**МИНИСТЕРСТВО ТРАНСПОРТА РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «РОССИЙСКИЙ УНИВЕРСИТЕТ ТРАНСПОРТА» (РУТ(МИИТ)**

Институт управления и цифровых технологий

Кафедра «Цифровые технологии управления транспортными процессами»

**КУРСОВАЯ РАБОТА**

**по дисциплине «Архитектура и программное обеспечение вычислительных систем»**

**на тему «Анализ и сравнение алгоритмов сортировки»»**

Выполнил:

студент 2 курса очной формы обучения

Яненко В. Д.

группа УПМ-211

Проверил:

канд. физ.-мат. наук, доц.

Соломатин А. Н.

Подпись \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# ВВЕДЕНИЕ

Сортировка данных является фундаментальным элементом в области информатики и программирования, играя ключевую роль в оптимизации производительности алгоритмов и программ. Решая задачу упорядочивания элементов набора данных, алгоритмы сортировки позволяют повысить эффективность поиска, доступа и обработки информации. Сортировка – это важная процедура, которая обеспечивает:

1. Упорядочивание данных: Сортировка позволяет упорядочивать наборы данных в заданном порядке. Это может быть порядок по возрастанию или убыванию числовых значений, алфавитный порядок для строк, или другой определенный порядок в зависимости от контекста задачи.
2. Улучшение производительности поиска: Отсортированные данные обеспечивают эффективный поиск элементов. Например, для отсортированного массива можно использовать бинарный поиск, который имеет логарифмическую сложность, в отличие от линейного поиска для неупорядоченного массива, который имеет линейную сложность.
3. Облегчение вставки и удаления: В некоторых случаях, особенно когда данные часто изменяются, сортировка может упростить операции вставки и удаления элементов. Например, при использовании сортировки вставками, новые элементы можно легко вставлять в уже отсортированный массив.
4. Подготовка данных для других алгоритмов: Отсортированные данные могут улучшить производительность других алгоритмов. Например, отсортированные данные могут быть более эффективными при использовании алгоритмов слияния или поиска в неупорядоченных структурах данных.
5. Повышение читаемости и визуальной наглядности: Отсортированные данные обычно более удобны для восприятия человеком. Это может быть важным аспектом, например, при отображении данных на веб-странице или в пользовательском интерфейсе.

Каждый из алгоритмов обладает своими характеристиками, преимуществами и ограничениями, что делает их использование зависимым от конкретных требований и характеристик задачи.

В рамках данной курсовой работы проведется анализ и сравнение нескольких известных алгоритмов сортировки, таких как сортировка вставками, пузырьковая сортировка, быстрая сортировка, сортировка выбором и сортировка кучи на языке программирования C++ с использованием свободной кроссплатформенной мультимедийной библиотеки SFML (англ. Simple and Fast Multimedia Library ) для создания оконного интерфейса.

# Основная часть

## Язык C++

### **Общая характеристика**

Объектно-ориентированный язык: C++ является мощным и эффективным объектно-ориентированным языком программирования, что позволяет структурировать код в виде классов и объектов, облегчая повторное использование кода и улучшая его структуру.

Низкоуровневые возможности: C++ предоставляет возможности низкоуровневого программирования, что делает его подходящим для реализации эффективных алгоритмов и структур данных.

### **Управление памятью**

Ручное управление памятью: Одной из отличительных особенностей C++ является возможность ручного управления памятью. Это может быть полезно при оптимизации производительности, но требует осторожности в избежании утечек памяти.

### **Перфоманс**

Высокая производительность: C++ известен своей высокой производительностью благодаря низкоуровневым возможностям и возможности оптимизации кода. Это особенно важно при работе с большими объемами данных и реализации алгоритмов, таких как сортировка.

**Применение в курсовой работе**

Эффективная реализация алгоритмов: Использование C++ позволяет эффективно реализовывать и тестировать различные алгоритмы сортировки, а также проводить анализ и сравнение их производительности.

Интеграция с различными библиотеками: C++ предоставляет возможность интеграции с различными библиотеками для визуализации данных, измерения производительности и других задач, что может быть полезно при выполнении курсовой работы.

## Библиотека SFML

SFML (Simple and Fast Multimedia Library) - это библиотека для разработки мультимедийных приложений, написанная на C++. Она предоставляет инструменты для работы с графикой, звуком, сетью и другими мультимедийными компонентами. Вот несколько ключевых аспектов SFML и способы его применения в контексте курсовой работы:

### Графическое программирование

Отрисовка графики: SFML обеспечивает простые и эффективные средства для отрисовки 2D-графики, что может быть полезным для визуализации процесса сортировки и результатов алгоритмов.

### Обработка ввода

Управление событиями: SFML предоставляет обработку событий, таких как нажатие клавиш и перемещение мыши. Это может быть использовано для интерактивного управления и тестирования алгоритмов.

### Звуковые эффекты

Воспроизведение звука: Если вам интересно добавить звуковые эффекты к вашей курсовой работе, SFML предоставляет средства для воспроизведения аудиофайлов.

### Многоплатформенность

Поддержка различных платформ: SFML предоставляет абстракции для работы с графикой и вводом, что делает приложения, написанные с использованием этой библиотеки, переносимыми между различными операционными системами.

Интуитивный интерфейс: SFML имеет простой и понятный интерфейс, что делает его подходящим для студентов и начинающих программистов. Он может быть использован для быстрого прототипирования и визуализации результатов вашего анализа алгоритмов.

### Применение в курсовой работе

Визуализация алгоритмов сортировки: SFML может быть использован для создания графического интерфейса, визуализирующего процесс сортировки и отображающего изменения в данных на каждом шаге.

