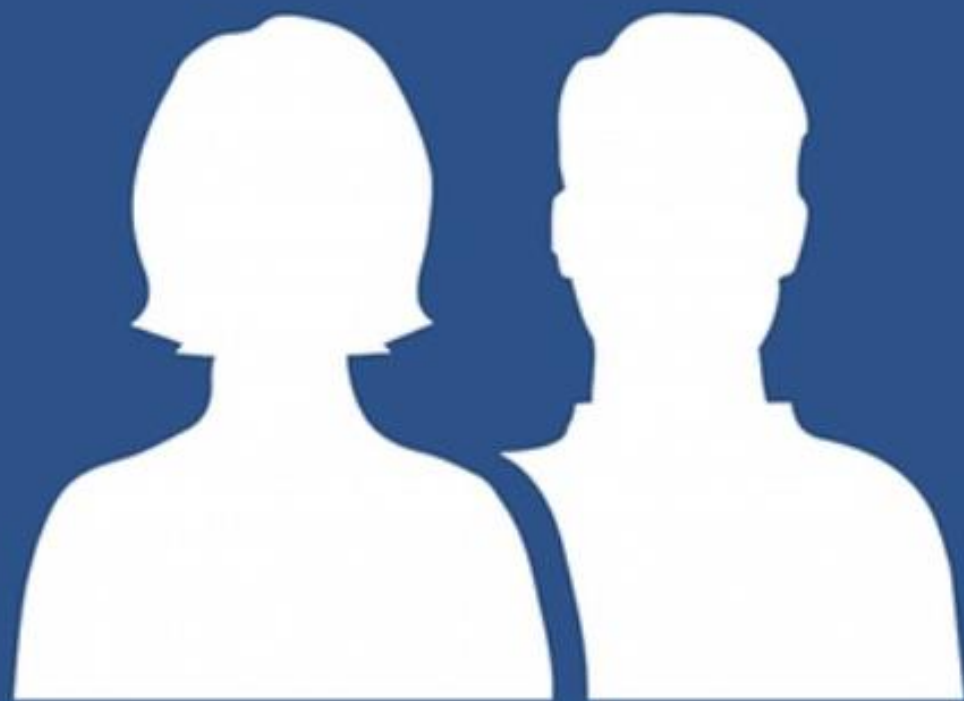


BINARY_SEARCH





«karlking»

1 2 3 ... 100



X 2 3 ... 100



X X 3 ... 100

Плохой способ
угадать число

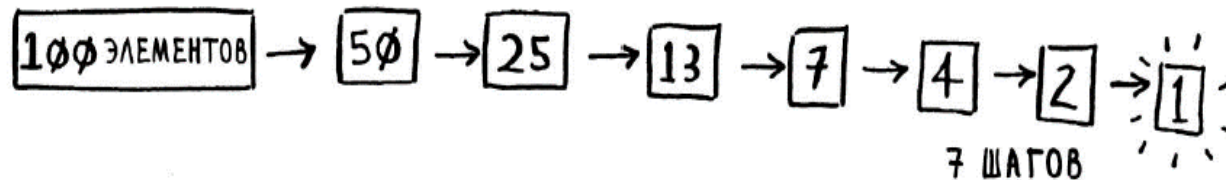


X X X X X X X ... 100



X X X X X 51 52 53 ... 100

ВСЕ ЭТИ ЧИСЛА
СЛИШКОМ МАЛЫ!

