# Министерство образования и науки Российской Федерации

## Федеральное государственное бюджетное образовательное учреждение высшего образования

## «Новосибирский государственный технический университет»

NSTU_Logo_blue

## Кафедра параллельных вычислительных технологий

### Домашнее задание № 1 по дисциплине «Методы построения и анализа алгоритмов»

**Сортировки**

|  |  |  |
| --- | --- | --- |
|  | Факультет: | ПМИ |
| Группа: | ПМИ-81 |
| Бригада: | 1 |
| Студенты: | Демидович Е. Ю. |
|  |  |
|  |  |
| Преподаватель: | Щукин Георгий Анатольевич |

Новосибирск

2019

1. **Условие задачи**

Требуется реализовать сортировку или поиск с использованием алгоритма/структуры данных, указанного в варианте. В качестве входных данных использовать множество записей о людях, каждая запись содержит такие обязательные поля, как имя, фамилия и год рождения. Входные данные генерировать самостоятельно (случайным образом, загружать из файлов). Алгоритм или структура данных должны поддерживать работу с произвольным числом записей.

1. **Текст программы**

**merge.h**

#pragma once

#include <fstream>

#include <iostream>

#include <vector>

#include <string>

#include <math.h>

#include <time.h>

#include <algorithm>

using namespace std;

struct Info

{

int Birth;

char Name[15];

char SurName[20];

};

enum keys

{

NAME, SURNAME, YEAR

};

void AddPerson(vector<Info>& Mas, Info temp)

{

Mas.push\_back(temp);

}

template <keys T>

struct Comparer {

bool operator ()(const Info &m1, const Info &m2) const {

return (T == YEAR ? m1.Birth > m2.Birth : strcmp(m1.Name, m2.Name) > 0);

}

};

void Input(vector<Info>& Mas, int count)

{

ifstream In("input.txt");

Info temp;

int i = 0;

while (!In.eof() && i<count)

{

In >> temp.Name;

In >> temp.SurName;

In >> temp.Birth;

AddPerson(Mas, temp);

i++;

}

In.close();

}

template <keys T>

void merge(vector<Info>& vArray, vector<Info>& vTemp, int head, int middle, int tail)

{

Comparer<T> c;

int tmp = 0, lower = head, upper = middle + 1;

while (lower <= middle && upper <= tail)

{

if (c(vArray[upper], vArray[lower]))

{

vTemp[tmp++] = vArray[lower++];

}

else

{

vTemp[tmp++] = vArray[upper++];

}

}

if (lower <= middle)

{

for (; lower <= middle; vTemp[tmp++] = vArray[lower++]);

}

else

{

for (; upper <= tail; vTemp[tmp++] = vArray[upper++]);

}

int arrayPointer = head;

for (tmp = 0; arrayPointer <= tail; vArray[arrayPointer++] = vTemp[tmp++]);

}

template <keys T>

void merge\_sort\_helper(vector<Info>& vArray, vector<Info>& vTemp, int head, int tail)

{

if (head == tail)

{

return;

}

int middle = (head + tail) / 2;

merge\_sort\_helper<T>(vArray, vTemp, head, middle);

merge\_sort\_helper<T>(vArray, vTemp, middle + 1, tail);

merge<T>(vArray, vTemp, head, middle, tail);

}

template <keys T>

void MergeSort(vector<Info>& vArray)

{

vector<Info> v(vArray.size(), vArray[0]);

merge\_sort\_helper<T>(vArray, v, 0, vArray.size() - 1);

}

void Print(const vector<Info> Mas)

{

ofstream Out("Sort.txt");

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

Out << Mas[i].Name << " " << Mas[i].SurName << " " << Mas[i].Birth << "\n";

Out.close();

}

template<keys T>

bool isSorted(vector<Info> Mas)

{

Comparer<T> c;

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

if (c(Mas[i],Mas[i + 1]))

return false;

return true;

}

**Source.cpp**

#include "merge.h"

#include "catch.hpp"

#include <chrono>

#include <conio.h>

using namespace chrono;

using namespace std;

void main()

{

int rows = 10;

for (int i = 0; i < 8; i++)

{

vector<Info> Mas;

Input(Mas, pow(rows,i));

auto mas\_copy = Mas;

auto t1 = high\_resolution\_clock::now();

