МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Отчет по выполнению практического задания 3

Тема:

Эмпирический анализ алгоритмов сортировки.

Выполнил студент Цемкало А. Р.

Фамилия И.О.

группа ИКБО-10-20

Москва 2021

СОДЕРЖАНИЕ

Вариант 2.

Задание 1. Оценить зависимость времени выполнения алгоритма простой сортировки на массиве, заполненном случайными числами.

1.1. Алгоритм сортировки по методу простого обмена (Пузырёк) (Exchange sort) на случайно заполненном массиве.

```
void bubble_sort(int* list, int n) {
    for (int i = 0; i < n; i++) {
        for (int j = n - 1; j > i; j--) {
            if (list[j - 1] > list[j]) {
                int t = list[j - 1];
                list[j - 1] = list[j];
                list[j] = t;
            }
        }
    }
}
```

1.2. Определение функции роста времени выполнения сортировки методом простого обмена (Пузырёк) (Exchange sort) при увеличении объёма массива n.

Все данные алгоритма, для вывода функции роста представлены в таблице:

Номер оператора	Оператор	Время выполнения одного оператора	Кол-во выполнений оператора в строке
	1	2	3
1	for (int i = 0; i < n; i++) {	C1	n +1 раз
2	for (int j = n - 1; j > i; j--) {	C2	$\sum_{i=0}^{n-1} t_i$
3	if (list[j - 1] > list[j]) {	C3	$\sum_{i=1}^{n-1} t_i$

4	int t = list[j];
	list[j] = list[j - 1];
	list[j - 1] = t;
		C4	$\sum_{i=1}^{n-1} t_i$

5	}		
6	}		
7	}		

Выведем функцию роста для времени выполнения алгоритма:

$$\mathrm{T(n)} = \mathrm{C1} * (\mathrm{n} + 1) + \mathrm{C2} * \sum_{i=0}^{n-1} t_i + \mathrm{C3} * \sum_{i=1}^{n-1} t_i + C4 * \sum_{i=1}^{n-1} t_i \qquad (1)$$

Определим порядок роста в лучшем случае, т.е. когда тело вложенного цикла не выполняется (массив уже отсортирован по рассматриваемому в алгоритме правилу). В наилучшем случае в сумме $\sum_{i=0}^{n-1} t_i$ значение t_i равно i.

Тогда значение $\sum_{i=0}^{n-1} t_i = \frac{n(n-1)}{2} = \frac{n^2-n}{2}$. Оператор 3 выполнится $\frac{(n-1)(n-2)}{2} = \frac{n^2-2n+1}{2}$. C4 не выполнится вообще.

Подставим эти значения в формулу (1):

$$\mathrm{T(n)} = \mathrm{C1} * (\mathrm{n}+1) + \mathrm{C2} * \frac{n^2-n}{2} + \mathrm{C3} * \frac{n^2-2n+1}{2} + C4 * 0 =$$
$$= \left(\frac{\mathrm{C2}}{2} + \frac{\mathrm{C3}}{2}\right) n^2 + \left(\mathrm{C1} - \frac{\mathrm{C2}}{2} - \mathrm{C3}\right) * \mathrm{n} + \left(\mathrm{C1} + \frac{\mathrm{C3}}{2}\right) = \mathrm{An}^2 + \mathrm{Bn} + \mathrm{C}$$

Пренебрегаем константой С. Получаем T(n)= An^2+Bn. Функция n^2 имеет порядок роста выше, чем функция n. Таким образом, в выражении T(n)= An^2+Bn, доминирующей функцией является n^2, и она определяет порядок роста для алгоритма в лучшем случае. Т.е. T(n)=Θ(n^2).

Определим порядок роста *в худшем случае*, т.е. когда оператор 4 выполняется полное количество раз (массив упорядочен, но по правилу, противоположному тому, что рассматривает алгоритм).

В сумме $\sum_{i=0}^{n-1} t_i$ значение t_i равно i. Тогда значение $\sum_{i=0}^{n-1} t_i = \frac{n(n-1)}{2} = \frac{n^2-n}{2}$. Операторы 3 и 4 выполнятся $\frac{(n-1)(n-2)}{2} = \frac{n^2-2n+1}{2}$.

Подставим эти значения в формулу (1):

$$\mathrm{T(n)} = \mathrm{C1} * (\mathrm{n}+1) + \mathrm{C2} * \frac{n^2-n}{2} + \mathrm{C3} * \frac{n^2-2n+1}{2} + C4 * \frac{n^2-2n+1}{2} =$$
$$= \left(\frac{\mathrm{C2}}{2} + \frac{\mathrm{C3}}{2} + \frac{\mathrm{C4}}{2}\right) n^2 + \left(\mathrm{C1} - \frac{\mathrm{C2}}{2} - \mathrm{C3} - C4\right) * \mathrm{n} + \left(\mathrm{C1} + \frac{\mathrm{C3}}{2} + \frac{\mathrm{C4}}{2}\right)$$
$$= \mathrm{An}^2 + \mathrm{Bn} + \mathrm{C}$$

Пренебрегаем константой С. Получаем T(n)= An^2+Bn. Функция n^2 имеет порядок роста выше, чем функция n. Таким образом, в выражении T(n)= An^2+Bn, доминирующей функцией является n^2, и она определяет порядок роста для алгоритма в худшем случае. Т.е. T(n)=Θ(n^2).

1.3. Сводная таблица результатов выполнения сортировки по указанным объёмам на случайно заполненном массиве для всех указанных объёмов.

