Федеральное государственное автономное образовательное учреждение высшего образования «Пермский национальный исследовательский политехнический университет», ПНИПУ

ЛАБОРАТОРНАЯ РАБОТА

ПРОСТЫЕ СОРТИРОВКИ

Выполнил: студент группы РИС-23-3б

Артем Владимирович Швецов

Проверила: доцент кафедры ИТАС

Ольга Андреевна Полякова

Пермь 2024

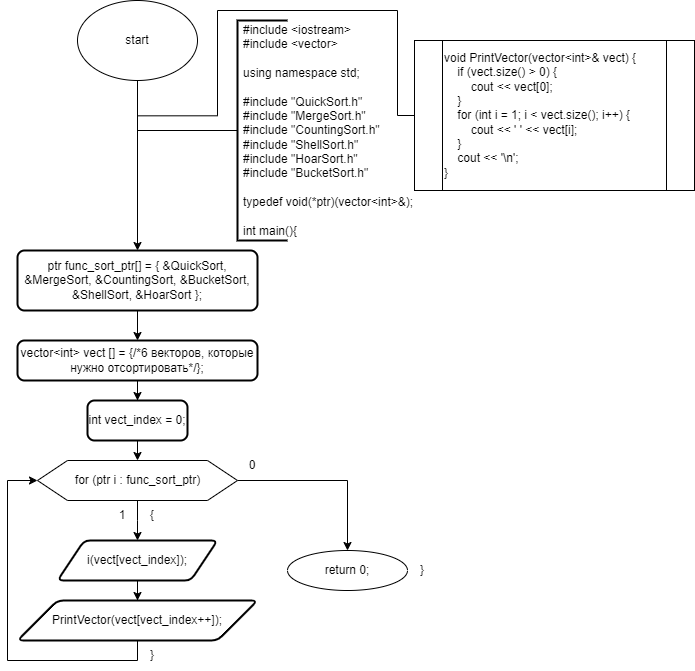
**Постановка задачи**

Создать алгоритмы сортировки массива целочисленных элементов.

