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»

Тема: Алгоритмы сортировки и поиска на примере языка C++

Вариант 1

Студент 0301 _____ Сморжок В. Е.

Преподаватель _____ Тутуева А.В.

Санкт-Петербург,
2021

Задание лабораторной работы

Алгоритмы сортировки и поиска:

1. Двоичный поиск (BinarySearch)

2. Быстрая сортировка (QuickSort)

3. Сортировка вставками (InsertionSort)

4. Глупая сортировка (BogoSort)

5. Сортировка подсчётом (CountingSort) для типа char

Текст программы:

```
#include <iostream>
#include <stdio.h>
#include <chrono>
#include <thread>

using namespace std;


int BinarySearch(int array[], int find)
{

	int left = 0;
	int right = sizeof(int);
	int middle = 0; // middle of array. index
	bool answer = false;
	while (array[middle] != find)
	{
		if (array[middle] > find)
			right = middle;
		else
			left = middle;
		middle = (right + left) / 2;
		answer = true;
	}
	if (answer)
		cout << "Element has index = " << middle;
	else cout << "Incorrect data" << endl;
	return middle;
}

void QuickSort(int array[], int unconstnum)
{
	auto start = chrono::high_resolution_clock::now();
	int index_of_main = 0;
	bool answer = true;
	while (answer == true)
	{
		for (int i = index_of_main + 1; i < unconstnum; i++)
		{
			int remember = array[i];
			if (array[i] <= array[index_of_main])
			{
				for (int j = i; j < unconstnum - 1; j++)
				{
					array[j] = array[j + 1];
				}
				for (int j = unconstnum - 1; j > index_of_main; j--)
```

```
                        {
                                array[j] = array[j - 1];
                        }
                        array[index_of_main] = remember;
                        index_of_main++;
                }
        }
        answer = false;
        for (int i = 0; i < unconstnum - 1; i++)
        {
                for (int j = i; j < unconstnum; j++)
                {
                        if (array[i] > array[j])
                        {
                                answer = true;
                                index_of_main = i;
                        }
                }
        }
}

        this_thread::get_id();
        auto end = chrono::high_resolution_clock::now();
        chrono::duration<float> duration = end - start;
        cout << unconstnum << " elements " << duration.count() << " sec" << endl;

}

void InsertionSort(int array1[], int unconstnum)
{
        auto start = chrono::high_resolution_clock::now();
        int i = 0; int j;
        for (i = 1; i < unconstnum; i++)
                for (j = i; j > 0; j--) // пока j>0 и элемент j-1 > j, x-массив int
                        if (array1[j - 1] > array1[j])
                                swap(array1[j - 1], array1[j]);
                        else break;
        this_thread::get_id();
        auto end = chrono::high_resolution_clock::now();
        chrono::duration<float> duration = end - start;
        cout << unconstnum << " elements " << duration.count() << " sec" << endl;
}

void BogoSort(int array[], int n)
{
        bool answer = true;
        while (answer == true)
        {
                for (int i = 0; i < n; i++)
                {
                        swap(array[i], array[(rand() % n)]);
                }
                answer = false;
                for (int i = 0; i < n - 1; i++)
                {
                        if (array[i] > array[i + 1])
                        {
                                answer = true;
                        }
                }
        }
        for (int i = 0; i < n; i++)
        {
                cout << array[i] << " ";
        }
```

```
}

void CountingSort(char arrchar[],const int charconst)
