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

Лабораторная работа

«Простые сортировки»

Выполнил:

студент первого курса

ЭТФ группы РИС-23-3б

Акбашева Софья Руслановна

Проверила:

Доцент кафедры ИТАС О. А. Полякова

Пермь, 2024

**Простые сортировки**

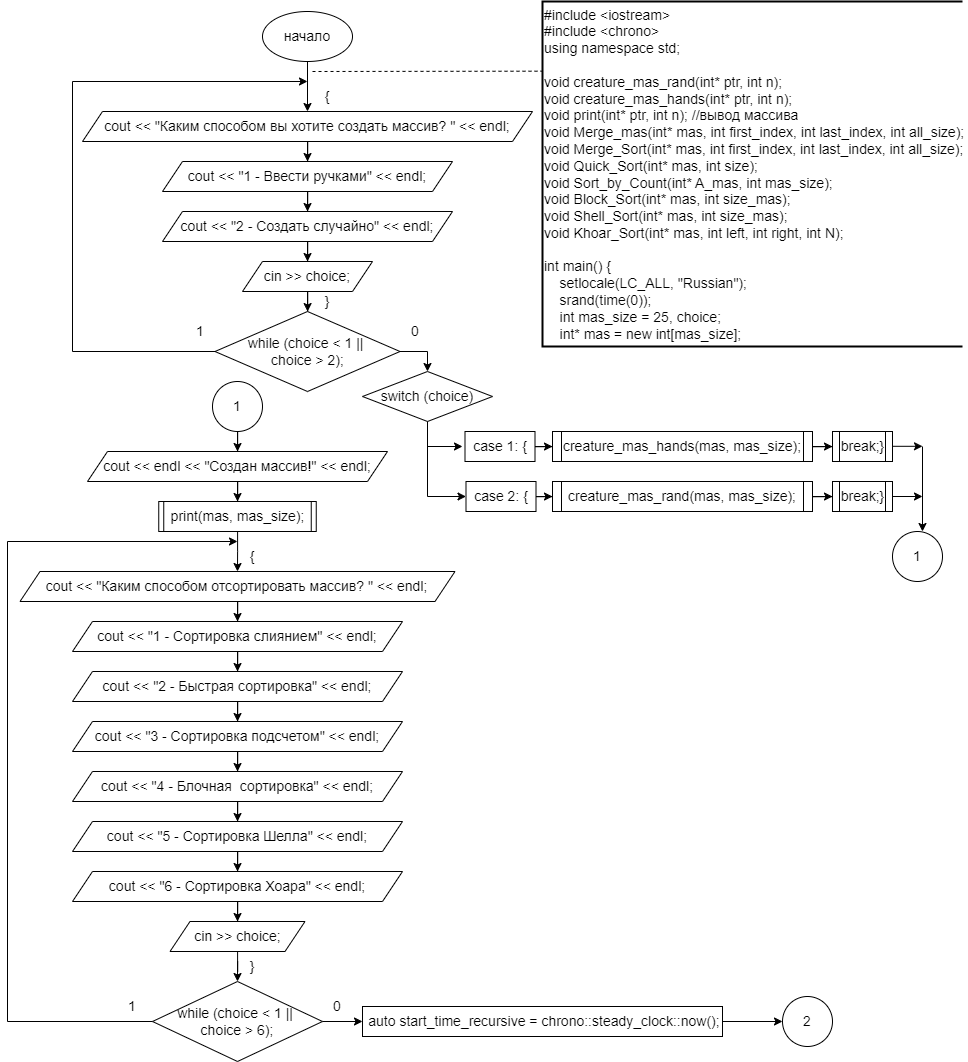
**Цель**: знакомство с простыми сортировками, в числе которых сортировки: слиянием, подсчетом, быстрая, блочная, Шелла, Хоара.

