**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. 4372 |  | Казакова К.Е. |
| Преподаватель |  | Глущенко А. Г. |

Санкт-Петербург

2024

**Цель работы.**

Разработать программу, которая выполняет ряд операций с целочисленным массивом, включая его создание, сортировку, поиск элементов и анализ времени выполнения различных алгоритмов. Конкретные задачи программы включают: создание целочисленного массива фиксированной размерности с случайными значениями в заданном диапазоне, реализация алгоритма сортировки массива и измерение времени, затраченного на эту операцию, с использованием библиотеки chrono, поиск максимального и минимального элементов массива, а также измерение времени поиска в отсортированном и неотсортированном массиве, вычисление и вывод среднего значения максимального и минимального элементов, а также индексов всех элементов, равных этим значениям, с подсчетом времени поиска, определение количества элементов в отсортированном массиве, которые меньше заданного пользователем числа, определение количества элементов в отсортированном массиве, которые больше заданного пользователем числа, реализация бинарного поиска для проверки наличия введенного пользователем числа в отсортированном массиве и сравнение его скорости с обычным перебором, обмен значениями элементов массива по индексам, введенным пользователем, с измерением времени выполнения операции.

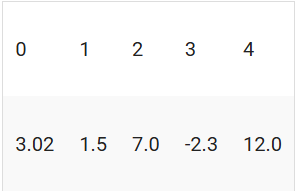
**Основные теоретические положения.**

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.

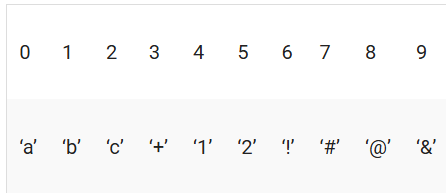
Элементы массива нумеруются с нуля. При описании массива используются те же модификаторы (класс памяти, const и инициализатор), что и для простых переменных.

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. Наглядно одномерный массив можно представить как набор пронумерованных ячеек, в каждой из которых содержится определенное значение:



Это пример одномерного массива из 5 элементов, каждый из которых представляет собой некоторое вещественное значение и каждое из этих значений имеет индекс от 0 до 4.

А вот пример одномерного массива из десяти элементов, представляющих собой одиночные символы:



Каждый элемент в этих массивах определяется значением индекса элемента. Например, в последнем массиве элемент с индексом 5 равен символу  ‘2’.

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.

Обращение к определенному элементу массива осуществляется с помощью указания значения индекса этого элемента.

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.

Значения всех элементов массива в памяти располагаются в непрерывной области одно за другим. Общий объем памяти, выделяемый компилятором для массива, определяется как произведение объема одного элемента массива на количество элементов в массиве.

Многомерные массивы определяются аналогично одномерным массивам. Количество элементов по каждому измерению указывается отдельно в квадратных скобках:

int  A1 [5] [3];                 //  Двумерный массив, элементами которого являются                                          //   значения типа int double D [10] [15] [3];   //  Трехмерный массив, элементами которого являются                                          //   значения типа double

Общее число элементов в многомерном массиве определяется как произведение количества элементов по каждому измерению. Также как и в одномерном массиве, элементы многомерных массивов располагаются друг за другом в непрерывном участке памяти.

При определении многомерные массивы могут инициализироваться определенными значениями.

Для доступа к определенному элементу многомерного массива необходимо указать в квадратных скобках конкретные значения всех индексов этого элемента.

Простейший циклический алгоритм ввода и вывода значений элементов некоторого одномерного массива выглядит так:

const int size = 5;

int arr[size];

*// Ввод элементов массива*

cout << "Введите " << size << " элементов массива: ";

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

cin >> arr[i];

}

*// Вывод элементов массива*

cout << "Элементы массива: ";

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

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

}

При заполнении двумерных массивов ввод значений элементов лучше осуществлять в табличной форме:

const int n = 10, m = 5;

short A [n] [m];

