**Министерство образования и науки Российской Федерации**

Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский университет

«Высшая школа экономики»

Кафедра Компьютерной безопасности

**ОТЧЕТ**

**К ЛАБОРАТОРНОЙ РАБОТЕ №1**

**по дисциплине**

**«Методы программирования»**

## Постановка задачи.

1) Реализовать на языке С++ сортировки для массива объектов в соответствии с вариантом.

2) Перегрузить операторы сравнения (>, <, >=, <=) для сравнения объектов.

3) Входные данные для сортировки массива обязательно считывать из внешних источников: текстовый файл, файл MS Excel, MS Access, данные из СУБД (любое на выбор).

4) Выбрать 7-10 наборов данных для сортировки размерности от 100 и более (но не менее 100000). Засечь (программно) время сортировки каждым алгоритмом. По полученным точкам построить графики зависимости времени сортировки от размерности массива для каждого из алгоритмов сортировки на одной оси координат. Сделать вывод о том, в каком случае, какой из методов лучше применять. Графики можно строить программно или в любой из прикладных программ (MS Excel, Matlab, MathCad и т.д.).

5) Сделать отчет, содержащий титульный лист, код программы со спецификациями каждого метода и подробными комментариями, графики скоростей сортировок и выводы.

**Варианты сортировок:**

а) Сортировка выбором

в) Сортировка простыми вставками

г) Шейкер-сортировка

**Код программы:**

**main.cpp** – файл вызова сортировок, таймер работы функции, реализация функций сортировок

**sort\_class.cpp** – реализация класса, реализация конструктора, реализация перегрузок операторов

**sort\_class.hpp** – объявление класса, перегрузок и конструктора

**generator.cpp** – генерация данных для обработки

## sort\_class.hpp

#ifndef SORTING\_LAB1\_SORT\_CLASS\_HPP

#define SORTING\_LAB1\_SORT\_CLASS\_HPP

#include <string>

#include <tuple>

struct pers\_information{

int year\_of\_been;

std::string squad;

int team\_number;

std::string fio;

pers\_information(int year\_of\_been,const std::string& squad,int team\_number,const std::string& fio);

bool operator <(const pers\_information& character) const;

bool operator >(const pers\_information& character) const;

bool operator <=(const pers\_information& character) const;

bool operator >=(const pers\_information& character) const;

bool operator ==(const pers\_information& character) const;

bool operator !=(const pers\_information& character) const;

};

#endif //SORTING\_LAB1\_SORT\_CLASS\_HPP

## sort\_class.cpp

#include "sort\_class.hpp"

#include <tuple>

pers\_information::pers\_information(int year\_of\_been,const std::string& squad,int team\_number,const std::string& fio)

{

this->year\_of\_been=year\_of\_been;

this->squad=squad;

this->team\_number=team\_number;

this->fio=fio;

}

bool pers\_information::operator <(const pers\_information& character) const

{

return std::tie(year\_of\_been,squad,team\_number,fio)<std::tie(character.year\_of\_been,character.squad,character.team\_number,character.fio);

}

bool pers\_information::operator >(const pers\_information& character) const

{

return std::tie(year\_of\_been,squad,team\_number,fio)>std::tie(character.year\_of\_been,character.squad,character.team\_number,character.fio);

}

bool pers\_information::operator <=(const pers\_information& character) const

{

return !(\*this>character);

}

bool pers\_information::operator >=(const pers\_information& character) const

{

return !(\*this<character);

}

bool pers\_information::operator ==(const pers\_information& character) const

{

return year\_of\_been==character.year\_of\_been && squad==character.squad && team\_number==character.team\_number && fio==character.fio;

}

bool pers\_information::operator !=(const pers\_information& character) const

{

return !(\*this==character);

}

## Main.cpp

#include <iostream>

#include <vector>

#include <cstddef>

#include <utility>

#include "sort\_class.hpp"

#include <fstream>

#include <sstream>

#include <tuple>

#include <ctime>

// selection sorting (a)

template<typename T>

void select\_sort(std::vector<T>&mass)

{

for(size\_t ind\_i=0; ind\_i<mass.size()-1;ind\_i++) {

int min\_ind = ind\_i;

for (size\_t ind\_j = ind\_i + 1; ind\_j < mass.size(); ind\_j++) {

if (mass[ind\_j] < mass[min\_ind]) {

min\_ind = ind\_j;

}

}

if (min\_ind != ind\_i) {

std::swap(mass[ind\_i], mass[min\_ind]);

}

}

}

//simple insertion sorting (v)

template<typename T>

void insert\_sort(std::vector<T>&mass1)

