МИНОБРНАУКИ РОСИИ
Федеральное государственное бюджетное образовательное учреждение высшего образования
«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Отчет по выполнению практического задания №1.2

Тема:

Эмпирический анализ сложности простых алгоритмов сортировки

Дисциплина: «Структуры и алгоритмы обработки данных»

Выполнил студент: Павлов Н.С.

Группа: ИКБО-30-23

Москва 2024

СОДЕРЖАНИЕ

ЦЕЛЬ РАБОТЫ

Актуализация знаний и приобретение практических умений по эмпирическому определению вычислительной сложности алгоритмов.

1 ЗАДАНИЕ №1

1.1 ФОРМУЛИРОВКА ЗАДАЧИ

Оценить эмпирически вычислительную сложность алгоритма простой сортировки на массиве, заполненном случайными числами (средний случай).

1.2 РАБОТА С АЛГОРИТМОМ

1.2.1 Реализация

Составим функцию простой сортировки Exchange sort (рисунок 1) и проведём тестирование программы при n = 10 (рисунок 2).

```
void bubble(int* x, long n, long long& sravn, long long& del)
{
    for (long i = 0; i < n - 1; i++) {
        sravn++;
        for (long j = i + 1; j < n; j++) {
            sravn++;
            if (x[i] > x[j]) {
                int k = x[i];
                x[i] = x[j];
                x[j] = k;
                del += 3;
            }
            sravn++;
        }
        sravn++;
    }
    sravn++;
}
```

Рисунок 1 – Реализация Exchange sort

```
Введите размер массива: 10
Исходный массив: 6 7 2 0 2 1 1 8 9 6
Отсортированный массив: 0 1 1 2 2 6 6 7 8 9
Количество сравнений: 109
Количество смещений: 48
Время работы: 0 ms
```

Рисунок 2 – Тестирование Exchange sort при n = 10

1.2.2 Определение вычислительной сложности алгоритма:

Определим вычислительную сложность алгоритма используя теоретический подход, т.е. выведем функцию роста T(n) отдельно для худшего и лучшего случаев.

Таблица 1 – Таблица подсчета кол-ва выполнений оператора в строке

Номер оператора	Оператор	Кол-во выполнений оператора в строке	
		В худшем случае	В лучшем случае
1	for (long i = 0; i < n - 1; i++) {	n+1	n+1
2	for (long j = i + 1; j < n; j++) {	(n+1)n	(n+1)n
3	if (x[i] > x[j]) {	n^2	n^2
4	swap(a[i],a[j]);	n^2	0
5	}		
6	}		
7	}		

Лучшим случаем для алгоритма будет отсортированный массив, так как в таком случае будет перестановки не совершаются, худшим случаем будет являться массив, отсортированный в порядке убывания.

Худший случай:

$$T(n) = n + 1 + (n + 1)n + n^2 + n^2 = 3n^2 + 2n + 1 \quad (1)$$

Лучший случай:

$$T(n) = n + 1 + (n + 1)n + n^2 = 2n^2 + 2n + 1 \quad (2)$$

И в том, и в том случае скорость роста - квадратичная. Значит, в целом, скорость роста – квадратичная.

1.2.3 Контрольные тесты

Проведем контрольные на массиве случайных чисел длинной n = 100, 1000, 10000, 100000 элементов с вычислением времени выполнения T(n) – (в миллисекундах) (рисунки 3-6).

