

Алгоритмы имеют сложность, которая влияет на время выполнения зависимости от количества элементов, которые необходимо обработать

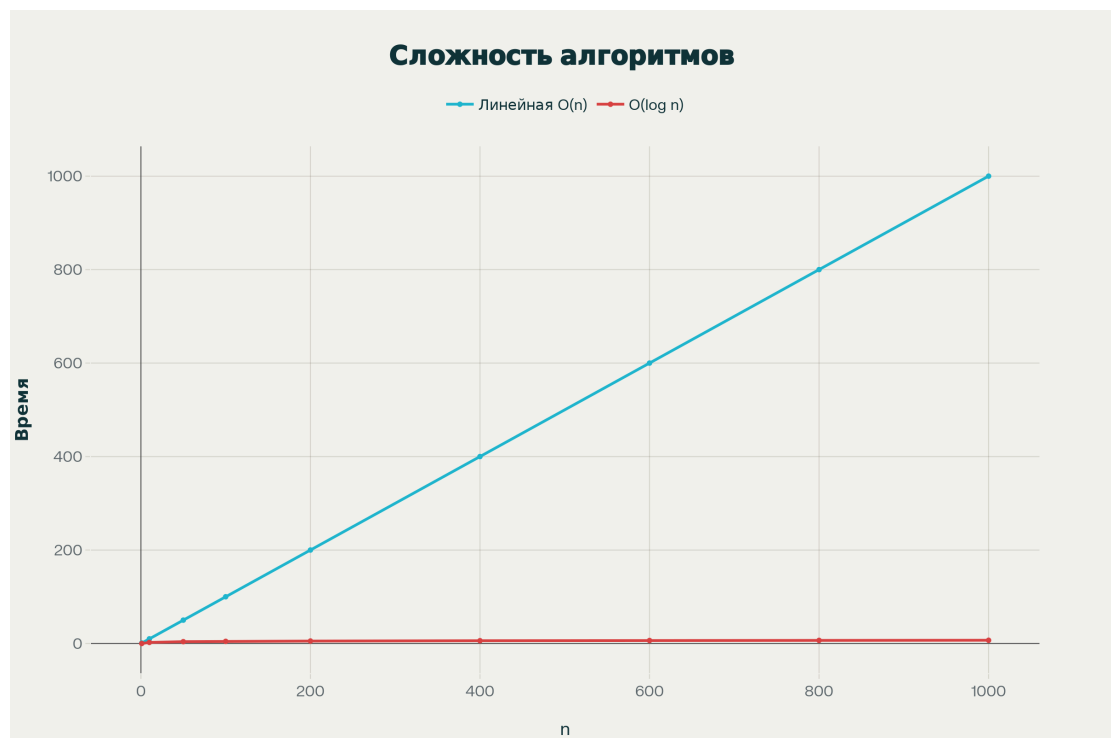
Самый простой способ найти конкретный элемент в массиве - последовательно проверять каждый элемент, пока не найдем нужный

```
In [1]: a=10 #Искомое
arr= [1,2,3,4,5,6,7,8,9,10]
for i in range(len(arr)):
    if arr[i]==a:
        print("Номер элемента:", i)
        break
```

Номер элемента: 9

В худшем случае такой алгоритм совершит  $n$  операций сравнения, где  $n$  - количество элементов в массиве. И если для маленького массива это не критично, то на больших алгоритм будет уже тормозить и занимать много времени.

Сложность такого поиска можно обозначить с помощью нотации BigO, а именно  $O(n)$  и её можно визуализировать на графике. Обратите внимание, что сложность алгоритма не обозначает время исполнения, а обозначает как это время будет возрастать с увеличением количества элементов.



На этом же графике есть вторая линия, которая визуализирует алгоритм с сложностью  $O(\log^2(n))$ . Такую сложность имеет алгоритм Бинарного (двоичного) поиска. То есть, если обычный поиск сделает 1000 операций для поиска среди тысячи элементов, то бинарный поиск только 10.

# Бинарный поиск

Бинарный поиск позволяет быстро найти элемент в уже **отсортированном массиве**.

Допустим есть отсортированный массив из 1000 элементов от 1 до 1000:

```
[1 ... 1000]
```

Мы будем искать какой-то элемент  $n$  в этом массиве

```
n=?
```

Мы определяем его границы:

Левая - 0 (индекс первого элемента)

```
l=0
```

Правая - Длина массива-1 (индекс последнего элемента)

```
r=len(arr)-1
```

После этого находим середину индекс середины этого отрезка:

```
m = (r+l)//2
```

Проверяем больше ли элемент под индексом середины, чем искомый элемент

```
if arr[m]>n:
```

Если больше, то это значит, что все элементы, которые находятся правее середины тоже больше  $n$  и не могут быть тем элементом, который мы ищем.

В этом случае мы сдвигаем правую границу до середины

```
r = m
```

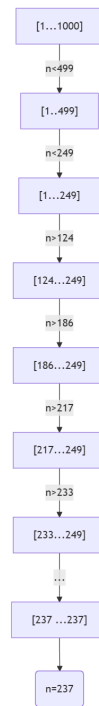
В противном случае мы сдвигаем левую границу

```
l = m
```

После этого мы повторяем эти же действия, но уже в новых границах массива, пока у нас отрезок не сузится до одного элемента, который и будет искомым

## Пример

Ищем  $n$  в массиве от 1 до 1000



# Задача

Написать алгоритм бинарного поиска элемента в массиве.