Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования

**«Пермский национальный исследовательский
политехнический университет»**

Электротехнический факультет
Кафедра «Информационные технологии и автоматизированные системы»
направление подготовки: 09.03.01– «Информатика и вычислительная техника»

Лабораторная работа № 12
по дисциплине
«Информатика»
на тему
«Сложные сортировки»

Выполнил студент гр. ИВТ-23-1б
Южаков Федор Алексеевич

Проверил:
доцент кафедры ИТАС
Денис Владимирович Яруллин

(оценка) (подпись)

(дата)

г. Пермь, 2024

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

1 Вариант задания

Постановка задачи.

Выполнить сортировку массива 6 методами сортировки:

1. Быстрая сортировка;
2. Сортировка Хоара;
3. Сортировка слиянием;
4. Сортировка подсчетом;
5. Блочная сортировка;
6. Сортировка Шелла.

Массив статический и содержит 25 элементов, заполнение массива организовать любым удобным способом, при этом учесть, что некоторые сортировки не работают или плохо работают с некоторыми числами.

Создать меню, в котором пользователь выбирает необходимый метод сортировки.

2 Анализ задачи

В данной задаче основная часть программы находится в функции меню, которая вызывается из основного тела программы. В функцию меню передается массив и его размер. Сама функция состоит из текстового меню, запроса от пользователя номера необходимого пункта меню и оператора множественного выбора, где цифры от одного до шести – различные виды сортировок, а 0 – выход из программы, default – проверка на дурака, после которой снова вызывается меню. В каждом кейсе получаем от пользователя границы случайных чисел, если сортировка не работает с какими-нибудь числами, то переспрашиваем пользователя в случае, если он ввел неверные границы, если все верно, то запускаем функцию заполнения массива случайными числами в указанных границах. После работы этой функции выводим массив. Затем вызываем необходимую сортировку и снова выводим массив, на этот раз отсортированный. После работы снова вызываем меню, чтобы пользователь мог запустить другие сортировки или же ту же самую, но с другими данными.

Алгоритмы сортировок

Общий алгоритм быстрых сортировок:

1. Если диапазон менее 2-х элементов, сразу же прекращаем действия, т.к. сортировать нечего.

2. Выбираем точку (pivot, значение), которая встречается в диапазоне. Конкретный принцип выбора такого значения зависит от выбранного извода и может включать элементы случайности (рандомизации).
3. Разбиваем диапазон на части: меняем порядок элементов, одновременно определяя точку деления таким образом, чтобы элементы со значением меньшим, чем pivot, шли до точки деления, тогда как элементы с большим значением — после точки деления. Элементы, равные значению pivot, могут записываться в любую часть (зависит от извода). Т.к. минимум одно значение равно pivot, большинство изводов стремятся точку деления приравнять к pivot.
4. Рекурсивно применяем быструю сортировку к каждой из полученных частей (до и после точки деления), пока алгоритм это позволяет (т.е. пока он не остановится на шаге 1)

Алгоритм сортировки слиянием:

1. Принимаем на вход массив
2. Определяем его левую (l) и правую (r) границы, а также середину массива (округляя в большую сторону, если нужно)
3. Выполняем рекурсивное слияние.

Алгоритм сортировки подсчетом:

1. Находим максимальный элемент массива
2. Создаем массив для подсчета длиной max+1, заполняем его 0
3. В массиве подсчета храним число вхождений каждого элемента (на соответствующих индексах)
4. Вычисляем префиксную сумму для элементов в массиве подсчета (это сумма элементов, стоящих до заданного и самого этого элемента)
5. Начинаем идти с конца исходного массива и обновляем выходной массив при помощи алгоритма:

```
outArr[countArr[inpArr[i]]-1] = inpArr[i];
countArr[inpArr[i]] = countArr[inpArr[i]]--;
```

Алгоритм блочной сортировки:

1. Создадим массив, где каждый слот будет ведром.
2. Распределим элементы по ведрам на основании их значений.
   - берем элементы по очереди
   - умножаем элемент на размер массива с ведрами

- получаем целое число, которое и даст индекс ведер
- элемент помещается в ведро с этим индексом
- повторяем для всех остальных элементов

3. Сортируем каждое отдельно взятое ведро, берем любой алгоритм простой сортировки
4. Собираем по очереди из каждого ведра элементы и помещаем их в исходный массив
   - идем по ведрам по очереди
   - идем по элементам ведер по очереди
   - помещаем каждый элемент в том же порядке в исходный массив
   - как только все элементы помещены в исходный массив, ведро считается пустым
   - переходим к следующему ведру
5. Массив отсортирован

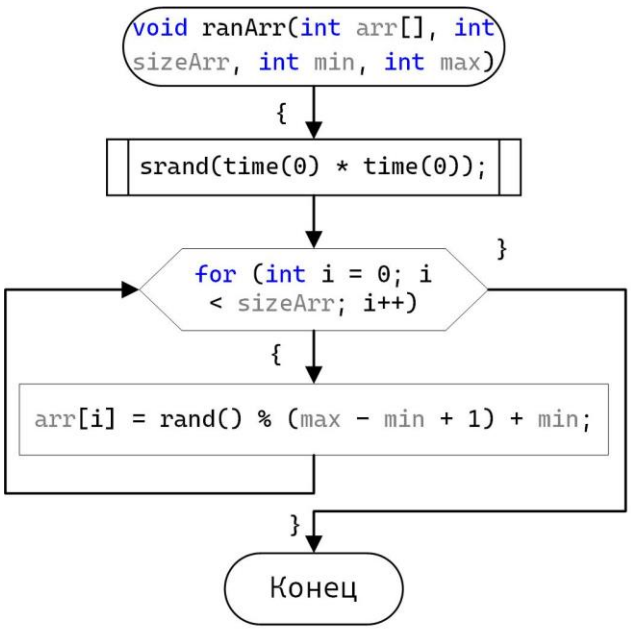
Алгоритм сортировки Шелла:

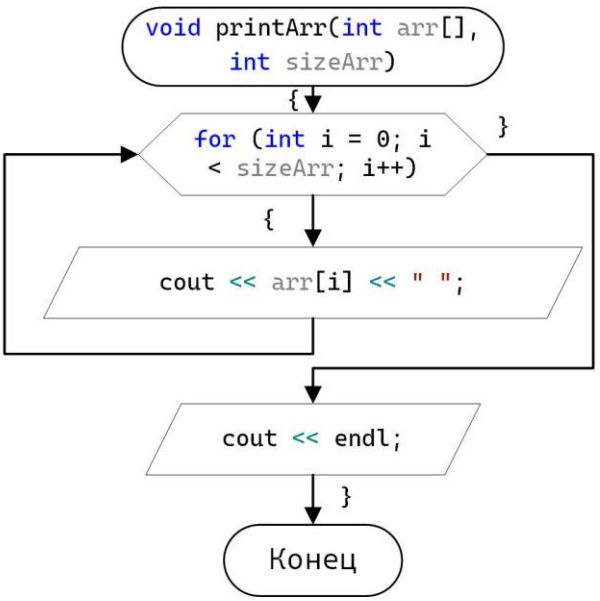
1. Выбрать размер окна
2. Разделить массив на несколько меньших частей, каждая должна нацело делиться на размер окна
3. Отсортировать каждую из частей при помощи простой сортировки вставкой
4. Продолжать с шага 1 до тех пор, пока не отсортируется весь массив

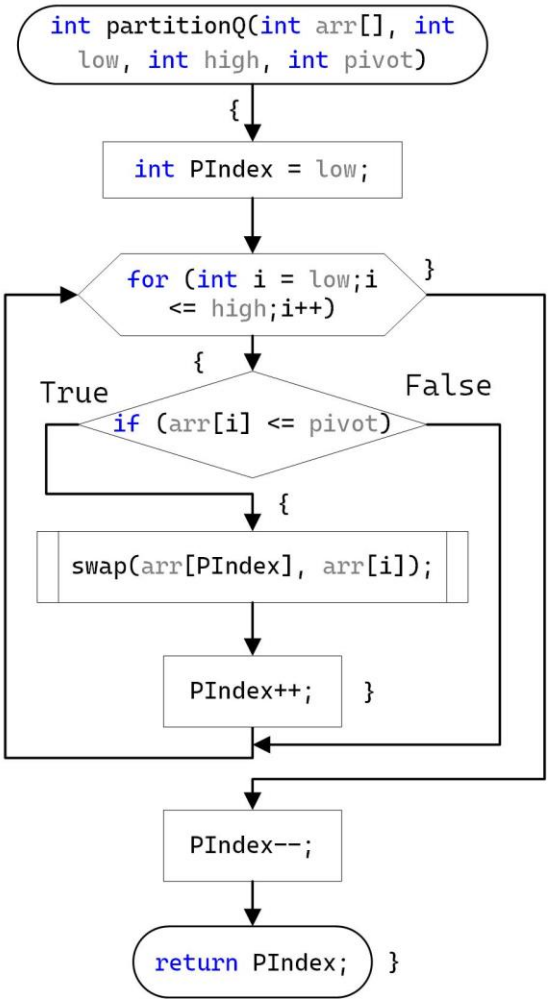
После завершения функции «меню» программа завершается.

