Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение

высшего профессионального образования

«Южно-Уральский государственный университет»

ИНСТИТУТ ЕСТЕСТВЕННЫХ И ТОЧНЫХ НАУК

Факультет математики, механики и компьютерных технологий

Кафедра прикладной математики и программирования

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА  
К КУРСОВОЙ РАБОТЕ  
ПО ДИСЦИПЛИНЕ «АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ»

ТЕМА: «Сравнение быстродействия методов сортировки, быстрой сортировки Хоара и метода пузырька»

|  |
| --- |
| Руководитель работы,  \_\_\_\_\_\_\_\_\_\_\_\_\_ /Шелудько А.С.  «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_2020 г. |
| Автор работы  Студент группы ЕТ-2020  \_\_\_\_\_\_\_\_\_\_\_\_/Шерстобитов Т.С.  «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_2020 г. |

Челябинск 2020

# АННОТАЦИЯ

Шерстобитов Т.С. Сравнение быстродействия методов сортировки, быстрой сортировки Хоара и метода пузырька. – Челябинск: ЮУрГУ, ЕТ-212, 2020. – 27с, библиографический список – 1 наим., 1 прил.

В курсовой работе описывается разработка программы выполняющей сравнение быстродействия методов сортировки, сортировки Хоара и метода пузырька. Работа содержит результаты объектно-ориентированного анализа и проектирования, инструкции по использованию программы.

В результате работы была разработана программа выполняющая сравнение быстродействия методов сортировки, быстрой сортировки Хоара и метода пузырька, код которой приводится в приложении.

# ОГЛАВЛЕНИЕ

Оглавление

[АННОТАЦИЯ 3](#_Toc37887908)

[ОГЛАВЛЕНИЕ 4](#_Toc37887909)

[ВВЕДЕНИЕ 5](#_Toc37887910)

[1.ПОСТАНОВКА ЗАДАЧИ 6](#_Toc37887911)

[2.ОПИСАНИЕ АЛГОРИТМА 7](#_Toc37887912)

[3.ИНСТРУКЦИЯ ПО УСТАНОВКЕ И ТРЕБОВАНИЯ К СИСТЕМЕ 15](#_Toc37887913)

[4.РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 16](#_Toc37887914)

[БИБЛИОГРАФИЧЕСКИЙ СПИСОК 19](#_Toc37887915)

[ПРИЛОЖЕНИЕ А 20](#_Toc37887916)

[А.1 Sort.hpp 20](#_Toc37887917)

[А.2 Sort.cpp 20](#_Toc37887918)

[А.3 Measurement.hpp 21](#_Toc37887919)

[А.4 Measurement.cpp 22](#_Toc37887920)

[А.5 Analysis.hpp 23](#_Toc37887921)

[А.6 Analysis.cpp 23](#_Toc37887922)

[А.7 Writer.hpp 24](#_Toc37887923)

[А.8 Writer.cpp 25](#_Toc37887924)

[А.9 Main.cpp 27](#_Toc37887925)

# ВВЕДЕНИЕ

**Актуальность темы.** Правильный выбор алгоритма сортировки позволяет значительно оптимизировать время работы программ. Фактические данные и их визуализация позволяют начинающим программистам выбрать более подходящий алгоритм сортировки.

**Цель работы** – Разработать программу, выполняющую сравнение быстродействия двух методов сортировки, быстрой сортировки Хоара и метода пузырька

**Объект работы** – Программа выполняющая сравнение алгоритмов сортировки

**Результаты работы** можно использовать в процессе последующего обучения в соответствии с учебным планом подготовки бакалавров по направлению «Прикладная математика и информатика»

## 1.ПОСТАНОВКА ЗАДАЧИ

Необходимо разработать программу, выполняющую сравнение быстродействия двух методов сортировки:

* метод пузырька;
* быстрая сортировка Хоара.

Сравнение выполнить методом статистических испытаний. Приблизительная схема сравнения времени сортировки целого массива из N чисел:

1.Генерировать случайный целочисленный массив A (функции random и randomise). Генерируемые числа принадлежат диапазону от 0 до 109-1.

2.Копировать массив А в массив B.

3.Выполнить сортировку массива B методом пузырька и вычислить потраченное время (используйте функцию GetLocalTime или timeGetTime. Описание функций ищите в интернет)

4.Копировать массив А в массив B.