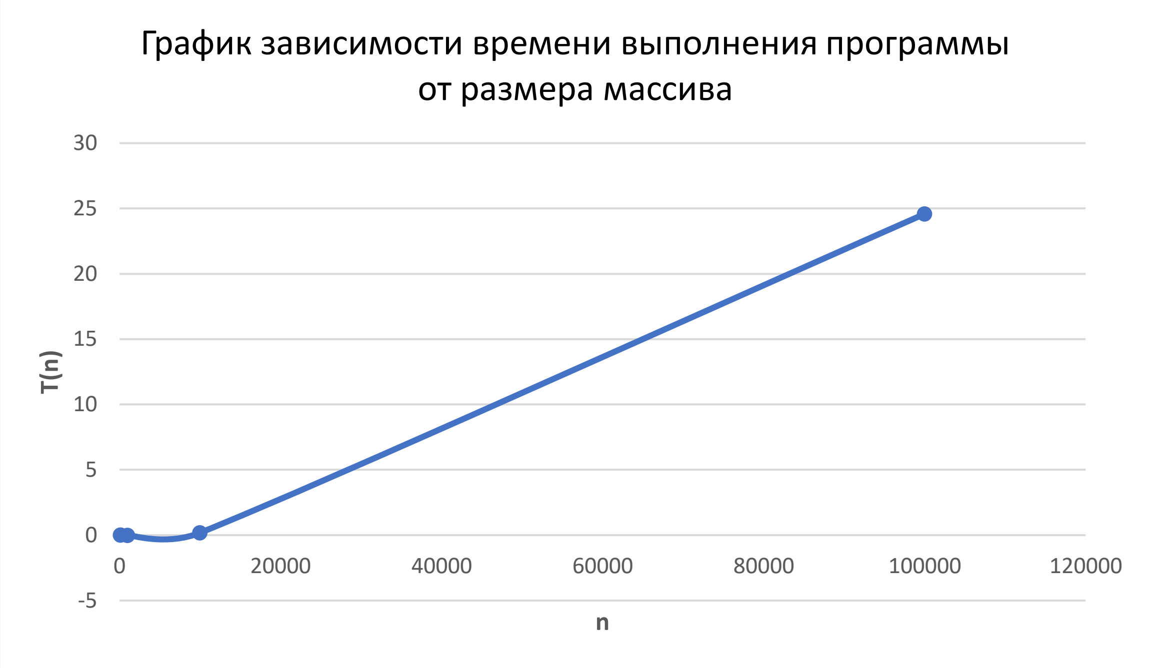
Таблица 1

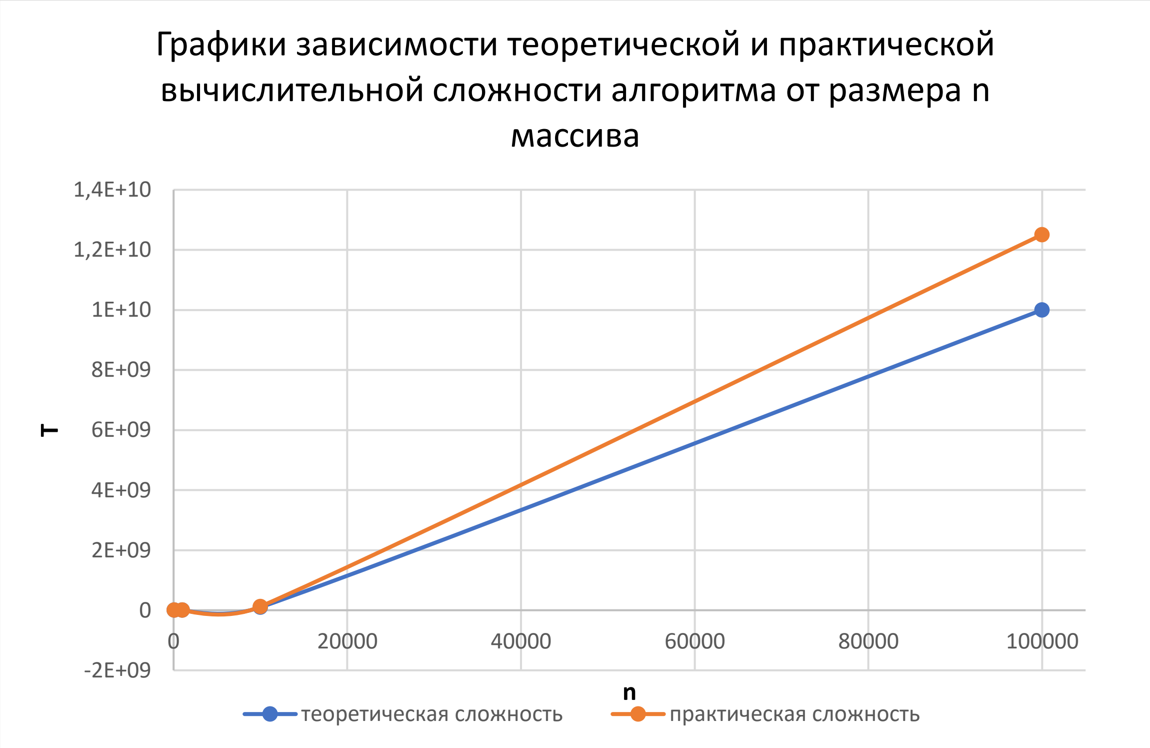
n	**T(n), сек**	**T_T=f(C+M)**	**$T_п$=Cф+Mф**
100	0.0074663	10000	10002+2360
1000	0.0067894	1000000	1000002+248157
10000	0.184543	100000000	100000002+24952888
100000	24.5813	10000000000	10000000002+2506044511
1000000	-	-	-

