**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра информационных систем**

отчет

**по практической работе №1**

**по дисциплине «Программирование»**

Тема: Одномерные статические массивы.

|  |  |  |
| --- | --- | --- |
| Студент гр. 4372 |  | Федюшкин Л.В. |
| Преподаватель |  | Глущенко А. Г. |

Санкт-Петербург

2024

**Цель работы.**

Разработать алгоритм и написать программу, которая позволяет выводить целочисленный массив размерности *N* = 100, отсортировать заданный в пункте 1 массив […] сортировкой (от меньшего к большему), найти максимальный и минимальный элемент массива, выводит среднее значение (если необходимо, число нужно округлить) максимального и минимального значения в отсортированном и неотсортированном, выводит количество элементов в отсортированном массиве, которые меньше числа *a,* выводит количество элементов в отсортированном массиве, которые больше числа *b,* выводит информацию о том, есть ли введенное пользователем число в отсортированном массиве.

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 |
| 3.02 | 1.5 | 7.0 | -2.3 | 12.0 |

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:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| ‘a’ | ‘b’ | ‘c’ | ‘+’ | ‘1’ | ‘2’ | ‘!’ | ‘#’ | ‘@’ | ‘&’ |

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

### ****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 будетупорядочена. 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.

### ****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 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.

### ****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.

**Постановка задачи.**

Разработать алгоритм и написать программу, которая позволяет выводить целочисленный массив размерности *N* = 100, отсортировать заданный в пункте 1 массив […] сортировкой (от меньшего к большему), найти максимальный и минимальный элемент массива, выводит среднее значение (если необходимо, число нужно округлить) максимального и минимального значения в отсортированном и неотсортированном, выводит количество элементов в отсортированном массиве, которые меньше числа *a,* выводит количество элементов в отсортированном массиве, которые больше числа *b,* выводит информацию о том, есть ли введенное пользователем число в отсортированном массиве.

**Выполнение работы.**

| Ввод пользователем и обработка данных | Работа алгоритма и вывод на экран | |
| --- | --- | --- |
| Меню | | |
| При запуске программы перед пользователем появляется, выбор задания. | Меню: | |
| Вывод несортированного списка | | |
| Выводится несортированный список |  | |
| Вывод на экран сортированного списка | | |
| Выводится сортированный список |  | |
| Вывод на экран минимального, максимального элементов и время поиска | | |
|  | | |
| Вывод на экран среднего значения, количество элементов равных среднему значению, а также время поиска. | | |
|  | | |
| Вывод на экран количество элементов меньших a | | |
| Ввод с клавиатуры числа a | |  |
| Вывод на экран количество элементов больших b | | |
| Ввод с клавиатуры числа b | |  |
| Выводит информацию о том, есть ли введенное пользователем число в отсортированном массиве. | | |
| Меняет местами элементы массива, индексы которых вводит пользователь. | | |

**Выводы.**

В процессе выполнения работы научились работать с одномерными массивами, создавать алгоритмы сортировки: пузырьковая сортировка, Shaker sort, Comb sort, insert sort, quick sort.

Приложение А

рабочий код

#include <iostream>

#include <conio.h>

#include <windows.h>

#include <chrono>

#include <vector>

using namespace std;

using namespace chrono;

//Палитра цветов

namespace Color {

// ANSI escape codes для изменения цвета текста

const string RESET = "\033[0m"; //сброс

const string RED = "\033[1m\033[31m"; //красный

const string GREEN = "\033[1m\033[32m";//зеленый

const string YELLOW = "\033[1m\033[33m"; //желтый

const string BLUE = "\033[1m\033[34m";//синий

const string MAGENTA = "\033[1m\033[35m";// пурпурный

const string CYAN = "\033[1m\033[36m";// голубой

const string BLACK = "\033[1m\033[30m";// черный

const string WHITE = "\033[1m\033[37m"; // белый

void Red(const string& text) {

cout << RED << text << RESET << endl;

}

void Green(const string& text) {

cout << GREEN << text << RESET << endl;

}

void Yellow(const string& text) {

cout << YELLOW << text << RESET << endl;

}

void Blue(const string& text) {

cout << BLUE << text << RESET << endl;

}

void Magenta(const string& text) {

cout << MAGENTA << text << RESET << endl;

}

void Cian(const string& text) {

cout << CYAN << text << RESET << endl;

