1 Введение

Цель: определить асимптотики поиска различных алгоритмов поиска (линейного и бинарного), найти оптимальный алгоритм поиска элементов в сумме дающих заданное число. После чего оценить асмиптотику стратегий данных в задании.

Задачи:

- Определить асимптотики поиска различных алгоритмов поиска (линейного и бинарного).
- Доказать асимптотику линейного поиска элементов в сумме дающих заданное число.
- Найти оптимальный алгоритм поиска элементов в сумме дающих заданное число (в упорядоченном массиве).

Для разичных стратегий проверить:

- изменится ли асимптотика поиска, если при поиске запросы будут равномерно распределены по множеству значений, включая неудачный поиск (нет такого элемента).
- изменится ли асимптотика поиска, если распределение не равномерно (какие-то данные при поиске встречаются гораздо чаще);
- есть ли различия в асимптотической сложности этих стратегий при равномерном и неравновероном распределении данных.

2 Ход работы

2.1 Линейный и бинарный поиск

Для начала стоит написать алгоритм линейного и бинарного поиска.

Линейный поиск:

```
    int search (vector<int>& var, int val)
{
    for (int i = 0; i < var.size() ; ++i){
        if (var[i] == val)
            return i;
    }
    return -1;
}
```

Бинарный поиск:

```
    int Bin_Search(vector<int>& var, int val)
{
    int l = 0, r;
    r = var.size() - 1;

    while (r > l)
    {
        int m = (l + r) / 2;

        if (val > var[m])
            l = m + 1;
        else if (var[m] > val)
            r = m - 1;
        else
            return m;
    }

    return -1;
}
```

Массив для обработки будем создавать из последовательности натуральных чисел для удобства, на асимптотику это не повлияет. Для более

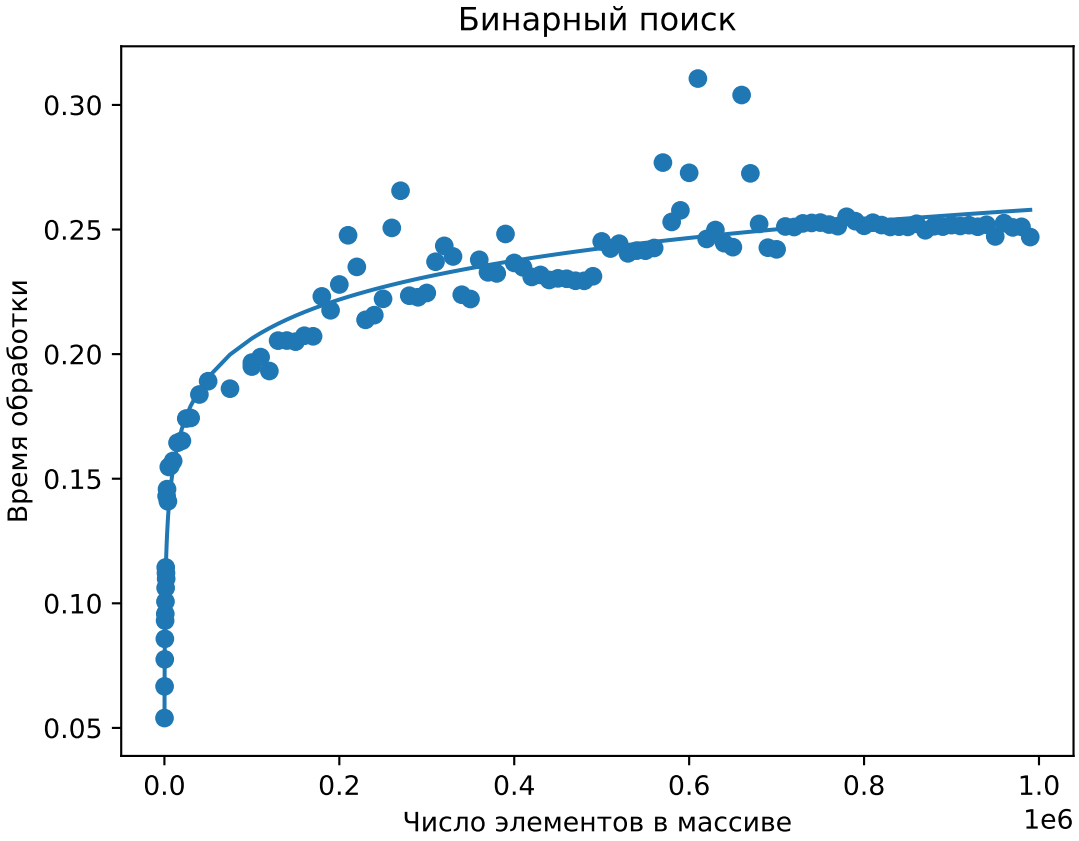
точной оценки будем прогонять поиск одного значения (которого нет в массиве) несколько раз.

