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

**РТУ МИРЭА**

---

# ОТЧЕТ
# ПО ПРАКТИЧЕСКОЙ РАБОТЕ №7

Алгоритмы поиска в таблице (массиве)
**по дисциплине**
«СТРУКТУРЫ И АЛГОРИТМЫ ОБРАБОТКИ ДАННЫХ»


Выполнил студент Иолович Е.А.

группа ИНБО-03-22


**Москва 2023**

**СОДЕРЖАНИЕ**

# 1 ЗАДАНИЕ 1

## 1.1 Постановка задачи

Разработать программу поиска записи по ключу в таблице записей с применение двух алгоритмов линейного поиска

1. Таблица содержит записи, структура которых определена вариантом. Ключи уникальны в пределах таблицы.

2. Разработать функцию линейного поиска (метод грубой силы).

3. Разработать функцию поиска с барьером.

4. Провести практическую оценку времени выполнения алгоритмов на таблицах объемом 100, 1000, 10 000 записей.

5. Составить таблицу с указанием: времени выполнения алгоритма, его фактическую и теоретическую вычислительную сложность.

6. Сделать выводы об эффективности алгоритмов.

Номер индивидуального варианта 10%16=10

| **№** | **Алгоритмы поиска** | **Структура записи файла (ключ – подчеркнутое поле)** |
|---|---|---|
| 2 | Интерполяционный поиск | Страхование авто средства: <u>регистрационный номер</u> – шестизначное число, название страховой компании |

## 1.2 Описание подхода к решению

Структура записи в таблице будет определяться с помощью пользовательского типа данных AutoInsurance.

Заполнение массива типа AutoInsurance* будет происходить с помощью функциии generateAutoInsuranceData. Параметр функции – целочисленная переменная – размер создаваемого массива. Возвращаемое значение – типа void.

Функция нахождения элемента в массиве с помощью Линейного поиска LinearSearch. Параметры – массив пользовательского типа данных AutoInsurance (итоговая таблица), целочисленная переменная – длина массива, целочисленная переменная – искомое значение (key).

Функция нахождения элемента в массиве с помощью Линейного поиска с барьером LinearSearchWithBarrier. Параметры – массив пользовательского типа данных AutoInsurance (итоговая таблица), целочисленная переменная – длина массива, целочисленная переменная – искомое значение (key).

Для подсчета времени используем библиотеку chrono.

### 1.3 Структура записи таблицы

```
struct AutoInsurance // структура для хранения данных об авто страховании.
{
    int regNumber; // регистрационный номер автомобиля.
    string insuranceCompany; // название страховой компании.
};
```

### 1.4 Определение размера таблицы в байтах

Структура AutoInsurance содержит два поля - regNumber типа int и insuranceCompany типа string. Размер типа int зависит от платформы и может быть разным, но обычно это 4 байта. Размер типа string также зависит от платформы, но обычно он составляет 24 байта на 64-битной платформе (8 байтов для указателя на массив символов char, 8 байтов для размера массива и 8 байтов для флагов). Таким образом, размер одной записи в таблице будет равен 4 + 24 = 28 байтам.

### 1.5 Коды функций, реализующих алгоритмы

**Линейный поиск:**

```
// функция линейного поиска на массиве.
int linearSearch(AutoInsurance arr[], int n, int key)
{
    for (int i = 0; i < n; i++)
    {
        if (arr[i].regNumber == key)
        {
            return i; // возвращаем индекс найденного элемента.
        }
    }
    return -1; // возвращаем -1, если элемент не найден.
}
```

### Линейный поиск с барьером:

```
// функция линейного поиска с барьером на массиве.
int linearSearchWithBarrier(AutoInsurance arr[], int n, int key)
{
    AutoInsurance last = arr[n - 1]; // сохраняем последний элемент массива.
    arr[n - 1].regNumber = key; // устанавливаем барьер.

    int i = 0;
    while (arr[i].regNumber != key)
    {
        i++;
    }

    arr[n - 1] = last; // восстанавливаем последний элемент массива.

    if (i < n - 1 || arr[n - 1].regNumber == key) //индекс находится в переделах
массива или если последний элемент равен key.
    {
        return i; // возвращаем индекс найденного элемента.
    }
    else
    {
        return -1; // возвращаем -1, если элемент не найден.
    }
}
```

### 1.6 Теоретическая сложность алгоритмов

- **Лучший случай:** O(1) - когда искомый элемент находится в первой позиции или ячейке.