5.Выполнить сортировку массива B с помощью быстрой сортировки Хоара и вычислить потраченное время

Повторять пункты 1-5 L раз.

Найти среднее время сортировки для каждого из методов.

6.Перейти к следующему значению N.

Программа должна получить достаточное количество точек для построения графиков зависимостей времени сортировки от размера массива.

Значения L,N и другие исходные данные вводятся на старте программы из входного файла.

Напечатать таблицы зависимостей и построить совмещённые графики зависимостей времени сортировки от размера массива в координатах: по оси х – log10 (N), по оси у – log10(время(ms)).

Для построения графиков подберите в интернете подходящую программу, например, http://soft.mydiv.net/win/download-Graph.html.

## 2.ОПИСАНИЕ АЛГОРИТМА

Программа выполняет сравнение двух следующих алгоритмов сортировки:

Метод пузырька.



Рисунок 2. Алгоритм сортировки метода пузырька

Реализация алгоритма на языке программирования C++:

void bubbleSort(int\* array, int n){

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

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

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

int tmp = array[j];

array[j] = array[j + 1];

array[j + 1] = tmp;

}

}

}

}

Быстрая сортировка Хоара



Рисунок 2. Алгоритм быстрой сортировки Хоара

Реализация алгоритма на языке программирования C++:

void quickSort(int\* array,int first, int last){

if (first < last){

int left = first;

int right = last;

int middle = array[(left + right) / 2];

do{

while (array[left] < middle){

left++;

}

while (array[right] > middle){

right--;

}

if (left <= right){

int tmp = array[left];

array[left] = array[right];

array[right] = tmp;

left++;

right--;

}

} while (left <= right);

quickSort(array, first, right);

quickSort(array, left, last);

}

}

Для выполнения сравнения методов сортировки был разработан модуль Analysis (интерфейсная часть в файлах Analysis.hpp и Measurement.cpp, реализация в файлах Analysis.cpp и Measurement.cpp) выполняющий сравнение и содержащий данные о сравнении.

class Analysis{

public:

/\*\*

\* Конструктор.

\* Число N - Размер массива для анализа

\*/

Analysis(unsigned int N);

virtual ~Analysis();

/\*\*

\* Возвращает данные о работе алгоритма сравнения

\* метода пузырька

\*/

Measurement \*getBubbleMeasurement() const;

/\*\*

\* Возвращает данные о работе алгоритма сравнения

\* быстрой сортировки Хоара

\*/

Measurement \*getQuickMeasurement() const;

private:

Measurement \*bubble, \*quick;

unsigned int N;

};

/\*\*

\* Даные о работе алгоритма

\*/

class Measurement{

public:

/\*\*

\* Создаёт новый объект класса данных о замерах,

\* с случайно сгенерированым массивом размера size.

\*/

Measurement(unsigned int size);

Measurement(const Measurement &orig);

virtual ~Measurement();

/\*\*

\* Установка точки начала замера

\*/

void begin();

/\*\*

\* Установка точки окончания замера

\*/

void end();

/\*\*

\* Возращает начальный массив;

\*/

int\* getRowArray() const;

/\*\*

\* Возращает массив для сортировки

\*/

int\* getArray() const;

/\*\*

\* Возращает размер массивов

\*/

unsigned int getArraySize() const;

/\*\*

\* Возращает время начала замера

\*/

clock\_t getBeginTime() const;

/\*\*

\* Возращает время окончания замера

\*/

clock\_t getEndTime() const;

/\*\*

\* Возращает время замера

\*/

clock\_t getDeltaTime() const;

private:

//Копия массива до сортировки

int\* rowArray;

//Массив для сортировки

int\* array;

//Размер массива

unsigned int size;

//Время начала/конца замера

clock\_t beginTime = 0, endTime = 0;

};



Рисунок 2. UML схема модуля Analysis



Рисунок 2. Алгорит выполнения сравнения алгоритмов сортировки

Так как пользователю может быть удобна запись результата в различных вариантах, был так же разработан модуль Writer (интерфейсная часть в файле Writer.cpp, реализация в файле Writer.cpp)

class Writer{

public:

virtual ~Writer(){}

virtual void write(Analysis \*a) = 0;

protected:

std::ofstream outfile;

};

