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 № 1**

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

**Тема: «Сортировки Яндекса»**

Выполнил: Сидоров Д. С., группа ИТС-122

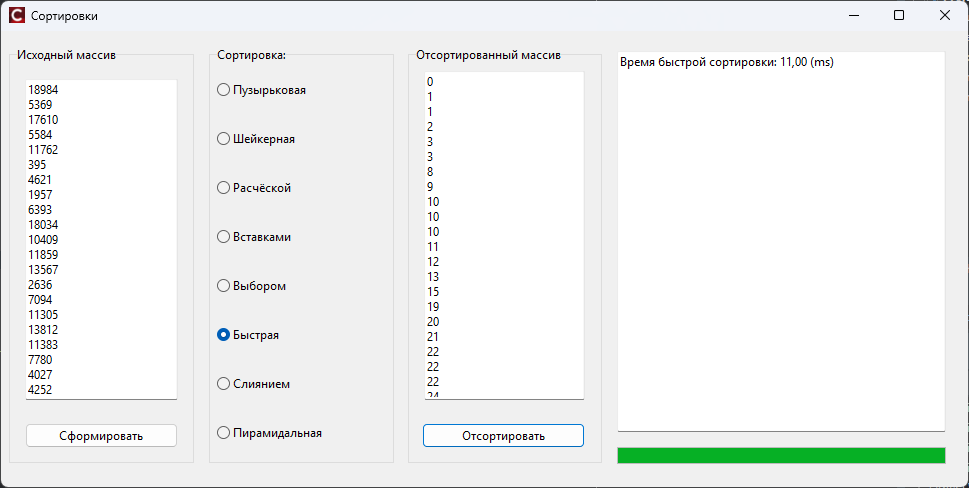
Проверил: к.т.н., доц. Семёнов А. А.

Москва, 2025г

**Сортировки Яндекса**

Лабораторная работа посвящена изучению основных алгоритмов сортировок и способов их реализации на примере Яндекса (см. https://education.yandex.ru/journal/osnovnye-vidy-sortirovok-i-primery-ikh-realizatsii). Лабораторную работу необходимо реализовать в виде визуального приложения.

Задание: изучить подготовленный мной иллюстрированный пример реализации на C++ основных видов сортировок (см. ниже). В примере я демонстрирую реализацию видов сортировок, но без "пирамидальной" и "слиянием". Необходимо самостоятельно воспроизвести реализацию этого проекта и по аналогии доделать пирамидальную сортировку и сортировку слиянием.

  
Рисунок 1. Вид работы готовой программы.

Теперь все сортировки работают, кроме того добавлен прогресс бар и улучшено время вывода в мемо, улучшена читаемость кода.

**Листинг кода**

**Unit1.cpp**

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

#include <vcl.h>

#include <vector>

#include <iterator>

#include <algorithm>

#include <cmath>

#pragma hdrstop

#include "Unit1.h"

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

#pragma package(smart\_init)

#pragma resource "\*.dfm"

using namespace std;

TForm1 \*Form1;

const int N = 20000;

int array1[N];

SYSTEMTIME st1, st2;

// Глобальные переменные для прогресс бара

TProgressBar\* GlobalProgressBar = nullptr;

\_\_int64 TotalOperations = 0;

\_\_int64 CurrentOperation = 0;

int progressThrottle = 1000000;

// Функция обновления прогресса

void UpdateProgress() {

if (GlobalProgressBar && TotalOperations > 0) {

GlobalProgressBar->Position = (CurrentOperation \* 100) / TotalOperations;

Application->ProcessMessages();

}

}

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

// Пузырьковая сортировка

void BubbleSort(vector<int>& values) {

TotalOperations = values.size() \* values.size();

CurrentOperation = 0;

for (size\_t i = 0; i + 1 < values.size(); ++i) {

for (size\_t j = 0; j + 1 < values.size() - i; ++j) {

if (values[j + 1] < values[j]) {

swap(values[j], values[j + 1]);

}

CurrentOperation++;

if (CurrentOperation % progressThrottle == 0) UpdateProgress();

}

}

}

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

// Шейкерная сортировка

void ShakerSort(vector<int>& values) {

if (values.empty()) return;

TotalOperations = values.size() \* values.size();

CurrentOperation = 0;

int left = 0, right = values.size() - 1;

while (left <= right) {

for (int i = right; i > left; --i) {

if (values[i - 1] > values[i]) {

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

}

CurrentOperation++;

if (CurrentOperation % progressThrottle == 0) UpdateProgress();