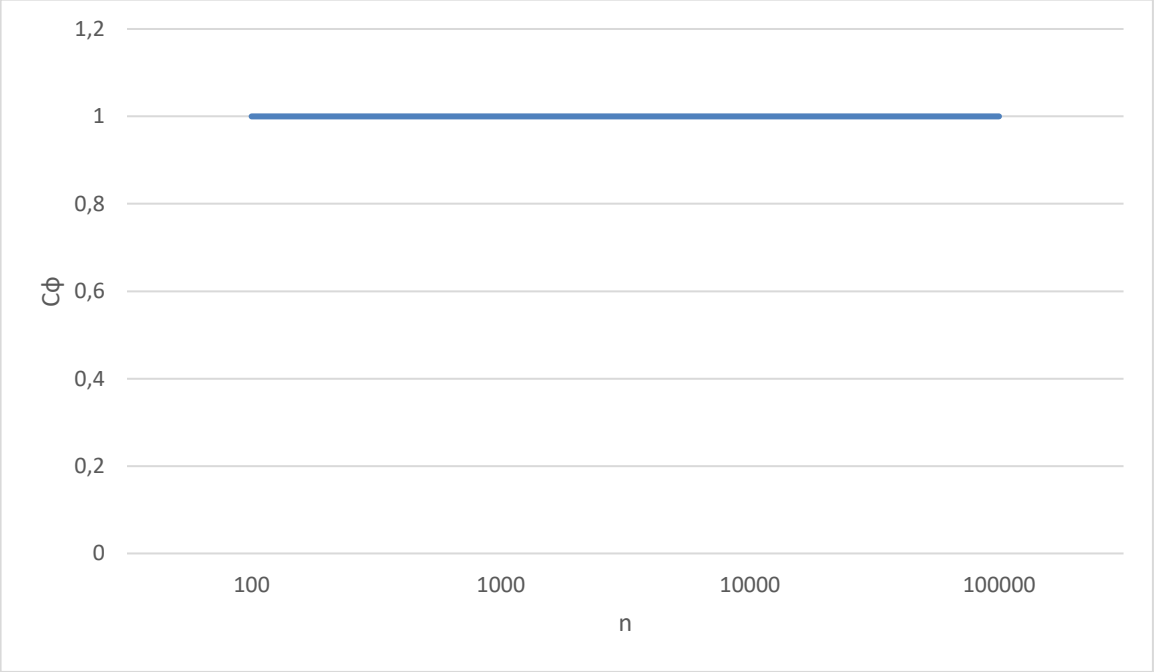
- **Худший случай:** O(n) - когда искомый элемент находится в последней позиции или ячейке, либо его нет вообще, в таком случае алгоритм пройдет по всем элементам списка, чтобы найти его или подтвердить его отсутствие.

- **Средний случай:** O(n/2) = O(n) - когда искомый элемент находится примерно в середине списка, но алгоритм может пройти половину списка, чтобы его найти.

### 1.7 Сводная таблица результатов для Линейного поиска (лучший случай)

| n | T, микросекунд | Тэп = f(C+M) - функция | Сф - количество |
|---|---|---|---|
| 100 | 0 | O(1) | 1 |
| 1000 | 0 | O(1) | 1 |
| 10000 | 1 | O(1) | 1 |
| 100000 | 2 | O(1) | 1 |

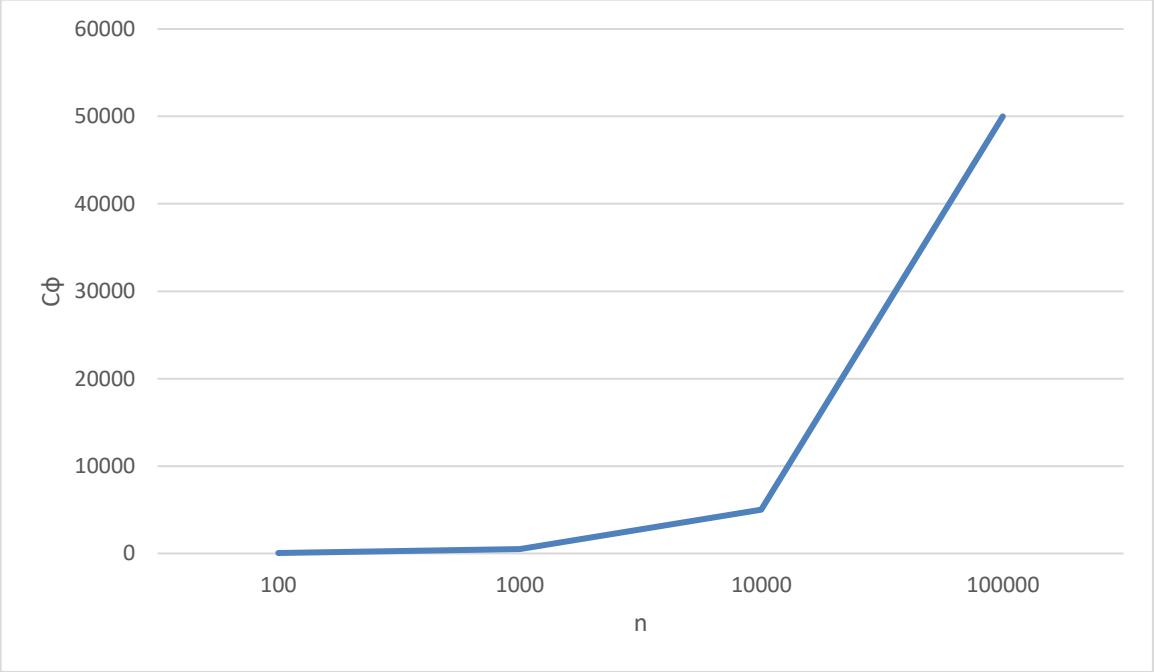
### 1.8 График зависимости Сф от n для Линейного поиска (лучший случай)



### 1.9 Сводная таблица результатов для Линейного поиска (средний случай)

| n | T, микросекунд | Тэп = f(C+M) - функция | Сф - количество |
|---|---|---|---|
| 100 | 1 | O(n) | 51 |
| 1000 | 3 | O(n) | 501 |
| 10000 | 31 | O(n) | 5001 |
| 100000 | 341 | O(n) | 50001 |

