков

    for (int i = 0; i < total_blocks; i++) { //инициализация массива
        blocks[i] = new int[size_mas];
    }

    for (int i = 0; i < size_mas; i++) { //распределение чисел по блокам
        int bucket_index = ((mas[i] - min_element) / total_blocks); //в какой блок
определить элемент
        blocks[bucket_index][i] = mas[i]; //записываю число в блок
    }
    //сортировка каждого блока методом вставки
    for (int i = 0; i < total_blocks; i++) { //проход по блокам
        for (int j = 0; j < size_mas; j++) { //первое число для сравнения
            for (int k = j + 1; k < size_mas; k++) { //второе число для сравнения
элементы
                if (blocks[i][j] > blocks[i][k]) { //если необходимо поменять
запоминания
                    int memory = blocks[i][j]; //временная переменная для
                    blocks[i][j] = blocks[i][k];
                    blocks[i][k] = memory;
                }
            }
        }
    }

    int counter = 0; //счетчик для основного массива
    for (int i = 0; i < total_blocks; i++) { //прохожу по блокам
        for (int j = 0; j < size_mas; j++) { //прохожу по элементам в блоке
            if (blocks[i][j] > -1) { //если элемент - НЕ мусор
                mas[counter++] = blocks[i][j]; //запись элемента
            }
        }
    }
}

```



```

    for (int i = 0; i < total_blocks; i++) { //освобождение памяти
        delete[] blocks[i]; //удаление элемента в блоке
    }
    delete[] blocks; //удаление массива
}

void Shell_Sort(int* mas, int size_mas) { //Сортировка Шелла
    int tmp, j;
    for (int step = size_mas / 2; step > 0; step /= 2) { //пока длина шага больше
0. после каждой итерации шаг уменьшается
        for (int i = step; i < size_mas; i++) { //прохожу по массиву с шагом step
            tmp = mas[i]; //запоминаю значение текущего элемента
            for (j = i; j >= step; j -= step) { //прохожу с шагом step
                if (tmp < mas[j - step]) { //Если tmp меньше, мы меняем местами
tmp и элемент массива на j - step
                    mas[j] = mas[j - step];
                }
                else { //Если tmp больше или равен элементу массива на j - step,
мы прерываем цикл
                    break;
                }
            }
            mas[j] = tmp; //меняю значение текущего элемента
        }
    }
}

void Khoar_Sort(int* mas, int left, int right, int N) { //Сортировка Хоара
    int i, j, support_element, tmp; // tmp - временная переменная для обмена
    i = left; // левая граница
    j = right; // правая граница
    support_element = mas[(left + right) / 2]; //опорный элемент для сортировки
    do {
        while ((mas[i] < support_element) && (i < right)) i++; //пока элемент
массива на позиции i не станет больше или равен опорному и i не станет больше
правой границы
        while ((support_element < mas[j]) && (j > left)) j--; //пока элемент
массива на позиции j не станет меньше или равен опорному и j не станет меньше
левой границы
        if (i <= j) { //меняем местами элементы массива на позициях i и j с
помощью переменной tmp
            tmp = mas[i];
            mas[i] = mas[j];
            mas[j] = tmp;
            i++; //увеличиваю левую границу
            j--; //уменьшаю правую границу
        }
    } while (i <= j);
    //проверяем, не выходят ли границы i и j за пределы массива
    if (left < j) Khoar_Sort(mas, left, j, N);
    if (i < right) Khoar_Sort(mas, i, right, N);
}

```

## Результат работы

Пусть дан такой массив: 156 492 707 267 184 305 418 537 1 818 845 805  
1 136 209 996 231 231 58 879 435 180 839 975 231. Отсортирую этот массив,  
используя разные сортировки.

### 1) Сортировка слиянием.

```
Каким способом вы хотите создать массив?
1 - Ввести вручную
2 - Создать случайно
1

Введите элемент 1: 156
Введите элемент 2: 492
Введите элемент 3: 707
Введите элемент 4: 267
Введите элемент 5: 184
Введите элемент 6: 305
Введите элемент 7: 418
Введите элемент 8: 537
Введите элемент 9: 1
Введите элемент 10: 818
Введите элемент 11: 845
Введите элемент 12: 805
Введите элемент 13: 1
Введите элемент 14: 136
Введите элемент 15: 209
Введите элемент 16: 996
Введите элемент 17: 231
Введите элемент 18: 231
Введите элемент 19: 58
Введите элемент 20: 879
Введите элемент 21: 435
Введите элемент 22: 180
Введите элемент 23: 839
Введите элемент 24: 975
Введите элемент 25: 231

Создан массив!

