**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»**
**Вариант №1**

Студент гр. 0302 _____ Приезжих Т.А.

Преподаватель _____ Тутуева А.В.

Санкт-Петербург
**2021**

## Постановка задачи

Реализовать алгоритмы двоичного поиска, сортировки: быстрая, вставками, пузырьком, глупая (тип данных int), подсчётом (тип данных char).

## Описание реализуемых алгоритмов

1. Двоичный поиск (BinarySearch)

Алготим строит из элементов сортируемого массива двоичное дерево поиска. Обходит полученное дерево поиска обратным обходом в глубину.

2. Быстрая сортировка (QuickSort)

Алгоритм использует принцип «разделяй и властвуй»: решаемая задача рекурсивно разбивается на 2 или более подзадачи того же типа,но меньшего размера. Разбиения выполняются до тех пор, пока все подзадачи не окажутся элементарнымии легко решаемыми. В алгоритме массив разбивается на подмассивы, и для этих подмассивов рекурсивно вызываются алгоритм QuickSort.

3. Сортировка вставками (InsertionSort)

Алгоритм помещает несортированный элемент в подходящее место на каждой итерации. Сортировка вставками работает аналогично сортировке игральных карт в руке: предполагаем, что первая карта уже отсортирована, тогда выбираем неотсортированную карту. Если неотсортированная карта больше, чем карта в руке, она помещается справа, в противном случае - слева. Таким же образом берутся и другие неотсортированные карты и перемещаются на свои места.

5. Глупая сортировка (BogoSort)

Алгоритм выполняет случайное перемешивание элементов массива. Затем выполняется проверка массива на отсортированность. Если она нарушена, то происходит повторное перемешивание, пока массив не будет отсортирован.

6. Сортировка подсчётом (CountingSort) для типа char

Алгоритм сортирует элементы массива путем подсчета количества появлений каждого уникального элемента в массиве. Счетчик хранится во вспомогательном массиве, а сортировка выполняется путем сопоставления счетчика с индексом вспомогательного массива.

**Оценка временной сложности каждого алгоритма**

| Алгоритм | Временная сложность |
|---|---|
| **Двоичный поиск** | **O(n*log(n))** |
| **Быстрая сортировка** | **$O(n^2)$** |
| **Сортировка вставками** | **$O(n^2)$** |
| **Глупая сортировка** | **O(n*n!)** |
| **Сортировка подсчётом** | **O(max+n)** |

**Сравнение временной сложности алгоритмов 2 и 3**

| Количество элементов | Быстрая сортировка (в наносекудах) | Сортировка вставками (в наносекундах) |
|---|---|---|
| **10** | **1400** | **900** |
| **100** | **8600** | **5300** |
| **1000** | **97400** | **382600** |
| **10000** | **1018400** | **38719700** |
| **100000** | **10443800** | **3789929900** |

## Описание реализованных unit-тестов

Сначала тестируется метод isSorted(); путём подачи ему двух массивов, отсортированного и неотсортированного. На отсортированном массиве ожидается возврат True, на неотсортированном False.

Затем на основе этого метода тестируются все алгоритмы сортировки (отсортированный массив должен возвращать значение True).

Тестирование двоичного поиска заключается в подаче ему отсортированного массива. На поиск задаётся элемент массива (в данном случае 30 с индексом 2). Ожидается что алгоритм вернёт значение индекса (2).

## Пример работы

Пример работы QuickSort и Binary Search

```
Choose method of sort (Type a number)

        1 - Quick Sort
        2 - Insertion Sort
        3 - Bogo Sort
        4 - Counting Sort

1
        Array:
41 18467 6334 26500 19169 15724 11478 29358 26962 24464

        Char Array:
f u g c l q u r c l

        Quick Sort:
41 6334 11478 15724 18467 19169 24464 26500 26962 29358

QuickSort time :1300 ns


        Binary Search

Enter any number in array: 6334
That number got index: 1
```

