**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 гр. 3372 |  | Надточаев Ф.А. |
| Преподаватель |  | Глущенко А. Г. |

Санкт-Петербург

2023

**Цель работы.**

Изучить одномерные статические массивы; получить практические навыки работы с массивами, различными сортировками, использования библиотеки chrono для измерения времени работы.

**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​ 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.

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.

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.

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.

**Постановка задачи.**

Разработать алгоритм и написать программу, которая позволяет:

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 в отсортированном и неотсортированном. Выводит индексы всех элементов, которые равны этому значению, и их количество. Подсчитайте время поиска.

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.

**Выполнение работы.**

Код программы представлен в приложении А.

|  |  |
| --- | --- |
| Ввод пользователем и обработка данных | Работа алгоритма и вывод на экран |
| Старт меню | |
| На экран выводится меню из 9-и пунктов программы, каждый из которых выполняет одну из функций, описанных в пункте постановки задачи. |  |
| Пункт 1. Вывод случайно сгенерированного массива. | |
| Пользователь вводит цифру 1. На экран выводится случайно сгенерированный массив из чисел в диапазоне от -99 до 99. |  |
| Пункт 2. Сортировка массива. | |
| На экран выводится изначальный и отсортированный с помощью quick sort массив, а так же время, потраченное на выполнение сортировки. |  |
| Пункт 3. Двоичное представление типа float в памяти. | |
| На экран выводится минимальный и максимальный элементы массива, время поиска в отсортированном и неотсортированном массиве. |  |
| Пункт 4. Вывод среднего значения в памяти . | |
| На экран выводится среднее значение, индексы элементов, равных среднему значению, их количество, время его поиска в отсортированном и неотсортированном массиве. |  |
| Пункт 5. Вывод количества элементов в отсортированном массиве, которые меньше числа, вводимого пользователем. | |
| Пользователь вводит число 8. На экран выводится количество элементов меньше числа 8, введенного пользователем. |  |
| Пункт 6. Вывод количества элементов в отсортированном массиве, которые больше числа, вводимого пользователем. | |
| Пользователь вводит число 11. На экран выводится количество элементов больше числа 11, введенного пользователем. |  |
| Пункт 7. Проверка наличия числа в массиве. | |
| Пользователь вводит число 94, на экран выводится информация о наличии элемента в массиве, а также время, затраченное на поиск обычным перебором и бинарным поиском. |  |
| Пункт 8. Смена элементов в массиве местами по индексам | |
| На экран выводится изначальный массив. Пользователь вводит идексы элементов (0, 1), которые нужно поменять местами. На экран выводится массив, с элементами, поменянными местами по индексам, а также время, потраченное на обмен. |  |

**Выводы.**

По итогам работы мы изучили одномерные статические массивы, получили практические навыки работы с массивами, различными сортировками, использования библиотеки chrono. Результатом проделанной работы является программа, принцип работы которой представлен в пункте «Выполнение работы» и код которой находится в приложении А.

Приложение А

рабочий код

#include <algorithm>

#include <clocale>

#include <cstdlib>

#include <iostream>

#include <chrono>

using namespace std;

// #2

void bubbleSort(int \*array, int length)

{

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

{

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

{

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

{

swap(array[j], array[j+1]);

}

}

}

}

void shakerSort(int \*array, int length)

{

int start = 0;

int end = length - 2;

while (start <= end)

{

for (int i = start; i < length - 1 - start; ++i)

{

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

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

}

for (int i = end ; i > 0 + start; --i)

{

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

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

}

start += 1;

end -= 1;

}

}

void combSort(int \*array, int length)

{

int gap = length;

while (gap >= 1)

{

for (int i = 0; i + gap < length; ++i)

{

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

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

}

gap /= 1.247;

}

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

{

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

{

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

swap(array[j], array[j+1]);

}

}

}

void insertSort(int \*array, int length)

{

int i, j, key;

for (int i = 1; i < length; ++i)

{

key = array[i];

j = i - 1;

while (j >= 0 && array[j] > key)

{

array[j+1] = array[j];

j--;

}

array[j+1] = key;

}

}

void quickSort(int \*array, int finish, int begin)

{

int r = begin;

int l = finish;

int mid = array[(r+l)/2];

while (r < l)