Текущий массив:
156 492 707 267 184 305 418 537 1 818 845 805 1 136 209 996 231 231 58 879 435 180 839 975 231

Каким способом отсортировать массив?
1 - Сортировка слиянием
2 - Быстрая сортировка
3 - Сортировка подсчетом
4 - Блочная сортировка
5 - Сортировка Шелла
6 - Сортировка Хоара
1

Текущий массив:
1 1 58 136 156 180 184 209 231 231 231 267 305 418 435 492 537 707 805 818 839 845 879 975 996

Сортировка выполнена за 0.0135 ms
```

## 2) Быстрая сортировка.

```
Каким способом вы хотите создать массив?
1 - Ввести вручную
2 - Создать случайно
1

Введите элемент 1: 156
Введите элемент 2: 492
Введите элемент 3: 707
Введите элемент 4: 267
Введите элемент 5: 184
Введите элемент 6: 305
Введите элемент 7: 418
Введите элемент 8: 537
Введите элемент 9: 1
Введите элемент 10: 818
Введите элемент 11: 845
Введите элемент 12: 805
Введите элемент 13: 1
Введите элемент 14: 136
Введите элемент 15: 209
Введите элемент 16: 996
Введите элемент 17: 231
Введите элемент 18: 231
Введите элемент 19: 58
Введите элемент 20: 879
Введите элемент 21: 435
Введите элемент 22: 180
Введите элемент 23: 839
Введите элемент 24: 975
Введите элемент 25: 231

Создан массив!

Текущий массив:
156 492 707 267 184 305 418 537 1 818 845 805 1 136 209 996 231 231 58 879 435 180 839 975 231

Каким способом отсортировать массив?
1 - Сортировка слиянием
2 - Быстрая сортировка
3 - Сортировка подсчетом
4 - Блочная сортировка
5 - Сортировка Шелла
6 - Сортировка Хоара
2

Текущий массив:
1 1 58 136 156 180 184 209 231 231 231 267 305 418 435 492 537 707 805 818 839 845 879 975 996

Сортировка выполнена за 0.0022 ms
```

### 3) Сортировка подсчетом.

```
Каким способом вы хотите создать массив?
1 - Ввести ручками
2 - Создать случайно
1

Введите элемент 1: 156
Введите элемент 2: 492
Введите элемент 3: 707
Введите элемент 4: 267
Введите элемент 5: 184
Введите элемент 6: 305
Введите элемент 7: 418
Введите элемент 8: 537
Введите элемент 9: 1
Введите элемент 10: 818
Введите элемент 11: 845
Введите элемент 12: 805
Введите элемент 13: 1
Введите элемент 14: 136
Введите элемент 15: 209
Введите элемент 16: 996
Введите элемент 17: 231
Введите элемент 18: 231
Введите элемент 19: 58
Введите элемент 20: 879
Введите элемент 21: 435
Введите элемент 22: 180
Введите элемент 23: 839
Введите элемент 24: 975
Введите элемент 25: 231

Создан массив!

Текущий массив:
156 492 707 267 184 305 418 537 1 818 845 805 1 136 209 996 231 231 58 879 435 180 839 975 231

Каким способом отсортировать массив?
1 - Сортировка слиянием
2 - Быстрая сортировка
3 - Сортировка подсчетом
4 - Блочная сортировка
5 - Сортировка Шелла
6 - Сортировка Хоара
3

Текущий массив:
1 1 58 136 156 180 184 209 231 231 231 267 305 418 435 492 537 707 805 818 839 845 879 975 996

Сортировка выполнена за 0.0171 ms
```

4) Блочная сортировка.