}

++left;

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

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

swap(values[i], values[i + 1]);

}

CurrentOperation++;

if (CurrentOperation % progressThrottle == 0) UpdateProgress();

}

--right;

}

}

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

// Сортировка расческой

void CombSort(vector<int>& values) {

TotalOperations = values.size() \* log2(values.size());

CurrentOperation = 0;

const double factor = 1.247;

double step = values.size() - 1;

while (step >= 1) {

for (int i = 0; i + step < values.size(); ++i) {

if (values[i] > values[i + step]) {

swap(values[i], values[i + step]);

}

CurrentOperation++;

if (CurrentOperation % progressThrottle == 0) UpdateProgress();

}

step /= factor;

}

// Финальная пузырьковая сортировка

for (size\_t i = 0; i + 1 < values.size(); ++i) {

for (size\_t j = 0; j + 1 < values.size() - i; ++j) {

if (values[j + 1] < values[j]) {

swap(values[j], values[j + 1]);

}

CurrentOperation++;

if (CurrentOperation % progressThrottle == 0) UpdateProgress();

}

}

}

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

// Сортировка вставками

void InsertionSort(vector<int>& values) {

TotalOperations = values.size() \* values.size();

CurrentOperation = 0;

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

int x = values[i];

size\_t j = i;

while (j > 0 && values[j - 1] > x) {

values[j] = values[j - 1];

--j;

CurrentOperation++;

if (CurrentOperation % progressThrottle == 0) UpdateProgress();

}

values[j] = x;

}

}

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

// Сортировка выбором

void SelectionSort(vector<int>& values) {

TotalOperations = values.size() \* values.size();

CurrentOperation = 0;

for (vector<int>::iterator i = values.begin(); i != values.end(); ++i) {

vector<int>::iterator j = min\_element(i, values.end());

swap(\*i, \*j);

CurrentOperation += values.end() - i;

if (CurrentOperation % progressThrottle == 0) UpdateProgress();

}

}

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

// Быстрая сортировка

int Partition(vector<int>& values, int l, int r) {

int x = values[r];

int less = l;

for (int i = l; i < r; ++i) {

if (values[i] <= x) {

swap(values[i], values[less]);

++less;

}

CurrentOperation++;

if (CurrentOperation % progressThrottle == 0) UpdateProgress();

}

swap(values[less], values[r]);

return less;

}

void QuickSortImpl(vector<int>& values, int l, int r) {

if (l < r) {

int q = Partition(values, l, r);

QuickSortImpl(values, l, q - 1);

QuickSortImpl(values, q + 1, r);

}

}

void QuickSort(vector<int>& values) {

TotalOperations = values.size() \* log2(values.size());

CurrentOperation = 0;

if (!values.empty()) {

QuickSortImpl(values, 0, values.size() - 1);

}

}

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

// Сортировка слиянием

void Merge(vector<int>& values, vector<int>& temp, int left, int mid, int right) {

int i = left;

int j = mid + 1;

int k = left;

while (i <= mid && j <= right) {

if (values[i] <= values[j])

temp[k++] = values[i++];

else

temp[k++] = values[j++];

CurrentOperation++;

if (CurrentOperation % progressThrottle == 0) UpdateProgress();

}

while (i <= mid) {

temp[k++] = values[i++];

CurrentOperation++;

}

for (i = left; i < k; i++) {

values[i] = temp[i];

}

}

void MergeSortImpl(vector<int>& values, vector<int>& temp, int left, int right) {

if (left < right) {

int mid = (left + right) / 2;

MergeSortImpl(values, temp, left, mid);

MergeSortImpl(values, temp, mid + 1, right);

Merge(values, temp, left, mid, right);

}

}

void MergeSort(vector<int>& values) {

TotalOperations = values.size() \* log2(values.size());

CurrentOperation = 0;

vector<int> temp(values.size());

MergeSortImpl(values, temp, 0, values.size() - 1);

}

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

// Пирамидальная сортировка

void Heapify(vector<int>& values, int n, int i) {

int largest = i;

int left = 2 \* i + 1;

int right = 2 \* i + 2;

if (left < n && values[left] > values[largest])

largest = left;

if (right < n && values[right] > values[largest])

largest = right;

if (largest != i) {

swap(values[i], values[largest]);