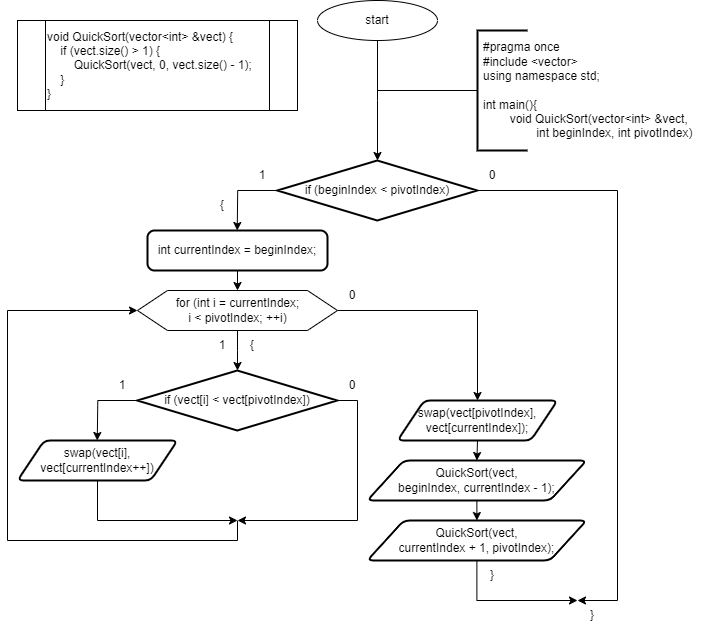
**Анализ задачи**

Создание методов в соответствии методам быстрой сортировки, слияния,

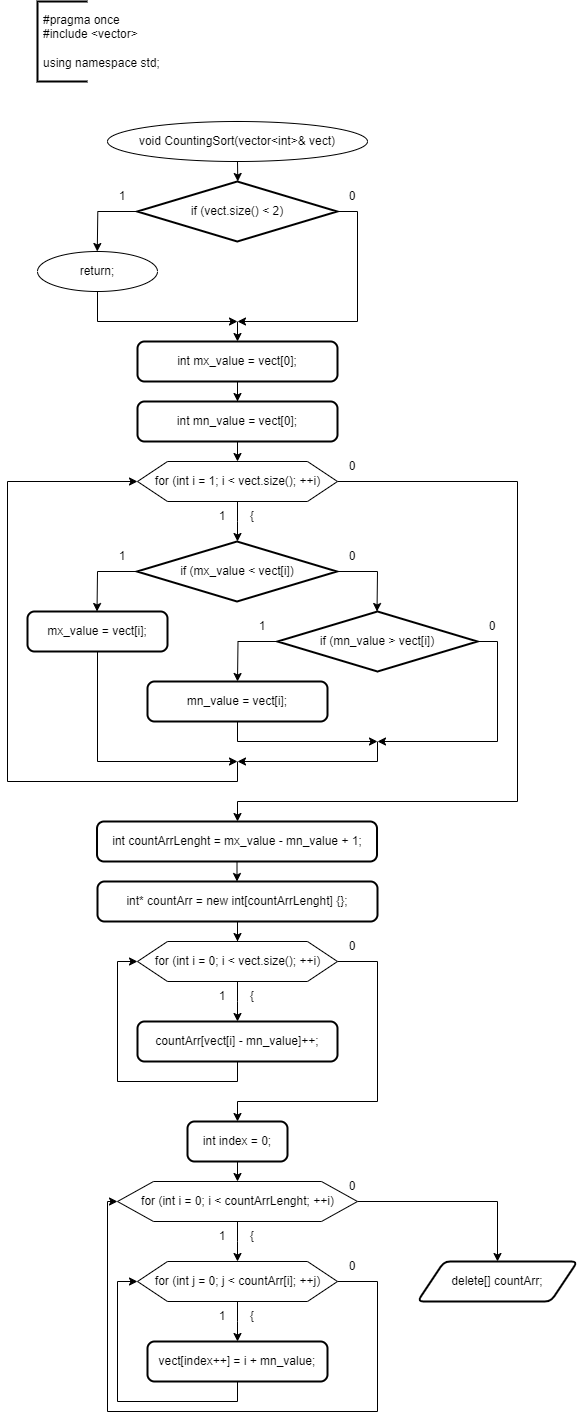
**Блок-схема**



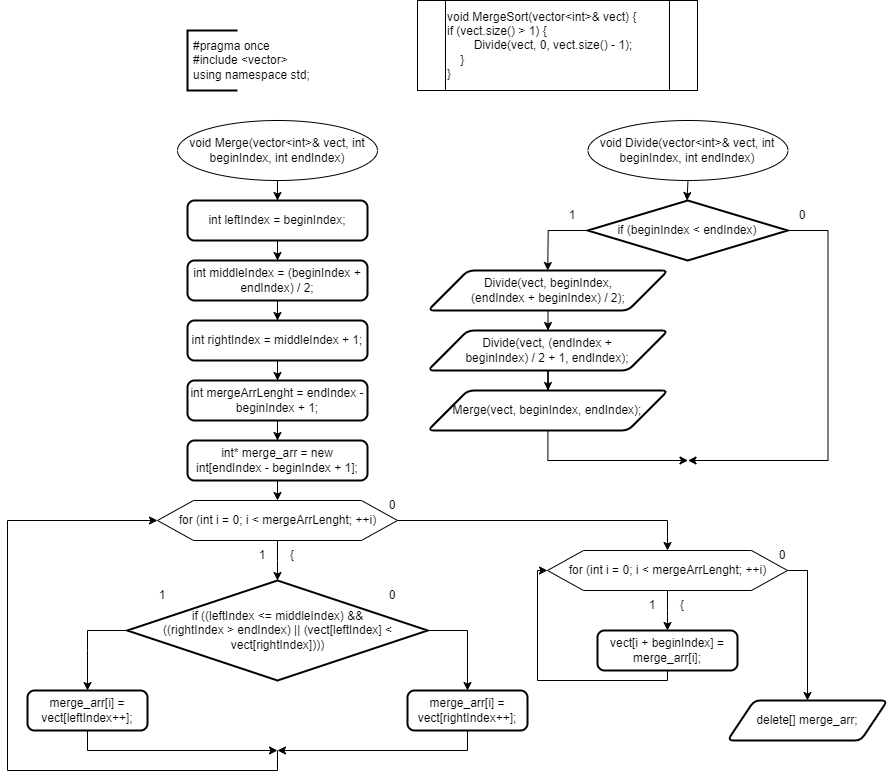
Блок схема для главного файла



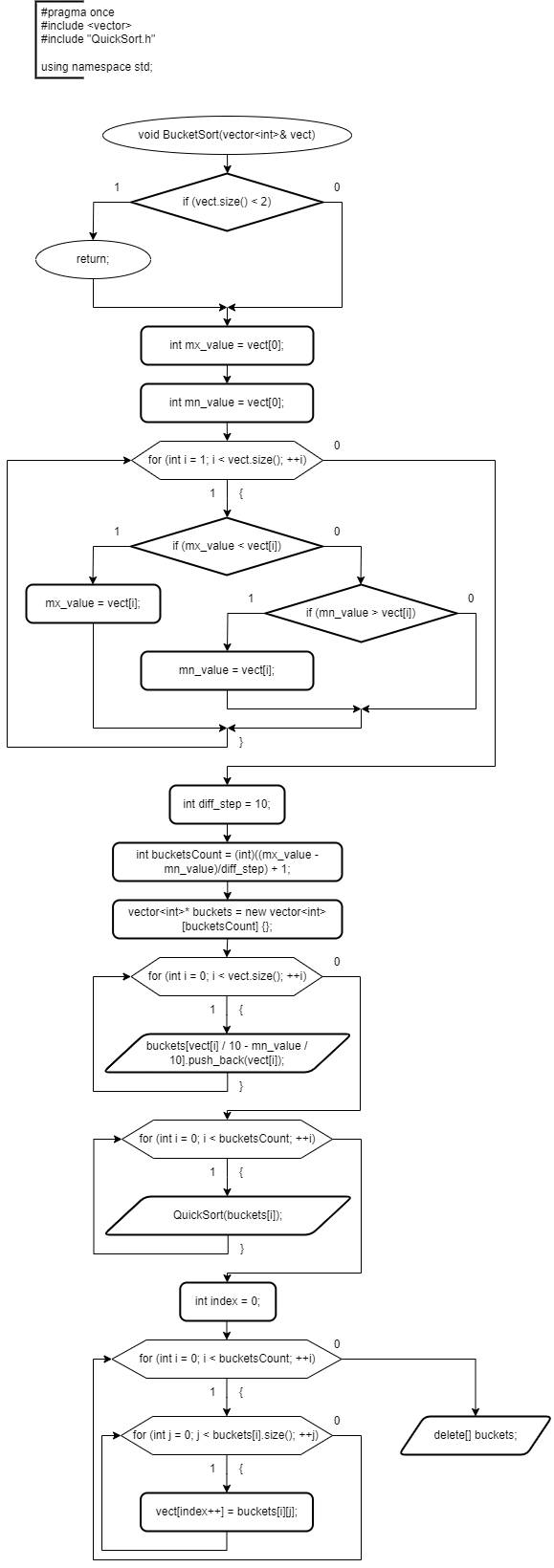
Блок схема файла QuickSort.h



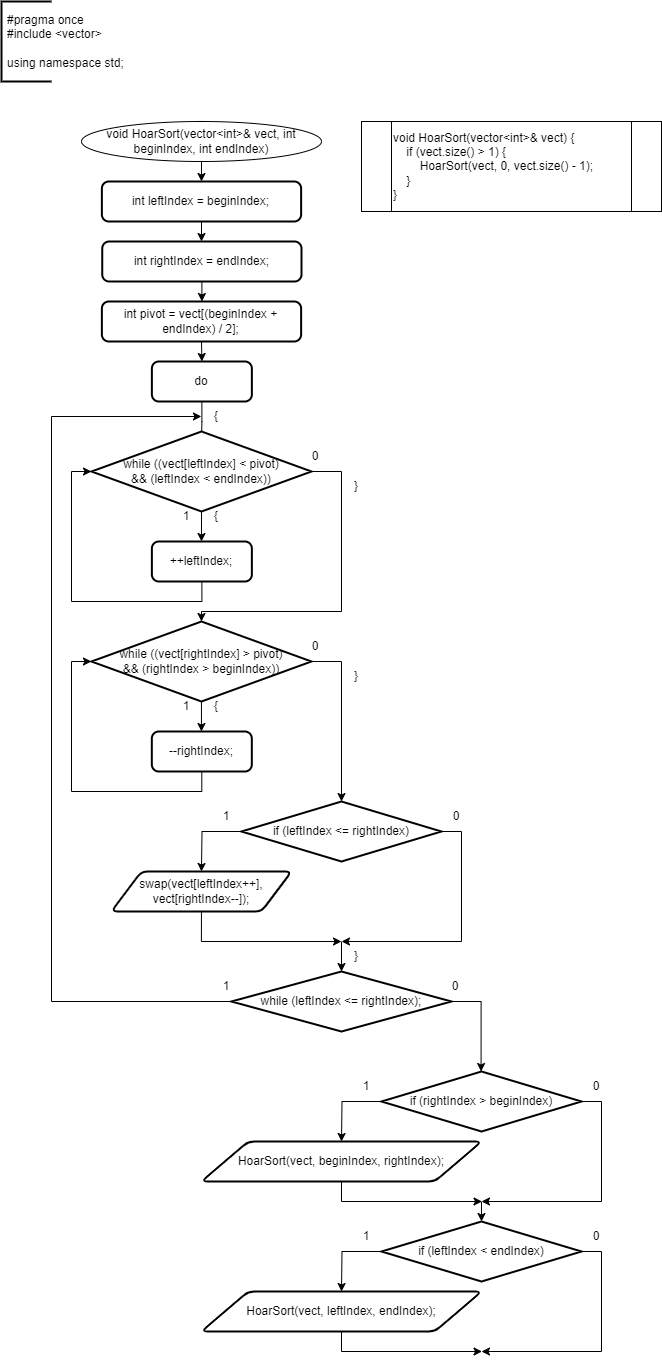
Блок схема файла CountingSort.h



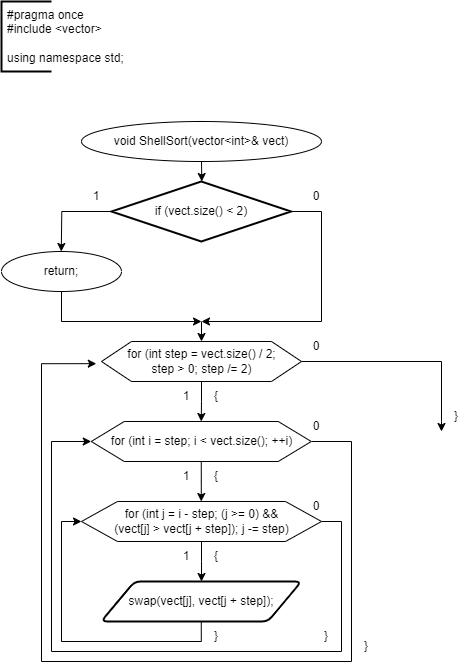
Блок схема файла MergeSort.h



Блок схема файла BucketSort.h



Блок схема файла HoarSort.h



Блок схема файла ShellSort.h

**Код**

Main

#include <iostream>

#include <vector>

using namespace std;