Аналитически время должно быть $O(N)$ для линейного поиска, а для $O(\log(N))$ для бинарного поиска.

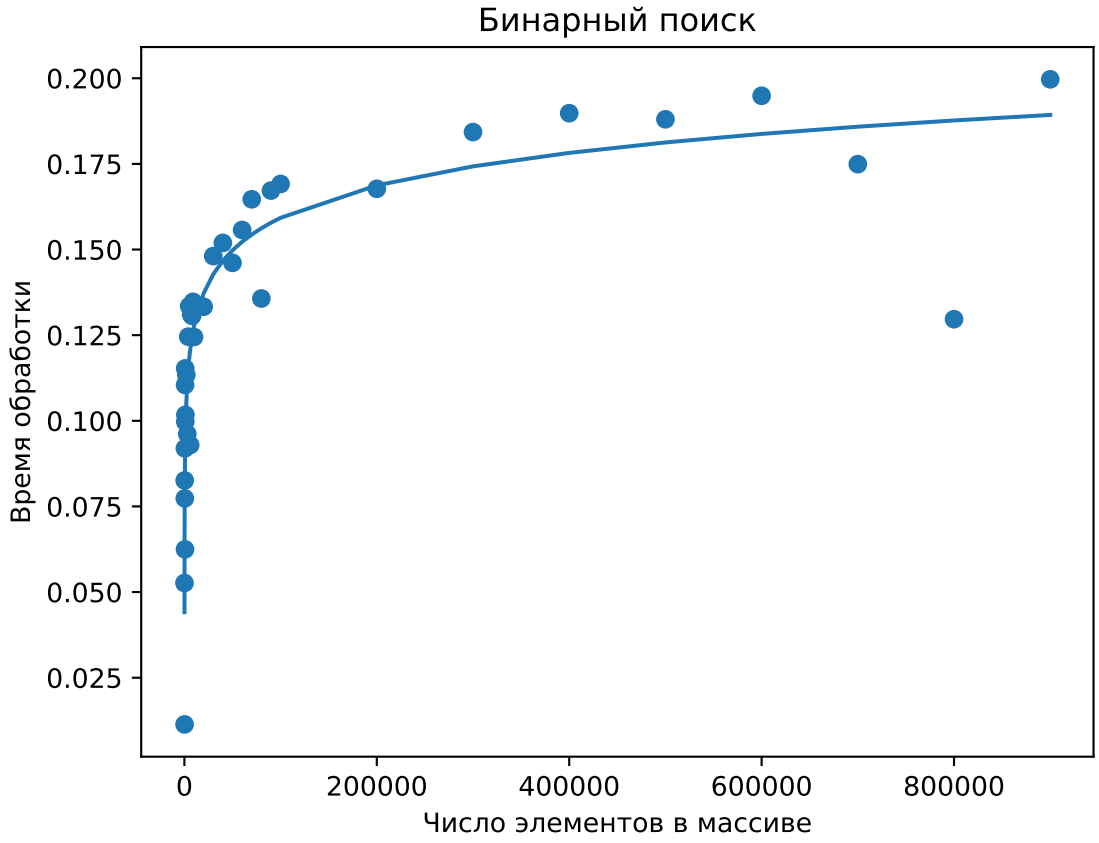
Рассмотрим результаты и построим графики, и попробуем апроксимировать.