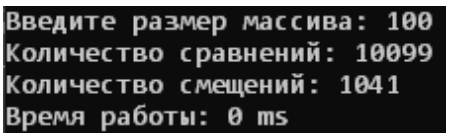

Рисунок 3 – Тестирование алгоритма при n = 100

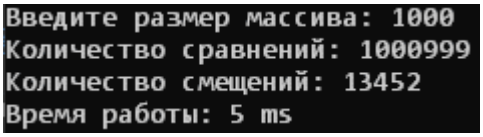

Рисунок 4 – Тестирование алгоритма при n = 1000

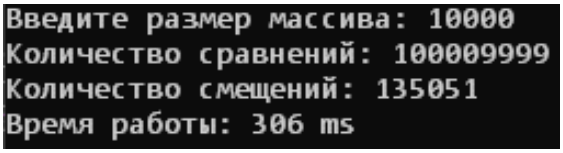

Рисунок 5 – Тестирование алгоритма при n = 10000

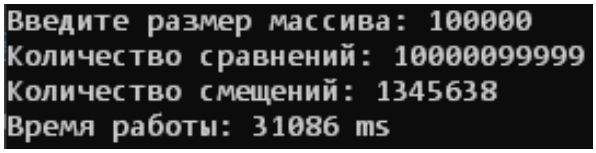

Рисунок 6 – Тестирование алгоритма при n = 100000

1.2.4 Оценка эмпирической сложности

Результаты эмпирической оценки вычислительной сложности алгоритма представлены в таблице 2.

Таблица 2 – Сводная таблица результатов

n	**T(n), мс**	**$T_п = C_п + M_п$**
100	0	11140
1000	5	1014451
10000	306	100145050
100000	31086	10001445637

1.2.5 График функции роста

На рисунке 8 представлен график зависимости вычислительной сложности алгоритма от размера массива.

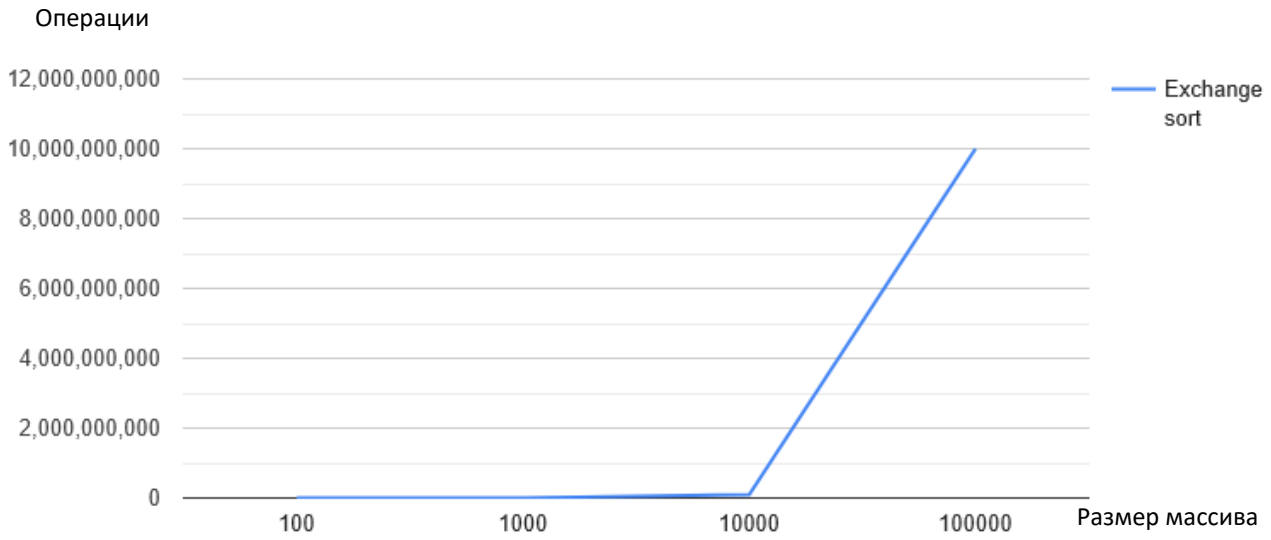

Рисунок 7 – График зависимости вычислительной сложности Exchange sort

1.2.6 Оценка емкостной сложности

Определим количество выделяемой памяти на каждый тип переменных с помощью функции sizeof() для конкретной системы (рис. 8).

```
int     4bytes
long    4bytes
int*    8bytes
```

Рисунок 8 – количество памяти, занимаемой переменными в конкретной системе

Перечислим все переменные, одновременно существующие в алгоритме, и размер в памяти, занимаемый ими (табл. 3).

