**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра САПР**

отчет

**по лабораторной работе №2**

**по дисциплине «Алгоритмы и структуры данных»**

## Вариант №1

Студент гр. 9309 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Корягин Е.А.

Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Тутуева А.В.

Санкт-Петербург

2020

## Постановка задачи

Написать программу с реализацией различных способов сортировки массивов. Необходимо реализовать наличие unit-тестов ко всем реализуемым методам. А также реализовать конструктор и деструктор.

## Описание реализуемого класса и метода

class Array (класс хранит в себе массив типа int)

class Char\_Array (класс хранит в себе массив типа char)

~Array() и ~Char\_Array() – деструкторы классов Array и Char\_Array

Фукнции:

void filling(int, int) – функция заполнения массива вручную (для UnitTest1).

int getValue(int) – функция получения элемента массива с заданным индексом.

void BogoSort() – функция глупой сортировки массива.

int cor() – функция для работы BogoSort.

void quicksort(int, int) – функция быстрой сортировки массива.

void BinarySearch(int) – функция двоичного поиска элемента в массиве по заданному индексу.

void InsertionSort() – функция сортировки вставками массива.

void randfilling() – функция заполнения массива случайными числами от 0 до 9.

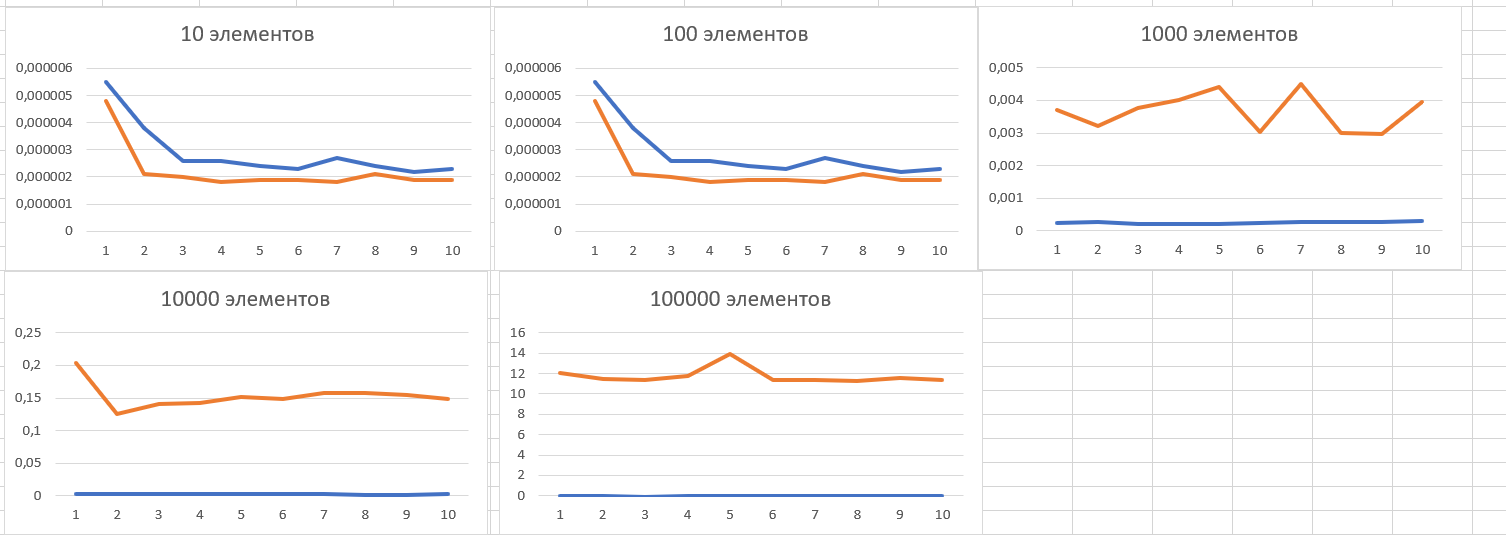
void CountingSort() – функция сортировки массива подсчетом для типа char.

## Оценка временной сложности методов

1. Filling = O(1).
2. getValue = O(1).
3. BogoSort = O(1).
4. Cor = O(N).
5. Quicksort = O(N^2).
6. BinarySearch = O(N^N).
7. InsertionSort = O(N^2).
8. Randfilling = O(1).
9. CountingSort = O(N^3).

**Сравнение временной сложности алгоритмов**

Синяя линия – quick sort, Оранжевая – insertion sort



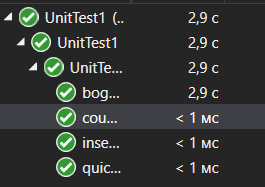
## Описание реализованных unit-тестов

**quick\_sort** – Производится заполнение 2х массивов, один из которых отсортирован, а второй сортируется функцией в дальнейшем. После сортировки массивы сравниваются.

