**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. | Красножён Е. Д. |  |
| 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 второго пункта реализована возможность выбора сортировки для сравнения времени работы каждого алгоритма. 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 steady\_clock из библиотеки chrono. Для выполнения третьего пункта реализован алгоритм поиска наибольшего и наименьшего значений перебором. Для выполнения четвёртого пункта я просто сортирую массив и работаю с первым и последним значениями, т. к. они являются минимальным и максимальным соответственно. 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**

Скриншот 1

Изображение выглядит как текст

Автоматически созданное описание

Скриншот 2

Изображение выглядит как текст

Автоматически созданное описание

**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;

int main()

{

setlocale(0, "");

const int N = 100;

int input;

int arr[N];

int sortarr[N];

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. Поменять местами элементы массива\n";

cout << "8. Выйти из матрицы\n";

Menu: {

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

cin >> input;

cout << "\n";

}

switch (input) {

case 1: {

cout << "Новый массив:\n";

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

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

sortarr[i] = arr[i];

cout << arr[i] << ' ';

}

cout << "\n";

goto Menu;

}

case 2: {

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

for (int step = 1; step < N; step++) {

int key = sortarr[step];

int j = step - 1;

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

sortarr[j + 1] = sortarr[j];

--j;

}

sortarr[j + 1] = key;

}

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

duration<double> sec = end - start;

cout << "Отсортированный массив:";

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

cout << sortarr[i] << " ";

cout << '\n';

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

goto Menu;

}

case 3: {

cout << "Максимальный:" << sortarr[N - 1] << "\nМинимальный:" << sortarr[0];

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

int min = 100;

int max = -100;

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

if (sortarr[i] > max)

max = sortarr[i];

if (sortarr[i] < min)

min = sortarr[i];

}

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

cout <<"\nВремя поиска в отсортированном: " << duration\_cast<microseconds>(end - start).count() << " микросекунд\n";

start = steady\_clock::now();

min = 100;

max = -100;

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

if (arr[i] > max)

max = arr[i];

if (arr[i] < min)

min = arr[i];

}

end = steady\_clock::now();

cout << "Время поиска в несортированном: " << duration\_cast<microseconds>(end - start).count() << " микросекунд\n";

goto Menu;

}

case 4: {

int max = sortarr[N - 1], min = sortarr[0];

int mid = (max + min) / 2;

int count = 0;

cout << "Индексы чисел, равных среднему: ";

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

if (sortarr[i] == mid)

cout << i << ' ';

}

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

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

if (sortarr[i] == mid)

count += 1;

}

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

cout << "\nНайдено " << count << " чисел, равных среднему\n";

cout << "Время поиска в отсортированном: " << duration\_cast<microseconds>(end - start).count() << " микросекунд\n";

start = steady\_clock::now();

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

if (arr[i] == mid)

count += 1;

}

end = steady\_clock::now();

cout << "Время поиска в несортированном: " << duration\_cast<microseconds>(end - start).count() << " микросекунд\n";

goto Menu;

}

case 5: {

cout << "Введите число:";

int num;

int count = 0;

cin >> num;

int i = 0;

while (sortarr[i] < num) {

++count;

++i;

}

cout << '\n' << count << " чисел, меньших " << num<<'\n';

goto Menu;

}

case 6: {

cout << "Введите число:";

int num;

int count = 0;

cin >> num;

int i = N-1;

while (sortarr[i] > num) {

++count;

--i;

}

cout << '\n' << count << " чисел, больших " << num<<'\n';

goto Menu;

}

case 7: {

cout << "Введите индексы чисел: ";

int inp1, inp2;

cin >> inp1 >> inp2;

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

swap(sortarr[inp1], sortarr[inp2]);

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

cout << '\n';

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

cout << sortarr[i] << " ";

duration<double> sec = end - start;

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

swap(sortarr[inp1], sortarr[inp2]);

goto Menu;

}

case 8: {

break;

}

}

}