CurrentOperation++;

if (CurrentOperation % progressThrottle == 0) UpdateProgress();

Heapify(values, n, largest);

}

}

void HeapSort(vector<int>& values) {

TotalOperations = values.size() \* log2(values.size());

CurrentOperation = 0;

int n = values.size();

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

Heapify(values, n, i);

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

swap(values[0], values[i]);

Heapify(values, i, 0);

}

}

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

// Оптимизированный вывод в Memo

void DisplayArray(TMemo\* Memo, const vector<int>& arr) {

Memo->Clear();

Memo->Lines->BeginUpdate();

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

Memo->Lines->Add(IntToStr(arr[i]));

}

Memo->Lines->EndUpdate();

}

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

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

{

Memo1->Clear();

Memo2->Clear();

Memo3->Clear();

}

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

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

{

srand(time(NULL));

Memo1->Clear();

Memo1->Lines->BeginUpdate();

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

array1[i] = rand() % 20000;

Memo1->Lines->Add(IntToStr(array1[i]));

}

Memo1->Lines->EndUpdate();

Button2->Enabled = true;

}

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

void \_\_fastcall TForm1::Button2Click(TObject \*Sender)

{

typedef void (\*SortFunction)(vector<int>&);

struct SortInfo {

SortFunction func;

const char\* name;

};

const SortInfo sorts[] = {

{BubbleSort, "пузырьковой"},

{ShakerSort, "шейкерной"},

{CombSort, "расческой"},

{InsertionSort, "вставками"},

{SelectionSort, "выбором"},

{QuickSort, "быстрой"},

{MergeSort, "слиянием"},

{HeapSort, "пирамидальной"}

};

int sortIndex = RadioGroup1->ItemIndex;

if(sortIndex >= 0 && sortIndex < 8) {

vector<int> values(array1, array1 + N);

ProgressBar1->Position = 0;

GlobalProgressBar = ProgressBar1;

GetLocalTime(&st1);

sorts[sortIndex].func(values);

GetLocalTime(&st2);

GlobalProgressBar = nullptr;

ProgressBar1->Position = 100;

DisplayArray(Memo2, values);

double time = (st2.wMinute\*60\*1000 + st2.wSecond\*1000 + st2.wMilliseconds) -

(st1.wMinute\*60\*1000 + st1.wSecond\*1000 + st1.wMilliseconds);

AnsiString timeStr = FloatToStrF(time, ffFixed, 8, 2);

AnsiString message = "Время " + AnsiString(sorts[sortIndex].name) +

" сортировки: " + timeStr + " (ms)";

Memo3->Lines->Add(message);

}

}

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

**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>

#include <Vcl.ComCtrls.hpp>

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

class TForm1 : public TForm

{

\_\_published: // IDE-managed Components

TGroupBox \*GroupBox1;

TGroupBox \*GroupBox2;

TMemo \*Memo1;

TMemo \*Memo2;

TButton \*Button1;

TButton \*Button2;

TMemo \*Memo3;

TRadioGroup \*RadioGroup1;

TProgressBar \*ProgressBar1;

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

void \_\_fastcall Button2Click(TObject \*Sender);

private: // User declarations

public: // User declarations

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

};

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

extern PACKAGE TForm1 \*Form1;

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

#endif

**Вывод:** В ходе выполнения лабораторной работы №1 было разработано приложение с графическим интерфейсом для сравнения эффективности различных алгоритмов сортировки. Реализация приложения потребовала применения объектно-ориентированного подхода для организации логики сортировок и взаимодействия с пользовательским интерфейсом. Благодаря использованию принципов ООП удалось создать модульную структуру программы, где алгоритмы сортировки представлены как отдельные классы с унифицированным интерфейсом. Это позволило легко добавлять новые алгоритмы без изменения основной логики приложения, что демонстрирует принцип открытости/закрытости. Интересным аспектом работы стало измерение времени выполнения различных сортировок на массивах разной размерности и степени упорядоченности, что позволило наглядно сравнить их производительность. Пользовательский интерфейс, реализованный с помощью форм, обеспечил удобное взаимодействие с приложением, предоставляя возможности для выбора параметров тестирования и визуализации результатов. Практические результаты подтвердили теоретические оценки сложности алгоритмов и продемонстрировали преимущества объектно-ориентированного подхода при разработке программного обеспечения для анализа и сравнения алгоритмов.