…

cout  << "Введите " << n << " строк из "<< m << " целых чисел: \n" for (int i = 0; i < n; ++i) {

       cout << "Строка " <<  i + 1 << ": ";        //  Вводим поэлементно значения i-й строки массива

for (int j = 0; j < m; ++j)

cin  >>  A [i] [j];        //  Очищаем поток ввода от возможных лишних введенных чисел

cin.sync();

   }

Вывод двумерных массивов, как правило, осуществляется в табличной форме. Реализация такого алгоритма может быть, например, такой:

const int n = 10, m = 10;

short A [n] [m];

for (int i = 0; i < n; ++i) //  Выводим i-ю строку массива {        for (int j = 0; j < m; ++j)        //  Выводим j-й элемент i-й строки массива

             cout << setw(7) << right << A [i] [j];

         cout << endl;

}

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.

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.

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 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.

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.

Алгоритм бинарного поиска – классический алгоритм поиска в отсортированном массиве, который использует дробление массива на половины. Если элемент, который необходимо найти, присутствует в списке, то бинарный поиск возвращает ту позицию, в которой он был найден. 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 создает целочисленный массив размерности N, равной 100, где элементы массива принимают случайные значения в диапазоне от -99 до 99. После этого программа должна отсортировать заданный массив, используя один из алгоритмов сортировки, и определить время, затраченное на сортировку, с помощью библиотеки chrono. Далее необходимо найти максимальный и минимальный элементы массива, а также подсчитать время поиска этих элементов как в отсортированном, так и в неотсортированном массиве, используя ту же библиотеку chrono. Программа должна выводить среднее значение максимального и минимального элементов, округляя его при необходимости, а также индексы всех элементов, которые равны этому значению, и их количество, с подсчетом времени поиска. Кроме того, программа должна выводить количество элементов в отсортированном массиве, которые меньше заданного пользователем числа a, а также количество элементов, которые больше числа b, также инициализируемого пользователем. Программа должна предоставлять информацию о том, содержится ли введенное пользователем число в отсортированном массиве, реализуя алгоритм бинарного поиска и сравнивая его скорость с обычным перебором. В завершение, программа должна позволять пользователю менять местами элементы массива по введенным индексам и выводить скорость этой операции, используя библиотеку chrono.

**Выполнение работы.**

| Ввод пользователем и обработка данных | | Работа алгоритма и вывод на экран |
| --- | --- | --- |
| Меню | | |
| При запуске программы перед пользователем появляется, выбор задания. | | Меню: |
| Вывод целочисленного массив размерности *N* = 100, элементы которого принимают случайное значение в диапазоне от -99 до 99. | | |
| При вводе пользователем корректного значения пункта меню программа выполняется. | | Для выполнения первого задания нужно создать и вывести целочисленный массив размерности *N* = 100. Элементы массивы должны принимать случайное значение в диапазоне от -99 до 99. |
| Сортировка массива, созданного в первом пункте практической работы. Определение времени, затраченного на сортировку с помощью библиотеки chrono. | | |
| Пользователь вводит номер задания и в результате получает исходный массив, а затем его отсортированную версию. Также в консоли отображается время, затраченное на сортировку данного массива. | | Для выполнения второго задания нужно вывести на экран двоичное представление в памяти (все разряды) целого числа |
| Вывод на экран максимального и минимального элемента массива. Подсчет времени поиска этих элементов в отсортированном массиве и неотсортированном, используя библиотеку chrono. | | |
| Пользователь вводит номер задания, а на выходе получает отсортированный массив, его минимальный и максимальный элемент, время поиска в неотсортированном и отсортированном виде. | Для выполнения третьего задания нужно вывести на экран максимального и минимального элемента массива, время поиска этих элементов в отсортированном массиве и неотсортированном. | |
| Вывод на экран среднего значения максимального и минимального значения в отсортированном и неотсортированном массиве, индексы всех элементов, которые равны этому значению, и их количество. Время поиска. | | |
| Пользователь вводит номер задания, а на выходе получает среднего значения максимального и минимального значения в отсортированном и неотсортированном массиве, индексы всех элементов, которые равны этому значению, и их количество. Время поиска. | Для выполнения четвертого задания нужно вывести на экран среднего значения максимального и минимального значения в отсортированном и неотсортированном массиве, индексы всех элементов, которые равны этому значению, и их количество. Время поиска. | |
| Вывод количества элементов в отсортированном массиве, которые меньше числа *a*, которое инициализируется пользователем. | | |
| Пользователь вводит номер задания и число, в результате получает количество элементов отсортированного массива, которые меньше инициализированного. | Для выполнения задания нужно обеспечить инициализацию числа, которое вводит пользователь, вывести на экран количество элементов, которые меньше заданного. | |
| Вывод количества элементов в отсортированном массиве, которые больше числа *b*, которое инициализируется пользователем. | | |
| Пользователь вводит номер задания и число, в результате получает количество элементов отсортированного массива, которые больше инициализированного. | Для выполнения задания нужно обеспечить инициализацию числа, которое вводит пользователь, вывести на экран количество элементов, которые больше заданного. | |
| Вывод информации о том, есть ли введенное пользователем число в отсортированном массиве. Реализация алгоритма бинарного поиска. Сравнение скорости его работы с обычным перебором. | | |
| Пользователь вводит номер задания на выходе получает информацию о том есть ли заданное число в массиве, а также сравнения скорости выполнения бинарного поиска и обычного перебора. | Для выполнения задания нужно обеспечить инициализацию числа, которое вводит пользователь, вывести на экран информацию о том имеется ли в массиве заданное число, скорость выполнения алгоритма с помощью бинарного поиска и обычного перебора. | |
| Смена мест элементов массива, индексы которых вводит пользователь. | | |
| Пользователь вводит номер задания, индексы элементов массива. На выходе получает исходный массив, а затем тот же массив, но с учетом смены мест элементов массива, а также время выполнения обмена. | Для выполнения задания нужно обеспечить инициализацию индексов элементов массива, которые вводит пользователь, вывести на экран массив после смены мест переменных и время, затраченное на обмен. | |

