МИНОБРНАУКИ РОССИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технологический университет «СТАНКИН»
(ФГБОУ ВО «МГТУ «СТАНКИН»)

Институт цифровых интеллектуальных систем
Кафедра компьютерных систем управления

Направление подготовки: 15.03.04 Автоматизация технологических процессов и производств

Отчет по лабораторной работе №1

по дисциплине «Программирование и Алгоритмизация»

Вариант: 1

Выполнил:
студент гр. АДБ-22-06 _____ _____ Ахмадуллин А.Э.
(дата) (подпись)

Принял:
Преподаватель _____ _____ Пушков Р.Л.
(дата) (подпись)

Москва 2023

Оглавление

Решение заданий общей части

Задание для алгоритмов сортировки

Код:

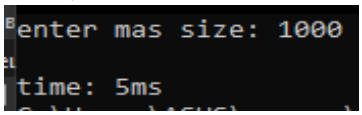
```
#include <iostream>
#include <stdlib.h>
#include <chrono>
#include <ctime>
using namespace std;
using namespace chrono;
void print(int* arr, int size)
{
	for (int i = 0; i < size; i++)
		cout << arr[i] << " ";
}

int* generate(int n)
{

	int* arr = new int[n];
	for (int i = 0; i < n; i++)
		arr[i] = rand() % 32767;
	return arr;
}

void bubbleSort(int a[], int n) {
	int buff = 0;
	for (int i = 0; i < n - 1; i++)
	{
		for (int j = n - 1; j > i; j--)
		{
			if (a[j] < a[j - 1])
			{
				buff = a[j - 1];
				a[j - 1] = a[j];
				a[j] = buff;
			}
		}
	}
}

int increment(long inc[], long size)
{
	int p1, p2, p3, s;
```

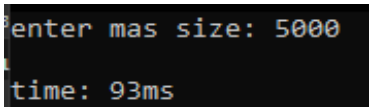
```
	p1 = p2 = p3 = 1;
	s = -1;
	do
	{
		if (++s % 2) {
			inc[s] = 8 * p1 - 6 * p2 + 1;
		}
		else {
			inc[s] = 9 * p1 - 9 * p3 + 1;
			p2 *= 2;
			p3 *= 2;
		}
		p1 *= 2;
	} while (3 * inc[s] < size);
	return s > 0 ? --s : 0;
}


void shellSort(int a[], long size) {
	long inc, i, j, seq[40];
	int s;
	// вычисление последовательности приращений
	s = increment(seq, size);
	while (s >= 0)
	{
		// сортировка вставками с инкрементами inc
		inc = seq[s--];
		for (i = inc; i < size; i++) {
			int temp = a[i];
			for (j = i - inc; (j >= 0) && (a[j] < temp); j -= inc)
				a[j + inc] = a[j];
			a[j + inc] = temp;
		}
	}
}



int main()
{
	setlocale(LC_ALL, "RU");
	srand(steady_clock::now().time_since_epoch().count());
	int size;
	cout << "enter mas size: "; cin >> size;
	int* a = generate(size);

	cout << endl << "первоначальный массив: "; print(a, size);
	time_point<steady_clock> start = steady_clock::now();
	bubbleSort(a, size);
	time_point<steady_clock> fin = steady_clock::now();
	cout << endl << "массив после сортировки по возрастанию: "; print(a, size);
	long dur = duration_cast<milliseconds>(fin - start).count();
	cout << endl << "time: " << dur << "ms";


	time_point<steady_clock> start1 = steady_clock::now();
	shellSort(a, size);
	cout << endl << "массив после сортировки по убыванию: "; print(a, size);
	time_point<steady_clock> fin1 = steady_clock::now();

	long dur2 = duration_cast<milliseconds>(fin1 - start1).count();
	delete[] a;
	cout << endl << "time: " << dur2 << "ms";
	return 0;
}
```

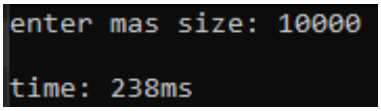
Замеры скорости работы пузырькового алгоритма (bubble-sort):

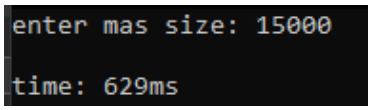
1. Сделать 10 замеров для количества 100000 чисел.
Из полученных значений после 10 замеров скорости работы алгоритма получены следующие значения (мс): 27532, 27665, 27401, 27721, 27394, 27539, 27465, 27583, 27508, 27472.
Среднее время работы алгоритма на основе полученных данных ~ 27528 мс.

