**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. 4373 |  | Лакалин С.Н. |
| Преподаватель |  | Глущенко А. Г. |

Санкт-Петербург

2024

**Цель работы.**

Использование сортировки массива для удобного и быстрого поиска различных значений

**Основные теоретические положения.**

**Массив**

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: aiai​ при  i = 0, 1, 2, … n – одномерный вектор. Каждый элемент такой последовательности представляет собой некоторое значение определенного типа данных.

**Сортировка**

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.

### ****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.

## Бинарный поиск

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.**

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.

**Выводы:**

Успешно удалось воспользоваться алгоритмами сортировки и быстрым поиском, чтобы выполнить различные задания.

Приложение А

рабочий код

// Практика 2.cpp : Этот файл содержит функцию "main". Здесь начинается и заканчивается выполнение программы.

//

#include <cstdlib>

#include <ctime>

#include <iostream>

#include <iomanip>

#include <chrono>

#include <cmath>

#include <algorithm>

using namespace std;

using namespace chrono;

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

{

int mid;

int f = begin;

int l = end;

mid = arr[(f + l) / 2];

while (f < l)

{

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

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

if (f <= l)

{

swap(arr[f], arr[l]);

f++;

l--;

}

}

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

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

}

int binarySearch(int arr[], int value, int start, int end) {

if (end >= start) {

int mid = start + (end - start) / 2;

if (arr[mid] == value) {

return mid;

}

if (arr[mid] > value) {

return binarySearch(arr, value, start, mid - 1);

}

return binarySearch(arr, value, mid + 1, end);

}

return -1;

}

void pairInsertionSort2(int arr[], int n) {

for (int i = 1; i < n; i += 2) {

if (arr[i] < arr[i - 1]) {

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

}

}

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

int j = i;

while (j > 0 && arr[j] < arr[j - 1]) {

swap(arr[j], arr[j - 1]);

j--;

}

}

}

void pairInsertionSort(int arr[], int n) {

int left = 0;

int right = n - 1;

for (int k = left; ++left <= right; k = ++left) {

//Очередную пару рядом стоя́щих элементов

//заносим в пару буферных переменных

int a1 = arr[k], a2 = arr[left];

if (a1 < a2) {

a2 = a1; a1 = arr[left];

}

//Вставляем больший элемент из пары

while (a1 < arr[--k]) {

arr[k + 2] = arr[k];

}

arr[++k + 1] = a1;

//Вставляем меньший элемент из пары

while (a2 < arr[--k]) {

arr[k + 1] = arr[k];

}

arr[k + 1] = a2;

}

int last = arr[right];

while (last < arr[--right]) {

arr[right + 1] = arr[right];

}

arr[right + 1] = last;

}

void insertionSort(int arr[], int n) {

int i, key, j;

for (i = 1; i < n; i++) {

key = arr[i];

j = i - 1;

// Сдвигаем элементы массива, которые больше key, на одну позицию вперед

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

arr[j + 1] = arr[j];

j = j - 1;

}

arr[j + 1] = key;

}

}

int main()

{

setlocale(LC\_ALL, "Russian");

int N = 100;

int seed = time(NULL);

int max, min;

int a, b, lh=1;

int ne[100], ot[100], kv[100];

while (true) { // Цикл для многократного запуска пунктов

if (lh == 1) {

cout << "\nВыберите пункт для выполнения \n\n"

<< "0) Выход из программы.\n\n"

<< "1) Создает целочисленный массив размерности N = 100.\n\n"

<< "2) Cортирует заданный в пункте 1 массив[…] сортировкой Quick sort.\n\n"

<< "3) Находит максимальный и минимальный элемент массива.\n\n"

<< "4) Выводит среднее значение(если необходимо, число нужно округлить) максимального и минимального значения в отсортированном и неотсортированном.\n\n"

<< "5) Выводит количество элементов в отсортированном массиве, которые меньше числа a, которое инициализируется пользователем.\n\n"

<< "6) Выводит количество элементов в отсортированном массиве, которые больше числа b, которое инициализируется пользователем.\n\n"

<< "7) Выводит информацию о том, есть ли введенное пользователем число в отсортированном массиве.\n\n"

<< "8) Меняет местами элементы массива, индексы которых вводит пользователь.Выведите скорость обмена, используя библиотеку chrono.\n\n"

<< "9) Cортирует заданный в пункте 1 массив[…] сортировкой парными вставками.\n\n"

<< "10) Выводит отсортированный массив.\n";

lh = 0;

}

int choice;

cin >> choice;

if (choice == 0) {

return 0;

}

srand(seed); // Переместить srand внутрь цикла, но вне генерации массива

// чтобы каждый прогон использовал новую последовательность

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

nanoseconds result;

switch (choice) {

case 1: {

//1) Массив размера 100

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

{

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

ot[i] = ne[i];

}

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

if (i % 10 == 0 and i != 0) { cout << endl; }

cout << setw(4) << ne[i] << " ";

}

cout << endl << endl;

} break;

case 2: {

//2) Сортировка массива

int en = N - 1, beg = 0;

start = steady\_clock::now();

quicksort(ot, en, beg);

end = steady\_clock::now();

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

cout << "Время сортировки массива: " << result.count() << endl << endl;

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

if (i % 10 == 0 and i != 0) { cout << endl; }

cout << setw(4) << ot[i] << " ";

}

cout << endl << endl;

} break;

case 3: {

//3) Поиск max и min

//В неотсортированном

start = steady\_clock::now();

max = ne[0]; min = ne[0];