Бинарный поиск:

Худший случай:



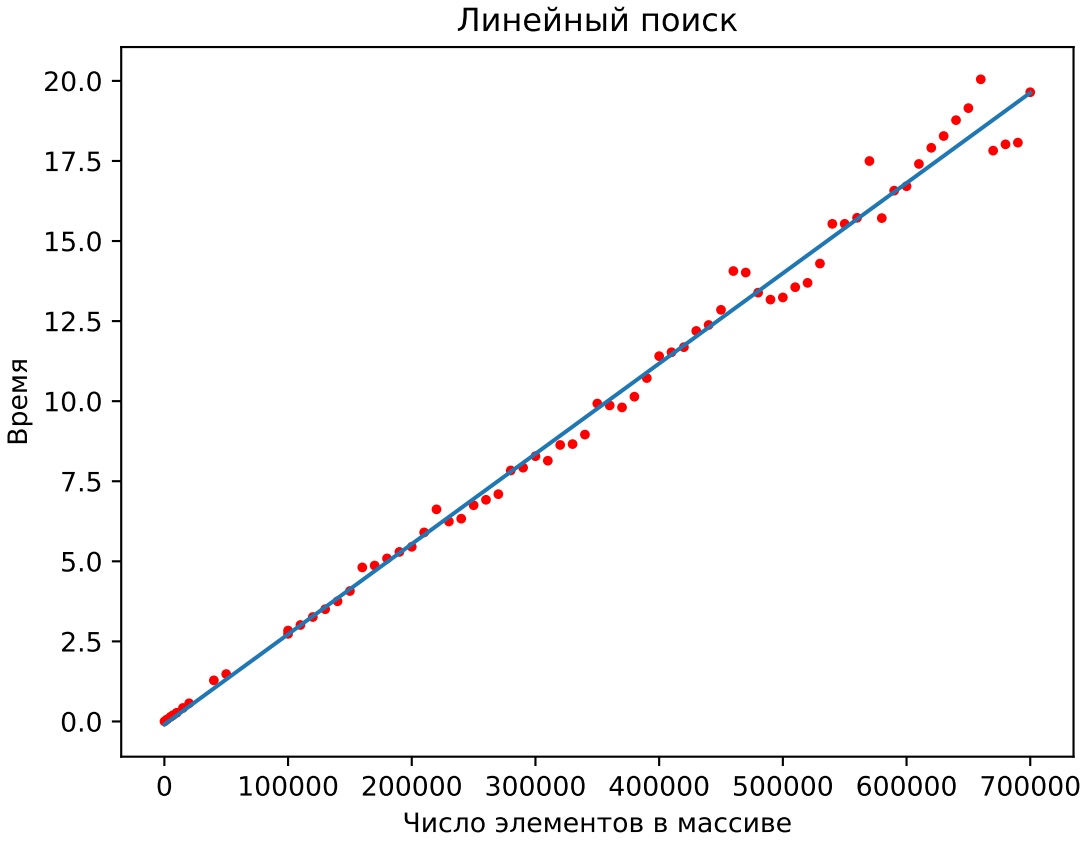
Лучший случай:



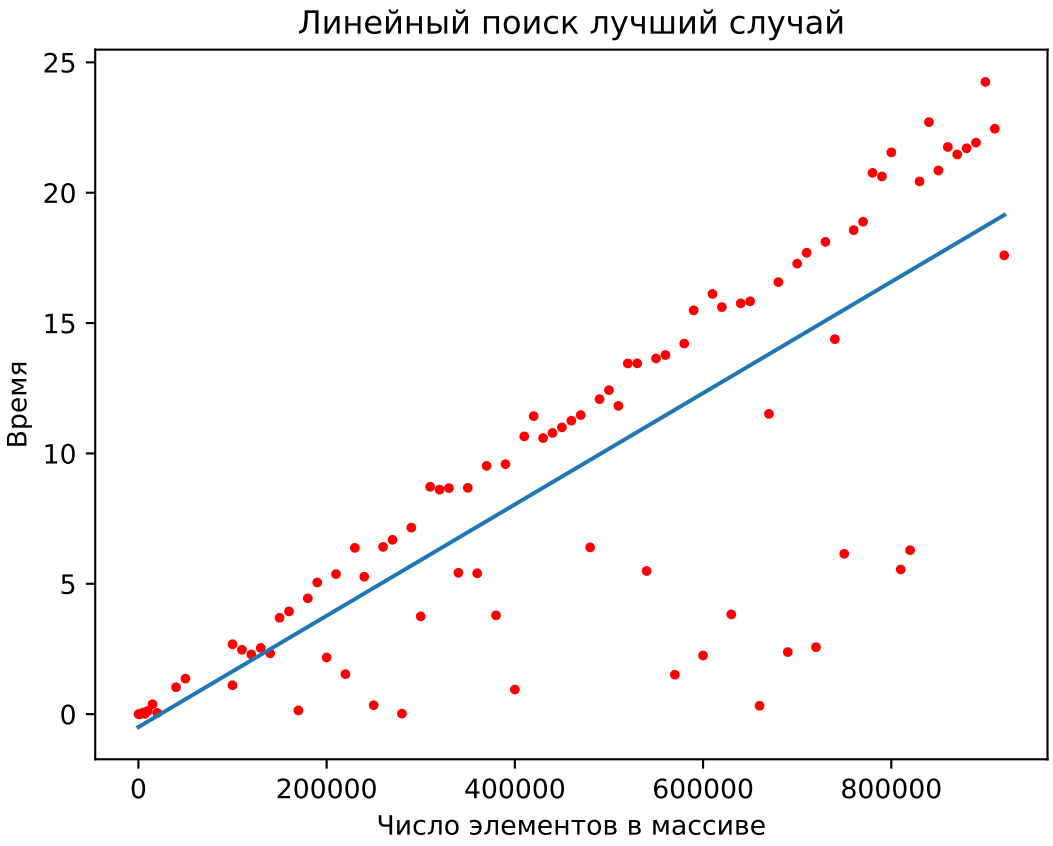
Как видно из графиков в обоих случаях данные апроксимируются логарифмом.

Линейный поиск:

Худший случай:


Линейный поиск
Время
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0
100000
200000
300000
400000
500000
600000
700000
Число элементов в массиве

Лучший случай:



Как видно из графиков в обоих случаях данные апроксимируются прямой (во втором случае с точностью до значений меньшего чиса значений).

Вывод по первой части: Приходим к выводу о том, что асимтотики согласуются с теорией.