1.4. Код алгоритма и основной программы, доказывающей выполнение тестирования.

```cpp
#include <iostream>
#include <chrono>
using namespace std;

void bubble_sort(int* list, int n) {
    long long int compare = 2, swapping = 0;
    for (int i = 0; i < n; i++) {
        compare++;
        for (int j = n - 1; j > i; j--) {
            compare++;
            compare++;
            if (list[j - 1] > list[j]) {
                swapping++;
                int t = list[j];
                list[j] = list[j - 1];
                list[j - 1] = t;
            }
        }
    }
    cout << "Кол-во сравнений: " << compare << endl;
    cout << "Кол-во перемещений: " << swapping << endl;
}

void print_list(int *list, int n) {
    for (int i = 0; i < n; i++) {
        cout << list[i];
        if (i != n - 1) {
            cout << " ";
        }
    }
}

int main() {
    setlocale(0, "");
    using clock_t = chrono::high_resolution_clock;
    using second_t = chrono::duration<double, std::ratio<1> >;
    chrono::time_point<clock_t> start;
    start = clock_t::now();

    const int n = 100000;
    int a[n];

    for (int i = 0; i < n; i++) {
        a[i] = rand();
    }

    //print_list(a, n);
    //cout << endl;
    bubble_sort(a, n);
    //print_list(a, n);

    cout << endl;
    cout << chrono::duration_cast<second_t>(clock_t::now() - start).count();

       return 0;
}
```

1.5. График зависимости теоретической и практической вычислительной сложности алгоритма.





1.6. Анализ результатов

Графики зависимости теоретической, практической вычислительной сложности алгоритма и времени выполнения программы от размера массива схожи. С ростом размера массива время выполнения программы увеличивается. График подтверждает вычисленную теоретическую квадратичную сложность алгоритма.

Задание 2. Оценить вычислительную сложность алгоритма простой сортировки в наихудшем и наилучшем случаях.

2.1. Сводная таблица результатов при применении метода к массиву, упорядоченному по возрастанию.

Таблица 2

n	T(n), сек	T_T=f(C+M)	$T_п$=Сф+Мф
100	0.0047063	10000	10002+0
1000	0.0132258	1000000	1000002+0
10000	0.1416	100000000	100000002+0
100000	10.4971	10000000000	10000000002+0
1000000	-	-	-

2.2. Сводная таблица результатов при применении метода к массиву, упорядоченному по убыванию.

Таблица 3

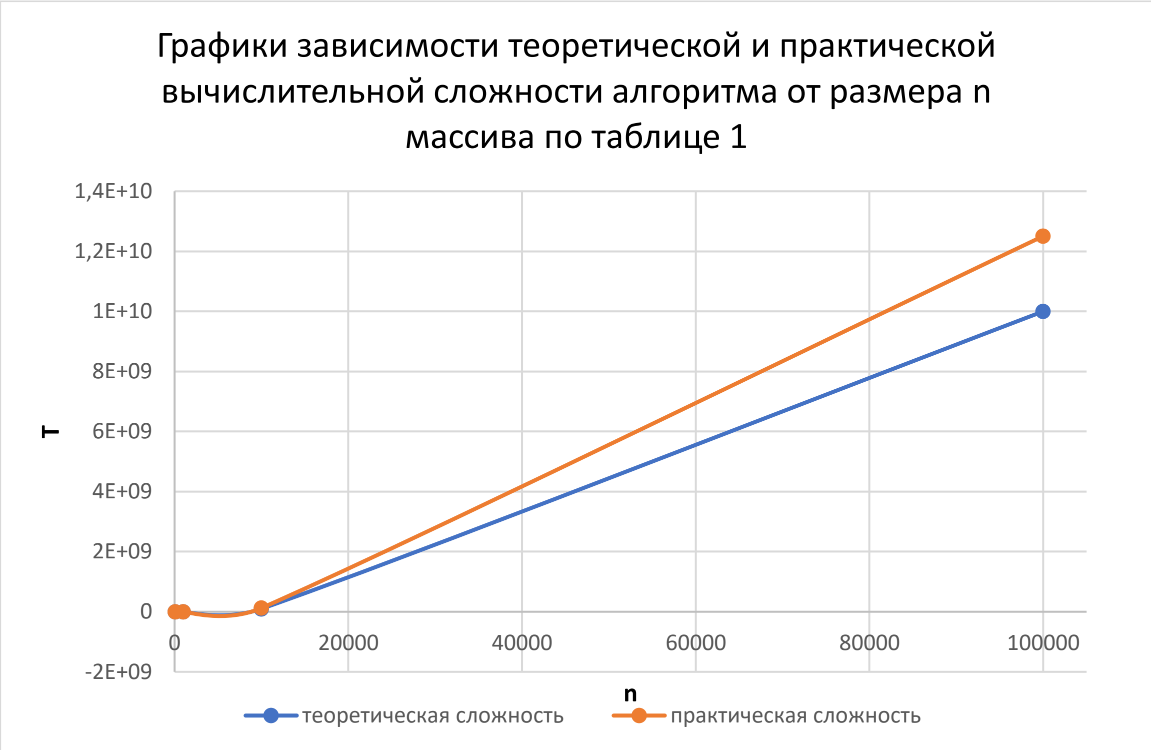
n	T(n), сек	T_T=f(C+M)=O(f(n))	$T_п$=Сф+Мф
100	0.0040259	10000	10002+4950
1000	0.0157962	1000000	1000002+499500
10000	0.425837	100000000	100000002+49995000
100000	21.9026	10000000000	10000000002+4999950000
1000000	-	-	-