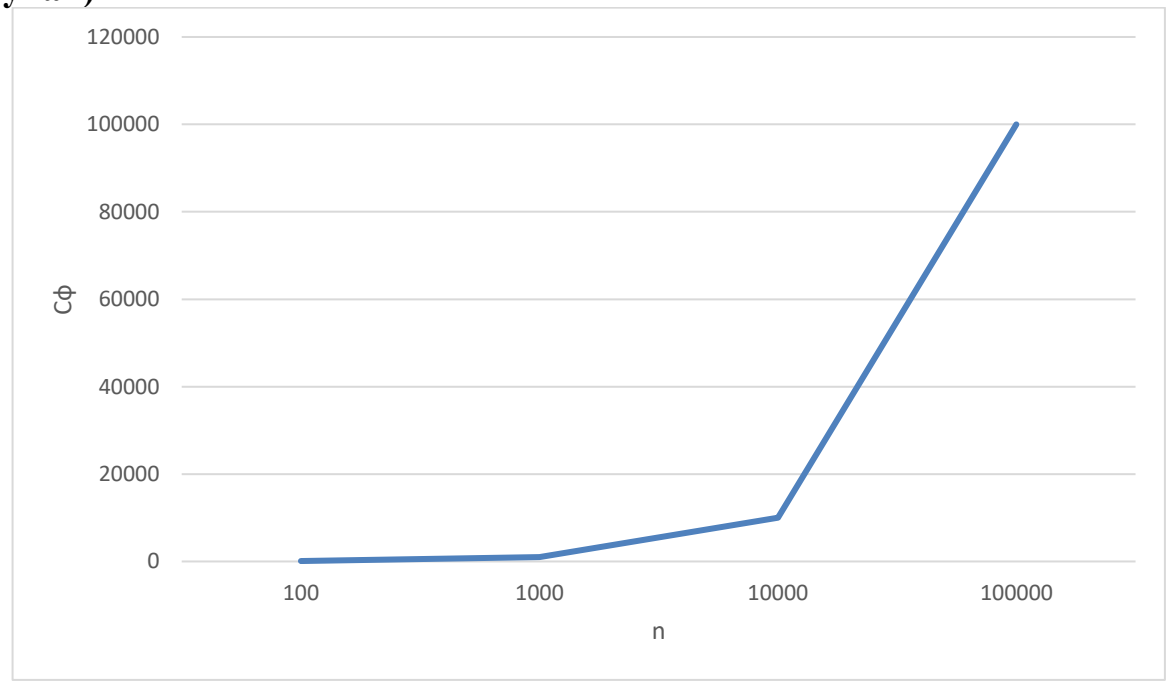
### 1.10 График зависимости Сф от n для Линейного поиска (средний случай)



**1.11 Сводная таблица результатов для Линейного поиска (худший случай)**

| n | T, микросекунд | Тэп = f(C+M) - функция | Сф - количество |
|---|---|---|---|
| 100 | 1 | O(n) | 100 |
| 1000 | 1 | O(n) | 1000 |
| 10000 | 12 | O(n) | 10000 |
| 100000 | 737 | O(n) | 100000 |

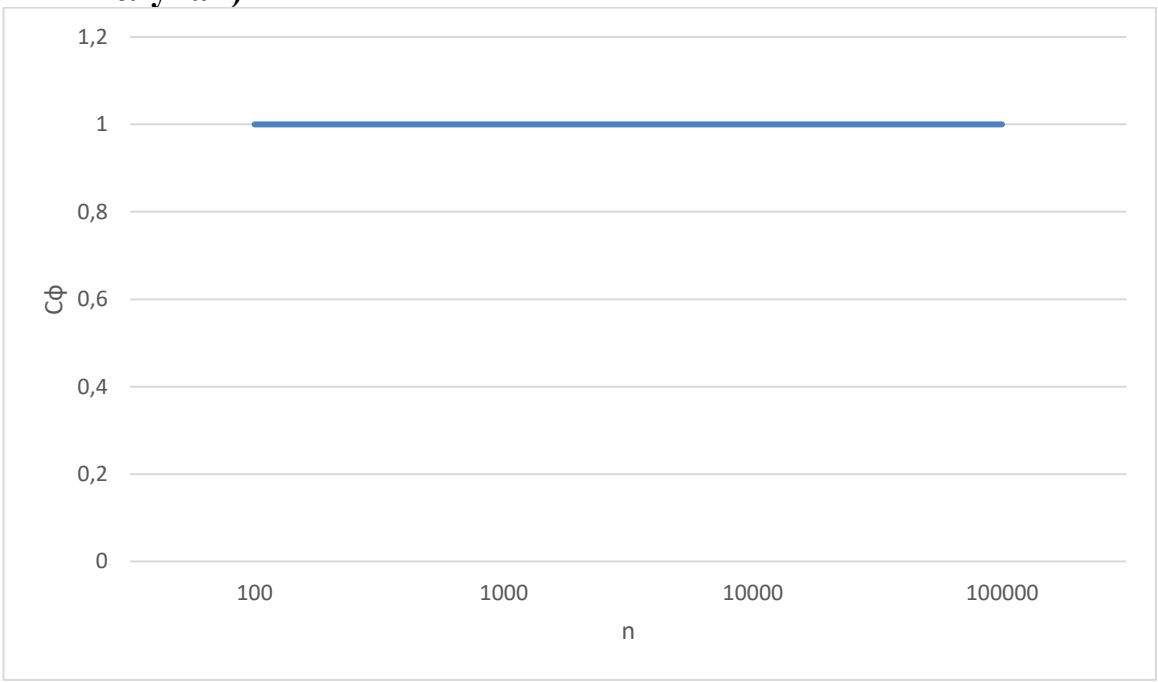
**1.12 График зависимости Сф от n для Линейного поиска (худший случай)**



