**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

|  |  |  |
| --- | --- | --- |
| Студент(ка) гр. | Чишко В. О. |  |
| Преподаватель | Глущенко А. Г. |  |

Санкт-Петербург

2022

**Цель работы.**

Изучение одномерных статических массивов. Изучение алгоритмов сортировки и поиска.

**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.

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

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.

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.

**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. Выводит индексы всех элементов, которые равны этому значению, и их количество.

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.

Должна присутствовать возможность запуска каждого пункта многократно.

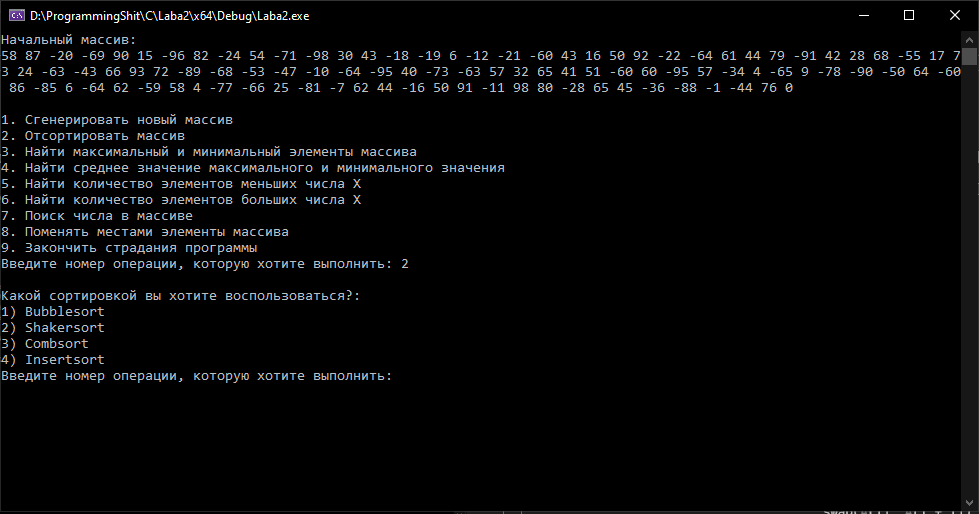
**Выполнение работы.**

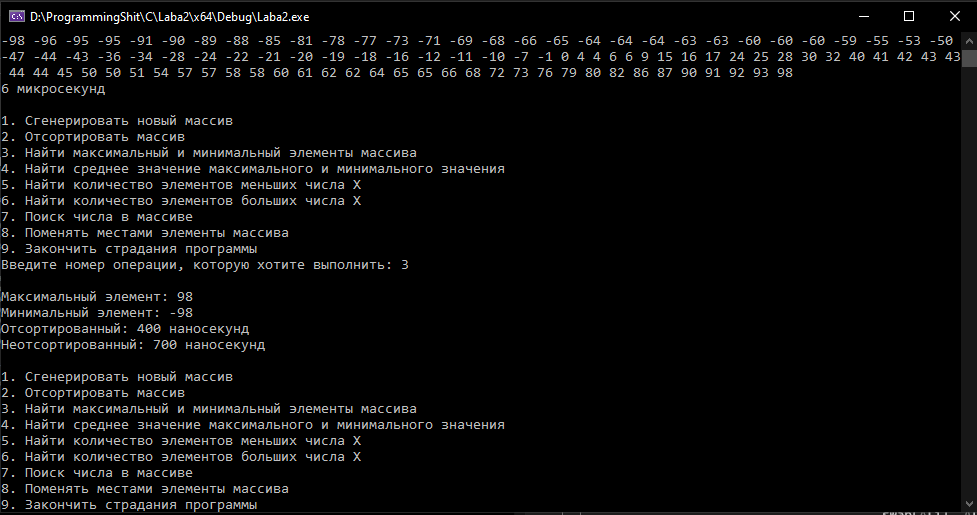
Код программы представлен в приложении А.

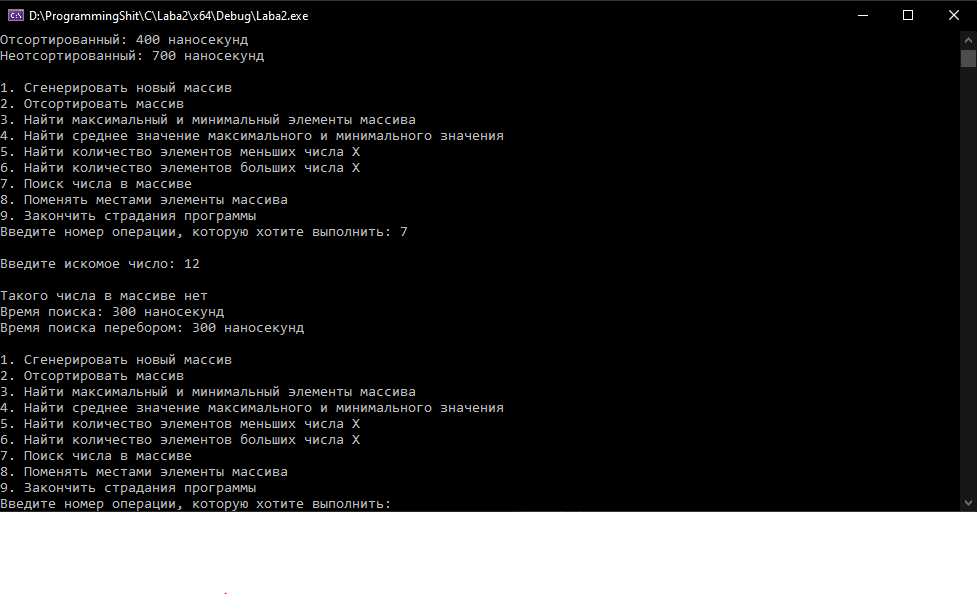
Блок описания кода и использованных алгоритмов