/\*

\* Выводит информацию о замере в файл формата json

\*/

class JsonWriter : public Writer{

public:

JsonWriter(unsigned int N, unsigned int L);

~JsonWriter();

void write(Analysis \*a);

private:

unsigned int i = 1, L;

};

/\*

\* Выводит информацию о замере в файл формата txt

\*/

class TxtWriter : public Writer{

public:

TxtWriter(unsigned int N, unsigned int L);

~TxtWriter();

void write(Analysis \*a);

};

/\*

\* Выводит подробную информацию о замере в файл формата txt

\*/

class DetailTxtWriter : public Writer{

public:

DetailTxtWriter(unsigned int N, unsigned int L);

~DetailTxtWriter();

void write(Analysis \*a);

private:

void printArray(int\* array);

unsigned int i = 1, L, N;

};



Рисунок 2. UML схема модуля Writer

## 3.ИНСТРУКЦИЯ ПО УСТАНОВКЕ И ТРЕБОВАНИЯ К СИСТЕМЕ

Для установки программы, необходимо распаковать архив Analysis.zip в любую папку. Для использования визуализатора необходимо установить JRE 8 или новее.

Системные требования для работы программы: Операционная система семейства Windows NT, (опционально) JRE 8 или новее.

## 4.РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Программа работает в двух режимах:

* Получая данные через параметры при вызове в консоли, без UI;
* Получая данные через ввод пользователя в самом приложении, с UI.

Вывод производится в файлы формата “N\_L.O”, где N – Размер массива,  
L – Кол-во тестов, O – Выбранный формат вывода.

Для визуализации результатов работы программы рекомендуется использовать программу GUI.jar

Инструкция по использованию программы с UI.

