**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 |  | Кипень В. А. |
| Преподаватель |  | Глущенко А. Г. |

Санкт-Петербург

2023

**Цель работы.**

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.

**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.

Внутреннее представление величин целого типа – целое число в двоичном коде. При использовании спецификатора signed старший бит числа интерпретируется как знаковый (0 – положительное число, 1 – отрицательное). Для кодирования целых чисел со знаком применяется прямой, обратный и дополнительный коды.

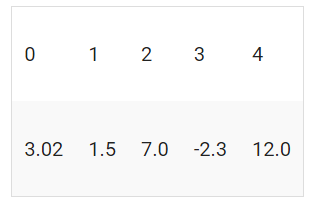


Рисунок 1 – одномерный массив из 5 float-элементов

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.

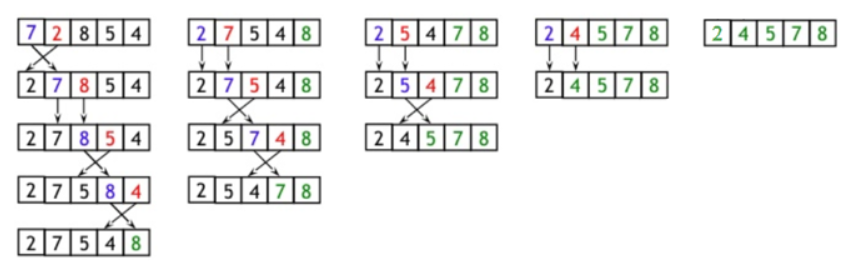


Рисунок 2 – Визуальная интерпретация пузырьковой сортировки

Быстрая сортировка – 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.

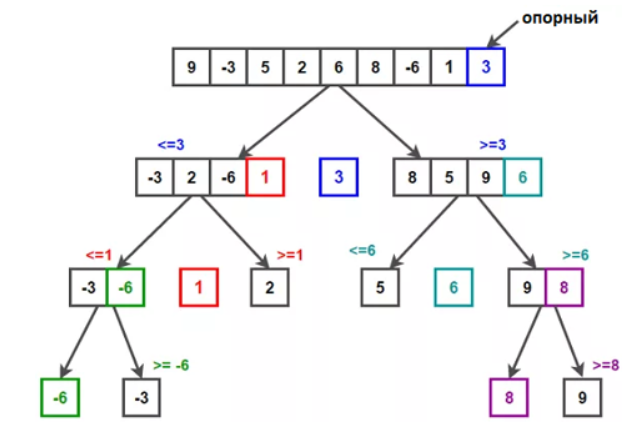


Рисунок 3 - Визуальная интерпретация быстрой сортировки

**Постановка задачи.**

Необходимо 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.**

|  |  |
| --- | --- |
| Ввод пользователем и обработка данных | Работа алгоритма и вывод на экран |
| Меню | |
| При запуске программы перед пользователем появляется окно с меню, где он может выбрать пункт, который требуется выполнить | Меню:    Проверка на ввод символов, которые не входят в диапазон выбора: |

|  |  |
| --- | --- |
| Инициализация массива | |
| При вводе пользователем корректного значения пункта меню (1) изначально происходит инициализация массива, длиной 100 элементов и диапазоном значений от -99 до 99 | Как только все данные заданы, на экран выводится двоичное представление введённого десятичного числа, а также появляется возможность завершить или продолжить выполнение программы. |
| Сортировка массива | |
| При вводе пользователем корректного значения пункта меню (2) происходит сортировка массива по возрастанию (алгоритм быстрой сортировки), вывод дополнительной информации и (опционально) вывод получившегося массива |  |
| Нахождение минимального и максимального элементов массива | |
| При вводе пользователем корректного значения пункта меню (3) происходит поиск минимального и максимального элементов в отсортированном и неотсортированном массивах, вывод дополнительной информации и (опционально) вывод получившегося массива |  |
| Нахождение среднего значений элементов массива | |
| При вводе пользователем корректного значения пункта меню (4) происходит вычисление среднего в отсортированном и неотсортированном массивах, вывод дополнительной информации и (опционально) вывод получившегося массива |  |
| Количество элементов, меньших/больших чем заданное значение | |
| При вводе пользователем корректного значения пункта меню (5/6) происходит вычисление количества элементов, меньших/больших чем заданное, вывод дополнительной информации и (опционально) вывод получившегося массива |  |
| Поиск в массиве методом бинарного поиска и перебора | |
| При вводе пользователем корректного значения пункта меню (7) происходит поиск заданного значения в отсортированном массиве, вывод дополнительной информации и (опционально) вывод получившегося массива |  |
| Перестановка элементов массива на заданных индексах | |
| При вводе пользователем корректного значения пункта меню (8) происходит перестановка двух элементов, по заданным индексам, вывод дополнительной информации и (опционально) вывод получившегося массива |  |

**Выводы.**

Была создана программа, в которой реализован следующий функционал:

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.