Для выполнения первого пункта реализован алгоритм быстрой сортировки. Для выбора опорного элемента написан алгоритм подсчёта среднего значения и поиска среди значений массива, ближайшего к среднему. Для измерения времени выполнения той или иной функции используются объекты типа time\_point из библиотеки chrono. Для выполнения второго пункта реализован алгоритм поиска наибольшего и наименьшего значений перебором. Для выполнения третьего пункта не реализовывался алгоритм поиска max и min значений и алгоритм из второго не использовался, т.к. функция из второго пункта просто выводила найденные значения в консоль, ничего не возвращая. Поэтому я просто сортирую массив и работаю с первым и последним значением. В пятом и шестом пунктах просто перебором считаем количество элементов больше или меньше введённого значения. В седьмом пункте реализован алгоритм бинарного пункта без рекурсии. В восьмом пункте просто считываем два значения и меняем их местами как в любой сортировке.

**Блок скриншотов работы программы**







**Выводы.**

Изучен навык работы с одномерными статическими массивами. Изучены базовые алгоритмы сортировок и поиска. Изучен навык бинарного поиска.

Бинарный поиск может быть медленнее на таких множествах значений как у нас (малого размера), но при увеличении размера массива бинарный поиск будет показывать куда большую скорость, чем обычный перебор.

Приложение А

рабочий код

#include <iostream>

#include <ctime>

#include <chrono>

using namespace std;

using namespace chrono;

void Bubblesort(int A[]) {

for (int n = 100; n > 0; --n) {

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

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

swap(A[i], A[i + 1]);

}

}

}

void Shakersort(int A[]) {

for (int n = 100; n > 0; --n) {

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

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

swap(A[i], A[i + 1]);

}

for (int i = n - 2; i > 0; --i) {

if (A[i] < A[i - 1])

swap(A[i], A[i - 1]);

}

}

}

void Combsort(int A[]) {

int gap = 100;

while (gap != 1) {

gap = int(gap / 1.3);

if (gap < 1)

gap = 1;

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

if (A[i] > A[i + gap])

swap(A[i], A[i + gap]);

}

}

}

void Insertsort(int A[]) {

int key, j;

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

key = A[i];

j = i - 1;

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

A[j + 1] = A[j];

j -= 1;

}

A[j + 1] = key;

}

}

void Viewmassive(int A[]) {

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

cout << A[i] << " ";

cout << "\n";

}//Эта функция просто показывает массив

int main() {

setlocale(0, "");

int input = 0;

int A[100], Ac[100];//Два массива создаётся для удобства и стабильной работы меню

srand(time(0));

cout << "Начальный массив:\n";

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

A[i] = -99 + rand() % 199;//1 массив

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

Ac[i] = A[i];//2 массив

Viewmassive(A);

cout << "\n";

Menu: {

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

A[i] = Ac[i];//Массив Ac[] выступает как маска, с которой впоследствии копируется массив A[]. Вот уже над ним творятся всякие надругательства

cout << "1. Сгенерировать новый массив\n";

cout << "2. Отсортировать массив\n";

cout << "3. Найти максимальный и минимальный элементы массива\n";

cout << "4. Найти среднее значение максимального и минимального значения\n";

cout << "5. Найти количество элементов меньших числа X\n";

cout << "6. Найти количество элементов больших числа X\n";

cout << "7. Поиск числа в массиве\n";

cout << "8. Поменять местами элементы массива\n";

cout << "9. Закончить страдания программы\n";

cout << "Введите номер операции, которую хотите выполнить: ";

cin >> input;

cout << "\n";

}//Как понятно из названия - это меню вызова заданий

switch (input) {

case 1: {

cout << "Новый массив:\n";

srand(time(0));

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

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

Viewmassive(A);

cout << "\n";

goto Menu;

}

case 2: {// Я дурачёк, и поэтому сделал возможность отсортировать массив разными способами

cout << "Какой сортировкой вы хотите воспользоваться?:\n";

cout << "1) Bubblesort\n";

cout << "2) Shakersort\n";

cout << "3) Combsort\n";

cout << "4) Insertsort\n";

cout << "Введите номер операции, которую хотите выполнить: ";

cin >> input;

cout << "\n";

if (input == 1) {

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

Bubblesort(A);

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

duration<double> sec = end - start;

Viewmassive(A);

cout << chrono::duration\_cast<chrono::microseconds>(end - start).count() << " микросекунд\n\n";

goto Menu;//Эта штука может показаться страшной, но на деле тут только сортировка и вывод массива. Страх появляется только из-за подсчитывания времени.