**1.13 Сводная таблица результатов для Линейного поиска с барьером (лучший случай)**

| n | T, микросекунд | Тэп = f(C+M) - функция | Сф - количество |
|---|---|---|---|
| 100 | 9 | O(1) | 1 |
| 1000 | 11 | O(1) | 1 |
| 10000 | 6 | O(1) | 1 |
| 100000 | 12 | O(1) | 1 |

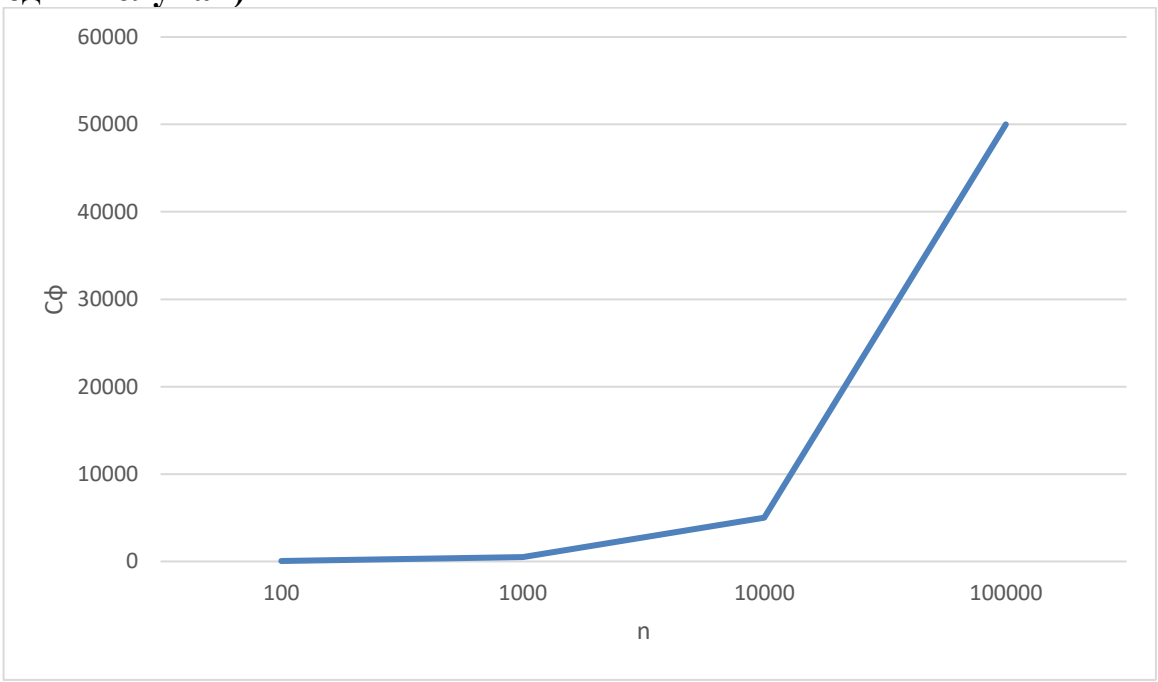
**1.14 График зависимости Сф от n для Линейного поиска с барьером (лучший случай)**



**1.15 Сводная таблица результатов для Линейного поиска с барьером (средний случай)**

| n | T, микросекунд | Тэп = f(C+M) - функция | Сф - количество |
|---|---|---|---|
| 100 | 13 | O(n) | 51 |
| 1000 | 13 | O(n) | 501 |
| 10000 | 33 | O(n) | 5001 |
| 100000 | 234 | O(n) | 50001 |

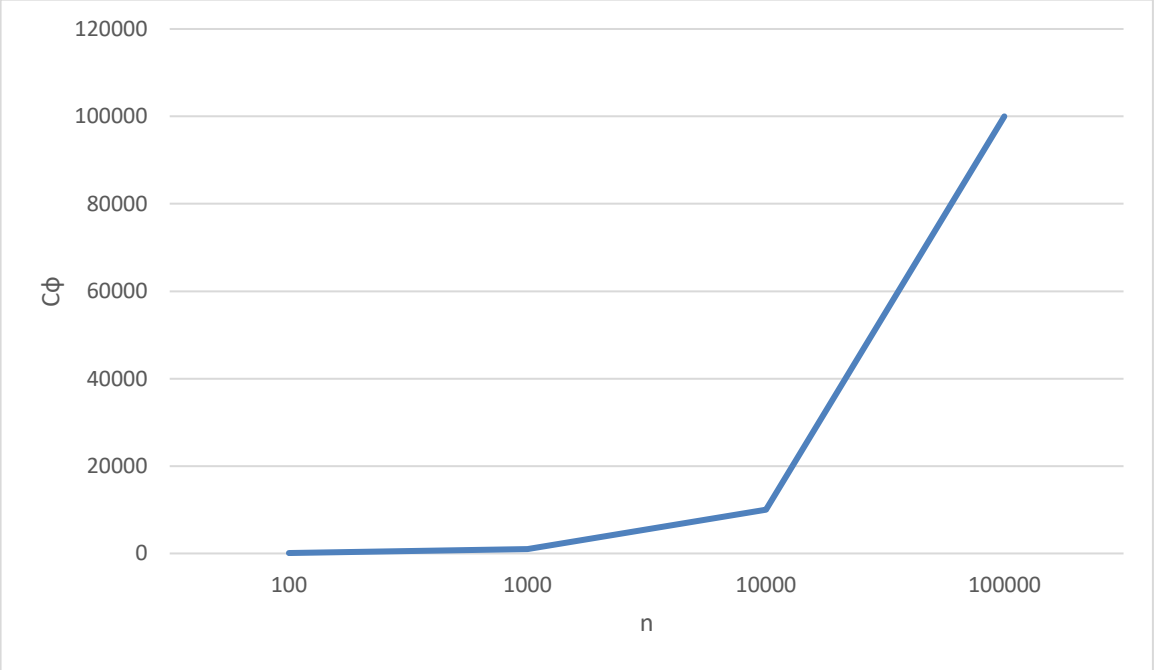
**1.16 График зависимости Сф от n для Линейного поиска с барьером (средний случай)**