1. Запустите analysis.exe
2. Введите размер анализируемого массива

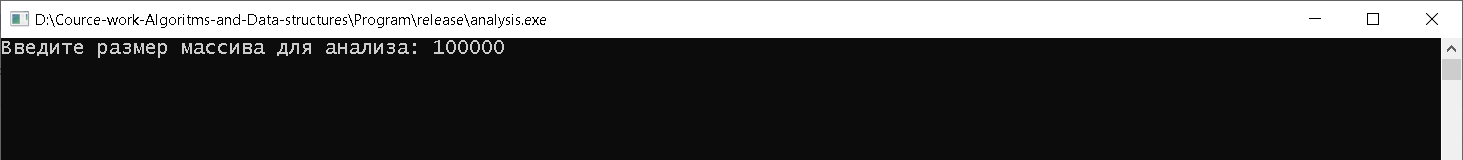


Рисунок 4.1 Пример ввода размера массива

1. Введите количество тестов

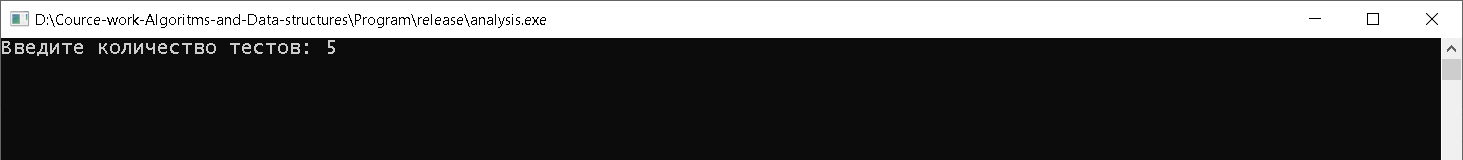


Рисунок 4.2 Пример ввода кол-ва тестов

1. Выберите формат вывода

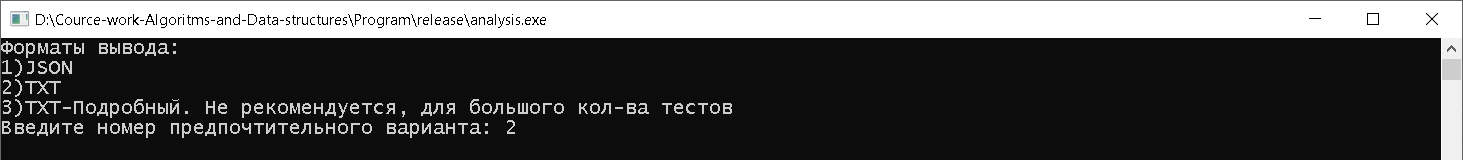


Рисунок 4.3 Пример ввода формата вывода

1. Дождитесь окончания работы программы

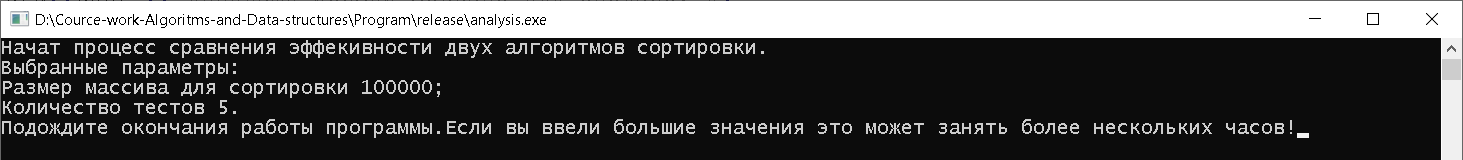


Рисунок 4.5 Скриншот с информацией о работе программы

1. Закройте программу

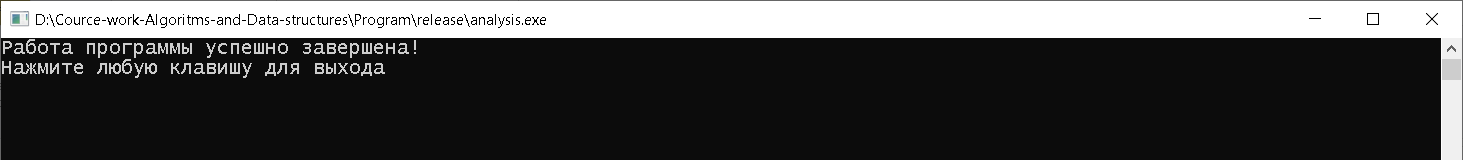


Рисунок 4.6 Завершение работы программы

Инструкция по использованию программы без UI.

1. Если используете терминал перейдите в директорию с программой, если .bat файл, создайте его в папке с программой.
2. Используйте данную команду “*analysis N L O*”, где N – размер массива для тестов, L – количество тестов, O – тип вывода (1-.json, 2-.txt, 3-подробный .txt)

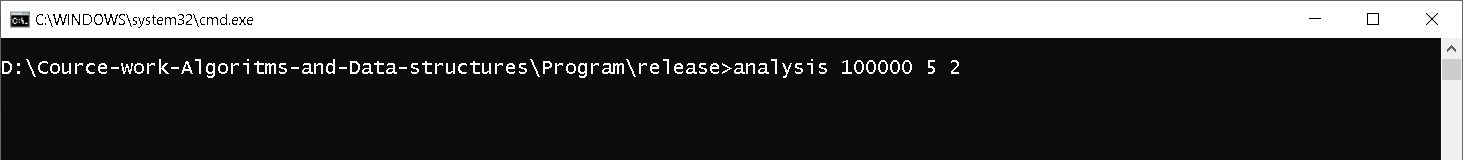


Рисунок 4.7 Пример ввода команды в консоль

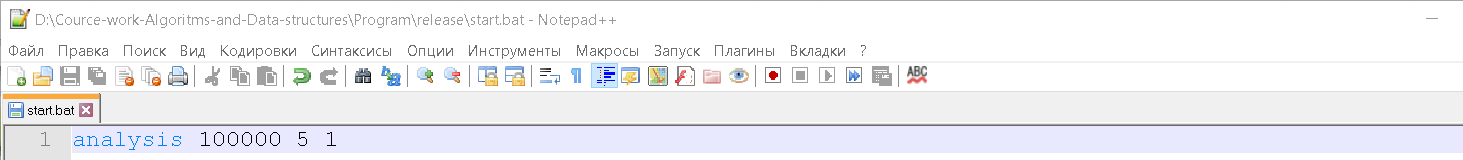


Рисунок 4.8 Пример команды в .bat файле

1. Запустите выполнение в консоли или .bat файле
2. Дождитесь завершения работы программы

Инструкция по использованию программы с визуализатором GUI.jar.