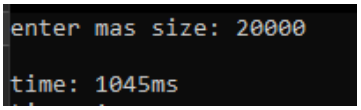
2. Сделать такие же замеры для различного числа элементов.
Полученные значения скорости работы относительно количества элементов:

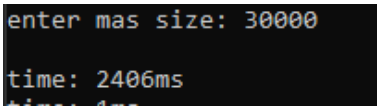




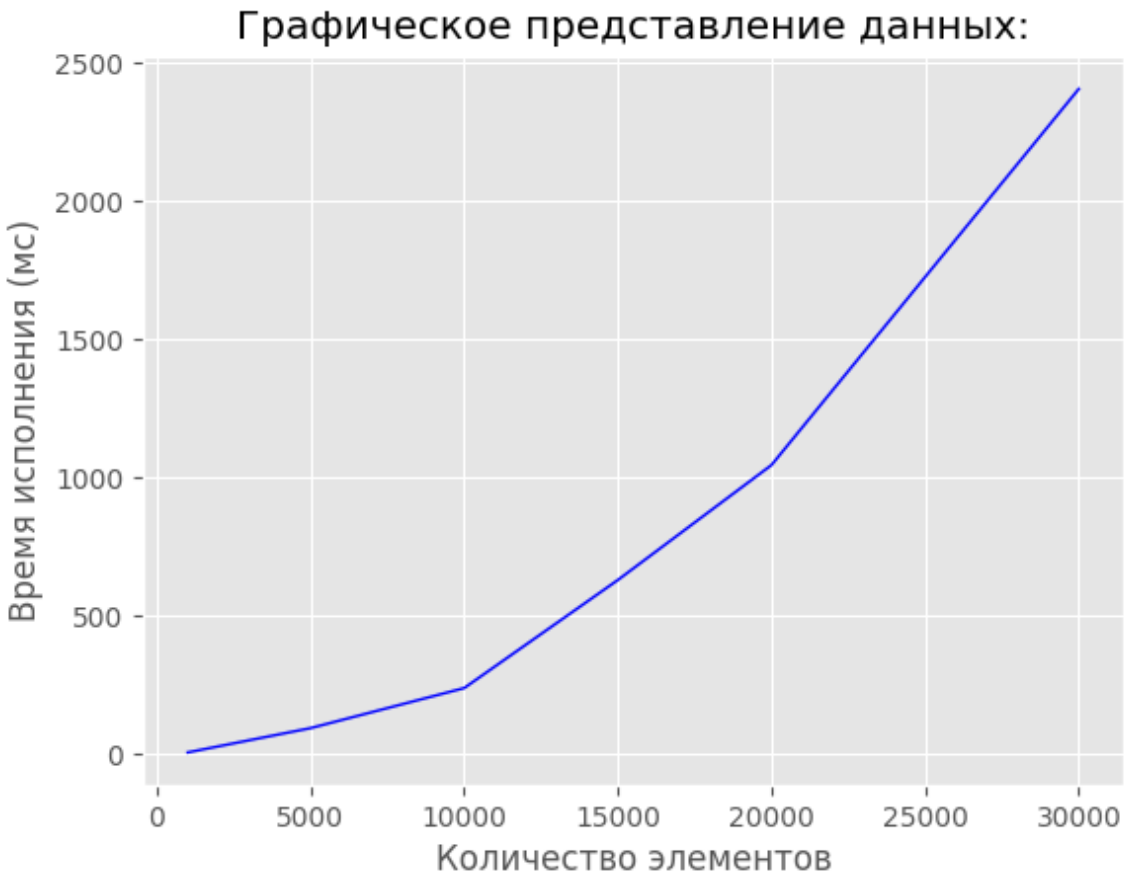








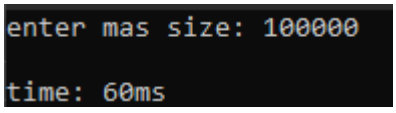
Количество элементов	**Время исполнения (мс)**
1000	5
5000	93
10000	238
15000	629
20000	1045
30000	2406