**1.17 Сводная таблица результатов для Линейного поиска с барьером (худший случай)**

| n | T, микросекунд | Тэп = f(C+M) - функция | Сф - количество |
|---|---|---|---|
| 100 | 10 | O(n) | 100 |
| 1000 | 11 | O(n) | 1000 |
| 10000 | 23 | O(n) | 10000 |
| 100000 | 531 | O(n) | 100000 |

**1.18 График зависимости Сф от n для Линейного поиска с барьером (худший случай)**



### 1.19 Графики зависимости Сф от n для всех случаев


### 1.20 Анализ полученных результатов (Линейный поиск, Линейный поиск с барьером)

Алгоритм линейного поиска и алгоритм линейного поиска с барьером являются простыми методами поиска элемента в массиве. Они основаны на переборе всех элементов в массиве один за другим до тех пор, пока не будет найден искомый элемент.

Линейный поиск – алгоритм прост в реализации, но его скорость работы линейно зависит от размера массива, поэтому для больших массивов он может быть очень медленным. Линейный поиск с барьером – алгоритм обеспечивает более быстрое выполнение по сравнению с линейным поиском, потому что можно сократить количество проверок в цикле до одного. Однако, если элемент, который мы ищем, находится в конце массива, то этот алгоритм не дает преимущества по скорости.

# 2 ЗАДАНИЕ 2

## 2.1 Постановка задачи

Разработать программу поиска записи по ключу в таблице записей с применением алгоритмов, определенных в задании варианта.

1. Таблица содержит записи, структура которых определена вариантом. Ключи уникальны в пределах таблицы.

2. Разработать алгоритм поиска, определенный в варианте. Реализовать алгоритм функцией.

3. Провести практическую оценку времени выполнения алгоритмов на таблицах объемом 100, 1000, 10 000 записей на случайно заполненных таблицах (худший случай). На таблицах с лучшим временем и средним.

4. Составить таблицу с указанием: времени выполнения алгоритма, его фактическую и теоретическую вычислительную сложность.

Номер индивидуального варианта 10%16=1

| № | Алгоритмы поиска | Структура записи файла (ключ – подчеркнутое поле) |
|---|---|---|
| 2 | Интерполяционный поиск | Страхование авто средства: регистрационный номер – шестизначное число, название страховой компании |

## 2.2 Алгоритм на псевдокоде

**Подробное описание:**

1. Установить переменные low=0 и high=n-1, где n - размер массива.

2. Проверить, что значение ключа key находится в пределах [arr[low].regNumber, arr[high].regNumber], иначе элемент не может быть найден в массиве. Если условие не выполнено, вернуть -1.

3. Пока low<=high и key>=arr[low].regNumber && key<=arr[high].regNumber:

a. Если low == high, то проверить, равно ли arr[low].regNumber ключу key. Если равно, вернуть low, иначе вернуть -1.

b. Вычислить позицию элемента pos = low + (key - arr[low].regNumber) * (high - low) / (arr[high].regNumber - arr[low].regNumber).

c. Если arr[pos].regNumber равно ключу key, вернуть pos.

d. Если arr[pos].regNumber меньше ключа key, обновить переменную low = pos + 1.

e. Если arr[pos].regNumber больше ключа key, обновить переменную high = pos - 1.

4. Элемент не найден в массиве, вернуть -1.

**Псевдокод:**

```
function interpolationSearch(arr, n, key):
    low = 0
    high = n - 1

    while low <= high AND key >= arr[low] AND key <= arr[high]:
        # вычисляем позицию с помощью интерполяционной формулы
        pos = low + ((key - arr[low]) * (high - low)) // (arr[high] - arr[low])

        if arr[pos] == key:
            return pos

        if arr[pos] < key:
            low = pos + 1
        else:
            high = pos - 1

    return -1
```

## 2.3 Код функции поиска

```
3   // функция интерполяционного поиска на массиве.
4   int interpolationSearch(AutoInsurance arr[], int n, int key)
5   {
6       int low = 0, high = n - 1;
7
8       while (low <= high && key >= arr[low].regNumber && key <= arr[high].regNumber)
9       {
10          if (low == high)
11          {
12              if (arr[low].regNumber == key)
13              {
14                  return low;
15              }
16              return -1;
17          }
18
19          int pos = low + ((double)(key - arr[low].regNumber) / (arr[high].regNumber
    - arr[low].regNumber)) * (high - low); // вычисление позиции.
20
21          if (arr[pos].regNumber == key)
22          {
23              return pos;
24          }
25          if (arr[pos].regNumber < key)
26          {
27              low = pos + 1;
28          }
29          else
30          {
31              high = pos - 1;
32          }
33      }
34
35      return -1;
36  }
```

## 2.4 Теоретическая сложность алгоритма

**В худшем случае** сложность алгоритма интерполяционного поиска составляет O(n). Это происходит, когда искомый элемент находится в крайней позиции и массив содержит большое кол-во элементов.

