МИНИСТЕРСТВО ЦИФРОВОГО РАЗФИТИЯ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ “СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ”

Кафедра вычислительных систем

# **Курсовая работа**

по дисциплине “Технологии разработки программного обеспечения”

на тему “Сортировка входных данных”

Выполнил:

гр. ИП-211

Панчина В.С.

Проверил:

ст. преподаватель Токмашева Е. И.

Новосибирск, 2023 г

# **Содержание**

Содержание…………………………………………………………………………………………………………………………………. 2

Введение и постановки задачи……………………………………………………………………………………………………. 3

Техническое задание……………………………………………………………………………………………………………………. 4

Описание проекта…………………………………………………………………………………………………………………………. 6

Личный вклад в проект…………………………………………………………………………………………………………………. 8

[Приложение. Текст программы 2](#_Toc757985360)

# **Введение и постановка задачи**

Цель работы: продемонстрировать работу алгоритмов сортировки, используя различные размеры входных данных.

Задачи: сравнить по времени работу всех представленных сортировок и выявить, наиболее быструю из них и наиболее медленную. Вывести результаты работы сортировок на экран (в секундах).

# **Техническое задание**

1. **Описание функциональности:**

Продукт решает задачу исследования производительности различных алгоритмов сортировки на разных размерах входного набора данных. Пользователь может выбрать один из нескольких алгоритмов сортировки, таких как selection, bubble, insertion, merge, quick. Затем пользователь задает размер входного набора данных, и продукт запускает выбранный алгоритм на этом наборе данных и измеряет время выполнения. Результаты измерений выводятся на экран.

1. **Сценарии использования:**

1) Исследование производительности алгоритмов сортировки на различных размерах входных данных. Пользователь запускает приложение и выбирает один из нескольких алгоритмов сортировки. Затем пользователь задает размер входного набора данных. Продукт запускает выбранный алгоритм на этом наборе данных и измеряет время выполнения. Результаты измерений выводятся на экран.

1. **Формат входных данных:**

Пользователь задает размер входного набора данных в виде целого числа.

1. **Интерфейс приложения:**

Приложение работает в интерактивном режиме. Пользователю предоставляется меню, в котором он может выбрать один из нескольких алгоритмов сортировки и задать размер входного набора данных. Продукт запускает выбранный алгоритм на этом наборе данных и измеряет время выполнения. Результаты измерений выводятся на экран.

# **Описание работы проекта**

Проект представляет из себя программу, написанную на языке программирования “с++”. Пользователю предлагается выбор одного из пяти видов сортировки. После выбора сортировки пользователь вводит размер входных данных, не превышающих 100 000 символов. Затем программа выводит результат времени сортировки (в секундах).

**Пример:**

Выберите вид сортировки:

1. Сортировка выбором
2. Пузырьковая сортировка
3. Сортировка вставками
4. Сортировка слиянием
5. Быстрая сортировка

Пользователь: вводит 1

Введите длину массива

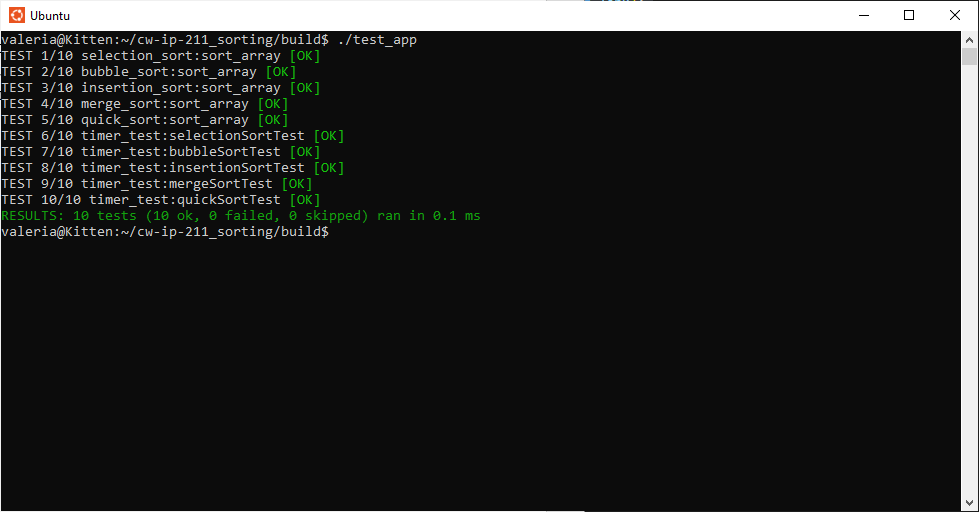
Пользователь: вводит 69 000

Программа: выводит 2.961 с.

# **Личный вклад в проект**

Мой вклад в проект заключается в написании кода на С++ для реализации основной части проекта (исследование производительности алгоритмов сортировок). Для анализа я взяла такие сортировки как selection, bubble, insertion, merge, quick.

Так же в часть моей работы входила сборка самого приложения (создание структуры проекта, сборка приложения с помощью CMake и реализация функциональности), модульное тестирование приложения.