**Выводы.**

В результате выполнения поставленных задач была разработана программа, которая эффективно работает с целочисленным массивом размером 100, заполняя его случайными значениями в диапазоне от -99 до 99. Программа успешно реализует алгоритм сортировки, позволяя пользователю выбрать метод сортировки, и точно измеряет время, затраченное на эту операцию, с использованием библиотеки chrono. После сортировки программа находит максимальный и минимальный элементы массива, а также подсчитывает время, необходимое для поиска этих элементов как в отсортированном, так и в неотсортированном массиве. Это позволяет продемонстрировать разницу в производительности между двумя состояниями массива. Программа также вычисляет среднее значение максимального и минимального элементов, округляя его при необходимости, и выводит индексы всех элементов, равных этому значению, а также их количество, с подсчетом времени поиска. Кроме того, программа предоставляет пользователю возможность вводить значения a и b для подсчета количества элементов в отсортированном массиве, которые меньше a и больше b соответственно. Это взаимодействие с пользователем делает программу более гибкой и полезной. Реализованный алгоритм бинарного поиска позволяет быстро определить, содержится ли введенное пользователем число в отсортированном массиве, и программа сравнивает скорость бинарного поиска с обычным перебором, что подчеркивает эффективность бинарного поиска. В завершение, программа позволяет пользователю менять местами элементы массива по введенным индексам и выводит время, затраченное на эту операцию, что демонстрирует возможность работы с массивами и манипуляции с их элементами.

Приложение А

рабочий код

#include <iostream>

#include <chrono>

#include <time.h>

#include <ctime>

#include <cmath>

using namespace std;

using namespace chrono;

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

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

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

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

int b = arr[j]; // создали дополнительную переменную

arr[j] = arr[j + 1]; // меняем местами

arr[j + 1] = b; // значения элементов

}

}

}

}

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

bool swapped = true;

int start = 0;

int end = n - 1;

while (swapped) {

swapped = false;

// Проход слева направо

for (int i = start; i < end; ++i) {

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

std::swap(arr[i], arr[i + 1]);

swapped = true;

}

}

if (!swapped) {

break;

}

swapped = false;

--end;