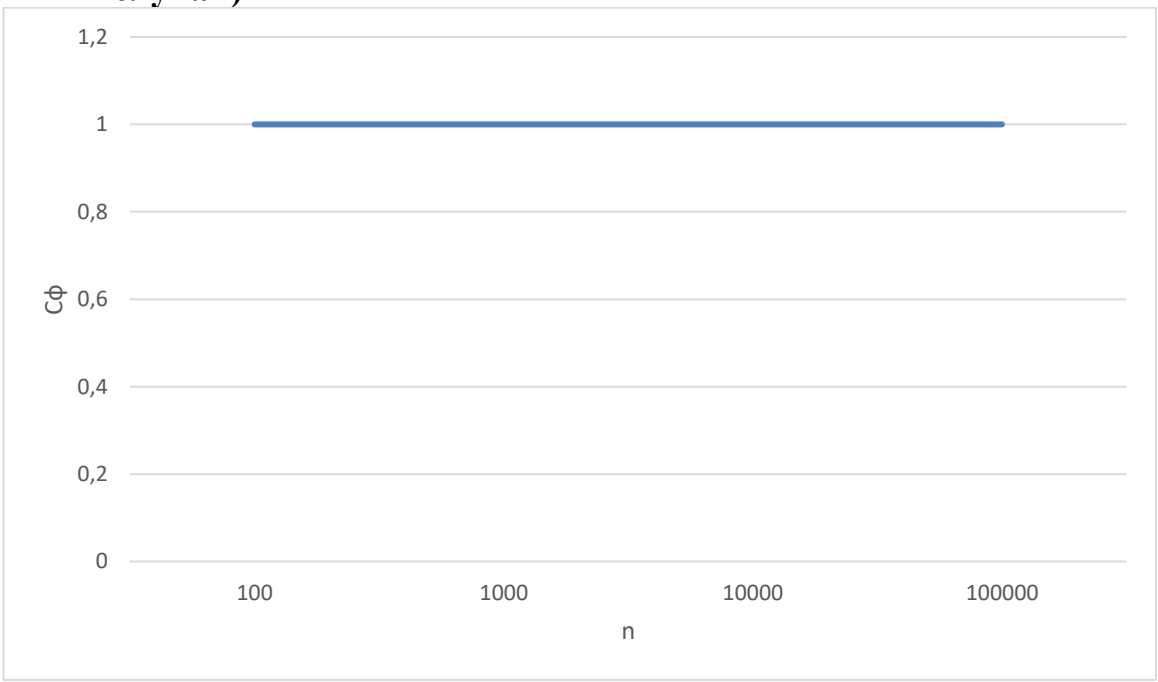
**В лучшем случае**, когда искомый элемент находится в середине отсортированного массива, сложность алгоритма является константой, т.е. O(1).

**В среднем случае**, если элементы массив равномерно распределены и нет повторяющихся значений, тогда поиск сможет быстро находить нужный элемент, используя линейную интерполяцию. Сложность в таком случае будет O(log log n). Если элементы распределены неравномерно, то сложность – O(log n).

**2.5 Сводная таблица результатов для Интерполяционного (лучший случай)**

| n | T, микросекунд | Тэп = f(C+M) - функция | Сф - количество |
|---|---|---|---|
| 100 | 0 | O(1) | 1 |
| 1000 | 0 | O(1) | 1 |
| 10000 | 1 | O(1) | 1 |
| 100000 | 3 | O(1) | 1 |

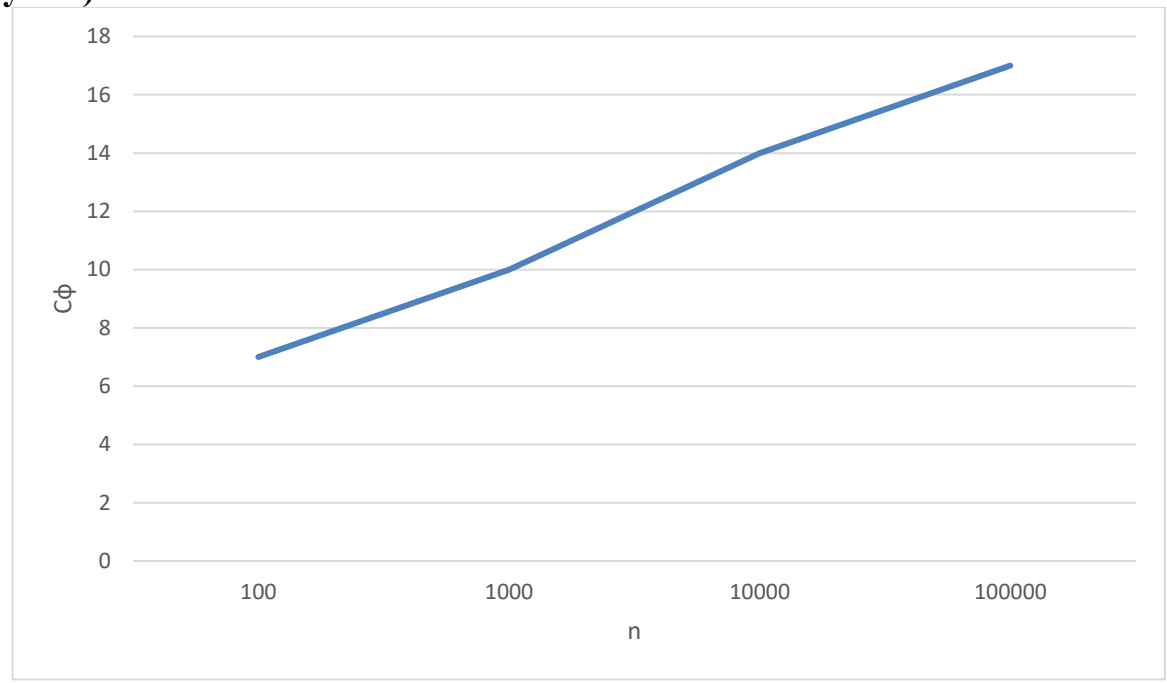
**2.6 График зависимости Сф от n для Интерполяционного поиска (лучший случай)**