## Краткое описание алгоритмов

**Сортировка вставками** – перебирая неотсортированный массив, программа начинает с 1 элемента. Проходя элементы слева на право, она, находясь на ***i*** элементе, постепенно сравнивает ***i+1*** элемент со всеми предыдущими и помещает его между элементами, таким образом, что левый элемент меньше или равен, а правый больше или равен.

В лучшем случае массив уже отсортирован и алгоритм просто сравнивает ***n*** раз, поэтому сложность — ***O(n).***

В худшем случае массив отсортирован в обратном порядке и производится ***n***\*(***n***-1)/2 сравнений или присваиваний, следовательно сложность равна ***2).***

**Пузырьковая сортировка** – перебирая неотсортированный массив, программа начинает с 1 элемента. Проходя элементы слева на право, она, находясь на ***i*** элементе, сравнивает его с ***i+1*** и, если ***i+1*** элемент меньше, то программа меняет их местами.

В лучшем случае делает ***n*** сравнений, тогда ***O(n).***

В худшем случае делает ***n***\*(***n***-1) сравнений или перестановок, тогда оценка сложности ***2).***

**Быстрая сортировка** – программа работает рекурсивно. На каждом шаге рекурсии программа:

1. Выбирает опорный элемент с ***i*** индексом. Правила выбора опорного элемента определяются заранее.
2. Сравнивает все остальные элементы с опорным и переставить их в массиве или подмассиве так, чтобы разбить массив на две части: «элементы меньшие опорного» и «элементы большие опорного».
3. Далее выполняет первые два шага в подмассивах, пока их длина больше единицы.

В лучшем случае при каждой операции разделения массив делится на две одинаковые части, следовательно, максимальная глубина рекурсии, при которой размеры обрабатываемых подмассивов достигнут 1, составит **.** В результате сложность алгоритма составляет ***.***

В **среднем** случае сложность можно определить лишь вероятностно. Заметим, что в действительности необязательно, чтобы опорный элемент делил массив на две одинаковых части. Например, если на каждом этапе будет происходить разделение на массивы длиной 75% и 25% от исходного, то сложность по прежнему будет , поскольку глубина рекурсии составит . Будем считать «удачным» разделением такое, при котором опорный эоемент окажется среди центральных 50% элементов разделяемой части массива. Поскольку вероятность удачи равна 0,5, для получения ***k***удачных разделений в среднем потребуется *2\*****k***рекурсивных вызовов, чтобы опорный элемент ***k***раз оказался среди центральных 50% массива. В среднем глубина рекурсии составляет . Поэтому сложность составит также .

В **худшем** случае каждое разбиение дает два подмассива размерами 1 и , то есть при каждом рекурсивном вызове больший массив будет на 1 короче, чем в предыдущий раз. Потребуется операций разделения и сложность будет равна .

**Сортировка выбором** – перебирая неотсортированный массив, программа начинает с 1 элемента. Идея состоит в поиске наименьшего элемента в списке, который затем меняется местами с элементом на вершине списка. Затем находится наименьший элемент из оставшихся, и меняется местами со вторым элементом. Процесс продолжается до тех пор, пока все элементы не займут свое конечное положение.

В лучшем случае - **.**

В худшем -

**Сортировка** **кучей** – программа работает рекурсивно. На каждом шаге рекурсии программа:

1. Поскольку дерево удовлетворяет свойству убывающей, самый большой элемент сохраняется в корневом узле. Программа удаляет корневой элемент и поместит в конец массива (n-я позиция). Далее помещает последний элемент дерева (кучу) в свободное место.
2. Уменьшает размер кучи на 1 и снова укрупняет корневой элемент, чтобы у вас был самый большой элемент в корне.
3. Процесс повторяется до тех пор, пока все элементы списка не будут отсортированы

Сортировка использует бинарное дерево, поэтому в любом случае сложность -

# Приложение

## Описание

Приложение представляет собой многофайловый проект на основе объекто-ориентированного программирования, включающий в себя более 1000 строк кода, десятки определенных функций, классы и перечисления (enum). При запуске приложения, пользователю предлагается выбор – запуск консольного или оконного приложения. Я буду писать про оконное приложение, потому как именно оно притягивает внимание и возбуждает интерес. Затем открывается меню, имеющее 8 кнопок:

* Generate (Генерировать)
* Calculate (Рассчитать)
* Save (Сохранить)
* Download (Загрузить данные из файла)
* Display the table (Отобразить данные)
* Display a plot (Отобразить график)
* Restart (Перезапуск)
* Exit (Выйти)

Поговорим про каждое отдельно. Нажимая на кнопку генерации данных, в проекте запускается функция заполнения массива данных рандомными числами до 5000 включительно. Функционал второй кнопки уже поинтереснее, пользователь нажимающий на эту кнопку одновременно получает: отработку всех 5-ти прописанных в программе сортировок, сохранение времени их выполнения в двумерных массив, первая строка которого представляет собой список параметров сложности от 50 до 1000, вторая строка которого заполняется временем – t в секундах. При нажатии на кнопку Save или Download происходит открытие файла в директории проекта, куда вводятся данные либо откуда выводятся. Кнопка Display the table – отображает график. Она сработает только при условии, что данные были либо сгенерированы и отсортированы, либо загружены из файла. То же можно сказать и про кнопку Display a plot. Кнопка Restart (перезапуск) обнуляет данные массивов и перезапускает окно. Exit – полностью прекращает работу оконного приложения, после чего мы возвращаемся к самому первому пункту.

В работе были реализованы кнопки для перехода обратно в меню, обработаны события по изменению положения окна, его закрытию, изменению анимации при наведении на кнопки, а так же описаны функции, шаблоны для изменения позиции фигур, их цвета, размера и многое и другое.