Таблица 3 – Таблица определения размера переменных

Имя переменной	Тип переменной	Занимаемая память в байтах
i	long	4
j	long	4
n	long	4
x	int*	8
k	int	4
x[1]…x[n]	Массив int	4*n

Сложив значения третьего столбца, получаем, что емкостная сложность алгоритма: 4*4+8+4*n = 24+4n байт

1.3 ВЫВОД

Эмпирической оценкой сложности алгоритма Exchange sort была установлена квадратическая зависимость длительности выполнения алгоритма от размера входного массива.

2 ЗАДАНИЕ №2

2.1 ПОСТАНОВКА ЗАДАЧИ

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

2.2 РАБОТА С АЛГОРИТМОМ

2.2.1 Тестирование для наилучшего случая

Проведем контрольные тесты упорядоченном по возрастанию массиве длинной n = 100, 1000, 10000, 100000 элементов с вычислением времени выполнения T(n) – (в миллисекундах) (рисунки 9-12).

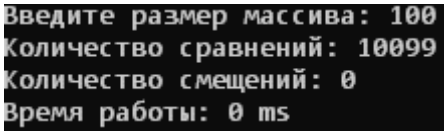

Рисунок 9 – Тестирование алгоритма при n = 100

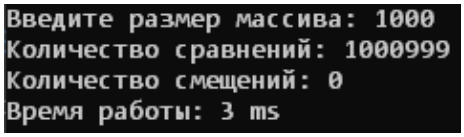

Рисунок 10 – Тестирование алгоритма при n = 1000

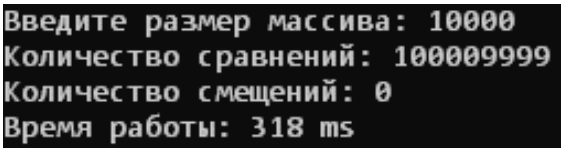

Рисунок 11 – Тестирование алгоритма при n = 10000

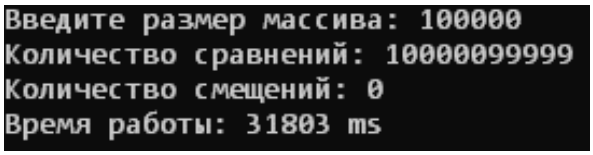

Рисунок 12 – Тестирование алгоритма при n = 100000

2.2.2 Оценка эмпирической сложности

Результаты эмпирической оценки вычислительной сложности алгоритма представлены в таблице 4.

Таблица 4 – Сводная таблица результатов

n	**T(n), мс**	$\mathbf{T_п = C_п + M_п}$
100	0	10099
1000	3	1000999
10000	318	100009999
100000	31803	10000099999

2.2.3 Тестирование для худшего случая

Проведем контрольные тесты упорядоченном по убыванию массиве длинной n = 100, 1000, 10000, 100000 элементов с вычислением времени выполнения T(n) – (в миллисекундах) (рисунки 13-16).

