ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ПЕРМСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ

ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Лабораторная работа

Сортировки

Многофазная сортировка

Выполнил студент группы РИС-23-3Б

Блинов А. Е.

Проверила доцент кафедры ИТАС

О. А. Полякова

2024 г.

**Постановка задачи**

Дан массив из 25 элементов. Отсортировать его при помощи многофазной сортировки.

**Блок-схема**

Скрины блок-схем добавил в отдельную папку, т.к. некоторые из них очень большие

**Код на языке C++**

**Файл NaturalMerge.h**

#pragma once

#include <iostream>

#include <fstream>

#include "CommonFunctions.h"

using namespace std;

void SortUsingNaturalMerge(ifstream\* file, int Lenght)

{

Series\* SeriesArr = new Series[Lenght];

int AmountOfSeries = 1;

int indexOfCurSeries = 0;

SeriesArr[0].elements = new int[Lenght];

SeriesArr[0].lenght = 1;

file->read((char\*)&SeriesArr[0].elements[0], sizeof(int));

for (int i = 1; i < Lenght; i++) {

int curElement;

file->read((char\*)&curElement, sizeof(curElement));

if (curElement >= SeriesArr[indexOfCurSeries].elements[SeriesArr[indexOfCurSeries].lenght - 1]) {

SeriesArr[indexOfCurSeries].elements[SeriesArr[indexOfCurSeries].lenght] = curElement;

SeriesArr[indexOfCurSeries].lenght++;

}

else {

indexOfCurSeries++;

AmountOfSeries++;

SeriesArr[indexOfCurSeries].elements = new int[Lenght];

SeriesArr[indexOfCurSeries].lenght = 1;

SeriesArr[indexOfCurSeries].elements[0] = curElement;

}

}

while (AmountOfSeries > 1) {

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

Series MergedSeries = MergeSeries(SeriesArr[i], SeriesArr[i + 1]);

delete[] SeriesArr[i].elements;

SeriesArr[i] = MergedSeries;

RemoveElement(SeriesArr, i + 1, AmountOfSeries);

AmountOfSeries--;

}

}

Series sortedContent = SeriesArr[0];

ofstream Output("F2.txt");

Output << sortedContent.elements[0];

cout << sortedContent.elements[0];

for (int i = 1; i < sortedContent.lenght; i++) {

cout << " " << sortedContent.elements[i];

Output << " " << sortedContent.elements[i];

}

Output.close();

delete[] sortedContent.elements;

delete[] SeriesArr;

}

**Файл ManyPhaseMerge.h**

#pragma once

#include <iostream>

#include <fstream>

#include "CommonFunctions.h"

using namespace std;

struct Bucket {

Series\* seriesArr;

int lenght;

};

pair<int, int> GetFibonnachiNumbers(int numberToCompareTo, pair<int, int> PreviousTwoNumbers) {

int currentNumber = PreviousTwoNumbers.first + PreviousTwoNumbers.second;

pair <int, int> PairToReturn;

if (currentNumber < numberToCompareTo)

PairToReturn = GetFibonnachiNumbers(numberToCompareTo, { PreviousTwoNumbers.second, currentNumber });

else

PairToReturn = PreviousTwoNumbers;

return PairToReturn;

}

pair<int, int> GetFibonnachiNumbers(int numberToCompareTo) { return GetFibonnachiNumbers(numberToCompareTo, { 0, 1 }); }

void SortUsingManyPhaseMerging(ifstream\* file, int Lenght) {

Series\* SeriesArr = new Series[Lenght];

int AmountOfSeries = 1;

int indexOfCurSeries = 0;

SeriesArr[0].elements = new int[Lenght];

SeriesArr[0].lenght = 1;

file->read((char\*)&SeriesArr[0].elements[0], sizeof(int));

for (int i = 1; i < Lenght; i++) {

int curElement;

file->read((char\*)&curElement, sizeof(int));

if (curElement >= SeriesArr[indexOfCurSeries].elements[SeriesArr[indexOfCurSeries].lenght - 1]) {

SeriesArr[indexOfCurSeries].elements[SeriesArr[indexOfCurSeries].lenght] = curElement;

SeriesArr[indexOfCurSeries].lenght++;

}

else {

indexOfCurSeries++;

AmountOfSeries++;

SeriesArr[indexOfCurSeries].elements = new int[Lenght];

SeriesArr[indexOfCurSeries].lenght = 1;

SeriesArr[indexOfCurSeries].elements[0] = curElement;

}

}

pair<int, int> PerfectFibonachiNumbers = GetFibonnachiNumbers(Lenght);

Bucket\* Buckets = new Bucket[3];