1. Запустите программу GUI.jar
2. В главном меню выберете желаемый режим работы программы

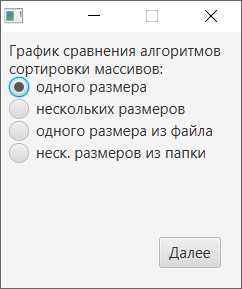


Рисунок 4.9 Главное меню визуализатора

1. Если выбраны первые два варианта, необходимо ввести запрошенные программой данные

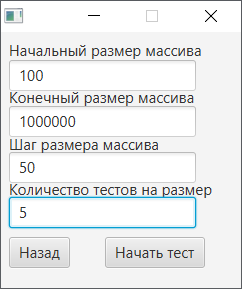
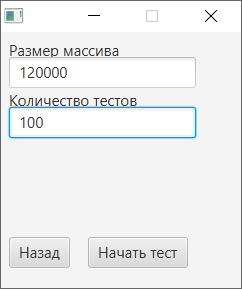


Рисунок 4.10 Меню ввода данных

Если выбраны варианты с выбором файла, откроется файловый менеджер в котором нужно выбрать .json файл или папку с .json файлами с данными об сравнениях.

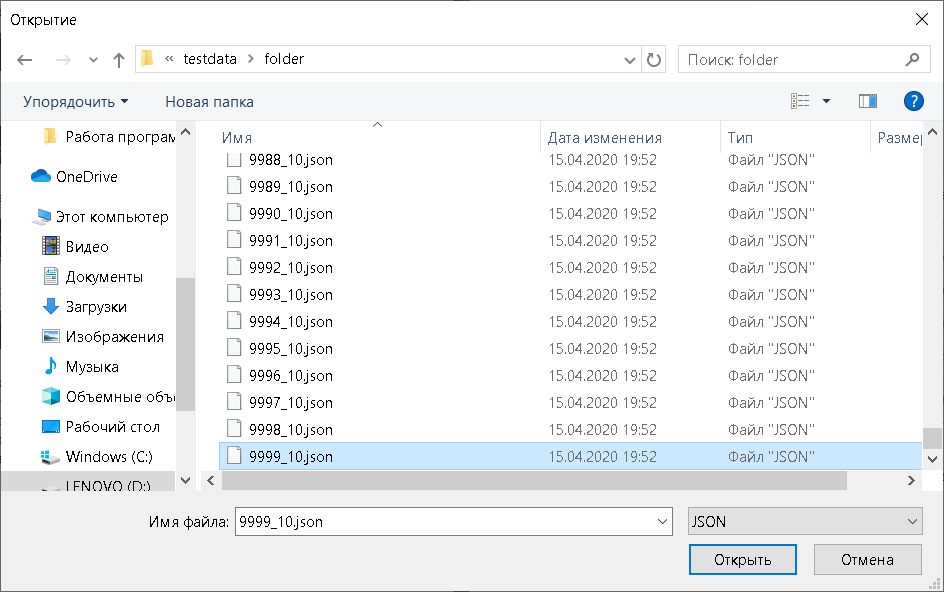


Рисунок 4.11 Выбор файла для построения графика

1. Для первых двух вариантов, необходимо дождаться окончания работы сравнения алгоритмов сортировки. Если был выбран режим сравнения нескольких размеров массивов, будет показан прогресс операции

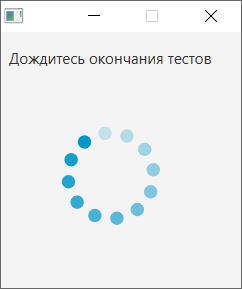
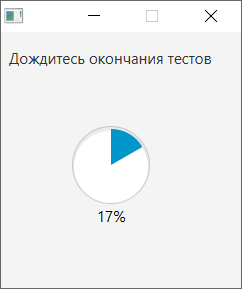


Рисунок 4.12 Ожидание окончания работы программы

1. График построен, есть возможность распечатать его или сохранить в качестве изображения

