**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. | Минин М. А. |  |
| 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.

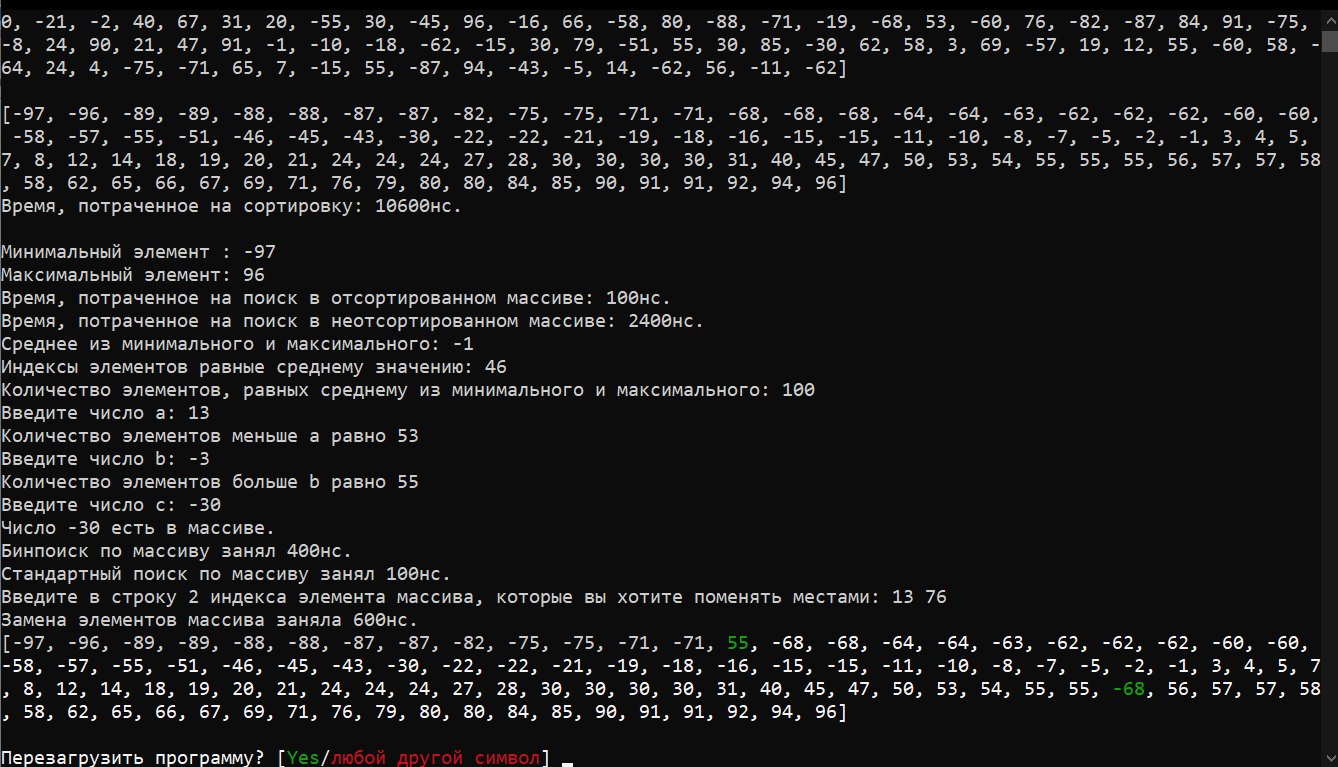
**Выполнение работы.**

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 <windows.h>

#include <ctime>

#include <string>

#include <chrono>

using namespace std;

void setTextColor(int textColorIndex) { // Смена цвета текста

HANDLE h = GetStdHandle(STD\_OUTPUT\_HANDLE);

SetConsoleTextAttribute(h, (0 << 4) + textColorIndex);

}

void quicksort(int\* arr, int end, int begin)

{

int mid;

int start = begin;

int finish = end;

mid = arr[(start + finish) / 2];

while (start < finish)

{

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

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

if (start <= finish)

{

swap(arr[start], arr[finish]);

start++;

finish--;

}

}

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

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

}

bool isRestartNeeded() { // Функция перезапуска программы

cin.clear(); // Очистка ввода от ошибок

cin.sync();

cout << "Перезагрузить программу? [";

setTextColor(2);

cout << "Yes";

setTextColor(15);

cout << "/";

setTextColor(4);

cout << "любой другой символ";

setTextColor(15);

cout << "] ";

string answer;

cin >> answer;

if (answer == "yes" || answer == "Yes" || answer == "YES")

return true;

else

return false;

}

bool notCorrectInput() { // Остановка программы при ошибочном вводе

if (cin.fail()) {

setTextColor(4);

cout << "Введён неверный тип данных!" << endl;

setTextColor(15);

cout << "Программа будет завершена через 2 с.";

double nowTime;

nowTime = time(0);

while (time(0) - nowTime != 2) {}

return true;

}

return false;

}

void fill(int N, int\* array, int\* arraySorted) {

cout << '[';

for (int i = 0; i < N; i++) { // Заполнение массива случайными числами от -99 до 99

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

arraySorted[i] = array[i];

if (i != N - 1)

cout << array[i] << ", ";

else

cout << array[i];

}

cout << ']' << '\n' << '\n';