```
Каким способом вы хотите создать массив?
1 - Ввести вручную
2 - Создать случайно
1

Введите элемент 1: 156
Введите элемент 2: 492
Введите элемент 3: 707
Введите элемент 4: 267
Введите элемент 5: 184
Введите элемент 6: 305
Введите элемент 7: 418
Введите элемент 8: 537
Введите элемент 9: 1
Введите элемент 10: 818
Введите элемент 11: 845
Введите элемент 12: 805
Введите элемент 13: 1
Введите элемент 14: 136
Введите элемент 15: 209
Введите элемент 16: 996
Введите элемент 17: 231
Введите элемент 18: 231
Введите элемент 19: 58
Введите элемент 20: 879
Введите элемент 21: 435
Введите элемент 22: 180
Введите элемент 23: 839
Введите элемент 24: 975
Введите элемент 25: 231

Создан массив!

Текущий массив:
156 492 707 267 184 305 418 537 1 818 845 805 1 136 209 996 231 231 58 879 435 180 839 975 231

Каким способом отсортировать массив?
1 - Сортировка слиянием
2 - Быстрая сортировка
3 - Сортировка подсчетом
4 - Блочная сортировка
5 - Сортировка Шелла
6 - Сортировка Хоара
4

Текущий массив:
1 1 58 136 156 180 184 209 231 231 231 267 305 418 435 492 537 707 805 818 839 845 879 975 996

Сортировка выполнена за 0.2332 ms
```

## 5) Сортировка Шелла.

```
Каким способом вы хотите создать массив?
1 - Ввести вручную
2 - Создать случайно
1

Введите элемент 1: 156
Введите элемент 2: 492
Введите элемент 3: 707
Введите элемент 4: 267
Введите элемент 5: 184
Введите элемент 6: 305
Введите элемент 7: 418
Введите элемент 8: 537
Введите элемент 9: 1
Введите элемент 10: 818
Введите элемент 11: 845
Введите элемент 12: 805
Введите элемент 13: 1
Введите элемент 14: 136
Введите элемент 15: 209
Введите элемент 16: 996
Введите элемент 17: 231
Введите элемент 18: 231
Введите элемент 19: 58
Введите элемент 20: 879
Введите элемент 21: 435
Введите элемент 22: 180
Введите элемент 23: 839
Введите элемент 24: 975
Введите элемент 25: 231

Создан массив!

Текущий массив:
156 492 707 267 184 305 418 537 1 818 845 805 1 136 209 996 231 231 58 879 435 180 839 975 231

Каким способом отсортировать массив?
1 - Сортировка слиянием
2 - Быстрая сортировка
3 - Сортировка подсчетом
4 - Блочная сортировка
5 - Сортировка Шелла
6 - Сортировка Хоара
5

Текущий массив:
1 1 58 136 156 180 184 209 231 231 231 267 305 418 435 492 537 707 805 818 839 845 879 975 996

Сортировка выполнена за 0.0013 ms
```

## 6) Сортировка Хоара.

```

Каким способом вы хотите создать массив?
1 - Ввести вручную
2 - Создать случайно
1

Введите элемент 1: 156
Введите элемент 2: 492
Введите элемент 3: 707
Введите элемент 4: 267
Введите элемент 5: 184
Введите элемент 6: 305
Введите элемент 7: 418
Введите элемент 8: 537
Введите элемент 9: 1
Введите элемент 10: 818
Введите элемент 11: 845
Введите элемент 12: 805
Введите элемент 13: 1
Введите элемент 14: 136
Введите элемент 15: 209
Введите элемент 16: 996
Введите элемент 17: 231
Введите элемент 18: 231
Введите элемент 19: 58
Введите элемент 20: 879
Введите элемент 21: 435
Введите элемент 22: 180
Введите элемент 23: 839
Введите элемент 24: 975
Введите элемент 25: 231

Создан массив!

Текущий массив:
156 492 707 267 184 305 418 537 1 818 845 805 1 136 209 996 231 231 58 879 435 180 839 975 231

Каким способом отсортировать массив?
1 - Сортировка слиянием
2 - Быстрая сортировка
3 - Сортировка подсчетом
4 - Блочная сортировка
5 - Сортировка Шелла
6 - Сортировка Хоара
6

Текущий массив:
1 1 58 136 156 180 184 209 231 231 231 267 305 418 435 492 537 707 805 818 839 845 879 975 996

Сортировка выполнена за 0.002 ms

```

Создам массив случайно

```
Каким способом вы хотите создать массив?
1 - Ввести ручками
2 - Создать случайно
2

Создан массив!

