**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ**

**УЧРЕЖДЕНИЕ**

**ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ**

**«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

**ИНЖЕНЕРНО-ТЕХНОЛОГИЧЕСКАЯ АКАДЕМИЯ**

**Институт компьютерных технологий и информационной безопасности**

**Кафедра математического обеспечения и применения ЭВМ**

**Отчет по лабораторной работе №3**

**по курсу «Структуры и алгоритмы обработки данных»**

**на тему**

**«Алгоритмы сортировки массивов»**

**Вариант №2**

Выполнил:

студент гр. КТбо2-1

Самардак А.В

Проверил:

Доцент кафедры МОП ЭВМ

Дроздов С.Н

**Цель работы**

1. Изучение основных идей, алгоритмов, приемов программирования, связанных с классической задачей сортировки массивов;
2. Получение навыков сравнения алгоритмов по эффективности.

**Формулировка задачи**

В данной лабораторной работе требуется запрограммировать и протестировать алгоритмы внутренней сортировки, указанные в варианте задания.

В данном задания требуется запрограммировать два алгоритма сортировки (Choice и HeapSort) и проверить их работу на ряде тестовых примеров.

В качестве тестовых массивов следует использовать:

1. массив из 1000 первых натуральных чисел в порядке возрастания (т.е. пример уже отсортированного массива);
2. массив из 1000 первых натуральных чисел в порядке убывания (т.е. пример массива, отсортированного «наоборот»);
3. не менее 4 сгенерированных массивов псевдослучайных чисел разного размера, от 100 до 100000 элементов (максимальный размер массива может быть изменен в зависимости от производительности процессора).

Таблица результатов должна для каждого алгоритма и для каждого тестового массива включать время работы алгоритма, число выполненных сравнений и присваиваний элементов массива.

По крайней мере, для одного не очень большого примера следует для визуальной проверки правильности работы алгоритма выдать на экран весь сортируемый массив до и после сортировки.

**Листинг программы на языке С++**

**Main.cpp**

#include "Console.h"

int main()

{

setlocale(LC\_ALL, "Russian");

Console().start();

return 0;

}

**ArrayOfNumber.h**

#pragma once

class ArrayOfNumber {

public:

ArrayOfNumber();

ArrayOfNumber(int size);

~ArrayOfNumber();

int& operator [] (int i) const;

private:

int\* \_arr;

};

**ArrayOfNumber.cpp**

#include "ArrayOfNumber.h"

ArrayOfNumber::ArrayOfNumber(): \_arr(new int[1000]) {}

ArrayOfNumber::ArrayOfNumber(int size): \_arr(new int[size]) {}

ArrayOfNumber::~ArrayOfNumber()

{

delete[] \_arr;

}

int& ArrayOfNumber::operator [] (int i) const

{

return \_arr[i];

}

**Sorting.h**

#pragma once

#include <algorithm>

#include "ArrayOfNumber.h"

class Sorting {

public:

virtual ~Sorting() {}

virtual void SortArr(const ArrayOfNumber& arr, int size) const = 0;

};

**Choice.h**

#pragma once

#include "Sorting.h"

class Choice final : public Sorting {

public:

void SortArr(const ArrayOfNumber& arr, int size) const;

};

**Choice.cpp**

#include "Choice.h"

void Choice::SortArr(const ArrayOfNumber& arr, int size) const

{

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

{

int i\_min = i;

for (int j = i + 1; j < size; ++j)

{

if (arr[j] < arr[i\_min])

{

i\_min = j;

}

}

std::swap(arr[i], arr[i\_min]);

}

}

**HeapSort.h**

#pragma once

#include "Sorting.h"

class HeapSort final : public Sorting {

public:

void SortArr(const ArrayOfNumber& arr, int size) const;

private:

void heapify(const ArrayOfNumber& arr, int size, int i) const;

};

**HeapSort.cpp**

#include "HeapSort.h"

void HeapSort::SortArr(const ArrayOfNumber& arr, int size) const {

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

{

heapify(arr, size, i);

}

for (int i = size - 1; i >= 0; i--)

{

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

heapify(arr, i, 0);

}

}

void HeapSort::heapify(const ArrayOfNumber& arr, int size, int i) const

{

int largest = i;

int l = 2 \* i + 1;

int r = 2 \* i + 2;

if (l < size && arr[l] > arr[largest])

{

largest = l;

}

if (r < size && arr[r] > arr[largest])

{

largest = r;

}

if (largest != i)