// Проход справа налево

for (int i = end - 1; i >= start; --i) {

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

std::swap(arr[i], arr[i + 1]);

swapped = true;

}

}

++start;

}

}

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

int swap, count = 0;

float k = 1.247, S = n - 1;

while (S >= 1)

{

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

{

if (arr[i] > arr[int(i + S)])

{

swap = arr[int(i + S)];

arr[int(i + S)] = arr[i];

arr[i] = swap;

}

}

S /= k;

}

while (true)

{

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

{

if (arr[i] > arr[i + 1])

{

swap = arr[i + 1];

arr[i + 1] = arr[i];

arr[i] = swap;

}

else count++;

}

if (count == n - 1)

break;

else

count = 0;

}

}

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

int i, key, j;

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

key = arr[i];

j = i - 1;

// Сдвигаем элементы массива, которые больше key, на одну позицию вперед

while (j >= 0 && arr[j] > key) {

arr[j + 1] = arr[j];

j = j - 1;

}

arr[j + 1] = key;

}

}

void printArray(int arr[], int n) { //Функция для вывода массива

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

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

}

cout << endl;

}

void quicksort(int arr[], int end, int begin)

{

int mid;

int f = begin;

int l = end;

mid = arr[(f + l) / 2];

while (f < l)

{

while (arr[f] < mid) f++;

while (arr[l] > mid) l--;

if (f <= l)

{

swap(arr[f], arr[l]);

f++;

l--;

}

}

if (begin < l) quicksort(arr, l, begin);

if (f < end) quicksort(arr, end, f);

}

int binarySearch(int arr[], int value, int start, int end) {

if (end >= start) {

int mid = start + (end - start) / 2;

if (arr[mid] == value) {

return mid;

}

if (arr[mid] > value) {

return binarySearch(arr, value, start, mid - 1);

}

return binarySearch(arr, value, mid + 1, end);

}

return -1;

}

void merge(int arr[], int left, int mid, int right) {

int n1 = mid - left + 1;

int n2 = right - mid;

// Создаем временные массивы

int\* L = new int[n1];

int\* R = new int[n2];

// Копируем данные во временные массивы

for (int i = 0; i < n1; i++)

L[i] = arr[left + i];

for (int j = 0; j < n2; j++)

R[j] = arr[mid + 1 + j];

// Слияние временных массивов обратно в arr[left..right]

int i = 0, j = 0, k = left;

while (i < n1 && j < n2) {

if (L[i] <= R[j]) {

arr[k] = L[i];

i++;

}

else {

arr[k] = R[j];

j++;

}

k++;

}

// Копируем оставшиеся элементы L[], если есть

while (i < n1) {

arr[k] = L[i];

i++;

k++;

}

// Копируем оставшиеся элементы R[], если есть

while (j < n2) {

arr[k] = R[j];

j++;

k++;

}

// Освобождаем выделенную память

delete[] L;

delete[] R;

}

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);

}

}

int main(int argc, char\* argv[]) {

setlocale(LC\_ALL, "rus");

srand(time(0));

const int n = 100;

int arr[n];

const int f = 99;

const int min = -99;

const int max = 99;

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

arr[i] = rand() % (max - min + 1) + min;

}

int k;

do {

cout << "1) Создает целочисленный массив размерности N = 100. Элементы массивы должны принимать случайное значение в диапазоне от -99 до 99." << endl;

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

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

cout << "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." << endl;

cout << "5) Выводит количество элементов в отсортированном массиве, которые меньше числа a, которое инициализируется пользователем." << endl;

cout << "6) Выводит количество элементов в отсортированном массиве, которые больше числа b, которое инициализируется пользователем." << endl;

cout << "7) Выводит информацию о том, есть ли введенное пользователем число в отсортированном массиве. Реализуйте алгоритм бинарного поиска. Сравните скорость его работы с обычным перебором. " << endl;

cout << "8) Меняет местами элементы массива, индексы которых вводит пользователь. Выведите скорость обмена, используя библиотеку chrono." << endl;

cout << "9) ИДЗ№2: Сортировка слиянием. Вывод времени выполнения сортировки слиянием в сравнении с сортировкой Quick sort. " << endl;

time\_point<steady\_clock, duration<\_\_int64, ratio<1, 1000000000>>> start1, end1;

nanoseconds result1;

cout << "Введите номер задания: ";

cin >> k;

switch (k) {

case 1: {

cout << "Целочисленный массив со значениями от -99 до 99: " << endl;

int arr1[n];