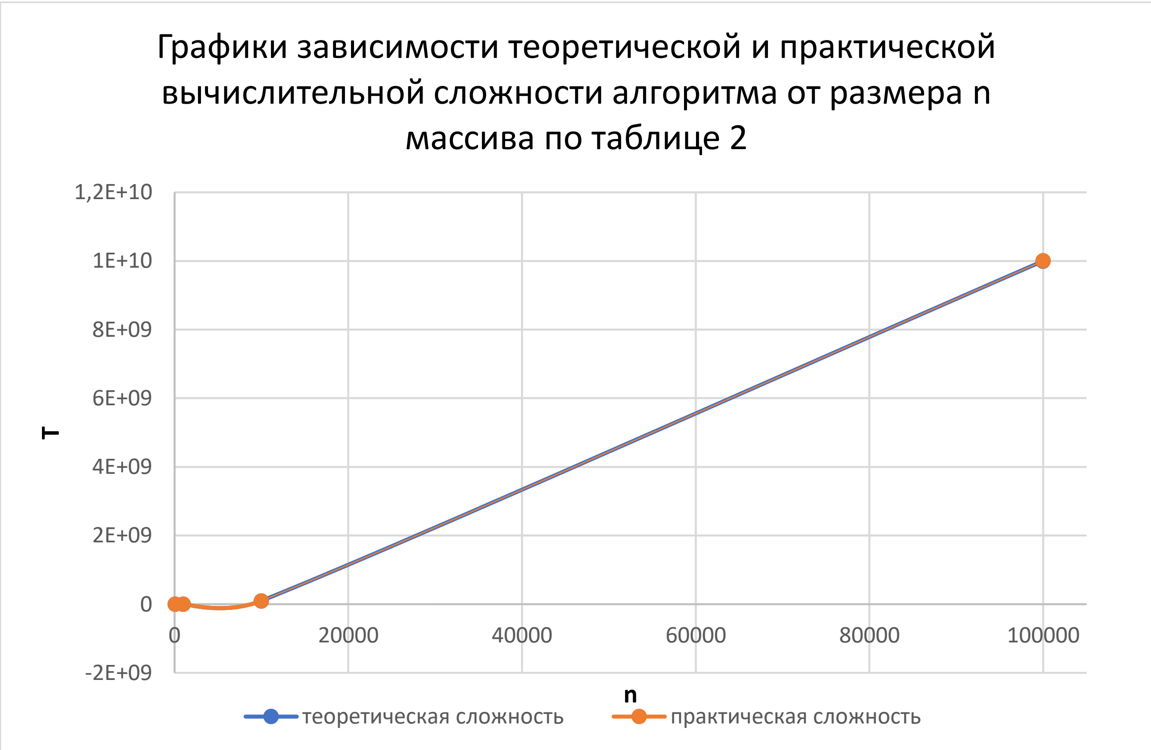
2.3. Код программы, которая выполняет тестирование алгоритма.

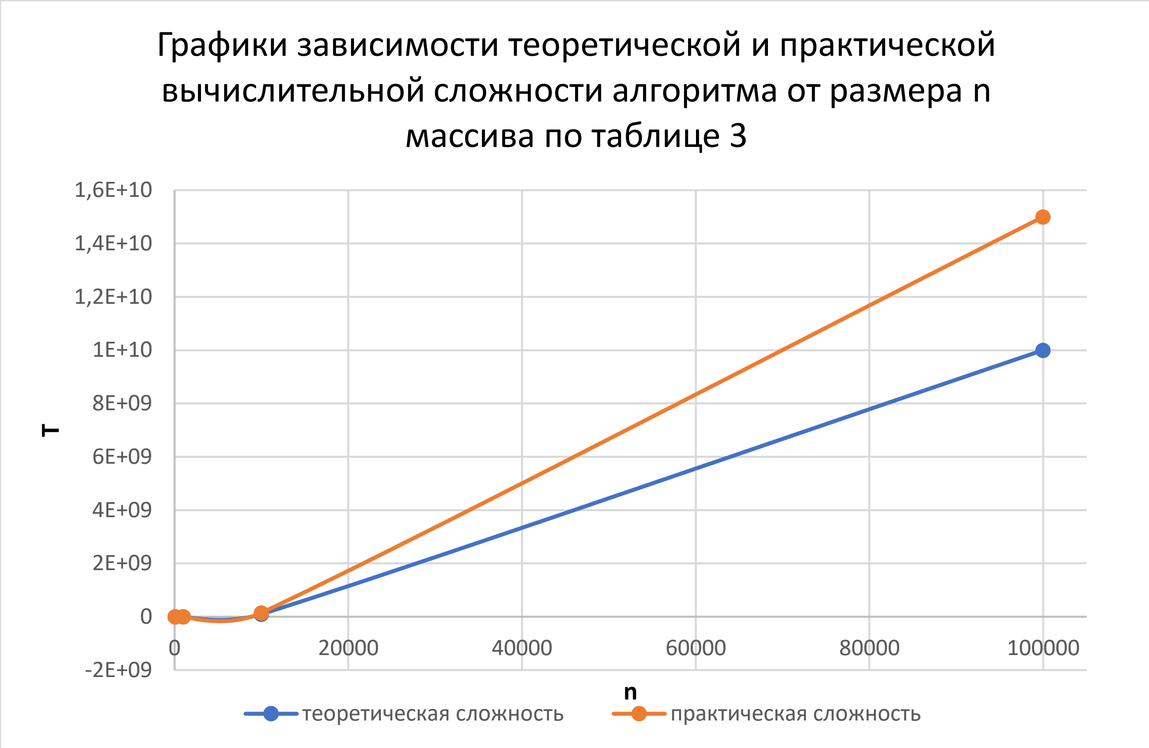
```
#include <iostream>
#include <chrono>
using namespace std;

void bubble_sort(int* list, int n) {
    long long int compare = 2, swapping = 0;
    for (int i = 0; i < n; i++) {
        compare++;
        for (int j = n - 1; j > i; j--) {
            compare++;
            compare++;
            if (list[j - 1] > list[j]) {
                swapping++;
                int t = list[j];
                list[j] = list[j - 1];
                list[j - 1] = t;
            }
        }
    }
    cout << "Кол-во сравнений: " << compare << endl;
    cout << "Кол-во перемещений: " << swapping << endl;
}

void print_list(int *list, int n) {
    for (int i = 0; i < n; i++) {
        cout << list[i];
        if (i != n - 1) {
            cout << " ";
```

```
        }
    }
}

int main() {
    setlocale(0, "");

    const int n = 100;
    int a[n];

    //С клавиатуры
    //for (int i = 0; i < n; i++) {
    //    cin >> a[i];
    //}

    //Рандомно
    //for (int i = 0; i < n; i++) {
    //    a[i] = rand();
    //}

    //По убыванию
    for (int i = 0; i < n; i++) {
        a[i] = n - i;
    }

    //По возрастанию
    //for (int i = 0; i < n; i++) {
    //    a[i] = i;
    //}

    //print_list(a, n);
    //cout << endl;

    using clock_t = chrono::high_resolution_clock;
    using second_t = chrono::duration<double, std::ratio<1> >;
    chrono::time_point<clock_t> start;
    start = clock_t::now();

    bubble_sort(a, n);

    cout << endl;
    cout << chrono::duration_cast<second_t>(clock_t::now() - start).count();
    //print_list(a, n);

       return 0;
}
```