## Листинг

Lab2.h

```
#pragma once
#include <iostream>
#include <stdlib.h>
#include <time.h>
#include <cstdlib>

using namespace std;

void PrintArray(int arr[], int SIZE) {
	for (int i = 0; i < SIZE; i++) {
		cout << arr[i] << " ";
	}
}

int Search_Binary(int arr[], int left, int right, int key)
{
	int mid = 0;
	while (1)
	{
		mid = (left + right) / 2;
		if (key < arr[mid])
			right = mid - 1;
		else if (key > arr[mid])
			left = mid + 1;
		else
			return mid;
		if (left > right)
			return -1;
	}
}

void QuickSort(int arr[], int SIZE) {
	int i = 0;
	int j = SIZE - 1;
```

```
	int mid = arr[SIZE / 2];
	do {
		while (arr[i] < mid) {
			i++;
		}
		while (arr[j] > mid) {
			j--;
		}
		if (i <= j) {
			int tmp = arr[i];
			arr[i] = arr[j];
			arr[j] = tmp;
			i++;
			j--;
		}
	} while (i <= j);

	if (j > 0) {
		QuickSort(arr, j + 1);
	}
	if (i < SIZE) {
		QuickSort(&arr[i], SIZE - i);
	}
}

void InsertionSort(int SIZE, int arr[]) {
	for (int k = 1; k < SIZE; k++)
	{
		int temp = arr[k];
		int j = k - 1;
		while (j >= 0 && temp <= arr[j])
		{
			arr[j + 1] = arr[j];
			j = j - 1;
		}
		arr[j + 1] = temp;
	}
}

bool isSorted(int arr[], int SIZE)
{
	while (--SIZE >= 1)
		if (arr[SIZE] < arr[SIZE - 1]) return false;
	return true;
}

bool isSortedChar(char arr[], int SIZE)
{
	while (--SIZE >= 1)
		if (arr[SIZE] < arr[SIZE - 1]) return false;
	return true;
}

void shuffle(int arr[], int SIZE)
{
	for (int i = 0; i < SIZE; i++)
		swap(arr[i], arr[rand() % SIZE]);
}

void BogoSort(int arr[], int SIZE)
{
	while (!isSorted(arr, SIZE))
		shuffle(arr, SIZE);
}
```

```
void CountingSort(char a[], int n)
{
	int max = INT_MIN, min = INT_MAX;
	for (int i = 0; i < n; i++) {
		if (a[i] > max)
			max = a[i];
		if (a[i] < min)
			min = a[i];
	}
	int* c = new int[max + 1 - min];
	for (int i = 0; i < max + 1 - min; i++) {
		c[i] = 0;
	}
	for (int i = 0; i < n; i++) {
		c[a[i] - min] = c[a[i] - min] + 1;
	}
	int i = 0;
	for (int j = min; j < max + 1; j++) {
		while (c[j - min] != 0) {
			a[i] = j;
			c[j - min]--;
			i++;
		}
	}
}

int Random(int min, int max) {
	return min + rand() % (max - min);
}
```