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

arr1[i] = rand() % (max - min + 1) + min; //включая -99 и 99

}

printArray(arr1, n);

break;

}

case 2: {

cout << "Неотсортированный массив: " << endl;

printArray(arr, n);

auto start = steady\_clock::now();

quicksort(arr, f, 0);

auto end = steady\_clock::now();

auto result = duration\_cast<nanoseconds>(end - start);

cout << "Отсортированный массив: " << endl;

printArray(arr, n);

cout << "Время выполнения сортировки Quick sort (быстрая сортировка): " << result.count() << " наносекунд." << endl;

break;

}

case 3: {

int arr3[n];

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

arr3[i] = rand() % (max - min + 1) + min; //случайный массив

}

int min3 = arr3[0];

int max3 = arr3[0];

auto start2 = steady\_clock::now();

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

if (arr3[i] < min3) {

min3 = arr3[i];

}

if (arr3[i] > max3) {

max3 = arr3[i];

}

}

auto end2 = steady\_clock::now();

auto result2 = duration\_cast<nanoseconds>(end2 - start2);

int minArr3, maxArr3;

quicksort(arr3, f, 0);

auto start3 = steady\_clock::now();

minArr3 = arr3[0];

maxArr3 = arr3[f];

auto end3 = steady\_clock::now();

auto result3 = duration\_cast<nanoseconds>(end3 - start3);

cout << "Массив: " << endl;

printArray(arr3, n);

cout << "Минимальный элемент массива: " << minArr3 << endl;

cout << "Максимальный элемент массива: " << maxArr3 << endl;

cout << "Время поиска в неотсортированном массиве: " << result2.count() << " наносекунд." << endl;

cout << "Время поиска в отсортированном массиве: " << result3.count() << " наносекунд." << endl;

}

break;

case 4: {

int arr4[n];

int count = 0, count1 = 0;

float mid, mid1;

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

arr4[i] = rand() % (max - min + 1) + min; //включая -99 и 99

}

int min4 = arr4[0];

int max4 = arr4[0];

cout << "Неотсортированный массив: ";

printArray(arr4, n);

cout << endl;

auto start4 = steady\_clock::now();

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

if (arr4[i] < min4) {

min4 = arr4[i];

}

if (arr4[i] > max4) {

max4 = arr4[i];

}

}

mid = round((min4 + max4) / 2);

cout << "Среднее значение максимального и минимального элементов массива: " << mid << endl;

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

if (arr4[i] == mid) {

cout << i << "-й элемент массива равен среднему значению" << endl;

count += 1;

}

}

if (count == 0) {

cout << "Такого элемента в массиве не найдено." << endl;

}

auto end4 = steady\_clock::now();

auto result4 = duration\_cast<nanoseconds>(end4 - start4);

cout << "Количество элементов массива, которые равны среднему значению: " << count << endl;

cout << "Время поиска в неотсортированном массиве: " << result4.count() << " наносекунд." << endl;

quicksort(arr4, f, 0);

cout << "Отсортированный массив: " << endl;

printArray(arr4, n);

int minArr4, maxArr4;

minArr4 = arr4[0];

maxArr4 = arr4[f];

mid1 = round((minArr4 + maxArr4) / 2);

cout << "Среднее значение максимального и минимального элементов массива: " << mid1 << endl;

auto start5 = steady\_clock::now();

if (binarySearch(arr4, mid1, 0, f) == -1) {

cout << "Такого элемента в массиве не найдено." << endl;

}

else {

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

if (arr4[i] <= mid1) {

if (arr4[i] == mid1) {

cout << i << "-й элемент массива равен среднему значению" << endl;

count1 += 1;

}

else {

break;