2.2 Поиск элементов сумма которых равна заданому числу

Из предыдущего задания очеивдно вытекает решение проблемы, в массиве необходимо искать дополнение к каждому элементу (такой элемент прибавив к которому получим искомое, то есть $n - i$).

Простой вариант:

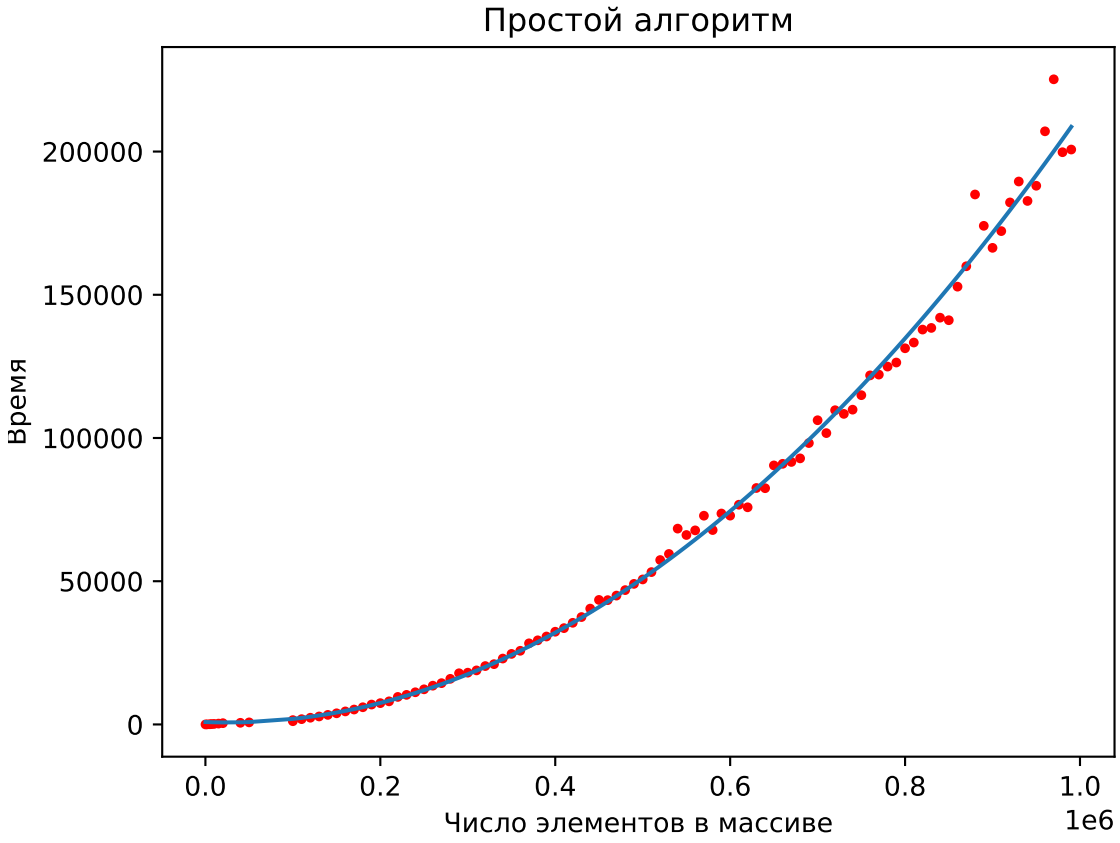
```
int search_sum(vector<int>& var, int n, int size)
{
    for (int i = 0; i < var.size(); ++i)
    {
        if (search(var, size, n-i) >=0 and search(var, size, n-i) !=0)
            return (i, search(var, size, n-i));
    }
    return -1;
}
```

