МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ   
РОССИЙСКОЙ ФЕДЕРАЦИИ

**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Российский государственный университет им. А.Н. Косыгина   
(Технологии. Дизайн. Искусство)»**

Институт ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ И ЦИФРОВЫЕ ТРАНСФОРМАЦИИ

**Отчет по лабораторной работе № 2**

**по дисциплине «Объектно-ориентированное программирование»**

**Тема: «Визуализация сортировки»**

Выполнил: Сидоров Д. С., группа ИТС-123

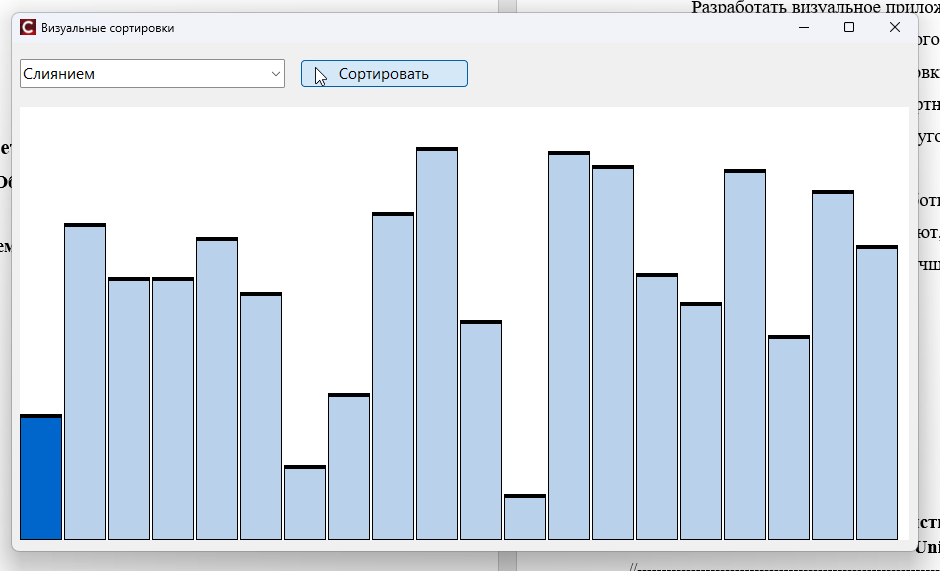
Проверил: к.т.н., доц. Семёнов А. А.

Москва, 2025г

**Визуализация сортировки**

Лабораторная работа посвящена визуализации одного из алгоритмов сортировки Яндекса (см. https://education.yandex.ru/journal/osnovnye-vidy-sortirovok-i-primery-ikh-realizatsii). Лабораторную работу необходимо выполнить в виде визуального приложения.

Разработать визуальное приложение, позволяющее визуализировать процесс сортировки целочисленного массива из 7 случайных элементов с анимацией (визуализация сортировки как на рисунке ниже). Допускается использовать подходящий стандартный компонент, либо конструкцию из прямоугольников.

  
Рисунок 1. Вид работы готовой программы.

**Листинг кода**

**Unit1.cpp**

///---------------------------------------------------------------------------

#include <vcl.h>

#include "Unit2.h"

#pragma hdrstop

#include "Unit1.h"

//---------------------------------------------------------------------------

#pragma package(smart\_init)

#pragma resource "\*.dfm"

TForm1 \*Form1;

//---------------------------------------------------------------------------

\_\_fastcall TForm1::TForm1(TComponent\* Owner) : TForm(Owner)

{

ShuffleData();

DrawData(PaintBox1);

}

//---------------------------------------------------------------------------

void \_\_fastcall TForm1::Button1Click(TObject \*Sender)

{

ShuffleData();

DrawData(PaintBox1);

int index = ComboBox1->ItemIndex;

SortType type = static\_cast<SortType>(index);

Sort(type, PaintBox1);

}

//---------------------------------------------------------------------------

**Unit1.h**

///---------------------------------------------------------------------------

#ifndef Unit1H

#define Unit1H

//---------------------------------------------------------------------------

#include <System.Classes.hpp>

#include <Vcl.Controls.hpp>

#include <Vcl.StdCtrls.hpp>

#include <Vcl.Forms.hpp>

#include <Vcl.ExtCtrls.hpp>

//---------------------------------------------------------------------------

class TForm1 : public TForm

{

\_\_published: // IDE-managed Components

TPaintBox \*PaintBox1;