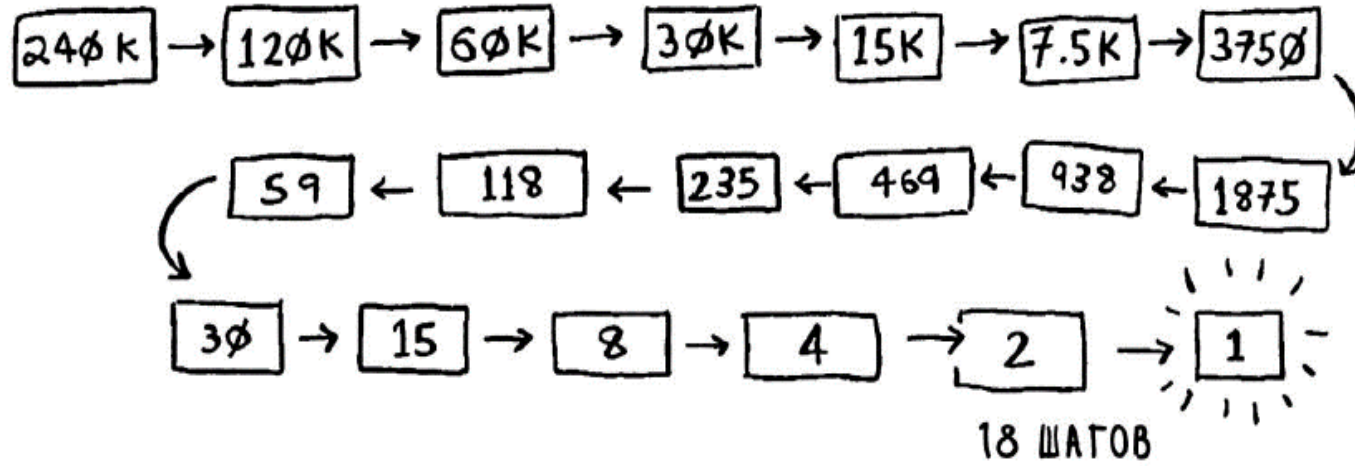


240 000

ПРОСТОЙ ПОИСК: ____ ШАГОВ

БИНАРНЫЙ ПОИСК: ____ ШАГОВ

$\log_2 n$



Боб пишет алгоритм поиска для NASA. Его алгоритм заработает, когда ракета будет подлетать к Луне, и поможет вычислить точку посадки. Боб пытается выбрать между простым и бинарным поиском. Его алгоритм должен работать быстро и правильно. С одной стороны, бинарный поиск работает быстрее. У Боба есть всего 10 секунд, чтобы выбрать место посадки; если он не уложится в это время, то момент для посадки будет упущен.

	ПРОСТОЙ ПОИСК	БИНАРНЫЙ ПОИСК
100 ЭЛЕМЕНТОВ	100 мс	7 мс
10 000 ЭЛЕМЕНТОВ	10 секунд	14 мс
1 000 000 ЭЛЕМЕНТОВ	11 дней	32 мс

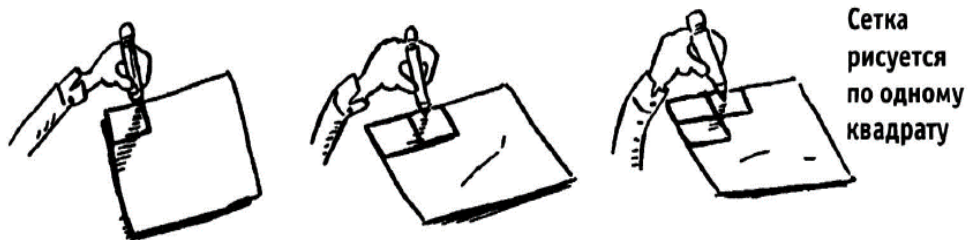
«О-БОЛЬШОЕ» $O(n)$ КОЛИЧЕСТВО ОПЕРАЦИЙ

Время выполнения растет с совершенно разной скоростью!

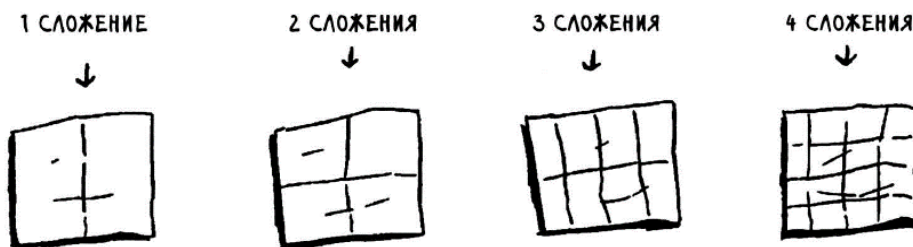
Допустим, вы должны построить сетку из 16 квадратов.

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Как должен выглядеть хороший алгоритм для построения этой сетки?



