# Алгоритмы поиска

Шовкопляс Григорий

Введение в алгоритмы и структуры данных


# Задача поиска в массиве и ее решения

# Задача поиска

- Дан массив (например)
- Требуется найти в нем
    - Содержится элемент равный X
    - Минимальный элемент неменьший X
    - Сколько раз содержится элемент X
- Как это сделать?
- Линейный поиск за O(n)
- Если массив не отсортирован, быстрее нельзя
- А если отсортирован?

# Двоичный поиск

- Петя и Вася играют в игру
- Петя загадал число, а Вася его угадывает
- Вася может назвать любое число, а Петя ответит больше, меньше или равно оно загаданного
- Вася хочет как можно быстрее угадать число

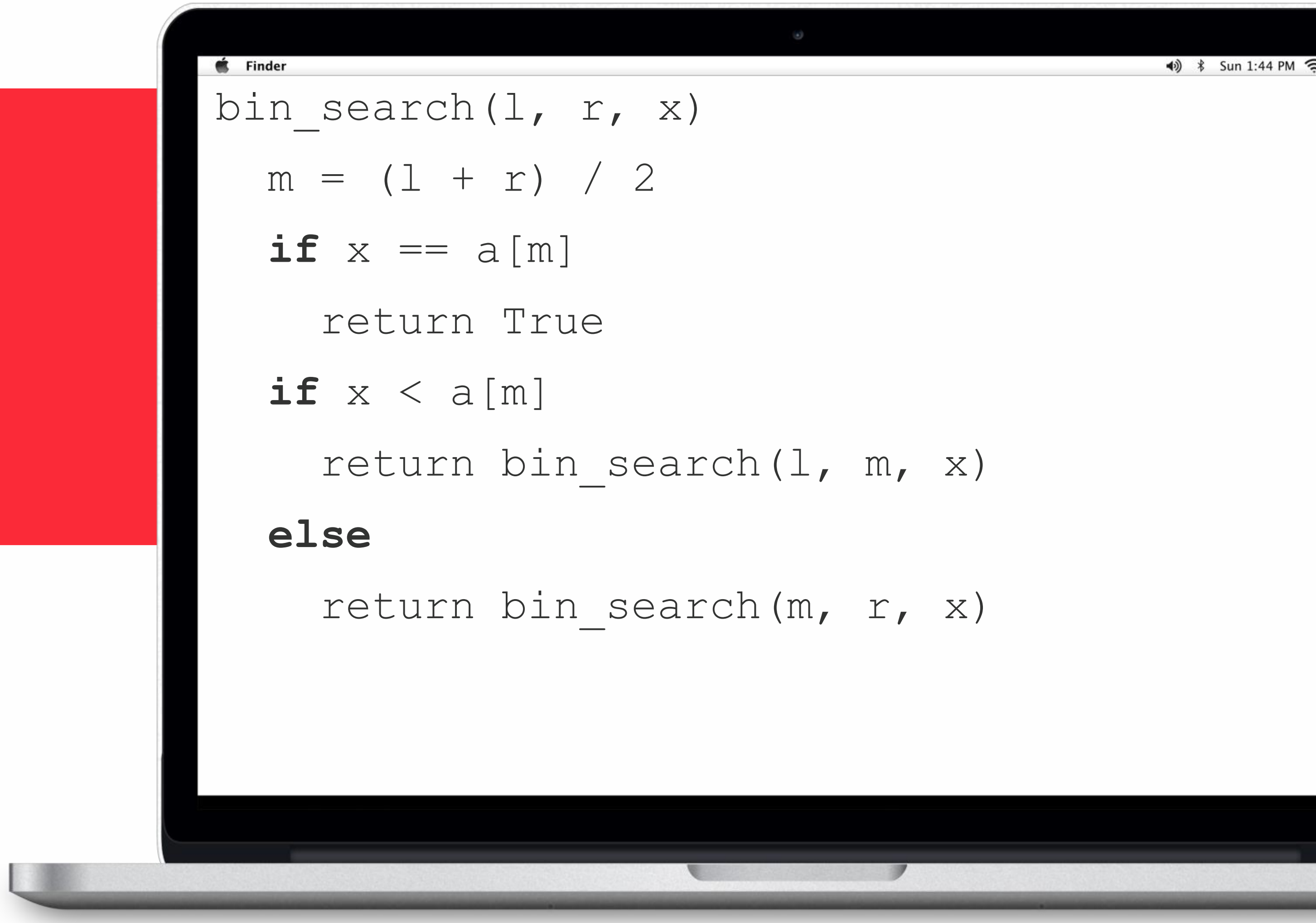
# Двоичный поиск

- Дан отсортированный массив (Если что отсортируем)
- Требуется проверить есть ли в нем конкретное число
- Та же логика, что с игрой
- За сколько работает?
- Почему это не всегда лучше линейного поиска?