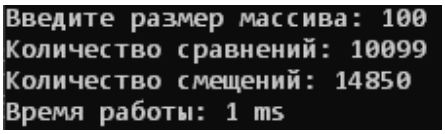

Рисунок 13 – Тестирование алгоритма при n = 100

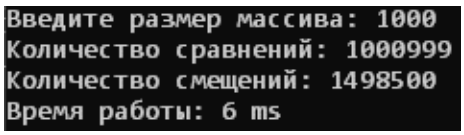

Рисунок 14 – Тестирование алгоритма при n = 1000

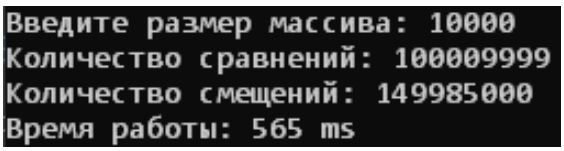

Рисунок 15 – Тестирование алгоритма при n = 10000

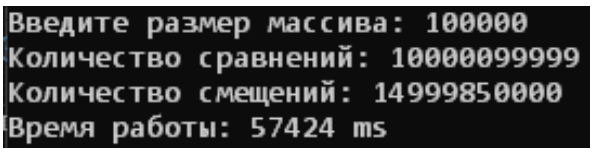

Рисунок 16 – Тестирование алгоритма при n = 100000

2.2.2 Оценка эмпирической сложности

Результаты эмпирической оценки вычислительной сложности алгоритма представлены в таблице 5.

Таблица 5 – Сводная таблица результатов

n	**T(n), мс**	**$T_п = C_п + M_п$**
100	1	24949
1000	6	2499499
10000	565	249994999
100000	57424	24999949999

2.3 ВЫВОД

Проанализировав полученные сводные таблицы, можно сделать вывод о том, что вычислительная сложность алгоритма Exchange sort зависит от исходной упорядоченности массива.

ЗАДАНИЕ №3

3.1 ПОСТАНОВКА ЗАДАЧИ

Сравнить эффективность алгоритмов простых сортировок.

3.2 РАБОТА С АЛГОРИТМОМ

3.2.1 Реализация

Составим функцию простой сортировки Insertion Sort (рисунок 17) и проведём тестирование программы при n = 10 (рисунок 18).

```
void insertion(int* x, long n, long long& sravn, long long& del)
{
    for (long i = 1; i < n; i++) {
        sravn++;
        int k = x[i];
        long j = i - 1;
        while (j >= 0 && x[j] > k) {
            sravn++;
            x[j + 1] = x[j];
            del++;
            j--;
        }
        sravn++;
        x[j + 1] = k;
        del++;
    }
    sravn++;
}
```

Рисунок 17 – Реализация Insertion Sort

```
Введите размер массива: 10
Исходный массив: 9 0 2 9 4 7 5 8 2 5
Отсортированный массив: 0 2 2 4 5 5 7 8 9 9
Количество сравнений: 40
Количество смещений: 30
Время работы: 0 ms
```

Рисунок 18 – Тестирование Insertion Sort при n = 10

3.2.2 Определение вычислительной сложности алгоритма:

Определим вычислительную сложность алгоритма используя теоретический подход, т.е. выведем функцию роста T(n) отдельно для худшего и лучшего случаев.

Таблица 6 – Таблица подсчета кол-ва выполнений оператора в строке

Номер оператора	Оператор	Кол-во выполнений оператора в строке	
		В худшем случае	В лучшем случае
1	for (long i = 1; i < n; i++) {	n	n
2	int k = x[i];	n-1	n-1
3	long j = i - 1;	n-1	n-1
4	while (j >= 0 && x[j] > k) {	(n-1)(n/2+1)	n-1
5	x[j + 1] = x[j];	(n-1)n/2	0
6	j--;		
7	}		
	x[j + 1] = k;	n-1	n-1
	}		

Лучшим случаем для алгоритма будет отсортированный массив, так как в таком случае будет перестановки не совершаются, худшим случаем будет являться массив, отсортированный в порядке убывания.

Худший случай:

$$T(n) = n + (n-1) + (n-1) + (n+1)\left(\frac{n}{2}+1\right) + \frac{n}{2}(n-1) + (n-1) \quad (3)$$
$$= n^2 + 5n - 2$$

Лучший случай:

$$T(n) = n + (n-1) + (n-1) + (n+1) + (n-1) = 5n - 2 \quad (4)$$

В лучшем случае скорость роста линейная, а в худшем квадратичная.

3.2.3 Тестирование и составление таблицы для среднего случая

Проведем контрольные тесты упорядоченном по убыванию массиве длинной n = 100, 1000, 10000, 100000 элементов с вычислением времени выполнения T(n) – (в миллисекундах) (рисунки 19-22).