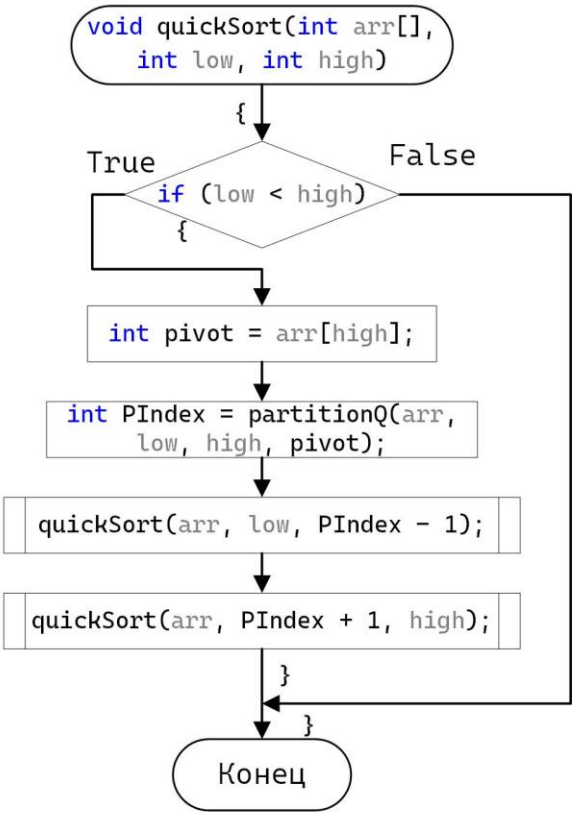
3 Блок схема

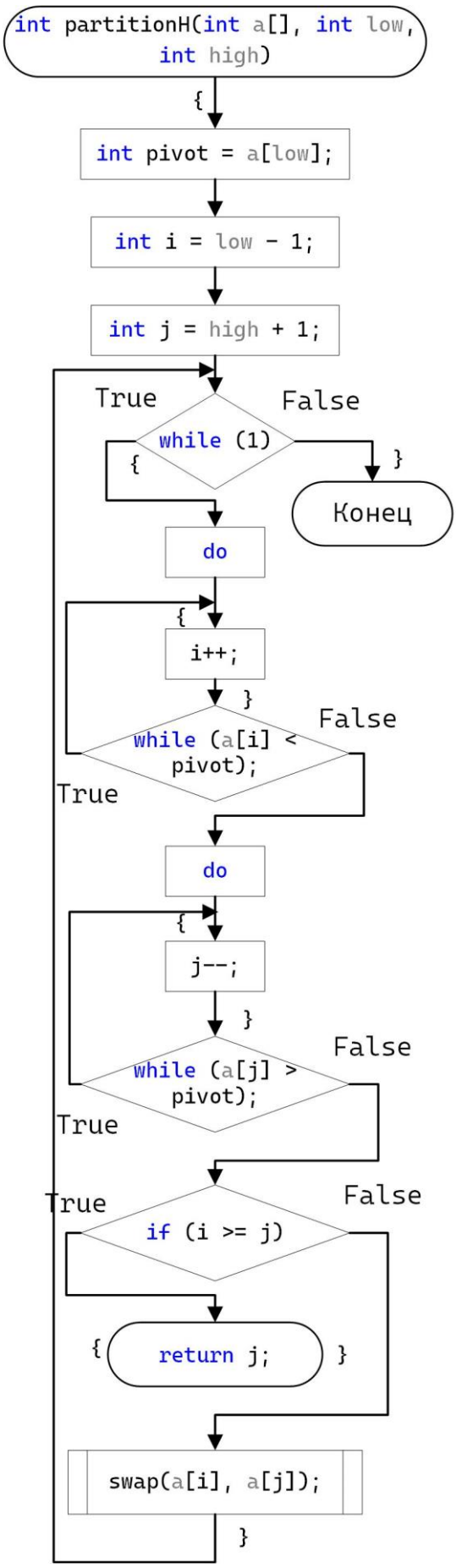
Проанализировав задачу, составим подробную блок схему основной программы и использованных функций.

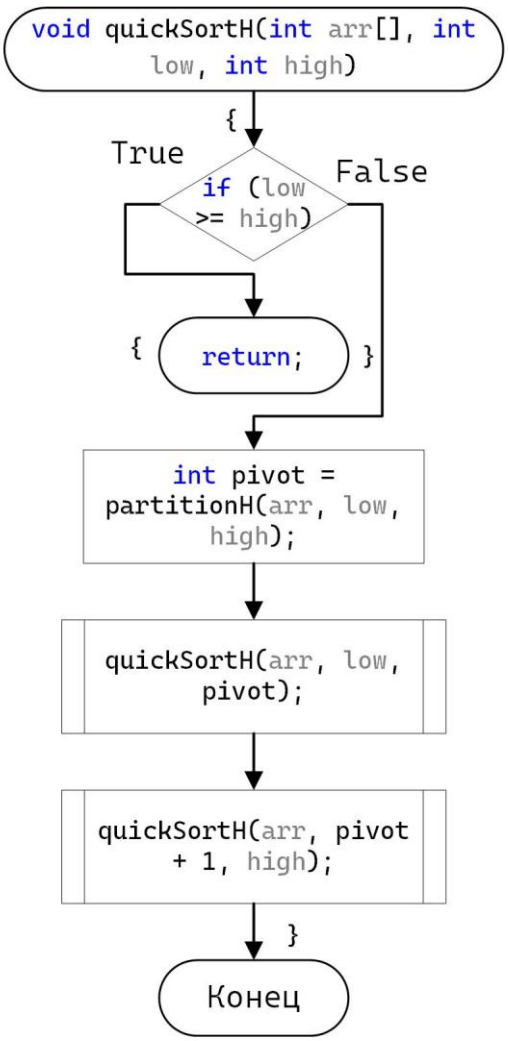

void ranArr(int arr[], int sizeArr, int min, int max)
{
srand(time(0) * time(0));
for (int i = 0; i < sizeArr; i++)
{
arr[i] = rand() % (max - min + 1) + min;
}
}
Конец

