**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра CАПР**

отчет

**по лабораторной работе №2**

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

**Вариант 2**

**Тема: Алгоритмы сортировки и поиска**

|  |  |  |
| --- | --- | --- |
| Студент гр. 9302 |  | Бабкин И. А. |
| Преподаватель |  | Тутуева.А.В |

Санкт-Петербург

2020

**Постановка задачи**

Необходимо написать алгоритмы сортировки и поиска для целочисленного типа данный int, если не указано иное. Сортируются массивы.

**Описание реализуемых алгоритмов**

|  |  |
| --- | --- |
| Название метода | Описание |
| int BinarySearch(int\* array, int left, int right, int key) | Бинарный поиск |
| void QuickSort(int\* array, int low, int high) | Быстрая сортировка |
| void BubbleSort(int\* array, int n) | Сортировка пузырьком |
| void BogoSort(int\* array, int n) | Болотная сортировка |
| void CountingSort(int\* array, int n) | Сортировка подсчетом |

**Оценка временной сложности каждого алгоритма**

|  |  |
| --- | --- |
| Название метода | Оценка временной сложности |
| BinarySearch | O(log2(n)) |
| QuickSort | O(n\*log(n)) |
| BubbleSort | O(n^2) |
| BogoSort | O(n\*n!) |
| CountingSort | O(n^2) |

**Сравнение временной сложности алгоритмов QuickSort и BubbleSort**

QuickSort(10) QuickSort(100) QuickSort(1000) QuickSort(10000)

|  |  |
| --- | --- |
| 1 | 0 |
| 2 | 0 |
| 3 | 0 |
| 4 | 0 |
| 5 | 0 |
| 6 | 0 |
| 7 | 0 |
| 8 | 0 |
| 9 | 0 |
| 10 | 0 |
| Среднее | 0 |

|  |  |
| --- | --- |
| 1 | 0 |
| 2 | 0 |
| 3 | 0 |
| 4 | 0 |
| 5 | 0 |
| 6 | 0 |
| 7 | 0 |
| 8 | 0 |
| 9 | 0 |
| 10 | 0 |
| Среднее | 0 |

|  |  |
| --- | --- |
| 1 | 0.001 |
| 2 | 0 |
| 3 | 0 |
| 4 | 0 |
| 5 | 0.001 |
| 6 | 0 |
| 7 | 0 |
| 8 | 0 |
| 9 | 0 |
| 10 | 0 |
| Среднее | 0.0002 |

|  |  |
| --- | --- |
| 1 | 0.003 |
| 2 | 0.002 |
| 3 | 0.002 |
| 4 | 0.002 |
| 5 | 0.001 |
| 6 | 0.002 |
| 7 | 0.001 |
| 8 | 0.002 |
| 9 | 0.001 |
| 10 | 0.002 |
| Среднее | 0.0018 |

QuickSort(100000)

|  |  |
| --- | --- |
| 1 | 0.018 |
| 2 | 0.017 |
| 3 | 0.018 |
| 4 | 0.017 |
| 5 | 0.017 |
| 6 | 0.018 |
| 7 | 0.02 |
| 8 | 0.019 |
| 9 | 0.02 |
| 10 | 0.018 |
| Среднее | 0.0182 |

BubbleSort(10) BubbleSort(100) BubbleSort(1000)

|  |  |
| --- | --- |
| 1 | 0 |
| 2 | 0 |
| 3 | 0 |
| 4 | 0 |
| 5 | 0 |
| 6 | 0 |
| 7 | 0 |
| 8 | 0 |
| 9 | 0 |
| 10 | 0 |
| Среднее | 0 |

|  |  |
| --- | --- |
| 1 | 0 |
| 2 | 0 |
| 3 | 0 |
| 4 | 0 |
| 5 | 0 |
| 6 | 0 |
| 7 | 0 |
| 8 | 0 |
| 9 | 0 |
| 10 | 0 |
| Среднее | 0 |

|  |  |
| --- | --- |
| 1 | 0.003 |
| 2 | 0.001 |
| 3 | 0.001 |
| 4 | 0.001 |
| 5 | 0.001 |
| 6 | 0.001 |
| 7 | 0.001 |
| 8 | 0.001 |
| 9 | 0.001 |
| 10 | 0.001 |
| Среднее | 0.0012 |