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

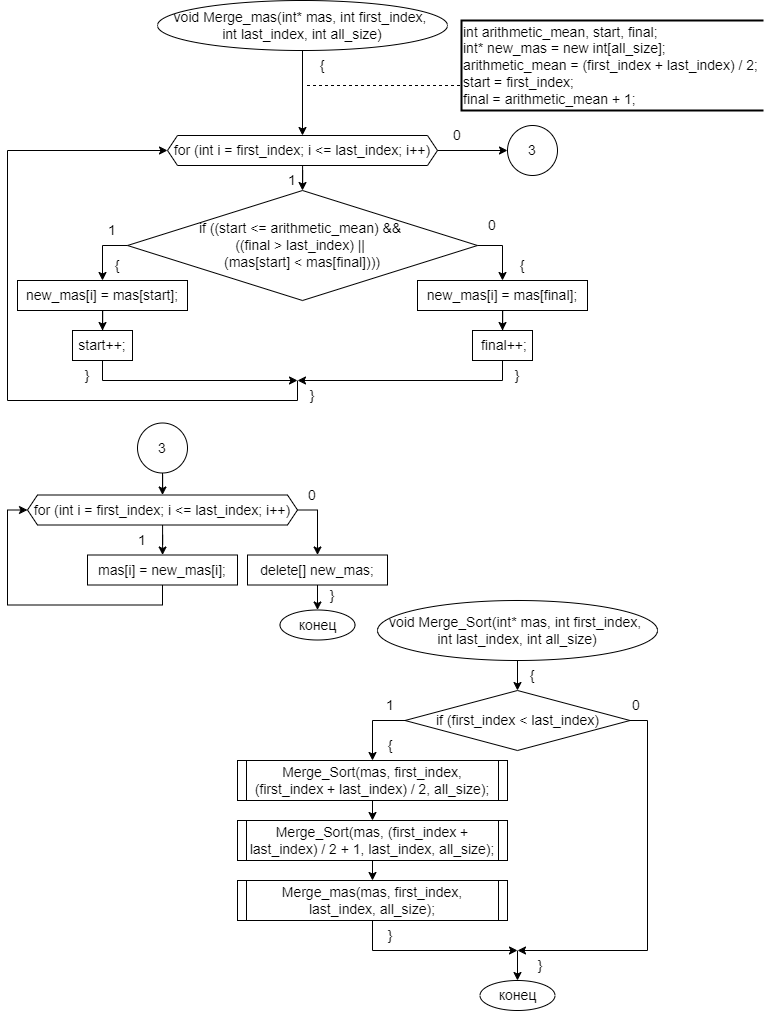
# Анализ задачи

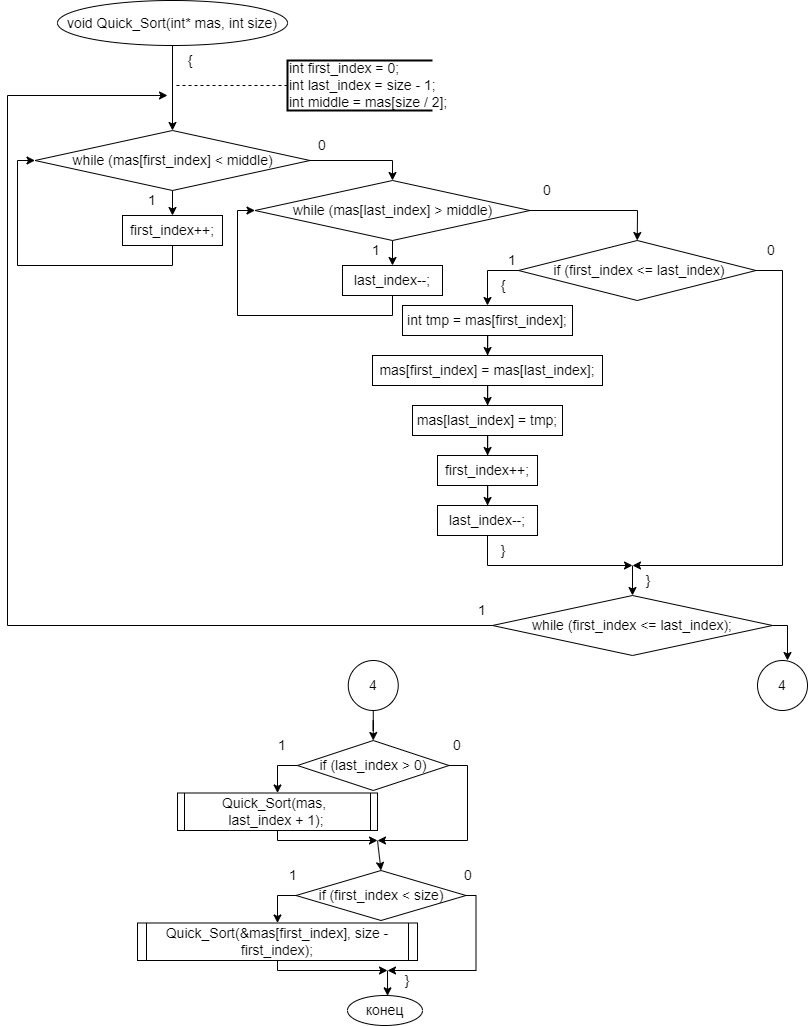
1. Программа запрашивает у пользователя, каким способом он хочет создать массив: ввести числа вручную или создать массив случайно. Если необходимо создать массив случайно, то используется датчик случайных чисел, иначе пользователь вводит каждый элемент с клавиатуры с помощью арифметического цикла.
2.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