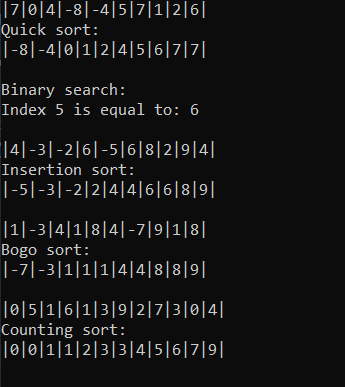
**insertion\_sort** – Производится заполнение 2х массивов, один из которых отсортирован, а второй сортируется функцией в дальнейшем. После сортировки массивы сравниваются.

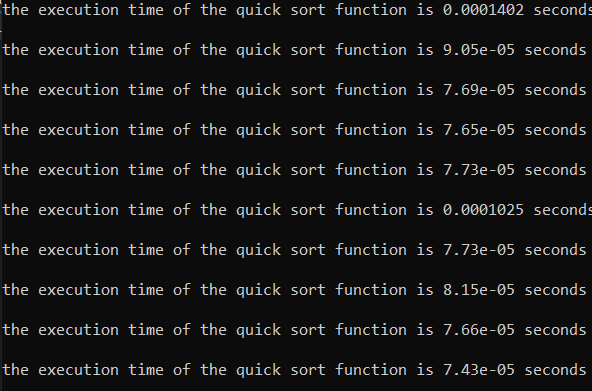
**bogo\_sort** – Производится заполнение 2х массивов, один из которых отсортирован, а второй сортируется функцией в дальнейшем. После сортировки массивы сравниваются.

**counting\_sort** – Производится заполнение 2х массивов, один из которых отсортирован, а второй сортируется функцией в дальнейшем. После сортировки массивы сравниваются.



## Пример работы





## Листинг

**main.cpp**

#include "class\_func.h"

#include <iostream>

#include <chrono>

#include <cstdlib>

#include <stdexcept>

using namespace std;

int main()

{

/\*for (int i = 0; i < 10; i++)

{

int length = 1000;

Array ar(length);

ar.randfilling();

chrono::system\_clock::time\_point start = chrono::system\_clock::now();

ar.quicksort(0, length-1);

chrono::system\_clock::time\_point end = chrono::system\_clock::now();

chrono::duration<double> sec = end - start;

cout << "the execution time of the quick sort function is " << sec.count() << " seconds" << endl << endl;

}\*/

Array ar(10);

ar.randfilling();

ar.print\_to\_console();

cout << "Quick sort:" << endl;

ar.quicksort(0, 9);

ar.print\_to\_console();

cout << endl;

cout << "Binary search:" << endl;

ar.BinarySearch(5);

cout << endl;

ar.randfilling();

ar.print\_to\_console();

cout << "Insertion sort:" << endl;

ar.InsertionSort();

ar.print\_to\_console();

cout << endl;

ar.randfilling();

ar.print\_to\_console();

cout << "Bogo sort:" << endl;

ar.BogoSort();

ar.print\_to\_console();

cout << endl;

Char\_Array ch\_ar(12);

ch\_ar.randfilling();

ch\_ar.print\_to\_console();

cout << "Counting sort:" << endl;

ch\_ar.CountingSort();

ch\_ar.print\_to\_console();

cout << endl;

}

**Class\_func.h**

#pragma once

#include <iostream>

#include <cstdlib>

#include <stdexcept>

using namespace std;

class Array

{

int\* array;

int length;

public:

Array(int ar\_len) // constructor

{

array = new int[ar\_len];

length = ar\_len;

}

~Array() // destructor

{

delete[] array;

}

void randfilling() //Array filling

{

int temp;

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

{

temp = rand() % 2;

if (temp == 0)

{

array[i] = (rand() % 10) \* (-1);

}

else

array[i] = rand() % 10;

}

}

int getValue(int ind) //Getting array`s value

{

return array[ind];

}

int cor() // Bogo sort`s part

{

int len = length;

while (--len > 0)

if (array[len - 1] > array[len])

return 0;

return 1;

}

void BogoSort() //Bogo sort

{

while (!cor())

{

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

swap(array[i], array[(rand() % length)]);

}

}

void quicksort(int start, int end) //QuickSort

{

int mid, counter;

int start\_2 = start, end\_2 = end;

mid = array[(start\_2 + end\_2) / 2];

do

{

while (array[start\_2] < mid) start\_2++;

while (array[end\_2] > mid) end\_2--;

if (start\_2 <= end\_2)

{

counter = array[start\_2];

array[start\_2] = array[end\_2];

array[end\_2] = counter;

start\_2++;

end\_2--;

}

} while (start\_2 < end\_2);

if (start < end\_2) quicksort(start, end\_2);

if (start\_2 < end) quicksort(start\_2, end);