## Содержание

Для обеспечения такого функционала приложения было описано следующее:

### Классы (Class)

* **class menu** (класс меню) - самый первый класс, описывающий менюшку в консоли, содержащий всего 3 функции: запуск проекта, завершение работы проекта, выбор команды.
* **class functions** (класс функций) – класс содержащий функции по осуществлению работы проекта «под капотом», в том числе выполнение сортировок, запись и загрузка данных в файл или из файла, очистка данных в массиве. Именно в этом классе хранятся данные времени исполнения алгоритмов сортировок. Передается по ссылке другим классам.
* **class graph\_menu** (класс графического меню) – класс основного меню, к которому подключаются остальные классы, представляющие графическое оформление проекта. Содержит в себе помимо описания меню – описание диалогового окна, всплывающего при нажатии кнопок или ошибок.
* **class graph\_table** (класс графического представления таблицы) – класс, описывающий таблицу с данными параметров сложности и сортировок
* **class plot** (класс графического представления графика) – класс, описывающий график, выводящий результаты по данным из таблицы

### Структура (Struct)

* **struct global** (глобальная структура с общими данными членами и функциями, а так же шаблонами) – в проекте передавалась в классы поддерживаемых окон, о которых будет сказано ниже

### Перечисления (Enum)

* **enum button** (кнопки) – используется для определения, по какой кнопке в меню пользователь «попал»

## Код header файлов

**menu.h**

#pragma once

#include "functions.h"

class menu {

private:

functions\* Func;

bool State = false;

public:

menu(functions\* f) : Func(f) {}

void start();

void end();

void choice();

};

**graph\_menu.h**

#include <SFML/Graphics.hpp>

#include <iostream>

#include <string>

#include <type\_traits>

#include "functions.h"

#include "graph\_table.h"

#include "graph\_menu\_global.h"

#include "plot.h"

#pragma once

enum buttons {

generate,

calculate,

save,

download,

table,

plot,

restart,

quit,

none

};

class graph\_menu {

private:

functions\* Func;

global g;

table\_window t;

plot\_window p;

buttons active\_button;

buttons get\_button() { return active\_button; }

void set\_button(buttons button) { this->active\_button = button; }

void set\_button(std::vector <sf::Text>& texts, sf::Vector2i& mouse\_pos);

protected:

std::string background\_img = "background\_4.jpg"; // Картинка фона

unsigned int windowWidth = static\_cast<unsigned int>(g.screenWidth \* 0.3); // Ширина окна

unsigned int windowHeight = static\_cast<unsigned int>(g.screenHeight \* 0.7); // Высота окна

unsigned int windowMessageWidth = static\_cast<unsigned int>(g.screenWidth \* 0.15); //Ширина окна с сообщением

unsigned int windowMessageHeight = static\_cast<unsigned int>(g.screenHeight \* 0.10); // Высота окна с сообщением

std::vector <std::string> text\_operation = { "Generate", "Calculate", "Save", "Download",

"Display the table", "Display a plot", "Restart", "Quit" }; // Наименование кнопок в графическом окне

std::string str\_font = "JosefinSans-Bold.ttf"; // Используемый шрифт в меню

std::string str\_font\_window = "Kanit-Thin.ttf"; // Используемый шрифт в окне

float position\_operation\_y = windowHeight \* 0.11f; // Отступ кнопок сверху вниз

float padding\_bottom\_operation = windowHeight \* 0.1f; // Отступы снизу

float position\_message\_y = windowMessageHeight \* 0.25f; // Позиция сообщения по оси OY

float position\_agreement\_y = windowMessageHeight \* 0.7f; // Позиции ok по оси OY

float position\_rectangle\_y = windowMessageHeight \* 0.65f; // Отступ прямоугольника по OY

unsigned int size\_font = static\_cast<unsigned int>(windowWidth / 15); // Размер шрифта в меню

unsigned int size\_font\_message = static\_cast<unsigned int>(windowMessageWidth / 15); // Размер шрифта в диалоговом окне

float rectangle\_outline\_thickness = 2;

float border\_size\_button = 3; // Размер границ у кнопок в главном меню

float border\_size\_button\_window = 0.1; // Размер границ у кнопок в диалоговом окне

std::vector <float> rectangle\_window\_size = {

static\_cast<float>(size\_font\_message\*4),

static\_cast<float>(size\_font\_message\*1.2)

};

sf::Color rectangle\_theFill\_color = sf::Color::White;

std::vector <int> custom\_color\_button = { 102, 255, 255, 190 }; // Цвет первых четырех кнопок по умолчанию

std::vector <int> second\_custom\_color\_button = { 255, 102, 255, 190 }; // Цвет следующий четырех кнопок

std::vector <int> event\_color\_button = { 102, 255, 255, 255 }; // Цвет кнопок при наведении первых четырех кнопок

std::vector <int> second\_event\_color\_button = { 255, 102, 255, 255 }; // Цвет кнопок при наведении вторых четырех кнопок

std::vector <int> border\_color\_button = { 0, 0, 0 }; // Цвет границы у кнопок

std::vector <int> rectangle\_theFill\_color\_animation = { 160,160,160,20 };

std::vector <int> rectangle\_outline\_color = { 160,160,160,255 };

std::vector <sf::Color> definition\_color(std::vector <int>& first, std::vector <int>& second,

std::vector <int>& third, std::vector <int>& fourth); // Функция определения вектора с rgb цветами

void definition\_button\_texts(std::vector <sf::Color>& rgb\_default); // Описание операций в меню

void mouse\_response\_button(sf::Vector2i& mouse\_pos, std::vector <sf::Color>& rgb,

int& iter, int& count); // Функция описывающая событие, возникающее при наведении на кнопку курсора

public:

graph\_menu(functions\* f) : Func(f), g(f), t(f), p(f) {}

bool window(); // Функция для создания окна, отрисовки содержимого и обработки событий

void window\_message(std::string message, std::string banner, sf::Vector2i window\_pos);