Приложение А

рабочий код

#include<iostream>

#include <chrono>

#include <unistd.h>

using namespace std;

int getRandomValueFromRange(int leftLimit, int rightLimit) {

    return leftLimit + rand() % (rightLimit - leftLimit + 1);

}

void fillArray(int arr[], const int N, int leftLimit, int rightLimit) {

    srand(time(0));

    for (int i = 0; i < N; i++) {

        arr[i] = getRandomValueFromRange(-99, 99);

    }

}

void findArrayMinMax(int arr[], const int N, int &mn, int &mx) {

    mn = arr[0];

    mx = arr[0];

    for (int i = 0; i < N; i++) {

        mn = min(mn, arr[i]);

        mx = max(mx, arr[i]);

    }

}

int findArrayAverage(int arr[], const int N) {

    int mn, mx;

    findArrayMinMax(arr, N, mn, mx);

    return (mx + mn) / 2;

}

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;

}

int defaultSearch(int arr[], int value, const int N) {

    for (int i = 0; i < N; i++) {

        if (arr[i] == value) return i;

    }

    return -1;

}

void bubbleSort(int arr[], const int N) {

    for (int i = 0; i < N - 1; i++) {

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

            int swap = arr[j];

            if (arr[j] > arr[j + 1]) {

                arr[j] = arr[j + 1];

                arr[j + 1] = swap;

            }

        }

    }

}

void shakerSort(int arr[], const int N) {

    for (int i = 0; i < N / 2; i++) {

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

            int swap = arr[j];

            if (arr[j] > arr[j + 1]) {

                arr[j] = arr[j + 1];

                arr[j + 1] = swap;

            }

        }

        for (int j = N - i - 2; j > i; j--) {

            int swap = arr[j];

            if (arr[j - 1] > arr[j]) {

                arr[j] = arr[j - 1];

                arr[j - 1] = swap;

            }

        }

    }

}

void combSort(int arr[], const int N) {

    float k = 1.247;

    float S = N - 1;

    int count = 0;

    while (S >= 1)

    {

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

        {

            if (arr[i] > arr[int(i + S)])

            {

                int swap = arr[int(i + S)];

                arr[int(i + S)] = arr[i];

                arr[i] = swap;

            }

        }

        S /= k;

    }

    bubbleSort(arr, N);

}

void insertSort(int arr[], const int N) {

    int i, key, j;

    for (i = 1; i < N; i++) {

        key = arr[i];

        j = i - 1;

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

            arr[j + 1] = arr[j];

            j = j - 1;

        }

        arr[j + 1] = key;

    }

}

void quickSort(int arr[], int start, int end)

{

    int mid;

    int f = start;

    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 (start < l) quickSort(arr, start, l);

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

}

int main() {

    setlocale(LC\_ALL, "Russian");

    const int N = 100;

    int arr[N] = {};

    cout << "Navigation:" << "\n"

    << "1) Create an integer array of size N = 100. The elements of the array must take a random value in the range from -99 to 99" << "\n"

    << "2) Sort the array (from smallest to largest)" << "\n"

    << "3) Find the max and min element of the array. Calculate the search time for these elements in sorted array and unsorted" << "\n"

    << "4) Prints the average of max and min values in sorted and unsorted arrays. Prints the indexes of all elements that are equal to this value and their number" << "\n"

    << "5) Prints the number of elements in sorted array that are less then number <a>" << "\n"

    << "6) Prints the number of elements in sorted array that are greater than the number <b>" << "\n"

    << "7) Displays information about whether the <number> in sorted array. And compare binary search with regular search alorithm (\*)" << "\n"

    << "8) Swaps array elements on <indexes>" << "\n"

    << "9) IDZ №5" << "\n";

    while(true) {

        cin.clear(); *// Clearing the input stream from possible errors*

        cin.sync();

        short int workPoint;

        cout << "Select point of work (number 1 to 8): ";

        cin >> workPoint;

        switch (workPoint)

        {

            case 1: {

                fillArray(arr, N, -99, 99);

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

                    cout << arr[i] << " ";

                }

                break;

            }

            case 2: {

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

                quickSort(arr, 0, N - 1);

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

                cout << "Quick sort time: " << chrono::duration\_cast<chrono::microseconds>(end - start).count() << " mcs" << "\n";

                break;

            }

            case 3: {

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

                int mn, mx;

                findArrayMinMax(arr, N, mn, mx);

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

                cout << "Unsorted: Min " << mn << ", Max " << mx << "\n"

                << "Search time: " << chrono::duration\_cast<chrono::microseconds>(end - start).count() << " mcs" << "\n";

                quickSort(arr, 0, N - 1);

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

                cout << "\n" << "Sorted: Min " << arr[0] << ", Max " << arr[N - 1] << "\n";

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

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

                break;

            }

            case 4: {

                int average = findArrayAverage(arr, N);

                cout << "Unsorted average: " << average << "\n";

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

                int count = 0;

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

                    if (arr[i] == average) {

                        cout << i << " ";

                        count++;

                    }

                }

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

                if (count != 0) cout << "\n";