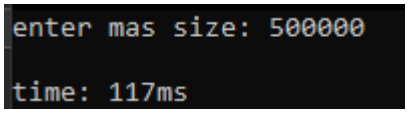


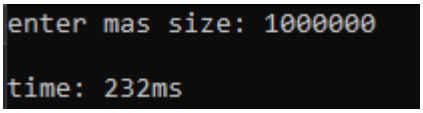
Замеры скорости работы алгоритма Шелла (shellsort):

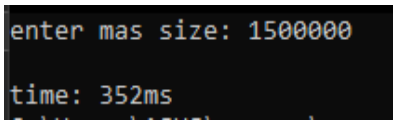
1. Сделать 10 замеров для количества 10000000 чисел.
Из полученных значений после 10 замеров скорости работы алгоритма получены следующие значения (мс): 2527, 2503, 2524, 2533, 2505, 2553, 2524, 2491, 2582, 2495.
Среднее время работы алгоритма на основе полученных данных ~ 2524 мс.

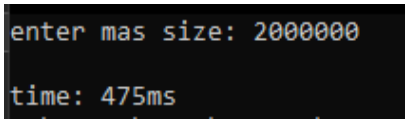
2. Сделать такие же замеры для различного числа элементов.
Полученные значения скорости работы относительно количества элементов:

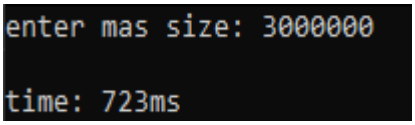


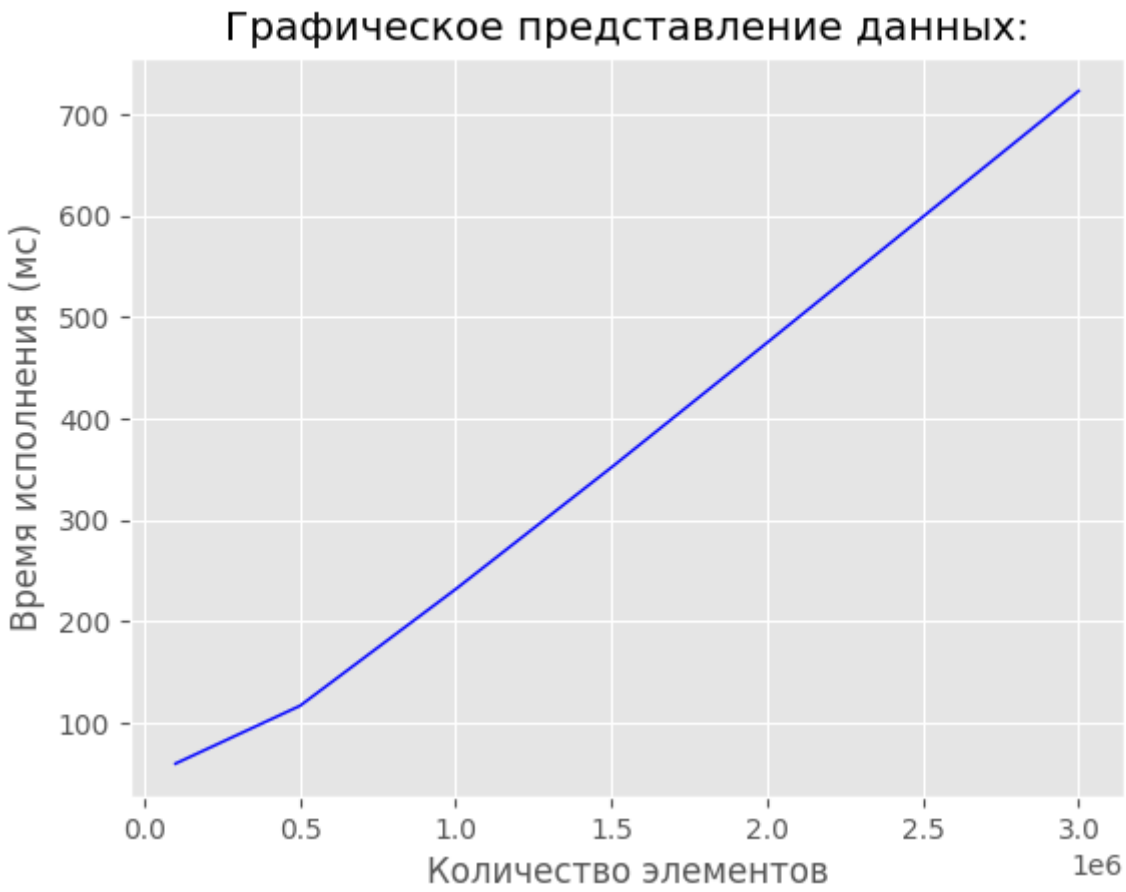












Количество элементов	**Время исполнения (мс)**
100000	60
500000	117
1000000	232
1500000	352
2000000	475
3000000	723

Вывод:
Проанализировав скорости работы алгоритмов мы видим, что алгоритм сортировки пузырьком намного более ресурсно-затратный чем алгоритм сортировки Шелла (~ 3960 эл/мс у ‘Шелла’ против ~3 эл/мс у ‘Пузырька’)

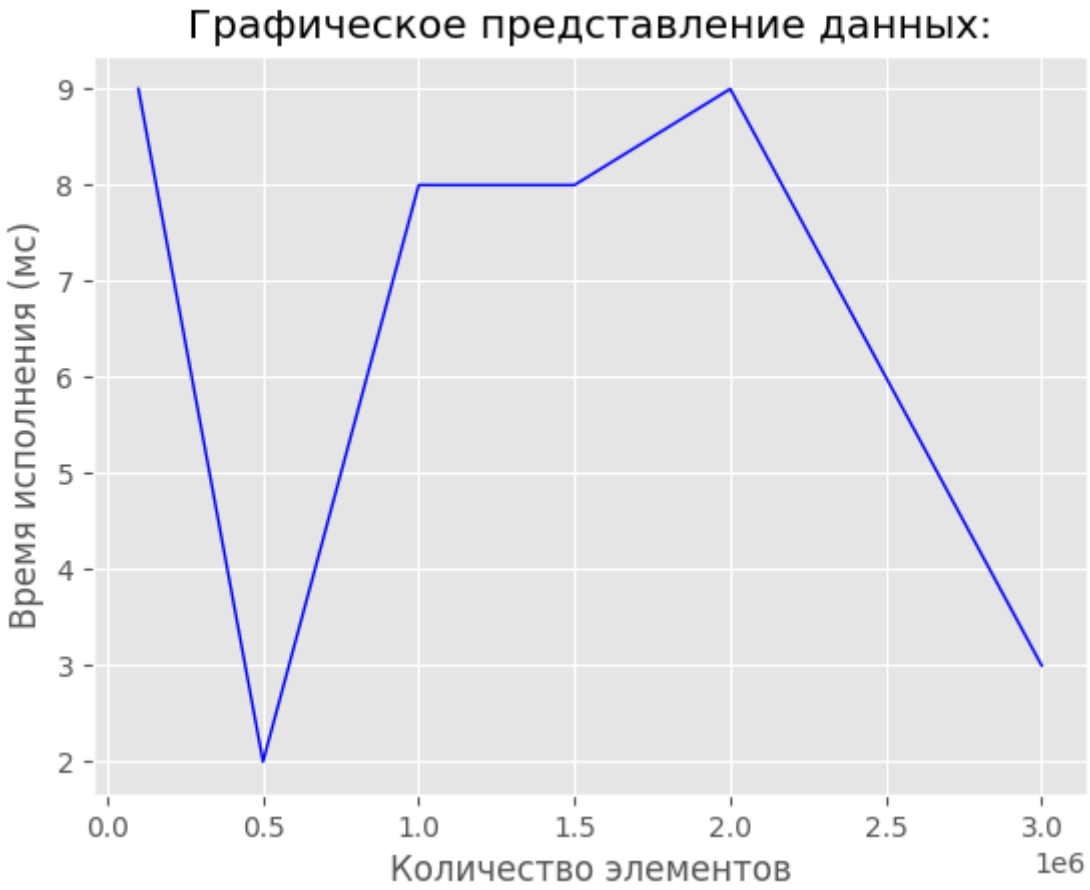
Задание для алгоритмов поиска

Замеры скорости работы переборного алгоритма:

1. Сделать 10 замеров для количества 1000000 чисел.
Из полученных значений после 10 замеров скорости работы алгоритма получены следующие значения (мс): 5, 5, 5 ,6 ,7 ,6, 5, 7, 6, 5.
Среднее время работы алгоритма на основе полученных данных ~ 5.7 мс.