TButton \*Button1;

TComboBox \*ComboBox1;

void \_\_fastcall Button1Click(TObject \*Sender);

private: // User declarations

public: // User declarations

\_\_fastcall TForm1(TComponent\* Owner);

};

//---------------------------------------------------------------------------

extern PACKAGE TForm1 \*Form1;

//---------------------------------------------------------------------------

#endif

**Unit2.cpp**

#include "Unit2.h"

#include <vcl.h>

#include <algorithm>

#include <thread>

#include <cmath>

std::vector<int> data;

int highlightedIndex = -1;

void BubbleSort(TPaintBox\* PaintBox) {

int n = data.size();

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

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

highlightedIndex = j;

DrawData(PaintBox);

Sleep(20);

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

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

}

}

void CocktailSort(TPaintBox\* PaintBox) {

bool swapped = true;

int start = 0, end = data.size() - 1;

while (swapped) {

swapped = false;

for (int i = start; i < end; ++i) {

highlightedIndex = i;

DrawData(PaintBox);

Sleep(20);

if (data[i] > data[i + 1]) {

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

swapped = true;

}

}

if (!swapped) break;

swapped = false;

--end;

for (int i = end - 1; i >= start; --i) {

highlightedIndex = i;

DrawData(PaintBox);

Sleep(20);

if (data[i] > data[i + 1]) {

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

swapped = true;

}

}

++start;

}

}

void CombSort(TPaintBox\* PaintBox) {

int gap = data.size();

bool swapped = true;

const float shrink = 1.3f;

while (gap > 1 || swapped) {

gap = std::max(1, int(gap / shrink));

swapped = false;

for (int i = 0; i + gap < data.size(); ++i) {

highlightedIndex = i;

DrawData(PaintBox);

Sleep(20);

if (data[i] > data[i + gap]) {

std::swap(data[i], data[i + gap]);

swapped = true;

}

}

}

}

void InsertionSort(TPaintBox\* PaintBox) {

for (size\_t i = 1; i < data.size(); ++i) {

int key = data[i];

int j = i - 1;

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

data[j + 1] = data[j];

highlightedIndex = j;

DrawData(PaintBox);

Sleep(20);

--j;

}

data[j + 1] = key;

}

}

void SelectionSort(TPaintBox\* PaintBox) {

for (size\_t i = 0; i < data.size(); ++i) {

size\_t minIdx = i;

for (size\_t j = i + 1; j < data.size(); ++j) {

highlightedIndex = j;

DrawData(PaintBox);

Sleep(20);

if (data[j] < data[minIdx])

minIdx = j;

}

std::swap(data[i], data[minIdx]);

}

}

void QuickSort(TPaintBox\* PaintBox, int low, int high) {

if (low >= high) return;

int pivot = data[high];

int i = low - 1;

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

highlightedIndex = j;

DrawData(PaintBox);

Sleep(20);

if (data[j] < pivot) {

++i;

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

}

}

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

QuickSort(PaintBox, low, i);

QuickSort(PaintBox, i + 2, high);

}

void Merge(TPaintBox\* PaintBox, int left, int mid, int right) {

std::vector<int> L(data.begin() + left, data.begin() + mid + 1);

std::vector<int> R(data.begin() + mid + 1, data.begin() + right + 1);

int i = 0, j = 0, k = left;

while (i < L.size() && j < R.size()) {

highlightedIndex = k;

DrawData(PaintBox);

Sleep(20);

data[k++] = (L[i] <= R[j]) ? L[i++] : R[j++];

}

while (i < L.size()) data[k++] = L[i++];

while (j < R.size()) data[k++] = R[j++];

}

void MergeSort(TPaintBox\* PaintBox, int left, int right) {

if (left < right) {

int mid = (left + right) / 2;

MergeSort(PaintBox, left, mid);

MergeSort(PaintBox, mid + 1, right);

Merge(PaintBox, left, mid, right);

}

}

void Heapify(TPaintBox\* PaintBox, int n, int i) {

int largest = i;

int l = 2 \* i + 1;

int r = 2 \* i + 2;

if (l < n && data[l] > data[largest]) largest = l;

if (r < n && data[r] > data[largest]) largest = r;

if (largest != i) {

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

highlightedIndex = i;