Текущий массив:
298 169 547 387 117 707 72 705 855 623 640 920 973 698 804 91 633 841 336 459 780 176 609 817 257

Каким способом отсортировать массив?
1 - Сортировка слиянием
2 - Быстрая сортировка
3 - Сортировка подсчетом
4 - Блочная сортировка
5 - Сортировка Шелла
6 - Сортировка Хоара
2

Текущий массив:
72 91 117 169 176 257 298 336 387 459 547 609 623 633 640 698 705 707 780 804 817 841 855 920 973

Сортировка выполнена за 0.0021 ms
```

## Вывод

В ходе работы я применила знания о работе различных сортировок, в числе которых: сортировки: слиянием, подсчетом, быстрая, блочная, Шелла, Хоара. Мне удалось реализовать поставленную задачу: были созданы все необходимые сортировки.

Сортировка слиянием — это алгоритм сортировки, который разбивает исходный массив на две части, сортирует каждую часть отдельно, а затем объединяет две отсортированные части. Этот алгоритм является одним из самых быстрых и эффективных алгоритмов сортировки. Он имеет сложность  $O(n \log n)$ , где  $n$  - это количество элементов в массиве.

Сортировка подсчетом — это алгоритм сортировки, который использует счетчик для каждого уникального значения в массиве. Этот алгоритм работает быстрее, чем сортировка пузырьком, но медленнее, чем сортировка слиянием. Он имеет сложность  $O(n + k)$ , где  $n$  - это количество элементов в массиве, а  $k$  - это количество уникальных значений.

Быстрая сортировка — это алгоритм сортировки, который разделяет массив на две части, одна из которых содержит все элементы, меньшие или равные некоторому выбранному элементу, а другая - все элементы, большие этого элемента. Этот алгоритм является одним из самых быстрых и



эффективных алгоритмов сортировки. Он имеет сложность  $O(n \log n)$ , где  $n$  - это количество элементов в массиве.

Блочная сортировка — это алгоритм сортировки, который разбивает исходный массив на блоки, сортирует каждый блок отдельно, а затем объединяет отсортированные блоки. Этот алгоритм имеет сложность  $O(n \log n)$ , где  $n$  - это количество элементов в массиве.

Сортировка Шелла — это алгоритм сортировки, который использует принцип "перестановки с шагом". Он работает, перемещая элементы на определенное расстояние друг от друга, пока они не будут отсортированы. Этот алгоритм имеет сложность  $O(n^2)$ , где  $n$  - это количество элементов в массиве.





Сортировка Хоара — это алгоритм сортировки, который использует принцип "перестановки с шагом". Он работает, перемещая элементы на определенное расстояние друг от друга, пока они не будут отсортированы. Этот алгоритм имеет сложность  $O(n^2)$ , где  $n$  - это количество элементов в массиве.


В моей работе был использован массив из 25 элементов, поэтому самыми быстрыми сортировками оказались: быстрая, Шелла и Хоара. Самой медленной сортировкой можно назвать – блочную.


Стоит отметить, что на больших массивах сортировки Шелла и Хоара не будут столь же эффективными, в виду того, что они имеют сложность  $O(n^2)$ .

## **GitHub**

Ссылка: <https://github.com/SonyAkb/simple-sorting.git>

<div>  <span>SonyAkb</span> <span>Обновление сортировки простые блок-схема.drawio.png</span> <span>b6b394e · 2 hours ago</span> <span>17 Commits</span> </div>		
 <span>README.md</span>	<span>Update README.md</span>	<span>yesterday</span>
 <span>Сортировки простые.cpp</span>	<span>Измерение времени сортировок</span>	<span>yesterday</span>
 <span>сортировки простые блок-схема.drawio.png</span>	<span>Обновление сортировки простые блок-схема.drawio.png</span>	<span>2 hours ago</span>


README



Простые сортировки

Цель: знакомство с простыми сортировками, в числе которых сортировки: слиянием, подсчетом, быстрая, блочная, Шелла, Хоара.

Постановка задачи: реализовать сортировку массива из 25 элементов, с помощью сортировок: слиянием, подсчетом, быстрая, блочная Шелла, Хоара. Необходимо зафиксировать быстродействие каждой сортировки.