**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 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Юшин Е.В.

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

Санкт-Петербург

2021

## Цель работы

Оценка временной сложности алгоритмов.

## 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 алгоритмов:

void BogoSort() – функция глупой сортировки массива.

bool sorted() – функция для работы BogoSort.

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

size\_t BiniarySearch(int) – функция двоичного поиска элемента в массиве по заданному индексу.

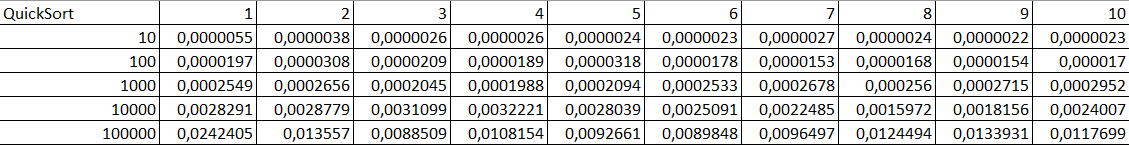
void InsertionSort() – функция сортировки вставками массива.

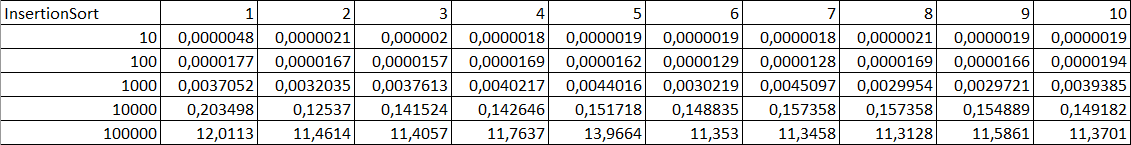
void CountingSort() – функция сортировки массива подсчетом для типа char.

## Оценка временной сложности:

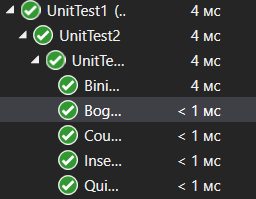
1. BogoSort = O(1).
2. sorted = O(N).
3. QuickSort = O(N^2).
4. BiniarySearch = O(N^N).
5. InsertionSort = O(N^2).
6. CountingSort = O(N^3).

**Сравнение временной сложности алгоритмов**





## Юнит-тесты:



## Текст юнит-теста:

#include "pch.h"

#include "CppUnitTest.h"

#include "../Lab2Aistrd\_Ushin/Func.h"

using namespace

Microsoft::VisualStudio::CppUnitTestFramework;

namespace UnitTest2

{

TEST\_CLASS(UnitTest2)

{

public:

TEST\_METHOD(CountingSort\_test)

{

char\_arr Ch(10);

Ch.CountingSort();

Assert::IsTrue(Ch.sorted());

Ch.CountingSort();

Assert::IsTrue(Ch.sorted());

}

TEST\_METHOD(QuickSort\_test)

{

Array Int(10);

Int.QuickSort();

Assert::IsTrue(Int.sorted());

Int.QuickSort();

Assert::IsTrue(Int.sorted());

}

TEST\_METHOD(InsertionSort\_test)

{

Array Int(10);

Int.InsertionSort();

Assert::IsTrue(Int.sorted());

Int.InsertionSort();

Assert::IsTrue(Int.sorted());

}

TEST\_METHOD(BogoSort\_test)

{

Array Int(5);

Int.BogoSort();

Assert::IsTrue(Int.sorted());

Int.BogoSort();

Assert::IsTrue(Int.sorted());

}

TEST\_METHOD(BiniarySearch\_test)

{

Array Int(10);

int elem = Int.getelem(4);

Int.InsertionSort();

size\_t index = Int.BiniarySearch(elem);

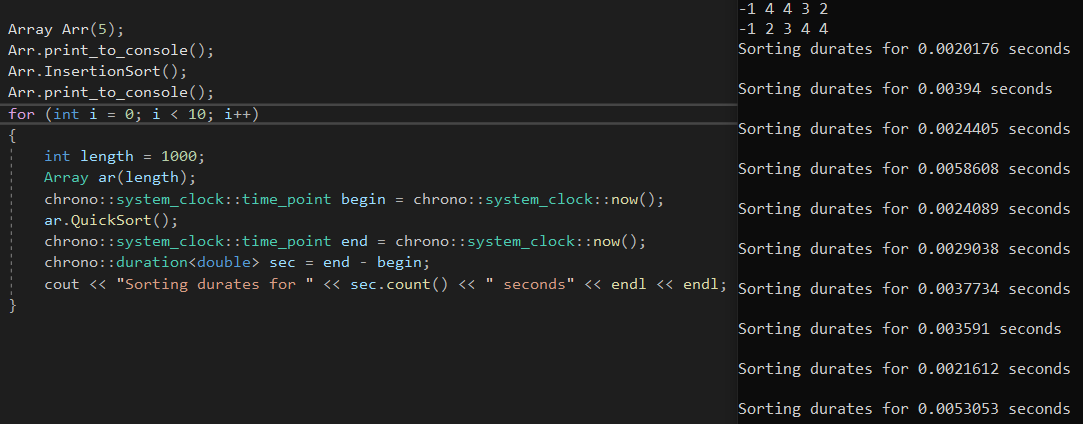
Assert::AreEqual(elem, Int.getelem(index));

}

};