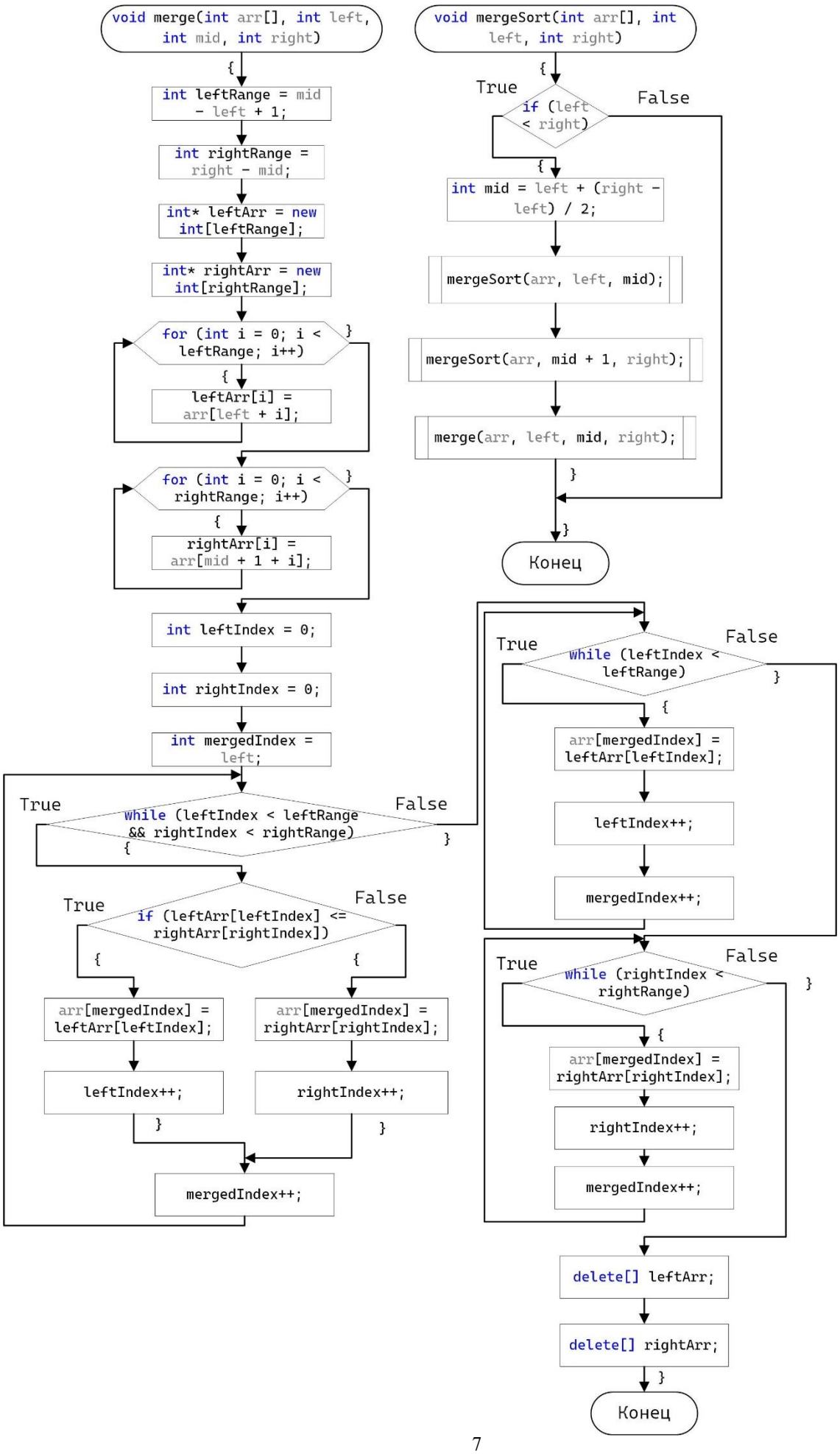

void printArr(int arr[], int sizeArr)
{
for (int i = 0; i < sizeArr; i++)
{
cout << arr[i] << " ";
}
cout << endl;
}
Конец

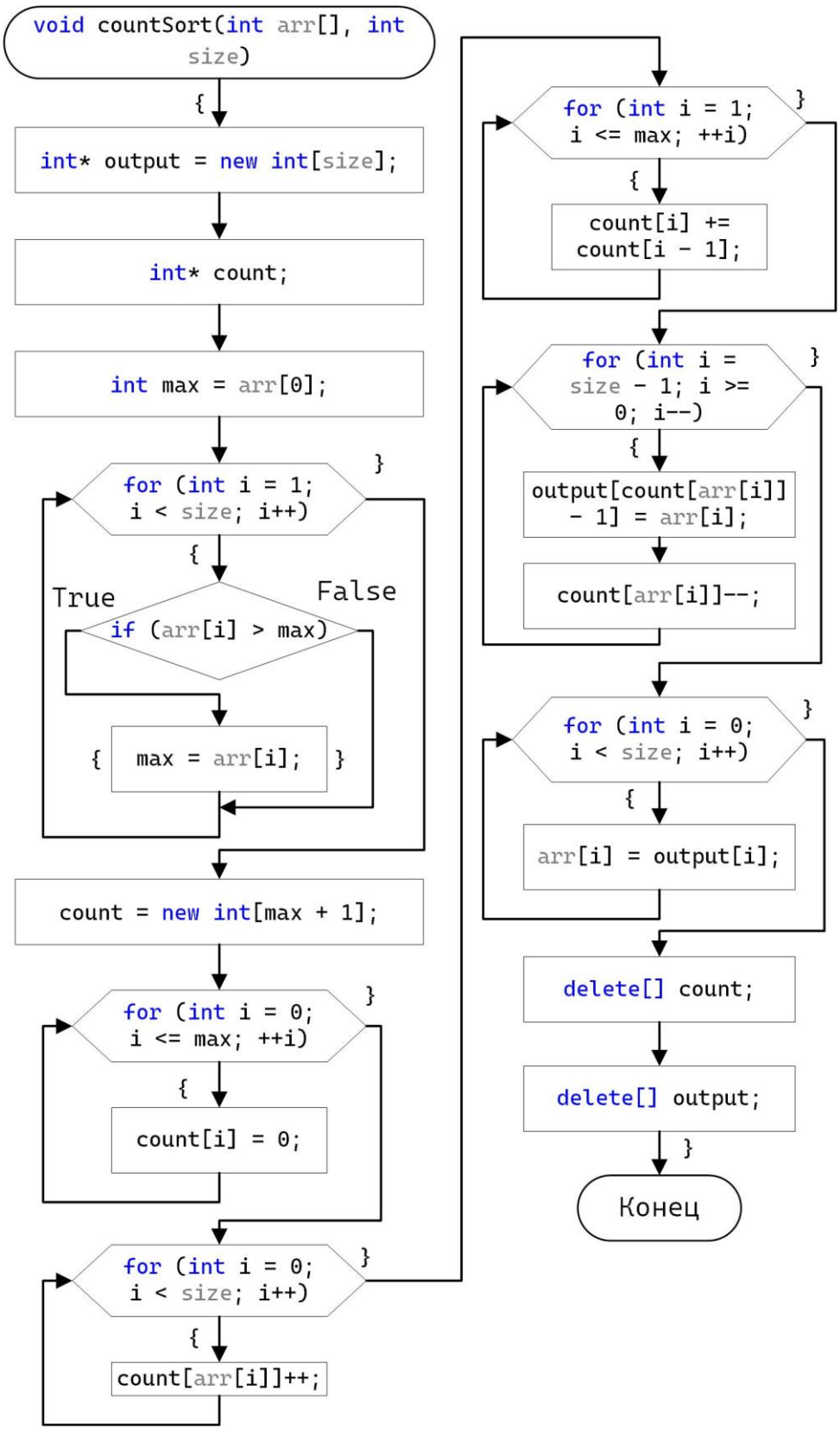

int partitionQ(int arr[], int low, int high, int pivot)
{
int PIndex = low;
for (int i = low;i <= high;i++)
{
True
if (arr[i] <= pivot)
False
{
swap(arr[PIndex], arr[i]);
PIndex++;
}
}
PIndex--;
return PIndex;
}

