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

Лабораторная работа
«Сложные сортировки»

Выполнил:
студент первого курса
ЭТФ группы РИС-23-3б
Акбашева Софья Руслановна

Проверила:
Доцент кафедры ИТАС О. А. Полякова

Пермь, 2024

Сложные сортировки

Цель: знакомство со сложными сортировками, в числе которых: сортировка естественным слиянием и многофазная сортировка.

Постановка задачи: реализовать сортировку массива из 25 элементов, с помощью сортировок: естественным слиянием и многофазной.

Сортировка естественным слиянием

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) Функция Merge_Sort_Recursion является рекурсивной реализацией алгоритма сортировки слиянием. Она разделяет массив на две части и сортирует каждую часть отдельно, а затем объединяет отсортированные части.

4) Функция Merger_Sort вызывает Merge_Sort_Recursion для сортировки массива и затем рекурсивно вызывает себя для сортировки оставшейся части массива.

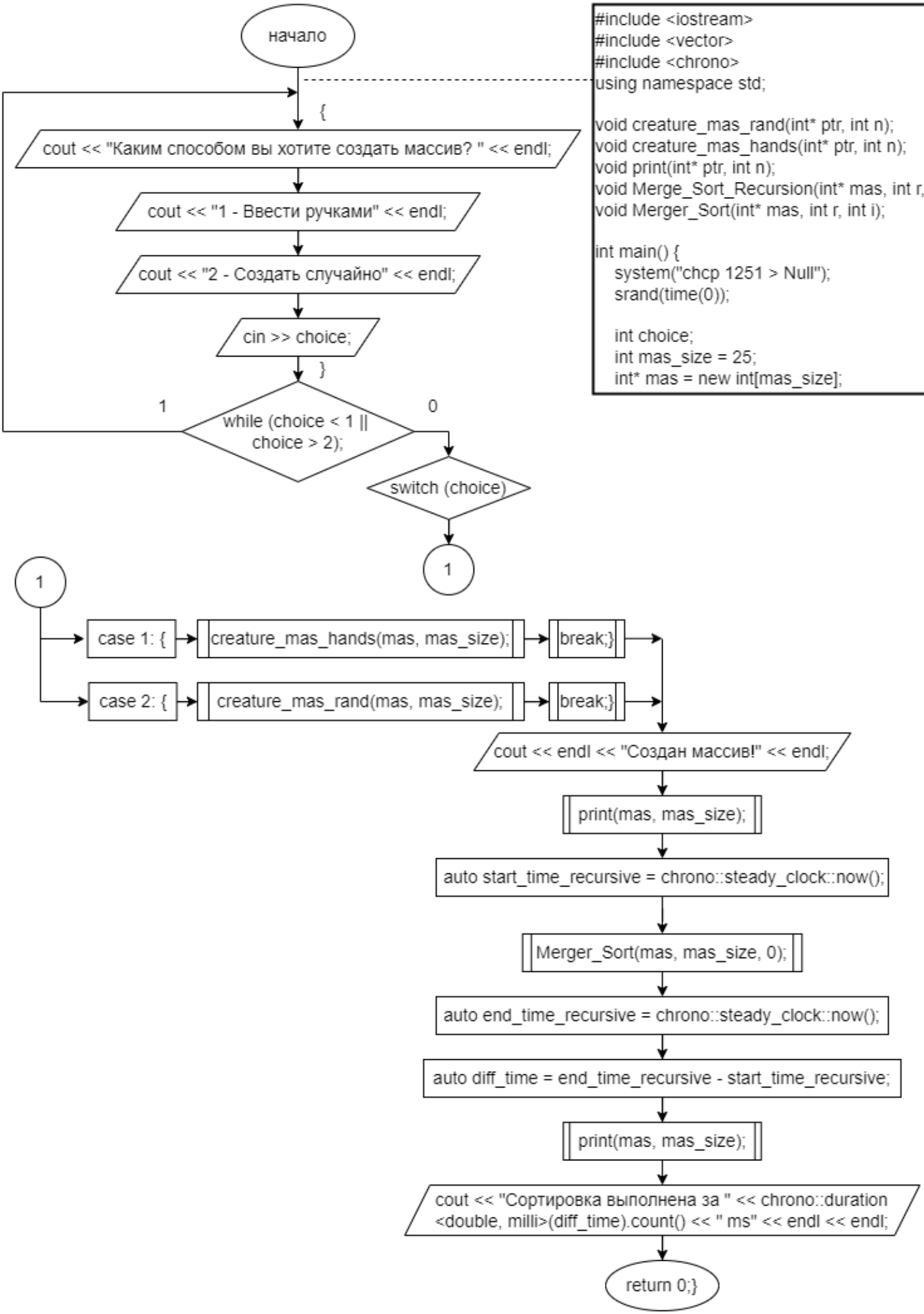
5) Время выполнения сортировки измеряется и выводится на экран.

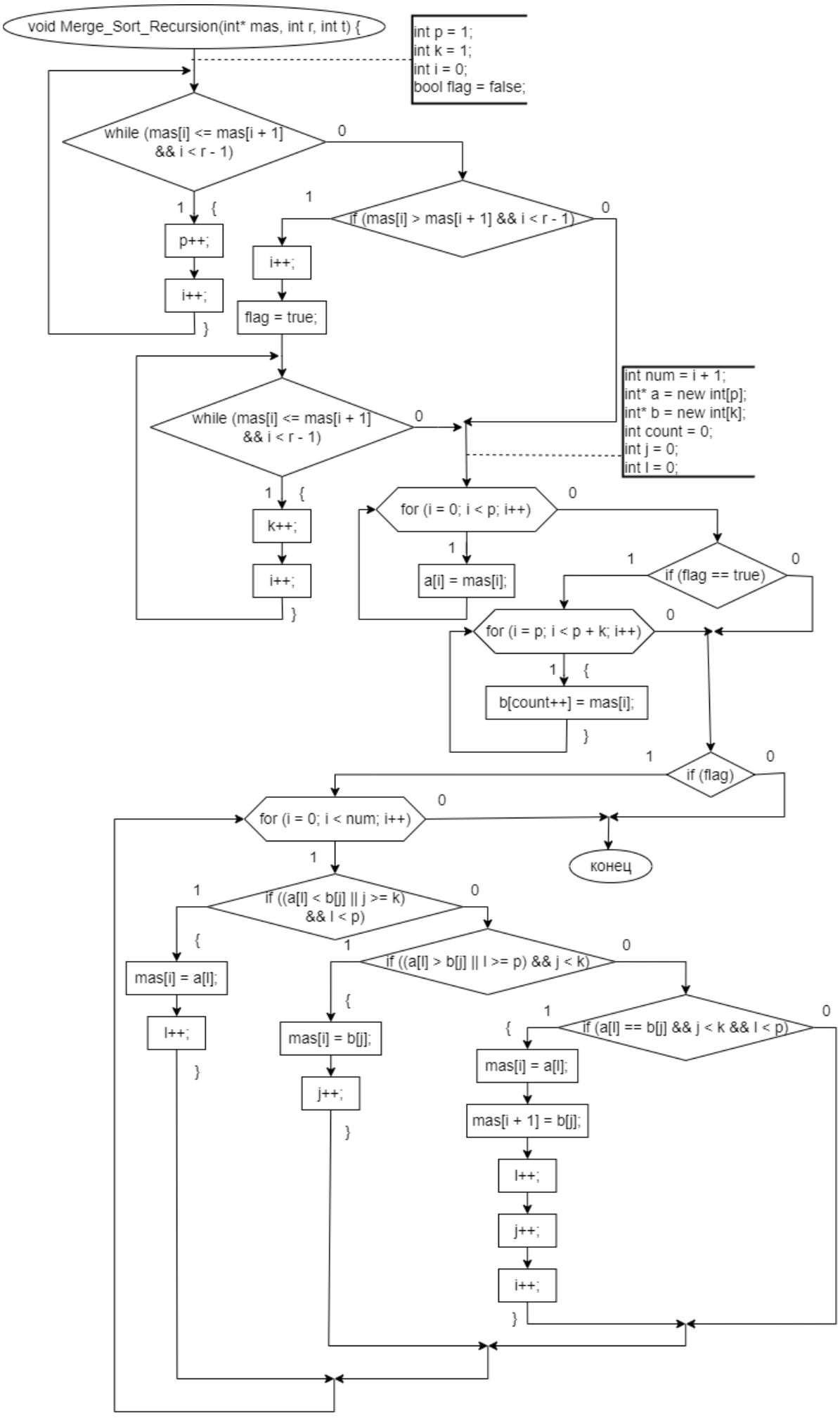
6) В коде используется функция srand для инициализации генератора случайных чисел с текущим временем, что обеспечивает уникальность каждого запуска программы.