MergeSort<YEAR>(Mas);

auto t2 = high\_resolution\_clock::now();

double time = duration<double>(t2 - t1).count();

printf("1)N: %.0f,\t time: %.5f sec\n", pow(rows, i), time);

auto t3 = high\_resolution\_clock::now();

sort(mas\_copy.begin(), mas\_copy.end(), Comparer<YEAR>());

auto t4 = chrono::high\_resolution\_clock::now();

double time1 = duration<double>(t4 - t3).count();

printf("2)N: %.0f,\t time: %.5f sec\n\n", pow(rows, i), time1);

}

\_getch();

}

**3.График времени**

**4. Юнит-тесты**

#include "pch.h"

#include "CppUnitTest.h"

#include "../mpiaa1lab/merge.h"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace UnitTest1

{

TEST\_CLASS(UnitTest1)

{

public:

TEST\_METHOD(TestAddPerson)

{

vector<Info> Mas;

AddPerson(Mas, { 1999,"Michael", "SurName " });

int result = 1;

Assert::AreEqual(result, (int)Mas.size());

}

TEST\_METHOD(TestOnePerson)

{

vector<Info> Mas;

AddPerson(Mas, { 1999,"Michael", "SurName " });

Assert::IsTrue(isSorted<YEAR>(Mas));

}

TEST\_METHOD(TestnotIsSorted)

{

vector<Info> Mas;

AddPerson(Mas, { 1999,"Zoe", "SurName " });

AddPerson(Mas, { 1999,"Bernard", "SurName " });

AddPerson(Mas, { 1999,"Michael", "SurName " });

AddPerson(Mas, { 1999,"Warren", "SurName " });

AddPerson(Mas, { 1999,"Dante", "SurName " });

Assert::IsFalse(isSorted<NAME>(Mas));

}

TEST\_METHOD(TestIsSorted)

{

vector<Info> Mas;

AddPerson(Mas, { 1999,"Bernard", "SurName " });

AddPerson(Mas, { 1999,"Dante", "SurName " });

AddPerson(Mas, { 1999,"Michael", "SurName " });

AddPerson(Mas, { 1999,"Warren", "SurName " });

AddPerson(Mas, { 1999,"Zoe", "SurName " });

Assert::IsTrue(isSorted<NAME>(Mas));

}

TEST\_METHOD(SortedbyName)

{

vector<Info> Mas;

AddPerson(Mas, { 1999,"Zoe", "SurName " });

AddPerson(Mas, { 1999,"Bernard", "SurName " });

AddPerson(Mas, { 1999,"Michael", "SurName " });

AddPerson(Mas, { 1999,"Warren", "SurName " });

AddPerson(Mas, { 1999,"Dante", "SurName " });

MergeSort<NAME>(Mas);

Assert::IsTrue(isSorted<NAME>(Mas));

}

TEST\_METHOD(NonSortedbyName)

{

vector<Info> Mas;

AddPerson(Mas, { 1999,"Zoe", "SurName " });

AddPerson(Mas, { 1999,"Bernard", "SurName " });

AddPerson(Mas, { 1999,"Michael", "SurName " });

AddPerson(Mas, { 1999,"Warren", "SurName " });

AddPerson(Mas, { 1999,"Dante", "SurName " });

Assert::IsFalse(isSorted<NAME>(Mas));

}

TEST\_METHOD(SortedbyYear)

{

vector<Info> Mas;

AddPerson(Mas, { 1932,"Zoe", "SurName " });

AddPerson(Mas, { 1999,"Bernard", "SurName " });

AddPerson(Mas, { 1986,"Michael", "SurName " });

AddPerson(Mas, { 1999,"Warren", "SurName " });

AddPerson(Mas, { 1923,"Dante", "SurName " });

MergeSort<YEAR>(Mas);

Assert::IsTrue(isSorted<YEAR>(Mas));

}

TEST\_METHOD(NonSortedbyYear)

{

vector<Info> Mas;

AddPerson(Mas, { 1932,"Zoe", "SurName " });

AddPerson(Mas, { 1999,"Bernard", "SurName " });

AddPerson(Mas, { 1986,"Michael", "SurName " });

AddPerson(Mas, { 1999,"Warren", "SurName " });

AddPerson(Mas, { 1923,"Dante", "SurName " });

Assert::IsFalse(isSorted<YEAR>(Mas));

}

};

}