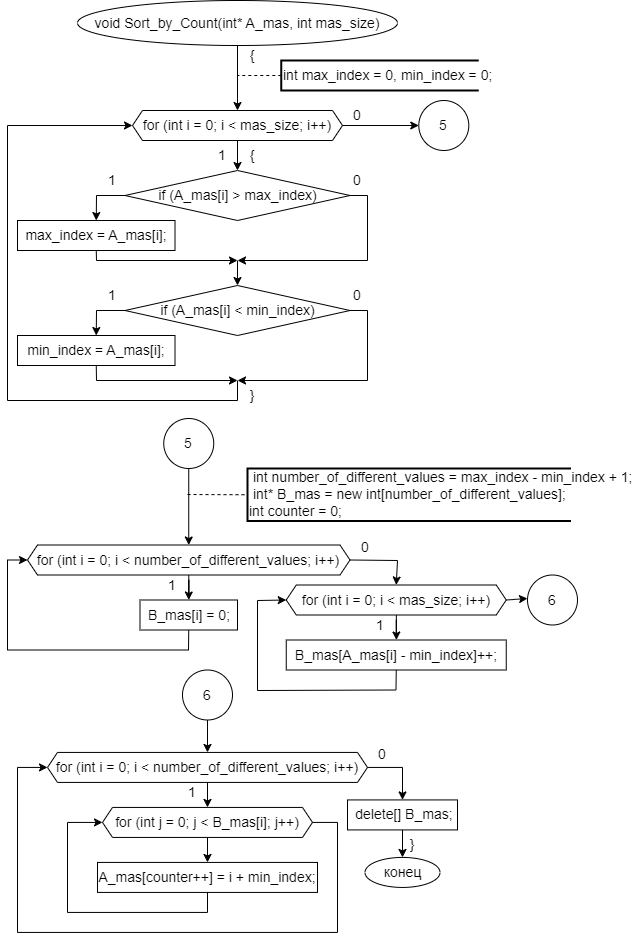
# Блок схема

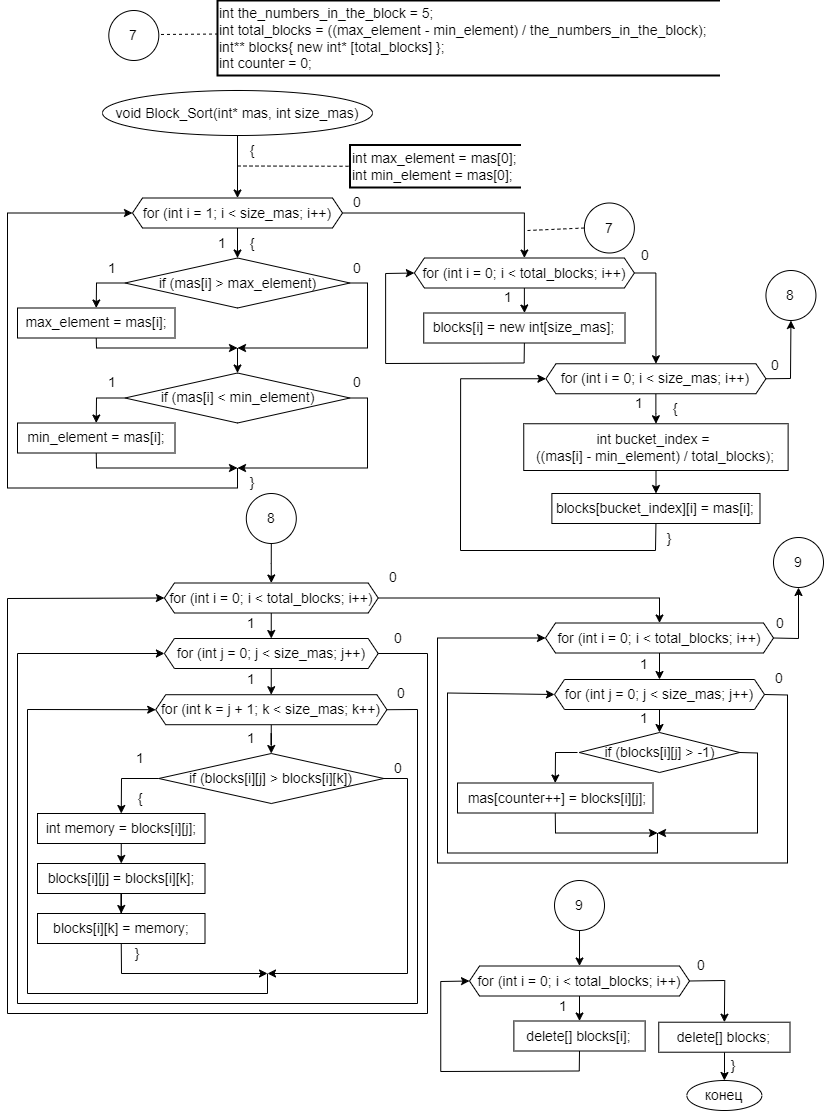


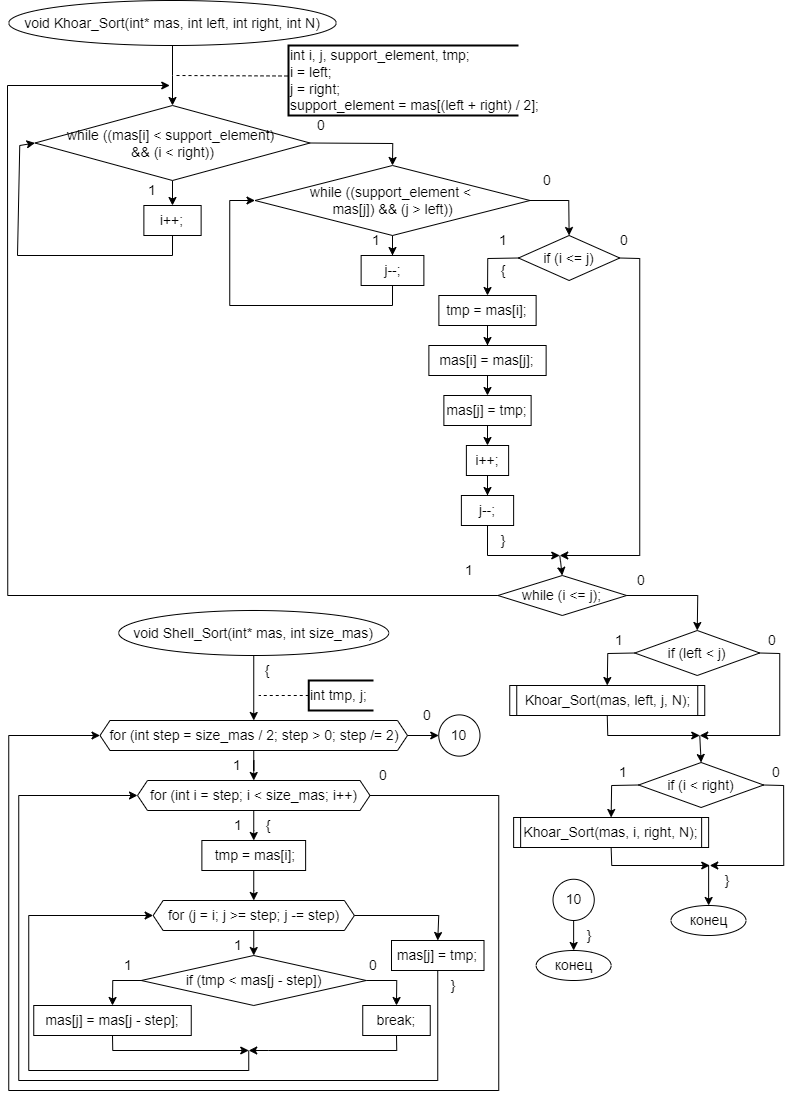












# Код

#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); //Сортровка 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 