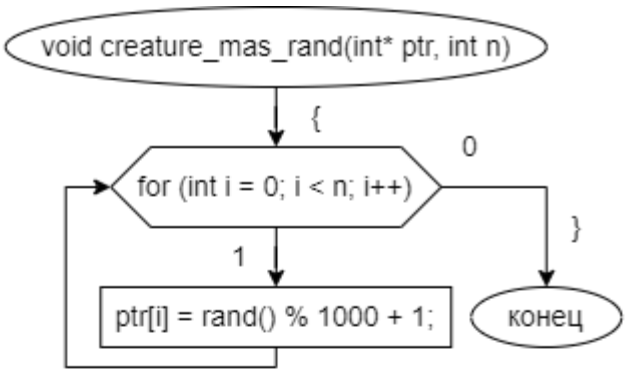
7) В коде используется функция rand для генерации случайных чисел. Она используется в функции creature_mas_rand для создания массива случайных чисел.

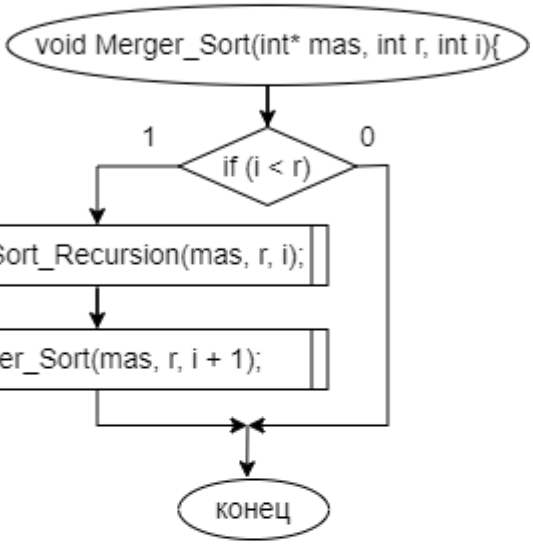
8) В коде используется функция chrono::steady_clock::now для измерения времени выполнения сортировки. Она используется для определения времени, затраченного на сортировку массива.

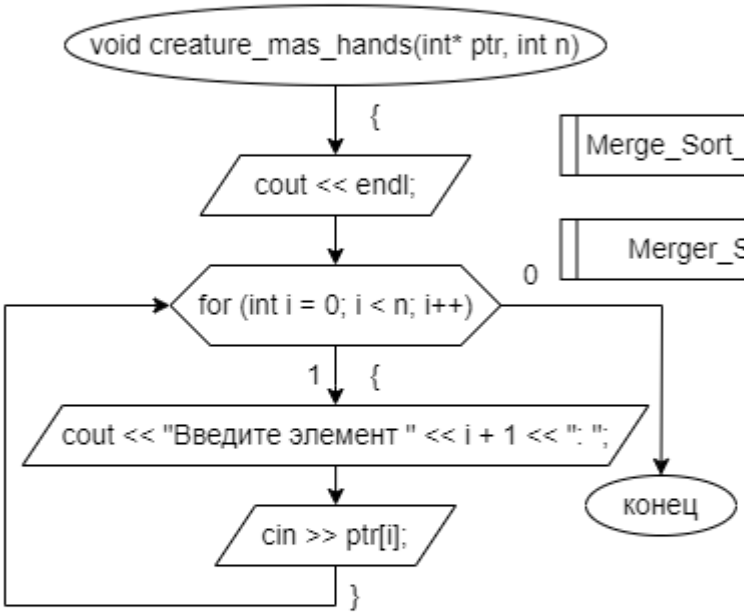
Блок схема

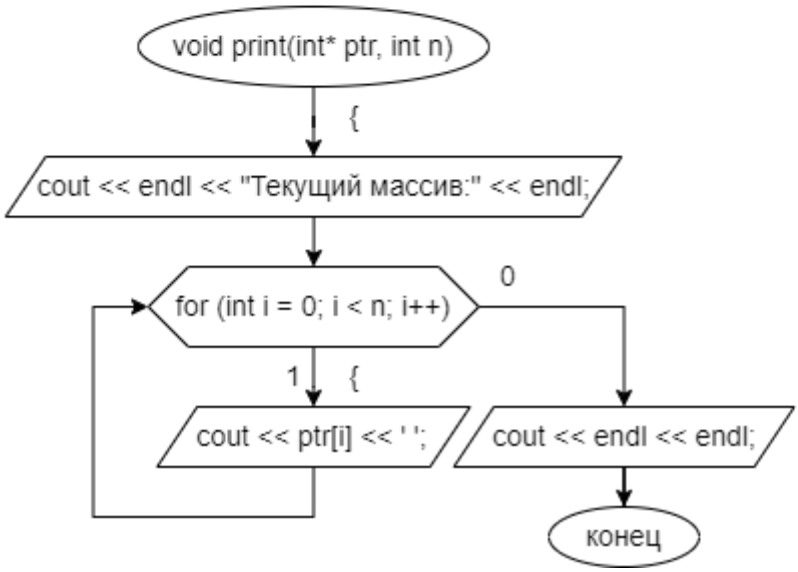

начало
{
cout << "Каким способом вы хотите создать массив? " << endl;
cout << "1 - Ввести ручками" << endl;
cout << "2 - Создать случайно" << endl;
cin >> choice;
}
1
while (choice < 1 || choice > 2);
0
switch (choice)
1
#include <iostream>
#include <vector>
#include <chrono>
using namespace std;
void creature_mas_rand(int* ptr, int n);
void creature_mas_hands(int* ptr, int n);
void print(int* ptr, int n);
void Merge_Sort_Recursion(int* mas, int r, int t);
void Merger_Sort(int* mas, int r, int i);
int main() {
system("chcp 1251 > Null");
srand(time(0));
int choice;
int mas_size = 25;
int* mas = new int[mas_size];
1
case 1: {
creature_mas_hands(mas, mas_size);
break;}
case 2: {
creature_mas_rand(mas, mas_size);
break;}
cout << endl << "Создан массив!" << endl;
print(mas, mas_size);
auto start_time_recursive = chrono::steady_clock::now();
Merger_Sort(mas, mas_size, 0);
auto end_time_recursive = chrono::steady_clock::now();
auto diff_time = end_time_recursive - start_time_recursive;
print(mas, mas_size);
cout << "Сортировка выполнена за " << chrono::duration
<double, milli>(diff_time).count() << " ms" << endl << endl;
return 0;}


void Merge_Sort_Recursion(int* mas, int r, int t) {
int p = 1;
int k = 1;
int i = 0;
bool flag = false;
while (mas[i] <= mas[i + 1] && i < r - 1)
1
{
p++;
i++;
}
0
if (mas[i] > mas[i + 1] && i < r - 1)
1
i++;
flag = true;
0
while (mas[i] <= mas[i + 1] && i < r - 1)
1
{
k++;
i++;
}
0
int num = i + 1;
int* a = new int[p];
int* b = new int[k];
int count = 0;
int j = 0;
int l = 0;
for (i = 0; i < p; i++)
1
a[i] = mas[i];
0
if (flag == true)
1
0
for (i = p; i < p + k; i++)
1
{
b[count++] = mas[i];
}
0
if (flag)
1
0
for (i = 0; i < num; i++)
1
конец
0
if ((a[l] < b[j] || j >= k) && l < p)
1
{
mas[i] = a[l];
l++;
}
0
if ((a[l] > b[j] || l >= p) && j < k)
1
{
mas[i] = b[j];
j++;
}
0
if (a[l] == b[j] && j < k && l < p)
1
{
mas[i] = a[l];
mas[i + 1] = b[j];
l++;
j++;
i++;
}
0