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

{

if (ne[i] > max) {

max = ne[i];

}

if (ne[i] < min) {

min = ne[i];

}

}

end = steady\_clock::now();

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

cout << "Время поиска max и min в неотсортированном массиве: " << result.count() << endl;

cout << "max и min: " << max << " " << min << endl << endl;

//В отсортированном

start = steady\_clock::now();

min = ot[0]; max = ot[N - 1];

end = steady\_clock::now();

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

cout << "Время поиска max и min в отсортированном массиве: " << result.count() << endl;

cout << "max и min: " << max << " " << min << endl << endl;

} break;

case 4: {

//4) Поиск среднего max и min

int average = round((double)(max + min) / 2);

cout << "Среднее max и min: " << average << endl;

cout << "Индексы элементов: ";

// В неотсортированном массиве

start = steady\_clock::now();

int count\_unsorted = 0;

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

if (ne[i] == average) {

cout << i << " ";

count\_unsorted++;

}

}

end = steady\_clock::now();

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

cout << endl << endl;

cout << "Время поиска среднего в неотсортированном массиве: " << result.count() << endl;

cout << "Количество элементов равных среднему в неотсортированном массиве: " << count\_unsorted << endl;

cout << "Индексы элементов: ";

// В отсортированном массиве

start = steady\_clock::now();

int count\_sorted = 0;

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

if (ot[i] == average) {

cout << i << " ";

count\_sorted++;

}

else {

if (ot[i] < average) { break; }

}

}

end = steady\_clock::now();

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

cout << endl << endl;

cout << "Время поиска среднего в отсортированном массиве: " << result.count() << endl;

cout << "Количество элементов равных среднему в отсортированном массиве: " << count\_sorted << endl;

} break;

case 5: {

//5) Поиск меньших, чем а

cout << "Введите число a: ";

cin >> a;

int count\_a = 0;

int u=0;

while (u < N and ot[u] < a) {

count\_a++;

u++;

}

cout << "Количество элементов меньших заданного числа а: " << count\_a << endl;

} break;

case 6: {

//6)

cout << "Введите число b: ";

cin >> b;

int count\_b = 0;

int u = N - 1;

while (u >= 0 and ot[u] > b) {

count\_b++;

u--;

}

cout << "Количество элементов больших заданного числа b: " << count\_b << endl;

} break;

case 7: {

//7) Поиск элемента

cout << "Введите число x: ";

int x, xi = -1;

cin >> x;

cout << "Индекс элемента ";

start = steady\_clock::now();

binarySearch(ot, x, 0, N - 1);

end = steady\_clock::now();

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

cout << binarySearch(ot, x, 0, N - 1) << endl;

cout << "Время поиска элемента в отсортированном массиве: " << result.count() << endl;

cout << "Введите число x: ";

cin >> x;

start = steady\_clock::now();

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

if (ot[i] == x) {

xi = i;

break;

}

}

cout << "Индекс элемента " << xi << endl;

end = steady\_clock::now();

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

cout << "Время поиска элемента в неотсортированном массиве: " << result.count() << endl;

} break;

case 8: {

//8) Обмен элементов

int i1, i2;

cout << "Введите индексы элементов, которые хотите поменять местами: " << endl;

while (true) {

cin >> i1 >> i2;

if ((i1 >= 0 and i1 < N) and (i2 >= 0 and i2 < N)) {

break;

}

cout << "Неправильные индексы. Введите индексы от 0 до " << N - 1 << ":" << endl;

}

start = steady\_clock::now();

swap(ot[i1], ot[i2]);

end = steady\_clock::now();

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

cout << "Время обмена элементов: " << result.count() << endl;

} break;

case 9: { // Новый пункт меню для pairInsertionSort

start = steady\_clock::now();

pairInsertionSort(ot, N);

end = steady\_clock::now();

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

cout << "Время сортировки парными вставками: " << result.count() << endl;

} break;

case 10: {

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

if (i % 10 == 0 and i != 0) { cout << endl; }

cout << setw(4) << ot[i] << " ";

}

}break;

default:

cout << "Неверный выбор пункта." << endl;

}

seed = time(NULL);

}

return 0;

}

|  |  |
| --- | --- |
| Ввод с клавиатуры | Работа алгоритма |
|  | |
|  | Выводится меню выбора того, что хочет сделать пользователь. |
| Если введена 1 | Программа создает массив и выводит его на экран. |
| Если введена 2 или 9 | Программа сортирует массив и выводит его на экран, а так же выводит время затраченное на сортировку в наносекундах. |
| Если введена 3 | Программа ищет минимальный и максимальный элемент в массиве. |
| Если введена 4 | Программа считает среднее максимального и минимального элементов, выводит время поиска, колличество совпадающих элементов в массиве и их индексы. |
| Если введена 5, а затем целое число | Программа ищет элементы меньше, чем введенное число, и выводит их колличество. |
| Если введена 6, а затем целое число | Программа ищет элементы больше, чем введенное число, и выводит их колличество. |
| Если введена 7, а затем 2 целых числа | Программа, используя алгоритм бинарного поиска, ищет заданное число. Затем это же число ищется в неотсортированном массиве. Выводится индекс числа и время его поиска. |
| Если введена 8, а затем 2 целых числа | Программа меняет элементы под индексами, которые задал пользователь, а затем выводит время обмена на экран. |
| Если введена 10 | Программа выводит отсортированный массив, с учетом манипуляций пункта 8 |