**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра информационных систем**

отчет

**по практической работе №2**

**по дисциплине «Программирование»**

Тема: **ОДНОМЕРНЫЕ СТАТИЧЕСКИЕ МАССИВЫ**

|  |  |  |
| --- | --- | --- |
| Студентка гр. 4373 |  | Репков А.А |
| Преподаватель |  | Глущенко А. Г. |

Санкт-Петербург

2024

**Цель работы.**

Изучение работы с массивами и сортировок.

**Основные теоретические положения.**

При использовании простых переменных каждой области памяти для хранения данных соответствует свое имя. Если с группой величин одинакового типа требуется выполнить однообразные действия, им дают одно имя, а различают по порядковому номеру (индексу). Это дает возможность компактно записать множество операций с использованием циклов.

Массив представляет собой индексированную последовательность однотипных элементов с заранее определенным количеством элементов. Наглядно одномерный массив можно представить, как набор пронумерованных ячеек, в каждой из которых содержится определенное значение.

Все массивы можно разделить на две группы: одномерные и многомерные. Описание массива в программе отличается от объявления обычной переменной наличием размерности массива, которая задается в квадратных скобках после имени.

Элементы массива нумеруются с нуля. При описании массива используются те же модификаторы (класс памяти, const и инициализатор), что и для простых переменных.

Аналогом одномерного массива из математики может служить последовательность некоторых элементов с одним индексом: ai*ai*​ при  i = 0, 1, 2, … n – одномерный вектор. Каждый элемент такой последовательности представляет собой некоторое значение определенного типа данных.

Объявление в программах одномерных массивов выполняется в соответствии со следующим правилом:

<Базовый тип элементов> <Идентификатор массива> [<Количество элементов>]

Значения индексов элементов массивов всегда начинается с 0. Поэтому максимальное значение индекса элемента в массиве всегда на единицу меньше количества элементов в массиве.

При обращении к конкретному элементу массива этот элемент можно рассматривать как обычную переменную, тип которой соответствует базовому типу элементов массива, и осуществлять со значением этого элемента любые операции, которые характерны для базового типа. Например, поскольку базовым типом массива A1 является тип данных int, с любым элементом этого массива можно выполнять любые операции, которые можно выполнять над значениями типа int.

Пузырьковая сортировка массива (bubble sort)

Сортировка – процесс размещения элементов заданного множества объектов в определенном порядке. Когда элементы отсортированы, их проще найти, производить с ними различные операции. Сортировка напрямую влияет на скорость алгоритма, в котором нужно обратиться к определенному элементу массива.

Простейшая из сортировок – сортировка обменом (пузырьковая сортировка). Вся суть метода заключается в попарном сравнении элементов и последующем обмене. Таким образом, если следующий элемент меньше текущего, то они меняются местами, максимальный элемент массива постепенно смещается в конец массива, а минимальный – в начало. Один полный проход по массиву может гарантировать, что в конце массива находится максимальный элемент.

Затем процесс повторяется до тех пор, пока вся последовательность не будет упорядочена. Важно заметить, что после первого прохода по массиву, уже имеется один упорядоченный элемент, он стоит на своем месте, и менять его не надо. Таким образом на следующем шаге будут сравниваться *N*-1 элемент.

Очевидно, что хуже всего алгоритм будет работать, когда на вход подается массив, отсортированный в обратную сторону (от большего к меньшу). Быстрее же всего алгоритм работает с уже отсортированным массивом.

Но стандартный алгоритм пузырьковой сортировки предполагает полный циклический проход по массиву. Если изначально подается упорядоченная последовательность, то работа алгоритма все равно продолжиться. Исправить это можно, добавив условие проверки: если на текущей итерации ни один элемент не изменил свой индекс, то работа алгоритма прекращается.