Код

```
#include <iostream>
#include <vector>
#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);
void Merge_Sort_Recursion(int* mas, int r, int t);
void Merger_Sort(int* mas, int r, int i);

int main() {
    system("chcp 1251 > Null");
    srand(time(0));

    int choice;
    int mas_size = 25;
    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
    auto start_time_recursive = chrono::steady_clock::now();  // начальное время
    Merger_Sort(mas, mas_size, 0);
    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);
    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
    cout << endl;
    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_Sort_Recursion(int* mas, int r, int t) {
    int p = 1; // кол-0 элементов в первой серии
    int k = 1; // кол-0 элементов во второй серии
    int i = 0;
    bool flag = false; //определение надобности второй серии

    while (mas[i] <= mas[i + 1] && i < r - 1) { //подсчет кол-а элементов первой
серии
        p++; //увеличиваю кол-о элементов в 1 серии
        i++;
    }
    if (mas[i] > mas[i + 1] && i < r - 1) {
        i++;
        flag = true; //есть вторая серия
        while (mas[i] <= mas[i + 1] && i < r - 1) { //подсчет кол-а элементов
второй серии
            k++; //увеличиваю кол-о элементов в 2 серии
            i++;
        }
    }
    int num = i + 1; // количество элементов в 2х сериях

    int* a = new int[p];//серия 1
    int* b = new int[k];//серия 2

    for (i = 0; i < p; i++) { //заполнение первой серии
        a[i] = mas[i];
    }

    int count = 0;//счетчик
    if (flag == true) { //если есть вторая серия
        for (i = p; i < p + k; i++) { //заполнение второй серии
            b[count++] = mas[i];
        }
    }

    int j = 0;
    int l = 0;

    if (flag) {
        for (i = 0; i < num; i++) {
            if ((a[l] < b[j] || j >= k) && l < p) {
                mas[i] = a[l];
                l++;
            }
            else if ((a[l] > b[j] || l >= p) && j < k) {
                mas[i] = b[j];
                j++;
            }
            else if (a[l] == b[j] && j < k && l < p) {
                mas[i] = a[l];
                mas[i + 1] = b[j];
                l++;
                j++;
                i++;
            }
        }
    }

}
void Merger_Sort(int* mas, int r, int i) {
    if (i < r) {
        Merge_Sort_Recursion(mas, r, i);//сортировка
        Merger_Sort(mas, r, i + 1);//рекурсия
    }
}
```

Результат работы

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

Создан массив!

Текущий массив:
10 924 241 383 403 736 490 182 394 745 923 592 690 582 548 665 892 974 780 418 560 679 144 507 18


Текущий массив:
10 18 144 182 241 383 394 403 418 490 507 548 560 582 592 665 679 690 736 745 780 892 923 924 974