{

for(int i=1;i<mass1.size();i++)

for(int j=i;j>0 && mass1[j-1]>mass1[j];j--)

std::swap(mass1[j-1],mass1[j]);

}

//Shaker sorting (g)

template<typename T>

void shaker\_sort(std::vector<T>&mass2)

{

int left,right;

left=1;

right=mass2.size()-1;

while(left<=right)

{

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

{

if(mass2[i-1]>mass2[i])

{

std::swap(mass2[i-1],mass2[i]);

}

}

right--;

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

{

if(mass2[i-1]>mass2[i])

{

std::swap(mass2[i-1],mass2[i]);

}

}

left++;

}

}

int main() {

std::vector<pers\_information> children;

std::string buf\_working;

std::ifstream in("/home/rafman/Desktop/100.txt");

while (std::getline(in,buf\_working))

{

std::istringstream for\_search(buf\_working);

std::string word;

std::vector <std::string> words;

while (getline (for\_search,word,';'))

{

words.push\_back(word);

}

pers\_information child(std::stoi(words[0]),words[1],std::stoi(words[4]),words[2]);

children.push\_back(child);

}

std::vector<pers\_information> select\_children=children;

std::vector<pers\_information> insert\_children=children;

std::vector<pers\_information> shaker\_children=children;

clock\_t select\_start=clock();

select\_sort(select\_children);

clock\_t select\_end=clock();

double select\_time = (double)(select\_end-select\_start)/CLOCKS\_PER\_SEC;

std::cout<<"selection: "<<select\_time<<'\n';

clock\_t insert\_start=clock();

insert\_sort(insert\_children);

clock\_t insert\_end=clock();

double insert\_time = (double)(insert\_end-insert\_start)/CLOCKS\_PER\_SEC;

std::cout<<"insertion: "<<insert\_time<<'\n';

clock\_t shaker\_start=clock();

shaker\_sort(shaker\_children);

clock\_t shaker\_end=clock();

double shaker\_time = (double)(shaker\_end - shaker\_start)/CLOCKS\_PER\_SEC;

std::cout<<"shaker: "<<shaker\_time<<'\n';

for (int i=0;i<select\_children.size();i++)

{

std::cout<<children[i].year\_of\_been<<" "<<children[i].squad<<" "<<hildren[i].team\_number<<" "<<children[i].fio<<'\n';

}

return 0;

}

## Generation.cpp

#include <iostream>

#include <string>

int main()

{

std::string surname[10]={"Иванов", "Петров", "Сидоров",

"Ахмедов", "Ерошкин", "Выхин",

"Андеев", "ВинДизель", "Картошкин", "Чубайс"};

std::string names[10]={"Артем","Олег","Кирилл","Антон","Владимир",

"Валерий","Виктор","Михаил","Аристарх","Никита"};

std::string last[10]={"Артемович","Олегович","Кириллович","Антонович","Владимирович",

"Валерийевич","Викторович",

"Михаилович","Аристархович","Никитович"};

std::string squad[5]={"Орлы", "Волки","Тигры","Медвежата","Каракатицы"};

std::string yaer[5]={"2007","2006","2005","2004","2003"};

std::string birth[5]={"1990","1989","1988","1987","1986"};

std::string smena[4]={"1","2","3","4"};

std::string guy="";

std::string name="";

int i;

srand(time(NULL));

std::ofstream f("/home/user/Desktop/dict1.txt",std::ios::out);

for (i=0;i<200;i++){

name=surname[rand()%10]+" "+names[rand()%10]+" "+last[rand()%10];

guy=yaer[rand()%5]+";"+squad[rand()%5]+";"+name+";"+birth[rand()%5]+";"+smena[rand()%4]+"\n";

f<<guy;

}

f.close();

return 0;

}

## 

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 100 | 1000 | 5000 | 10000 | 20000 | 50000 | 70000 | 99000 |
| Выбор | 0,000592 | 0,057408 | 1,44585 | 5,78564 | 23,1118 | 145,579 | 285,474 | 574,387 |
| Вставка | 0,000387 | 0,03627 | 0,920248 | 3,58798 | 14,2142 | 90,6965 | 178,138 | 357,014 |
| Шейкер | 0,000853 | 0,092763 | 2,39405 | 9,58798 | 39,5231 | 265,411 | 520,665 | 1078,74 |

## Вывод

По итогу исследования зависимости объема информации от времени, видно, что сортировка простыми вставками работает быстрее всего. Шейкер-сортировка работает работает медленнее всего.