Lab2.cpp

```
#include <iostream>
#include <stdlib.h>
#include <time.h>
#include <cstdlib>
#include "Lab2.h"
#include <ctime>
#include <cstdio>
#include <chrono>

using namespace std;
using nanoseconds = chrono::duration<long long, nano>;


int main()
{
	setlocale(LC_ALL, "ru");

	const int SIZE = 10;
	int array[SIZE];
	int key = 0;
	int target = 0;
	char arrChar[SIZE];
	int Choice = 0;



	cout << "Choose method of sort (Type a number) \n\n";
	cout << "\t1 - Quick Sort\n";
	cout << "\t2 - Insertion Sort\n";
	cout << "\t3 - Bogo Sort\n";
	cout << "\t4 - Counting Sort\n\n";
	cin >> Choice;
```

```
	cout << "\tArray:\n";
	for (int i = 0; i < SIZE; i++) {
		array[i] = rand();
		//cout << array[i] << " ";
	}

	cout << "\n\n\tChar Array:\n";
	for (int i = 0; i < SIZE; i++) {
		arrChar[i] = Random(97, 122);
		//cout << arrChar[i] << " ";
	}

	if (Choice == 1) {

		auto start_time = std::chrono::steady_clock::now();

		QuickSort(array, SIZE);

		auto end_time = std::chrono::steady_clock::now();
		auto elapsed_ns =
std::chrono::duration_cast<std::chrono::nanoseconds>(end_time - start_time);

		cout << "\n\n\tQuick Sort:\n";
		//PrintArray(array, SIZE);
		cout << "\n\nQuickSort time :" << elapsed_ns.count() << " ns\n";
	}


	if (Choice == 2) {

		auto start_time = std::chrono::steady_clock::now();

		InsertionSort(SIZE, array);

		auto end_time = std::chrono::steady_clock::now();
		auto elapsed_ns =
std::chrono::duration_cast<std::chrono::nanoseconds>(end_time - start_time);

		cout << "\n\n\tInsertion Sort:\n";
		//PrintArray(array, SIZE);
		cout << "\n\nInsetrionSort time :" << elapsed_ns.count() << " ns\n";
	}


	if (Choice == 3) {
		BogoSort(array, SIZE);

		cout << "\n\n\tBogo Sort:\n";
		PrintArray(array, SIZE);
	}


	if (Choice == 4) {
		CountingSort(arrChar, SIZE);

		cout << "\n\n\tCounting Sort:\n";
		for (int i = 0; i < SIZE; i++)
			cout << arrChar[i] << " ";
	}

	cout << "\n\n\tBinary Search\n";
	cout << "\nEnter any number in array: ";
	cin >> key;
```

```
	target = Search_Binary(array, 0, SIZE, key);

	if (target >= 0)
		cout << "That number got index: " << target << "\n\n";
	else
		cout << "Array do not have that number\n\n";
}
```

UnitTest2.cpp

```
#include "pch.h"
#include "CppUnitTest.h"
#include "../Lab2/Lab2.h"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace UnitTest2
{
	TEST_CLASS(UnitTest2)
	{
	public:

		TEST_METHOD(Sorted) {
			const int SIZE = 5;
			int array1[SIZE] = { 10, 20, 30, 40, 50 };
			int array2[SIZE] = { 10, 220, 30, 40, 50 };

			Assert::IsTrue(isSorted(array1, SIZE));  //Expected that array1 is
sorted
			Assert::IsFalse(isSorted(array2, SIZE));  //Expected that array2 is
not sorted
		}
		TEST_METHOD(BinarySearch)
		{
			const int SIZE = 5;
			int array[SIZE] = { 10, 20, 30, 40, 50 };
			int search = 30;

			Assert::AreEqual(2, Search_Binary(array, 0, SIZE, search));
//Expexted that index of 30 is 2
		}

		TEST_METHOD(Quick_Sort)
		{
			const int SIZE = 5;
			int array[SIZE] = { 5, 4, 20, 40, 1 };

			Assert::IsFalse(isSorted(array, SIZE)); //Expected that array is not
sorted

			QuickSort(array, SIZE);

			Assert::IsTrue(isSorted(array, SIZE)); //Expected that array is
sorted
		}

		TEST_METHOD(Insertion_Sort)
		{
			const int SIZE = 5;
			int array[SIZE] = { 5, 4, 20, 40, 1 };

			Assert::IsFalse(isSorted(array, SIZE)); //Expected that array is not
sorted

			InsertionSort(SIZE, array);
```

```
			Assert::IsTrue(isSorted(array, SIZE)); //Expected that array is
sorted
		}

		TEST_METHOD(Bogo_Sort)
		{
			const int SIZE = 5;
			int array[SIZE] = { 5, 4, 20, 40, 1 };

			Assert::IsFalse(isSorted(array, SIZE)); //Expected that array is not
sorted

			BogoSort(array, SIZE);

			Assert::IsTrue(isSorted(array, SIZE)); //Expected that array is
sorted
		}

		TEST_METHOD(Counting_Sort)
		{
			const int SIZE = 5;
			char array[SIZE] = { 97, 99, 106, 98, 104 };

			Assert::IsFalse(isSortedChar(array, SIZE)); //Expected that array is
not sorted

			CountingSort(array, SIZE);

			Assert::IsTrue(isSortedChar(array, SIZE)); //Expected that array is
sorted
		}
	};
}
```