Сортировка выполнена за 0.014 ms
```

Многофазная сортировка

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) Для первичной сортировки серий используется быстрая сортировка. Быстрая сортировка. Функция Quick_Sort принимает качестве аргумента. 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.

4) Многофазная сортировка. В начале кода создается вектор vect_of_series, который содержит подсерии исходного вектора vect. Затем каждая подсерия сортируется отдельно с помощью функции Quick_Sort.

5) Далее создается вектор векторов stack_of_series, который будет использоваться для хранения последовательностей подсерий. Затем последовательности подсерий распределяются по головам с помощью цикла.

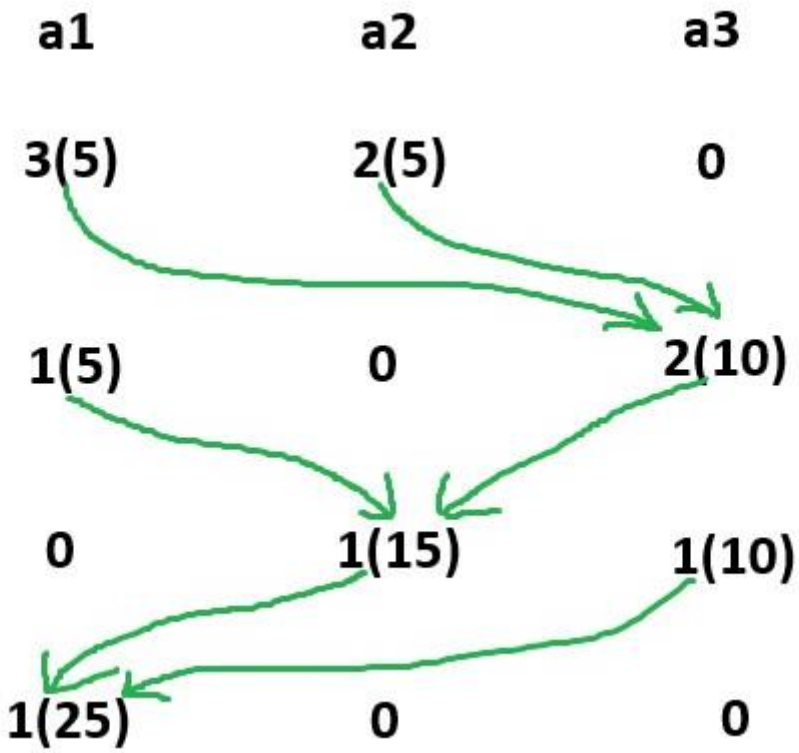
6) После этого начинается основной цикл многофазной сортировки. В начале каждого цикла определяется индекс пустой головы с помощью функции empty_ind. Затем определяется индекс головы с наименьшим числом серий с помощью функции mn_ind. Также определяется индекс головы с наибольшим числом серий с помощью функции mx_ind.