};

**graph\_menu\_global.h**

#pragma once

#include <SFML/Graphics.hpp>

#include "functions.h"

struct global

{

functions\* Func;

global(functions\* f) : Func(f) {}

// Объявление констант и объектов

sf::Sprite arrow\_sprite;

sf::Texture arrow\_texture;

sf::Texture arrow\_texture\_after;

std::string arrow\_img = "arrow\_before.png";

std::string arrow\_img\_after = "arrow\_after.png";

sf::Font font; // Объект шрифта

sf::Font font\_window; // Объект шрифта в диалоговом окне

sf::Sprite background\_sprite; // Спрайт фона

sf::Texture background\_texture; // Текстура фона

std::vector<sf::Text> texts; // Вектор объектов текст

std::vector<sf::RectangleShape> rectangles; // Вектор объектов прямоугольников

std::vector<sf::CircleShape> circles;

std::vector <sf::Color> rgb; // Вектор цветов

sf::VideoMode desktop = sf::VideoMode::getDesktopMode(); // Получение разрешения экрана

unsigned int screenWidth = desktop.width; // Ширина экрана

unsigned int screenHeight = desktop.height; // Высота экрана

int count = 0; //Счётчик

bool isDragging; // Булевая переменная, показывающая, нажимается ли левая кнопка мыши

sf::Vector2i start\_position; sf::Vector2i current\_position; sf::Vector2i delta; // Векторные объекты для изменения позиции окна

int define\_index(sf::Text& elem, std::vector <sf::Text>& vector, int count); /\*Определение индекса элемента типа Text в векторе\*/

void define\_center\_position\_window(sf::RenderWindow& window, sf::Vector2i& external\_window\_position, sf::Vector2i& external\_window\_size);

void download\_font(sf::Font& font, std::string& str); // Подгрузка шрифта из файла

void download\_image(sf::Texture& texture, const std::string& str); // Загрузка картинки в текстуру

void set\_decoration\_text(sf::Text& text, sf::Font& font, unsigned int& size, const sf::Color& color); // Декор текстового объекта

void set\_decoration\_rectangle(sf::RectangleShape& rectangle, sf::Vector2f& size,

float& outline\_thickness, sf::Color& outline\_color, sf::Color& theFill\_color);

void download\_backround(std::string background\_img, unsigned int windowWidth, unsigned int windowHeight);

unsigned int fps = 144; };

**template<typename T>**

**void define\_center\_position\_x(T& object, unsigned int& window\_width) {**

**float Center\_X = window\_width / 2.0f;**

**sf::FloatRect textBounds = object.getLocalBounds();**

**object.setPosition(Center\_X - textBounds.width / 2.0f, object.getPosition().y);**

**}**

**template<typename T>**

**void define\_center\_position\_x(T& object, sf::RectangleShape& rectangle) {**

**int left\_border = rectangle.getPosition().x; // Позиция Левой границы**

**int right\_border = left\_border + rectangle.getSize().x; // Правая граница**

**float Center\_X = left\_border + (right\_border - left\_border) / 2.0f;**

**sf::FloatRect objectBounds = object.getGlobalBounds();**

**object.setPosition(Center\_X - objectBounds.width / 2.0f, object.getPosition().y);**

**}**

**template<typename T>**

**void define\_center\_position\_area(T& object, sf::RectangleShape rectangle) // Не использовать для текстового объекта!!!**

**{**

**float left\_border = rectangle.getPosition().x; // Позиция Левой границы**

**float right\_border = left\_border + rectangle.getSize().x; // Правая граница**

**float top\_border = rectangle.getPosition().y; // Верхняя граница**

**float buttom\_border = top\_border + rectangle.getSize().y; // Нижняя граница**

**float Center\_X = left\_border + (right\_border - left\_border) / 2.0f;**

**float Center\_Y = top\_border + (buttom\_border - top\_border) / 2.0f;**

**sf::FloatRect objectBounds = object.getGlobalBounds();**

**object.setPosition(Center\_X - objectBounds.width / 2.0f, Center\_Y - objectBounds.height / 2.0f);**

**}**

**template<typename T>**

**void define\_center\_position\_area(T& object, unsigned int window\_width, unsigned int window\_height)**

**{**

**float Center\_X = window\_width / 2.0f;**

**float Center\_Y = window\_height / 2.0f;**

**sf::FloatRect objectBounds = object.getLocalBounds();**

**object.setPosition(Center\_X - objectBounds.width / 2.0f, Center\_Y - objectBounds.height / 2.0f);**

**}**

**template<typename T>**

**void define\_center\_position\_y(T& object, unsigned int& window\_height)**

**{**

**float Center\_Y = window\_height / 2.0f;**

**sf::FloatRect objectBounds = object.getLocalBounds();**

**object.setPosition(object.getPosition().x, Center\_Y - objectBounds.height / 2.0f);**

**}**

**template<typename T>**

**void define\_center\_position\_y(T& object, sf::RectangleShape& rectangle) {**

**int top\_border = rectangle.getPosition().y; // Позиция верхней границы внешнего объекта относительно окна**

**int buttom\_border = top\_border + rectangle.getSize().y; // нижняя граница**

**float Center\_Y = top\_border + (buttom\_border - top\_border) / 2.0f;**

**sf::FloatRect objectBounds = object.getGlobalBounds();**

**object.setPosition(object.getPosition().x, Center\_Y - objectBounds.height / 2.0f);**

**}**

**template<typename T>**

void define\_center\_position\_area\_text(sf::Text& text, sf::Font& font, unsigned int& size\_font\_message, T& rectangle) {

sf::String str\_text = text.getString();

sf::FloatRect textBounds = text.getGlobalBounds();

float left\_border = rectangle.getPosition().x; // Позиция Левой границы

float right\_border = left\_border + rectangle.getSize().x; // Правая граница

float top\_border = rectangle.getPosition().y; // Верхняя граница

float buttom\_border = top\_border + rectangle.getSize().y; // Нижняя граница

float Center\_X = left\_border + (right\_border - left\_border) / 2.0f;

float Center\_Y = top\_border + (buttom\_border - top\_border) / 2.0f;

text.setPosition(Center\_X - textBounds.width / 2.0f, Center\_Y - (textBounds.height));