}

}

}

}

auto end5 = steady\_clock::now();

auto result5 = duration\_cast<nanoseconds>(end5 - start5);

cout << "Количество элементов массива, которые равны среднему значению: " << count << endl;

cout << "Время поиска в отсортированном массиве: " << result5.count() << " наносекунд." << endl;

}

break;

case 5: {

int a5, count5 = 0;

cout << "Введите число: ";

cin >> a5;

int arr5[n];

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

arr5[i] = rand() % (max - min + 1) + min;

}

quicksort(arr5, f, 0);

printArray(arr5, n);

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

if (arr5[i] <= a5) {

if (arr5[i] < a5) {

count5 += 1;

}

else {

break;

}

}

}

cout << "Количество элементов меньше введённого: " << count5 << endl;

}

break;

case 6: {

int a6, count6 = 0;

cout << "Введите число: ";

cin >> a6;

int arr6[n];

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

arr6[i] = rand() % (max - min + 1) + min;

}

quicksort(arr6, f, 0);

printArray(arr6, n);

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

if (arr6[i] >= a6) {

if (arr6[i] > a6) {

count6 += 1;

}

else {

break;

}

}

}

cout << "Количество элементов больше введённого: " << count6 << endl;

}

break;

case 7: {

int a7, val;

int arr7[n];

bool flag = true;

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

arr7[i] = rand() % (max - min + 1) + min;

}

quicksort(arr7, f, 0);

printArray(arr7, n);

cout << "Введите число, которое необходимо найти: ";

cin >> a7;

cout << endl;

auto start6 = steady\_clock::now();

val = binarySearch(arr7, a7, 0, f);

auto end6 = steady\_clock::now();

auto result6 = duration\_cast<nanoseconds>(end6 - start6);

auto start7 = steady\_clock::now();

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

if (arr7[i] == a7) {

val = arr7[i];

flag = true;

break;

}

else {

flag = false;

}

}

if (flag == true) {

cout << "Введенное число есть в массиве." << endl;

}

else {

cout << "Введенного числа в массиве нет." << endl;

}

auto end7 = steady\_clock::now();

auto result7 = duration\_cast<nanoseconds>(end7 - start7);

cout << "Время бинарного поиска: " << result6.count() << " наносекунд." << endl;

cout << "Время поиска перебором: " << result7.count() << " наносекунд." << endl;

}

break;

case 8: {

int a8, b8;

cout << "Введите индексы элементов массива: ";

cin >> a8;

cout << " ";

cin >> b8;

cout << endl;

const int n7 = 100;

int arr8[n];

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

arr8[i] = rand() % (max - min + 1) + min;

}

quicksort(arr8, f, 0);

cout << "Изначальный массив: " << endl;

printArray(arr8, n);

auto start8 = steady\_clock::now();

swap(arr8[a8], arr8[b8]);

auto end8 = steady\_clock::now();

auto result8 = duration\_cast<nanoseconds>(end8 - start8);

cout << "Массив после обмена переменных: " << endl;

printArray(arr8, n);

cout << "Скорость обмена: " << result8.count() << " наносекунд." << endl;

}

break;

case 9: {

int arr9[n], arr10[n];

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

arr9[i] = rand() % (max - min + 1) + min;

arr10[i] = arr9[i];

}

cout << "Неотсортированный массив: ";

printArray(arr9, n);

cout << endl;

auto start9 = steady\_clock::now();

mergeSort(arr9, 0, f);

auto end9 = steady\_clock::now();

auto result9 = duration\_cast<nanoseconds>(end9 - start9);

cout << "Отсортированный массив: ";

printArray(arr9, n);

cout << endl;

auto start10 = steady\_clock::now();

quicksort(arr10, f, 0);

auto end10 = steady\_clock::now();

auto result10 = duration\_cast<nanoseconds>(end10 - start10);

cout << "Время сортировки слиянием: " << result9.count() << endl;

cout << "Время сортировки Quick sort: " << result10.count() << endl;

}

break;

}

cout << "Введите любое число чтобы продолжить" << endl;

} while (cin >> k);

}