количество 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 чесел 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 всавки

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

if (blocks[i][j] > blocks[i][k]) { //если необходимо поменять элементы

int memory = blocks[i][j]; //временная переменная для запоминания

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) { //Сортровка 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);

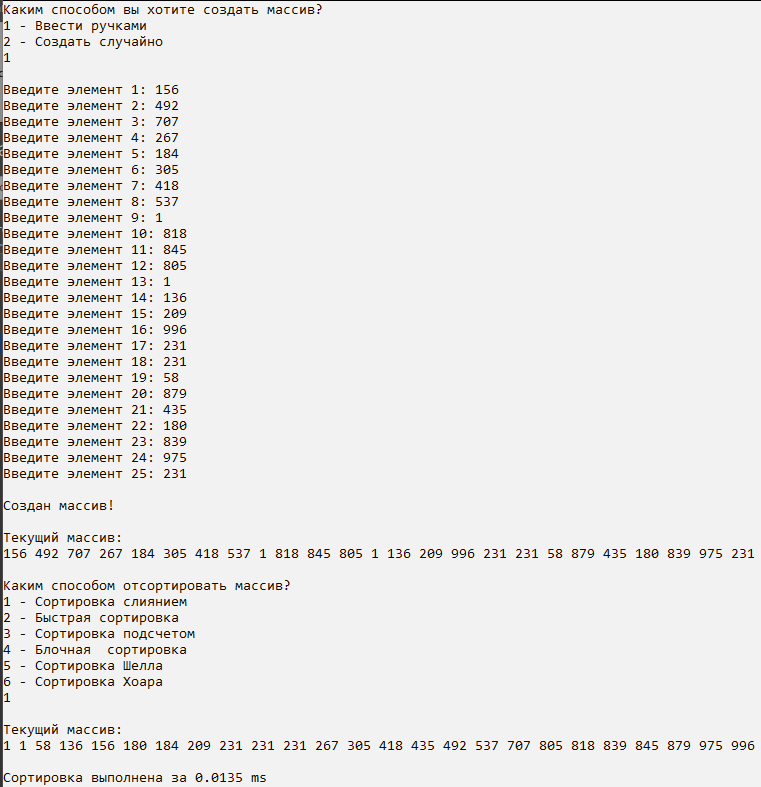
if (i < right) Khoar\_Sort(mas, i, right, N);