}

template<typename T>

bool is\_there\_cursor(T& object, sf::Vector2i& mouse\_pos)

{

sf::FloatRect objectBounds = object.getGlobalBounds();

return objectBounds.contains(static\_cast<float>(mouse\_pos.x), static\_cast<float>(mouse\_pos.y));

}

**graph\_table.h**

#pragma once

#include "functions.h"

#include "graph\_menu\_global.h"

#include <string>

#include <vector>

class table\_window {

private:

global g;

functions\* Func;

protected:

unsigned int windowTableWidth = static\_cast<unsigned int>(g.screenWidth \* 0.6);

unsigned int windowTableHeight = static\_cast<unsigned int>(g.screenHeight \* 0.5);

unsigned int num\_columns = 12;

unsigned int num\_rows = 6;

unsigned int size\_title = static\_cast<int>(windowTableWidth / 60);

float size\_cell\_outline = 1;

sf::Color color\_cell\_outline = sf::Color::Black;

sf::Color color\_cell\_theFill;

sf::Color color\_title;

float size\_cell\_height = windowTableHeight / static\_cast<float>(num\_rows);

float size\_cell\_width = windowTableWidth / static\_cast<float>(num\_columns + 1);

std::vector <int> names\_column = { 50, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000 }; // Массив с набором параметров сложности

std::vector <std::string> names\_sort = {"Bubble sort", "Heap sort",

"Quick sort", "Selection sort", "Insertion sort"};

std::string font = "arial.ttf";

std::string background\_img = "background\_table.jpg"; // Картинка фона

void arrow();

void desctiption\_title(sf::RectangleShape rectangle, std::string text\_in);

void cells(std::vector<std::vector<double>> sort\_time, int number\_row);

void header();

void body();

void animation\_arrow(float width, float width\_after, float height, float height\_after, sf::Vector2i mouse\_pos);

public:

table\_window(functions\* f) : Func(f), g(f) {}

void table();

};

**plot.h**

#pragma once

#include "functions.h"

#include "graph\_menu\_global.h"

#include <string>

#include <SFML/Graphics.hpp>

#include <vector>

class plot\_window

{

private:

functions\* Func;

global g;

protected:

sf::Color color\_cells;

sf::Color color\_axes;

unsigned int plotWidth = static\_cast<unsigned int>(g.screenWidth \* 0.7);

unsigned int plotHeight = static\_cast<unsigned int>(g.screenHeight \* 0.7);

std::string str\_font = "JosefinSans-Bold.ttf";

std::string background\_img = "backgroundPlot\_6.jpg";

float size\_delta\_OX = plotWidth / 100.0f;

float size\_delta\_OY = size\_delta\_OX;

float size\_delta\_X\_number = plotWidth/ 200.0f;

float unit;

float outline\_rectangle = 2;

int num\_paramentres = 11;

float count\_cells = (num\_paramentres + 1) \* (4 / 3.0f);

float size\_cells\_height = plotHeight / count\_cells; // Единичный отрезок по времени

float size\_cells\_width = plotWidth / count\_cells; // Единичный отрезок по параметру сложности

std::vector <std::string> names\_param = { "50", "100", "200", "300", "400", "500", "600", "700", "800", "900", "1000"};

std::vector <std::string> information = { "- bubble sorting", "- insertion sorting", "- pyramid sorting", "- quick sorting", "- selection sorting" };

std::vector <sf::Color> information\_circles\_colors;

float position\_OX = plotWidth \* 0.25f;

float position\_OY = plotHeight \* 0.75f;

float radius = plotWidth / 250.0f;

sf::VertexArray axes; sf::VertexArray cells;

sf::VertexArray straight; sf::VertexArray unit\_segments;

sf::Text unit\_segment; sf::CircleShape circle\_shape;

sf::Color color\_segments; sf::Color color\_bubble;

sf::Color color\_insertion; sf::Color color\_heap;

sf::Color color\_quick; sf::Color color\_selection;

sf::Color color\_text\_in; sf::Color color\_arr\_rec\_outline = sf::Color::Transparent;

sf::Color color\_arr\_rec\_theFill;

sf::Vector2f size\_information;

unsigned int size\_segments = static\_cast<unsigned int>(plotWidth / 100.0f);

void graph(sf::Color& color, std::vector<std::vector<double>>& sort\_time);

void define\_designination();

void define\_button\_back();

void arrow();

void define\_unit\_segments\_line();

void define\_axes();

void define\_rectangle\_inf();

void define\_cell\_background();

void circle(float pos\_x, float before,

float radius, float trickness,

sf::Color color\_outline, sf::Color theFill);

void information\_in(sf::RectangleShape rectangle);

public:

plot\_window(functions\* f) : Func(f), g(f), axes(sf::Lines, 4),

cells(sf::Lines, count\_cells \* 4),

unit\_segments(sf::Lines, count\_cells \* 4),

size\_information(10 \* size\_cells\_width, size\_cells\_height),

straight(sf::Lines, 150),

color\_cells(120, 120, 120),

color\_axes(255, 255, 255),

color\_segments(255, 255, 255),

color\_bubble(255, 255, 204),

color\_heap(204, 255, 204),

color\_quick(204, 255, 255),

color\_selection(204, 204, 255),

color\_insertion(255, 204, 255),

color\_text\_in(0, 0, 0),

color\_arr\_rec\_theFill(255,255,255,80)