{

while (array[r] < mid)

r++;

while (array[l] > mid)

l--;

if (r <= l)

{

swap(array[r], array[l]);

r++;

l--;

}

}

if (begin < l)

quickSort(array, l, begin);

if (r < finish)

quickSort(array, finish, r);

}

// #3

int mxNonSorted(int \*array, int length)

{

int mx = array[0];

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

{

if (array[i] > mx)

mx = array[i];

}

return mx;

}

int mnNonSorted(int \*array, int length)

{

int mn = array[0];

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

{

if (array[i] < mn)

mn = array[i];

}

return mn;

}

int mxSorted(int \*array, int length)

{

return array[length-1];

}

int mnSorted(int \*array, int length)

{

return array[0];

}

void mxMnNonSorted(int \*array, int length)

{

int mxA = array[0];

int mnA = array[0];

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

{

if (array[i] < mnA)

mnA = array[i];

if (array[i] > mxA)

mxA = array[i];

}

cout << mxA << ' ' << mnA << endl;

}

void mxMnSorted(int \*array, int length)

{

int mxA = array[length-1];

int mnA = array[0];

//cout << mxA << ' ' << mnA;

}

int round(float number)

{

if (abs(number) - int(abs(number)) >= 0.5)

{

if (number > 0)

{

return int(number)+1;

}

return int(number)-1;

}

else

return int(number);

}

// search

int count(int \*array, int length, int aim)

{

int counter = 0;

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

{

if (array[i] == aim)

{

counter += 1;

}

}

return counter;

}

int searchD(int \*array, int length, int aim)

{

int r = -1;

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

{

if (array[i] == aim)

{

r = i;

break;

}

}

return r;

}

int binarySearch(int \*array, int length, int aim)

{

int start = 0;

int end = length-1;

while (start <= end)

{

int mid = (start + end) / 2;

if (aim == array[mid])

return mid;

else if (aim > array[mid])

start = mid + 1;

else end = mid - 1;

}

return -1;

}

int binaryCount(int \*array, int length, int aim)

{

int counter = 0;

int begin = binarySearch(array, length, aim);

for (int i = begin; i < length && (array[i] != aim); ++i, counter += 1);

for (int i = begin-1; i >= 0; --i, counter += 1);

return counter;

}

// #8

void swapByIndecies(int \*array, int length)

{

cout << "До обмена:" << endl;

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

{

cout << array[i] << ' ';

}

cout << endl;

int n1, n2;

cout << "Индексы для обмена:" << endl;

cin >> n1 >> n2;

swap(array[n1], array[n2]);

cout << "После обмена:" << endl;

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

{

cout << array[i] << ' ';

}

cout << endl;

}

int main()

{

setlocale(LC\_ALL, "Russian");

srand(time(0));

const int N = 100;

int a[N];

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

{

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

}

int b[N];

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

{

b[i] = a[i];

}

while (true)

{

cout << "Выбeрете пункт программы: " << endl

<< "1. Получить случайный массив" << endl

<< "2. Получить отсортированный массив и время на его сортировку" << endl

<< "3. Получить мин/макс эл-ты массива, время их поиска в неотсорт/отсорт массиве" << endl

<< "4. Получить среднее значение элемента массива, индексы эл-т равных среднему, их кол-во" << endl

<< "5. Получить кол-во элементов, меньших заданного значения" << endl

<< "6. Получить кол-во элементов, больших заданного значения" << endl

<< "7. Проверить наличие числа в массиве перебором и бинарным поиском" << endl

<< "8. Поменять выбранные эл-ты массива местами по индексам" << endl

<< "9. ИДЗ #9" << endl

<< "Нажмите Ctrl+c для выхода из программы" << endl;

int choice;

cin >> choice;

switch (choice)

{

case 1:

cout << "Случайно сгенерированный массив:" << endl;

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

{

cout << b[i] << ' ';

}

break;

case 2:

{

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

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

{

cout << b[i] << ' ';

}

cout << endl;

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

int finish = N - 1;

int begin = 0;

quickSort(a, finish, begin);

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

chrono::duration<float> duration = end - start;

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

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

{

cout << a[i] << ' ';