}

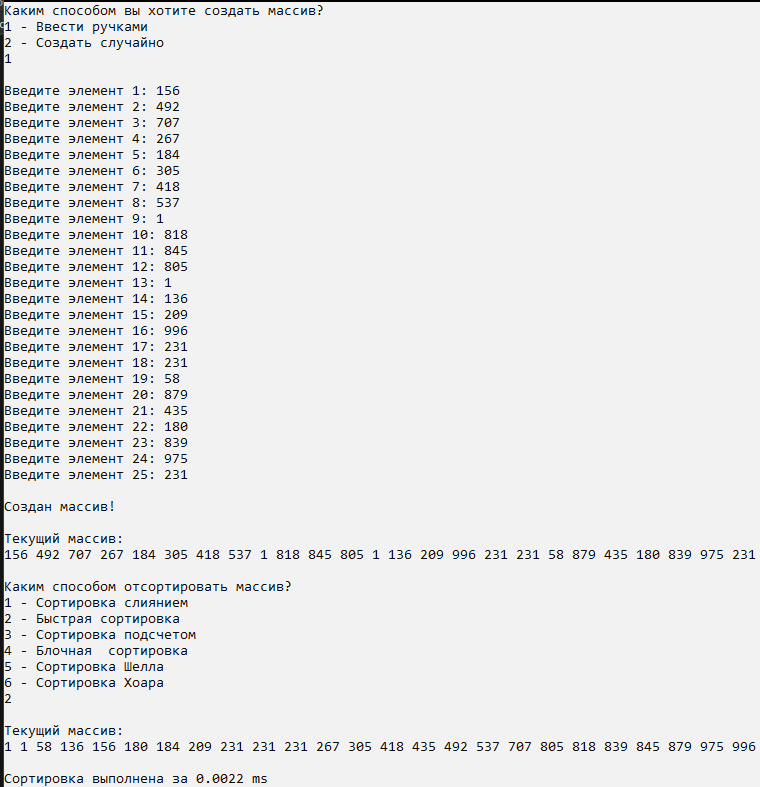
# Результат работы

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, использую разные сортировки.

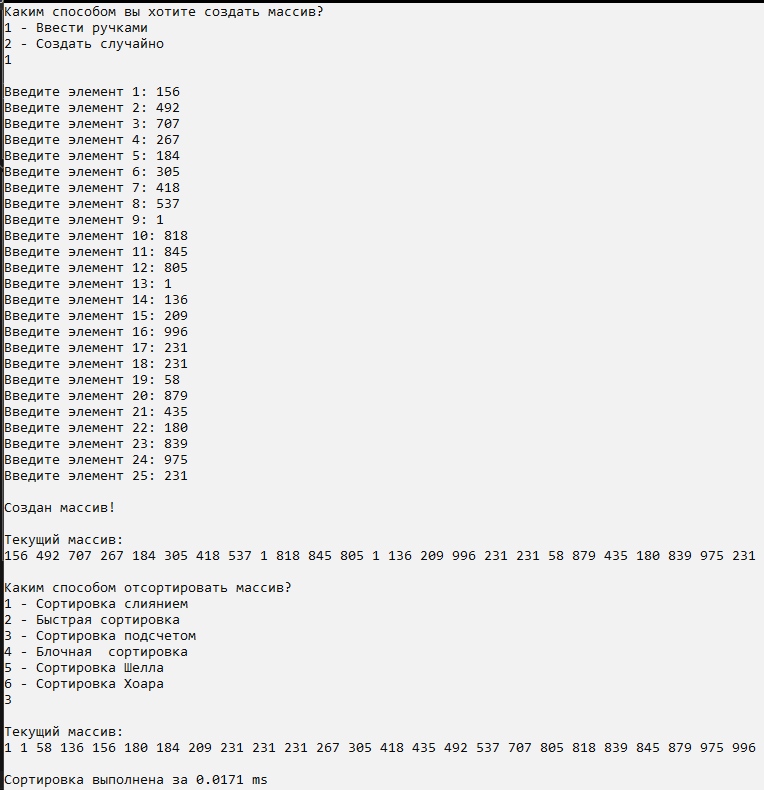
## Сортировка слиянием.