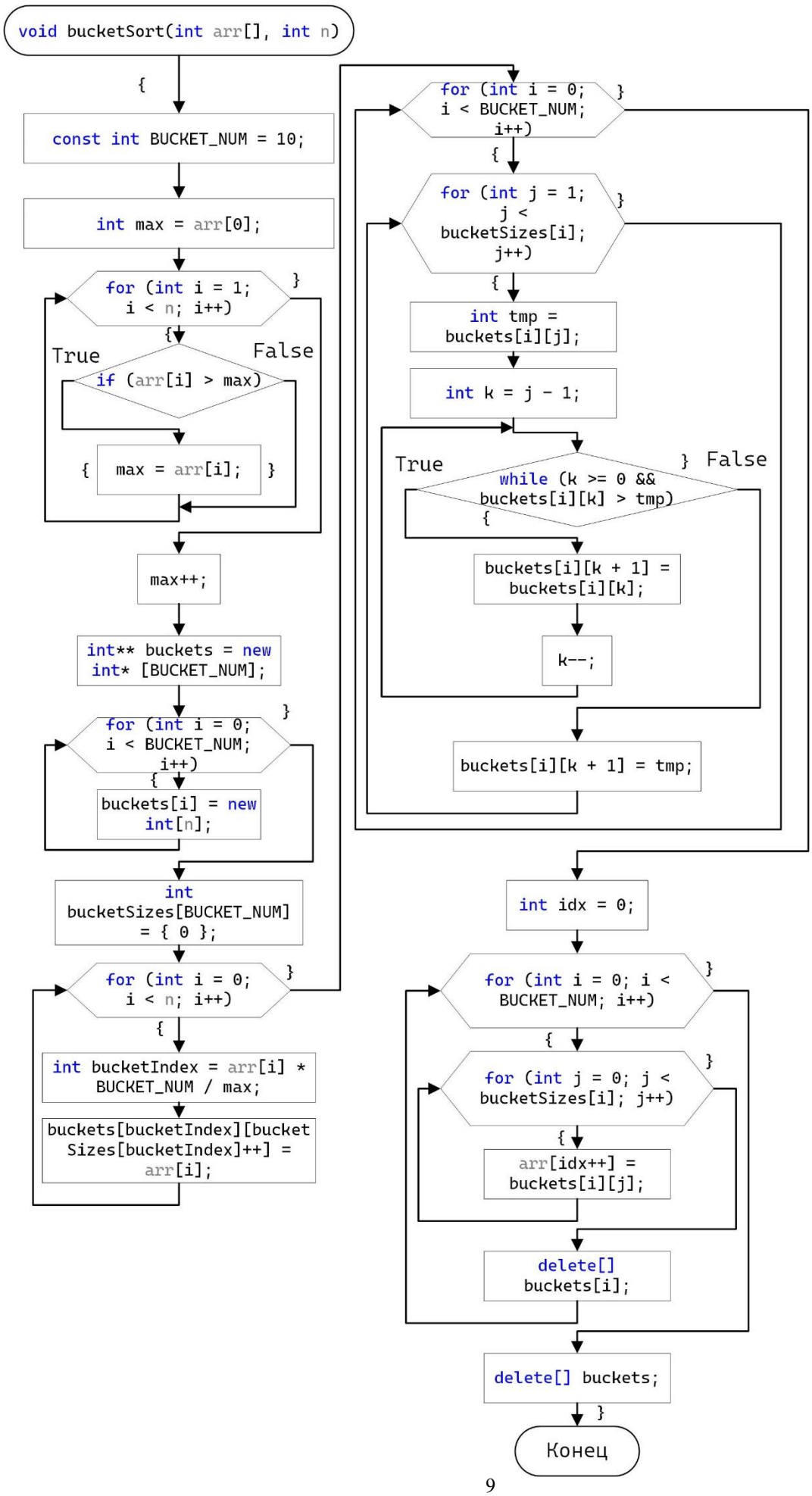

void quickSort(int arr[], int low, int high)
{
True
if (low < high)
False
{
int pivot = arr[high];
int PIndex = partitionQ(arr, low, high, pivot);
quickSort(arr, low, PIndex - 1);
quickSort(arr, PIndex + 1, high);
}
}
Конец

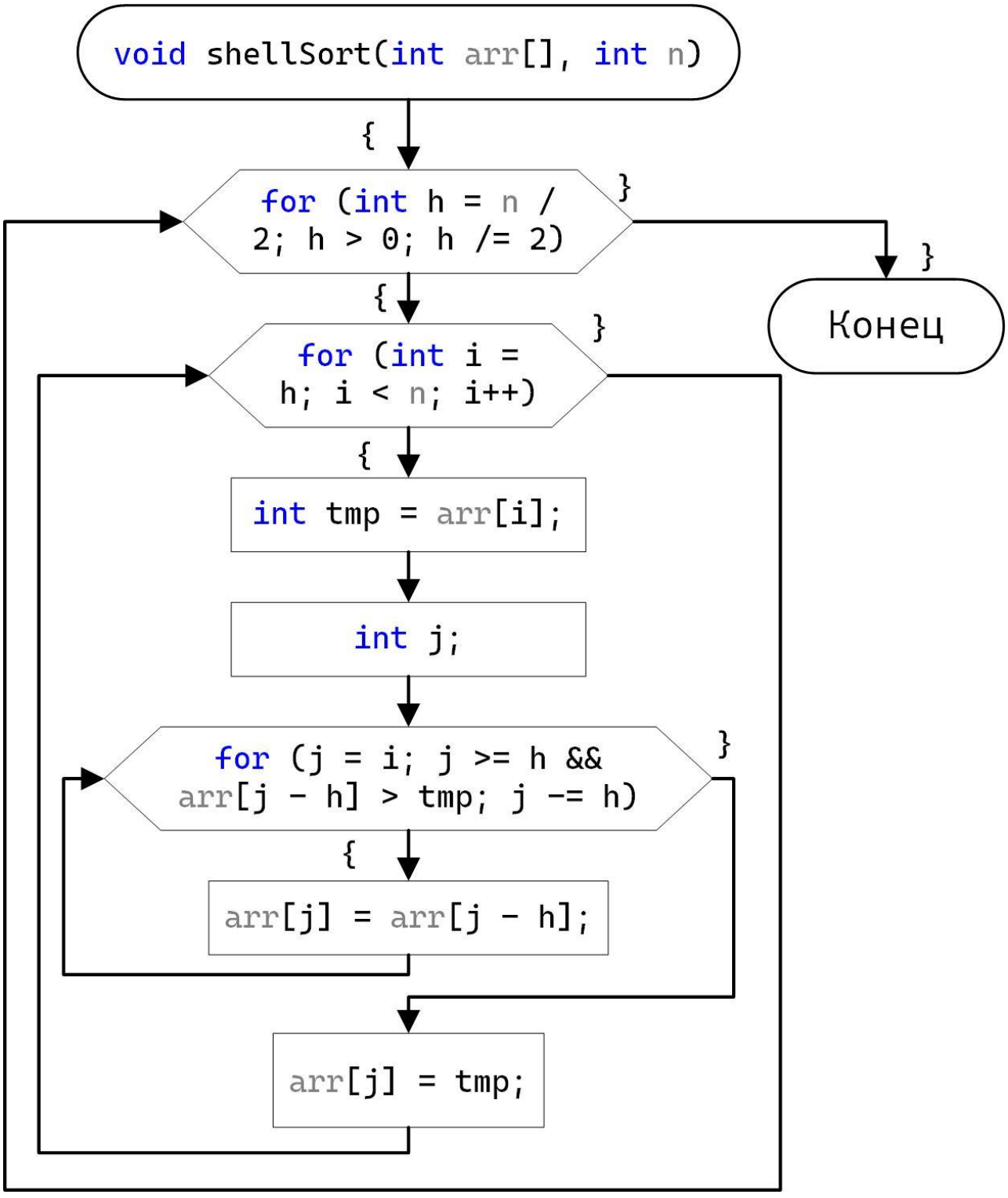

int partitionH(int a[], int low, int high)
{
int pivot = a[low];
int i = low - 1;
int j = high + 1;
True
False
while (1)
{
}
Конец
do
{
i++;
}
while (a[i] < pivot);
False
True
do
{
j--;
}
while (a[j] > pivot);
False
True
True
False
if (i >= j)
{
return j;
}
swap(a[i], a[j]);
}