{
	int* arrcount;
	arrcount = (int*)malloc(charconst * sizeof(int));
	int end = charconst - 1;
	int i; int j;
	for (i = 0; i < charconst; i++)
	{
		arrcount[i] = 1;
	}
	for (i = 0; i < charconst; i++)
	{
		for (j = i + 1; j < charconst; j++)
		{
			if (arrchar[i] == arrchar[j])
			{
				arrcount[i] ++;
				int k = j;
				while (k != charconst - 1)
				{
					arrchar[k] = arrchar[k + 1];
					k++;
				}
				arrchar[end] = 0;
				arrcount[end] = 0;
				end--;
			}
		}
	}
	int count = 0;
	for (i = 0; i < charconst; i++)
	{
		if (arrcount[i] != 0)
		{
			count++;
		}
	}
	for (i = 0; i <= end; i++)
		for (j = i; j > 0; j--) // пока j>0 и элемент j-1 > j, х-массив int
			if (arrchar[j - 1] > arrchar[j])
			{
				swap(arrchar[j - 1], arrchar[j]);
				swap(arrcount[j - 1], arrcount[j]);
			}
			else break;

	char* newarr;
	newarr = (char*)malloc(charconst * sizeof(char));
	j = 0;
	i = 0;
	while (j < count)
	{
		while (arrcount[j] != 0)
		{
			newarr[i] = arrchar[j];
			i++;
			arrcount[j]--;
		}
		j++;
	}
	for (i = 0; i < charconst; i++)
	{
		arrchar[i] = newarr[i];
```

```
	}
	for (int i = 0; i < charconst; i++)
	{
		cout << arrchar[i] << " ";
	}
	cout << endl;
}



int main()
{
	setlocale(LC_ALL, "Ru");
	int n = 6;
	int* a;  // указатель на массив
	// Выделение памяти
	a = (int*)malloc(n * sizeof(int));
	// Ввод элементов массива
	for (int i = 0; i < n; i++)
	{
		a[i] = rand() % 30;
	}
	QuickSort(a, n);
	cout << "BinarySearch\n Array: " << endl;
	for (int i = 0; i < n; i++)
	{
		cout << a[i] << " ";
	}
	int find;
	cout << endl;
	cout << "Enter element of array you want to get" << endl;
	cin >> find;
	BinarySearch(a, find);
	cout << endl;
	cout << "_____________" << endl << endl;
	const int num = 100000;
	int unconstnum = 10;
	for (int i = 0; i < unconstnum; i++)
	{
		a[i] = rand() % 30;
	}
	cout << "QuickSort\nArray: " << endl;
	for (int i = 0; i < unconstnum; i++)
	{
		cout << a[i] << " ";
	}
	cout << endl;
	QuickSort(a, unconstnum);
	for (int i = 0; i < unconstnum; i++)
	{
		cout << a[i] << " ";
	}
	cout << endl;
	for (int i = 0; i < n; i++)
	{
		a[i] = rand() % 30;
	}
	cout << endl;
	cout << "_____________" << endl << endl;
	cout << "InsertionSort\nArray: " << endl;
	for (int i = 0; i < n; i++)
	{
		cout << a[i] << " ";
	}
	cout << endl;
```

```
InsertionSort(a, unconstnum);
for (int i = 0; i < n; i++)
{
	cout << a[i] << " ";
}cout << endl;
cout << "_____________" << endl << endl;
for (int i = 0; i < n; i++)
{
	a[i] = rand() % 30;
}
cout << endl;

cout << "BogoSort\nArray: " << endl;
for (int i = 0; i < n; i++)
{
	cout << a[i] << " ";
}
cout << endl;
BogoSort(a, n); cout << endl;
cout << "_____________" << endl << endl;
const int charconst = 6;
char arraychar[charconst] = { 'c','1','d','b','c','a' };
cout << endl;
cout << "CountingSort\nArray: " << endl;
for (int i = 0; i < n; i++)
{
	cout << arraychar[i] << " ";
}
cout << endl;
CountingSort(arraychar, charconst);


cout << endl;

cout << "Running time of algorithms " << endl;

cout << "QuickSort" << endl;


unconstnum = 10;
while (unconstnum != num)
{
	// Выделение памяти
	a = (int*)malloc(unconstnum * sizeof(int));
	// Ввод элементов массива
	for (int i = 0; i < unconstnum; i++)
	{
		a[i] = rand() % 30;
	}
	QuickSort(a,unconstnum);
	unconstnum *= 10;
}
cout << "____________________" << endl;
cout << endl << "InsertionSort" << endl;
unconstnum = 10;
while (unconstnum != num)
{
	// Выделение памяти
	a = (int*)malloc(unconstnum * sizeof(int));
	// Ввод элементов массива
	for (int i = 0; i < unconstnum; i++)
	{
		a[i] = rand() % 30;
	}
	InsertionSort(a, unconstnum);
```

```
            unconstnum *= 10;
        }


        return 0;
}
```