Шейкер-сортировка массива (shaker sort)

Shaker sort – модификация пузырьковой сортировки. Принцип работы этой сортировки аналогичен bubble sort: попарное сравнение элементов и последующий обмен местами. Но имеется существенное отличие. Как только максимальный элемент становится на свое место, алгоритм не начинает новую итерацию с первого элемента, а запускает сортировку в обратную сторону. Алгоритм гарантирует, что после выполнения первой итерации, минимальный и максимальный элемент будут в начале и конце массива соответственно.

Затем процесс повторяется до тех пор, пока массив не будет отсортирован. За счет того, что сортировка работает в обе стороны, массив сортируется на порядок быстрее. Очевидным примером этого был бы случай, когда в начале массива стоит максимальный элемент, а в конце массива – минимальный. Shaker sort справится с этим за 1 итерацию, при условии, что другие элементы стоят на правильном месте.

Сортировка массива расчёской (comb sort)

Очевидный недостаток bubble и shaker sort заключается в том, что элементы переставляются максимум на одну позицию.

Comb sort (сортировка расческой) – ещё одна модификация сортировки пузырьком. Алгоритм был разработан специально для случаев, когда минимальные элементы стоят слишком далеко, или максимальные – слишком близко к началу массива. В сортировке расческой переставляются элементы, стоящие на расстоянии.

Оптимально изначально взять расстояние равным длине массива , а далее уменьшать его на определенный коэффициент, который примерно равен 1.247. Когда расстояние станет равно 1, выполняется обычная сортировка пузырьком.

Сортировка массива вставками (insert sort)

Сортировка вставками (insert sort) – алгоритм сортировки, в котором элементы массива просматриваются по одному, и каждый новый элемент размещается в подходящее место среди ранее упорядоченных элементов.

Общая суть сортировки вставками такова:

1)    Перебираются элементы в неотсортированной части массива.

2)    Каждый элемент вставляется в отсортированную часть массива на то место, где он должен находится.

Сортировка вставками делить массив на 2 части – отсортированную и неотсортированную. С каждым новым элементом отсортированная часть будет увеличиваться, а неотсортированная уменьшаться. Причем найти нужное место для очередного элемента в отсортированном массиве достаточно легко.

Рассмотрим самый простой способ (рис. 3.5). Необходимо пройти массив слева направо и обработать каждый элемент. Слева будет наращиваться отсортированная часть массива, а справа – уменьшаться неотсортированная. В отсортированной части массива ищется точка вставки для очередного элемента. Сам элемент отправляется в буфер, что освобождает место в массиве и позволяет сдвинуть элементы и освободить точку вставки.

Быстрая сортировка массива (quick sort)

Быстрая сортировка (quick sort) – одна из самых быстрых сортировок. Эта сортировка по сути является существенно улучшенной версией алгоритма пузырьковой сортировки.

Общая идея алгоритма состоит в том, что сначала выбирается из массива элемент, который называется опорным. От выбора опорного элемента не зависит корректность алгоритма, но в отдельных случаях может сильно зависеть его эффективность. Затем необходимо сравнить все остальные элементы с опорным и переставить их в массиве так, чтобы разбить массив на три непрерывных отрезка, следующие друг за другом: меньше опорного, раны опорному и больше опорного. Для меньших и больших значений необходимо выполнить рекурсивно ту же последовательность операций, если длина отрезка больше единицы.

Бинарный поиск работает только в топ случае, если массив отсортирован. Например, если бы искомое минимальное значение стояло не на своем положенном месте, а на месте максимального элемента, то мы бы откинули его на первой же итерации. Сам алгоритм имеет вид:

1)    Определение значения в середине массива (или иной структуры данных). Полученное значение сравнивается с ключом (значением, которое необходимо найти).

2)    Если ключ меньше значения середины, то необходимо осуществлять поиск в первой половине элементов, иначе – во второй.

3)    Поиск сводится к тому, что вновь определяется значение серединного элемента в выбранной половине и сравнивается с ключом.

4)    Процесс продолжается до тех пор, пока не будет определен элемент, равный значению ключа или не станет пустым интервал для поиска.

**Постановка задачи.**

Необходимо написать программу, которая:

1)    Создает целочисленный массив размерности *N* = 100. Элементы массивы должны принимать случайное значение в диапазоне от -99 до 99.

