МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное
учреждениевысшего образования
«МИРЭА – Российский технологический университет»

РТУ МИРЭА

ОТЧЕТ
ПО ПРАКТИЧЕСКОЙ РАБОТЕ №5

Алгоритмы простых сортировок. Анализ алгоритмов сортировок
по дисциплине
«СТРУКТУРЫ И АЛГОРИТМЫ ОБРАБОТКИ ДАННЫХ»

Выполнил студент Иолович Е.А.

группа ИНБО-03-22

Москва 2023

СОДЕРЖАНИЕ

0 ОТВЕТЫ НА ВОПРОСЫ

1) Какие сортировки называют простыми?

Это алгоритмы сортировки, которые являются простыми в понимании и реализации. Они обычно используются для небольших наборов данных или как первоначальные шаги более сложных алгоритмов сортировки.

2) Что означает понятие «внутренняя сортировка»?

Это алгоритм сортировки, который работает с данными, хранящимися в оперативной памяти компьютера. В отличие от внешней сортировки, которая работает с данными, хранящимися на диске или в других внешних устройствах, внутренняя сортировка может обрабатывать только небольшие объемы данных, ограниченные размером оперативной памяти компьютера. В этом случае сортировка происходит максимально быстро, так как скорость доступа к оперативной памяти значительно выше, чем к периферийным устройствам

3) Какие операции считаются основными при оценке сложности алгоритма сортировки?

При оценке сложности алгоритма сортировки основными операциями считаются сравнение элементов и перестановка элементов местами.

Однако не все алгоритмы сортировки используют только эти две операции. Некоторые алгоритмы могут использовать дополнительные операции, такие как вставка или копирование элементов, но в большинстве алгоритмов сортировки, сравнение и перестановка элементов являются главными операциями для достижения цели - получения отсортированного списка или массива.

4) Какие характеристики сложности алгоритма используются при оценке эффективности алгоритма?

Временная сложность — это количество времени, необходимое для

выполнения алгоритма в зависимости от размера входных данных. Обычно время работы алгоритма измеряется в терминах количества элементарных операций, таких как сравнение или перемещение элементов.

Пространственная сложность — это количество памяти, необходимое для выполнения алгоритма. Обычно пространственная сложность измеряется в терминах количества ячеек памяти, занимаемых алгоритмом.

Сложность по количеству операций сравнения — это количество операций сравнения, необходимых для выполнения алгоритма. Эта характеристика особенно важна для алгоритмов сортировки, где количество операций сравнения напрямую влияет на скорость работы алгоритма.

Сложность по количеству операций обмена — это количество операций обмена, необходимых для выполнения алгоритма. Эта характеристика также важна для алгоритмов сортировки, так как количество операций обмена может быть прямо связано с количеством времени, необходимого для выполнения алгоритма.

5) Какая вычислительная и емкостная сложность алгоритма: Простого обмена, Простой вставки, Простого выбора?

Простой обмен (Bubble Sort):

- Вычислительная сложность: $O(n^2)$
- Емкостная сложность: $O(1)$

Простая вставка (Insertion Sort):

- Вычислительная сложность: $O(n^2)$
- Емкостная сложность: $O(1)$

Простой выбор (Selection Sort):

- Вычислительная сложность: $O(n^2)$
- Емкостная сложность: $O(1)$

6) Какую роль в сортировке играет условие Айверсона?

Условие Айверсона (или инверсии) — сортировку можно прекратить

досрочно, если на каком-то этапе в ходе сравнения не будет сделано ни одной перестановки. Например, в отсортированном массиве условие Айверсона не выполняется.

Условие Айверсона играет важную роль в алгоритмах сортировки, потому что количество инверсий в массиве напрямую связано с временной сложностью алгоритма. Чем больше инверсий в массиве, тем дольше время выполнения сортировки.

7) Примените условие Айверсона к сортировке Простого обмена, заполните столбцы сводной таблицы для этого алгоритма с применением условия Айверсона. Сравните результаты с данными для сортировки Простого обмена без условия Айверсона. Сформулируйте выводы.

Сортировка Простого обмена с условием Айверсона работает значительно быстрее, чем без него. Это связано с тем, что в условии Айверсона мы пропускаем элементы, которые уже находятся на своем месте, что позволяет уменьшить количество операций сравнения.

8) Определите, каким алгоритмом, рассмотренным в этом задании, сортировался массив. Дан массив 5 6 1 2 3. Шаги выполнения сортировки: 1) 1 5 6 2 3 2) 1 2 5 6 3 3) 1 2 3 5 6.

На основании шагов выполнения сортировки можно сделать вывод, что массив был отсортирован алгоритмом Простой вставки (Insertion Sort).

9) Определите вычислительную теоретическую сложность алгоритма сортировки, рассмотренную в вопросе 8.

Теоретическая вычислительная сложность алгоритма Простой вставки (Insertion Sort) в худшем случае составляет $O(n^2)$, где n - количество элементов в массиве.

Средняя вычислительная сложность алгоритма Простой вставки также составляет $O(n^2)$, поскольку среднее количество операций, требуемых для

каждого элемента, также равно O(n).

1. ЗАДАНИЕ

1.1 Условие задания и требования

Разработать алгоритм сортировки одномерного целочисленного массива A[n] и реализовать его функцией, используя алгоритм согласно варианту, индивидуального задания - Простого обмена (Пузырек) (Exchange sort). Провести тестирование программы на исходном массиве, сформированном вводом T(n) с клавиатуры, т.е. доказать ее работоспособность.

Разработать функцию заполнения рабочего массива A с использованием генератора псевдослучайных чисел.

Индивидуальный вариант – 10(Алгоритм задания – Простой обмен, Пузырек).

1.2 Постановка задачи

Необходимо получить знания и практические навыки определения теоретической и практической сложности алгоритмов.

1.3 Алгоритм сортировки обменом на псевдокоде

```
exchange_SORT(a,n)
    for i = 0 to n -1 do
        for j =i+1 to n do
            if ar[i] > ar[j] do
                swap(ar[i], ar[j])
            end if
        end for
    end for
    end exchange_SORT
```

1.4 Привести процесс определения функции роста времени простого обмена

Проход по массиву начинается с первого элемента и заканчивается на предпоследнем. На каждом проходе сравниваются соседние элементы, и если они находятся в неправильном порядке, то они меняются местами.

После первого прохода на конце массива оказывается максимальный элемент. Далее проходы повторяются, но уже не до конца массива, а до предпоследнего элемента и т.д. В результате каждого прохода на конце массива окажется очередной максимальный элемент.

Проходы повторяются до тех пор, пока на очередном проходе не произойдет ни одного обмена элементами. Это означает, что массив уже отсортирован, и дополнительных проходов больше не нужно.

Функция роста времени алгоритма сортировки пузырьком зависит от двух факторов: размера массива и его состояния (отсортированного или неотсортированного).

В наихудшем случае, когда массив уже отсортирован в обратном порядке, каждый элемент массива должен быть перемещен в начало массива путем обмена с каждым из предшествующих элементов. В этом случае общее количество операций равно O(n^2).

Таким образом, функция роста времени в наихудшем случае для сортировки пузырьком составляет O(n^2).

1.5 Сводная таблица результатов выполнения сортировки простого обмена на случайно заполненном массиве для всех указанных объемов

Таблица 1 – Сводная таблица результатов (средний случай)

n	T, микросекунд	Тэп = f(C+M) - функция	Сф + Мф - количество
100	0,005752	$O(n^2)$	4950 + 1928
1000	0,013772	$O(n^2)$	499500 + 45277
10000	0,330815	$O(n^2)$	49995000 + 493419
100000	29,663914	$O(n^2)$	704982704 + 4952143

1.6 Графики зависимости теоретической (эмпирической) и практической (экспериментальной) вычислительной сложности алгоритма

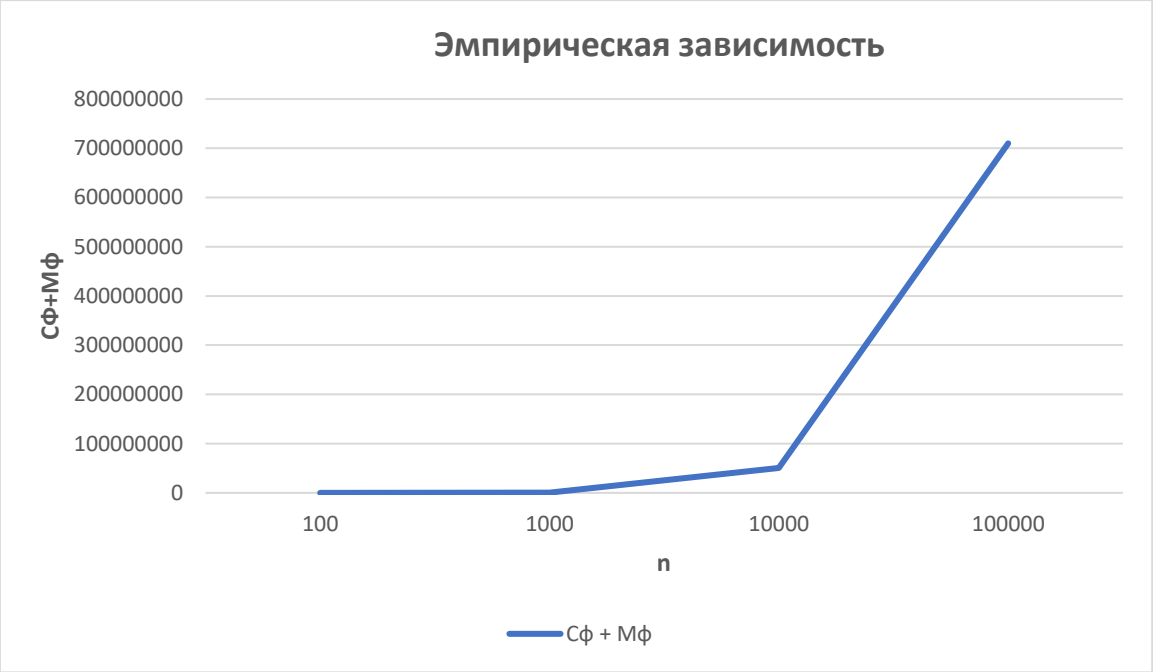

График 1 – Зависимость Сф + Мф от n

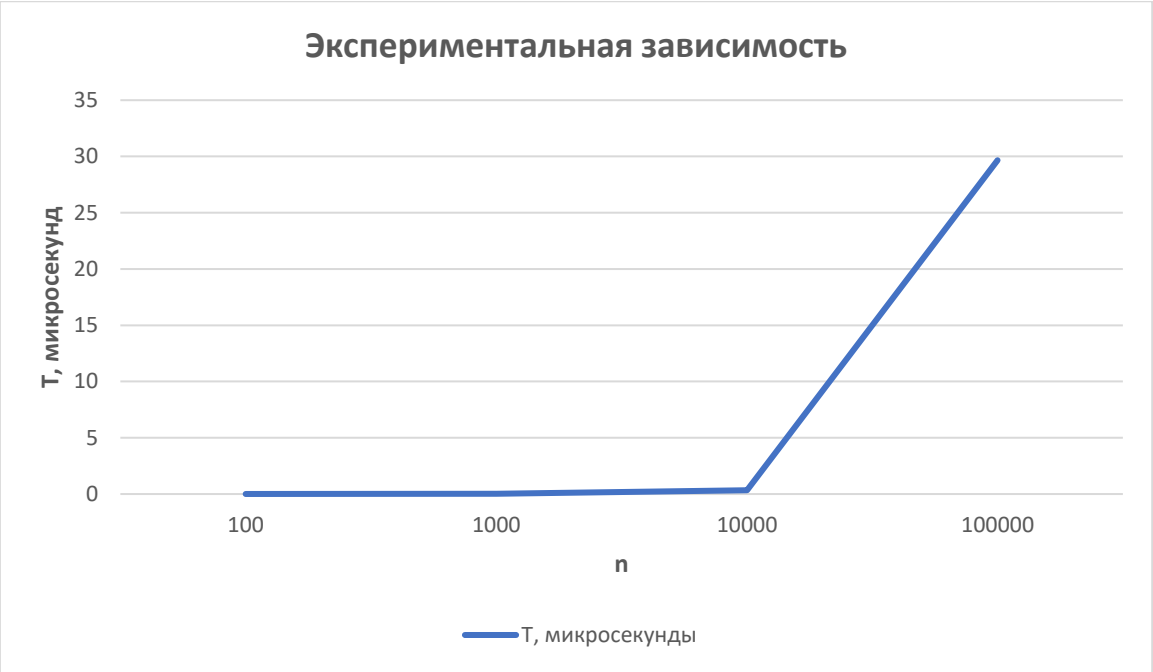

График 2 – Зависимость T от n

1.7 Программа

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

const int n = 1000;

void exchangeSort(int arr[], int n)
{
    int numCompares = 0;
    int numSwaps = 0;
    for (int i = 0; i < n - 1; i++)
    {
        for (int j = i + 1; j < n; j++)
        {
            numCompares++;

            if (arr[i] > arr[j])
            {
                numSwaps++;

                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
    }

    cout << endl;

    cout << "Количество сравнений: " << numCompares << endl;
    cout << "Количество перемещений: " << numSwaps << endl;

    cout << endl;
}
```

```
void fillArrayRandom(int arr[], int n)
{
    srand(time(NULL)); // Устанавливаем начальное значение генератора случайных
чисел.
    for (int i = 0; i < n; i++)
    {
        arr[i] = rand() % 100; // Генерируем случайное число от 0 до 99.
    }
}

int main()
{
    setlocale(LC_ALL, "Russian");
    int arr[n];
    fillArrayRandom(arr, n);
    cout << "Сгенерированный массив: ";
    for (int i = 0; i < n; i++)
    {
        cout << arr[i] << " ";
    }
    cout << endl;

    auto start = chrono::high_resolution_clock::now(); // Засекаем время.

    exchangeSort(arr, n);

    auto end = chrono::high_resolution_clock::now(); // Засекаем время после
выполнения сортировки.

    cout << "Отсортированный массив: ";
    for (int i = 0; i < n; i++)
    {
        cout << arr[i] << " ";
    }

    cout << endl;

    chrono::duration <double, micro> elapsed = end - start; // Вычисляем итоговое
время.
    cout << "Время выполнения: " << elapsed.count() << " микросекунд" << endl;

    return 0;
}
```

```
Сгенерированный массив: 75 49 6 87 19 77 14 14 59 37 64 29 67 88 7 67 14 97 73 65 35 16 23 91 95 7 95 68 67 28 28 77 53
2 55 33 26 57 89 14 35 43 97 15 56 53 80 1 28 19 50 58 43 31 6 52 16 35 45 10 7 17 15 14 18 79 38 87 78 47 39 47 0 32 58
 76 52 73 30 6 6 61 17 20 60 28 29 52 0 90 43 92 10 50 91 18 79 76 60 90

Сумма сравнений и перемещений: 7435
Количество сравнений: 4950
Количество перемещений: 2485

Отсортированный массив: 0 0 1 2 6 6 6 6 7 7 7 10 10 14 14 14 14 14 15 15 16 16 17 17 18 18 19 19 20 23 26 28 28 28 28 29
 29 30 31 32 33 35 35 35 37 38 39 43 43 43 45 47 47 49 50 50 52 52 52 53 53 55 56 57 58 58 59 60 60 61 64 65 67 67 67 68
 73 73 75 76 76 77 77 78 79 79 80 87 87 88 89 90 90 91 91 92 95 95 97 97
Затрачено: 0,005752 микросекунд
```

Рис 1 – Пример тестирования программы

1.8 Анализ результатов

Сортировка простая сортировка обменом имеет время выполнения $O(n^2)$, что доказали оба графика. Данная сортировка имеет смысл для массивов

небольших размеров, так как с увеличением объема данных время также начинает расти.

2. ЗАДАНИЕ

2.1 Сводная таблица результатов сортировки при применении к массиву, упорядоченному по возрастанию (лучший случай)

Таблица 2 – Сводная таблица результатов (лучший случай)

n	T, микросекунд	Тэп = f(C+M) - функция	Сф + Мф - количество
100	0,007725	O(n)	4950
1000	0,007192	O(n)	499500
10000	0,224281	O(n)	49995000
100000	13,179118	O(n)	704982704

2.2 Сводную таблицу результатов при применении метода к массиву, упорядоченному по убыванию (худший случай)

Таблица 3 – Сводная таблица результатов (худший случай)

n	T, микросекунд	Тэп = f(C+M) -функция	Сф + Мф - количество
100	0,006199	$O(n^2)$	4950 + 4909
1000	0,017075	$O(n^2)$	49950 + 494421
10000	0,345427	$O(n^2)$	49995000 + 49493352
100000	25,594601	$O(n^2)$	704982704 + 654988229

2.3 Программа

Лучший случай:

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

const int n = 100000;

void exchangeSort(int arr[], int size)
{
    int numCompares = 0;
    int numSwaps = 0;

    for (int i = 0; i < size - 1; i++)
```

```
    {
        for (int j = size - 1; j > i; j--)
        {
            numCompares++;

            if (arr[j] < arr[j - 1])
            {
                numSwaps++;

                int temp = arr[j];
                arr[j] = arr[j - 1];
                arr[j - 1] = temp;
            }
        }
    }

    cout << endl;

    cout << "Сумма сравнений и перемещений: " << numCompares + numSwaps << endl;
    cout << "Количество сравнений: " << numCompares << endl;
    cout << "Количество перемещений: " << numSwaps << endl;

    cout << endl;
}


void fillArrayRandom(int arr[], int n)
{
    srand(time(NULL)); // Устанавливаем начальное значение генератора случайных
чисел.
    for (int i = 0; i < n; i++)
    {
        arr[i] = rand() % 100; // Генерируем случайное число от 0 до 99.
    }
}

void sortArray(int arr[], int n) //Для сортировки по возрастанию отдельно вне
пузырька.
{
    for (int i = 0; i < n - 1; i++)
    {
        for (int j = i + 1; j < n; j++)
        {
            if (arr[i] > arr[j])
            {
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
    }
}


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

    int arr[n];

    fillArrayRandom(arr, n);

    //cout << "Сгенерированный массив: ";
    //for (int i = 0; i < n; i++)
    //{
```

```
    //    cout << arr[i] << " ";
    //}
    //cout << endl;

    sortArray(arr, n); //Сортирует тут по возрастанию до основной сортировки.

    auto start = chrono::steady_clock::now();

    exchangeSort(arr, n);

    auto end = chrono::steady_clock::now();

 /*   cout << "Отсортированный массив: ";
    for (int i = 0; i < n; i++)
    {
        cout << arr[i] << " ";
    }*/

    cout << endl;

    chrono::duration <float> t = chrono::duration_cast<chrono::microseconds>(end -
start);

    printf("Затрачено: %f микросекунд", t.count());

    return 0;
}
```

Худший случай:

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

const int n = 100000;

void exchangeSort(int arr[], int size)
{
    int numCompares = 0;
    int numSwaps = 0;

    for (int i = 0; i < size - 1; i++)
    {
        for (int j = size - 1; j > i; j--)
        {
            numCompares++;

            if (arr[j] < arr[j - 1])
            {
                numSwaps++;

                int temp = arr[j];
                arr[j] = arr[j - 1];
                arr[j - 1] = temp;
            }
        }
    }

    cout << endl;

    cout << "Сумма сравнений и перемещений: " << numCompares + numSwaps << endl;
    cout << "Количество сравнений: " << numCompares << endl;
```

```
    cout << "Количество перемещений: " << numSwaps << endl;

    cout << endl;
}


void fillArrayRandom(int arr[], int n)
{
    srand(time(NULL)); // Устанавливаем начальное значение генератора случайных
чисел.
    for (int i = 0; i < n; i++)
    {
        arr[i] = rand() % 100; // Генерируем случайное число от 0 до 99.
    }
}

void sortArray(int arr[], int n) //Сортировка по убыванию перед основной
сортировкой.
{
    for (int i = 0; i < n - 1; i++)
    {
        for (int j = 0; j < n - i - 1; j++)
        {

            if (arr[j] < arr[j + 1])
            {
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}



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

    int arr[n];

    fillArrayRandom(arr, n);

    //cout << "Сгенерированный массив: ";
    //for (int i = 0; i < n; i++)
    //{
    //    cout << arr[i] << " ";
    //}
    //cout << endl;

    sortArray(arr, n); //Сортирует тут по возрастанию до основной сортировки.

    auto start = chrono::steady_clock::now();

    exchangeSort(arr, n);

    auto end = chrono::steady_clock::now();

 /*   cout << "Отсортированный массив: ";
    for (int i = 0; i < n; i++)
    {
        cout << arr[i] << " ";
    }*/
```

```
    cout << endl;

    chrono::duration <float> t = chrono::duration_cast<chrono::microseconds>(end -
start);

    printf("Затрачено: %f микросекунд", t.count());

    return 0;
}
```

2.4 Графики зависимости теоретической и практической вычислительной сложности алгоритма случаев: по Таблица 1, по Таблица 2, по Таблица 3

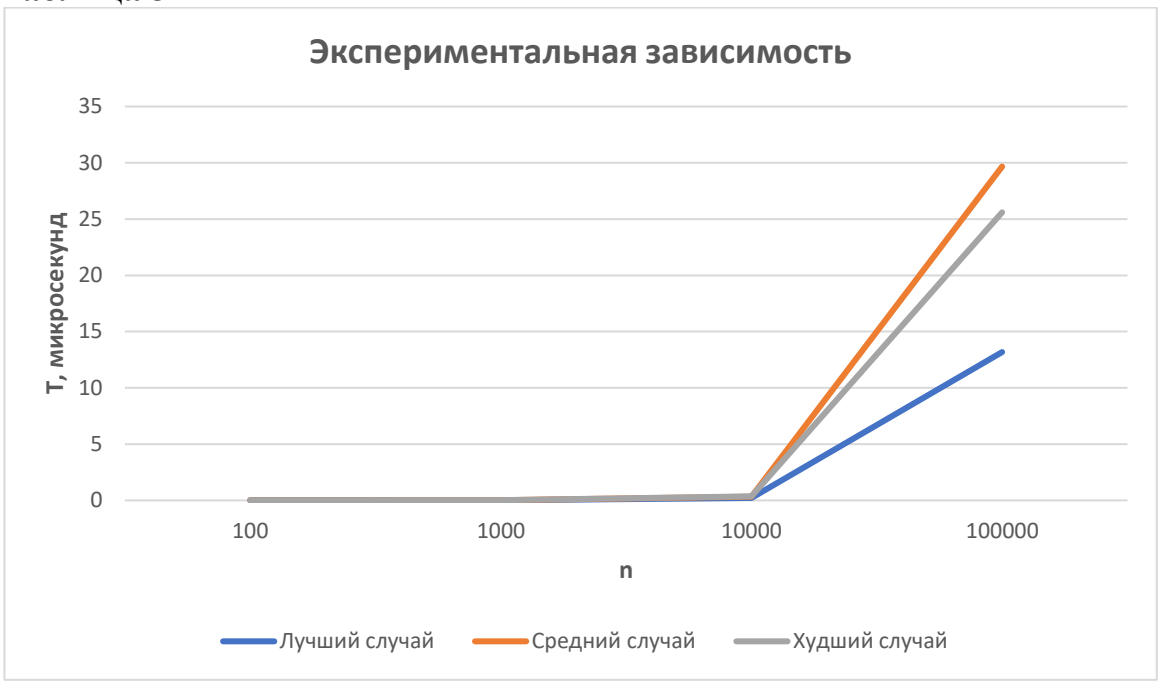

График 3 – Зависимость T от n

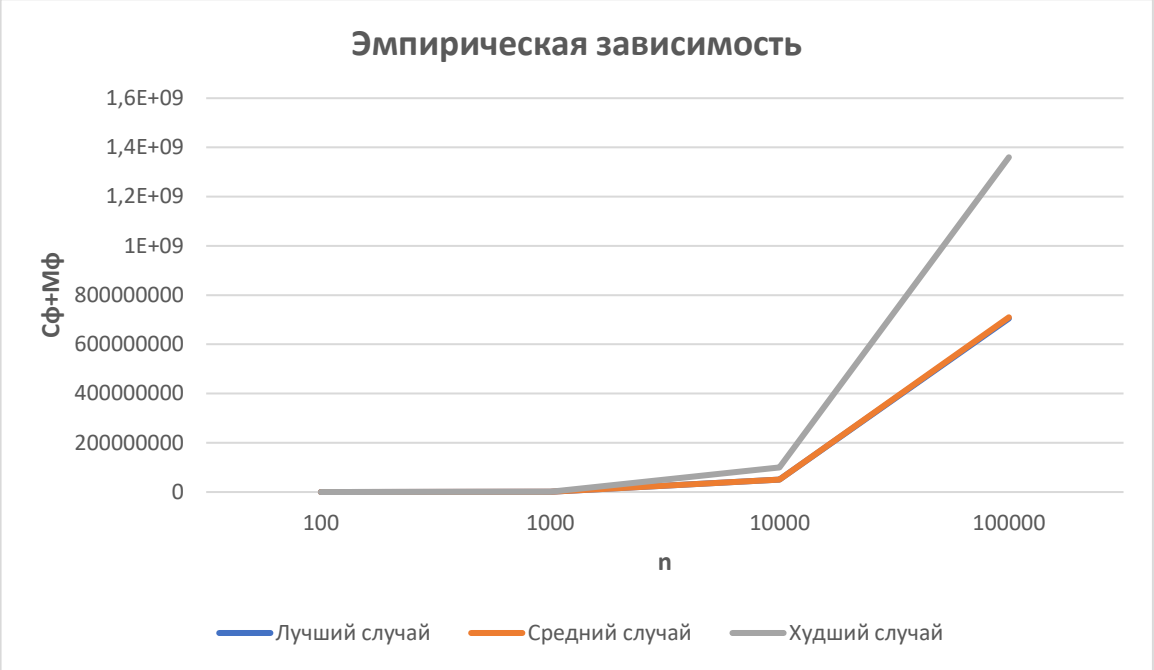

График 4 – Зависимость Сф + Мф от n

2.5 Анализ результатов и выводы

Простая сортировка пузырьком — это простой алгоритм сортировки, который работает за квадратичное время. Он не является самым эффективным алгоритмом сортировки, так как для сортировки массива из n элементов может потребоваться до n^2 операций.

Однако, простая сортировка пузырьком может быть полезна в случаях, когда массив имеет небольшой размер, или если массив уже почти отсортирован, в таких случаях сортировка пузырьком может работать быстрее, чем другие более сложные алгоритмы. Таким образом, можно сделать вывод, что простая сортировка пузырьком является не самым эффективным, но простым и интуитивно понятным алгоритмом сортировки, который может быть полезен в некоторых случаях.

3.ЗАДАНИЕ

3.1 Алгоритм сортировки простой вставкой на псевдокоде

```
for j = 2 to
        A.length do
        key = A[j]
        i = j-1
        while (int i >= 0 and A[i] > key) do
                A[i + 1] = A[i]
               i = i – 1
        end while
                A[i+1] = key
        end
```

3.2 Сводные таблицы результатов сортировки

Таблица 4 – Сводная таблица результатов (лучший случай)

n	**T, микросекунд**	**Тэп = f(C+M) - функция**	**Сф + Мф - количество**
100	0,004803	O(n)	99
1000	0,005116	O(n)	999
10000	0,006367	O(n)	9999
100000	0,007447	O(n)	99999

Таблица 5 – Сводная таблица результатов (средний случай)

n	**T, микросекунд**	**Тэп = f(C+M) -функция**	**Сф + Мф - количество**
100	0,004388	$O(n^2)$	99 + 2546
1000	0,007495	$O(n^2)$	999 + 259501
10000	0,178424	$O(n^2)$	9999 + 24910334
100000	13,194524	$O(n^2)$	99999 + 1820985250

Таблица 6 – Сводная таблица результатов (худший случай)

n	T, микросекунд	Тэп = f(C+M) -функция	Сф + Мф - количество
100	0,010912	$O(n^2)$	99 + 4896
1000	0,008710	$O(n^2)$	999 + 494552
10000	0,348362	$O(n^2)$	9999 + 49494284
100000	19,851906	$O(n^2)$	99999 + 654982385

3.3 Код всей программы, доказывающей тестирование алгоритма на указанных в сводной таблице объемах

```
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <chrono>

using namespace std;

const int n = 5;

void insertionSort(int arr[], int n)
{
    int numCompares = 0;
    int numSwaps = 0;

    for (int i = 1; i < n; i++)
    {
        int j = i;

        while (j > 0 && arr[j] < arr[j - 1])
        {
            int temp = arr[j];
            arr[j] = arr[j - 1];
            arr[j - 1] = temp;
            j--;

            numSwaps++;
        }

        numCompares++;
    }
    cout << endl;
    cout << "Сумма сравнений и перемещений: " << numCompares + numSwaps << endl;
    cout << "Количество сравнений: " << numCompares << endl;
    cout << "Количество перемещений: " << numSwaps << endl;
    cout << endl;

}

void fillArrayRandom(int arr[], int n)
{
    srand(time(NULL));
    for (int i = 0; i < n; i++)
```

```
    {
        arr[i] = rand() % 100;
    }
}

int main()
{
    setlocale(LC_ALL, "Russian");
    int arr[n];

    fillArrayRandom(arr, n);

    cout << "Сгенерированный массив: ";
    for (int i = 0; i < n; i++)
    {
        cout << arr[i] << " ";
    }

    cout << endl;
    cout << endl;

    auto start = chrono::steady_clock::now();

    insertionSort(arr, n);

    auto end = chrono::steady_clock::now();

    cout << "Отсортированный массив: ";
    for (int i = 0; i < n; i++)
    {
        cout << arr[i] << " ";
    }

    cout << endl;
    cout << endl;

    chrono::duration <float> t = chrono::duration_cast<chrono::microseconds>(end -
start);

    printf("Затрачено: %f микросекунд", t.count());

    cout << endl;

    return 0;

}
```

```
Сгенерированный массив: 1 63 7 62 14 65 60 13 37 99 63 65 49 5 97 12 93 8 8 10 45 31 85 94 7 92 46 18 71 44 64 14 84 43 5 23 48 68 98 0 93 27 28 50 60 75 0
73 95 51 57 78 90 80 1 97 71 69 55 74 43 49 19 64 38 52 33 58 74 20 27 26 99 78 46 22 82 45 88 77 88 59 4 46 88 30 37 79 9 92 55 88 21 62 74 97 73 81 36 90


Сумма сравнений и перемещений: 2218
Количество сравнений: 99
Количество перемещений: 2119

Отсортированный массив: 0 0 1 1 4 5 5 7 7 8 8 9 10 12 13 14 14 18 19 20 21 22 23 26 27 27 28 30 31 33 36 37 37 38 43 43 44 45 45 46 46 46 48 49 49 50 51 52
55 55 57 58 59 60 60 62 62 63 63 64 64 65 65 68 69 71 71 73 73 74 74 74 75 77 78 78 79 80 81 82 84 85 88 88 88 88 90 90 92 92 93 93 94 95 97 97 97 98 99 99

Затрачено: 0,009781 микросекунд
```

Рис 2 – Пример тестирования программы

3.5 Графики зависимости теоретической (эмпирической) и практической (экспериментальной) вычислительной сложности алгоритма

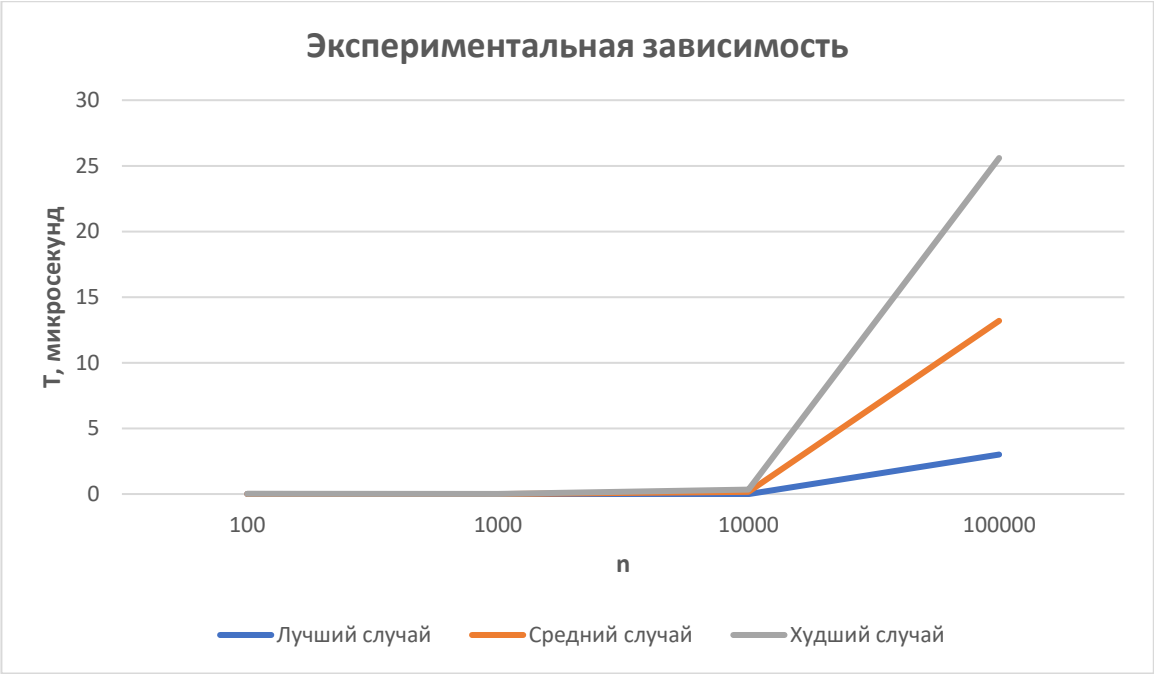

График 3 – Зависимость T от n

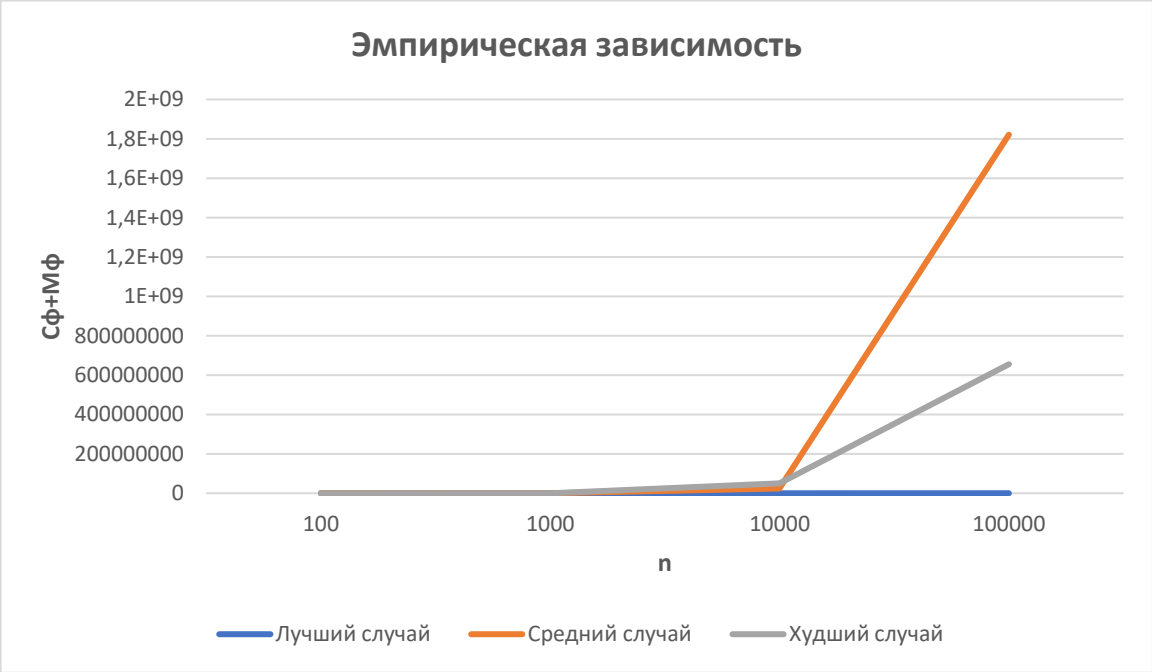

График 4 – Зависимость Сф + Мф от n

3.6 Анализ результатов и определение эффективности алгоритмов

Данная сортировка имеет временную сложность $O(n^2)$ как в худшем случае, так и в среднем. Её эмпирическая зависимость минимально в лучшем случае. Данная сортировка, как и сортировка простого обмена проста в понимании и проста в реализации поэтому пригодна для использования, однако, не в массивах с большим размером.

Выбор алгоритма сортировки зависит от многих факторов, таких как количество элементов в массиве, распределение значений элементов, доступная память, требования к стабильности сортировки и т.д. Однако, в общем случае, можно сказать следующее:

- Временная сложность алгоритма простого обмена (Bubble Sort) составляет $O(n^2)$, а алгоритма простой вставки (Insertion Sort) - также $O(n^2)$. Таким образом, оба алгоритма имеют одинаковую асимптотическую сложность, что означает, что они могут быть применены для сортировки небольших массивов.

- Однако, если массив уже отсортирован, то алгоритм простой вставки может отработать значительно быстрее, чем алгоритм простого обмена. Это связано с тем, что в отсортированном массиве будет меньше перестановок и сравнений.

- Если же массив имеет случайное распределение значений, то алгоритм простого обмена может показывать более быструю работу, чем алгоритм простой вставки.

Таким образом, нет однозначного ответа на вопрос, какой алгоритм сортировки выгоднее. Лучший выбор будет зависеть от конкретных условий, в которых используется алгоритм.

4. ВЫВОДЫ

При работе с простыми сортировками, такими как сортировка пузырьком и сортировка вставками, можно сделать следующие выводы:

Простые сортировки имеют квадратичную временную сложность $O(n^2)$, что означает, что они могут быть неэффективными для сортировки больших массивов.

В некоторых случаях, например, когда массив уже отсортирован или почти отсортирован, простые сортировки могут быть более эффективными, чем более сложные алгоритмы.

Простые сортировки являются основой для более сложных алгоритмов сортировки, таких как быстрая сортировка и сортировка слиянием.

5. ЛИТЕРАТУРА

1. Лекции по дисциплине «Структуры и алгоритмы обработки данных».
2. Habr: сайт. – URL: https://habr.com/ru/all/ (дата обращения: 10.04.2023).