# 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;

const int N\_MAX = 26;

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);

}

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>

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

{

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

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

{

if ((T == YEAR ? vArray[upper].Birth > vArray[lower].Birth : strcmp(vArray[upper].Name, vArray[lower].Name) > 0))

{

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>

inline 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 < Mas.size(); i++)

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

Out.close();

}

template<keys T>

bool isSorted(vector<Info> Mas)

{

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

{

if (T == YEAR)

if (Mas[i].Birth < Mas[i + 1].Birth)

return false;

else if (strcmp(Mas[i].Name, Mas[i + 1].Name) < 0)

return false;

}

return true;

}

**Source.cpp**

#include "merge.h"

#include <chrono>

using namespace chrono;

void main()

{

int rows = 10;

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

{

vector<Info> Mas;

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

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

MergeSort<YEAR>(Mas);

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

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

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

}

}

**3.График времени**

**4. Юнит-тесты**

#define CATCH\_CONFIG\_MAIN

#include "merge.h"

#include "catch.hpp"

TEST\_CASE("TestAddPerson")

{

vector<Info> Mas;

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

REQUIRE(Mas.size() == 1);

}

TEST\_CASE("Test OnePerson ")

{

vector<Info> Mas;

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

CHECK(isSorted<YEAR>(Mas));

}

TEST\_CASE("Sorted by Name")

{

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);

CHECK\_FALSE(isSorted<NAME>(Mas));

}

TEST\_CASE("NonSorted by Name")

{

vector<Info> Mas;

vector<vector<Info>> Bucket(N\_MAX);

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 " });

CHECK\_FALSE(isSorted<NAME>(Mas));

}

TEST\_CASE("Sorted by Year")

{

vector<Info> Mas;

vector<vector<Info>> Bucket(N\_MAX);

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);

CHECK\_FALSE(isSorted<YEAR>(Mas));

}

TEST\_CASE("NonSorted by Year")

{

vector<Info> Mas;

vector<vector<Info>> Bucket(N\_MAX);

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 " });

CHECK\_FALSE(isSorted<YEAR>(Mas));

}