}

cout << endl;

cout << "Время выполнения сортировки: " << (duration.count() \* 1000) << " мс";

}

break;

case 3:

{

cout << "Максимальный и минимальный элементы: ";

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

mxMnNonSorted(b, N);

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

chrono::duration<float> duration = end - start;

cout << "Время поиска мин/макс в неотсортированном массиве: " << (duration.count() \* 1000) << " мс" << endl;

int finish = N - 1;

int begin = 0;

quickSort(a, finish, begin);

start = chrono::high\_resolution\_clock::now();

mxMnSorted(a, N);

end = chrono::high\_resolution\_clock::now();

chrono::duration<float> duration1 = end - start;

cout << "Время поиска мин/макс в отсортированном массиве: " << (duration1.count() \* 1000) << " мс";

}

break;

case 4:

{

int mx = mxNonSorted(a, N);

int mn = mnNonSorted(a, N);

int average = round((float(mx) + float(mn))) / 2.0;

int averageCount = 0;

cout << "Среднее значение: " << average << endl;

cout << "Индексы элементов равных среднему значению: " << endl;

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

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

{

if (b[i] == average)

{

cout << i << ' ';

averageCount += 1;

}

}

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

chrono::duration<float> duration = end - start;

cout << endl;

cout << "Их количество: "<< averageCount << endl;

cout << "Время поиска в неотсорт. массиве: " << (duration.count() \* 1000) << " мс" << endl;

cout << "Время поиска в отсорт. массиве с использованием бин. поиска: ";

int finish = N - 1;

int begin = 0;

quickSort(a, finish, begin);

start = chrono::high\_resolution\_clock::now();

binaryCount(a, N, average);

end = chrono::high\_resolution\_clock::now();

chrono::duration<float> duration1 = end - start;

cout << (duration1.count() \* 1000) << " мс";

}

break;

case 5:

{

int finish = N - 1;

int begin = 0;

quickSort(a, finish, begin);

int lessNumber = 0;

int number;

cin >> number;

int start = searchD(a, N, number);

for (int i = start - 1; i >= 0; --i)

{

if (a[i] < number)

lessNumber += 1;

}

cout << lessNumber;

}

break;

case 6:

{

int finish = N - 1;

int begin = 0;

quickSort(a, finish, begin);

int greaterNumber = 0;

int number;

cin >> number;

int start = searchD(a, N, number);

for (int i = start + 1; i < N; ++i)

{

if (a[i] > number)

greaterNumber += 1;

}

cout << greaterNumber ;

}

break;

case 7:

{

cout << "Введите число для поиска: " << endl;

int number;

cin >> number;

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

int result = searchD(b, N, number);

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

chrono::duration<float> duration = end - start;

cout << "Время поиска перебором: " << (duration.count() \* 1000) << " мс" << endl;

int finish = N - 1;

int begin = 0;

quickSort(a, finish, begin);

start = chrono::high\_resolution\_clock::now();

result = binarySearch(a, N, number);

end = chrono::high\_resolution\_clock::now();

chrono::duration<float> duration1 = end - start;

cout << "Время поиска бинарным поиском: " << (duration1.count() \* 1000) << " мс" << endl;

cout << "Бинарный поиск быстрее в " << duration.count()/duration1.count() << " разa" << endl;

if (result != -1)

cout << "Есть такой элемент, его индекс = " << result;

}

break;

case 8:

{

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

swapByIndecies(a, N);

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

chrono::duration<float> duration = end - start;

cout << "Время обмена: " << (duration.count() \* 1000) << " мс";

break;

}

case 9:

{

int value;

cout << "Ваше число: ";

cin >> value;

srand(time(0));

int counter = 0;

for (int i = 1; i < N; i += 2)

{

a[i] -= value;

a[i] \*= (rand() % 9 + 1);

}

int del[9] = {0, 0, 0, 0, 0, 0, 0, 0, 0};

for (int i = 1; i <= 9; ++i)

{

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

{

if (a[j] % i == 0)

del[i-1] += 1;

}

}

for (int i = 0; i < 9; ++i)

{

cout << del[i] << " чисел делится на " << i+1 << endl;

}

break;

}

default:

cout << "Введено неверное значение";

}

cout << endl;

}

return 0;

}