2.4. График зависимости теоретической и практической вычислительной сложности алгоритма для трех рассмотренных случаев: по Таблица 1, по Таблица 2, по Таблица 3.







2.5. Ёмкостная сложность алгоритма

Ёмкостная сложность алгоритма определяет зависимость количества памяти от размера входных данных. В данном случае ёмкостная сложность равна O(n), так как зависимость линейная (используется один одномерный массив).

2.6. Анализ результатов

Время выполнения программы в лучшем случае меньше, чем в худшем. Особенно разница видна при больших размерах массивов.

В обоих случаях зависимость вычислительной сложности от размера массива является квадратичной.

Задание 3. Оценить эффективность алгоритмов простых сортировок

3.1. Алгоритм сортировки по методу простой вставки (Insertion sort).

```
void insertion_sort(int* list, int n) {
	for (int i = 1; i < n; i++) {
		for (int j = i; j > 0; j--) {
			if (list[j - 1] > list[j]) {
				swap(list[j], list[j - 1]);
			}
		}
	}
}
```

3.2. Определение функции роста времени выполнения сортировки методом простой вставки (Insertion sort) при увеличении объёма массива n.

Все данные алгоритма, для вывода функции роста представлены в таблице:

Номер оператора	Оператор	Время выполнения одного оператора	Кол-во выполнений оператора в строке
	1	2	3
1	for (int i = 1; i < n; i++) {	C1	n раз
2	for (int j = i; j > 0; j--) {	C2	$\sum_{i=0}^{n-1} t_i$
3	if (list[j - 1] > list[j]) {	C3	$\sum_{i=1}^{n-1} t_i$
4	swap(list[j], list[j - 1]);	C4	$\sum_{i=1}^{n-1} t_i$
5	}		
6	}		
7	}		

Выведем функцию роста для времени выполнения алгоритма:

$$\mathrm{T(n)} = \mathrm{C1} * \mathrm{n} + \mathrm{C2} * \sum_{i=0}^{n-1} t_i + \mathrm{C3} * \sum_{i=1}^{n-1} t_i + C4 * \sum_{i=1}^{n-1} t_i \qquad (1)$$

Определим порядок роста в лучшем случае, т.е. когда тело вложенного цикла не выполняется (массив уже отсортирован по рассматриваемому в алгоритме правилу). В наилучшем случае в сумме $\sum_{i=0}^{n-1} t_i$ значение t_i равно i. Тогда значение $\sum_{i=0}^{n-1} t_i = \frac{n(n-1)}{2} = \frac{n^2-n}{2}$. Оператор 3 выполнится $\frac{(n-1)(n-2)}{2} = \frac{n^2-2n+1}{2}$. C4 не выполнится вообще.

Подставим эти значения в формулу (1):

$$\mathrm{T(n)} = \mathrm{C1} * n + \mathrm{C2} * \frac{n^2 - n}{2} + \mathrm{C3} * \frac{n^2 - 2n + 1}{2} + C4 * 0 =$$

$$= \left(\frac{\mathrm{C2}}{2} + \frac{\mathrm{C3}}{2}\right) n^2 + \left(\mathrm{C1} - \frac{\mathrm{C2}}{2} - \mathrm{C3}\right) * \mathrm{n} + \frac{\mathrm{C3}}{2} = \mathrm{An^2} + \mathrm{Bn} + \mathrm{C}$$

Пренебрегаем константой C. Получаем $T(n)= An^2+Bn$. Функция n^2 имеет порядок роста выше, чем функция n. Таким образом, в выражении $T(n)= An^2+Bn$, доминирующей функцией является n^2, и она определяет порядок роста для алгоритма в лучшем случае. Т.е. $T(n)=\Theta(n^2)$.