7) Далее создается новая серия с помощью функции new_ser, которая объединяет последнюю серию из головы с наименьшим числом серий и последнюю серию из головы с наибольшим числом серий. Если новая серия содержит 25 элементов, то это означает, что все серии были слиты в одну отсортированную серию, и флаг flag устанавливается в false.

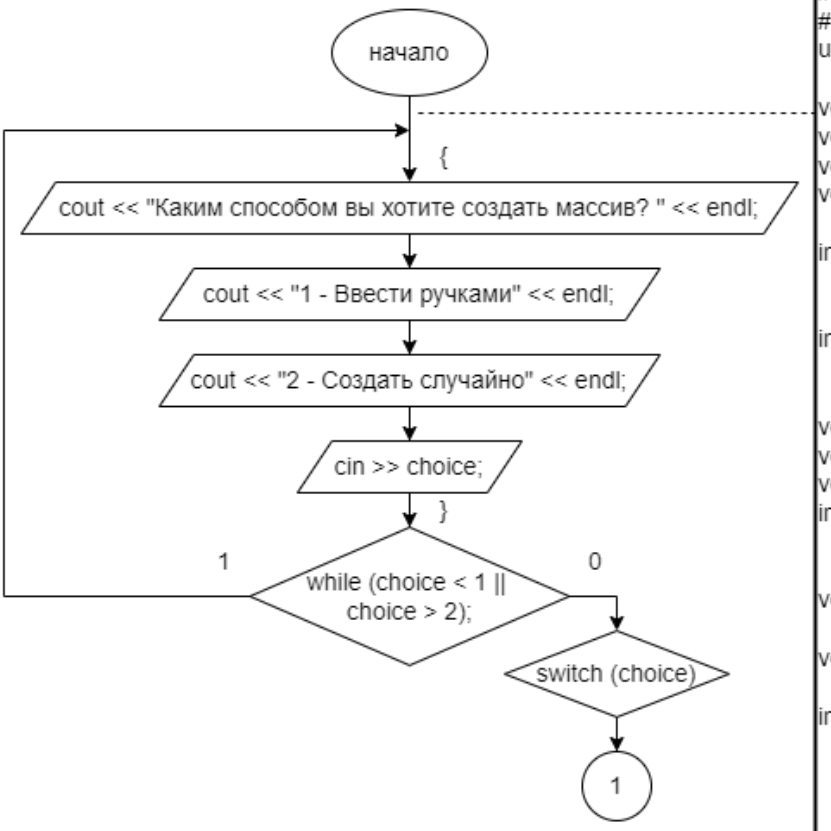
8) Затем новая серия добавляется в пустую голову с помощью функции push_back. После этого последняя серия из головы с наименьшим числом серий и последняя серия из головы с наибольшим числом серий удаляются с помощью функции erase.

9) Цикл продолжается до тех пор, пока флаг flag не станет равным false. В конце кода отсортированная серия возвращается в исходный вектор vect с помощью функции not_an_empty_head. В конце кода вектор stack_of_series освобождается от памяти с помощью функции clear_vector.

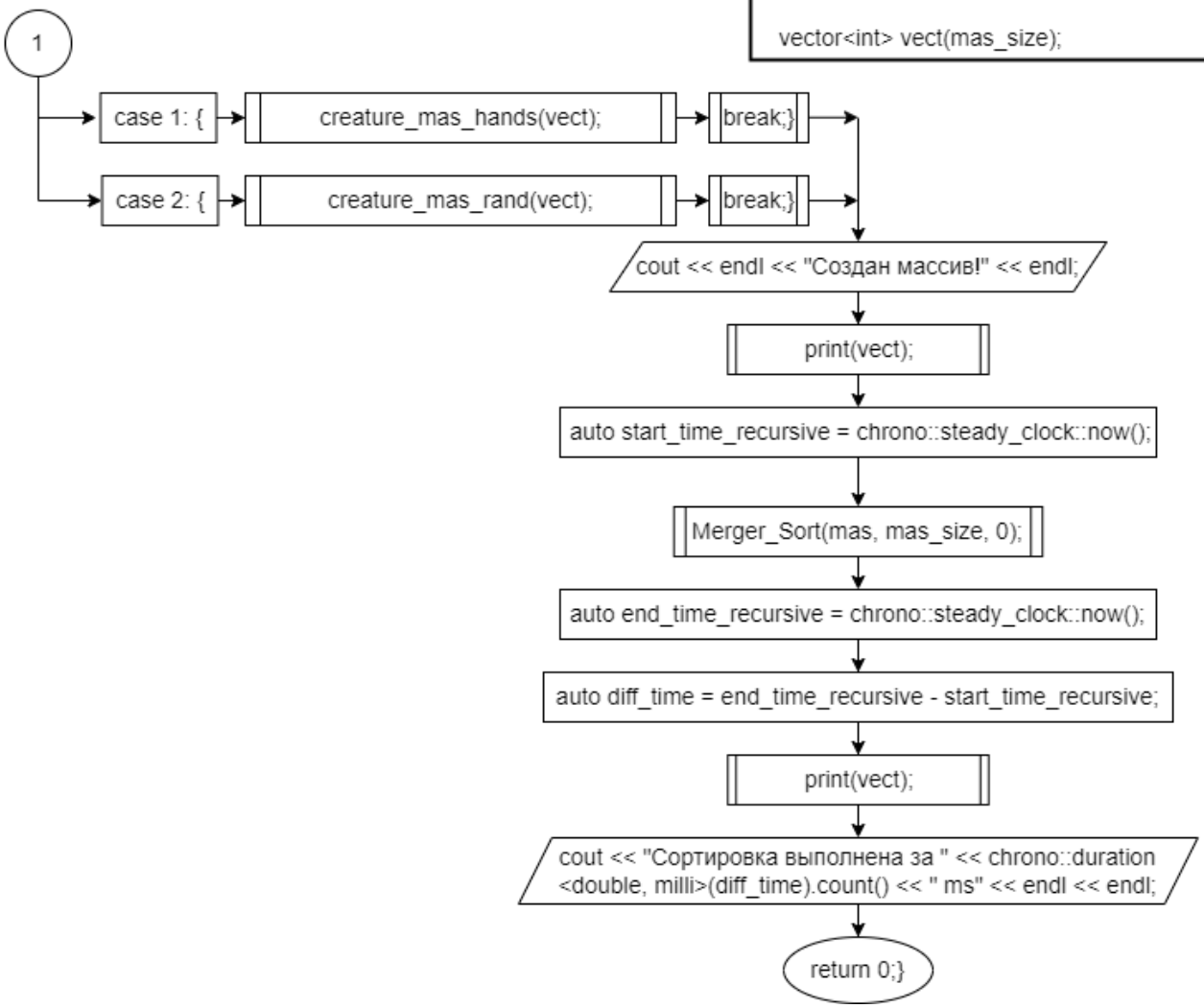
Многофазная сортировка:

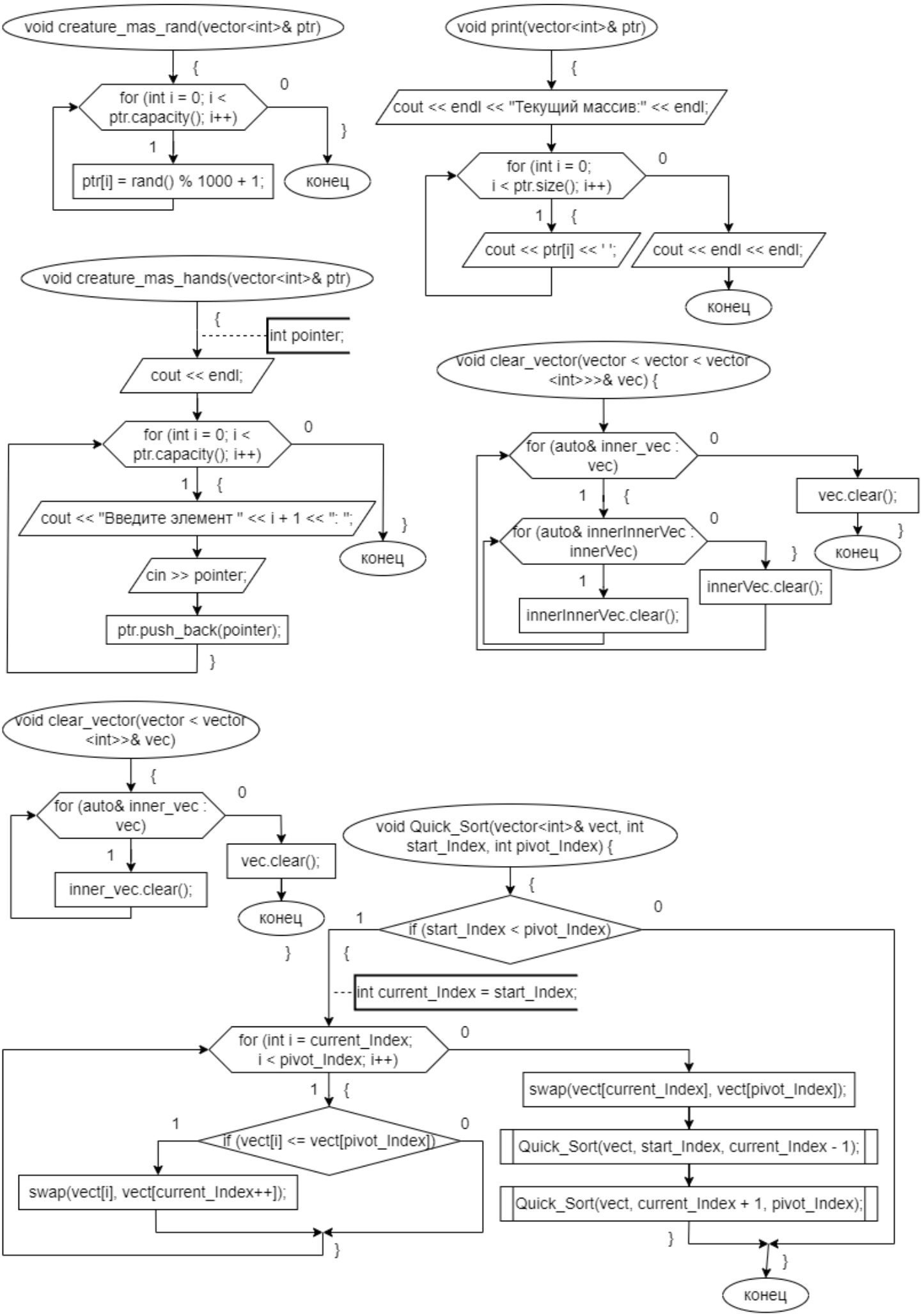


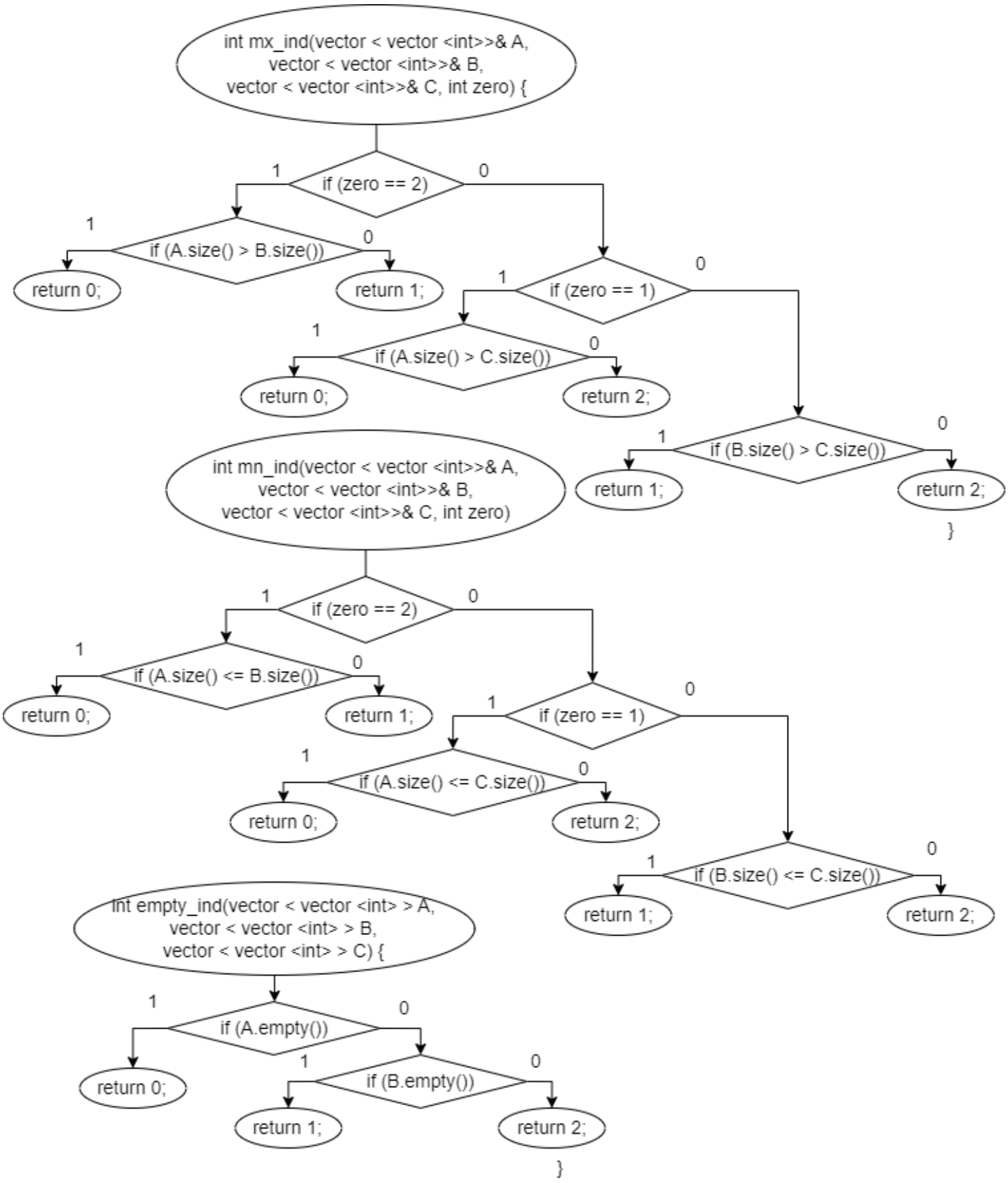
Блок схема

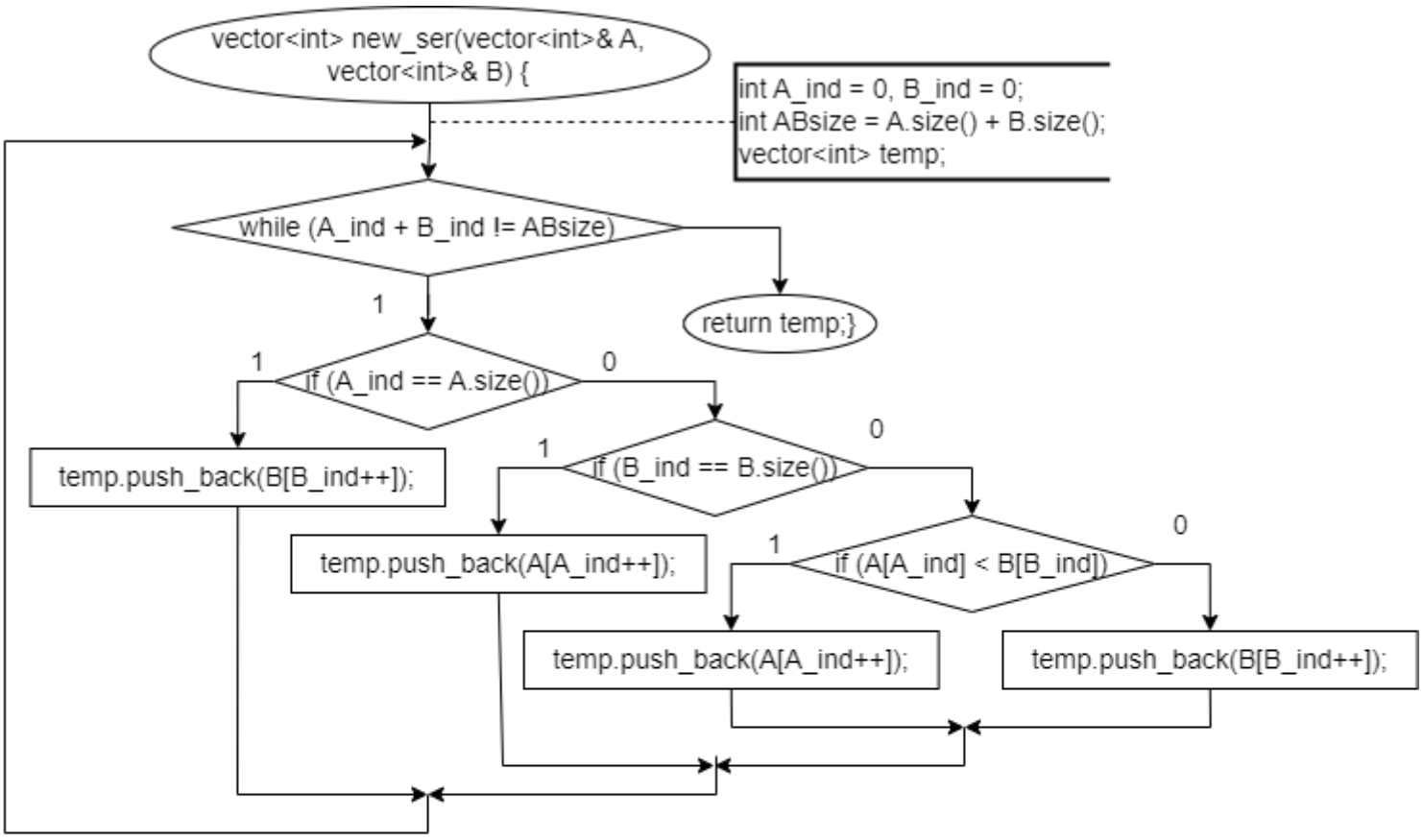


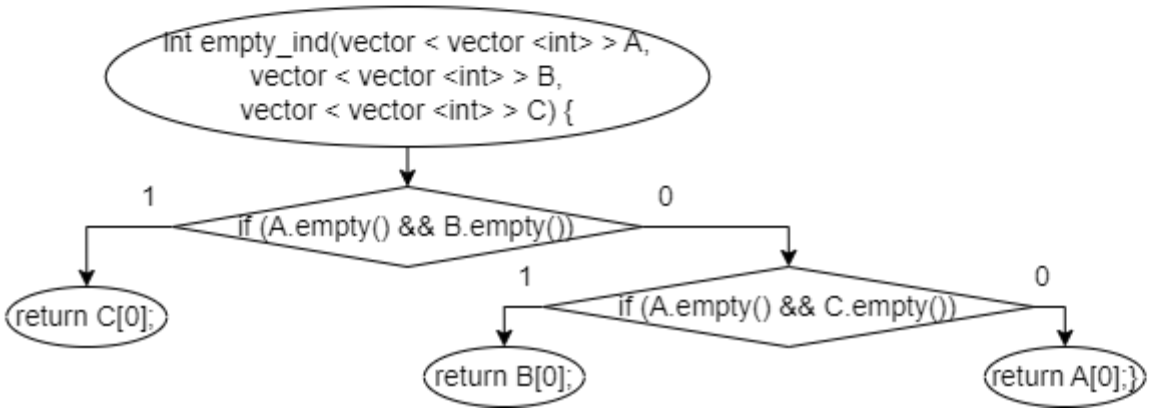