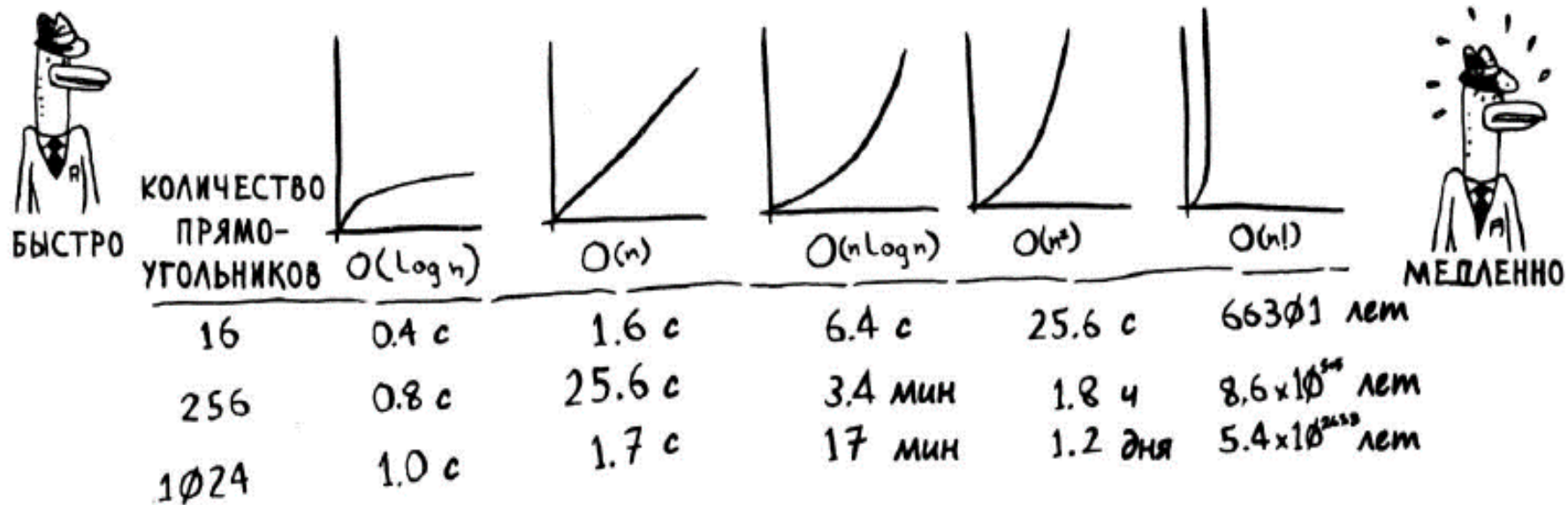
$O(n)$



Построение сетки за 4 сложения

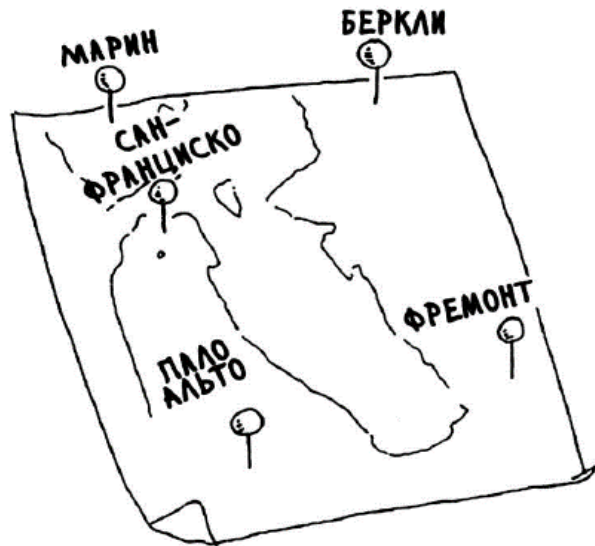
$O(\log n)$

- $O(\log n)$ – логарифмическое время (бинарный поиск);
- $O(n)$ – линейное время (простой поиск);
- $O(n \cdot \log n)$ – эффективные алгоритмы сортировки (быстрая сортировка);
- $O(n^2)$ – медленные алгоритмы сортировки (сортировка выбором);
- $O(n!)$ – очень медленные алгоритмы (задача о коммивояжере).

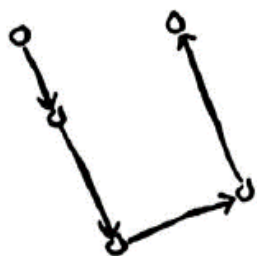


Задача о коммивояжере

для 5 городов потребует 120 операции

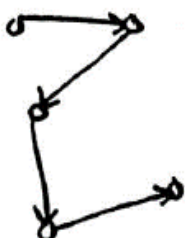


ГОРОДА	ОПЕРАЦИИ
6	720
7	5040
8	40320
...	...
15	1307674368000
...	...
30	26525285181211068636308480000000



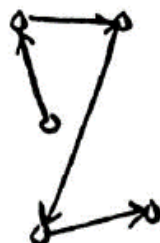
120
миль

или



103
мили

или



133
мили

и т. д.


```
def binary_search(list, item):  
    low = 0  
    high = len(list)-1  
  
    while low <= high:  
        mid = (low + high) // 2  
        guess = list[mid]  
        if guess == item:  
            return mid  
        if guess > item:  
            high = mid - 1  
        else:  
            low = mid + 1  
    return None
```

В переменных low и high хранятся границы той части списка, в которой выполняется поиск

Пока эта часть не сократится до одного элемента ...
... проверяем средний элемент

Значение найдено

Много

Мало

Значение не существует

```
my_list = [1, 3, 5, 7, 9]
```

А теперь протестируем функцию!

```
print binary_search(my_list, 3) # => 1  
print binary_search(my_list, -1) # => None
```

Вспомните: нумерация элементов начинается с 0. Второй ячейке соответствует индекс 1

"None" в Python означает "ничто". Это признак того, что элемент не найден