**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 1**

**Тема: Алгоритмы сортировки и поиска**

|  |  |  |
| --- | --- | --- |
| Студент гр. 9302 |  | Квитко Д.В. |
| 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 |
| void quickSort(int\* array, int low, int high) | Быстрая сортировка |
| void insertSort(int\* arr, int n) | Сортировка вставками |
| void countingSort(int\* array, int n) | Сортировка подсчетом |
| void bogosort(int\* array, int n) | Болотная сортировка |
| int binarysearch(int\* array, int n, int numb) | Бинарный поиск |

**Оценка временной сложности каждого алгоритма**

|  |  |
| --- | --- |
| Название метода | Оценка временной сложности |
| BinarySearch | O(log2(n)) |
| QuickSort | O(n\*log(n)) |
| InsertionSort | 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 |
| 2 | 0 |
| 3 | 0 |
| 4 | 0 |
| 5 | 0 |
| 6 | 0 |
| 7 | 0 |
| 8 | 0 |
| 9 | 0 |
| 10 | 0 |
| Среднее | 0 |

|  |  |
| --- | --- |
| 1 | 0.001 |
| 2 | 0.001 |
| 3 | 0.001 |
| 4 | 0.001 |
| 5 | 0.004 |
| 6 | 0.002 |
| 7 | 0.002 |
| 8 | 0.002 |
| 9 | 0.001 |
| 10 | 0.001 |
| Среднее | 0.00161 |

QuickSort(100000)

|  |  |
| --- | --- |
| 1 | 0.015 |
| 2 | 0.011 |
| 3 | 0.013 |
| 4 | 0.015 |
| 5 | 0.016 |
| 6 | 0.015 |
| 7 | 0.014 |
| 8 | 0.014 |
| 9 | 0.013 |
| 10 | 0.016 |
| Среднее | 0.0142 |

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 | 0.001 |
| 2 | 0.001 |
| 3 | 0.001 |
| 4 | 0.002 |
| 5 | 0.0021 |
| 6 | 0.001 |
| 7 | 0.001 |
| 8 | 0.001 |
| 9 | 0.001 |
| 10 | 0.001 |
| Среднее | 0.0012 |

BubbleSort(10000) BubbleSort(100000)

|  |  |
| --- | --- |
| 1 | 0.091 |
| 2 | 0.094 |
| 3 | 0.1 |
| 4 | 0.095 |
| 5 | 0.091 |
| 6 | 0.1 |
| 7 | 0.089 |
| 8 | 0.099 |
| 9 | 0.087 |
| 10 | 0.087 |
| Среднее | 0.0933 |

|  |  |
| --- | --- |
| 1 | 9.058 |
| 2 | 9.592 |
| 3 | 10.330 |
| 4 | 10.035 |
| 5 | 9.467 |
| 6 | 9.048 |
| 7 | 10.93 |
| 8 | 9.021 |
| 9 | 9.073 |
| 10 | 8.933 |
| Среднее | 8.565 |

**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 |
| binaryTest\_1,2,3 |
| quickTest\_1,2 |
| insertTest\_1,2 |
| countinTest\_1,2 |
| bogoTest\_1,2 |

**Листинг**

**functions.h**

using namespace std;

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

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

void insertSort(int\* arr, int n);

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

int binarysearch(int\* array, int n, int numb);

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

**Sorting.cpp**

#include <iostream>

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

{

int i = low;

int j = high - 1;

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 insertSort(int\* arr, int size) {

int iter, j;

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

iter = arr[i];

j = i - 1;

while (j >= 0 && arr[j] > iter) {

arr[j + 1] = arr[j];

j--;

}

arr[j + 1] = iter;

}

}

void countingSort(char\* array, int size) {

int arraycounting[256];

for (int i = 0; i < 256; i++) {

arraycounting[i] = 0;

}

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

arraycounting[array[i]]++;

}

int index = 0;

for (int i = 0; i < 256; i++) {

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

array[index] = i;

index++;

}

}

}

bool check(int\* array, int size) {

bool flag;

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

if (array[i - 1] <= array[i]) {

flag = true;

}

else flag = false;

if (flag == false) return false;

}

return true;

}

void bogosort(int\* array, int size) {

int temp, index;

while (!check(array, size))

{

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

index = rand() % size;

temp = array[i];

array[i] = array[index];

array[index] = temp;

}

}

}

**Search.cpp**

#include <iostream>

void printArray(int\* array, int size)

{

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

std::cout << array[i] << "(" << i << ")" << " ";

}

std::cout <<std::endl;

}

int binarysearch(int\* array, int size, int numb) {

bool flag = false;

int low = 0;

int high = size - 1;

int mid;

while ((low <= high) && (flag != true)) {

mid = (low + high) / 2;

if (array[mid] == numb) return mid;

if (array[mid] > numb) high = mid - 1;

else low = mid + 1;

}

return -1;

}

**Пример работы программы**

**Сортировка посчетом**

Неотсортированный массив



Отсортированный массив



**Быстрая сортировка**

Неотсортированный массив



Отсортированный массив



**Сортировка вставками**

Неотсортированный массив



Отсортированный массив



**Глупая сортировка**

Неотсортированный массив



Отсортированный массив