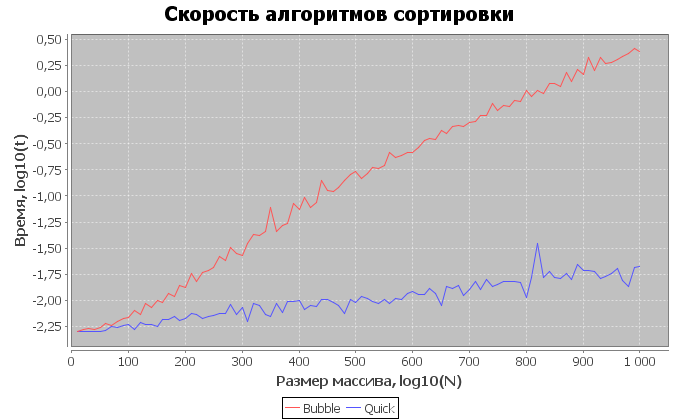


Рисунок 4.13 График визуализирующий данные сравнения алгоритмов

# БИБЛИОГРАФИЧЕСКИЙ СПИСОК

Кормен, Томас.Х. Алгоритмы: построение и анализ /Томас.Х. Кормен, Чарльз.И. Лейзерсон, Рональд.Л. Ривест, Штайн Клиффорд; Под ред. И. В. Красикова. — 2-е изд. — Вильямс, 2005. — 219с.

# ПРИЛОЖЕНИЕ А

## А.1 Sort.hpp

#ifndef SORT\_HPP

#define SORT\_HPP

void bubbleSort(int\* array, int n);

void quickSort(int\* array,int first, int last);

#endif //MEASUREMENT\_HPP

## А.2 Sort.cpp

#include "Sort.hpp"

void bubbleSort(int\* array, int n){

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

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

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

int tmp = array[j];

array[j] = array[j + 1];

array[j + 1] = tmp;

}

}

}

}

void quickSort(int\* array,int first, int last){

if (first < last){

int left = first, right = last, middle = array[(left + right) / 2];

do{

while (array[left] < middle){

left++;

}

while (array[right] > middle){

right--;

}

if (left <= right){

int tmp = array[left];

array[left] = array[right];

array[right] = tmp;

left++;

right--;

}

} while (left <= right);

quickSort(array, first, right);

quickSort(array, left, last);

}

}

## А.3 Measurement.hpp

#include <ctime>

#ifndef MEASUREMENT\_HPP

#define MEASUREMENT\_HPP

/\*\*

\* Даные о работе алгоритма

\*/

class Measurement{

public:

/\*\*

\* Создаёт новый объект класса данных о замерах,

\* с случайно сгенерированым массивом размера size.

\*/

Measurement(unsigned int size);

Measurement(const Measurement &orig);

virtual ~Measurement();

/\*\*

\* Установка точки начала замера

\*/

void begin();

/\*\*

\* Установка точки окончания замера

\*/

void end();

/\*\*

\* Возращает начальный массив;

\*/

int\* getRowArray() const;

/\*\*

\* Возращает массив для сортировки

\*/

int\* getArray() const;

/\*\*

\* Возращает размер массивов

\*/

unsigned int getArraySize() const;

/\*\*

\* Возращает время начала замера

\*/

clock\_t getBeginTime() const;

/\*\*

\* Возращает время окончания замера

\*/

clock\_t getEndTime() const;

/\*\*

\* Возращает время замера

\*/

clock\_t getDeltaTime() const;

private:

int\* rowArray;//Копия массива до сортировки

int\* array;//Массив для сортировки

unsigned int size;//Размер массива

clock\_t beginTime = 0, endTime = 0;//Время начала/конца замера

};

#endif //MEASUREMENT\_HPP

## А.4 Measurement.cpp

#include "Measurement.hpp"

#include <random>

//Максимально возможное сгенерированное число

#define MAX 999999999

Measurement::Measurement(unsigned int size) : size(size) {

static std::default\_random\_engine dre(time(0));

array = new int[size];

rowArray = new int[size];

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

array[i] = rowArray[i] = dre() % MAX;

}

}

Measurement::Measurement(const Measurement &orig){

size = orig.size;

array = new int[size];

rowArray = new int[size];

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

array[i] = orig.array[i];

rowArray[i] = orig.array[i];

}

}

Measurement::~Measurement() {

delete[] array;

delete[] rowArray;

}

void Measurement::begin() {

beginTime = clock();

}

void Measurement::end() {

endTime = clock();

}

int \*Measurement::getRowArray() const {

return rowArray;

}

int\* Measurement::getArray() const{

return array;

}

unsigned int Measurement::getArraySize() const {

return size;

}

clock\_t Measurement::getBeginTime() const {

return beginTime;

}

clock\_t Measurement::getEndTime() const {

return endTime;

}

clock\_t Measurement::getDeltaTime() const {

return endTime - beginTime;

}

## А.5 Analysis.hpp

#include "Measurement.hpp"

#ifndef ANALYSIS\_HPP

#define ANALYSIS\_HPP

class Analysis{

public:

/\*\*

\* Конструктор.