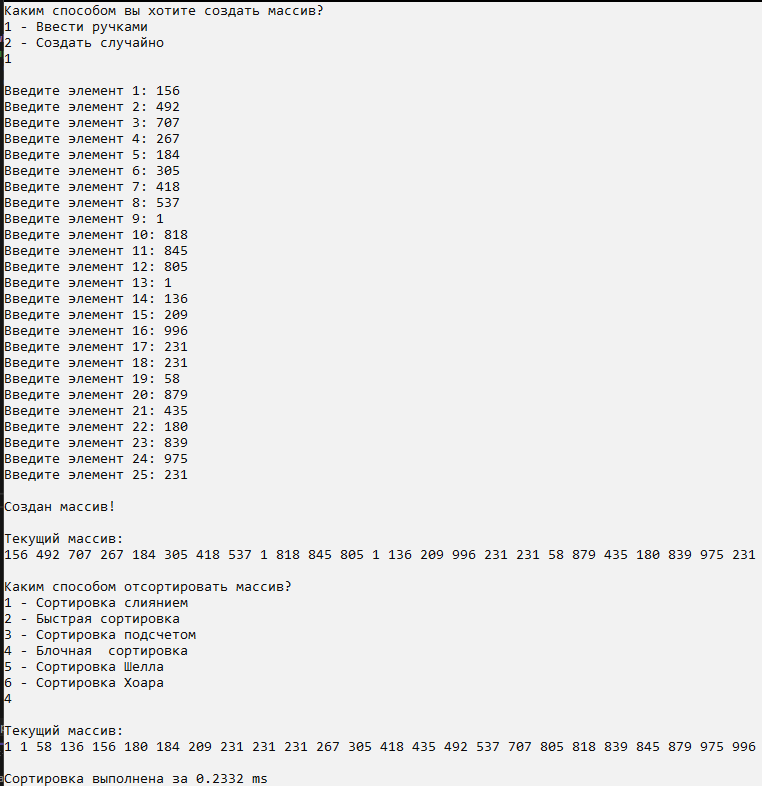
## Быстрая сортировка.



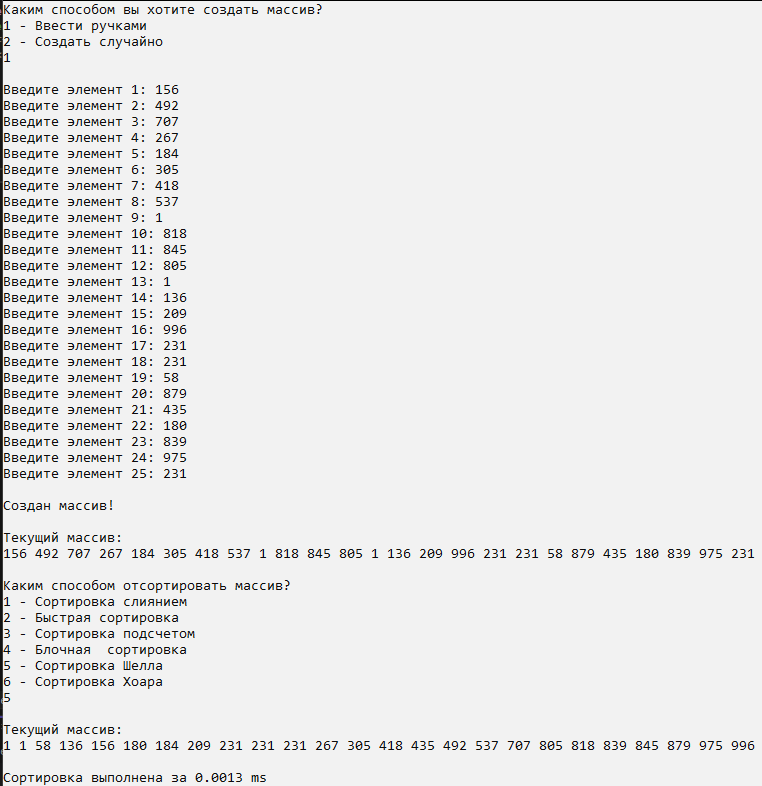
## Сортировка подсчетом.