Заметим, что в этом случае мы используем вложенный цикл, значит сложность алгоритма - это $O(N^2)$. подтвердим это прямыми измерениями. Дальше возникает идея для отсортированного массива воспользоваться бинарным поиском, тогда сложность падает до $N \log(N)$, но алгоритм можно упростить еще больше идя с двух концов и если оказывается, что сумма элементов слишком большая, то сдвигается один конец, иначе другой, тогда поскольку идет перебор не более чем N элементов сложность соответсвенно $O(N)$. (При поиске решения была идея сделать что то похожее на бин поиск, появилось такое название)

Быстрый вариант:

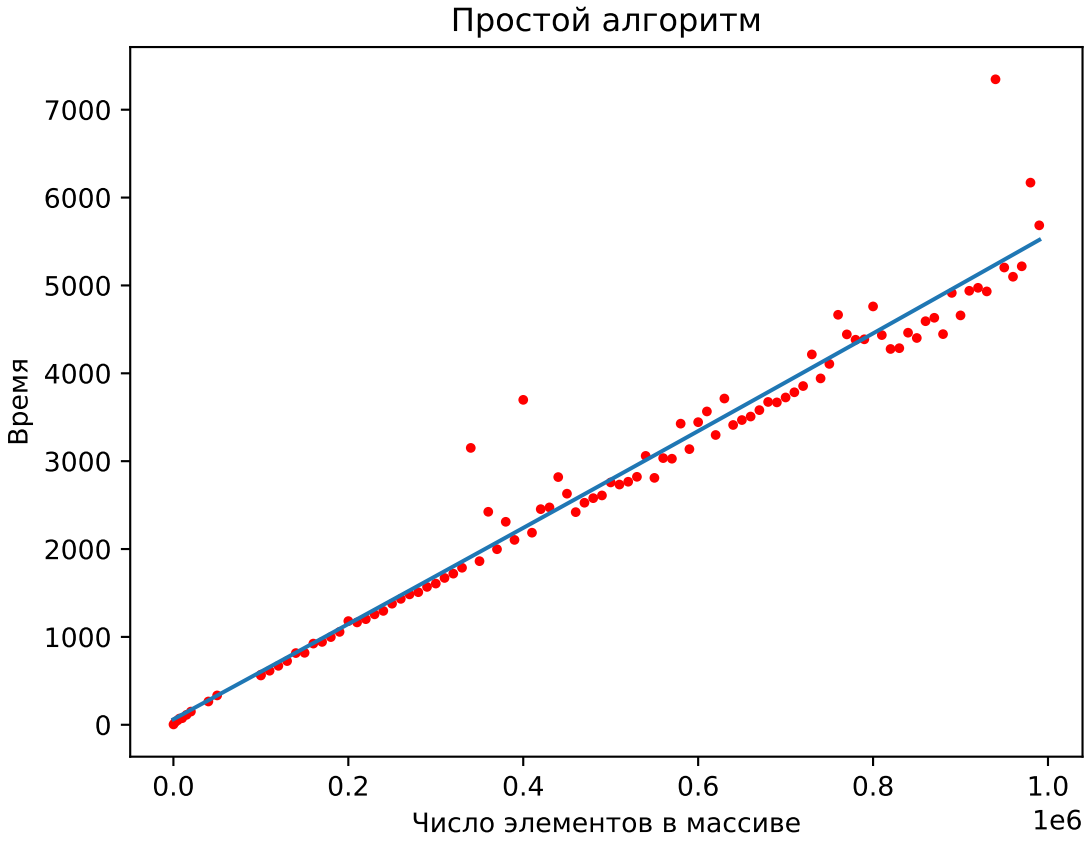
```
int Bin_search_sum(vector<int>& var, int n, int size)
{
    int lt = 0;
    int rt = size - 1;
    while (lt != rt)
    {
        int cursum = var[lt] + var[rt];
        if (cursum < n)
            lt++;
        else if (cursum > n)
            rt--;
        else
        {
            return (rt, lt);
        }
    }
    return -1;
}
```