}

else if (input == 2) {

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

Shakersort(A);

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

duration<double> sec = end - start;

Viewmassive(A);

cout << chrono::duration\_cast<chrono::microseconds>(end - start).count() << " микросекунд\n\n";

goto Menu;

}

else if (input == 3) {

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

Combsort(A);

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

duration<double> sec = end - start;

Viewmassive(A);

cout << chrono::duration\_cast<chrono::microseconds>(end - start).count() << " микросекунд\n\n";

goto Menu;

}

else if (input == 4) {

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

Insertsort(A);

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

duration<double> sec = end - start;

Viewmassive(A);

cout << chrono::duration\_cast<chrono::microseconds>(end - start).count() << " микросекунд\n\n";

goto Menu;

}

else {

cout << "Братан, тут ошибочка вышла. Пропробуй ещё раз.\n\n";

goto Menu;

}

}

case 3: {

Insertsort(A);

int min = 100, max = -100;

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

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

if (A[i] > max)

max = A[i];

if (A[i] < min)

min = A[i];

}

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

cout << "Максимальный элемент: " << max << "\nМинимальный элемент: " << min << "\n";

cout << "Отсортированный: ";

cout << chrono::duration\_cast<chrono::nanoseconds>(end - start).count() << " наносекунд\n";

min = 100;

max = -100;

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

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

if (Ac[i] > max)

max = A[i];

if (Ac[i] < min)

min = Ac[i];

}

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

cout << "Неотсортированный: ";

cout << chrono::duration\_cast<chrono::nanoseconds>(end - start).count() << " наносекунд\n\n";

goto Menu;

}

case 4: {

Insertsort(A);

int max = A[99], min = A[0];

double mid = round((max + min) / 2);

cout << "Среднее значение: " << mid << "\n";

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

int count = 0;

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

if (Ac[i] == mid)

count += 1;

}

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

cout << "Найдено " << count << " похожих чисел. Они находятся на позициях: ";

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

if (Ac[i] == mid)

cout << i + 1 << " ";

}

cout << "\nВремя поиска: ";

cout << chrono::duration\_cast<chrono::nanoseconds>(end - start).count() << " наносекунд\n\n";

goto Menu;

}// тут можно было объеденить эти циклы, но мне так было проще

case 5: {

int x, count = 0;

cout << "Введите число X: ";

cin >> x;

Insertsort(A);

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

if (A[i] < x)

count += 1;

}

cout << "Кол-во элементов меньших " << x << " = " << count << "\n\n";

goto Menu;

}

case 6: {

int x, count = 0;

cout << "Введите число X: ";

cin >> x;

Insertsort(A);

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

if (A[i] > x)

count += 1;

}

cout << "Кол-во элементов больших " << x << " = " << count << "\n\n";

goto Menu;

}

case 7: {

Insertsort(A);

int left = -99, right = 99;

int mid, key;

cout << "Введите искомое число: ";

cin >> key;

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

while (left < right) {

mid = (left + right) / 2;

if (A[mid] > key)

right = mid;

else

left = mid + 1;

}

right -= 1;

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

if (A[right] == key)

cout << "\nТакое число в массиве есть\n";

else

cout << "\nТакого числа в массиве нет\n";// Это бинарный поиск

cout << "Время поиска: ";

cout << chrono::duration\_cast<chrono::nanoseconds>(end - start).count() << " наносекунд\n";

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

bool k = false;

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

if (A[i] == key) {

k = true;

}

}// Это поиск перебором

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

cout << "Время поиска перебором: ";

cout << chrono::duration\_cast<chrono::nanoseconds>(end - start).count() << " наносекунд\n\n";

goto Menu;

}

case 8: {

int x1, x2;

cout << "Введите 2 элемента, которые вы хотите поменять местами: ";

cin >> x1 >> x2;

cout << "\n";

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

swap(Ac[x1 - 1], Ac[x2 - 1]);

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

Viewmassive(Ac);

cout << "Скорость обмена: ";

cout << chrono::duration\_cast<chrono::nanoseconds>(end - start).count() << " наносекунд\n\n";

goto Menu;

}//здесь вообще 2 строчки рабочего кода

default: {

cout << "Братан, тут ошибочка вышла. Пропробуй дрогое число.\n\n";

goto Menu;

}

case 9: {

exit;

}//Этот кейс идёт ниже дефолта, т. к. при выходе программа идёт дальше вниз. Если было бы иначе, получилась бы петля.

}

}