Описание реализуемых алгоритмов:

1. Двоичный поиск (BinarySearch)

   На вход подается отсортированный массив по возрастанию, пользователю на экран показывается весь список, из которого он должен выбрать желаемое число. Определяются левая и правая границы массива, ищется середина, которая впоследствии сравнивается с искомым элементом, в зависимости от чего границы сдвигаются. Это происходит рекурсивно до того момента, пока средний элемент не будет равен искомому значению. На экран покажется его индекс

2. Быстрая сортировка (QuickSort)

   На вход подается любой массив типа int, за главный элемент принимается нулевой. Относительно него происходит сравнение каждого элемента. Если элементы меньше, то они ставятся слева, иначе – справа. Это происходит до тех пор, пока массив не выстроится в порядке по возрастанию.

3. Сортировка вставками (InsertionSort)

   На вход подается любой массив типа int, элементы входной последовательности просматриваются по одному, и каждый новый поступивший элемент размещается в подходящее место среди ранее упорядоченных элементов.

4. Глупая сортировка (BogoSort)

   На вход подается любой массив типа int, элементы меняются в рандомном порядке до тех пор, пока не выстроятся в ряд по возрастанию

5. Сортировка подсчётом (CountingSort) для типа char

   На вход подается любой массив типа char, массив просматривается на наличие одинаковых элементов, их количество запоминается. Затем массив упорядочивается в порядке по возрастанию, а затем в него добавляются элементы, количество которых было больше одного.

Оценка временной сложности каждого метода

1. Двоичный поиск (BinarySearch) – $O(\log_2 n)$

2. Быстрая сортировка (QuickSort) – $O(n*\log n)$

3. Сортировка вставками (InsertionSort) – $O(n^2)$

4. Глупая сортировка (BogoSort) – O(n*n!)

5. Сортировка подсчётом (CountingSort) для типа char – O(n)

Текст Unit тестов:

```
#include "pch.h"
#include "CppUnitTest.h"
#include "../2 лабораторная работа/mas.h"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace UnitTestLab2
{
	TEST_CLASS(UnitTestLab2)
	{
	public:

		TEST_METHOD(Method_BinarySearch)
		{
			const int n = 6;
			int mas[n] = { 1,2,5,18,20,91 };
			int find = 5;
			int check;
			for (int i = 0; i < n; i++)
			{
				find = mas[i];
				check = BinarySearch(mas, find, n);
				Assert::IsTrue(check == i);
			}
		}
	TEST_METHOD(Method_QuickSearch)
		{
			const int n = 6;
			int mas[n] = { 5,7,0,9,21,2 };
			QuickSearch(mas, n);
			bool answer;
			for (int i = 0; i < n-1; i++)
			{
				if (mas[i] < mas[i + 1])
					answer = true;
				else answer = false;
			}
			Assert::IsTrue(answer == true);
		}
		TEST_METHOD(Method_InsertionSort)
		{
			const int n = 6;
			int mas[n] = { 5,7,0,9,21,2 };
			InsertionSort(mas, n);
			bool answer;
			for (int i = 0; i < n-1; i++)
			{
				if (mas[i] < mas[i + 1])
					answer = true;
				else answer = false;
			}
			Assert::IsTrue(answer == true);
		}
		TEST_METHOD(Method_BogoSort)
		{
			const int n = 6;
			int mas[n] = { 5,7,0,9,21,2 };
			BogoSort(mas, n);
```

```
			bool answer;
			for (int i = 0; i < n-1; i++)
			{
				if (mas[i] < mas[i + 1])
					answer = true;
				else answer = false;
			}
			Assert::IsTrue(answer == true);
		}
		TEST_METHOD(Method_CountingSort)
		{
			const int n = 6;
			char mas[n] = { 'c','d','e','v','g','v' };
			CountingSort(mas, n);
			bool answer;
			for (int i = 0; i < n-1; i++)
			{
				if (mas[i] <= mas[i + 1])
					answer = true;
				else answer = false;
			}
			Assert::IsTrue(answer == true);
		}
	};
}
```