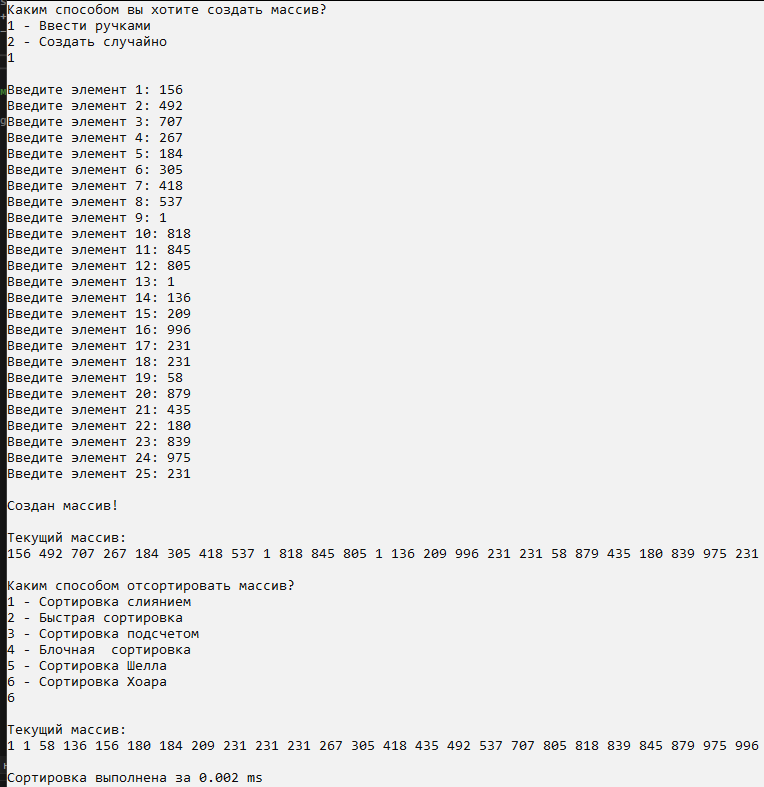
## Блочная сортировка.



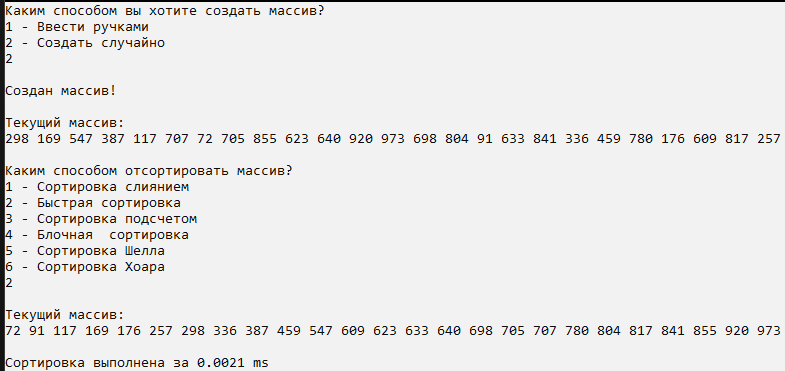
## Сортировка Шелла.



## Сортировка Хоара.



Создам массив случайно



# 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>