Определим порядок роста *в худшем случае*, т.е. когда оператор 4 выполняется полное количество раз (массив упорядочен, но по правилу, противоположному тому, что рассматривает алгоритм).

В сумме $\sum_{i=0}^{n-1} t_i$ значение t_i равно i. Тогда значение $\sum_{i=0}^{n-1} t_i = \frac{n(n-1)}{2} = \frac{n^2-n}{2}$. Операторы 3 и 4 выполнятся $\frac{(n-1)(n-2)}{2} = \frac{n^2-2n+1}{2}$.

Подставим эти значения в формулу (1):

$$\mathrm{T(n)} = \mathrm{C1} * n + \mathrm{C2} * \frac{n^2 - n}{2} + \mathrm{C3} * \frac{n^2 - 2n + 1}{2} + C4 * \frac{n^2 - 2n + 1}{2} =$$
$$= \left(\frac{\mathrm{C2}}{2} + \frac{\mathrm{C3}}{2} + \frac{\mathrm{C4}}{2}\right) n^2 + \left(\mathrm{C1} - \frac{\mathrm{C2}}{2} - \mathrm{C3} - C4\right) * \mathrm{n} + \left(\frac{\mathrm{C3}}{2} + \frac{\mathrm{C4}}{2}\right)$$
$$= \mathrm{An}^2 + \mathrm{Bn} + \mathrm{C}$$

Пренебрегаем константой C. Получаем T(n)= An^2+Bn. Функция n^2 имеет порядок роста выше, чем функция n. Таким образом, в выражении T(n)= An^2+Bn, доминирующей функцией является n^2, и она определяет порядок роста для алгоритма в худшем случае. Т.е. $T(n)=\Theta(n^2)$.

3.3. Сводная таблица результатов выполнения сортировки по указанным объёмам.

Таблица 4

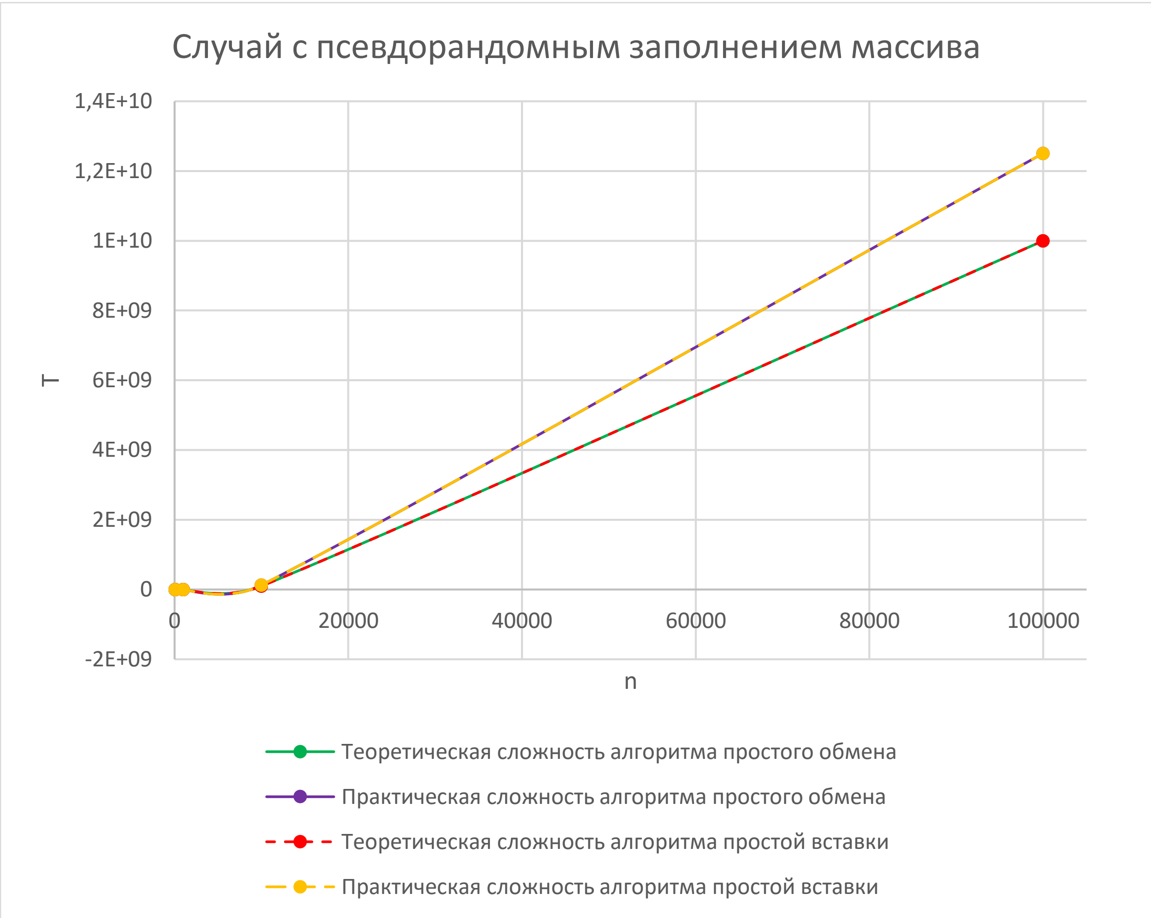
n	T(n), сек	T_T=f(C+M)	$T_п$=Сф+Мф
100	0.0043919	10000	10001+2360
1000	0.005957	1000000	1000001+248145
10000	0.193391	100000000	100000001+24952888
100000	13.0884	10000000000	10000000001+2506044511
1000000	-	-	-