}

void binSrch(int c,int left, int right, int N, int\* arraySorted) {

auto startTime3 = chrono::high\_resolution\_clock::now();

int medium = right / 2;

while (left + 1 < right) {

if (c >= arraySorted[medium])

left = medium;

else

right = medium;

medium = (right + left) / 2;

}

auto endTime3 = chrono::high\_resolution\_clock::now();

if (arraySorted[left] == c)

cout << "Число " << c << " есть в массиве." << '\n';

else

cout << "Числа " << c << " нет в массиве." << '\n';

cout << "Бинпоиск по массиву занял " << chrono::duration\_cast<chrono::nanoseconds>(endTime3 - startTime3).count() << "нс." << '\n';

}

int main()

{

const int N = 100;

setlocale(0, "");

int array[N];

int arraySorted[N];

srand(time(0));

// 1 Задание

fill(N, array, arraySorted);

// 2 Задание

auto startTime0 = chrono::high\_resolution\_clock::now();

quicksort(arraySorted, N - 1, 0);// Быстрая сортировка

auto endTime0 = chrono::high\_resolution\_clock::now();

cout << '[';

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

if (i != N - 1)

cout << arraySorted[i] << ", ";

else

cout << arraySorted[i];

}

cout << ']'<<'\n';

cout << "Время, потраченное на сортировку: " << chrono::duration\_cast<chrono::nanoseconds>(endTime0 - startTime0).count() << "нс." << '\n'<<'\n';

auto startTime1 = chrono::high\_resolution\_clock::now(); // Замер времени с сортировкой

int minimal = arraySorted[0];

int maximal = arraySorted[N - 1];

auto endTime1 = chrono::high\_resolution\_clock::now();

cout << "Минимальный элемент : " << minimal << '\n' << "Максимальный элемент: " << maximal << '\n';

cout << "Время, потраченное на поиск в отсортированном массиве: " << chrono::duration\_cast<chrono::nanoseconds>(endTime1 - startTime1).count() << "нс." << '\n';

// 3 Задание

auto startTime2 = chrono::high\_resolution\_clock::now();

float min = 999;

float max = -999;

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

if (array[i] < min)

min = array[i];

if (array[i] > max)

max = array[i];

}

auto endTime2 = chrono::high\_resolution\_clock::now();

cout << "Время, потраченное на поиск в неотсортированном массиве: " << chrono::duration\_cast<chrono::nanoseconds>(endTime2 - startTime2).count() << "нс." << '\n';

// 4 Задание

int middle = round((min + max) / 2);

cout << "Среднее из минимального и максимального: " << middle << '\n';

int cnt = 0;

cout << "Индексы элементов равные среднему значению: ";

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

if (arraySorted[i] == middle)

cout << i << ' ';

cnt++;

}

cout << '\n';

cout << "Количество элементов, равных среднему из минимального и максимального: " << cnt << '\n';

// 5 Задание

cout << "Введите число a: ";

int a;

cin >> a;

if (notCorrectInput())

return 0;

cnt = 0;

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

if (arraySorted[i] < a)

cnt++;

}

cout << "Количество элементов меньше a равно " << cnt << '\n';

// 6 Задание

cout << "Введите число b: ";

int b;

cin >> b;

if (notCorrectInput())

return 0;

cnt = 0;

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

if (arraySorted[i] > b)

cnt++;

}

cout << "Количество элементов больше b равно " << cnt << '\n';

// 7 Задание

cout << "Введите число c: ";

int c;

cin >> c;

if (notCorrectInput())

return 0;

int leftBorder = 0;

int rightBorder = N;

binSrch(c, leftBorder, rightBorder, N, arraySorted);

auto startTime4 = chrono::high\_resolution\_clock::now();

bool flag = false;

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

if (arraySorted[i] == c)

flag = true;

if (flag)

break;

}

auto endTime4 = chrono::high\_resolution\_clock::now();

cout << "Стандартный поиск по массиву занял " << chrono::duration\_cast<chrono::nanoseconds>(endTime4 - startTime4).count() << "нс." << '\n';

// 8 Задание

cout << "Введите в строку 2 индекса элемента массива, которые вы хотите поменять местами: ";

int firstElement;

int secondElement;

cin >> firstElement >> secondElement;

if (notCorrectInput())

return 0;

if (firstElement >= 0 and firstElement <= N - 1 and secondElement >= 0 and secondElement <= N - 1) {

auto startTime5 = chrono::high\_resolution\_clock::now();

swap(arraySorted[firstElement], arraySorted[secondElement]);

auto endTime5 = chrono::high\_resolution\_clock::now();

cout << "Замена элементов массива заняла " << chrono::duration\_cast<chrono::nanoseconds>(endTime5 - startTime5).count() << "нс." << '\n';

cout << '[';

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

if (i == firstElement or i == secondElement) {

setTextColor(2);

cout << arraySorted[i];

setTextColor(15);

}

else {

cout << arraySorted[i];

}

if (i != N - 1)

cout << ", ";

}

cout << ']';

cout << '\n';

cout << '\n';

}

else

cout << "Неверные индексы!" << '\n';

if (isRestartNeeded()) {

cout << '\n' << "Перезапуск программы..." << '\n' << '\n';

// Остановка времени на 2с

double nowTime;

nowTime = time(0);

while (time(0) - nowTime != 2) {}

// Перезапуск программы

main();

}

else { // Выход из программы

return 0;

}

}