Министерство науки и высшего образования Российской Федерации
Федеральное государственное образовательное учреждение высшего образования
«Алтайский государственный технический университет им. И. И. Ползунова»

Факультет информационных технологий

Отчет защищен с оценкой _____

_____Е. В. Астахова
(подпись преподавателя)
«____» _____2025 г.

Отчет по лабораторной работе № 7

Эффективность алгоритмов

по дисциплине Основы программной инженерии – 1 семестр

ЛР 09.03.04.2 ОТ
ЛР 09.03.04.11 ОТ
ЛР 09.03.04.12 ОТ
ЛР 09.03.04.20 ОТ

Студент группы ПИ-54 Балицкий Богдан
Студент группы ПИ-54 Дворников Марк
Студент группы ПИ-54 Елунин Игорь
Студент группы ПИ-54 Лавриненко Алексей

Преподаватель доцент, к.п.н Е. В. Астахова

Барнаул 2025

Задание 1. Анализ сложности алгоритма

Даны функции, используемые при вычислении эффективности алгоритмов: log2(n), n , n*log2(n), n2, 2n, n!

1) Вычислить значения функций (количество операций) для n, равном
   а) 10, 20, …., 100
   б) 10, 10^2, …, 10^9

Значения функций (кол-во операций)						
	log2(n)	**n**	**n*log2(n)**	**n^2**	**2^n**	**n!**
10	3,321928095	10	33,21928	100	1024	3628800
20	4,321928095	20	86,43856	400	1048576	2,4329E+18
30	4,906890596	30	147,2067	900	1073741824	2,65253E+32
40	5,321928095	40	212,8771	1600	1,09951E+12	8,15915E+47
50	5,64385619	50	282,1928	2500	1,1259E+15	3,04141E+64
60	5,906890596	60	354,4134	3600	1,15292E+18	8,32099E+81
70	6,129283017	70	429,0498	4900	1,18059E+21	1,1979E+100
80	6,321928095	80	505,7542	6400	1,20893E+24	7,1569E+118
90	6,491853096	90	584,2668	8100	1,23794E+27	1,4857E+138
100	6,64385619	100	664,3856	10000	1,26765E+30	9,3326E+157
10	3,321928095	10	33,21928	100	1024	3628800
100	6,64385619	100	664,3856	10000	1,26765E+30	9,3326E+157
1000	9,965784285	1000	9965,784	1000000	1,0715E+301	1000!
10000	13,28771238	10000	132877,1	1E+08	2 ^ 10000	10000!
100000	16,60964047	100000	1660964	1E+10	2 ^ 100000	100000!
1000000	19,93156857	1000000	19931569	1E+12	2 ^ 1000000	1000000!
10000000	23,25349666	10000000	2,33E+08	1E+14	2 ^ 10000000	10000000!
100000000	26,57542476	1E+08	2,66E+09	1E+16	2 ^ 100000000	100000000!
1000000000	29,89735285	1E+09	2,99E+10	1E+18	2 ^ 1000000000	1000000000!

2) Вычислить время (секунды, часы, дни, годы) выполнения операций при условии, что быстродействие компьютера 10^6 операций в секунду. Результат представить в целых числах.