#include "QuickSort.h"

#include "MergeSort.h"

#include "CountingSort.h"

#include "ShellSort.h"

#include "HoarSort.h"

#include "BucketSort.h"

void PrintVector(vector<int>& vect) {

if (vect.size() > 0) {

cout << vect[0];

}

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

cout << ' ' << vect[i];

}

cout << '\n';

}

typedef void(\*ptr)(vector<int>&);

int main() {

ptr func\_sort\_ptr[] = { &QuickSort, &MergeSort, &CountingSort, &BucketSort, &ShellSort, &HoarSort };

vector<int> vect [] = {

{ 34, 55, 19, 87, 43, 4343, 73, -95, 49, 32, 100, 87, 346, 12, 3, -76, -32, 89, 109, 41, 85, 293, 44, 21, 743, 202 },

{ 34, 55, 19, 87, 43, 4343, 73, -95, 49, 32, 100, 87, 346, 12, 3, -76, -32, 89, 109, 41, 85, 293, 44, 21, 743, 202 },

{ 1, 4, 1, 3, 4, 4, 6, 4, 2, 2, 4, 3, 1, 5, 6, 3, 3, 2, 5, 6, 1, 2, 2, 3, 4, 6, 4, 4, 5, 2, 1 },

{ 1, 4, 1, 3, 4, 4, 6, 4, 2, 2, 4, 3, 1, 5, 6, 3, 3, 2, 5, 6, 1, 2, 2, 3, 4, 6, 4, 4, 5, 2, 1 },

{ 34, -77, 55, 87, 99, 20, 83, 43, 33, 73, 89, 45, 32, 75, 2, 33, 19, 7, 23, 83, 54, 30, 49, 29, 4, 99, 30, 12},

{ 34, -77, 55, 87, 99, 20, 83, 43, 33, 73, 89, 45, 32, 75, 2, 33, 19, 7, 23, 83, 54, 30, 49, 29, 4, 99, 30, 12},

};

int vect\_index = 0;

for (ptr i : func\_sort\_ptr) {

i(vect[vect\_index]);

PrintVector(vect[vect\_index++]);

}

return 0;