Buckets[0].lenght = PerfectFibonachiNumbers.second;

Buckets[1].lenght = PerfectFibonachiNumbers.first;

Buckets[2].lenght = 0;

for (int i = 0; i < 3; i++) {

for (int j = 0; j < Lenght; j++)

Buckets[i].seriesArr = new Series[Lenght];

}

for (int i = 0; i < 3; i++) {

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

Buckets[i].seriesArr[j].elements = new int[Lenght];

Buckets[i].seriesArr[j].lenght = 0;

}

}

indexOfCurSeries = 0;

for (int i = 0; i < 2 && indexOfCurSeries < AmountOfSeries; i++) {

for (int j = 0; j < Buckets[i].lenght && indexOfCurSeries < AmountOfSeries; j++) {

Buckets[i].seriesArr[j] = SeriesArr[indexOfCurSeries];

indexOfCurSeries++;

}

}

AmountOfSeries = PerfectFibonachiNumbers.first + PerfectFibonachiNumbers.second;

int IndexOfTheBiggerBucket = 0;

int IndexOfTheSmallerBucket = 1;

int IndexOfTheHelpingBucket = 2;

while (AmountOfSeries > 1) {

int NewLenght = Buckets[IndexOfTheBiggerBucket].lenght - Buckets[IndexOfTheSmallerBucket].lenght;

int curIndexOfTheHelpingBucket = 0;

int lastIndexOfSmallerBucket = Buckets[IndexOfTheSmallerBucket].lenght - 1;

for (int i = Buckets[IndexOfTheBiggerBucket].lenght - 1; i > NewLenght - 1; i--) {

Series NewSeries = MergeSeries(Buckets[IndexOfTheBiggerBucket].seriesArr[i], Buckets[IndexOfTheSmallerBucket].seriesArr[lastIndexOfSmallerBucket]);

Buckets[IndexOfTheHelpingBucket].seriesArr[curIndexOfTheHelpingBucket] = NewSeries;

Buckets[IndexOfTheHelpingBucket].lenght++;

curIndexOfTheHelpingBucket++;

lastIndexOfSmallerBucket--;

AmountOfSeries--;

}

Buckets[IndexOfTheBiggerBucket].lenght -= Buckets[IndexOfTheSmallerBucket].lenght;

for (int i = 0; i < Buckets[IndexOfTheSmallerBucket].lenght; i++)

delete[] Buckets[IndexOfTheSmallerBucket].seriesArr[i].elements;

delete[] Buckets[IndexOfTheSmallerBucket].seriesArr;

Buckets[IndexOfTheSmallerBucket].seriesArr = new Series[Lenght];

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

Buckets[IndexOfTheSmallerBucket].seriesArr[i].elements = new int[Lenght];

Buckets[IndexOfTheSmallerBucket].seriesArr[i].lenght = 0;

}

Buckets[IndexOfTheSmallerBucket].lenght = 0;

int tmp = IndexOfTheBiggerBucket;

IndexOfTheBiggerBucket = IndexOfTheHelpingBucket;

IndexOfTheHelpingBucket = IndexOfTheSmallerBucket;

IndexOfTheSmallerBucket = tmp;

}

Series sortedContent = Buckets[IndexOfTheBiggerBucket].seriesArr[0];

ofstream Output("F2.txt");

Output << sortedContent.elements[0];

cout << sortedContent.elements[0];

for (int i = 1; i < sortedContent.lenght; i++) {

cout << " " << sortedContent.elements[i];

Output << " " << sortedContent.elements[i];

}

Output.close();

for (int i = 0; i < 3; i++) {

for (int j = 0; j < Lenght; j++)

delete[] Buckets[i].seriesArr[j].elements;

delete[] Buckets[i].seriesArr;

}

delete[] Buckets;

delete[] SeriesArr;

}

**Файл CommonFinctions.h**

#pragma once

#include <iostream>

using namespace std;

struct Series {

int\* elements;

int lenght;

void PrintSeriers() {

if (lenght <= 0) return;

cout << elements[0];

for (int i = 1; i < lenght; i++)

cout << " " << elements[i];

cout << endl;

}

};

void HoareQuickSorting(int\* Arr, int startIndex, int endIndex) {

if (startIndex <= endIndex) {

int middleElement = Arr[(startIndex + endIndex) / 2];

int LeftIndex = startIndex;

int RightIndex = endIndex;

while (LeftIndex <= RightIndex) {

while (Arr[LeftIndex] < middleElement)

LeftIndex++;

while (Arr[RightIndex] > middleElement)