{

g.download\_font(g.font, str\_font); // Подгрузка шрифта

g.download\_backround(background\_img, plotWidth, plotHeight); // Подгрузка фона

};

void plot();

};

#pragma once

#include <vector>

#include <iostream>

#include <string>

#include <vector>

#include <iomanip>

#include <chrono>

#include <fstream>

#include <functional>

void bubble\_sorting(std::vector <double> vector); /\* Пузырьковая сортировка \*/

void pyramid\_sorting(std::vector <double> vector); /\* Пирамидальная сортировка\*/

void insertion\_sorting(std::vector<double> vector); // Сортировка вставками

void selection\_sorting(std::vector<double> vector); // Сортировка вставками

void quick\_sorting(std::vector<double> vector); // Быстрая сортировка

class functions { /\* Класс для работы с функциями \*/

protected:

std::fstream file; // Объект класса для чтения и записи данных из файла и в файл

std::ofstream outfile;

std::ifstream inputfile;

int C = 1000;//Константа (размер массива вектора)

std::vector <double> vect;/\* Динамический массив с числами double

std::vector <double> trial\_vect; /\* Срезанный массив \*/

std::string path\_to\_file = "Results\_of\_sortings.txt"; m// Путь к файлу

std::vector <int> diff\_par = { 50, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000 }; // Массив с набором параметров сложности

void print\_table(const std::vector<std::vector<double>>& data);

void save\_time(void view\_sorting(std::vector <double> vector), std::vector<std::vector<double>>& view\_time, int n);

void save\_data(std::vector<std::vector<double>>& sort\_time);

void download\_data(std::vector<std::vector<double>>& sort\_time);

public:

functions() : bubble\_time(2, std::vector<double>(11, 0.0)), pyramid\_time(2, std::vector<double>(11, 0.0)),

insertion\_time(2, std::vector<double>(11, 0.0)), quick\_time(2, std::vector<double>(11, 0.0)),

selection\_time(2, std::vector<double>(11, 0.0)) {}

std::vector<std::vector<double>> bubble\_time;

std::vector<std::vector<double>> pyramid\_time;

std::vector<std::vector<double>> insertion\_time;

std::vector<std::vector<double>> quick\_time;

std::vector<std::vector<double>> selection\_time;

void generate(); /\*Генерация массива\*/

void sorting();/\*Выполнить сортировку и сохранить время в массив\*/

void saving();/\* Сохранение данных в файл\*/

void downloading(); /\* Чтение данных из файла\*/

void show(); /\* Показать данные в таблице\*/

bool check\_content\_massive(std::vector<std::vector<double>> sorting\_time);

void reset\_data(); // Обновить данные

void display(); /\* Отобразить график в консоли \*/

bool empty\_data();

};

## Код алгоритмов сортировок

#pragma once

#include <iostream>

#include <string>

#include <vector>

#include <iomanip>

#include <chrono>

#include "functions.h"

// Пузырьковая сортировка

void bubble\_sorting(std::vector <double> vector)

{

int k = vector.size();

bool swapped;

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

swapped = false;

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

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

std::swap(vector[j], vector[j + 1]);

swapped = true;

}

}

if (!swapped) {

break;

}

}

}

// Сортировка кучами

void heapify(std::vector<double>& arr, int n, int i) {

int largest = i;

int l = 2 \* i + 1;

int r = 2 \* i + 2;

if (l < n && arr[l] > arr[largest]) {

largest = l;

}

if (r < n && arr[r] > arr[largest]) {

largest = r;

}

if (largest != i) {

std::swap(arr[i], arr[largest]);

heapify(arr, n, largest);

}

};

void pyramid\_sorting(std::vector<double> vector) {

int n = vector.size();

for (int i = n / 2 - 1; i >= 0; i--) {

heapify(vector, n, i);

}

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

std::swap(vector[0], vector[i]);

heapify(vector, i, 0);

}

}

// Сортировка вставками

void insertion\_sorting(std::vector<double> vector) {

int n = vector.size();

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

double key = vector[i];

int j = i - 1;

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

vector[j + 1] = vector[j];

j = j - 1;

}

vector[j + 1] = key;

}

}

// Быстрая сортировка

int partition(std::vector<double>& vector, int low, int high) {

double pivot = vector[high];

int i = (low - 1);

for (int j = low; j <= high - 1; j++) {

if (vector[j] < pivot) {

i++;

std::swap(vector[i], vector[j]);

}

}

std::swap(vector[i + 1], vector[high]);

return (i + 1);

}

void quick\_sort\_helper(std::vector<double> vector, int low, int high) {

if (low < high) {

int pi = partition(vector, low, high);

quick\_sort\_helper(vector, low, pi - 1);

quick\_sort\_helper(vector, pi + 1, high);

}

}

void quick\_sorting(std::vector<double> vector) {

int low = 0;

int high = vector.size() - 1;

quick\_sort\_helper(vector, low, high);

}

// Сортировка выбором

void selection\_sorting(std::vector<double> vector) {

int n = vector.size();

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

int min\_index = i;

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

if (vector[j] < vector[min\_index]) {

min\_index = j;

}

}

std::swap(vector[i], vector[min\_index]);