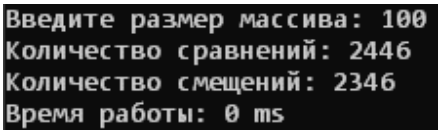

Рисунок 19 – Тестирование алгоритма при n = 100

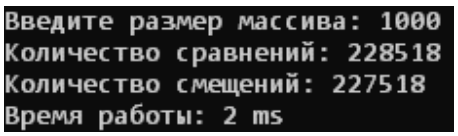

Рисунок 20 – Тестирование алгоритма при n = 1000

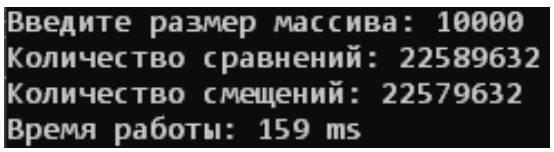

Рисунок 21 – Тестирование алгоритма при n = 10000

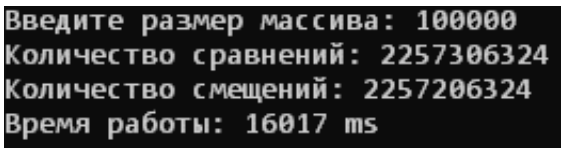

Рисунок 22 – Тестирование алгоритма при n = 100000

Результаты эмпирической оценки вычислительной сложности алгоритма представлены в таблице 7.

Таблица 7 – Сводная таблица результатов

n	**T(n), мс**	$\mathbf{T_п = C_п + M_п}$
100	0	4792
1000	2	456036
10000	159	45169264
100000	16017	4514512648

3.2.4 Тестирование и составление таблицы для лучшего случая

Проведем контрольные тесты упорядоченном по возрастанию массиве длинной n = 100, 1000, 10000, 100000 элементов с вычислением времени выполнения T(n) – (в миллисекундах) (рисунки 23-26).

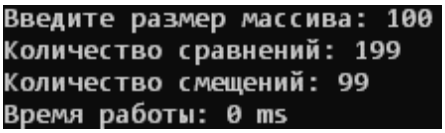

Рисунок 23 – Тестирование алгоритма при n = 100

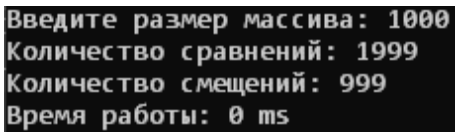

Рисунок 24 – Тестирование алгоритма при n = 1000

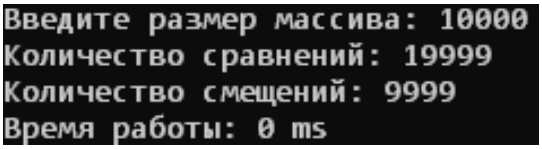

Рисунок 25 – Тестирование алгоритма при n = 10000

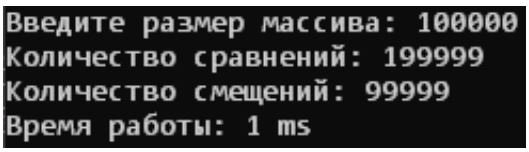

Рисунок 26 – Тестирование алгоритма при n = 100000

Результаты эмпирической оценки вычислительной сложности алгоритма представлены в таблице 8.

Таблица 8 – Сводная таблица результатов

n	**T(n), мс**	$\mathbf{T_п = C_п + M_п}$
100	0	298
1000	0	2998
10000	0	29998
100000	1	299998

3.2.5 Тестирование и составление таблицы для худшего случая

Проведем контрольные тесты упорядоченном по убыванию массиве длинной n = 100, 1000, 10000, 100000 элементов с вычислением времени выполнения T(n) – (в миллисекундах) (рисунки 27-30).