Рассмотрим асисмтотику простого варианта:



Заметим, что данные очень хорошо ложаться на параболу, что сходитья с теорией.

Теперь рассмотрим асимпотику быстрого варианта:



Заметим что она также имеет линейную асимптотику.

Вывод по второму заданию: Действительно асимтотика согласуется с теоритическим обоснование.

2.3 Стратегии

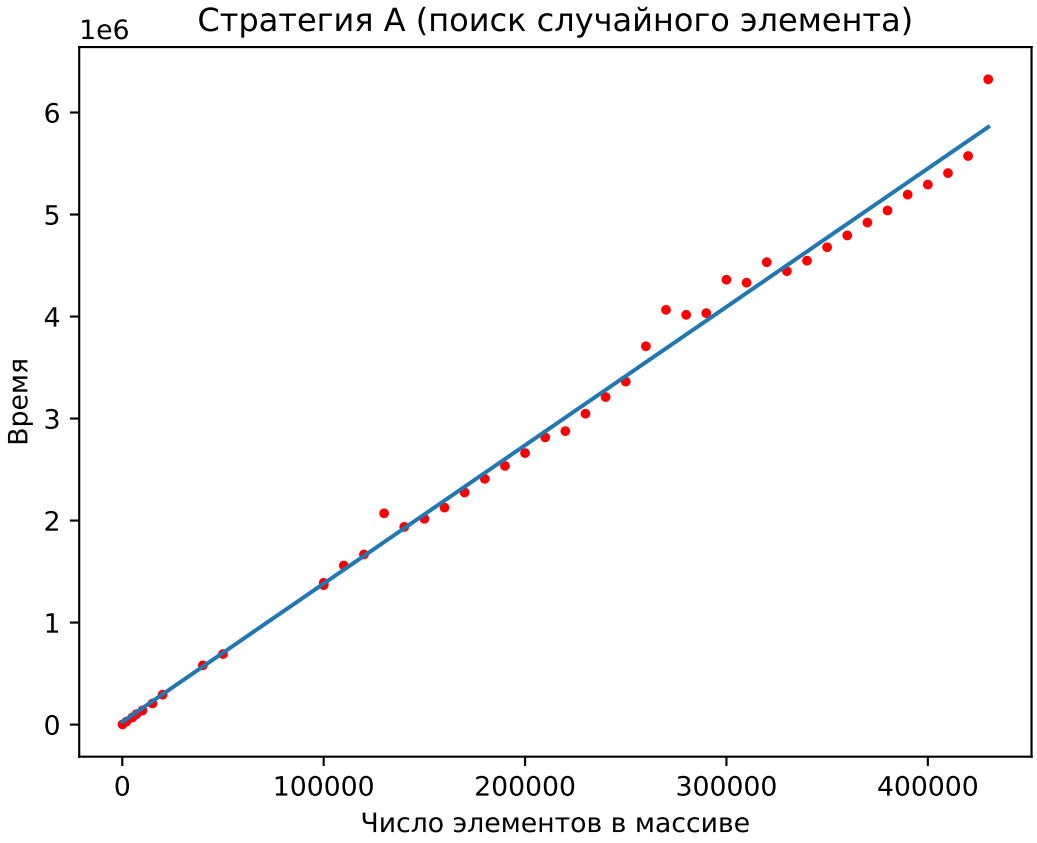
2.3.1 Стратегия A

Разберем и оценим асимптотику каждого поиска.

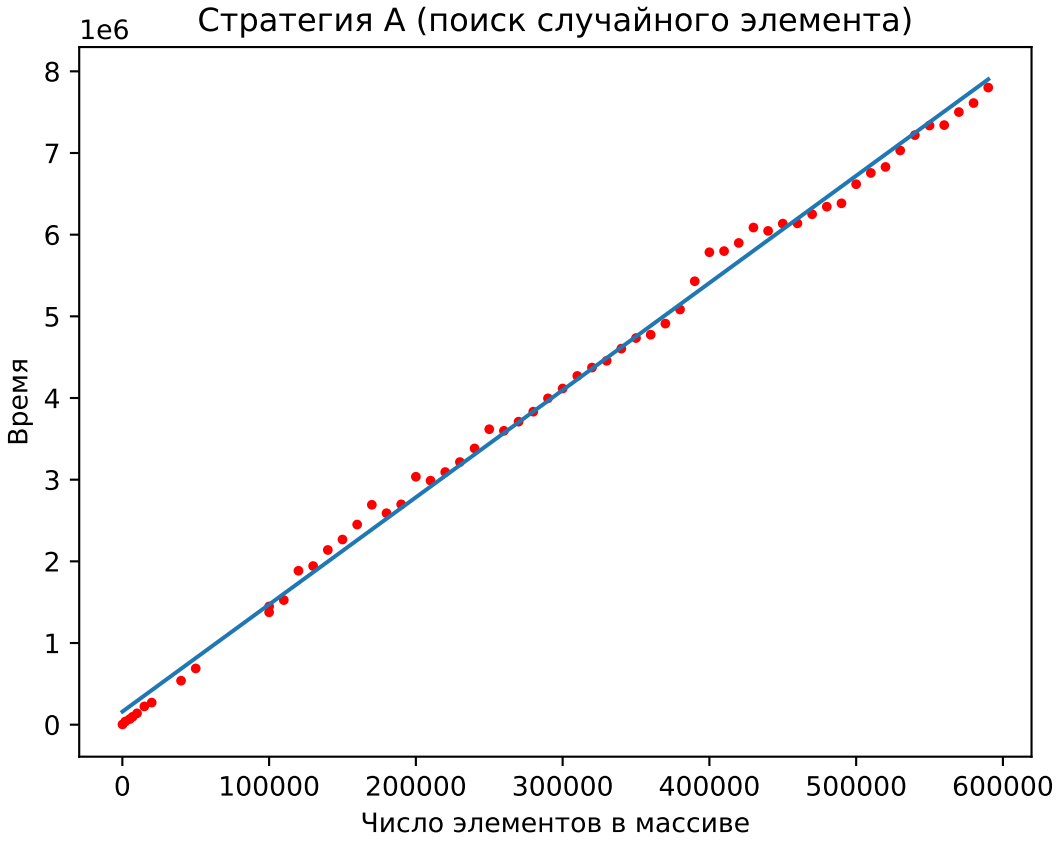
Первый случай: поиск случайного элемента с не равномерным запросом, асимптотика оказалась линейной (каждый 4 поиск был поиск одного элемента). Рассмотрим код и график.

```
int swap_A(vector<int>& var, int size, int el)
```

```
{
    int tmp = 0;
    for (int i = 0; i < size; ++i)
    {
        if (var[i] == el){
            tmp = var[0];
            var[0] = var[i];
            var[i] = tmp;
            return i;
        }
    }
    return -1;
}
```



Теперь рассмотрим равномерный поиск, асимптотика оказывается такая же.