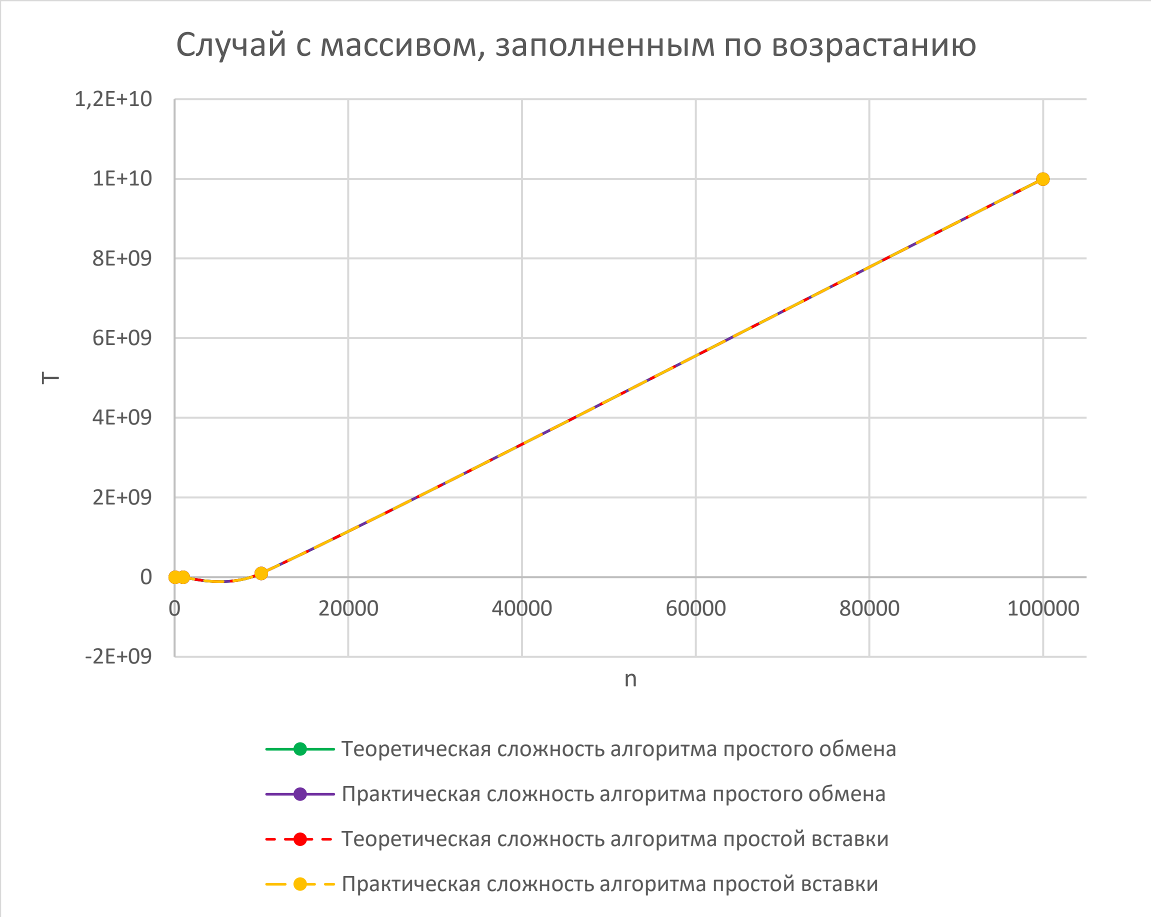
3.4. Код всей программы, доказывающей тестирование алгоритма на указанных в сводной таблице объёмах.