Быстродействие компьютера
1000000

Время выполнения операций						
	log2(n)	**n**	**n*log2(n)**	**n^2**	**2^n**	**n!**
10	0 с.	0 с.	0 с.	0 с.	0 с.	4 с.
20	0 с.	0 с.	0 с.	0 с.	1 с.	77147 лет
30	0 с.	0 с.	0 с.	0 с.	18 мин.	8411113007743250000 лет
40	0 с.	0 с.	0 с.	0 с.	13 д.	2,58725039081652E+34 лет
50	0 с.	0 с.	0 с.	0 с.	36 лет	9,6442456880116E+50 лет
60	0 с.	0 с.	0 с.	0 с.	36559 лет	2,63856770444615E+68 лет
70	0 с.	0 с.	0 с.	0 с.	37436315 лет	3,79838015917361E+86 лет
80	0 с.	0 с.	0 с.	0 с.	38334786264 лет	2,269452595328E+105 лет
90	0 с.	0 с.	0 с.	0 с.	39254821134113 лет	4,7111744180675E+124 лет
100	0 с.	0 с.	0 с.	0 с.	40196936841331500 лет	2,9593548783595E+144 лет
10	0 с.	0 с.	0 с.	0 с.	0 с.	4 с.
100	0 с.	0 с.	0 с.	0 с.	40196936841331500 лет	2,9593548783595E+144 лет
1000	0 с.	0 с.	0 с.	1 с.	3,397731504269E+287 лет	#ЗНАЧ!
10000	0 с.	0 с.	0 с.	2 мин.	#ЗНАЧ!	#ЗНАЧ!
100000	0 с.	0 с.	2 с.	3 ч.	#ЗНАЧ!	#ЗНАЧ!
1000000	0 с.	1 с.	20 с.	12 д.	#ЗНАЧ!	#ЗНАЧ!
10000000	0 с.	10 с.	4 мин.	3 лет	#ЗНАЧ!	#ЗНАЧ!
100000000	0 с.	2 мин.	44 мин.	317 лет	#ЗНАЧ!	#ЗНАЧ!
1000000000	0 с.	17 мин.	8 ч.	31710 лет	#ЗНАЧ!	#ЗНАЧ!

3) Вычислить, во сколько раз увеличивается время выполнения алгоритма при увеличении объема данных n

	log2(n)	n	n*log2(n)	n^2	2^n	n!
10	-	-	-	-	-	-
20	-	-	-	-	-	670442572800
30	-	-	-	-	1074	109027350432000
40	-	-	-	-	1024	3075990524006400
50	-	-	-	-	1024	37276043023296000
60	-	-	-	-	1024	273589847231501000
70	-	-	-	-	1024	1439561377475020000
80	-	-	-	-	1024	5974790569203450000
90	-	-	-	-	1024	20759078324729600000
100	-	-	-	-	1024	62815650955529500000
10	-	-	-	-	-	-
100	-	-	-	-	2 ^ 90	100! / 10!
1000	-	-	-	-	2 ^ 900	1000! / 100!
10000	-	-	-	100,000000	2 ^ 9000	10000! / 1000!
100000	-	-	-	100,000000	2 ^ 90000	100000! / 10000!
1000000	-	-	12,000000	100,000000	2 ^ 900000	1000000! / 100000!
10000000	-	10,000000	11,666667	100,000000	2 ^ 9000000	10000000! / 1000000!
100000000	-	10,000000	11,428571	100,000000	2 ^ 90000000	100000000! / 10000000!
1000000000	-	10,000000	11,250000	100,000000	2 ^ 900000000	1000000000! / 100000000!

Задание 2. Тестирование программы на время выполнения

Провести тестирование программы (программа на след. странице) на время выполнения в различных условиях.

Условия тестирования	**Время работы программы**
Начальные условия	21.577
Изменение загрузки компьютера	30.309
Массив содержит ряд натуральных чисел	23.292
Массив содержит одинаковые значения одноразрядного десятичного числа	16.809
Массив содержит одинаковые значения 9-разрядного десятичного числа	82.053

Задание 3. Оценка эффективности алгоритма

1) Оценить временную трудоемкость алгоритма T(n)

В худшем случае:

а) относительно операции сравнения - T(200.000), потому что программа будет сравнивать все n (в нашем случае 200.000) элементов

б) относительно операции присваивания - T(99), потому что мы берём значения диапазона, в котором лежит 99 чисел

В лучшем случае:

а) относительно операции сравнения - T(200.000), потому что программа будет сравнивать все n (в нашем случае 200.000) элементов

б) относительно операции присваивания - T(1), потому что мы сразу возьмём самое маленькое число

В среднем случае:

а) относительно операции сравнения - T(200.000), потому что программа будет сравнивать все n (в нашем случае 200.000) элементов