**2.7 Сводная таблица результатов для Интерполяционного поиска (средний случай)**

| n | T, микросекунд | Тэп = f(C+M) - функция, неравномерное распределение | Сф - количество |
|---|---|---|---|
| 100 | 0 | O(log n) | 7 |
| 1000 | 1 | O(log n) | 10 |
| 10000 | 2 | O(log n) | 14 |
| 100000 | 5 | O(log n) | 17 |

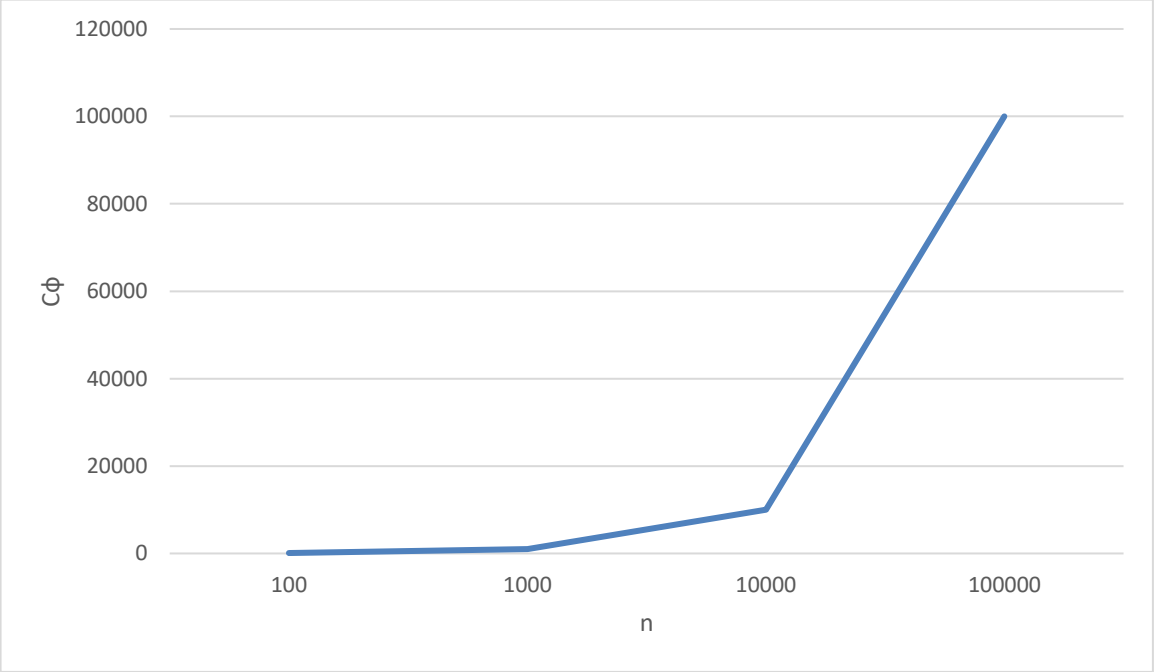
**2.8 График зависимости Сф от n для Интерполяционного (средний случай)**



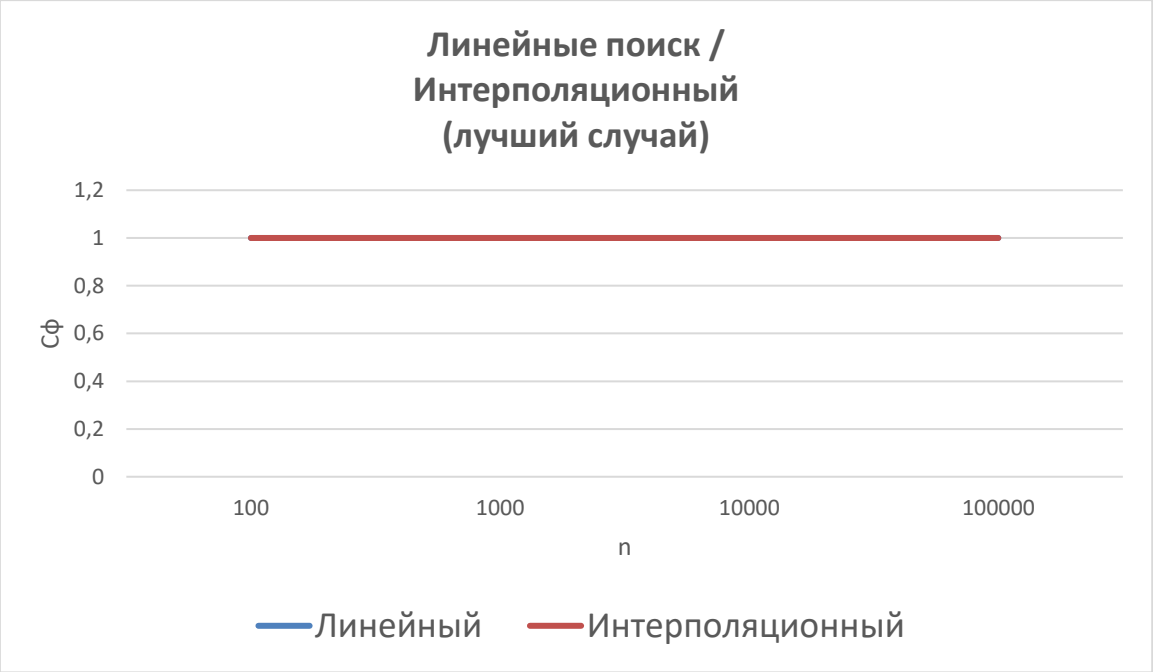
**2.9 Сводная таблица результатов для Интерполяционного поиска (худший случай)**