```
#include <iostream>
#include <vector>
#include <chrono>
using namespace std;

void creature_mas_rand(vector<int>& ptr);
void creature_mas_hands(vector<int>& ptr);
void print(vector<int>& ptr);
void Quick_Sort(vector<int>& vect, int start_Index,
        int pivot_Index);
int mx_ind(vector < vector <int>>& A,
        vector < vector <int>>& B,
        vector < vector <int>>& C, int zero);
int mn_ind(vector < vector <int>>& A,
        vector < vector <int>>& B,
        vector < vector <int>>& C, int zero);
void clear_vector(vector < vector <int>>& vec);
void clear_vector(vector < vector < vector <int>>>& vec);
vector<int> new_ser(vector<int>& A, vector<int>& B);
int empty_ind(vector < vector <int> > A,
        vector < vector <int> > B,
        vector < vector <int> > C);
vector<int> not_an_empty_head(vector < vector <int> > A,
        vector < vector <int> > B, vector < vector <int> > C);
void multiphase_sorting(vector<int>& vect);

int main() {
    system("chcp 1251 > Null");
    srand(time(0));

    int choice;
    int mas_size = 25;

    vector<int> vect(mas_size);
```

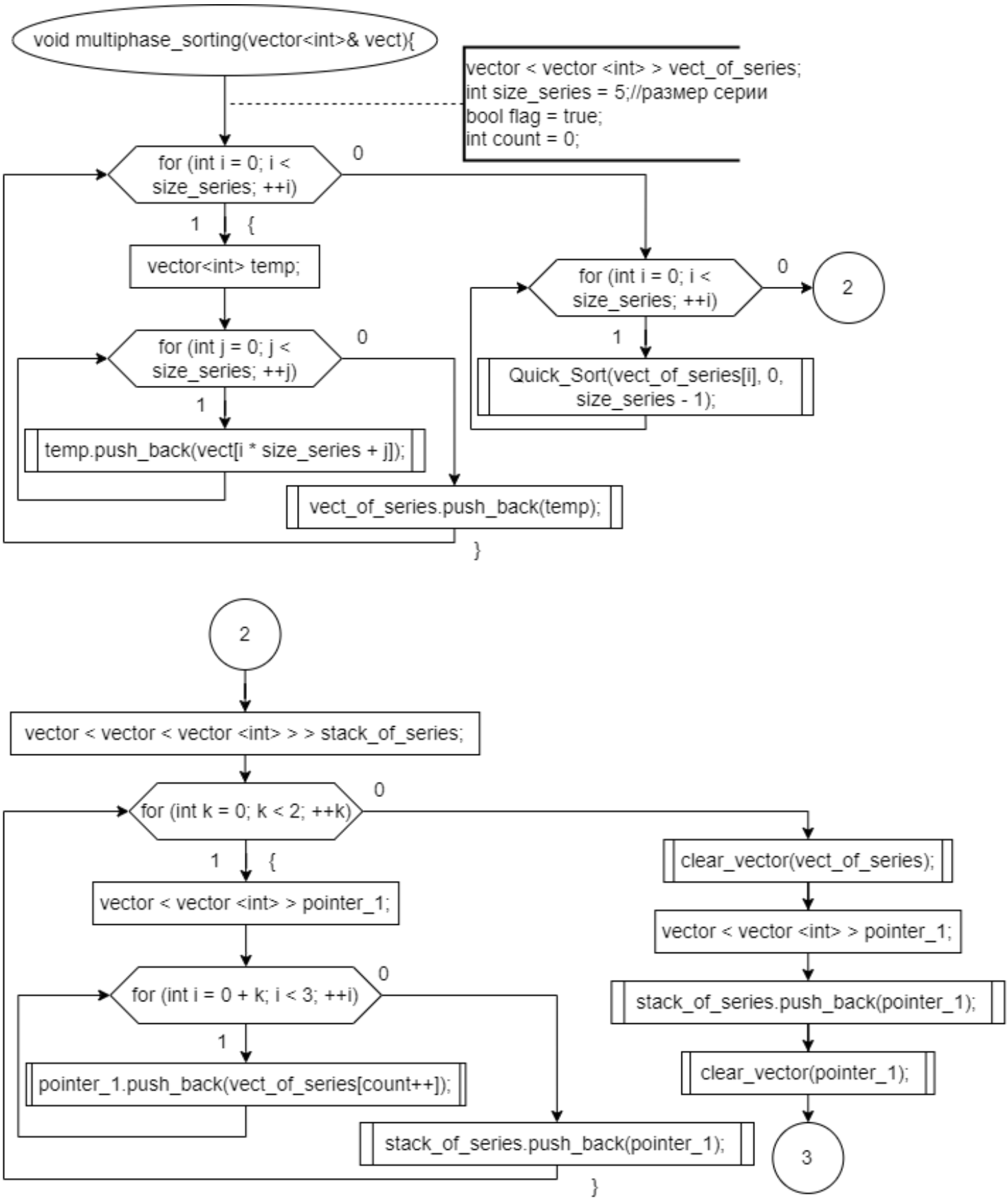


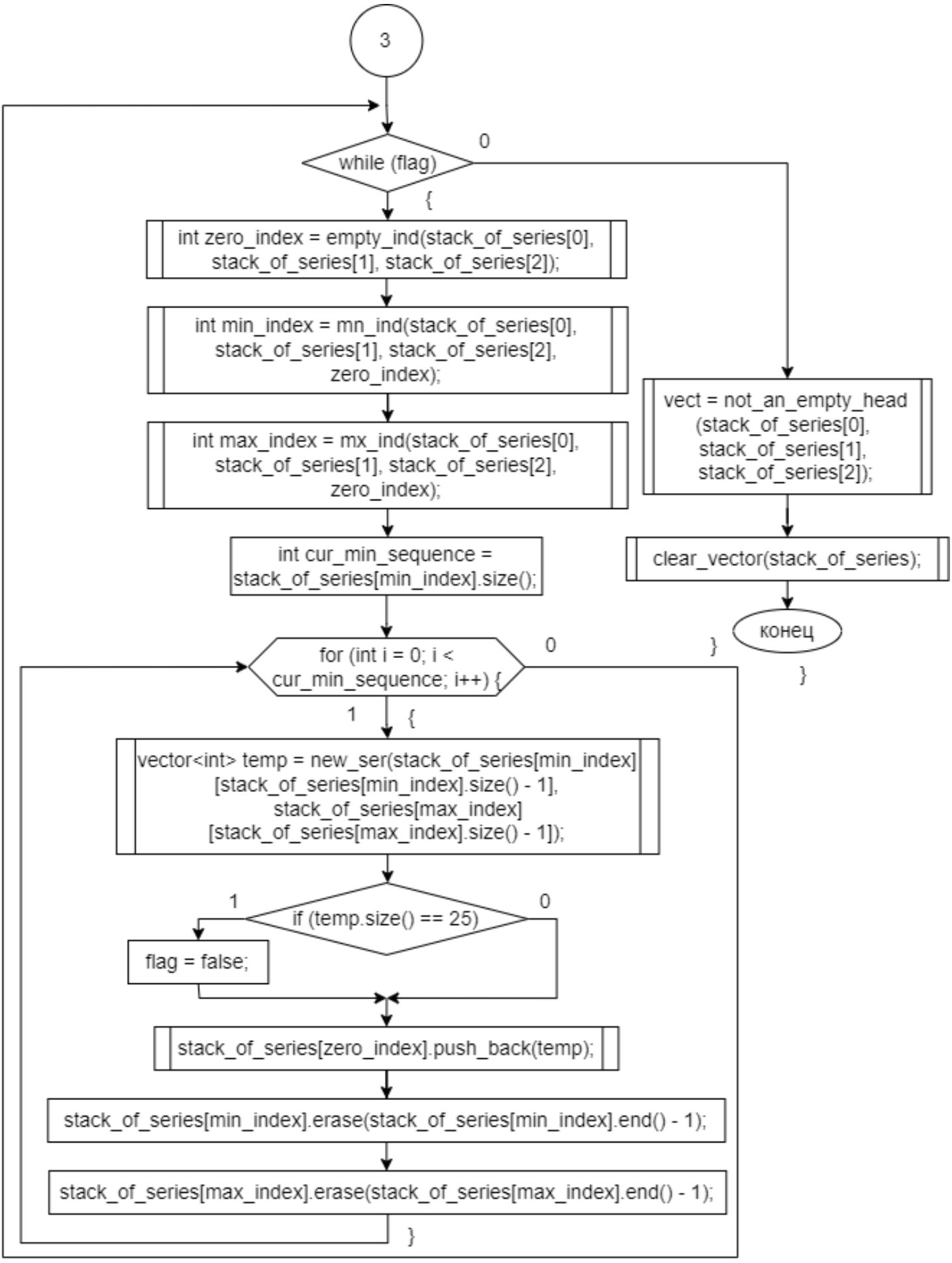

void creature_mas_rand(vector<int>& ptr)
{
for (int i = 0; i < ptr.capacity(); i++)
0
1
}
ptr[i] = rand() % 1000 + 1;
конец
void print(vector<int>& ptr)
{
cout << endl << "Текущий массив:" << endl;
for (int i = 0; i < ptr.size(); i++)
0
1
{
cout << ptr[i] << ' ';
cout << endl << endl;
конец
void creature_mas_hands(vector<int>& ptr)
{
int pointer;
cout << endl;
for (int i = 0; i < ptr.capacity(); i++)
0
1
{
cout << "Введите элемент " << i + 1 << ": ";
cin >> pointer;
ptr.push_back(pointer);
}
конец
void clear_vector(vector < vector < vector <int>>>& vec) {
for (auto& inner_vec : vec)
0
1
{
for (auto& innerInnerVec : innerVec)
0
1
innerInnerVec.clear();
}
innerVec.clear();
vec.clear();
конец
void clear_vector(vector < vector <int>>& vec)
{
for (auto& inner_vec : vec)
0
1
inner_vec.clear();
}
vec.clear();
конец
void Quick_Sort(vector<int>& vect, int start_Index, int pivot_Index) {
{
if (start_Index < pivot_Index)
1
0
{
int current_Index = start_Index;
for (int i = current_Index; i < pivot_Index; i++)
0
1
{
if (vect[i] <= vect[pivot_Index])
1
0
swap(vect[i], vect[current_Index++]);
}
swap(vect[current_Index], vect[pivot_Index]);
Quick_Sort(vect, start_Index, current_Index - 1);
Quick_Sort(vect, current_Index + 1, pivot_Index);
}
}
конец


int mx_ind(vector < vector <int>>& A,
vector < vector <int>>& B,
vector < vector <int>>& C, int zero) {
if (zero == 2)
1
0
if (A.size() > B.size())
1
0
return 0;
return 1;
if (zero == 1)
1
0
if (A.size() > C.size())
1
0
return 0;
return 2;
if (B.size() > C.size())
1
0
return 1;
return 2;
}
int mn_ind(vector < vector <int>>& A,
vector < vector <int>>& B,
vector < vector <int>>& C, int zero)
if (zero == 2)
1
0
if (A.size() <= B.size())
1
0
return 0;
return 1;
if (zero == 1)
1
0
if (A.size() <= C.size())
1
0
return 0;
return 2;
if (B.size() <= C.size())
1
0
return 1;
return 2;
int empty_ind(vector < vector <int> > A,
vector < vector <int> > B,
vector < vector <int> > C) {
if (A.empty())
1
0
return 0;
if (B.empty())
1
0
return 1;
return 2;
}


vector<int> new_ser(vector<int>& A,
vector<int>& B) {
int A_ind = 0, B_ind = 0;
int ABsize = A.size() + B.size();
vector<int> temp;
while (A_ind + B_ind != ABsize)
return temp;}
1
1
if (A_ind == A.size())
0
temp.push_back(B[B_ind++]);
1
if (B_ind == B.size())
0
temp.push_back(A[A_ind++]);
1
if (A[A_ind] < B[B_ind])
0
temp.push_back(A[A_ind++]);
temp.push_back(B[B_ind++]);


int empty_ind(vector < vector <int> > A,
vector < vector <int> > B,
vector < vector <int> > C) {
1
if (A.empty() && B.empty())
0
return C[0];
1
if (A.empty() && C.empty())
0
return B[0];
return A[0];}


void multiphase_sorting(vector<int>& vect){
vector < vector <int> > vect_of_series;
int size_series = 5;//размер серии
bool flag = true;
int count = 0;
for (int i = 0; i < size_series; ++i)
0
1
{
vector<int> temp;
for (int j = 0; j < size_series; ++j)
0
1
temp.push_back(vect[i * size_series + j]);
vect_of_series.push_back(temp);
}
for (int i = 0; i < size_series; ++i)
0
2
1
Quick_Sort(vect_of_series[i], 0, size_series - 1);
2
vector < vector < vector <int> > > stack_of_series;
for (int k = 0; k < 2; ++k)
0
1
{
vector < vector <int> > pointer_1;
for (int i = 0 + k; i < 3; ++i)
0
1
pointer_1.push_back(vect_of_series[count++]);
stack_of_series.push_back(pointer_1);
}
clear_vector(vect_of_series);
vector < vector <int> > pointer_1;
stack_of_series.push_back(pointer_1);
clear_vector(pointer_1);
3


3
while (flag)
0
{
int zero_index = empty_ind(stack_of_series[0], stack_of_series[1], stack_of_series[2]);
int min_index = mn_ind(stack_of_series[0], stack_of_series[1], stack_of_series[2], zero_index);
int max_index = mx_ind(stack_of_series[0], stack_of_series[1], stack_of_series[2], zero_index);
int cur_min_sequence = stack_of_series[min_index].size();
for (int i = 0; i < cur_min_sequence; i++) {
0
1
{
vector<int> temp = new_ser(stack_of_series[min_index] [stack_of_series[min_index].size() - 1], stack_of_series[max_index] [stack_of_series[max_index].size() - 1]);
if (temp.size() == 25)
1
0
flag = false;
stack_of_series[zero_index].push_back(temp);
stack_of_series[min_index].erase(stack_of_series[min_index].end() - 1);
stack_of_series[max_index].erase(stack_of_series[max_index].end() - 1);
}
vect = not_an_empty_head (stack_of_series[0], stack_of_series[1], stack_of_series[2]);
clear_vector(stack_of_series);
конец
}
}

Код

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

void creature_mas_rand(vector<int>& ptr);
void creature_mas_hands(vector<int>& ptr);
void print(vector<int>& ptr);
void Quick_Sort(vector<int>& vect, int start_Index, int pivot_Index);
int mx_ind(vector < vector <int>>& A, vector < vector <int>>& B, vector < vector
<int>>& C, int zero);
int mn_ind(vector < vector <int>>& A, vector < vector <int>>& B, vector < vector
<int>>& C, int zero);
void clear_vector(vector < vector <int>>& vec);
void clear_vector(vector < vector < vector <int>>>& vec);
vector<int> new_ser(vector<int>& A, vector<int>& B);
int empty_ind(vector < vector <int> > A, vector < vector <int> > B, vector <
vector <int> > C);
vector<int> not_an_empty_head(vector < vector <int> > A, vector < vector <int> >
B, vector < vector <int> > C);
void multiphase_sorting(vector<int>& vect);

int main() {
    system("chcp 1251 > Null");
    srand(time(0));

    int choice;
    int mas_size = 25;

    vector<int> vect(mas_size);

    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(vect);
        break;
    }
    default: {
        creature_mas_rand(vect); //создание массива случайно
        break;
    }
    }

    cout << endl << "Создан массив!" << endl;
    print(vect); //вывод массива