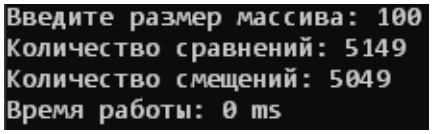

Рисунок 27 – Тестирование алгоритма при n = 100

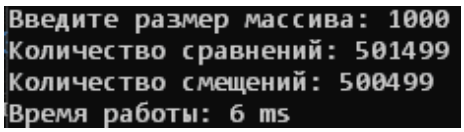

Рисунок 28 – Тестирование алгоритма при n = 1000

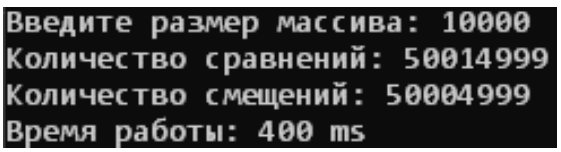

Рисунок 29 – Тестирование алгоритма при n = 10000

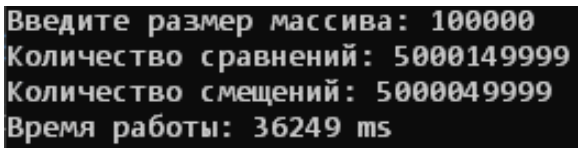

Рисунок 30 – Тестирование алгоритма при n = 100000

Результаты эмпирической оценки вычислительной сложности алгоритма представлены в таблице 9.

Таблица 9 – Сводная таблица результатов

n	**T(n), мс**	$\mathbf{T_п = C_п + M_п}$
100	0	10198
1000	6	1001998
10000	400	100019998
100000	36249	10000199998

3.2.6 Оценка емкостной сложности

Определим количество выделяемой памяти на каждый тип переменных с помощью функции sizeof() для конкретной системы (рис. 31).

```
int     4bytes
long    4bytes
int*    8bytes
```

Рисунок 31 – количество памяти, занимаемой переменными в конкретной системе

Перечислим все переменные, одновременно существующие в алгоритме, и размер в памяти, занимаемый ими (табл. 10).

Таблица 10 – Таблица определения размера переменных

Имя переменной	Тип переменной	Занимаемая память в байтах
i	long	4
j	long	4
n	long	4
x	int*	8
k	int	4
x[1]…x[n]	Массив int	4*n

Сложив значения третьего столбца, получаем, что емкостная сложность алгоритма: 4*4+8+4*n = 24+4n байт