2)    Отсортировать заданный в пункте 1 массив […] сортировкой (от меньшего к большему). Определить время, затраченное на сортировку, используя библиотеку chrono.

3)    Найти максимальный и минимальный элемент массива. Подсчитайте время поиска этих элементов в отсортированном массиве и неотсортированном, используя библиотеку chrono.

4)    Выводит среднее значение (если необходимо, число нужно округлить) максимального и минимального значения в отсортированном и неотсортированном. Выводит индексы всех элементов, которые равны этому значению, и их количество. Подсчитайте время поиска.

5)    Выводит количество элементов в отсортированном массиве, которые меньше числа *a*, которое инициализируется пользователем.

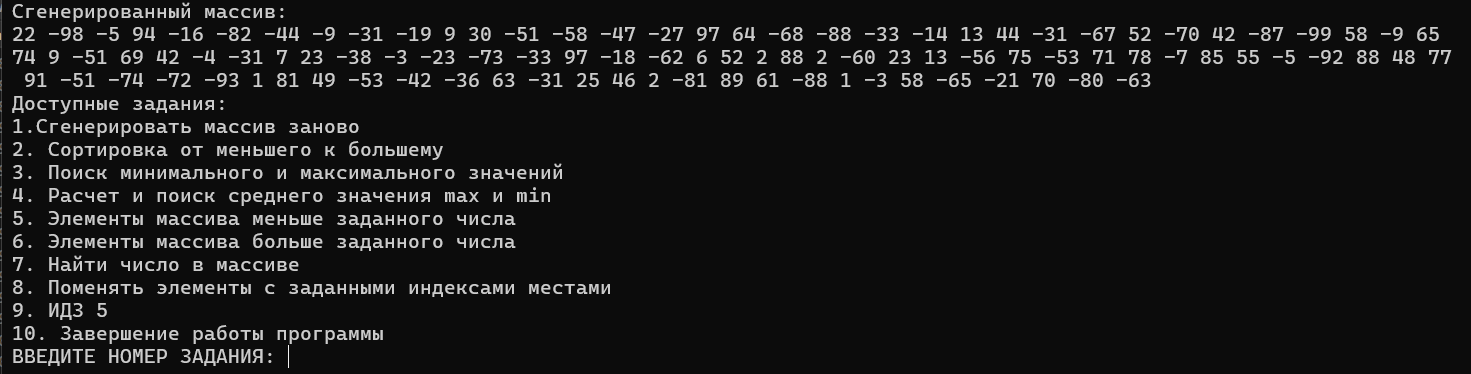
6)    Выводит количество элементов в отсортированном массиве, которые больше числа *b*, которое инициализируется пользователем.

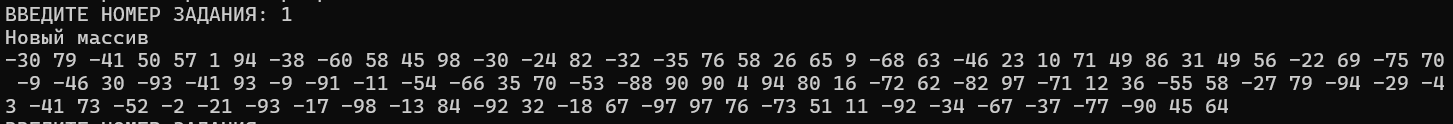
7)    Выводит информацию о том, есть ли введенное пользователем число в отсортированном массиве. Реализуйте алгоритм бинарного поиска. Сравните скорость его работы с обычным перебором. (\*)