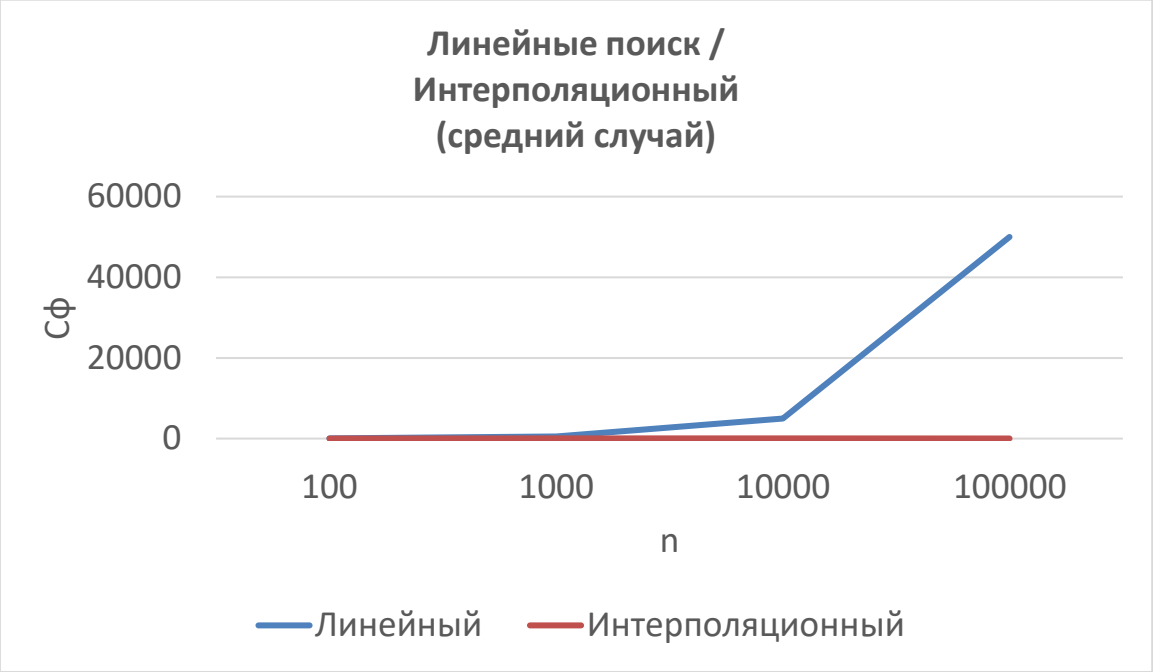
| n | T, микросекунд | Тэп = f(C+M) - функция | Сф - количество |
|---|---|---|---|
| 100 | 0 | O(n) | 100 |
| 1000 | 1 | O(n) | 1000 |
| 10000 | 1 | O(n) | 10000 |
| 100000 | 8 | O(n) | 100000 |

**2.10 График зависимости Сф от n для Интерполяционного поиска (худший случай)**



## 2.11 Сравнение алгоритмов Линейного и Интерполяционного поиска




### 2.12 Анализ полученных результатов

Лучшие случаи у двух поисков совпадают по сложности. По среднему случаю можно сделать выводы, что интерполяционный поиск работает быстрее и эффективнее линейного, но для интерполяционного поиска в худшем случае сложность зависит от распределения ключей в массиве. Если ключи не равномерно распределены и имеют множество повторяющихся значений (в данной реализации они ключи не повторяются, но они распределены неравномерно), то интерполяционный поиск может превратиться в линейный поиск по сложности алгоритма в худшем случае, поэтому их графики совпадают. В таком случае сложность будет O(n), как и у линейного поиска в худшем случае. Однако, если ключи равномерно распределены, то сложность интерполяционного поиска в худшем случае будет O(log log n), что лучше, чем O(n) у линейного поиска в худшем случае.

# 3 ВЫВОДЫ

В ходе практической работы было осуществлено сравнение трех алгоритмов поиска элемента в массиве: линейный поиск, линейный поиск с барьером и интерполяционный поиск.

Таким образом, для выбора наиболее эффективного алгоритма для решения конкретной задачи, необходимо учитывать размер и способ сортировки массива. Если массив не очень большой, то можно использовать линейный поиск или его модификацию с барьером. Если в массиве нет повторяющих элементов, и они равномерно распределены, то лучше использовать интерполяционный поиск.

# 4 СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ

1. Материалы по дисциплине (Сорокин А.В.).

2. Скворцова Л.А., Гусев К.В., Трушин С.М., Филатов А.С. Учебно-методическое пособие СиАОД.