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

                cout << "Count of elements: "<< count << "\n";

                quickSort(arr, 0, N - 1);

                average = (arr[0] + arr[N - 1]) / 2;

                cout << "\n" << "Sorted average: " << average << "\n";

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

                count= 0;

                int index = binarySearch(arr, average, 0, N - 1);

                for (int i = index - 1; i > -1; i--) {

                    if (arr[i] == average) {

                        cout << i << " ";

                        count++;

                    }

                    else break;

                }

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

                    if (arr[i] == average) {

                        cout << i << " ";

                        count++;

                    }

                    else break;

                }

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

                if (count != 0) cout << "\n";

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

                cout << "Count of elements: "<< count;

                break;

            }

            case 5: {

                int a;

                cout << "\n" << "Enter an integer: ";

                cin >> a;

                if (!cin.good()) {

                    cout << "\n" << "You entered an incorrect value";

                    break;

                }

                quickSort(arr, 0, N - 1);

                if (a > arr[(N - 1) / 2]) {

                    for (int i = N - 1; i >= 0; i--) {

                        if (arr[i] < a) {

                            cout << i + 1;

                            break;

                        }

                    }

                } else if (a < arr[(N - 1) / 2]) {

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

                        if (arr[i] >= a) {

                            cout << i;

                            break;

                        }

                    }

                } else cout << arr[(N - 1) / 2];

                break;

            }

            case 6: {

                int b;

                cout << "\n" << "Enter an integer: ";

                cin >> b;

                if (!cin.good()) {

                    cout << "\n" << "You entered an incorrect value";

                    break;

                }

                quickSort(arr, 0, N - 1);

                if (b > arr[(N - 1) / 2]) {

                    for (int i = N - 1; i >= 0; i--) {

                        if (arr[i] <= b) {

                            cout << N - i - 1;

                            break;

                        }

                    }

                } else if (b < arr[(N - 1) / 2]) {

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

                        if (arr[i] > b) {

                            cout << N - i;

                            break;

                        }

                    }

                } else cout << arr[(N - 1) / 2];

                break;

            }

            case 7: {

                int number;

                cout << "\n" << "Enter an integer: ";

                cin >> number;

                if (!cin.good()) {

                    cout << "\n" << "You entered an incorrect value";

                    break;

                }

                quickSort(arr, 0, N - 1);

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

                cout << "Binary: ";

                if (binarySearch(arr, number, 0, N - 1) != -1) cout << "true";

                else cout << "false";

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

                cout << ", search time: " << chrono::duration\_cast<chrono::microseconds>(end - start).count() << " mcs" << "\n";

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

                cout << "Default: ";

                if (defaultSearch(arr, number, N) != -1) cout << "true";

                else cout << "false";

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

                cout << ", search time: " << chrono::duration\_cast<chrono::microseconds>(end - start).count() << " mcs" << "\n";

                break;

            }

            case 8: {

                int firstIndex, secondIndex;

                cout << "\n" << "Enter an 2 integers (array indexes): ";

                cin >> firstIndex;

                cin >> secondIndex;

                if (!cin.good()) {

                    cout << "\n" << "You entered an incorrect value";

                    break;

                }

                if (firstIndex < 0 || secondIndex < 0 || firstIndex >= N || secondIndex >= N) {

                    cout << "\n" << "You entered incorrect indexes";

                    break;

                }

                swap(arr[firstIndex], arr[secondIndex]);

                break;

            }

            case 9: {

                int number;

                cout << "\n" << "Enter an integer: ";

                cin >> number;

                if (!cin.good()) {

                    cout << "\n" << "You entered an incorrect value";

                    break;

                }

                int count = 0;

                int randomValue = getRandomValueFromRange(1, 9);

                cout << "Calculated random value: " << randomValue << "\n";

                cout << "Number of even elements with even values: ";

                for (int i = 0; i < N; i += 2) {

                    arr[i] -= number;

                    arr[i] \*= randomValue;

                    if (arr[i] % 2 == 0) count++;

                }

                cout << count << "\n";

                count = 0;

                cout << "Number of odd elements with odd values: ";

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

                    if (arr[i] % 2 != 0) count++;

                }

                cout << count << "\n";

                for (int i = 1; i < 10; i++) {

                    count = 0;

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

                        if (arr[j] % i == 0) {

                            count++;

                        }

                    }

                    cout << "Number of elements divisible by " << i << ": " << count << "\n";

                }

                break;

            }

            default: {

                cout << "\n" << "You did not enter a number in the range from 1 to 8";

                break;

            }

        }

        cin.clear(); *// Clearing the input stream from possible errors*

        cin.sync();

        char printArray;

        cout << "\n" << "Print the result array? (Y/N) ";

        cin >> printArray;

        if (printArray == 'Y' || printArray == 'y') {

            for (int i = 0; i < N; i++) cout << arr[i] << " ";

        }

        cin.clear();

        cin.sync();

        char stopFlag;

        cout << "\n" << "Continue the program? (Y/N) ";

        cin >> stopFlag;

        if (stopFlag != 'Y' && stopFlag != 'y') {

            break;

        }

    }

    return 0;

}