}

QuickSort.h

#pragma once

#include <vector>

using namespace std;

void QuickSort(vector<int> &vect, int beginIndex, int pivotIndex) {

if (beginIndex < pivotIndex) {

int currentIndex = beginIndex;

for (int i = currentIndex; i < pivotIndex; ++i) {

if (vect[i] < vect[pivotIndex]) {

swap(vect[i], vect[currentIndex++]);

}

}

swap(vect[pivotIndex], vect[currentIndex]);

QuickSort(vect, beginIndex, currentIndex - 1);

QuickSort(vect, currentIndex + 1, pivotIndex);

}

}

void QuickSort(vector<int> &vect) {

if (vect.size() > 1) {

QuickSort(vect, 0, vect.size() - 1);

}

}

MergeSort.h

#pragma once

#include <vector>

using namespace std;

void Merge(vector<int>& vect, int beginIndex, int endIndex) {

int leftIndex = beginIndex;

int middleIndex = (beginIndex + endIndex) / 2;

int rightIndex = middleIndex + 1;

int mergeArrLenght = endIndex - beginIndex + 1;

int\* merge\_arr = new int[endIndex - beginIndex + 1];

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

if ((leftIndex <= middleIndex) &&

((rightIndex > endIndex) || (vect[leftIndex] < vect[rightIndex]))) {

merge\_arr[i] = vect[leftIndex++];

}

else {

merge\_arr[i] = vect[rightIndex++];

}

}

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

vect[i + beginIndex] = merge\_arr[i];

}

delete[] merge\_arr;

}

void Divide(vector<int>& vect, int beginIndex, int endIndex) {

if (beginIndex < endIndex) {

Divide(vect, beginIndex, (endIndex + beginIndex) / 2);

Divide(vect, (endIndex + beginIndex) / 2 + 1, endIndex);

Merge(vect, beginIndex, endIndex);

}

}

void MergeSort(vector<int>& vect) {

if (vect.size() > 1) {

Divide(vect, 0, vect.size() - 1);

}

}

CountingSort.h

#pragma once

#include <vector>

using namespace std;

void CountingSort(vector<int>& vect) {

if (vect.size() < 2) {

return;

}

int mx\_value = vect[0];

int mn\_value = vect[0];

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

if (mx\_value < vect[i]) {

mx\_value = vect[i];

}

else if (mn\_value > vect[i])

{

mn\_value = vect[i];

}

}

int countArrLenght = mx\_value - mn\_value + 1;

int\* countArr = new int[countArrLenght] {};

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

countArr[vect[i] - mn\_value]++;

}

int index = 0;

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

for (int j = 0; j < countArr[i]; ++j) {

vect[index++] = i + mn\_value;

}

}

delete[] countArr;

}

BucketSort.h

#pragma once

#include <vector>

#include "QuickSort.h"

using namespace std;

void BucketSort(vector<int>& vect) {

if (vect.size() < 2) {

return;

}

int mx\_value = vect[0];

int mn\_value = vect[0];

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

if (mx\_value < vect[i]) {

mx\_value = vect[i];

}

else if (mn\_value > vect[i])

{

mn\_value = vect[i];

}

}

int diff\_step = 10;

int bucketsCount = (int)((mx\_value - mn\_value)/diff\_step) + 1;

vector<int>\* buckets = new vector<int>[bucketsCount] {};

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

buckets[vect[i] / 10 - mn\_value / 10].push\_back(vect[i]);

}

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

QuickSort(buckets[i]);

}

int index = 0;

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

for (int j = 0; j < buckets[i].size(); ++j) {

vect[index++] = buckets[i][j];

}

}

delete[] buckets;

}

ShellSort.h

#pragma once

#include <vector>

using namespace std;

void ShellSort(vector<int>& vect) {

if (vect.size() < 2) {

return;

}

for (int step = vect.size() / 2; step > 0; step /= 2) {

for (int i = step; i < vect.size(); ++i) {

for (int j = i - step; (j >= 0) &&

(vect[j] > vect[j + step]); j -= step) {

swap(vect[j], vect[j + step]);

}

}

}

}

HoarSort.h

#pragma once

#include <vector>

using namespace std;

void HoarSort(vector<int>& vect, int beginIndex, int endIndex) {

int leftIndex = beginIndex;

int rightIndex = endIndex;

int pivot = vect[(beginIndex + endIndex) / 2];

do {

while ((vect[leftIndex] < pivot) && (leftIndex < endIndex)) {

++leftIndex;