}

void Black(const string& text) {

cout << BLACK << text << RESET << endl;

}

void White(const string& text) {

cout << WHITE << text << RESET << endl;

}

}

using namespace Color;

//Сортировка пузырек

void bubble\_sort(int res[100])

{

bool swapped = false;

for (int i = 0; i < 100; i++)

{

swapped = false;

for (int j = 0; j < 100 - i - 1; j++)

{

if (res[j] > res[j + 1])

{

std::swap(res[j], res[j + 1]);

swapped = true;

}

}

if (swapped == false)

break;

}

}

//Сортировка шейкер

void shaker\_sort(int res[100])

{

bool swapped = true;

while (swapped != false)

{

swapped = false;

bool AorB = true;

if (AorB)

{

for (int i = 0; i < 99; i++)

{

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

{

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

swapped = true;

AorB = false;

}

}

}

else

{

for (int j = 99; j > 0; j--)

{

if (res[j - 1] > res[j])

{

std::swap(res[j], res[j - 1]);

swapped = true;

AorB = true;

}

}

}

if (swapped == false) {

break;

}

}

}

//Бинарный поиск

bool BinarySearch(int res[100], int num)

{

int left = 0;

int right = 100 - 1;

int middle;

while (left <= right)

{

middle = (left + right) / 2;

if (res[middle] == num)

{

return 1;

}

if (res[middle] < num)

{

left = middle + 1;

}

else

{

right = middle - 1;

}

}

return 0;

}

//Сортировка расческой

void CombSort(int res[100])

{

int step = 100;

bool isSwaps = true;

while (step > 1 and isSwaps == true)

{

step = int(step / 1.247);

isSwaps = false;

for (int i = 0; i < 100 - step; i++)

{

if (res[i] > res[i + step])

{

swap(res[i], res[i + step]);

isSwaps = true;

}

}

}

}

//Сортировка вставками

void InsertSort(int res[100])

{

for (int j = 1; j < 100; j++)

{

int bufer = res[j];

int i = j - 1;

while (i >= 0 && res[i] > bufer)

{

res[i + 1] = res[i];

i--;

}

res[++i] = bufer;

}

}

//Быстрая сортировка

void quickSort(vector<int>& res, int first, int last) {

if (first < last) {

int left = first;

int right = last;

int middle = res[(left + right) / 2];

// Разделяем массив на две части

while (left <= right) {

// Ищем элементы, которые нужно поменять местами

while (res[left] < middle) {

left++;

}

while (res[right] > middle) {

right--;

}

if (left <= right) {

swap(res[left], res[right]);

left++;

right--;

}

}

if (first < right) {

quickSort(res, first, right);

}

if (left < last) {

quickSort(res, left, last);

}

}

}

int BinarySearch1(int res[100], int num)

{

int left = 0;

int right = 100 - 1;

int middle;

int k = 0;

while (left <= right)

{

middle = (left + right) / 2;

if (res[middle] == num)

{

return middle;

}

if (res[middle] < num)

{

left = middle + 1;

}

else

{

right = middle - 1;

}

}

return 0;

}

//Сортировка Шелла

void shellSort(int res[100]) {

// Начинаем с большого интервала и уменьшаем его

for (int gap = 100 / 2; gap > 0; gap /= 2) {

// Выполняем сортировку вставками для текущего интервала

for (int i = gap; i < 100; i++) {

int temp = res[i];

int j;

for (j = i; j >= gap && res[j - gap] > temp; j -= gap) {

res[j] = res[j - gap];

}

res[j] = temp;

}

}

}

int main() {

setlocale(LC\_ALL, "ru");

srand(time(NULL));

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

nanoseconds result, result1;

int yop;

int key;

int choice = 0;

int res[100];

int mini = 101;

int maxi = -101;

while (true) {

//Меню

cout << "\033[1m\033[31m";

Red("Выберите пункт (1-8) или 0 для выхода\n");

cout << "\033[0m";

cout << "1. Пункт 1. Массив размерности N = 100\n";

cout << "2. Пункт 2. Отсортировать\n";

cout << "3. Пункт 3. Найти максимальный и минимальный элемент\n";

cout << "4. Пункт 4. Выводит среднее значение\n";

cout << "5. Пункт 5. Меньше числа a\n";

cout << "6. Пункт 6. Больше числа b\n";

cout << "7. Пункт 7. Есть ли введенное пользователем число\n";

cout << "8. Пункт 8. Меняет местами элементы массива\n";

cout << "9. Пункт 9. Сортировка шелла\n";

cout << "0. Выход\n";

cin >> choice;

if (choice == 0) {

std::cout << "\nПрограмма завершила свою работу" << std::endl;

break;