2. Сделать такие же замеры для различного числа элементов.
Полученные значения скорости работы относительно количества элементов:

Количество элементов	**Время исполнения (мс)**
100000	9
500000	2
1000000	8
1500000	8
2000000	9
3000000	3



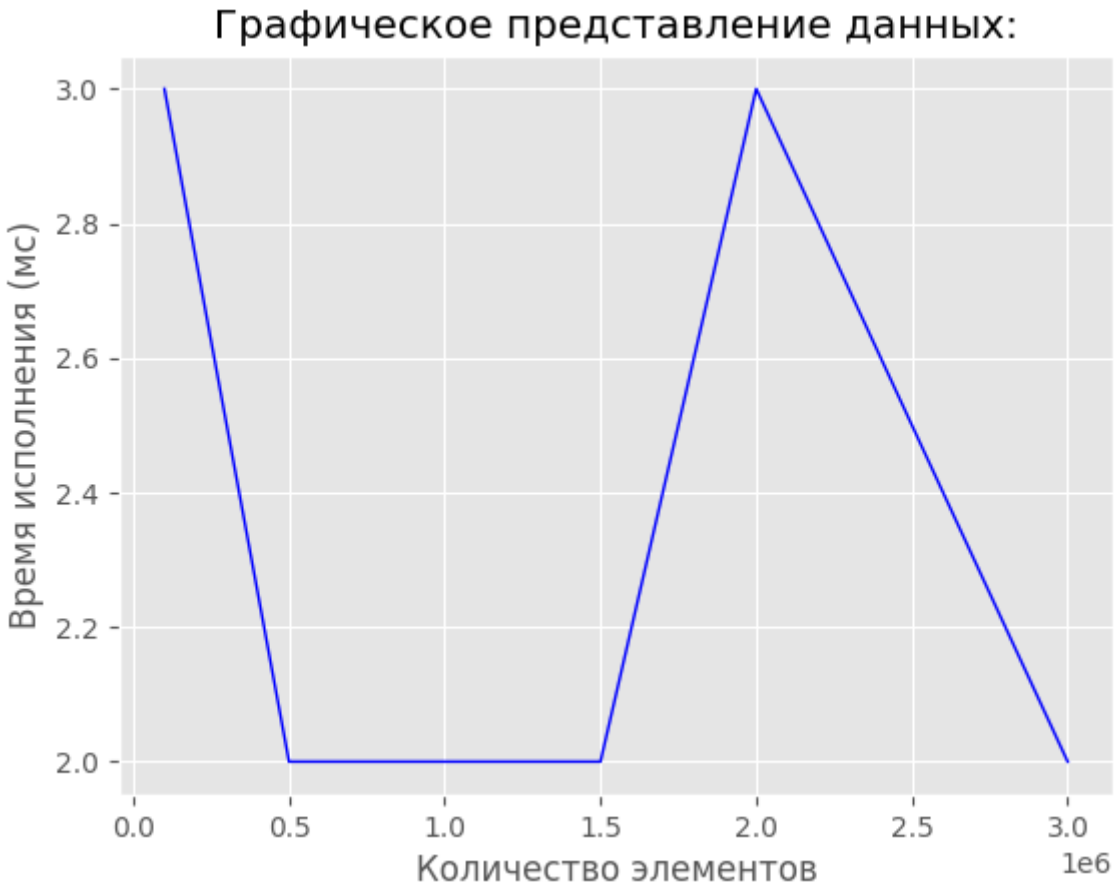
Замеры скорости работы алгоритма бинарного поиска:

1. Сделать 10 замеров для количества 1000000 чисел.
Из полученных значений после 10 замеров скорости работы алгоритма получены следующие значения (мс): 2, 2, 2, 2, 2, 3, 2, 3, 2 , 2.
Среднее время работы алгоритма на основе полученных данных ~ 2.2 мс.