}

while ((vect[rightIndex] > pivot) && (rightIndex > beginIndex)) {

--rightIndex;

}

if (leftIndex <= rightIndex) {

swap(vect[leftIndex++], vect[rightIndex--]);

}

} while (leftIndex <= rightIndex);

if (rightIndex > beginIndex) {

HoarSort(vect, beginIndex, rightIndex);

}

if (leftIndex < endIndex) {

HoarSort(vect, leftIndex, endIndex);

}

}

void HoarSort(vector<int>& vect) {

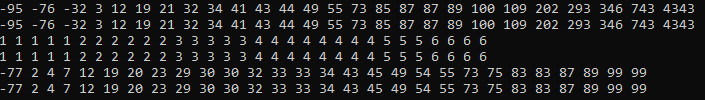
if (vect.size() > 1) {

HoarSort(vect, 0, vect.size() - 1);

}

}

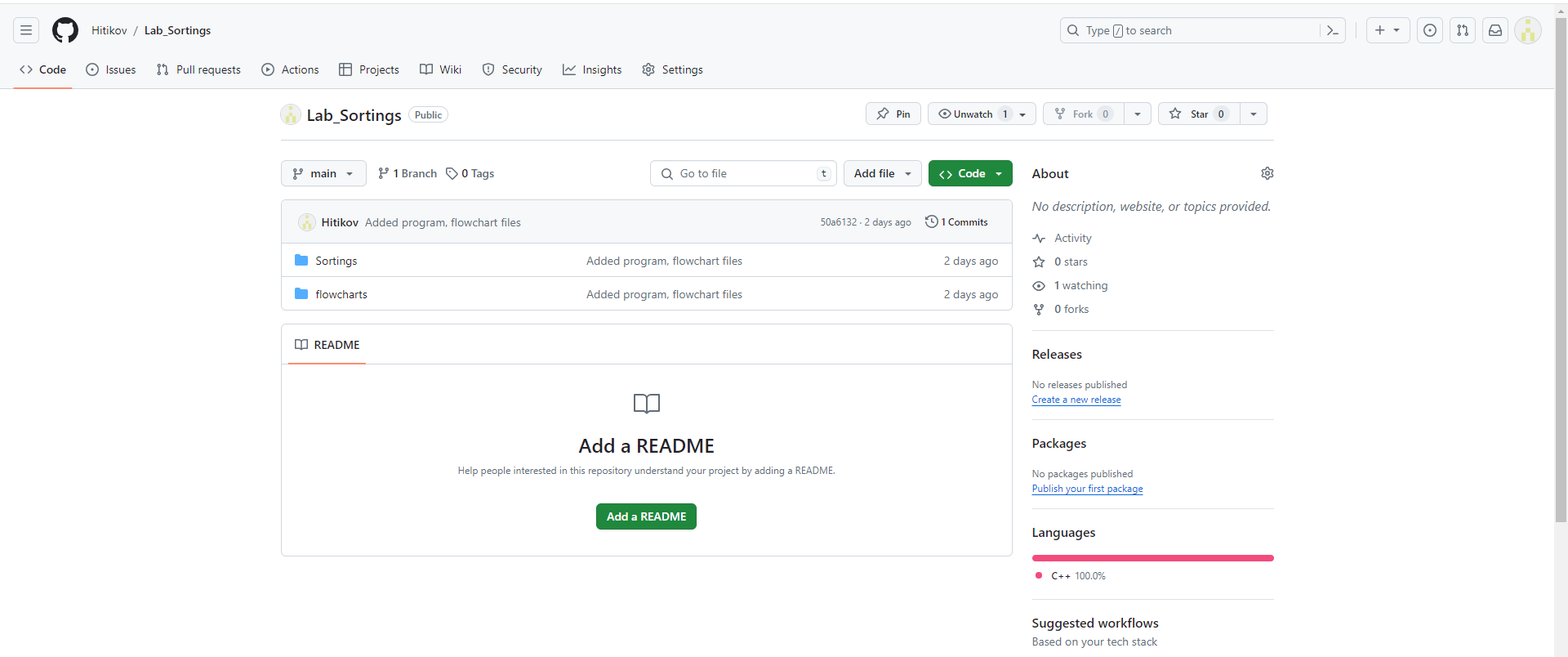
**Решение**



**Выводы**

В ходе данной работы были изучены простые сортировки и некоторые особенности работы с ними.

**Github**



<https://github.com/Hitikov/Lab_Sorting>