}

switch (choice) {

case 1:

while (true)

{

cout << "\033[1m\033[31m";

Green("Вы выбрали пункт 1");

cout << "\033[0m";

Cian("\nНесортированный список: ");

start = steady\_clock::now();

for (int l = 0; l < 100; l++)

{

int random1 = rand() % 200 - 100;

res[l] = random1;

if (random1 < mini)

{

mini = random1;

}

if (random1 > maxi)

{

maxi = random1;

}

cout << res[l] << " ";

}

end = steady\_clock::now();

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

key = result.count();

cout << endl;

char finish;

cout << "Хотите выйти из 1 пункта?: y/n: ";

cin >> finish;

if (finish == 'y' || finish == 'Y' || finish == 'н')

break;

}

case 2:

while (true)

{

Green("\nВы выбрали пункт 2");

cout << "\nСейчас вы можете выбрать тип сортировки:\n";

cout << "1: bubble\_sort\n";

cout << "2: shaker\_sort\n";

cout << "3: CombSort\n";

cout << "4: InsertSort\n";

cout << "5: quickSort\n";

cout << "0: Выход из второго пункта\n";

int sorty;

cin >> sorty;

if (sorty == 1) {

start = steady\_clock::now();

bubble\_sort(res);

Blue("Cортированный список: ");

for (int i = 0; i < 100; i++)

cout << res[i] << " ";

end = steady\_clock::now();

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

cout << "\033[1m\033[33m";

cout << "\n" << "Время работы: " << result.count() << "\n";

cout << "\033[0m";

}

if (sorty == 2) {

start = steady\_clock::now();

shaker\_sort(res);

Blue("Cортированный список: ");

for (int i = 0; i < 100; i++)

cout << res[i] << " ";

end = steady\_clock::now();

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

cout << "\033[1m\033[33m";

cout << "\n" << "Время работы: " << result.count() << "\n";

cout << "\033[0m";

}

if (sorty == 3) {

start = steady\_clock::now();

CombSort(res);

Blue("Cортированный список: ");

for (int i = 0; i < 100; i++)

cout << res[i] << " ";

end = steady\_clock::now();

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

cout << "\033[1m\033[33m";

cout << "\n" << "Время работы: " << result.count() << "\n";

cout << "\033[0m";

}

if (sorty == 4) {

start = steady\_clock::now();

InsertSort(res);

Blue("Cортированный список: ");

for (int i = 0; i < 100; i++)

cout << res[i] << " ";

end = steady\_clock::now();

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

cout << "\033[1m\033[33m";

cout << "\n" << "Время работы: " << result.count() << "\n";

cout << "\033[0m";

}

if (sorty == 5)

{

vector<int> arr(100);

for (int l = 0; l < 100; l++)

{

int random1 = rand() % 200 - 100;

arr[l] = random1;

}

cout << endl;

start = steady\_clock::now();

quickSort(arr, 0, 99);

Blue("Cортированный список: ");

for (int i = 0; i < 100; i++)

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

end = steady\_clock::now();

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

cout << "\033[1m\033[33m";

cout << "\n" << "Время работы: " << result.count() << "\n";

cout << "\033[0m";

}

if (sorty == 0)

break;

}

case 3:

while (true)

{

Green("Вы выбрали пункт 3\n");

cout << "min элемент массива : " << mini << endl;

cout << "max элемент массива : " << maxi << endl;

Yellow("Время затраченное на поиск минимального и максимального элемента в НЕсортированном массиве: ");

cout << key << endl;

Yellow("Время затраченное на поиск минимального и максимального элемента в Сортированном массиве: ");

start = steady\_clock::now();

int a = res[0], b = res[99];

end = steady\_clock::now();

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

cout << result.count() << "\n";

break;

}

case 4:

while (true)

{

Green("Вы выбрали пункт 4\n");

int srednee = int((mini + maxi) / 2);

cout << "Среднее значение min и max: " << srednee << endl;

int count = 0;

int indexi[50];

int o = 0;

for (int l = 0; l < 100; l++)