}

void BinarySearch(int key) //BinarySearch

{

bool flag = false;

int lt = 0;

int rt = length - 1;

int mid;

while ((lt <= rt) && (flag != true))

{

mid = (lt + rt) / 2;

if (array[mid] == key) flag = true;

if (array[mid] > key) rt = mid - 1;

else lt = mid + 1;

}

if (flag)

{

cout << "Index " << key << " is equal to: " << mid << endl;

}

else

{

throw domain\_error("There is no such an element in this array!\n");

}

}

void InsertionSort() //InsertionSort

{

int flag = 0, temp = 0;

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

{

flag = i + 1;

temp = array[flag];

for (int j = i + 1; j > 0; j--)

{

if (temp < array[j - 1])

{

array[j] = array[j - 1];

flag = j - 1;

}

}

array[flag] = temp;

}

}

void print\_to\_console() //Printing the array

{

cout << "|";

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

{

cout << getValue(i) << "|";

}

cout << endl;

}

void filling(int ind, int elem) //Manually filling the array

{

array[ind] = elem;

}

};

class Char\_Array

{

char\* array;

int length;

public:

Char\_Array(int ar\_len) // constructor

{

array = new char[ar\_len];

length = ar\_len;

}

~Char\_Array() // destructor

{

delete[] array;

}

void filling(int ind, char elem) //Filling the char array

{

array[ind] = elem + '0';

}

void print\_to\_console() //Printing the char array

{

cout << "|";

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

{

cout << getValue(i) << "|";

}

cout << endl;

}

void CountingSort() //Counting sort

{

int\* output = new int[length];

int\* numbers = new int[length];

int max = 0;

int temp = 0;

int temp2 = 0;

int temp3 = 0;

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

{

temp = array[i] - '0';

if (temp > max)

{

max = temp;

}

}

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

{

numbers[i] = 0;

}

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

{

numbers[array[i] - '0']++;

}

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

{

numbers[i] += numbers[i - 1];

}

for (int i = length - 1; i >= 0; i--)

{

output[numbers[array[i] - '0'] - 1] = array[i] - '0';

numbers[array[i] - '0']--;

}

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

{

array[i] = output[i] + '0';

}

}

char getValue(int ind) //Getting the value of array

{

return array[ind];

}

void randfilling() // Random filling of the char array

{

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

{

array[i] = rand() % 10 + '0';

}

}

};

**UnitTest1.cpp**

#include "pch.h"

#include "CppUnitTest.h"

#include <stdexcept>

#include "../Lab2Aistrd\_Koryagin/class\_func.h"

#include "../Lab2Aistrd\_Koryagin/main.cpp"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace UnitTest1

{

TEST\_CLASS(UnitTest1)

{

public:

int length;

TEST\_METHOD\_INITIALIZE(setUp)

{

length = 10;

}

TEST\_METHOD\_CLEANUP(cleanUP)

{

length = 0;

}

TEST\_METHOD(quick\_sort)

{

Array test(length);

int fill\_test[10] = { 2,3,5,7,10,14,17,23,52,100 };

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

{

test.filling(i, fill\_test[i]);

}

Array arr(length);

int arr\_test[10] = { 7,3,10,5,2,17,14,100,23,52 };

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

{

arr.filling(i, arr\_test[i]);

}

arr.quicksort(0, 9);

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

{

Assert::AreEqual(arr.getValue(i), test.getValue(i));

}

}

TEST\_METHOD(insertion\_sort)

{

Array test(length);

int fill\_test[10] = { 2,3,5,7,10,14,17,23,52,100 };

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

{

test.filling(i, fill\_test[i]);

}

Array arr(length);

int arr\_test[10] = { 7,3,10,5,2,17,14,100,23,52 };

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

{

arr.filling(i, arr\_test[i]);

}

arr.InsertionSort();

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

{

Assert::AreEqual(arr.getValue(i), test.getValue(i));

}

}

TEST\_METHOD(bogo\_sort)

{

Array test(length);

int fill\_test[10] = { 2,3,5,7,10,14,17,23,52,100 };

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

{

test.filling(i, fill\_test[i]);

}

Array arr(length);

int arr\_test[10] = { 7,3,10,5,2,17,14,100,23,52 };

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

{

arr.filling(i, arr\_test[i]);

}

arr.BogoSort();

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

{

Assert::AreEqual(arr.getValue(i), test.getValue(i));

}

}

TEST\_METHOD(counting\_sort)

{

Char\_Array test(length);

int fill\_test[10] = { 1,2,3,4,5,6,7,8,9,10 };

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

{

test.filling(i, fill\_test[i] + '0');

}

Char\_Array arr(length);

int arr\_test[10] = { 2,1,3,4,5,8,9,7,6,10 };

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

{

arr.filling(i, arr\_test[i] + '0');

}

arr.CountingSort();

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

{

Assert::AreEqual(arr.getValue(i), test.getValue(i));

}

}

};

}