При не равномерном распредлении данных видим линейную асимптотику.

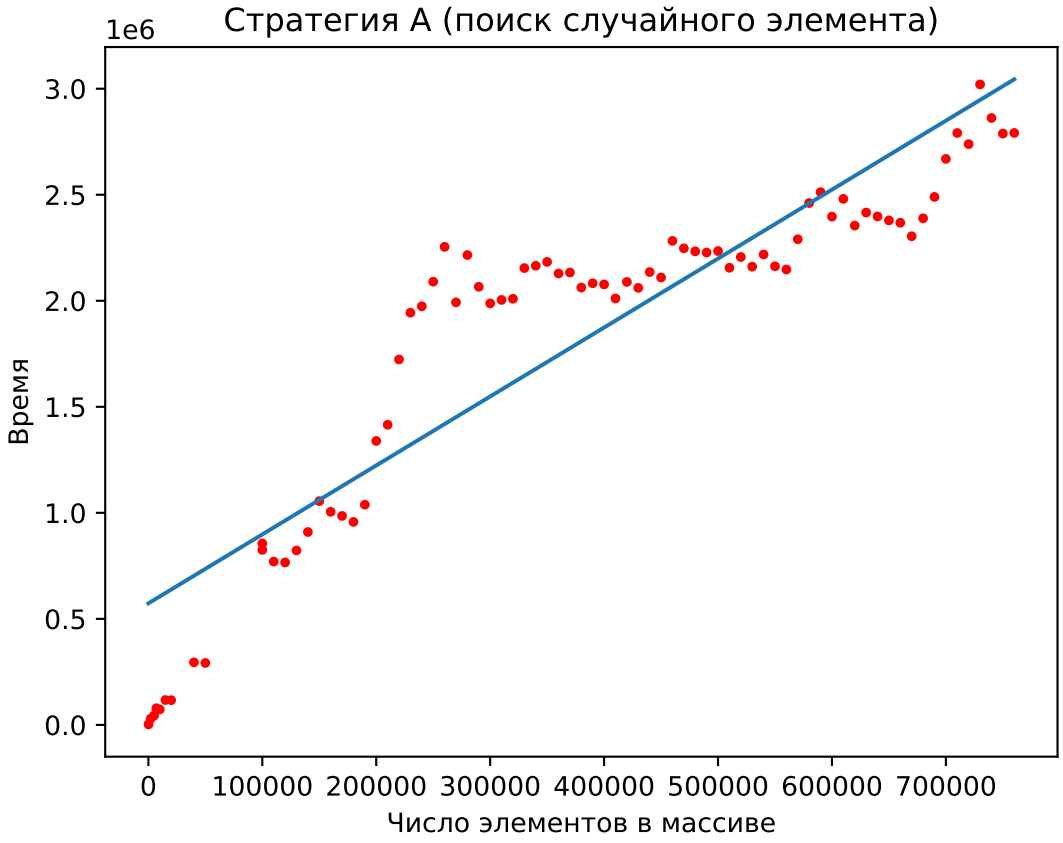


График зависимости в B и C получились также линейны (их асимптотики), тогда можно сделать вывод, что в целом все эти стратегии эквивалентны, с точностью до мультипликативной и аддитивной константы.

```
int swap_B(vector<int>& var, int size, int el){
    int tmp = 0;
    for (int i = 0; i < size; ++i) {
        if (var[i] == el){
            if (i>0) {
                tmp = var[i];
                var[i] = var[i - 1];
                var[i - 1] = tmp;
            }
```

```
            return i;
        }
    }
    return -1;
}


int swap_C(vector<int>& var, int size, int el){
    int ind[size];
    for (int i = 0; i < size; ++i){
        ind[i] = 0;
    }

    for (int i = 0; i < size; ++i) {
        if (var[i] == el){
            int tmp = 0;
            ++ind[i];
            if (i>0 and ind[i]>ind[i-1]) {
                tmp = var[i];
                var[i] = var[i - 1];
                var[i - 1] = tmp;
            }
            return i;
        }
    }
    return -1;
}
```