3.2.7 Сравнение графиков

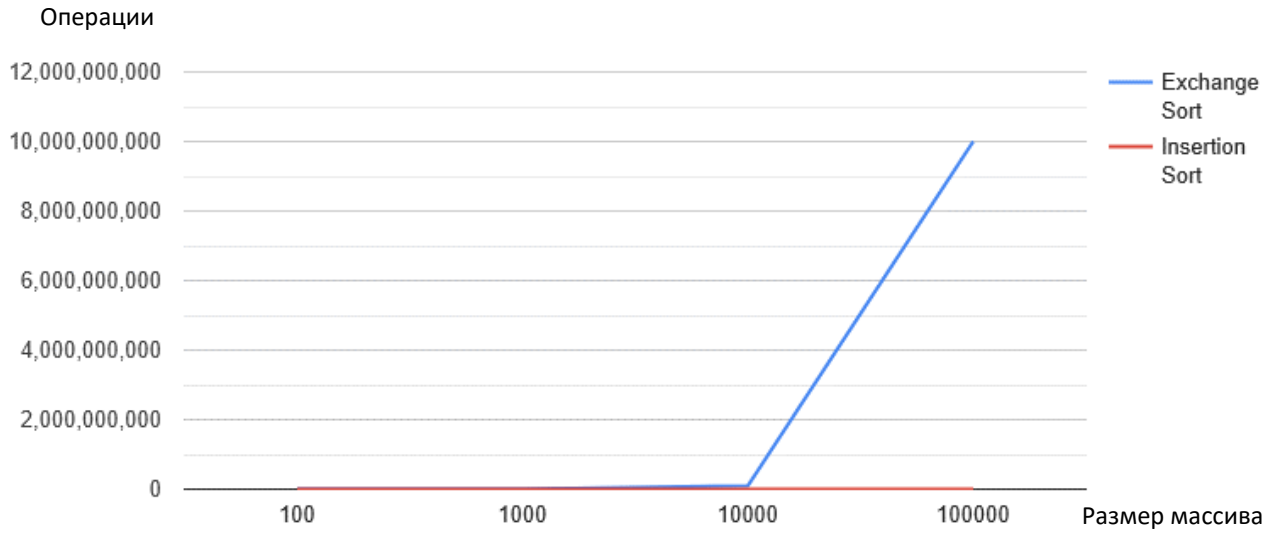

Рисунок 32 – График сравнения для лучшего случая

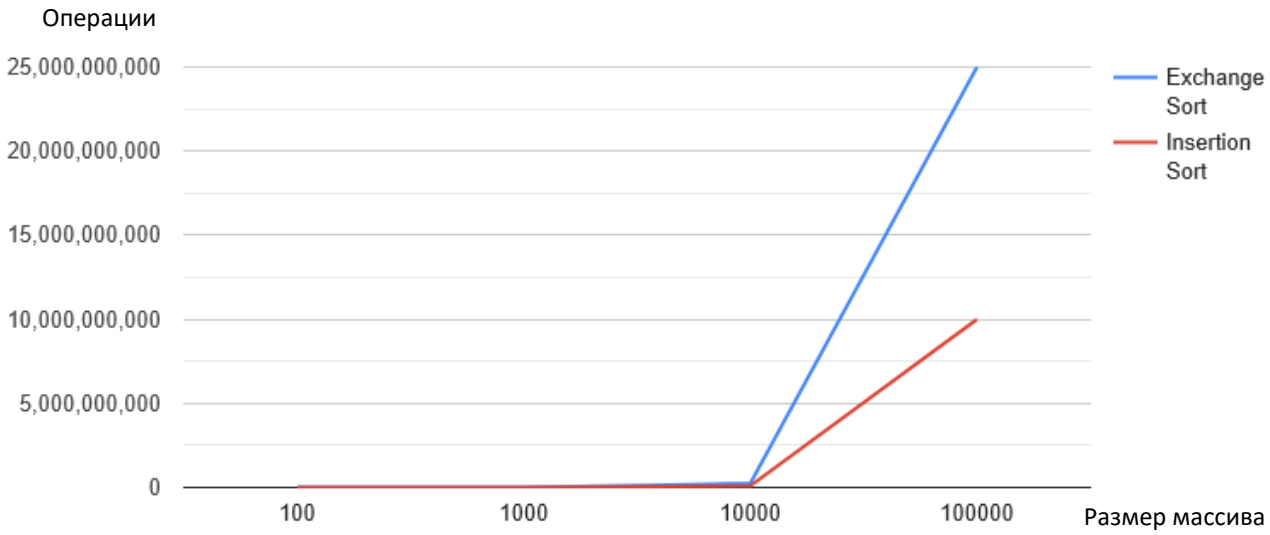

Рисунок 33 – График сравнения для худшего случая

3.3 ВЫВОД

Проанализировав полученные графики, можно сделать вывод о том, что сортировка вставками будет эффективнее, чем сортировка обменами и для худшего, и для лучшего случаев.

СПИСОК ИНФОРМАЦТОННЫХ ИСТОЧНИКОВ

1. Рысин, М. Л. Введение в структуры и алгоритмы обработки данных. Часть 1. Сложность алгоритмов. Сортировки. Линейные структуры данных. Поиск в таблице. : учеб. пособие / М. Л. Рысин, М. В. Сартаков, М. Б. Туманова ; МИРЭА – Российский технологический университет, 2022. – 109 с. – ISBN 978-5-7339-1612-5.

2. ГОСТ 19.701-90. Единая система программной документации. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения : межгосударственный стандарт : дата введения 1992-01-01 / Федеральное агентство по техническому регулированию. – Изд. официальное. – Москва : Стандартинформ, 2010. – 23 с.