\* Число N - Размер массива для анализа

\*/

Analysis(unsigned int N);

virtual ~Analysis();

/\*\*

\* Возвращает данные о работе алгоритма сравнения метода пузырька

\*/

Measurement \*getBubbleMeasurement() const;

/\*\*

\* Возвращает данные о работе алгоритма сравнения быстрой сортировки Хоара

\*/

Measurement \*getQuickMeasurement() const;

private:

Measurement \*bubble, \*quick;

unsigned int N;

};

#endif //ANALYSIS\_HPP

## А.6 Analysis.cpp

#include "Analysis.hpp"

#include "Sort.hpp"

Analysis::Analysis(unsigned int N){

bubble = new Measurement(N);

quick = new Measurement(\*bubble);

bubble->begin();

bubbleSort(bubble->getArray(), N);

bubble->end();

quick->begin();

quickSort(quick->getArray(), 0, N - 1);

quick->end();

}

Analysis::~Analysis(){

delete bubble;

delete quick;

}

Measurement \*Analysis::getBubbleMeasurement() const{

return bubble;

}

Measurement \*Analysis::getQuickMeasurement() const{

return quick;

}

## А.7 Writer.hpp

#include "Analysis.hpp"

#include <fstream>

#ifndef WRITER\_HPP

#define WRITER\_HPP

class Writer{

public:

virtual ~Writer(){}

virtual void write(Analysis \*a) = 0;

protected:

std::ofstream outfile;

};

/\*

\* Выводит информацию о замере в файл формата json

\*/

class JsonWriter : public Writer{

public:

JsonWriter(unsigned int N, unsigned int L);

~JsonWriter();

void write(Analysis \*a);

private:

unsigned int i = 1, L;

};

/\*

\* Выводит информацию о замере в файл формата txt

\*/

class TxtWriter : public Writer{

public:

TxtWriter(unsigned int N, unsigned int L);

~TxtWriter();

void write(Analysis \*a);

};

/\*

\* Выводит подробную информацию о замере в файл формата txt

\*/

class DetailTxtWriter : public Writer{

public:

DetailTxtWriter(unsigned int N, unsigned int L);

~DetailTxtWriter();

void write(Analysis \*a);

private:

void printArray(int\* array);

unsigned int i = 1, L, N;

};

#endif //WRITER\_HPP

## А.8 Writer.cpp

#include "Writer.hpp"

//void DetailTxtWriter::write(Analysis \*a) -- CLOCKS\_PER\_SEC

#include <ctime>

JsonWriter::JsonWriter(unsigned int N, unsigned int L) : L(L){

outfile.open(std::to\_string(N) + "\_" + std::to\_string(L) + ".json");

if(!outfile.is\_open())

exit(EXIT\_FAILURE);

outfile << "{\n\t\"N\" : " << N << ",\n\t\"L\" : " << L << ",\n\t\"Measurements\": [\n";

}

JsonWriter::~JsonWriter(){

outfile << "\t]\n}";

outfile.close();

}

void JsonWriter::write(Analysis \*a){

outfile << "\t\t{ \"bubble\" : " << a->getBubbleMeasurement()->getDeltaTime()

<< ", \"quick\" : " << a->getQuickMeasurement()->getDeltaTime() << "}";

if (i++ < L)

outfile << ',';

outfile << '\n';

}

TxtWriter::TxtWriter(unsigned int N, unsigned int L){

outfile.open(std::to\_string(N) + "\_" + std::to\_string(L) + ".txt");

if(!outfile.is\_open())

exit(EXIT\_FAILURE);

outfile << N << " " << L << "\n";

}

TxtWriter::~TxtWriter(){

outfile.close();

}

void TxtWriter::write(Analysis \*a){

outfile << a->getBubbleMeasurement()->getDeltaTime() << " "

<< a->getQuickMeasurement()->getDeltaTime() << "\n";