б) относительно операции присваивания - T(50), потому что это среднее между 99 и

Текст программы:

```
#include <stdio.h>
#include <iostream>
#include <ctime>
using namespace std;
int main(int argc, char* argv[])
{
	srand(time(0));
	const int array_size = 200000; // размер массива
	int a[array_size]; // объявление массива
	for (int counter = 0; counter < array_size; counter++)
	{
		a[counter] = rand() % 50 - rand() % 50; //Начальные условия
		//a[counter] = rand() % 100 + 1; - Массив содержит ряд натуральных чисел
		//a[counter] = 5; - Массив содержит одинаковые значения одноразрядного десятичного числа
		//a[counter] = 555555555; - Массив содержит одинаковые значения 9-разрядного десятичного числа
		cout << a[counter] << " "; // печать элементов массива
	}
	int min = a[0]; // переменная для хранения минимального значения
	for (int counter = 1; counter < array_size; counter++)
	{
		if (min > a[counter]) // поиск минимального значения в массиве
			min = a[counter];
	}
	cout << "\nmin = " << min << endl;
	cout << "runtime = " << clock() / 1000.0 << endl; // время работы программы
	system("pause"); return 0;
}
```

2) Определить, во сколько раз изменится трудоемкость, если размер массива увеличится вдвое.

   Трудоемкость относительно заполнения массива и операции сравнения увеличатся в 2 раза. (т.е. пропорционально увеличению массива)

Задание 4.1. Оценка сложности программы

1) Оценить сложность программы по метрике Холстеда

Операторы	Число операторов	Операнды	Число операндов
include	3	<stdio.h>	1
using	1	<iostream.h>	1
namespace	1	<ctime>	1
int	7	namespace	1
char	1	std	1
*	1	main	1
[]	7	argc	1
;	16	argv	1
,	1	0	4
const	1	array_size	4
=	6	200000	1
for	2	a	6
<	2	counter	10
++	2	50	2
{}	3	cout	3
()	10	“ “	1
%	2	min	4
-	1	1	1
<<	8	“\nmin = ”	1
if	1	endl	2
>	1	“runtime = “	1
/	1	1000.0	1
return	1	“pause”	1
		srand	1
		time	1
		rand	2
		clock	1
		system	1

n1 = 28 N1 = 83 n` = 2
n2 = 23 N2 = 52

Характеристика	**Формула**	**Результат**
Словарь программы	n = n1 + n2	n = 48
Длина реализации программы	N = N1 + N2	N = 135
Теоретическая длина программы	$N` = n1*\log_2(n1) + n2*\log_2(n2)$	N` = 220,138
Объём программы	$V = N * \log_2 n$ (бит)	V = 753,97
Потенциальный объём программы	$V` = (n` + 2)*\log_2(n` + 2)$	V`= 8
Уровень реализации программы	L = V` / V	L = 0,01

2) Какие выводы можно сделать по данным характеристикам?

Уровень реализации L ≈ 0.01, что характерно для программ с малым числом входов/выходов относительно общего объема кода. Длина реализации меньше, чем теоретическая длина программы, что означает высокое качество кода

Задание 4.2. Оценка временных затрат

1) Оценить затраты времени на создание кода программы по метрике Холстеда анализируя текст
   a) Исходной программы
   b) Программы с добавленными элементами защитного программирования (см. задание 5)

Заполнить таблицу

Характеристика	**Формула**	**Результат (а)**	**Результат (б)**
Сложность программы	D = (n1/2)*(N2/n2)	25,76	26,22
Усилия программиста при разработке	H = D*V	19422,266	20208,405
Время, затраченное на кодирование	T = H/18 (с)	1079,015 (с) ~ 18 минут	1122,689 (с) ~ 19 минут

Задание 5. Защитное программирование

1) Описать элементы защитного программирования, использованные в программе.

   Длинна массива сохраняется в отдельную переменную, поэтому цикл не выходит за пределы массива. Также переменная counter инициализируется в цикле:

```
const int array_size = 200000; // размер массива
  int a[array_size]; // объявление массива
  for (int counter = 0; counter < array_size; counter++)
```

Для нахождения минимального числа берётся самый первый элемент, значит минимальный элемент всегда будет лежать в массиве:

```
int min = a[0]; // переменная для хранения минимального значения
  for (int counter = 1; counter < array_size; counter++)
```

2) Добавить в программу свои элементы защитного программирования.

Просим пользователя нажать на кнопку, чтобы продолжить и принимаем символ через getchar, чтобы код не зависел от ОС
cout << "Press any key to continue...";
getchar(); return 0;

Это не увеличит n1 и n2
Увеличит N1 на 2, N2 на 1