}

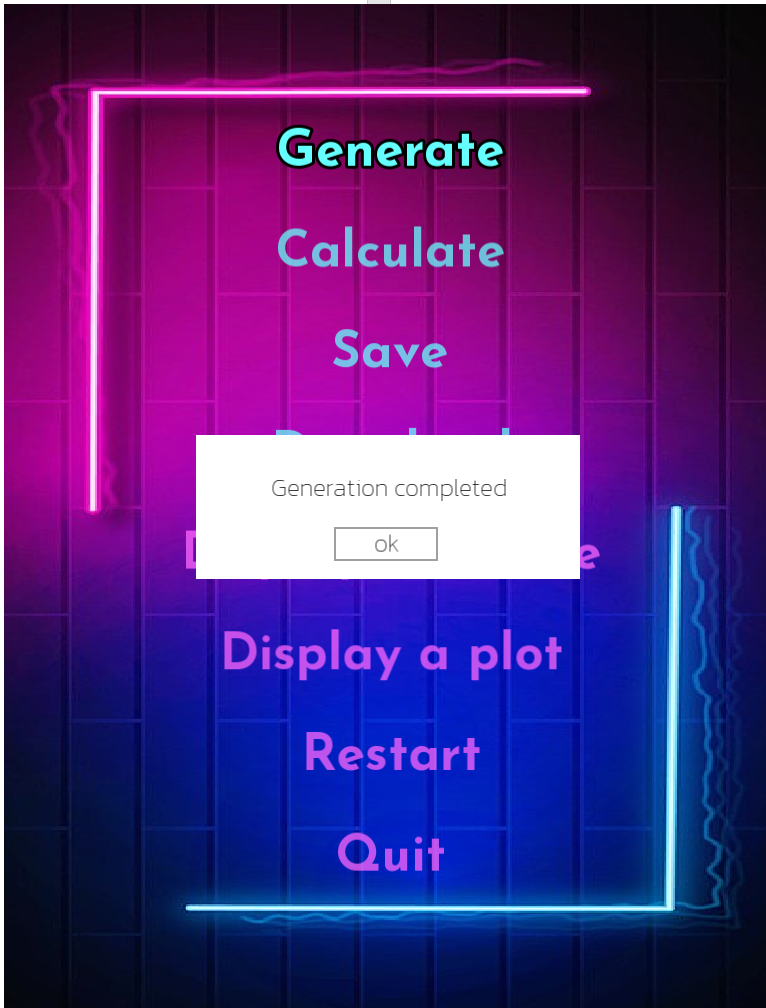
}

# Интерфейс и результаты

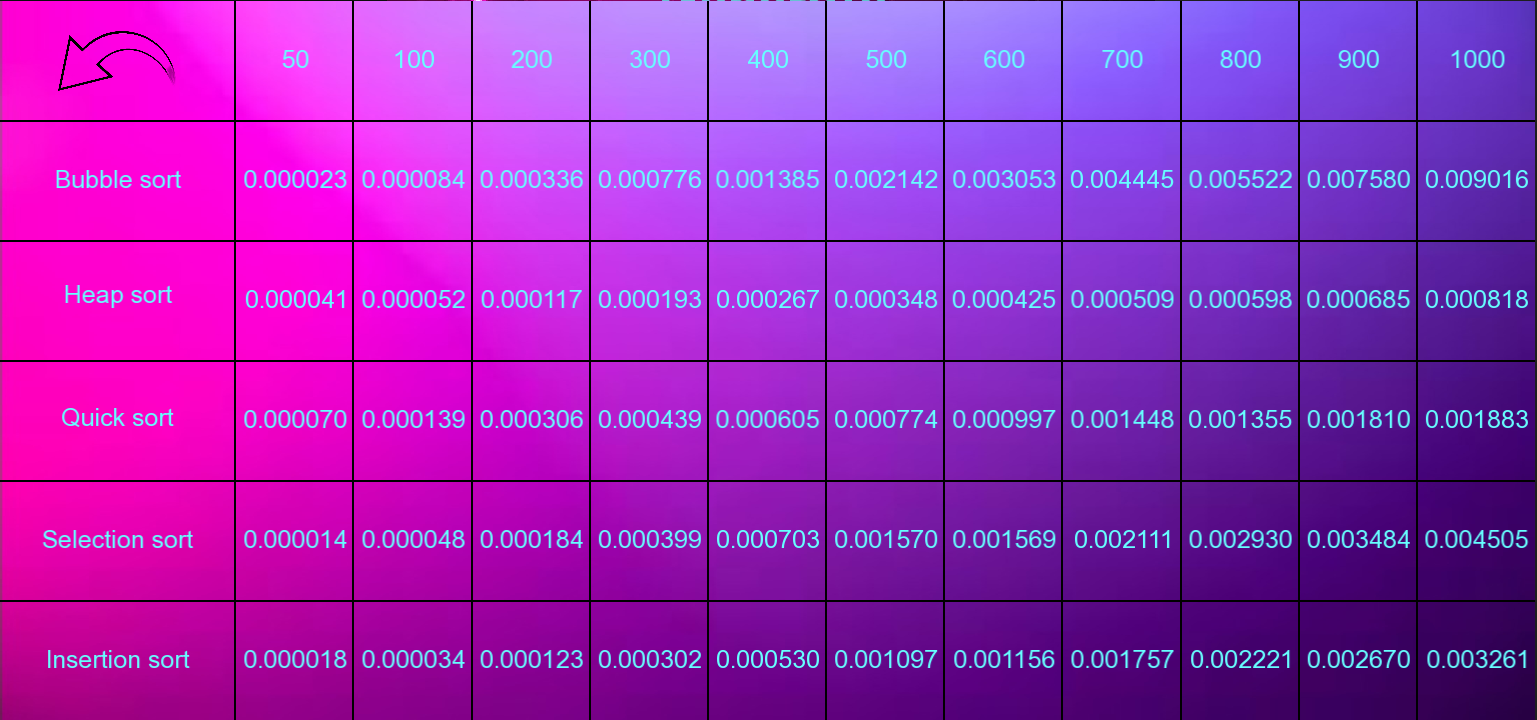
## Главное меню



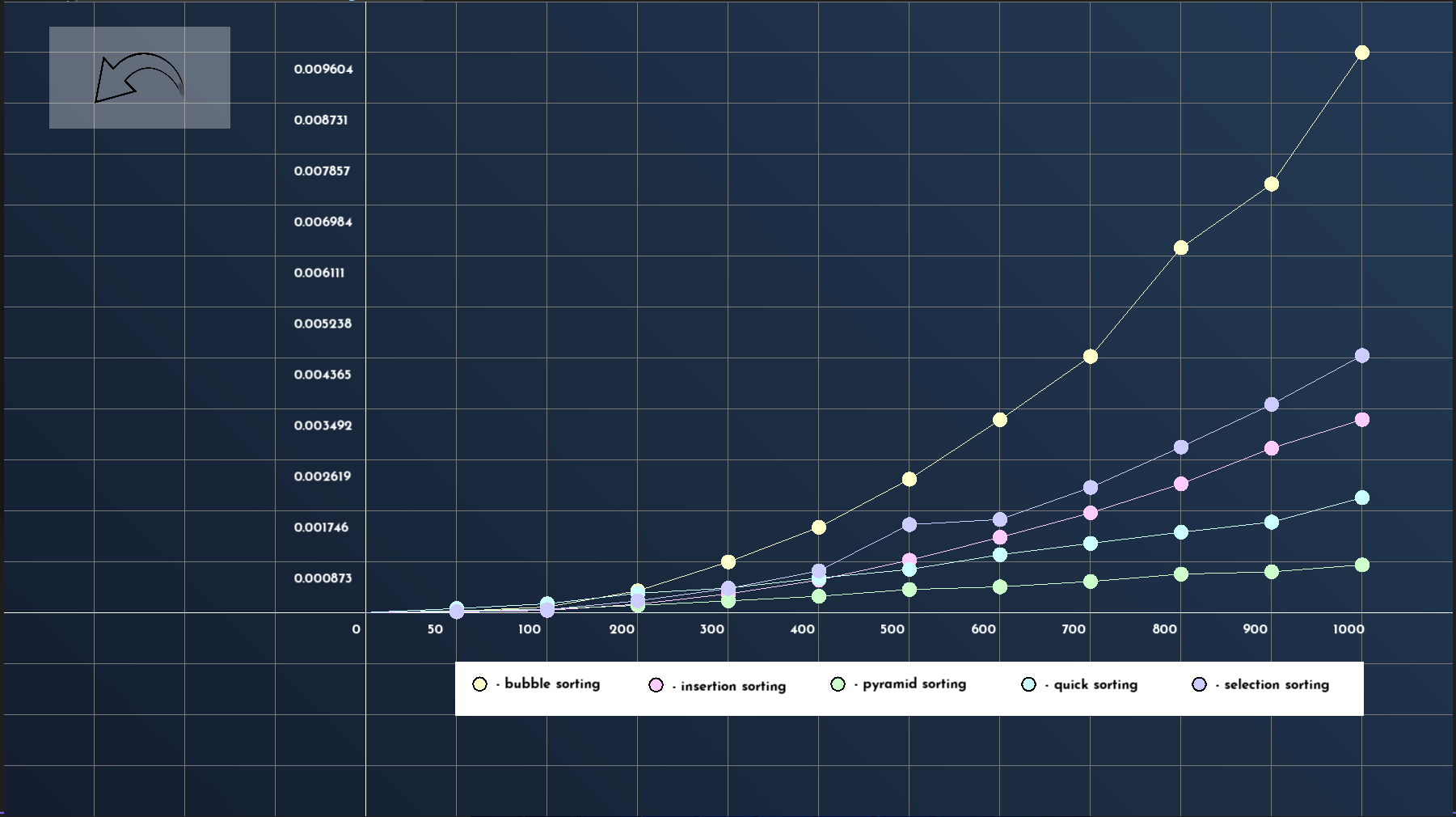
## Диалоговое окно



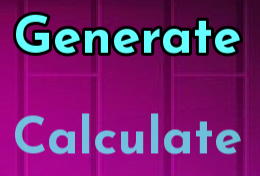
## Таблица

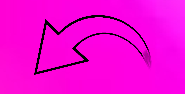


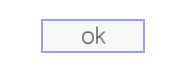
## График



## Анимация кнопок

# Заключение

В ходе данной курсовой работы был проведен обзор и анализ нескольких известных алгоритмов сортировки, включая сортировку вставками, пузырьковую сортировку, быструю сортировку, сортировку выбором и сортировку с использованием кучи. Каждый из этих алгоритмов обладает своими уникальными характеристиками, преимуществами и ограничениями.

Экспериментально были исследованы временные характеристики каждого алгоритма на различных входных данных. Некоторые алгоритмы, такие как быстрая сортировка, проявили высокую производительность на средних и больших объемах данных, в то время как другие, например, сортировка пузырьком, проявили себя менее эффективно на больших наборах данных из-за своей сложности.

При выполнении данной курсовой работы была использована библиотека SFML (Simple and Fast Multimedia Library) для создания графического интерфейса. С помощью SFML была реализована визуализация работы алгоритмов сортировки, что позволило наглядно продемонстрировать процесс сортировки и изменения в данных на каждом этапе. Использование SFML значительно облегчило создание интерактивной части работы и сделало её более доступной и понятной для аудитории.

В заключение, изучение и анализ алгоритмов сортировки играют важную роль в области программирования, позволяя разработчикам выбирать наиболее эффективные методы для обработки данных и оптимизации производительности программных приложений.

# Литература

1. Официальная документация по SFML [www.sfml-dev.org/](http://www.sfml-dev.org/)
2. официальная документация по C++ https://learn.microsoft.com/
3. Интернет-инциклопедия «Викиноспекты». Статья «Сортировки». neerc.ifmo.ru.
4. Видеохостинг. Блогер #SimpleCode