{

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

heapify(arr, size, largest);

}

}

**Console.h**

#pragma once

#include <iostream>

#include "Sorting.h"

#include "ShapeFactory.h"

#include "ScanFile.h"

#include "PrintFile.h"

#include "ArrayOfNumber.h"

class Console {

public:

~Console();

void start();

private:

int dialog\_get\_check();

std::string dialog\_get\_name\_file();

int dialog\_get\_size();

Sorting\* \_sort = nullptr;

};

**Console.cpp**

#include "Console.h"

Console::~Console()

{

delete \_sort;

}

void Console::start()

{

try

{

int check = dialog\_get\_check();

\_sort = ShapeFactory().createShape(check);

std::string name\_file = dialog\_get\_name\_file();

int size = dialog\_get\_size();

ArrayOfNumber arr(size);

ScanFile().Recording(arr, size, name\_file);

\_sort->SortArr(arr, size);

PrintFile().outFile(arr, size);

} catch (const std::exception& exept)

{

std::cout << exept.what() << std::endl;

}

}

int Console::dialog\_get\_check()

{

std::cout << "Какую сортировку выбрать?:\n 0. Choice\n 1. HeapSort" << std::endl;

int check;

std::cin >> check;

return check;

}

std::string Console::dialog\_get\_name\_file()

{

std::cout << "Название файла?:" << std::endl;

std::string name\_file;

std::cin >> name\_file;

return name\_file;

}

int Console::dialog\_get\_size()

{

std::cout << "Сколько элементов считать с файла?: " << std::endl;

int size;

std::cin >> size;

return size;

}

**PrintFile.h**

#pragma once

#include <fstream>

#include "ArrayOfNumber.h"

class PrintFile {

public:

void outFile(const ArrayOfNumber& arr, int size) const;

};

**PrintFile.cpp**

#include "PrintFile.h"

void PrintFile::outFile(const ArrayOfNumber& arr, int size) const

{

std::ofstream fout("sort\_array.txt");

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

{

fout << arr[i] << "\n";

}

fout.close();

}

**ScanFile.h**

#pragma once

#include <fstream>

#include <stdexcept>

#include "ArrayOfNumber.h"

class ScanFile {

public:

void Recording(const ArrayOfNumber& arr, int size, const std::string& name\_file) const;

};

**ScanFile.cpp**

#include "ScanFile.h"

void ScanFile::Recording(const ArrayOfNumber& arr, int size, const std::string& name\_file) const

{

std::ifstream fin(name\_file);

if (!fin.is\_open())

{

throw std::exception("Нет такого файла...");

}

else

{

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

{

if (!(fin >> arr[i]))

{

throw std::exception("Недопустимое количество элементов...");

}

}

}

fin.close();

}

**ShapeFactory.h**

#pragma once

#include "Sorting.h"

#include "Choice.h"

#include "HeapSort.h"

class ShapeFactory {

public:

Sorting\* createShape(int check);

};

**ShapeFactory.cpp**

#include "ShapeFactory.h"

Sorting\* ShapeFactory::createShape(int check)

{

switch (check)

{

case 0:

{

return new Choice();

}

case 1:

{

return new HeapSort();

}

default:

throw std::exception("Нет подходящей конструкции класса!");

}

}

**Заключение**

Дан вариант задания: запрограммировать алгоритм пирамидальной сортировки и алгоритм простого выбора, получить статистику работы алгоритмов на ряде массивов.

Ниже приводится таблица результатов работы программы для ряда тестовых массивов. Использовался процессор Ryzen 5-2600 3.4GHz.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | **Пирамидальная сортировка (HeapSort)** | | | **Алгоритм простого выбора (SelectionSort)** | | |
| Данные в массиве | N | Время (мс) | Сравнения | Присваивания | Время (мс) | Сравнения | Присваивания |
| Возрастающие | 1000 | 1.632 | 22422 | 10711 | 1.618 | 499500 | 999 |
| Убывающие | 1000 | 1.463 | 19626 | 9313 | 1.662 | 499500 | 999 |
| Случайные | 100 | 0.105 | 1466 | 683 | 0.028 | 4950 | 99 |
| Случайные | 1000 | 1.581 | 21146 | 10073 | 1.384 | 499500 | 999 |
| Случайные | 10000 | 21.682 | 278478 | 134239 | 128.329 | 49995000 | 9999 |
| Случайные | 100000 | 279.456 | 3449772 | 1674886 | 12496.8 | 704982704 | 99999 |