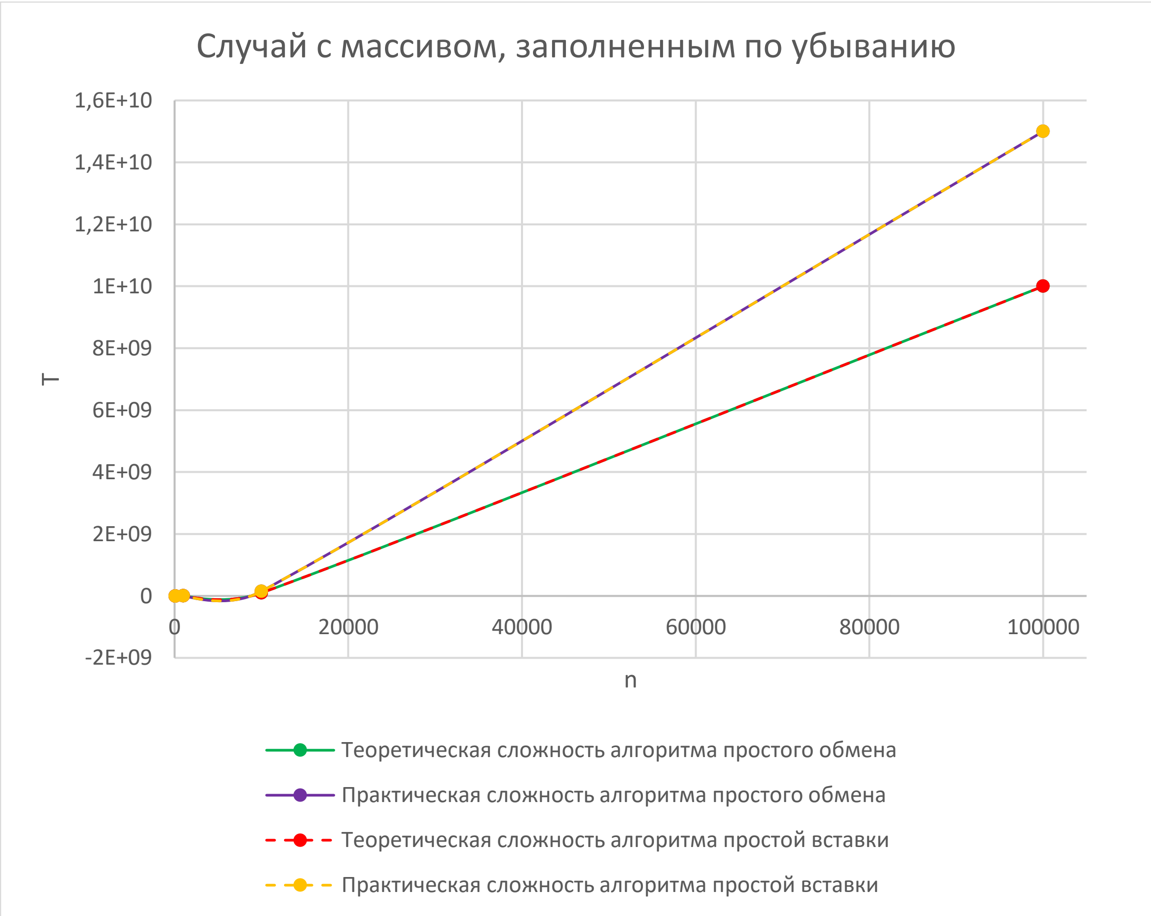
```
#include <iostream>
#include <chrono>
using namespace std;

void insertion_sort(int* list, int n) {
	long long int compare = 2, swapping = 0;
	for (int i = 1; i < n; i++) {
		compare++;
		for (int j = i; j > 0; j--) {
			compare++;
			compare++;
			if (list[j - 1] > list[j]) {
				swapping++;
				//swap(list[j], list[j - 1]);
				int t = list[j - 1];
				list[j - 1] = list[j];
				list[j] = t;
			}
		}
	}
	cout << "Кол-во сравнений: " << compare << endl;
	cout << "Кол-во перемещений: " << swapping << endl;
}

int main() {
	setlocale(0, "");
	const int n = 10000;
	int a[n];
```

```
	//Рандомно
	for (int i = 0; i < n; i++) {
		a[i] = rand();
	}

	using clock_t = chrono::high_resolution_clock;
	using second_t = chrono::duration<double, std::ratio<1> >;
	chrono::time_point<clock_t> start;
	start = clock_t::now();

	insertion_sort(a, n);

	cout << endl;
	cout << "Время выполнения программы: " << 
chrono::duration_cast<second_t>(clock_t::now() - start).count();

	return 0;
}
```

3.5. График зависимости теоретической и практической вычислительной сложности алгоритма для трех рассмотренных случаев: по Таблица 1, по Таблица 4




Случай с массивом, заполненным по возрастанию
1,2E+10
1E+10
8E+09
6E+09
4E+09
2E+09
0
-2E+09
T
0
20000
40000
60000
80000
100000
n
Теоретическая сложность алгоритма простого обмена
Практическая сложность алгоритма простого обмена
Теоретическая сложность алгоритма простой вставки
Практическая сложность алгоритма простой вставки


Случай с массивом, заполненным по убыванию
1,6E+10
1,4E+10
1,2E+10
1E+10
8E+09
6E+09
4E+09
2E+09
0
-2E+09
T
0
20000
40000
60000
80000
100000
n
Теоретическая сложность алгоритма простого обмена
Практическая сложность алгоритма простого обмена
Теоретическая сложность алгоритма простой вставки
Практическая сложность алгоритма простой вставки

3.6. Определение эффективности алгоритма.

Эффективность алгоритма — это свойство алгоритма, которое связано с вычислительными ресурсами, используемыми алгоритмом. Основными ресурсами являются время выполнения алгоритма (количество тривиальных шагов, необходимых для решения задачи) и пространство, используемое алгоритмом (определяется объёмом оперативной памяти или памяти на носителе данных).

3.7. Анализ результатов и определение, какой алгоритм эффективнее.

Ёмкостная сложность обоих алгоритмов является линейной, т.е. O(n). Вычислительная сложность также одинакова ($O(n^2)$). Однако из таблиц 1 и 4 видно, что на массивах небольших размеров алгоритм простой вставки (Insertion sort) выполняется быстрее алгоритма простого обмена (Пузырёк) (Exchange sort).

ВЫВОДЫ

В ходе выполнения задания получены практические навыки в:

1. Оценке зависимости времени выполнения алгоритма от размера массива;
2. Оценке вычислительной сложности алгоритма простой сортировки в наихудшем и наилучшем случаях;
3. Проведении эмпирической (практической) оценки вычислительной сложности алгоритма;
4. Определении емкостной сложности алгоритма от n;
5. Оценке эффективности алгоритмов простых сортировок;
6. Поиске наиболее эффективного алгоритма.

Тестирования всех операций пройдены успешно.

СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ

1. Процедурное программирование Языки программирования – Сайт lizochekk! [Электронный ресурс]: URL: https://lizochekk.jimdofree.com/программирование/
2. Документация по Microsoft C/C++ | Microsoft Docs – [Электронный ресурс] URL: https://docs.microsoft.com/ru-ru/cpp/?view=msvc-160
3. C++ – Типизированный язык программирования / Хабр – [Электронный ресурс] URL: https://habr.com/ru/hub/cpp/
4. Сортировка прямым обменом (метод «пузырька») – [Электронный ресурс] URL: https://prog-cpp.ru/sort-bubble/
5. В мире алгоритмов: Сортировка Вставками / Хабр – [Электронный ресурс] URL: https://habr.com/ru/post/181271/