BubbleSort(10000) BubbleSort(100000)

|  |  |
| --- | --- |
| 1 | 0.199 |
| 2 | 0.21 |
| 3 | 0.213 |
| 4 | 0.212 |
| 5 | 0.201 |
| 6 | 0.21 |
| 7 | 0.245 |
| 8 | 0.232 |
| 9 | 0.2 |
| 10 | 0.197 |
| Среднее | 0.2119 |

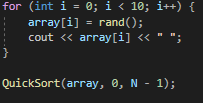
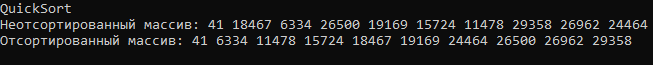
|  |  |
| --- | --- |
| 1 | 23.983 |
| 2 | 23.941 |
| 3 | 23.929 |
| 4 | 23.958 |
| 5 | 23.77 |
| 6 | 23.93 |
| 7 | 24.094 |
| 8 | 24.169 |
| 9 | 23.754 |
| 10 | 23.997 |
| Среднее | 23.9525 |

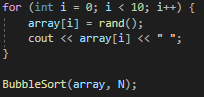
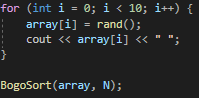
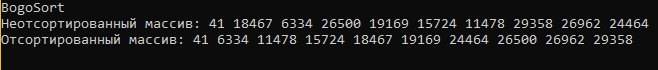
**Описание реализованных unit-тестов**

Для проверки реализованных методов были написаны unit-тесты. В таблице ниже представлены их названия. Названия совпадают с названиями проверяемых методов.

|  |
| --- |
| Название теста |
| Binary\_Search1,2,3 |
| Quick\_Sort1,2,3 |
| Bubble\_Sort1,2,3 |
| Bogo\_Sort1,2,3 |
| Counting\_Sort1,2,3 |

**Пример работы**

**Листинг**

Sorts.h

#pragma once

int BinarySearch(int\* array, int left, int right, int key);

void QuickSort(int\* array, int low, int high);

void InsertionSort(int\* array, int n);

void BubbleSort(int\* array, int n);

void BogoSort(int\* array, int n);

void CountingSort(int\* array, int n);

Sorts.cpp

#include <iostream>

#include "Sorts.h"

using namespace std;

int BinarySearch(int\* array, int left, int right, int key)

{

int midd = 0;

while (2 + 2 == 4) {

midd = (left + right) / 2;

if (key < array[midd])

right = midd - 1;

else if (key > array[midd])

left = midd + 1;

else

return midd;

if (left > right)

return -1;

}

}

void QuickSort(int\* array, int low, int high)

{

int i = low;

int j = high;

int pivot = array[(i + j) / 2];

int temp;

while (i <= j) {

while (array[i] < pivot)

i++;

while (array[j] > pivot)

j--;

if (i <= j)

{

temp = array[i];

array[i] = array[j];

array[j] = temp;

i++;

j--;

}

}

if (j > low)

QuickSort(array, low, j);

if (i < high)

QuickSort(array, i, high);

}

void InsertionSort(int\* array, int n)

{

int key;

int j;

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

key = array[i];

j = i - 1;

while (j >= 0 && array[j] > key) {

array[j + 1] = array[j];

j--;

}

array[j + 1] = key;

}

}

void BubbleSort(int\* array, int n)

{

int temp;

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

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

if (array[j] > array[j + 1]) {

temp = array[j];

array[j] = array[j + 1];

array[j + 1] = temp;

}

}

}

}

void BogoSort(int\* array, int n)

{

bool sorted = false;

int index;

int temp;

while (sorted == false) {

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

index = rand() % n;

temp = array[i];

array[i] = array[index];

array[index] = temp;

}

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

sorted = true;

if (array[i] > array[i + 1]) {

sorted = false;

break;

}

}

}

}

void CountingSort(int\* array, int n)

{

int c[128] = { 0 };

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

c[array[i]] = c[array[i]] + 1;

int b = 0;

for (int i = 0; i < 128; i++) {

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

array[b] = i;

b = b + 1;

}

}

}