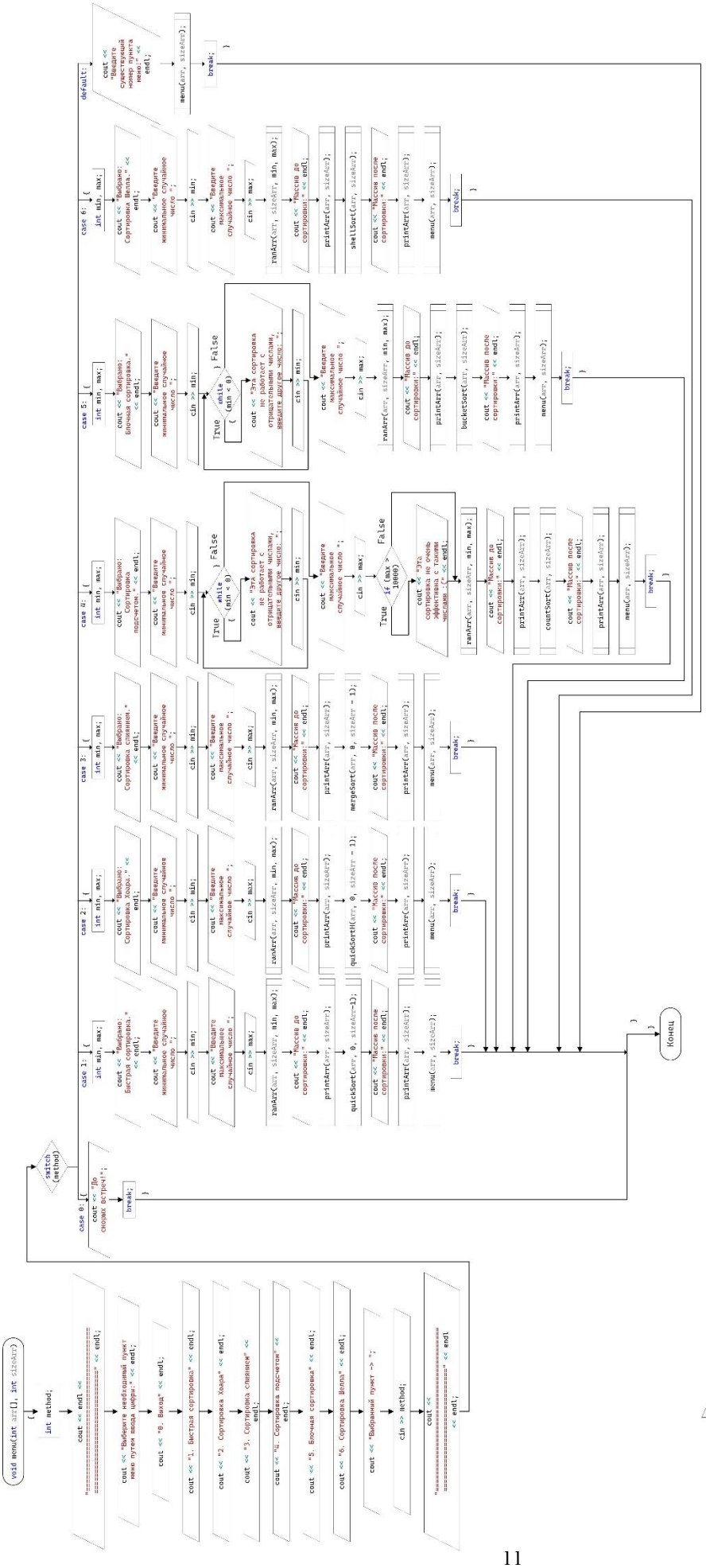

void quickSortH(int arr[], int low, int high)
{
True
False
if (low >= high)
{
return;
}
int pivot = partitionH(arr, low, high);
quickSortH(arr, low, pivot);
quickSortH(arr, pivot + 1, high);
}
Конец

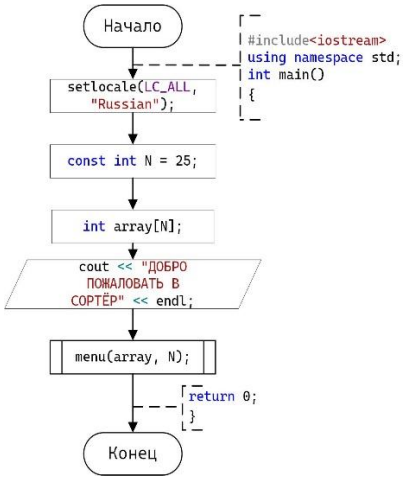

void merge(int arr[], int left, int mid, int right)
{
int leftRange = mid - left + 1;
int rightRange = right - mid;
int* leftArr = new int[leftRange];
int* rightArr = new int[rightRange];
for (int i = 0; i < leftRange; i++)
{
leftArr[i] = arr[left + i];
}
for (int i = 0; i < rightRange; i++)
{
rightArr[i] = arr[mid + 1 + i];
}
int leftIndex = 0;
int rightIndex = 0;
int mergedIndex = left;
True
False
while (leftIndex < leftRange && rightIndex < rightRange)
{
True
False
if (leftArr[leftIndex] <= rightArr[rightIndex])
{
arr[mergedIndex] = leftArr[leftIndex];
leftIndex++;
}
{
arr[mergedIndex] = rightArr[rightIndex];
rightIndex++;
}
mergedIndex++;
}
True
False
while (leftIndex < leftRange)
{
arr[mergedIndex] = leftArr[leftIndex];
leftIndex++;
mergedIndex++;
}
True
False
while (rightIndex < rightRange)
{
arr[mergedIndex] = rightArr[rightIndex];
rightIndex++;
mergedIndex++;
}
delete[] leftArr;
delete[] rightArr;
}
Конец
void mergeSort(int arr[], int left, int right)
{
True
False
if (left < right)
{
int mid = left + (right - left) / 2;
mergeSort(arr, left, mid);
mergeSort(arr, mid + 1, right);
merge(arr, left, mid, right);
}
}
Конец


void countSort(int arr[], int size)
{
int* output = new int[size];
int* count;
int max = arr[0];
for (int i = 1; i < size; i++)
{
True
False
if (arr[i] > max)
{ max = arr[i]; }
}
count = new int[max + 1];
for (int i = 0; i <= max; ++i)
{
count[i] = 0;
}
for (int i = 0; i < size; i++)
{
count[arr[i]]++;
}
for (int i = 1; i <= max; ++i)
{
count[i] += count[i - 1];
}
for (int i = size - 1; i >= 0; i--)
{
output[count[arr[i]] - 1] = arr[i];
count[arr[i]]--;
}
for (int i = 0; i < size; i++)
{
arr[i] = output[i];
}
delete[] count;
delete[] output;
}
Конец


void bucketSort(int arr[], int n)
{
const int BUCKET_NUM = 10;
int max = arr[0];
for (int i = 1; i < n; i++)
}
{
True
False
if (arr[i] > max)
{ max = arr[i]; }
max++;
int** buckets = new int* [BUCKET_NUM];
for (int i = 0; i < BUCKET_NUM; i++)
}
{
buckets[i] = new int[n];
int bucketSizes[BUCKET_NUM] = { 0 };
for (int i = 0; i < n; i++)
}
{
int bucketIndex = arr[i] * BUCKET_NUM / max;
buckets[bucketIndex][bucketSizes[bucketIndex]++] = arr[i];
for (int i = 0; i < BUCKET_NUM; i++)
}
{
for (int j = 1; j < bucketSizes[i]; j++)
}
{
int tmp = buckets[i][j];
int k = j - 1;
True
} False
while (k >= 0 && buckets[i][k] > tmp)
{
buckets[i][k + 1] = buckets[i][k];
k--;
buckets[i][k + 1] = tmp;
int idx = 0;
for (int i = 0; i < BUCKET_NUM; i++)
}
{
for (int j = 0; j < bucketSizes[i]; j++)
}
{
arr[idx++] = buckets[i][j];
delete[] buckets[i];
delete[] buckets;
}
Конец


void shellSort(int arr[], int n)
{
for (int h = n / 2; h > 0; h /= 2)
}
{
for (int i = h; i < n; i++)
}
{
int tmp = arr[i];
int j;
for (j = i; j >= h && arr[j - h] > tmp; j -= h)
}
{
arr[j] = arr[j - h];
arr[j] = tmp;
}
Конец


void menu(int arr[], int sizeArr)
int method;
switch (method)
case 0:
case 1:
case 2:
case 3:
case 4:
case 5:
case 6:
default:
Конец


Начало
#include<iostream>
using namespace std;
int main()
{
setlocale(LC_ALL, "Russian");
const int N = 25;
int array[N];
cout << "ДОБРО ПОЖАЛОВАТЬ В СОРТЁР" << endl;
menu(array, N);
return 0;
}
Конец

ПРАКТИЧЕСКАЯ ЧАСТЬ

4 Результат решения

4.1 Готовая программа

Исходя из подробных блок схем, составим программу на языке C++.