}

DetailTxtWriter::DetailTxtWriter(unsigned int N, unsigned int L) : N(N), L(L){

outfile.open(std::to\_string(N) + "\_" + std::to\_string(L) + ".txt");

if(!outfile.is\_open())

exit(EXIT\_FAILURE);

outfile << "Размер массива " << N << ".\tКоличество тестов " << L << ".\n\n";

}

DetailTxtWriter::~DetailTxtWriter(){

outfile.close();

}

void DetailTxtWriter::write(Analysis \*a){

outfile << "--------------------[" << i++ << '/' << L << "]--------------------\n"

<< "Сортировка пузырьком.\nВремя(такты): " << a->getBubbleMeasurement()->getDeltaTime()

<< "\nВремя(секунды): " << a->getBubbleMeasurement()->getDeltaTime()/CLOCKS\_PER\_SEC

<< "\n----------Массив до сортировки----------\n";

printArray(a->getBubbleMeasurement()->getRowArray());

outfile << "\n----------Отсортированый массив----------\n";

printArray(a->getBubbleMeasurement()->getArray());

outfile << "\n--------------------------------------------------"

<< "\nБыстрая сортировка Хоара.\nВремя(такты): "

<< a->getQuickMeasurement()->getDeltaTime() << "\nВремя(секунды): "

<< a->getQuickMeasurement()->getDeltaTime()/CLOCKS\_PER\_SEC

<< "\n----------Массив до сортировки----------\n";

printArray(a->getQuickMeasurement()->getRowArray());

outfile << "\n----------Отсортированый массив----------\n";

printArray(a->getQuickMeasurement()->getArray());

outfile << "\n\n\n\n";

}

void DetailTxtWriter::printArray(int\* array){

int temp = N-1;

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

outfile << array[i] << ',';

}

outfile << array[temp];

}

## А.9 Main.cpp

#include <iostream>

#include <conio.h>

#include "Writer.hpp"

//Работа программы с использованием пользовательского интерфейса в консоли

void uiMod();

//

void takeAnalysis(unsigned int N,unsigned int L, unsigned int outformat);

/\*\*

\* Старт программы выполняющей анализ алгоритмов сортировки

\* Если программе передаётся в качестве аргументов

\* размер массива (argv[1]), количество тестов (argv[2]) и формат вывода (argv[3]),

\* то программа выполняется без вывода/ввода консоли.

\* Во всех остальных случаях программа считывает размер массива и

\* кол-во тестов с помошью консольного пользовательского интерфейса.

\*/

int main(int argc, char\*\* argv){

if(argc != 4)

uiMod();

else

takeAnalysis(atoi(argv[1]), atoi(argv[2]), atoi(argv[3]));

return 0;

}

void uiMod(){

unsigned int N,L,out\_mode;

std::cout << "Введите размер массива для анализа: ";

std::cin >> N;

system("cls");

std::cout << "Введите количество тестов: ";

std::cin >> L;

system("cls");

std::cout << "Форматы вывода:\n1)JSON\n2)TXT\n3)TXT-Подробный. Не рекомендуется, для большого кол-ва тестов";

std::cout << "\nВведите номер предпочтительного варианта: ";

std::cin >> out\_mode;

system("cls");

std::cout << "Начат процесс сравнения эффекивности двух алгоритмов сортировки.\n"

<< "Выбранные параметры:\nРазмер массива для сортировки " << N

<< ";\nКоличество тестов " << L << ".\nПодождите окончания работы программы. "

<< "Если вы ввели большие значения это может занять более нескольких часов!";

takeAnalysis(N,L, out\_mode);

system("cls");

std::cout << "Работа программы успешно завершена!\nНажмите любую клавишу для выхода";

getch();

}

#define JSON\_WRITER 1

#define TXT\_WRITER 2

#define DETAIL\_TXT\_WRITER 3

void takeAnalysis(unsigned int N,unsigned int L, unsigned int outformat){

Writer \*w;

switch(outformat){

case JSON\_WRITER:

w = new JsonWriter(N,L);

break;

case TXT\_WRITER:

w = new TxtWriter(N,L);

break;

case DETAIL\_TXT\_WRITER:

w = new DetailTxtWriter(N,L);

break;

default:

exit(EXIT\_FAILURE);

}

for(unsigned int i = 0; i < L; i++){

Analysis \*a = new Analysis(N);

w->write(a);

delete a;

}

delete w;

}