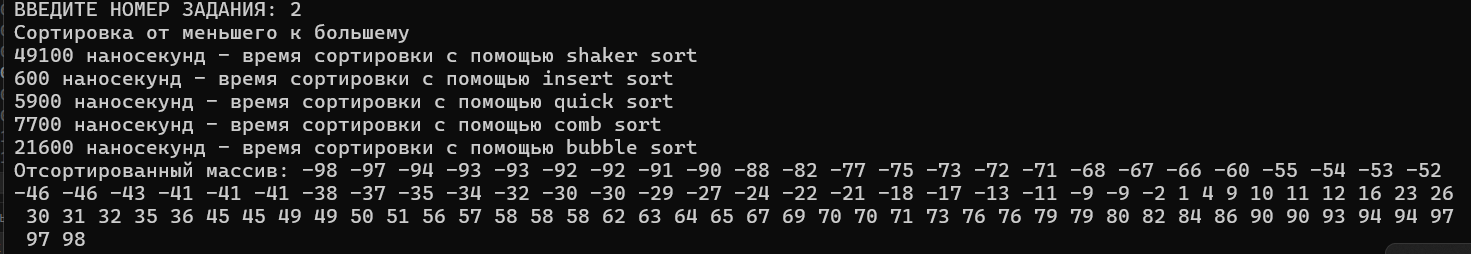
8)     Меняет местами элементы массива, индексы которых вводит пользователь. Выведите скорость обмена, используя библиотеку chrono.

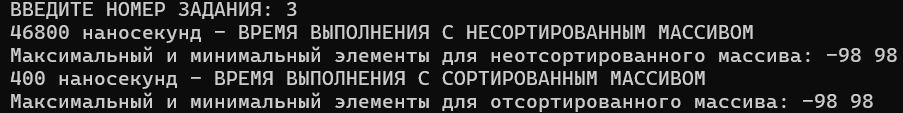
**Выполнение работы.**

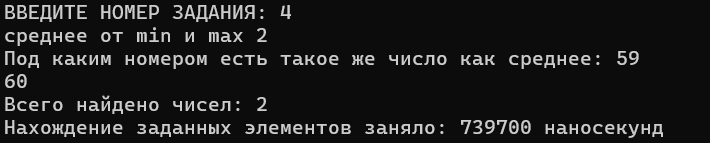
Код программы представлен в приложении А.