Описание реализуемых Unit тестов

1. BinarySearch – вводится желаемое число, запускается функция. Если результат функции равен введенному числу, то тест пройден. Проверка производится всех элементов
2. QuickSort – имеется массив, вызывается функция. Если все элементы массива стали упорядоченными по возрастанию, то тест пройден
3. InsertionSort – имеется массив, вызывается функция. Если все элементы массива стали упорядоченными по возрастанию, то тест пройден
4. BogoSort – имеется массив, вызывается функция. Если все элементы массива стали упорядоченными по возрастанию, то тест пройден
5. CountingSort - имеется массив, вызывается функция. Если все элементы массива стали упорядоченными по возрастанию, то тест пройден

Пример работы программы:

```
BinarySearch
 Array:
2 4 5 8 10 23
Enter element of array you want to get
5
Element has index = 2
_____________

QuickSearch
Array:
11 17 4 10 29 4 18 18 22 14
10 elements 9.2e-06 sec
4 4 10 11 14 17 18 18 22 29


_____________

InsertionSort
Array:
5 5 1 27 1 11
10 elements 2.1e-06 sec


_____________

BogoSort
Array:
25 2 27 6 21 24
2 6 21 24 25 27
_____________


CountingSort
Array:
c 1 d b c a
1 a b c c d
```

```
TEST_METHOD(Method_BinarySearch)
{
    const int n = 6;
    int mas[n] = { 1,2,5,18,20,91 };
    int find = 5;
    int check;
    for (int i = 0; i < n; i++)
    {
        find = mas[i];
        check = BinarySearch(mas, find, n);
        Assert::IsTrue(check == i);
    }
}
TEST_METHOD(Method_QuickSearch)
{
    const int n = 6;
    int mas[n] = { 5,7,0,9,21,2 };
    QuickSearch(mas, n);
    bool answer;
    for (int i = 0; i < n-1; i++)
    {
        if (mas[i] < mas[i + 1])
            answer = true;
        else answer = false;
    }
    Assert::IsTrue(answer == true);
}
TEST_METHOD(Method_InsertionSort)
{
    const int n = 6;
    int mas[n] = { 5,7,0,9,21,2 };
    InsertionSort(mas, n);
    bool answer;
    for (int i = 0; i < n-1; i++)
    {
        if (mas[i] < mas[i + 1])
            answer = true;
        else answer = false;
    }
    Assert::IsTrue(answer == true);
}

TEST_METHOD(Method_BogoSort)
{
    const int n = 6;
    int mas[n] = { 5,7,0,9,21,2 };
    BogoSort(mas, n);
    bool answer;
    for (int i = 0; i < n-1; i++)
    {
        if (mas[i] < mas[i + 1])
            answer = true;
        else answer = false;
    }
    Assert::IsTrue(answer == true);
}
TEST_METHOD(Method_CountingSort)
{
    const int n = 6;
    char mas[n] = { 'c','d','e','v','g','v' };
    CountingSort(mas, n);
    bool answer;
    for (int i = 0; i < n-1; i++)
    {
        if (mas[i] <= mas[i + 1])
            answer = true;
        else answer = false;
    }
    Assert::IsTrue(answer == true);
}
```

```
CountingSort
Array:
c 1 d b c a
1 a b c c d Running time of algorithms
QuickSearch
10 elements 5.4e-06 sec
100 elements 0.0029496 sec
1000 elements 2.10507 sec
10000 elements 2105.79 sec

InsertionSort
10 elements 2.1e-06 sec
100 elements 0.0003139 sec
1000 elements 0.0208597 sec
10000 elements 2.12496 sec
100000 elements 211.11 sec
```


```
QuickSearch
10 elements 5.4e-06 sec
100 elements 0.0029496 sec
1000 elements 2.10507 sec
10000 elements 2105.79 sec

InsertionSort
10 elements 2.1e-06 sec
100 elements 0.0003139 sec
1000 elements 0.0208597 sec
10000 elements 2.12496 sec
100000 elements 211.11 sec
```