# **Приложение. Текст программы**

# **Main.cpp**

#include <iostream>

#include <ctime>

#include <random>

#include "sorts.cpp"

#include <stdio.h>

using namespace std;

int main()

{

    setlocale(LC\_ALL, "rus");

    int m[100000];

    long long choice, len;

    double time;

    while (true)

    {

        cout << "Select the sort type:\n(0) exit\n(1) Sorting by choice\n(2) BubbleSort\n(3) InsertSort\n(4) MergeSort\n(5) QuickSort\n";

        cin >> choice;

        if (cin.fail() || choice < 0 || choice > 5)

        {

            cout << "Invalid input. Please enter a number between 0 and 5\n";

            cin.clear();

            cin.ignore(numeric\_limits<streamsize>::max(), '\n');

        }

        else

        {

            break;

        }

    }

    while (true)

    {

        cout << "Write the length of the array under test:\n";

        cin >> len;

        if (cin.fail() || len < 1 || len > 100000)

    {

        cout << "Invalid input. Please enter a number between 1 and 100000\n";

        cin.clear();

        cin.ignore(numeric\_limits<streamsize>::max(), '\n');

    }

        else

        {

            break;

        }

    }

    while (true)

    {

        cout << "Select the sort type:\n(0) exit\n(1) Sorting by choice\n(2) BubbleSort\n(3) InsertSort\n(4) MergeSort\n(5) QuickSort\n";

        cin >> choice;

        if (choice == 0)

        return 0;

        cout << "Write the length of the array under test:\n";// выбор длины тестируемого массива

        cin >> len;

        for (long long i = 0; i < len; i++)

        m[i] = rand();

        cout << timer(choice, m, len) << "s\n";

    }

}

# **sorts.cpp**

#include <string>

#include <stdio.h>

#include "sorts.h"

void selectionSort(int data[], int lenD)

{

    int j = 0;

    int tmp = 0;

    for (int i = 0; i < lenD; i++)

    {

        j = i;

        for (int k = i; k < lenD; k++)

        {

            if (data[j] > data[k])

            {

                j = k;

            }

        }

        tmp = data[i];

        data[i] = data[j];

        data[j] = tmp;

    }

}

void bubbleSort(int data[], int lenD)

{

    int tmp = 0;

    for (int i = 0; i < lenD; i++)

    {

        for (int j = (lenD - 1); j >= (i + 1); j--)

        {

            if (data[j] < data[j - 1])

            {

                tmp = data[j];

                data[j] = data[j - 1];

                data[j - 1] = tmp;

            }

        }

    }

}

void insertionSort(int data[], int lenD)

{

    int key = 0;

    int i = 0;

    for (int j = 1; j < lenD; j++)

    {

        key = data[j];

        i = j - 1;

        while (i >= 0 && data[i] > key)

        {

            data[i + 1] = data[i];

            i = i - 1;

            data[i + 1] = key;

        }

    }

}

void merge(int merged[], int lenD, int L[], int lenL, int R[], int lenR)

{

    int i = 0;

    int j = 0;

    while (i < lenL || j < lenR)

    {

        if (i < lenL && j < lenR)

        {

            if (L[i] <= R[j])

            {

                merged[i + j] = L[i];

                i++;

            }

            else

            {

                merged[i + j] = R[j];

                j++;

            }

        }

        else if (i < lenL)

        {

            merged[i + j] = L[i];

            i++;

        }

        else if (j < lenR)

        {

            merged[i + j] = R[j];

            j++;

        }

    }

}

void mergeSort(int data[], int lenD)

{

    if (lenD > 1)

    {

        int middle = lenD / 2;

        int rem = lenD - middle;

        int \*L = new int[middle];

        int \*R = new int[rem];

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

        {

            if (i < middle)

            {

                L[i] = data[i];

            }

            else

            {

                R[i - middle] = data[i];

            }

        }

        mergeSort(L, middle);

        mergeSort(R, rem);

        merge(data, lenD, L, middle, R, rem);

    }

}

void quickSort(int \*data, int const len)

{

    int const lenD = len;

    int pivot = 0;

    int ind = lenD / 2;

    int i, j = 0, k = 0;

    if (lenD > 1)

    {

        int \*L = new int[lenD];

        int \*R = new int[lenD];

        pivot = data[ind];

        for (i = 0; i < lenD; i++)

        {

            if (i != ind)

            {

                if (data[i] < pivot)

                {

                    L[j] = data[i];

                    j++;

                }

                else

                {

                    R[k] = data[i];

                    k++;

                }

            }

        }

        quickSort(L, j);

        quickSort(R, k);

        for (int cnt = 0; cnt < lenD; cnt++)

        {

            if (cnt < j)

            {

                data[cnt] = L[cnt];

                ;

            }

            else if (cnt == j)

            {

                data[cnt] = pivot;

            }

            else

            {

                data[cnt] = R[cnt - (j + 1)];

            }

        }

    }

}

double timer(int tip\_sort, int a[], long long n)

{

    unsigned int start = clock();

    if (tip\_sort == 1)

    {

        selectionSort(a, n);

    }

    else if (tip\_sort == 2)

    {

        bubbleSort(a, n);

    }

    else if (tip\_sort == 3)

    {

        insertionSort(a, n);

    }

    else if (tip\_sort == 4)

    {

        mergeSort(a, n);

    }

    else

    {

        quickSort(a, n);

    }

    return (clock() - start) / 1000.0;

}

**sorts.h**

#ifndef \_sorts\_H\_

#define \_sorts\_H\_

void selectionSort(int data[], int lenD);

void bubbleSort(int data[], int lenD);

void insertionSort(int data[], int lenD);

void merge(int merged[], int lenD, int L[], int lenL, int R[], int lenR);

void mergeSort(int data[], int lenD);

void quickSort(int\* data, int const len);

double timer(int tip\_sort, int a[], long long n);

#endif