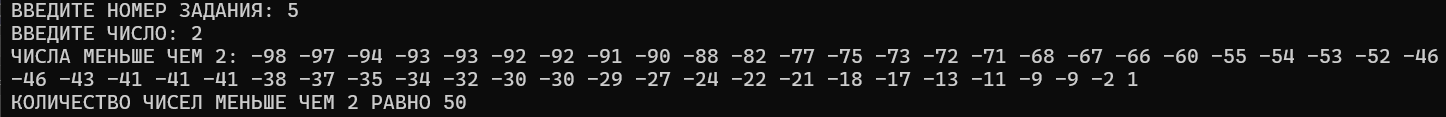


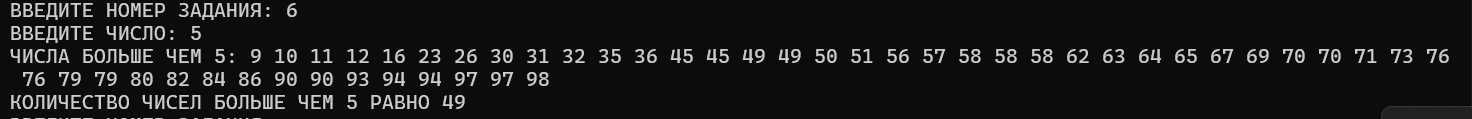


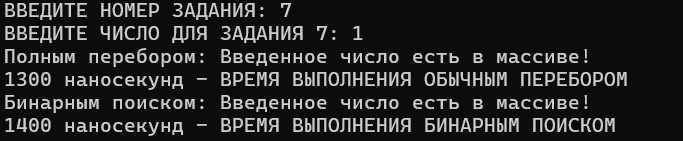


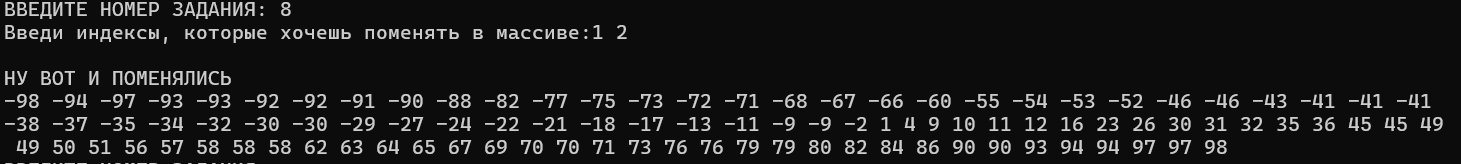


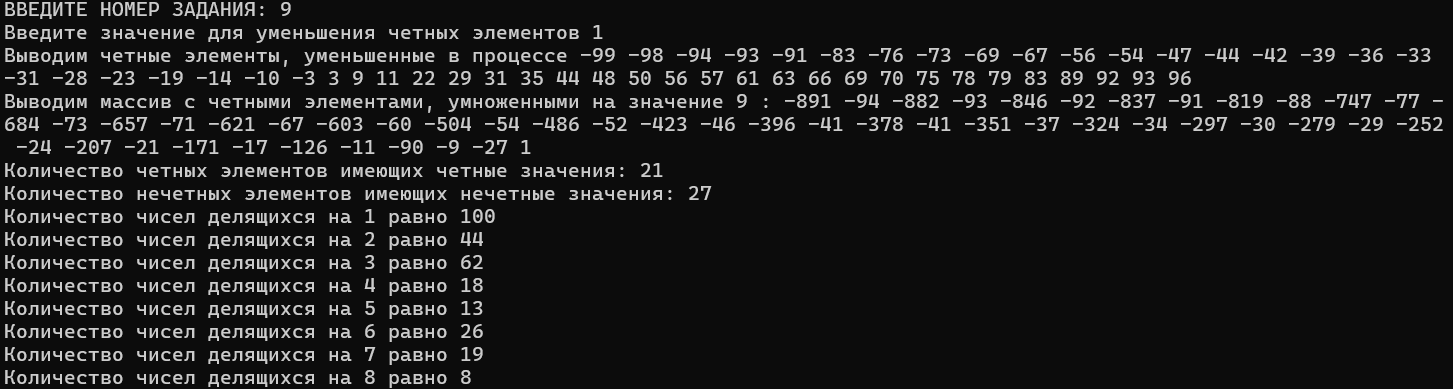














**Выводы.**

Я узнал, как создавать массивы, сортировать их, находить определенные элементы массивов, выводить и менять значения в массиве.

Приложение А

рабочий код

#include <iostream>

#include <chrono>

#include <cstdlib> // Для функции rand() и srand()

#include <ctime>

#include <vector>

using namespace std;

using namespace chrono;

void bubbleSort(int arr[], int n) {

setlocale(LC\_ALL, "RU");

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

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

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

swap(arr[j], arr[j + 1]);

}

}

}

cout << "\n";

}

void findMinMax(int arr[], int n, int& min, int& max) {

min = arr[0];

max = arr[0];

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

if (arr[i] < min) {

min = arr[i];

}

if (arr[i] > max) {

max = arr[i];

}

}

cout << "\n";

}

void findIndices(int arr[], int n, int value, vector<int>& indices) {

indices.clear(); // Очистка вектора перед использованием

for (int i = 0; i < n; ++i) {

if (arr[i] == value) {

indices.push\_back(i); // Добавление индекса в вектор

}

}

cout << "\n";

}

int countLessThan(int arr[], int n, int a) {

int count = 0;

for (int i = 0; i < n; ++i) {

if (arr[i] < a) {

count++;

}

}

return count;

}

int countBiggestThan(int arr[], int n, int b) {

int count = 0;

for (int i = 0; i < n; ++i) {

if (arr[i] > b) {

count++;

}

}

return count;

}

bool binarySearch(int arr[], int n, int value) {

int left = 0;

int right = n - 1;

while (left <= right) {

int mid = left + (right - left) / 2;

if (arr[mid] == value)

return true;

else if (arr[mid] < value)

left = mid + 1;

else

right = mid - 1;

}

return false;