Таблица 1 – Готовая программа задачи

```
#include <iostream>
using namespace std;
void ranArr(int arr[], int sizeArr, int min, int max)
{
    srand(time(0) * time(0));
    for (int i = 0; i < sizeArr; i++)
    {
        arr[i] = rand() % (max - min + 1) + min;
    }
}
void printArr(int arr[], int sizeArr)
{
    for (int i = 0; i < sizeArr; i++)
    {
        cout << arr[i] << " ";
    }
    cout << endl;
}

int partitionQ(int arr[], int low, int high, int pivot)
{
    int PIndex = low;
    for (int i = low;i <= high;i++) {
        if (arr[i] <= pivot)
        {
            swap(arr[PIndex], arr[i]);
            PIndex++;
        }
    }
    PIndex--;
    return PIndex;
}
void quickSort(int arr[], int low, int high)
{
    if (low < high)
    {
        int pivot = arr[high];
        int PIndex = partitionQ(arr, low, high, pivot);
        quickSort(arr, low, PIndex - 1);
        quickSort(arr, PIndex + 1, high);
    }
}

int partitionH(int a[], int low, int high)
{
    int pivot = a[low];
    int i = low - 1;
    int j = high + 1;
    while (1)
    {
        do
        {
            i++;
        } while (a[i] < pivot);
```

```
        do
        {
            j--;
        } while (a[j] > pivot);

        if (i >= j) return j;
        swap(a[i], a[j]);
    }
}
void quickSortH(int arr[], int low, int high)
{
    if (low >= high) return;
    int pivot = partitionH(arr, low, high);
    quickSortH(arr, low, pivot);
    quickSortH(arr, pivot + 1, high);
}

void merge(int arr[], int left, int mid, int right)
{
    int leftRange = mid - left + 1;
    int rightRange = right - mid;
    int* leftArr = new int[leftRange];
    int* rightArr = new int[rightRange];
    for (int i = 0; i < leftRange; i++) leftArr[i] = arr[left + i];
    for (int i = 0; i < rightRange; i++) rightArr[i] = arr[mid + 1 + i];
    int leftIndex = 0;
    int rightIndex = 0;
    int mergedIndex = left;
    while (leftIndex < leftRange && rightIndex < rightRange)
    {
        if (leftArr[leftIndex] <= rightArr[rightIndex])
        {
            arr[mergedIndex] = leftArr[leftIndex];
            leftIndex++;
        }
        else
        {
            arr[mergedIndex] = rightArr[rightIndex];
            rightIndex++;
        }
        mergedIndex++;
    }
    while (leftIndex < leftRange)
    {
        arr[mergedIndex] = leftArr[leftIndex];
        leftIndex++;
        mergedIndex++;
    }
    while (rightIndex < rightRange)
    {
        arr[mergedIndex] = rightArr[rightIndex];
        rightIndex++;
        mergedIndex++;
    }
    delete[] leftArr;
    delete[] rightArr;
}
void mergeSort(int arr[], int left, int right)
{
    if (left < right)
    {
        int mid = left + (right - left) / 2;
        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);
        merge(arr, left, mid, right);
```

```
    }
}

void countSort(int arr[], int size)
{
    int* output = new int[size];
    int* count;
    int max = arr[0];
    for (int i = 1; i < size; i++)
    {
        if (arr[i] > max) max = arr[i];
    }
    count = new int[max + 1];
    for (int i = 0; i <= max; ++i) count[i] = 0;
    for (int i = 0; i < size; i++) count[arr[i]]++;
    for (int i = 1; i <= max; ++i)
    {
        count[i] += count[i - 1];
    }
    for (int i = size - 1; i >= 0; i--)
    {
        output[count[arr[i]] - 1] = arr[i];
        count[arr[i]]--;
    }
    for (int i = 0; i < size; i++) arr[i] = output[i];
    delete[] count;
    delete[] output;
}

void bucketSort(int arr[], int n)
{
    const int BUCKET_NUM = 10;
    int max = arr[0];
    for (int i = 1; i < n; i++)
    {
        if (arr[i] > max) max = arr[i];
    }
    max++;
    int** buckets = new int* [BUCKET_NUM];
    for (int i = 0; i < BUCKET_NUM; i++)
    {
        buckets[i] = new int[n];
    }

    int bucketSizes[BUCKET_NUM] = { 0 };
    for (int i = 0; i < n; i++)
    {
        int bucketIndex = arr[i] * BUCKET_NUM / max;
        buckets[bucketIndex][bucketSizes[bucketIndex]++] = arr[i];
    }
    for (int i = 0; i < BUCKET_NUM; i++)
    {
        for (int j = 1; j < bucketSizes[i]; j++)
        {
            int tmp = buckets[i][j];
            int k = j - 1;
            while (k >= 0 && buckets[i][k] > tmp)
            {
                buckets[i][k + 1] = buckets[i][k];
                k--;
            }
            buckets[i][k + 1] = tmp;
        }
    }
    int idx = 0;
```

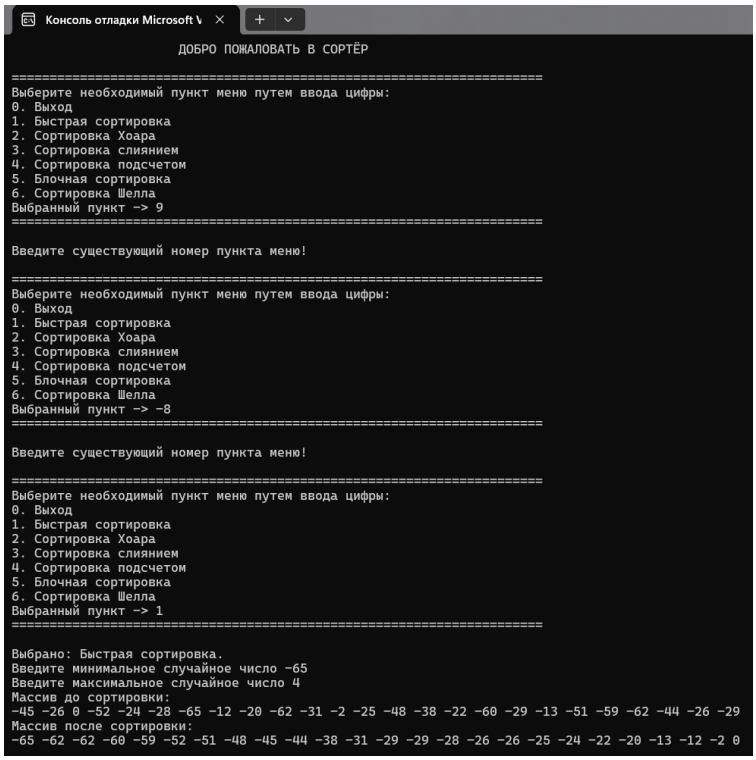
```
    for (int i = 0; i < BUCKET_NUM; i++)
    {
        for (int j = 0; j < bucketSizes[i]; j++)
        {
            arr[idx++] = buckets[i][j];
        }
        delete[] buckets[i];
    }
    delete[] buckets;
}

void shellSort(int arr[], int n)
{
    for (int h = n / 2; h > 0; h /= 2)
    {
        for (int i = h; i < n; i++)
        {
            int tmp = arr[i];
            int j;
            for (j = i; j >= h && arr[j - h] > tmp; j -= h)
            {
                arr[j] = arr[j - h];
            }
            arr[j] = tmp;
        }
    }
}

void menu(int arr[], int sizeArr)
{
    int method;
    cout << endl <<
"======================================================================" << endl;
    cout << "Выберите необходимый пункт меню путем ввода цифры:" << endl;
    cout << "0. Выход" << endl;
    cout << "1. Быстрая сортировка" << endl;
    cout << "2. Сортировка Хоара" << endl;
    cout << "3. Сортировка слиянием" << endl;
    cout << "4. Сортировка подсчетом" << endl;
    cout << "5. Блочная сортировка" << endl;
    cout << "6. Сортировка Шелла" << endl;
    cout << "Выбранный пункт -> ";
    cin >> method;
    cout << "======================================================================"
<< endl << endl;
    switch (method)
    {
    case 0:
    {
        cout << "До скорых встреч!";
        break;
    }
    case 1:
    {
        int min, max;
        cout << "Выбрано: Быстрая сортировка." << endl;
        cout << "Введите минимальное случайное число ";
        cin >> min;
        cout << "Введите максимальное случайное число ";
        cin >> max;
        ranArr(arr, sizeArr, min, max);
        cout << "Массив до сортировки:" << endl;
        printArr(arr, sizeArr);
        quickSort(arr, 0, sizeArr-1);
        cout << "Массив после сортировки:" << endl;
```