DrawData(PaintBox);

Sleep(20);

Heapify(PaintBox, n, largest);

}

}

void HeapSort(TPaintBox\* PaintBox) {

int n = data.size();

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

Heapify(PaintBox, n, i);

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

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

Heapify(PaintBox, i, 0);

}

}

void Sort(SortType type, TPaintBox\* PaintBox) {

switch (type) {

case SortType::Bubble: BubbleSort(PaintBox); break;

case SortType::Cocktail: CocktailSort(PaintBox); break;

case SortType::Comb: CombSort(PaintBox); break;

case SortType::Insertion: InsertionSort(PaintBox); break;

case SortType::Selection: SelectionSort(PaintBox); break;

case SortType::Quick: QuickSort(PaintBox, 0, data.size() - 1); break;

case SortType::Merge: MergeSort(PaintBox, 0, data.size() - 1); break;

case SortType::Heap: HeapSort(PaintBox); break;

}

highlightedIndex = -1;

DrawData(PaintBox);

}

void ShuffleData() {

int count = 20;

data.resize(count);

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

data[i] = rand() % 100 + 10;

}

void DrawData(TPaintBox\* PaintBox) {

int w = PaintBox->Width;

int h = PaintBox->Height;

int barWidth = w / data.size();

Graphics::TBitmap\* buffer = new Graphics::TBitmap();

buffer->Width = w;

buffer->Height = h;

TCanvas\* canvas = buffer->Canvas;

canvas->Brush->Color = clWhite;

canvas->FillRect(TRect(0, 0, w, h));

for (size\_t i = 0; i < data.size(); ++i) {

int barHeight = (data[i] \* h) / 120;

int x = i \* barWidth;

canvas->Brush->Color = (i == highlightedIndex) ? clHotLight : clGradientActiveCaption;

canvas->Rectangle(x, h - barHeight, x + barWidth - 2, h);

canvas->Brush->Color = clBlack;

canvas->Rectangle(x, h - barHeight, x + barWidth - 2, h - barHeight + 4);

}

PaintBox->Canvas->Draw(0, 0, buffer);

delete buffer;

Application->ProcessMessages();

}

**Unit2.h**

#ifndef Unit2H

#define Unit2H

#include <vector>

#include <Vcl.ExtCtrls.hpp>

extern std::vector<int> data;

extern int highlightedIndex;

enum class SortType {

Bubble,

Cocktail,

Comb,

Insertion,

Selection,

Quick,

Merge,

Heap

};

void ShuffleData();

void DrawData(TPaintBox\* PaintBox);

void Sort(SortType type, TPaintBox\* PaintBox);

void QuickSort(TPaintBox\* PaintBox, int low, int high);

void CocktailSort(TPaintBox\* PaintBox);

void BubbleSort(TPaintBox\* PaintBox);

void CombSort(TPaintBox\* PaintBox);

void InsertionSort(TPaintBox\* PaintBox);

void SelectionSort(TPaintBox\* PaintBox);

void MergeSort(TPaintBox\* PaintBox, int left, int right);

void HeapSort(TPaintBox\* PaintBox);

#endif

**Вывод:** В ходе выполнения лабораторной работы был разработан программный продукт, позволяющий визуализировать процесс сортировки целочисленного массива, что способствует лучшему пониманию принципов работы алгоритмов сортировки. Использование анимации при визуализации делает наглядным каждый шаг сортировки, демонстрируя перестановки элементов и сравнения между ними в режиме реального времени. Программа генерирует случайный массив из 7 элементов, что позволяет многократно наблюдать работу алгоритма на различных исходных данных и анализировать его эффективность. В процессе разработки были применены знания объектно-ориентированного программирования и работы с графическими элементами библиотеки VCL, что значительно расширило практические навыки программирования. Выбранный алгоритм сортировки Яндекса показал свою эффективность и наглядность в контексте учебной визуализации, позволяя отслеживать каждый этап работы. Разработанное приложение имеет понятный пользовательский интерфейс, что делает его доступным для использования в образовательных целях студентами и преподавателями. Данная работа является важным шагом в понимании основ алгоритмов сортировки и может быть использована как учебный материал при изучении темы "Алгоритмы и структуры данных".