}

bool linearSearch(int arr[], int n, int value) {

for (int i = 0; i < n; ++i) {

if (arr[i] == value)

return true;

}

return false;

}

void swapElements(int arr[], int index1, int index2) {

swap(arr[index1], arr[index2]);

}

int main() {

setlocale(LC\_ALL, "RU");

const int N = 100;

int arr[N];

srand(static\_cast<unsigned int>(time(0)));

for (int i = 0; i < N; ++i) {

arr[i] = rand() % 199 - 99;

}

cout << "Элементы массива до сортировки:" << endl;

for (int i = 0; i < N; ++i) {

cout << arr[i] << " ";

}

cout << endl;

cout << "\n";

int value\_to\_decrease;

cout << "Введите значение для уменьшения четных элементов: ";

cin >> value\_to\_decrease;

int end\_val = 9;

int start\_val = 1;

int random\_value = (rand() % (end\_val - start\_val + 1) + start\_val);

int count\_even = 0;

int count\_odd = 0;

for (int i = 0; i < N; i++) {

if (i % 2 == 0) {

arr[i] -= value\_to\_decrease;

if (arr[i] % 2 == 0) {

count\_even += 1;

}

}

else if (i % 2 != 0 && arr[i] % 2 != 0) {

count\_odd += 1;

}

}

cout << "\n";

cout << "Массив после вычитания значения " << value\_to\_decrease << ":" << endl;

for (int i = 0; i < N; ++i) {

cout << arr[i] << " ";

}

for (int i = 0; i < N; i++) {

if (i % 2 == 0) {

arr[i] \*= random\_value;

}

}

cout << "\n" << "\n";

cout << "Массив после умножения четных элементов на значение " << random\_value << ":" << endl;

for (int i = 0; i < N; ++i) {

cout << arr[i] << " ";

}

cout << "\n" << "\n";

cout << "Количество четных элементов: " << count\_even << endl;

cout << "Количество нечетных элементов: " << count\_odd << endl;

for (int i = 1; i < 10; i++) {

int counter\_del = 0;

for (int j = 0; j < N; j++) {

if (arr[j] % i == 0) {

counter\_del += 1;

}

};

cout << " Кол-во чисел делящихся на " << i << " равно " << counter\_del << endl;

};

//3 zad

// Время поиска минимального и максимального элемента в неотсортированном массиве

auto start = chrono::high\_resolution\_clock::now();

int minUnsorted, maxUnsorted;

findMinMax(arr, N, minUnsorted, maxUnsorted);

auto end = chrono::high\_resolution\_clock::now();

chrono::duration<double, std::nano> elapsedUnsorted = end - start;

cout << "Минимальный элемент (неотсортированный): " << minUnsorted << endl;

cout << "Максимальный элемент (неотсортированный): " << maxUnsorted << endl;

cout << "Время поиска минимального и максимального элемента (неотсортированный): "

<< elapsedUnsorted.count() << " наносекунд." << endl;

bubbleSort(arr, N);// Сортировка баблсортом

cout << "Элементы массива после сортировки:" << endl;

for (int i = 0; i < N; ++i) {

cout << arr[i] << " ";

}

cout << endl;

cout << "\n";

start = chrono::high\_resolution\_clock::now();

int minSorted, maxSorted;

findMinMax(arr, N, minSorted, maxSorted);

end = chrono::high\_resolution\_clock::now();

chrono::duration<double, std::nano> elapsedSorted = end - start;

cout << "Минимальный элемент (отсортированный): " << minSorted << endl;

cout << "Максимальный элемент (отсортированный): " << maxSorted << endl;

cout << "Время поиска (отсортированный): " << elapsedSorted.count() << " наносекунд" << endl;

//4 zad

//ср.знач макс и мин элементов

double averageUnsorted = round((minUnsorted + maxUnsorted) / 2.0);

double averageSorted = round((minSorted + maxSorted) / 2.0);

cout << "Среднее значение (неотсортированный): " << averageUnsorted << endl;

cout << "Среднее значение (отсортированный): " << averageSorted << endl;