    auto start_time_recursive = chrono::steady_clock::now();  // начальное время
    multiphase_sorting(vect);//многофазная сортировка
    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

    cout << "Массив отсортирован!" << endl;
    print(vect);
    cout << "Сортировка выполнена за " << chrono::duration <double,
milli>(diff_time).count() << " ms" << endl << endl;
    return 0;
}
```

```
void creature_mas_rand(vector<int>& ptr) { //создаю массив через датчик случайных
чисел
    for (int i = 0; i < ptr.size(); i++) {
        ptr[i] = rand() % 1000 + 1; //случайное число от 1 до 1000
    }
}
void creature_mas_hands(vector<int>& ptr) { //создание массива с клавиатуры
    cout << endl;
    for (int i = 0; i < ptr.size(); i++) {
        cout << "Введите элемент " << i + 1 << ": ";
        cin >> ptr[i];
    }
}

void print(vector<int>& ptr) { //вывод текущего массива
    cout << endl << "Текущий массив:" << endl;
    for (int i = 0; i < ptr.size(); i++) { //прохожу по массиву
        cout << ptr[i] << ' ';
    }
    cout << endl << endl;
}

void Quick_Sort(vector<int>& vect, int start_Index, int pivot_Index) { //быстрая
сортировка
    if (start_Index < pivot_Index) {
        int current_Index = start_Index;
        for (int i = current_Index; i < pivot_Index; i++) { //иду до опорного
элемента
            if (vect[i] <= vect[pivot_Index]) {
                swap(vect[i], vect[current_Index++]); //меняю элементы местами
            }
        }
        swap(vect[current_Index], vect[pivot_Index]); //меняю элементы местами
        Quick_Sort(vect, start_Index, current_Index - 1);
        Quick_Sort(vect, current_Index + 1, pivot_Index);
    }
}

int mx_ind(vector < vector <int>>& A, vector < vector <int>>& B, vector < vector
<int>>& C, int zero) {
    if (zero == 2) {
        if (A.size() > B.size()) {
            return 0;
        }
        return 1;
    }
    else if (zero == 1) {
        if (A.size() > C.size()) {
            return 0;
        }
        return 2;
    }
    else {
        if (B.size() > C.size()) {
            return 1;
        }
        return 2;
    }
}

int mn_ind(vector < vector <int>>& A, vector < vector <int>>& B, vector < vector
<int>>& C, int zero) {
    if (zero == 2) {
        if (A.size() <= B.size()) {
            return 0;
        }
        return 1;
```

```
    }
    if (zero == 1) {
        if (A.size() <= C.size()) {
            return 0;
        }
        return 2;
    }
    else {
        if (B.size() <= C.size()) {
            return 1;
        }
        return 2;
    }
}

void clear_vector(vector < vector <int>>& vec) { //очищаю двумерный вектор
    for (auto& inner_vec : vec) {
        inner_vec.clear();
    }
    vec.clear();
}
void clear_vector(vector < vector < vector <int>>>& vec) { //очищаю трехмерный
вектор
    for (auto& innerVec : vec) {
        for (auto& innerInnerVec : innerVec) {
            innerInnerVec.clear();
        }
        innerVec.clear();
    }
    vec.clear();
}

vector<int> new_ser(vector<int>& A, vector<int>& B) {//новая серия из двух серий
    int A_ind = 0, B_ind = 0;
    int ABsize = A.size() + B.size();
    vector<int> temp;

    while (A_ind + B_ind != ABsize) {
        if (A_ind == A.size()) {
            temp.push_back(B[B_ind++]);
        }
        else if (B_ind == B.size()) {
            temp.push_back(A[A_ind++]);
        }
        else {
            if (A[A_ind] < B[B_ind]) {
                temp.push_back(A[A_ind++]);
            }
            else {
                temp.push_back(B[B_ind++]);
            }
        }
    }
    return temp;
}

int empty_ind(vector < vector <int> > A, vector < vector <int> > B, vector <
vector <int> > C) {//нахожу пустую голову
    if (A.empty()) {
        return 0;
    }
    else if (B.empty()) {
        return 1;
    }
    return 2;
}
```

```
vector<int> not_an_empty_head(vector < vector <int> > A, vector < vector <int> >
B, vector < vector <int> > C) {
    if (A.empty() && B.empty()) {//ищу не пусткю последовательность
        return C[0];
    }
    else if (A.empty() && C.empty()) {
        return B[0];
    }
    return A[0];
}

void multiphase_sorting(vector<int>& vect) { //многофазная сортировка
    vector < vector <int> > vect_of_series;
    int size_series = 5;//размер серии
    bool flag = true;
    int count = 0;

    for (int i = 0; i < size_series; ++i) { //5 серий
        vector<int> temp;
        for (int j = 0; j < size_series; ++j)//по 5 элементов
            temp.push_back(vect[i * size_series + j]);
        vect_of_series.push_back(temp);
    }

    for (int i = 0; i < size_series; ++i) { //сортирую каждую серию отдельно
        Quick_Sort(vect_of_series[i], 0, size_series - 1);
    }
    vector < vector < vector <int> > > stack_of_series; //вектор векторов серий

    for (int k = 0; k < 2; ++k) { //рассовываю по головам последовательности серий
        vector < vector <int> > pointer_1;
        for (int i = 0 + k; i < 3; ++i) {//пихаю последовательности в головы
            pointer_1.push_back(vect_of_series[count++]);
        }
        stack_of_series.push_back(pointer_1);
    }

    clear_vector(vect_of_series); //освобождаю память
    vector < vector <int> > pointer_1;
    stack_of_series.push_back(pointer_1);//пустая голова с пустой серией
    clear_vector(pointer_1); //освобождаю память

    while (flag) { //пока не солью все серии в одну
        int zero_index = empty_ind(stack_of_series[0], stack_of_series[1],
stack_of_series[2]); //индекс пустой головы
        int min_index = mn_ind(stack_of_series[0], stack_of_series[1],
stack_of_series[2], zero_index);//индекс с меньшим числом серий
        int max_index = mx_ind(stack_of_series[0], stack_of_series[1],
stack_of_series[2], zero_index);//индекс с большим числом серий
        int cur_min_sequence = stack_of_series[min_index].size(); //по сколько
серий сплавлять
        for (int i = 0; i < cur_min_sequence; i++) {//запись сплавленных серий в
запасную голову
            vector<int> temp =
new_ser(stack_of_series[min_index][stack_of_series[min_index].size() - 1],
                stack_of_series[max_index][stack_of_series[max_index].size() -
1]); //сплав серий
            if (temp.size() == 25) {
                flag = false; //все серии слиты в одну отсортированную
            }
            stack_of_series[zero_index].push_back(temp);//добавляю отсортированную
серию
            stack_of_series[min_index].erase(stack_of_series[min_index].end() -
1); //убираю сплавленную серию
            stack_of_series[max_index].erase(stack_of_series[max_index].end() -
1); //убираю сплавленную серию
        }
```

```
    }
    vect = not_an_empty_head(stack_of_series[0], stack_of_series[1],
stack_of_series[2]);
    clear_vector(stack_of_series); //освобождаю память
}
```