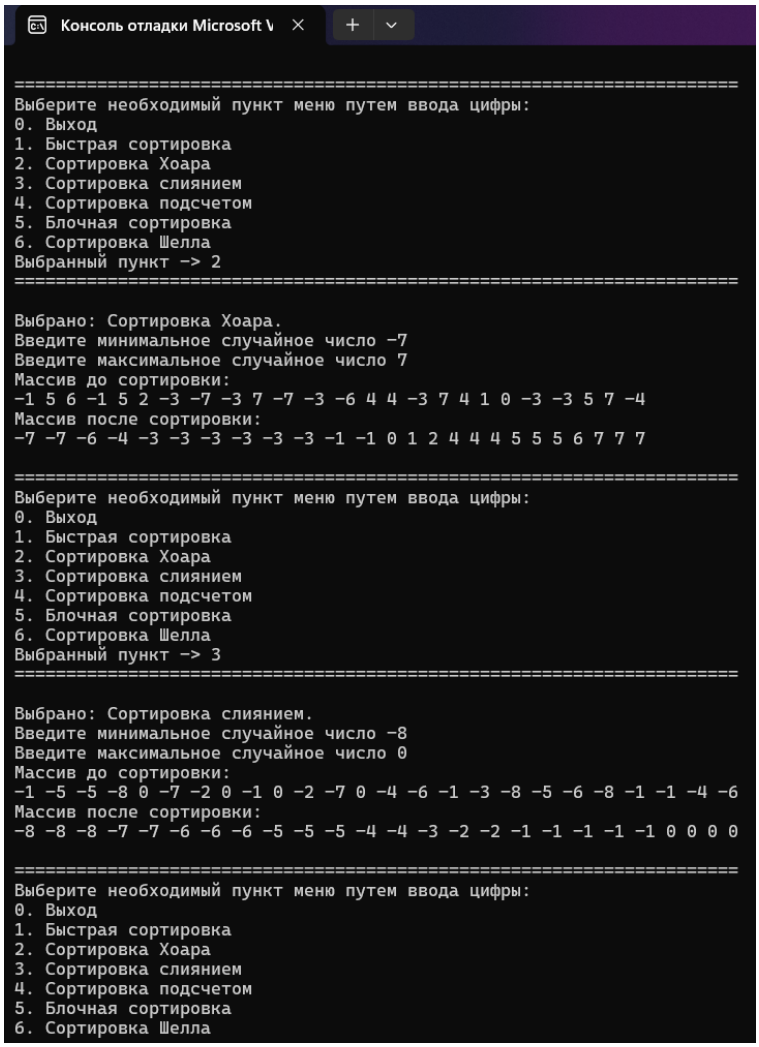
```
        printArr(arr, sizeArr);
        menu(arr, sizeArr);
        break;
    }
    case 2:
    {
        int min, max;
        cout << "Выбрано: Сортировка Хоара." << endl;
        cout << "Введите минимальное случайное число ";
        cin >> min;
        cout << "Введите максимальное случайное число ";
        cin >> max;
        ranArr(arr, sizeArr, min, max);
        cout << "Массив до сортировки:" << endl;
        printArr(arr, sizeArr);
        quickSortH(arr, 0, sizeArr - 1);
        cout << "Массив после сортировки:" << endl;
        printArr(arr, sizeArr);
        menu(arr, sizeArr);
        break;
    }
    case 3:
    {
        int min, max;
        cout << "Выбрано: Сортировка слиянием." << endl;
        cout << "Введите минимальное случайное число ";
        cin >> min;
        cout << "Введите максимальное случайное число ";
        cin >> max;
        ranArr(arr, sizeArr, min, max);
        cout << "Массив до сортировки:" << endl;
        printArr(arr, sizeArr);
        mergeSort(arr, 0, sizeArr - 1);
        cout << "Массив после сортировки:" << endl;
        printArr(arr, sizeArr);
        menu(arr, sizeArr);
        break;
    }
    case 4:
    {
        int min, max;
        cout << "Выбрано: Сортировка подсчетом." << endl;
        cout << "Введите минимальное случайное число ";
        cin >> min;
        while (min < 0)
        {
            cout << "Эта сортировка не работает с отрицательными числами, введите
другое число: ";
            cin >> min;
        }
        cout << "Введите максимальное случайное число ";
        cin >> max;
        if (max > 10000) cout << "Эта сортировка не очень эффективна с такими
числами :(" << endl;
        ranArr(arr, sizeArr, min, max);
        cout << "Массив до сортировки:" << endl;
        printArr(arr, sizeArr);
        countSort(arr, sizeArr);
        cout << "Массив после сортировки:" << endl;
        printArr(arr, sizeArr);
        menu(arr, sizeArr);
        break;
    }
    case 5:
    {
```

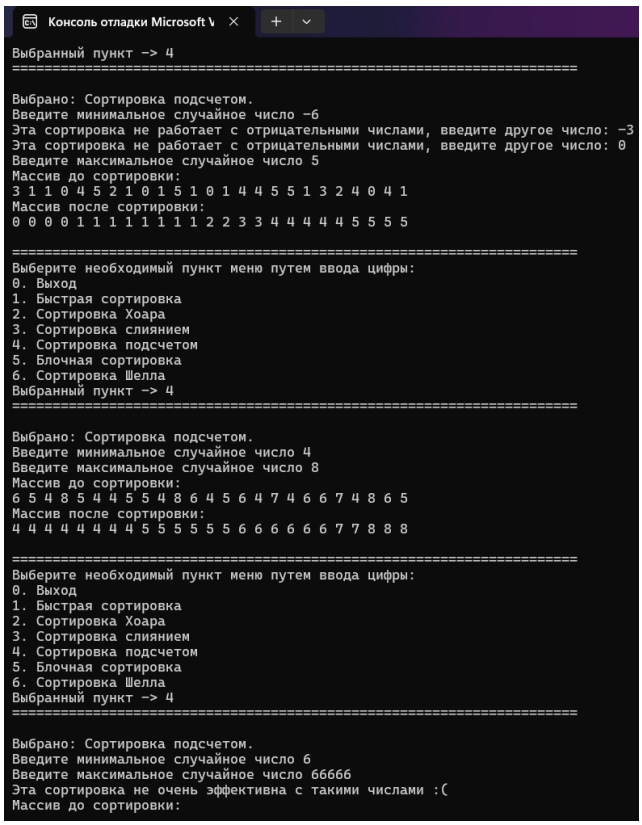
```
        int min, max;
        cout << "Выбрано: Блочная сортировка." << endl;
        cout << "Введите минимальное случайное число ";
        cin >> min;
        while (min < 0)
        {
            cout << "Эта сортировка не работает с отрицательными числами, введите
другое число: ";
            cin >> min;
        }
        cout << "Введите максимальное случайное число ";
        cin >> max;
        ranArr(arr, sizeArr, min, max);
        cout << "Массив до сортировки:" << endl;
        printArr(arr, sizeArr);
        bucketSort(arr, sizeArr);
        cout << "Массив после сортировки:" << endl;
        printArr(arr, sizeArr);
        menu(arr, sizeArr);
        break;
    }
    case 6:
    {
        int min, max;
        cout << "Выбрано: Сортировка Шелла." << endl;
        cout << "Введите минимальное случайное число ";
        cin >> min;
        cout << "Введите максимальное случайное число ";
        cin >> max;
        ranArr(arr, sizeArr, min, max);
        cout << "Массив до сортировки:" << endl;
        printArr(arr, sizeArr);
        shellSort(arr, sizeArr);
        cout << "Массив после сортировки:" << endl;
        printArr(arr, sizeArr);
        menu(arr, sizeArr);
        break;
    }
    default:
    {
        cout << "Введите существующий номер пункта меню!" << endl;
        menu(arr, sizeArr);
        break;
    }
    }
}
int main()
{
    setlocale(LC_ALL, "Russian");
    const int N = 25; // если изменить, работает и с другими размерами
    int array[N];
    cout << "                        ДОБРО ПОЖАЛОВАТЬ В СОРТЁР" << endl;
    menu(array, N);
    return 0;
}
```