// Поиск индексов для неотсортированного массива

vector<int> indicesUnsorted;

start = chrono::high\_resolution\_clock::now();

findIndices(arr, N, averageUnsorted, indicesUnsorted);

end = chrono::high\_resolution\_clock::now();

chrono::duration<double, std::nano> elapsedIndicesUnsorted = end - start;

cout << "Индексы элементов, равных среднему значению (неотсортированный): ";

for (int index : indicesUnsorted) {

cout << index << " ";

}

cout << "\n";

cout << "Количество: " << indicesUnsorted.size() << endl;

cout << "Время поиска индексов (неотсортированный): " << elapsedIndicesUnsorted.count() << " наносекунд" << endl;

vector<int> indicesSorted;

start = chrono::high\_resolution\_clock::now();

findIndices(arr, N, averageSorted, indicesSorted);

end = chrono::high\_resolution\_clock::now();

chrono::duration<double, std::nano> elapsedIndicesSorted = end - start; // Время в наносекундах

cout << "Индексы элементов, равных среднему значению (отсортированный): ";

for (int index : indicesSorted) {

cout << index << " ";

}

cout << "\n";

cout << "Количество: " << indicesSorted.size() << endl;

cout << "Время поиска индексов (отсортированный): " << elapsedIndicesSorted.count() << " наносекунд" << endl;

cout << "\n";

//5 zad

int a;

cout << "Введите число a: ";

cin >> a;

int countLess = countLessThan(arr, N, a);

cout << "Количество элементов в отсортированном массиве, которые меньше числа " << a << ": " << countLess << endl;

cout << "\n";

//6 zad

int b;

cout << "Введите число b: ";

cin >> b;

int countGreater = countBiggestThan(arr, N, b);

cout << "Количество элементов в отсортированном массиве, которые больше числа " << b << ": " << countGreater << endl;

cout << "\n";

//7 zad

int searchValue;

cout << "Введите число для поиска: ";

cin >> searchValue;

// Сравнение скорости бинарного поиска и линейного поиска

auto startLinear = chrono::high\_resolution\_clock::now();

bool foundLinear = linearSearch(arr, N, searchValue);

auto endLinear = chrono::high\_resolution\_clock::now();

chrono::duration<double, std::nano> elapsedLinear = endLinear - startLinear; // Время в наносекундах

auto startBinary = chrono::high\_resolution\_clock::now();

bool foundBinary = binarySearch(arr, N, searchValue);

auto endBinary = chrono::high\_resolution\_clock::now();

chrono::duration<double, std::nano> elapsedBinary = endBinary - startBinary; // Время в наносекундах

cout << "Результат линейного поиска: " << (foundLinear ? "Найдено" : "Не найдено") << endl;

cout << "Время линейного поиска: " << elapsedLinear.count() << " наносекунд" << endl;

cout << "Результат бинарного поиска: " << (foundBinary ? "Найдено" : "Не найдено") << endl;

cout << "Время бинарного поиска: " << elapsedBinary.count() << " наносекунд" << endl;

cout << "\n";

//8 zad

int index1, index2;

cout << "Введите индексы для обмена местами (от 0 до " << N - 1 << "): ";

cin >> index1 >> index2;

if (index1 >= 0 && index1 < N && index2 >= 0 && index2 < N) {

auto startSwap = chrono::high\_resolution\_clock::now();

swapElements(arr, index1, index2);

auto endSwap = chrono::high\_resolution\_clock::now();

chrono::duration<double, std::nano> elapsedSwap = endSwap - startSwap; // Время в наносекундах

cout << "Элементы массива после обмена местами:" << endl;

for (int i = 0; i < N; ++i) {

cout << arr[i] << " ";

}

cout << endl;

cout << "\n";

cout << "Время обмена: " << elapsedSwap.count() << " наносекунд" << endl;

}

else {

cout << "Ошибка: индексы должны быть в диапазоне от 0 до " << N - 1 << "." << endl;

}

return 0;

}