}

**Результат работы программы:**



## Текст программы:

**main.cpp**  
#include "Func.h"

#include <iostream>

#include <stdexcept>

#include <chrono>

using namespace std;

int main()

{

/\*Array Arr(5);

Arr.print\_to\_console();

Arr.InsertionSort();

Arr.print\_to\_console();\*/

for (int i = 0; i < 10; i++)

{

int length = 1000;

Array ar(length);

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

ar.QuickSort();

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

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

cout << "Sorting durates for " << sec.count() << " seconds" << endl << endl;

}

}

**Func.h**

#pragma once

#include <iostream>

#include <stdexcept>

#include <chrono>

using namespace std;

class char\_arr

{

char\* char\_ar;

size\_t sizee;

public:

char\_arr(size\_t size)

{

if (size == 0)

throw out\_of\_range("Incorrect size!\n");

char\_ar = new char[size];

sizee = size;

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

{

char\_ar[i] = rand() % sizee;

char\_ar[i] \*= pow(-1, rand() % 2);

}

}

bool sorted()

{

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

{

if (char\_ar[i] > char\_ar[i + 1])

return false;

}

return true;

}

void CountingSort()

{

int i;

int max = char\_ar[0];

int min = char\_ar[0];

int\* tmp\_arr = nullptr;

int\* tmp\_arr2 = nullptr;

for (i = 1; i < sizee; i++)

{

if (char\_ar[i] > max)

max = char\_ar[i];

else if (char\_ar[i] < min)

min = char\_ar[i];

}

if (max >= 0)

{

tmp\_arr = new int[max + 1];

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

tmp\_arr[i] = 0;

}

if (min < 0)

{

tmp\_arr2 = new int[min \* (-1) + 1];

for (i = 0; i <= (min \* (-1)); i++)

tmp\_arr2[i] = 0;

}

for (i = 0; i < sizee; i++)

{

if (char\_ar[i] < 0)

tmp\_arr2[char\_ar[i] \* (-1)]++;

else

tmp\_arr[char\_ar[i]]++;

}

int j = 0;

if (min < 0)

{

for (i = (min \* (-1)); i > 0; i--)

{

while (tmp\_arr2[i] != 0)

{

char\_ar[j] = i \* (-1);

j++;

tmp\_arr2[i]--;

}

}

delete[] tmp\_arr2;

}

if (max >= 0)

{

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

{

while (tmp\_arr[i] != 0)

{

char\_ar[j] = i;

j++;

tmp\_arr[i]--;

}

}

delete[] tmp\_arr;

}

}

void print\_to\_console()

{

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

{

cout << char\_ar[i] << ' ';

}

cout << endl;

}

~char\_arr()

{

delete[] char\_ar;

}

};

class Array

{

int\* int\_ar;

size\_t sizee;

void QuickSort(int first, int last)

{

int middle = last;

int temp, i, j;

for (i = first; i < middle; i++)

{

if (int\_ar[i] > int\_ar[middle])

{

for (j = i; j < middle; j++)

{

temp = int\_ar[j];

int\_ar[j] = int\_ar[j + 1];

int\_ar[j + 1] = temp;

}

middle--;

i--;

}

}

if (first < (middle - 1))

QuickSort(first, middle - 1);

if ((middle + 1) < last)

QuickSort(middle + 1, last);

}

int BinarySearch(int left, int right, int elem)

{

if (left > right)

return -1;

size\_t mid = ((right - left) / 2) + left;

size\_t answerr;

if (int\_ar[mid] == elem)

answerr = mid;

else if (int\_ar[mid] < elem)

answerr = BinarySearch(mid + 1, right, elem);

else

answerr = BinarySearch(left, mid - 1, elem);

return answerr;

}

public:

Array(size\_t size)

{

if (size == 0)

throw out\_of\_range("Incorrect size!\n");

int\_ar = new int[size];

sizee = size;

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

{

int\_ar[i] = rand() % sizee;

int\_ar[i] \*= pow(-1, rand() % 2);

}

}

int getelem(size\_t index)

{

if (index >= sizee)

throw out\_of\_range("Index is bigger than size of array!\n");

return int\_ar[index];

}

bool sorted()

{

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

{

if (int\_ar[i] > int\_ar[i + 1])

return false;

}

return true;

}

size\_t BiniarySearch(int search)

{

if (!sorted())

throw runtime\_error("Array must be sorted!\n");

if (search<int\_ar[0] || search>int\_ar[sizee - 1])

throw runtime\_error("There is no element!\n");

int index = BinarySearch(0, sizee - 1, search);

if (index < 0)

throw runtime\_error("There is no element!\n");

return index;

}

void QuickSort()

{

QuickSort(0, sizee - 1);

}

void InsertionSort()

{

int key;

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

{

key = int\_ar[i + 1];

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

{

if (key < int\_ar[j])

{

int\_ar[j + 1] = int\_ar[j];

int\_ar[j] = key;

}

else

{

break;

}

}

}

}

void BogoSort()

{

int temp;

int index;

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

{

temp = int\_ar[i];

index = rand() % sizee;

int\_ar[i] = int\_ar[index];

int\_ar[index] = temp;

}

if (!sorted())

{

BogoSort();

}

}

void print\_to\_console()

{

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

{

cout << int\_ar[i] << ' ';

}

cout << endl;

}

~Array()

{

delete[] int\_ar;

}

};