RightIndex--;

if (LeftIndex <= RightIndex) {

int tmp = Arr[LeftIndex];

Arr[LeftIndex] = Arr[RightIndex];

Arr[RightIndex] = tmp;

LeftIndex++;

RightIndex--;

}

}

HoareQuickSorting(Arr, startIndex, RightIndex);

HoareQuickSorting(Arr, LeftIndex, endIndex);

}

}

void HoareQuickSorting(int\* Arr, int Lenght) { if (Lenght > 1) HoareQuickSorting(Arr, 0, Lenght - 1); }

Series MergeSeries(Series& SeriesOne, Series& SeriesTwo) {

Series NewSeries;

NewSeries.elements = new int[SeriesOne.lenght + SeriesTwo.lenght];

for (int i = 0; i < SeriesOne.lenght; i++)

NewSeries.elements[i] = SeriesOne.elements[i];

int counter = SeriesOne.lenght;

for (int i = 0; i < SeriesTwo.lenght; i++) {

NewSeries.elements[counter] = SeriesTwo.elements[i];

counter++;

}

NewSeries.lenght = SeriesOne.lenght + SeriesTwo.lenght;

HoareQuickSorting(NewSeries.elements, NewSeries.lenght);

return NewSeries;

}

void RemoveElement(Series\* Arr, int Index, int Lenght) {

delete[] Arr[Index].elements;

for (int i = Index; i < Lenght - 1; i++)

Arr[i] = Arr[i + 1];

}

**Файл FileSorting.cpp**

#include <iostream>

#include <fstream>

#include "NaturalMerge.h"

#include "ManyPhaseMerge.h"

using namespace std;

typedef void(\*functionPointer)(ifstream\*, int);

int main()

{

srand(time(0));

system("chcp 1251 > Null");

functionPointer functions[] = { &SortUsingNaturalMerge, &SortUsingManyPhaseMerging };

int Lenght;

for (functionPointer function : functions) {

do

{

cout << "\nВведите кол-во элементов массива: "; cin >> Lenght;

} while (Lenght <= 1);

ofstream Output("F1.bin", ios\_base::binary);

cout << "\nИсходный массив: ";

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

int element = rand() % 101 - 50;

cout << element << " ";

Output.write(reinterpret\_cast<const char\*>(&element), sizeof(element));

}

cout << endl;

Output.close();

cout << "\nОтсортированный массив: ";

ifstream Input("F1.bin", ios\_base::binary);

function(&Input, Lenght);

Input.close();

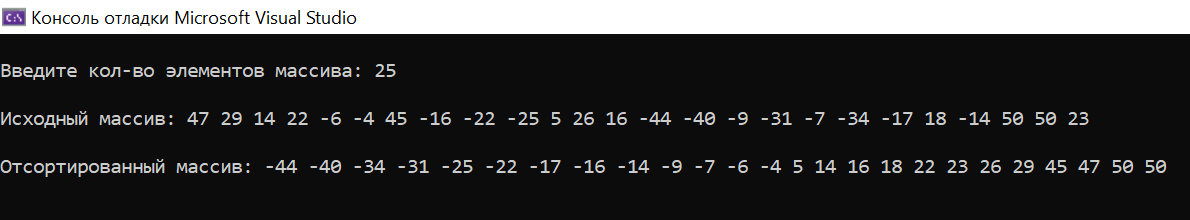
cout << endl << endl;

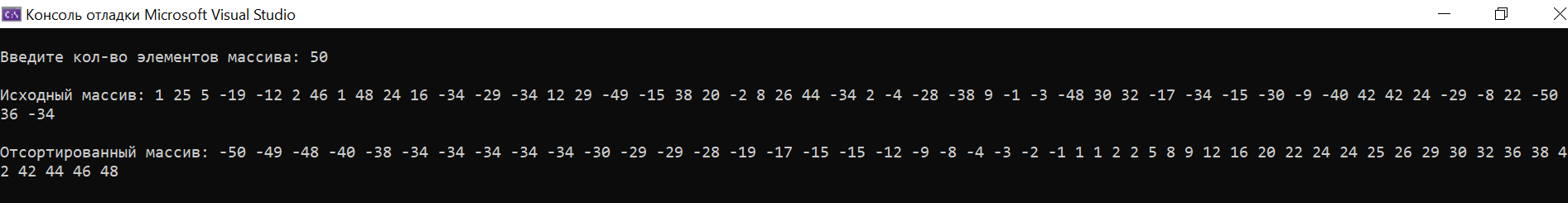
return 0;

}

}

**Результаты работы программы**

****

****

**Выводы**

Программа успешно работает, правильно сортирует исходный массив

GitHub: