МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
"МИРЭА - Российский технологический университет"

РТУ МИРЭА

Институт информационных технологий (ИТ)

ОТЧЕТ
ПО ПРАКТИЧЕСКОЙ РАБОТЕ № 4
Тема: «Эмпирический анализ алгоритмов сортировки»
по дисциплине
«СТРУКТУРЫ И АЛГОРИТМЫ ОБРАБОТКИ ДАННЫХ»

Выполнил студент Мызников Виталий Андреевич
Фамилия И.О.
Группы ИНБО-03-20
Номер группы

Москва 2021

Содержание

Цель работы

Приобретение практических навыков по определению:

– Эмпирическая оценка вычислительной сложности алгоритма;

– Оценка вычислительной сложности алгоритма при наилучшем/наихудшем случаях;

– Оценка эффективности алгоритмов простых сортировок.

Ход работы

Задание 1. Оценка зависимости времени от размера массива

Постановка задачи

Оценить зависимость времени выполнения алгоритма простой сортировки на массиве, заполненном случайными числами, для этого:

1. Составить программу сортировки (функцию) одномерного целочисленного массива A[n], используя алгоритм согласно варианту, индивидуального задания. Провести тестирование программы на исходном массиве, сформированном вводом с клавиатуры. Рабочий массив A сформировать с использованием генератора псевдослучайных чисел.
2. Провести контрольные прогоны программы для размеров массива $n = 100, 1000, 10000, 100000$ и 1000000 элементов с вычислением времени выполнения $T(n)$– $\left(\text{в}\,\frac{\text{миллисекундах}}{\text{секундах}}\right)$. Полученные результаты свести в сводную таблицу.
3. Построить график зависимости времени выполнения программы от размера массива.
4. Провести эмпирическую (практическую) оценку вычислительной сложности алгоритма, для чего предусмотреть в программе подсчет фактического количества операций сравнения $\text{С}_\text{ф}$ и количества операций перемещения $\text{М}_\text{ф}$. Полученные результаты $\text{С}_\text{ф} + \text{М}_\text{ф}$ вставить в сводную таблицу.
5. Построить в одной координатной плоскости графики зависимости теоретической $\text{Т}_\text{т} = f(\text{С} + \text{М}) = \text{O}\big(f(n)\big)$ и практической $\text{Т}_\text{п} = \left(\text{С}_\text{ф} + \text{М}_\text{ф}\right)$ вычислительной сложности алгоритма от размера n массива.
6. Определить емкостную сложность алгоритма от n.
7. Провести анализ полученных результатов. Сделать выводы о проделанной работе, основанные на полученных результатах.

Математическая модель решения

На каждом шаге алгоритма мы выбираем один из элементов входных данных и вставляем его на нужную позицию в уже отсортированном списке до тех пор, пока набор входных данных не будет исчерпан. Метод выбора очередного элемента из исходного массива произволен, но я выбрал метод, где элементы вставляются по порядку их появления во входном массиве.

Текст программы

Для написания текста алгоритма я использовал язык C++. Для реализации циклов я использовал оператор цикла for, для сравнения элементов я использовал условный оператор if, а также переменную temp для запоминания элемента, который вставляем в уже отсортированную часть массива (см. **Ошибка! Источник ссылки не найден.**).

```cpp
void InsertionSort(int* array, int size) // сортировка вставками
{
    int temp;
    for (int i = 1; i < size; i++)
    {
        temp = array[i];
        for (int j = i - 1; j >= 0; j--)
        {
            C++;
            if (array[j] < temp)
                break;
            M++;
            array[j + 1] = array[j];
            array[j] = temp;
        }
    }
};
```

Рисунок1 – текст написания функции отвечающей за сортировку вставками

```cpp
int main(){
    long long int M, C;
    setlocale(LC_ALL, "rus");
    cout << "Сортировка вставками:\n\n";
    int* A;
    for (int n = 100; n < 1000001; n *= 10)
    {
        C = M = 0;
        int* A= new int[n]; //создание одномерного динамического массива
        bad_arr(A, n);
        unsigned int start_time = clock(); //присваивание переменной начального времени
        InsertionSort(A, n); // вызов функции сортировки вставками
        unsigned int end_time = clock(); // конечное время
        unsigned int search_time = end_time - start_time; // итоговое время работа алгоритма
        cout << "Для n = " << n << " время составило " << search_time << "мс. Число сравнений C = " << C << " перемещаний M = " <<
        delete[] A; // отчистка памяти
    }
```

Рисунок2 -текст основной функции

```
#include <iostream>
#include <ctime>

using namespace std;
long long int M, C;

void random_arr(int* array, int size)//  массив состоящий из сгенерерированных чисел
{
    srand(0);
    for (int i = 0; i < size; i++)
        array[i] = rand() % 100; // заполняем массив случайными числами
};

void good_arr(int* array, int size) // для массива по возрастанию
{
    for (int i = 0; i < size; i++)
        array[i] = i;
};

void bad_arr(int* array, int size) // для массива по убыванию
{
    for (int i = 0; i < size; i++)
        array[i] = size - i;
};
```

Рисунок 3 -Использующиеся функции

Также для вывода массива на экран и заполнения его случайными элементами я написал функцию (см.Рисунок4).

Тестирование программы

По заданию нужно было засечь время в секундах, которое программа тратит на сортировку массивов разных размеров. Результаты предоставлены на таблице (см. Таблица 1).

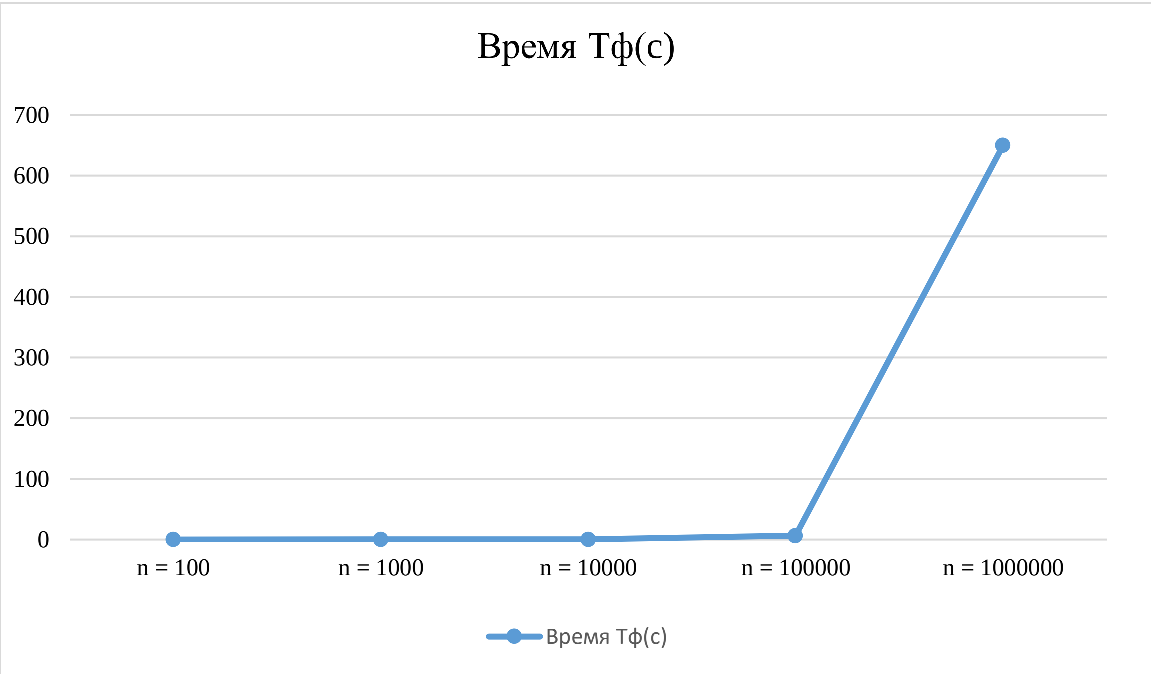
Таблица 1 – Сводная таблица результатов по времени

Размер массива	Скорость выполнения (с)
100	< 0,001
1000	0,001
10000	0,67
100000	6,476
1000000	650,176

Построим график зависимости времени выполнения сортировки от размера массива (см. График 1).

Int temp;	C1	1
For(int I = 1; I < n; i++)	C2	n - 1
Temp = array[i];	C3	n
For (int j = I – 1; j >= 0; j--)	C4	$\sum_j^n t_j$
If (array[j] < temp)	C5	n - 1
Break;	C6	n - 1
Array[j + 1] = array[j];	C7	$\sum_j^n t_j - 1$
Array[j] = temp;	C8	$\sum_j^n t_j - 1$

График 1 – График зависимости времени от размера массива



Оценка корректности и эффективности

Во внешний цикл мы входим n – 1 раз, а внутренний цикл открывается по-разному. В первый раз он откроется 1 раз, чтобы вставить 2 элемент в отсортированную часть массива (первый элемент я по умолчанию считаю уже отсортированным). А в последнюю итерацию внешнего цикла он откроется n – 1 раз). Тут четко прослеживается арифметическая прогрессия, так что воспользуемся ею (см. Формула 1).

$$\sum_{j}^{n} t_j = (1 + 2 + 3 \ldots . + n - 1) = \frac{(n - 1 + 1)}{2}(n - 1) = \frac{n^2 - n}{2} \qquad (1)$$

$$\sum_{j}^{n} t_j - 1 = \frac{(n-1)(n-1)}{2} = \frac{n^2 - 2n + 1}{2} \qquad (2)$$

T(n)=C1*1+C2*(n-1)/2+C3*(n)+C4*(n^2-n)/2 + C5 * (n-1) + C6 * (n-1) + C7 * (n^2-2n+1)/2 + C8 * (n^2-2n+1)/2 = $n^2 - n$

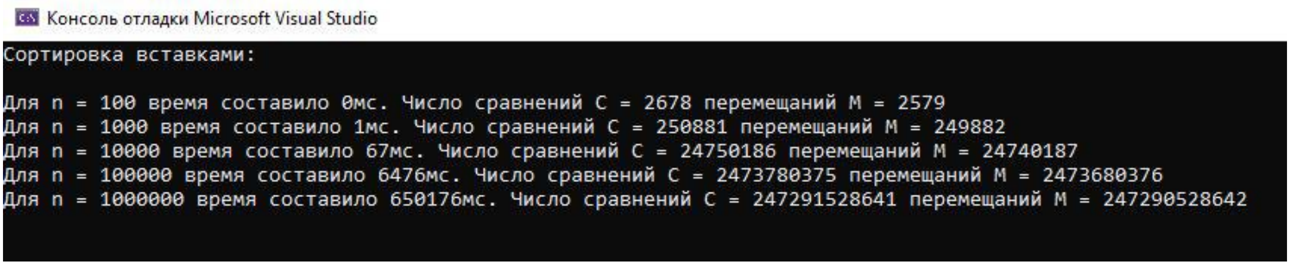
Функция n^2 имеет порядок роста выше, чем функция n. Таким образом, доминирующей функцией является $n^{2,}$ и она определяет порядок роста для алгоритма в худшем случае.

Теперь можно сделать сводную таблицу результатов по количеству операций (см. Таблица 2) и дать эмпирическую оценку сложности алгоритма.

Таблица 2 – Сводная таблица результатов по количеству операций

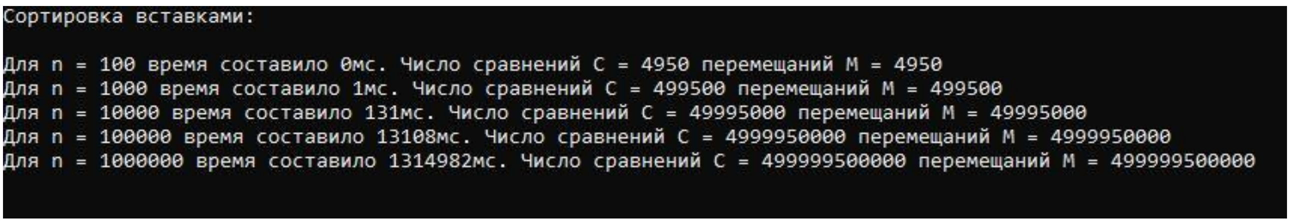
Размер массива	$T(n)$	$T_T(n) = f(C + M)$	$T_ф = C_ф + M_ф$
100	4950	9900	5257
1000	495000	990000	500763
10000	49500000	99000000	49490373
100000	4950000000	9900000000	4947460751
1000000	495000000000	990000000000	494582057283

При выполнении этих сортировок я не забыл сохранить результаты, предоставляю их ниже. (см. Рисунок 4), (см. Рисунок 5).



```
Консоль отладки Microsoft Visual Studio
Сортировка вставками:

Для n = 100 время составило 0мс. Число сравнений C = 2678 перемещаний M = 2579
Для n = 1000 время составило 1мс. Число сравнений C = 250881 перемещаний M = 249882
Для n = 10000 время составило 67мс. Число сравнений C = 24750186 перемещаний M = 24740187
Для n = 100000 время составило 6476мс. Число сравнений C = 2473780375 перемещаний M = 2473680376
Для n = 1000000 время составило 650176мс. Число сравнений C = 247291528641 перемещаний M = 247290528642
```

Рисунок 4 – Результаты работы программы средний случай



```
Сортировка вставками:

Для n = 100 время составило 0мс. Число сравнений C = 4950 перемещаний M = 4950
Для n = 1000 время составило 1мс. Число сравнений C = 499500 перемещаний M = 499500
Для n = 10000 время составило 131мс. Число сравнений C = 49995000 перемещаний M = 49995000
Для n = 100000 время составило 13108мс. Число сравнений C = 4999950000 перемещаний M = 4999950000
Для n = 1000000 время составило 1314982мс. Число сравнений C = 499999500000 перемещаний M = 499999500000
```

Рисунок 5 – Результаты работы программы худший случай

Емкостная сложность алгоритма

Емкостная сложность алгоритма зависит от количества элементов в массиве n и еще нужно место для переменной temp. Так что получается следующее:

$$M(n) = n + 1 \qquad (4)$$

Выводы

В процессе выполнения работы моя теоретическая оценка сложности не совпала с практическими результатами, так как для оценки теоретической сложности, я решил использовать худший вариант. А на практике применял обычный случайно сгенерированный массив.

Зависимость времени сортировки от размера массива квадратичная, а зависимость занимаемой памяти в процессе выполнения алгоритма от размера массива получилась линейной.

Задание 2. Оценка вычислительной сложности при разных случаях

Постановка задачи

Оценить вычислительную сложность алгоритма простой сортировки в наихудшем и наилучшем случаях, а для этого:

Провести дополнительные прогоны программы на массивах, отсортированных:

1. строго в убывающем порядке значений элементов, результаты представить в сводной таблице;
2. строго в возрастающем порядке значений элементов, результаты представить в сводной таблице;
3. провести анализ зависимости (или независимости) алгоритма сортировки от исходной упорядоченности массива.

Дополнительные функции

Для реализации выполнения алгоритма моей программой, где должны быть лучший и худший случаи я написал две дополнительные функции подпрограммы (см. Рисунок) и прогнал программу через них по очереди, сначала для лучшего случая (см. Рисунок), а потом для худшего случая (см. Рисунок 7) и результаты предоставил в таблице (см. Таблица 3).

```
Сортировка вставками:

Для n = 100 время составило 0мс. Число сравнений C = 4950 перемещаний M = 4950
Для n = 1000 время составило 1мс. Число сравнений C = 499500 перемещаний M = 499500
Для n = 10000 время составило 131мс. Число сравнений C = 49995000 перемещаний M = 49995000
Для n = 100000 время составило 13108мс. Число сравнений C = 4999950000 перемещаний M = 4999950000
Для n = 1000000 время составило 1314982мс. Число сравнений C = 499999500000 перемещаний M = 499999500000
```

Рисунок 1 – Результат работы для худшего случая

```
Консоль отладки Microsoft Visual Studio
Сортировка вставками:

Для n = 100 время составило 0мс. Число сравнений C = 99 перемещаний M = 0
Для n = 1000 время составило 0мс. Число сравнений C = 999 перемещаний M = 0
Для n = 10000 время составило 0мс. Число сравнений C = 9999 перемещаний M = 0
Для n = 100000 время составило 0мс. Число сравнений C = 99999 перемещаний M = 0
Для n = 1000000 время составило 2мс. Число сравнений C = 999999 перемещаний M = 0
```

Рисунок 6 – Результат работы для лучшего случая

Таблица 3 – Сводная таблица результатов работы для двух случаев

Размер массива	$T_х(n)$	$T_л(n)$
100	9900	99
1000	999000	999
10000	99990000	9999
100000	9999900000	99999
1000000	999999000000	999999

Анализ результатов

Из приведенных данных видно, что в худшем случае результаты остаются такими же, как ранее (см. Таблица 2), однако результаты работы программы для лучшего случая совсем другие. Это из-за того, что в процессе выполнения алгоритма не производятся перемещения, а также из-за того, что не выполняется условие о внутреннем цикле (см. **Ошибка! Источник ссылки не найден.**), из-за чего прекращается работа внутреннего цикла. Благодаря таким данным мы можем дать оценку теоретической сложности алгоритма для лучшего случая:

$$T_л(n) = n - 1 \qquad (5)$$

Программа просто зайдет во внешний цикл n – 1 раз, потом во внутренний, сделает сравнение и тут же из него выйдет. Поэтому получается, что сравнений будет n – 1раз (совпадает с количество срабатываний внешнего цикла).

Выводы

В процессе выполнения работы моя теоретическая оценка сложности совпала с практическими результатами.

Зависимость времени сортировки от размера массива квадратичная, а зависимость занимаемой памяти в процессе выполнения алгоритма от размера массива получилась линейной.

Задание 3. Сортировка выбором (Selection Sort)

Постановка задачи

1. Выполнить разработку алгоритма и программную реализацию Алгоритма задания 3 варианта.
2. Сформировать таблицу в соответствии с форматом Таблица 1 на тех же массивах, что и в задании 1.
3. Выполнить сравнительный анализом полученных результатов контрольных прогонов и построением соответствующих графиков.
4. Определить емкостную сложность алгоритма от n.

Математическая модель

В основе алгоритма сортировки выбором, как и во всех простых сортировках, лежит операция сравнения. Сравнивая каждый элемент с каждым, и в случае необходимости производя обмен, метод приводит последовательность к необходимому упорядоченному виду.

Основная идея такая: ищем минимальный элемент в массиве и ставим его в начало, после ищем в оставшемся неотсортированном массиве минимальный элемент и ставим в начало неотсортированного отрезка и так до конца.

Текст программы

Для написания текста алгоритма я использовал язык С++.

```
void InsertionSort(int* array, int size) // сортировка вставками
{
    int temp;
    for (int i = 1; i < size; i++)
    {
        temp = array[i];
        for (int j = i - 1; j >= 0; j--)
        {
            C++;
            if (array[j] < temp)
                break;
            M++;
            array[j + 1] = array[j];
            array[j] = temp;
        }
    }
};
```

Рисунок 8-Сортировка вставками

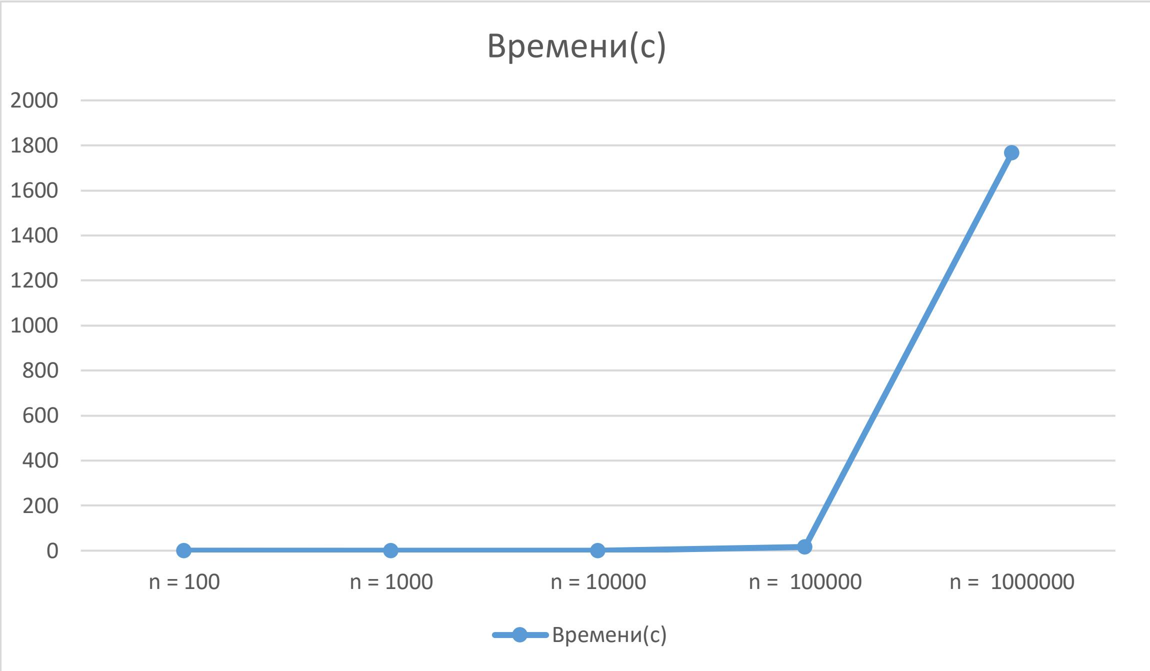
```cpp
void SortByChoise(int* array, int size) // сортировка выбором
{
    int i_min, temp;
    for (int i = 0; i < size - 1; i++)
    {
        i_min = i;
        for (int j = i + 1; j < size; j++)
        {
            C++;
            if (array[j] < array[i_min])
                i_min = j;
        }
        M++;
        temp = array[i];
        array[i] = array[i_min];
        array[i_min] = temp;

    }
}
```

Рисунок 9 -Сортировка массива выбором

По заданию нужно создать сводную таблицу и замерить время выполнения для разных размеров массива в секундах (см. Таблица 4).

Таблица 4 – Сводная таблица результатов тестирования по времени

Размер массива (n)	Время выполнения (с)
100	< 0,001
1000	0,002
10000	0,160
100000	16,785
1000000	1766,081



Оценка корректности и эффективности

Теперь надо дать теоретическую оценку сложности алгоритма. Во внешний цикл мы входим n - 1 раз, а во внутренний сначала n – 1, потом n – 2, и так до 1. Опять же арифметическая последовательность, поэтому получается:

$$\sum_{j}^{n} t_j = (n-1, n-2 \ldots\ldots 1) = \frac{n-1+1}{2}(n-1) = \frac{n^2-n}{2} \qquad (6)$$

$$\sum_{j=1}^{n-1} t_j = \frac{(n-1)(n-1)}{2} = \frac{n^2-2n+1}{2}$$

$T_T(n)$ = C1 * (n – 1) + C2 * (n) + C3 *$(n^2-n)/2$ + C4 * $(n^2-2n+1)/2$ + C5 * $(n^2-2n+1)/2$ + C6 * (n -1) + C7 * (n – 1) + C8 * (n – 1) = $\frac{n^2-n}{2}$ - 1

For (int I = 0; I < n -1; i++)	C1	n - 1
I_Min = I;	C2	n
For (int j = I + 1; j < n; j++)	C3	$\sum_j^n t_j$
If (arr[min] > arr[j])	C4	$\sum_j^{n-1} t_j$
Min = j;	C5	$\sum_j^{n-1} t_j$
Temp = arr[i];	C6	n - 1
Arr[i] = arr[i_min];	C7	n - 1
Arr[i_min] = temp;	C8	n - 1

Составим сводную таблицу (см.

Таблица 5):

Размер массива	$T(n)$	$T_T(n) = f(C+M)$	$T_ф = C_ф + M_ф$
100	4 950	5 049	5 049
1000	499 500	500 499	500 499
10000	49 995 000	50 004 999	50 004 999
100000	4 999 950 000	5 000 049 999	5 000 049 999
1000000	499 999 500 000	500 000 499 999	500 000 499 999

Таблица 5 – Сводная таблица сравнения T_T и $T_ф$

Выводы

В процессе выполнения работы моя теоретическая оценка сложности совпала с практическими результатами.

Зависимость времени сортировки от размера массива квадратичная, а зависимость занимаемой памяти в процессе выполнения алгоритма от размера массива получилась линейной.

Вывод

В ходе выполнения работы я приобрел практические навыки по определению:

- Эмпирическая оценка вычислительной сложности алгоритма;
- Оценка вычислительной сложности алгоритма при наилучшем/наихудшем случаях;
- Оценка эффективности алгоритмов простых сортировок.

Также я знаю алгоритмы простых сортировок, такие как: вставками, выбором. Их временную и емкостную сложности.