{

if (res[l] == srednee)

{

count += 1;

}

}

Cian("Количество элементов равных среднему значению: ");

cout << count << endl;

cout << "\033[1m\033[36m";

cout << "Индексы данных элементов: ";

cout << "\033[00m";

bool found = false;

start = steady\_clock::now();

for (int i = 0; i < 100; ++i)

{

if (res[i] == srednee)

{

cout << i << " ";

found = true;

}

}

end = steady\_clock::now();

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

if (!found) {

Red("Нет элементов, равных среднему значению.");

}

cout << endl;

cout << "Время поиска: " << result.count() << "\n";

break;

}

case 5:

while (true)

{

shaker\_sort(res);

Green("Вы выбрали пункт 5\n");

Blue("Введите число, чтобы найти количество элементов меньших a");

int a;

cout << "a = ";

cin >> a;

cout << "Количество элементов меньших a: ";

int fin = BinarySearch1(res, a);

if (fin != 0) {

cout << "Количество элементов меньших " << a << ": " << fin << endl;

}

else {

cout << "Число " << a << " нет в списке." << endl;

}

break;

}

case 6:

while (true)

{

shaker\_sort(res);

Green("Вы выбрали пункт 6\n");

Blue("Введите число, чтобы найти количество элементов больших a");

int a;

cout << "a = ";

cin >> a;

int fin = 100 - BinarySearch1(res, a);

if (fin != 0) {

cout << "Количество элементов больших " << a << ": " << fin << endl;

}

else {

cout << "\033[1m\033[31m";

cout << "Число " << a << " нет в списке." << endl;

cout << "\033[00m";

}

break;

}

case 7:

while (true)

{

Green("Вы выбрали пункт 7\n");

Blue("Введите число, которое хотите найти в списке");

int choice;

cout << "choice = ";

cin >> choice;

start = steady\_clock::now();

int otvet = BinarySearch(res, choice);

end = steady\_clock::now();

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

if (otvet != 0) {

cout << "Значение " << choice << " есть в массиве" << endl;

}

else {

cout << "\033[1m\033[31m";

cout << "Числа " << choice << " нет в списке." << endl;

cout << "\033[00m";

}

cout << "Время работы бинарного поиска: " << result.count() << endl;

start1 = steady\_clock::now();

for (int i = 0; i < 100; i++)

{

if (res[i] == choice)

return 1;

else

break;

}

end1 = steady\_clock::now();

result1 = duration\_cast<nanoseconds>(end1 - start1);

break;

}

case 8:

Green("\nВы выбрали пункт 8\n");

Cian("Сейчас необходимо поочередно ввести индексы элементов, нуждаяющихся в замене.");

int a, b;

cout << "Введите число A: ";

cin >> a;

cout << "Введите число B: ";

cin >> b;

int bufer;

start = steady\_clock::now();

bufer = res[a];

res[a] = res[b];

res[b] = bufer;

end = steady\_clock::now();

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

cout << "\nВот измененный массив: ";

for (int i = 0; i < 100; i++)

cout << res[i] << " ";

Blue("\nВремя смены элементов с данными индексами: ");

cout << result.count() << endl;

case 9:

while (true)

{

Green("Вы выбрали пункт 9. Сортировка Шелла\n");

Blue("Cейчас мы наглядно увидим разницу между сортировкой Шелла и Bubble\_Sort");

Cian("Введите 1 - бабл, 2 - сортировка шелла или 0-выход");

int sorty;

cin >> sorty;

if (sorty == 1) {

start = steady\_clock::now();

bubble\_sort(res);

Blue("Cортированный список: ");

for (int i = 0; i < 100; i++)

cout << res[i] << " ";

end = steady\_clock::now();

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

cout << "\033[1m\033[33m";

cout << "\n" << "Время работы: " << result.count() << "\n";

cout << "\033[0m";

}

if (sorty == 2) {

start = steady\_clock::now();

shellSort(res);

Blue("Cортированный список: ");

for (int i = 0; i < 100; i++)

cout << res[i] << " ";

end = steady\_clock::now();

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

cout << "\033[1m\033[33m";

cout << "\n" << "Время работы: " << result.count() << "\n";

cout << "\033[0m";

}

if (sorty == 0)

break;

}

default:

cout << "Неверный выбор. Пожалуйста, выберите от 0 до 8\n";

}

}

return 0;

}