Результат работы

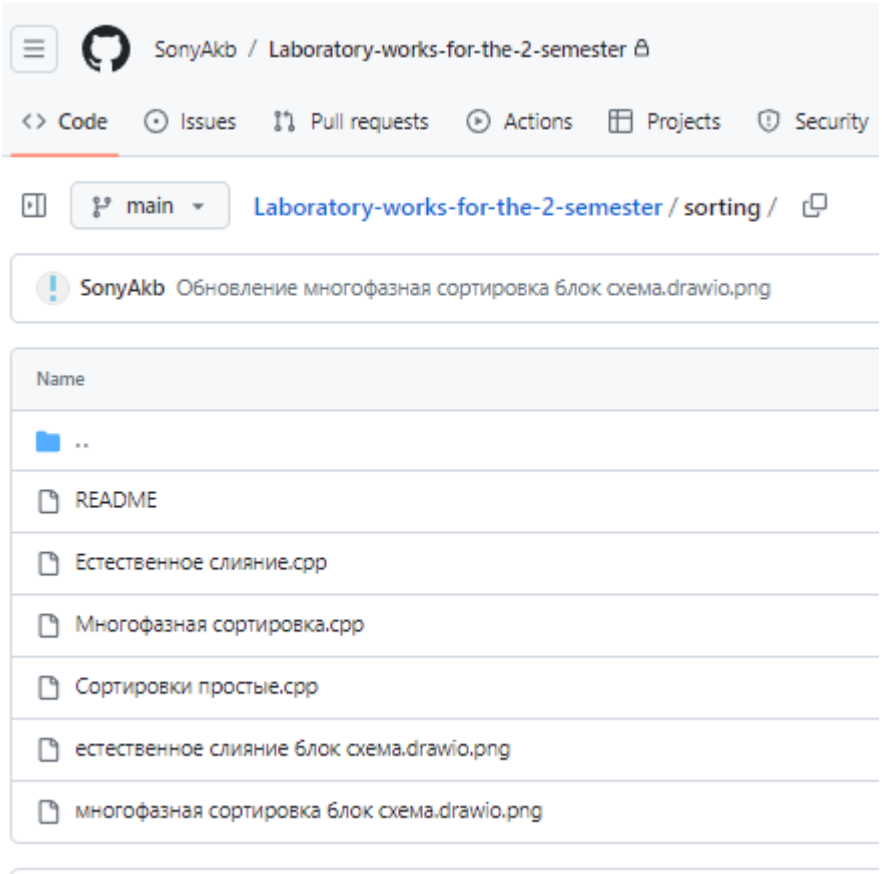
```
Каким способом вы хотите создать массив?
1 - Ввести ручками
2 - Создать случайно
2

Создан массив!

Текущий массив:
160 831 793 184 354 757 590 600 263 78 888 618 270 49 580 17 336 566 457 208 385 907

Массив отсортирован!

Текущий массив:
17 49 78 160 184 208 263 270 336 354 385 457 499 566 580 590 600 618 627 698 757 793

Сортировка выполнена за 0.1275 ms
```

Вывод

В ходе работы я применила знания о работе различных сортировок, в числе которых: сортировка естественным слиянием и многофазная. Мне удалось реализовать поставленную задачу: были созданы все необходимые сортировки.

GitHub

Ссылка: https://github.com/SonyAkb/Laboratory-works-for-the-2-semester/tree/main/sorting


SonyAkb / Laboratory-works-for-the-2-semester
Code
Issues
Pull requests
Actions
Projects
Security
main
Laboratory-works-for-the-2-semester / sorting /
SonyAkb Обновление многофазная сортировка блок схема.drawio.png
Name
..
README
Естественное слияние.cpp
Многофазная сортировка.cpp
Сортировки простые.cpp
естественное слияние блок схема.drawio.png
многофазная сортировка блок схема.drawio.png