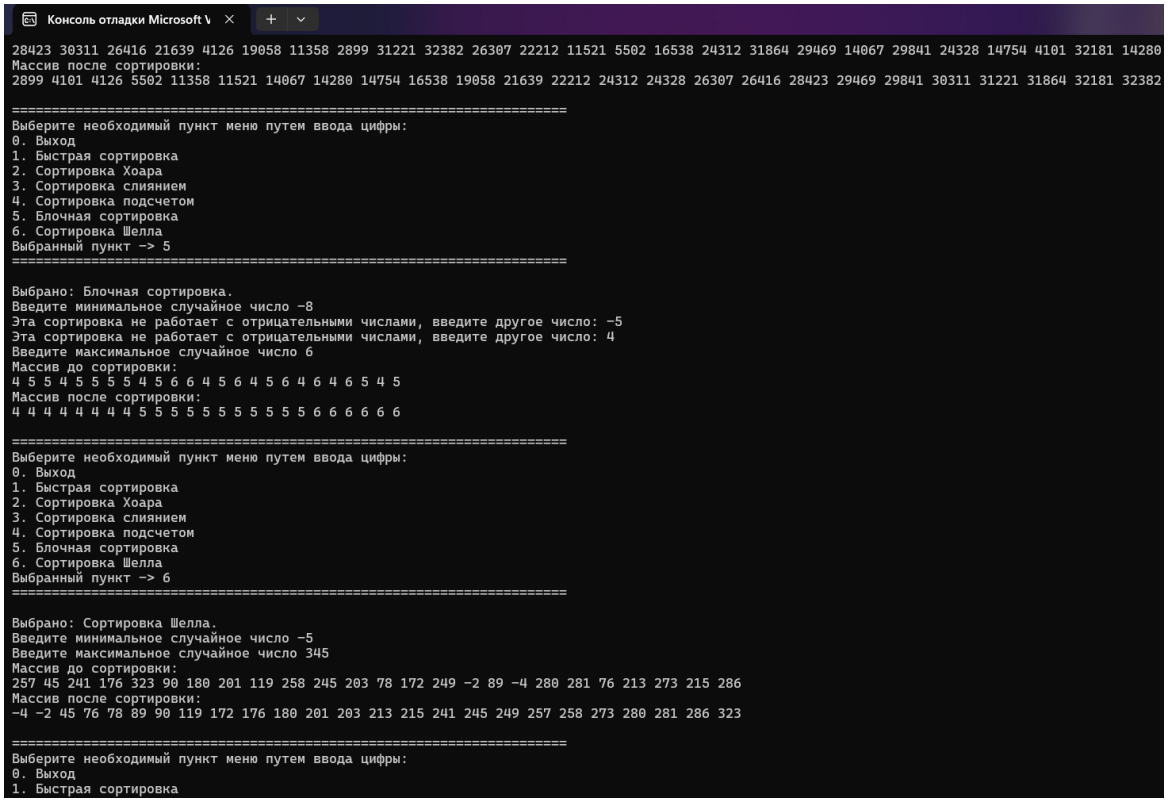
4.2 Скриншоты

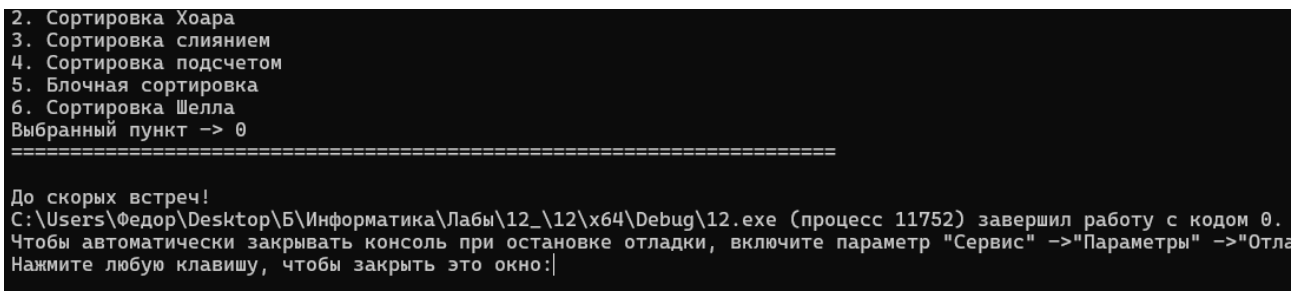
Запустим программу, чтобы убедится, что она работает верно.






Консоль отладки Microsoft \
Выбранный пункт -> 4
Выбрано: Сортировка подсчетом.
Введите минимальное случайное число -6
Эта сортировка не работает с отрицательными числами, введите другое число: -3
Эта сортировка не работает с отрицательными числами, введите другое число: 0
Введите максимальное случайное число 5
Массив до сортировки:
3 1 1 0 4 5 2 1 0 1 5 1 0 1 4 4 5 5 1 3 2 4 0 4 1
Массив после сортировки:
0 0 0 0 1 1 1 1 1 1 1 1 2 2 3 3 4 4 4 4 4 4 5 5 5 5
Выберите необходимый пункт меню путем ввода цифры:
0. Выход
1. Быстрая сортировка
2. Сортировка Хоара
3. Сортировка слиянием
4. Сортировка подсчетом
5. Блочная сортировка
6. Сортировка Шелла
Выбранный пункт -> 4
Выбрано: Сортировка подсчетом.
Введите минимальное случайное число 4
Введите максимальное случайное число 8
Массив до сортировки:
6 5 4 8 5 4 4 5 5 4 8 6 4 5 6 4 7 4 6 6 7 4 8 6 5
Массив после сортировки:
4 4 4 4 4 4 4 4 5 5 5 5 5 5 6 6 6 6 6 6 7 7 8 8 8
Выберите необходимый пункт меню путем ввода цифры:
0. Выход
1. Быстрая сортировка
2. Сортировка Хоара
3. Сортировка слиянием
4. Сортировка подсчетом
5. Блочная сортировка
6. Сортировка Шелла
Выбранный пункт -> 4
Выбрано: Сортировка подсчетом.
Введите минимальное случайное число 6
Введите максимальное случайное число 66666
Эта сортировка не очень эффективна с такими числами :(
Массив до сортировки:


Консоль отладки Microsoft \
28423 30311 26416 21639 4126 19058 11358 2899 31221 32382 26307 22212 11521 5502 16538 24312 31864 29469 14067 29841 24328 14754 4101 32181 14280
Массив после сортировки:
2899 4101 4126 5502 11358 11521 14067 14280 14754 16538 19058 21639 22212 24312 24328 26307 26416 28423 29469 29841 30311 31221 31864 32181 32382
Выберите необходимый пункт меню путем ввода цифры:
0. Выход
1. Быстрая сортировка
2. Сортировка Хоара
3. Сортировка слиянием
4. Сортировка подсчетом
5. Блочная сортировка
6. Сортировка Шелла
Выбранный пункт -> 5
Выбрано: Блочная сортировка.
Введите минимальное случайное число -8
Эта сортировка не работает с отрицательными числами, введите другое число: -5
Эта сортировка не работает с отрицательными числами, введите другое число: 4
Введите максимальное случайное число 6
Массив до сортировки:
4 5 5 4 5 5 5 5 4 5 6 6 4 5 6 4 5 6 4 6 4 6 5 4 5
Массив после сортировки:
4 4 4 4 4 4 4 4 5 5 5 5 5 5 5 5 5 5 5 6 6 6 6 6 6
Выберите необходимый пункт меню путем ввода цифры:
0. Выход
1. Быстрая сортировка
2. Сортировка Хоара
3. Сортировка слиянием
4. Сортировка подсчетом
5. Блочная сортировка
6. Сортировка Шелла
Выбранный пункт -> 6
Выбрано: Сортировка Шелла.
Введите минимальное случайное число -5
Введите максимальное случайное число 345
Массив до сортировки:
257 45 241 176 323 90 180 201 119 258 245 203 78 172 249 -2 89 -4 280 281 76 213 273 215 286
Массив после сортировки:
-4 -2 45 76 78 89 90 119 172 176 180 201 203 213 215 241 245 249 257 258 273 280 281 286 323
Выберите необходимый пункт меню путем ввода цифры:
0. Выход
1. Быстрая сортировка


2. Сортировка Хоара
3. Сортировка слиянием
4. Сортировка подсчетом
5. Блочная сортировка
6. Сортировка Шелла
Выбранный пункт -> 0
До скорых встреч!
C:\Users\Федор\Desktop\Б\Информатика\Лабы\12_\12\x64\Debug\12.exe (процесс 11752) завершил работу с кодом 0.
Чтобы автоматически закрывать консоль при остановке отладки, включите параметр "Сервис" ->"Параметры" ->"Отла
Нажмите любую клавишу, чтобы закрыть это окно:

ВЫВОД

В итоге этой работы была составлена программа с использованием шести сложных сортировок. Сделан вывод о полезности и различиях этих сортировок. Выявлены плюсы и минусы сложных сортировок перед простыми. В ходе работы были получены навыки работы с оператором множественного выбора и большим количеством функций.

Проведенная лабораторная работа была опубликована в общий доступ по адресу: https://github.com/Fedor0000/TheUltimateFolder/tree/main/Sem_2/Labs/12