2. Сделать такие же замеры для различного числа элементов.
Полученные значения скорости работы относительно количества элементов:

Количество элементов	**Время исполнения поиска и сортировки(мс)**	**Время исполнения поиска (мс)**
100000	72	3
500000	113	2
1000000	239	2

1500000	410	2
2000000	522	3
3000000	729	2



Код:

```
#include <iostream>
#include <stdlib.h>
#include <chrono>
#include <ctime>
using namespace std;
using namespace chrono;

void print(int* arr, int size)
{
	for (int i = 0; i < size; i++)
		cout << arr[i] << " ";
}

int* generate(int n)
{

	int* arr = new int[n];
	for (int i = 0; i < n; i++)
		arr[i] = rand() % 32767;
	return arr;
```

```
}


int increment(long inc[], long size)
{
	int p1, p2, p3, s;
	p1 = p2 = p3 = 1;
	s = -1;
	do
	{
		if (++s % 2) {
			inc[s] = 8 * p1 - 6 * p2 + 1;
		}
		else {
			inc[s] = 9 * p1 - 9 * p3 + 1;
			p2 *= 2;
			p3 *= 2;
		}
		p1 *= 2;
	} while (3 * inc[s] < size);
	return s > 0 ? --s : 0;
}


void shellSort(int a[], long size) {
	long inc, i, j, seq[40];
	int s;
	// вычисление последовательности приращений
	s = increment(seq, size);
	while (s >= 0)
	{
		// сортировка вставками с инкрементами inc
		inc = seq[s--];
		for (i = inc; i < size; i++) {
			int temp = a[i];
			for (j = i - inc; (j >= 0) && (a[j] > temp); j -= inc)
				a[j + inc] = a[j];
			a[j + inc] = temp;
		}
	}
}

int enumSearch(int a[], long size, int val)
{
	for (int i = 0; i < size; i++)
	{
		if (a[i] == val)
			return i;
	}
	return -1;
}

int binarySearch(int a[], long size, int val)
{
	int lastIndex = size - 1;
	int firstIndex = 0;
	while (lastIndex >= firstIndex)
	{
		int middleIndex = (firstIndex + lastIndex) / 2;
		if (a[middleIndex] == val)
		{
			return middleIndex;
		}
		else
			if (a[middleIndex] < val)
```

```
                        firstIndex = middleIndex + 1;
                else
                        if (a[middleIndex] > val)
                                lastIndex = middleIndex - 1;
        }
        return -1;
}


int main()
{
        setlocale(LC_ALL, "RU");
        srand(steady_clock::now().time_since_epoch().count());
        int size;
        cout << "enter mas size: "; cin >> size;
        int* a = generate(size);
        cout << endl << "начальный массив: "; print(a, size);

        int val; cout << "введите число для поиска: "; cin >> val;
        time_point<steady_clock> start = steady_clock::now();
        if (enumSearch(a, size, val) != -1)
                cout << endl <<"позиция значения: " << enumSearch(a, size, val);
        else
                cout << endl << "число не найдено";
        time_point<steady_clock> fin = steady_clock::now();

        time_point<steady_clock> start1 = steady_clock::now();
        shellSort(a, size);
        time_point<steady_clock> fin1 = steady_clock::now();

        time_point<steady_clock> start2 = steady_clock::now();
        if (binarySearch(a, size, val) != -1)
                cout << endl << "позиция значения : " << binarySearch(a, size, val);
        else
                cout << endl << "число не найдено";
        time_point<steady_clock> fin2 = steady_clock::now();


        long dur = duration_cast<milliseconds>(fin - start).count();
        long dur1 = duration_cast<milliseconds>(fin1 - start1).count();
        long dur2 = duration_cast<milliseconds>(fin2 - start2).count();
        cout << endl << "1st algorythm time : " << dur << "ms";
        cout << endl << "2nd algorythm + sort time: " << dur1 + dur2 << "ms";
        cout << endl << "2nd algorythm time: " << dur2 << "ms";
        delete[] a;
        return 0;
}
```

Вывод:

Проанализировав скорости работы алгоритмов мы видим, что алгоритм поиска перебором более ресурсно-затратный чем алгоритм бинарного поиска (Среднее время поиска для одинакового количества элементов перебором составляет ~5.7 мс, в то время как Бинарным поиском всего ~2.2 мс)

Выполнение индивидуального задания

Успеваемость студентов. Фамилиям студентов двух групп Вашего потока поставьте в соответствие случайное число из интервала от 25 до 54. Отсортируйте получившийся список по алфавиту, а затем по возрастанию

баллов для каждой буквы. Используйте два разных алгоритма сортировки. Найдите в получившемся списке свою фамилию, не используя поиск перебором.

Код:

```
#include <iostream>
#include <vector>
#include <windows.h>
#include <fstream>
#include <string>
using namespace std;

class Student
{
public:
    string surname;
    int number;
};


int increment(long inc[], long size) {
    int p1, p2, p3, s;
    p1 = p2 = p3 = 1;
    s = -1;
    do
    {
        if (++s % 2) {
            inc[s] = 8 * p1 - 6 * p2 + 1;
        }
        else {
            inc[s] = 9 * p1 - 9 * p3 + 1;
            p2 *= 2;
            p3 *= 2;
        }
        p1 *= 2;
    } while (3 * inc[s] < size);
    return s > 0 ? --s : 0;
}
void shellSort(vector<Student>& a, long size) {
    long inc, i, j, seq[40];
    int s;
    // вычисление последовательности приращений
    s = increment(seq, size);
    while (s >= 0)
    {
        // сортировка вставками с инкрементами inc
        inc = seq[s--];
        for (i = inc; i < size; i++) {
            string temp = a[i].surname;
            for (j = i - inc; (j >= 0) && (a[j].surname > temp); j -= inc)
                a[j + inc] = a[j];
            a[j + inc].surname = temp;
        }
    }
}



int binarySearch(vector<Student> students, string val)
{
    int lastIndex = students.size() - 1;
    int firstIndex = 0;
    while (lastIndex >= firstIndex)
```

```
    {
        int middleIndex = (firstIndex + lastIndex) / 2;
        if (students[middleIndex].surname == val)
        {
            return middleIndex;
        }
        else
            if (students[middleIndex].surname < val)
                firstIndex = middleIndex + 1;
            else
                if (students[middleIndex].surname > val)
                    lastIndex = middleIndex - 1;
    }
    return -1;
}

void bubbleSort(vector <Student>& a, int start, int fin) {
    Student buff;
    for (int i = start; i < fin - 1; i++)
    {
        for (int j = fin - 1; j > i; j--)
        {
            if (a[j].number < a[j - 1].number)
            {
                buff = a[j - 1];
                a[j - 1] = a[j];
                a[j] = buff;
            }
        }
    }
}

void GetSlice(vector <Student>& a)
{
    int buff = 0;

    for (int i = 1; i < a.size(); i++)
    {
        if (a[buff].surname[0] != a[i].surname[0])
        {
            if (i != 1)
            {
                bubbleSort(a, buff, i);
                buff = i;
            }
            else
            {
                buff = i;
            }
        }
    }
}

int main() {
    setlocale(LC_ALL, "RU");
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    vector<Student> students;

    ifstream file("text.txt"); // открываем файл
    if (file.is_open()) { // проверяем, что файл успешно открыт
        std::string str;
        int num;
        while (file >> str >> num) { // считываем строки и числа из файла
```

```
            students.push_back({ str, num }); // выводим считанные данные на экран
        }
        file.close(); // не забываем закрыть файл после использования
    }
    else {
        cout << "Ошибка чтения файла" << endl; // сообщаем об ошибке, если файл не
удалось открыть
    }
    shellSort(students, students.size());
    GetSlice(students);

    for (const auto& student : students) {
        cout << student.surname << " " << student.number << endl;
    }

    string target; cout << "Кого хотим найти? "; cin >> target;
    int result = binarySearch(students, target);

    if (result != -1)
        cout << "Найден(а) " << target << " на месте номер " << result << endl;
    else
        cout << target << " не найдено" << endl;

    return 0;
}
```

Вывод программы:

```
Ахмадуллин 54
Баранов 45
Гибадуллин 26
Гажев 35
Гайнетдинов 40
Голубев 52
Грязнов 52
Данилов 34
Дураджи 45
Еникеев 31
Забунов 29
Зиновьев 29
Искоростинский 32
Исхаков 37
Кищенко 54
Попов 54
Кого хотим найти? Кищенко
Найден(а) Кищенко на месте номер 14
```