# Двоичный поиск

Псевдокод

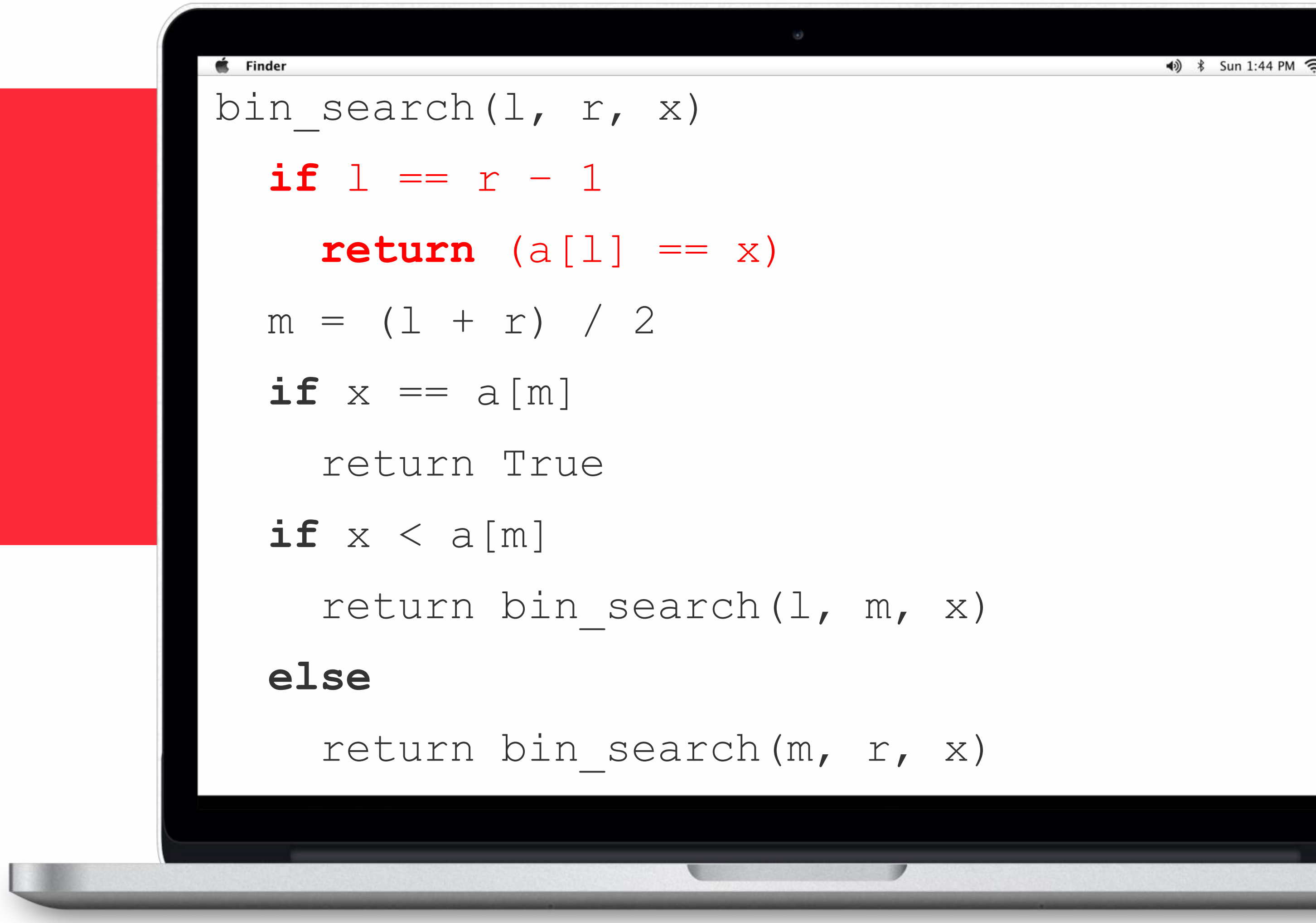
Инвариант: a[l] <= x < a[r]!



# Двоичный поиск

Терминальное условие☺

Инвариант: a[l] <= x < a[r]!



# Левое и правое вхождение

- Можно ли получить больше информации?
- Левое вхождение (нижняя граница) – такое наименьшее i, что $a[i] \geq x$
- Правое вхождение (правая граница) – такое наименьшее i, что $a[i] > x$

# Левое вхождение

Псевдокод

Инвариант: a[l] <= x < a[r]!

```
lower_bound(l, r, x)
  if l == r - 1
    return l
  m = (l + r) / 2
  if x <= a[m]
    return lower_bound(l, m + 1, x)
  else
    return lower_bound(m + 1, r, x)
```

# Правое вхождение

Псевдокод

Инвариант: a[l] <= x < a[r]!

```
upper_bound(l, r, x)
  if l == r - 1
    return l
  m = (l + r) / 2
  if x < a[m]
    return upper_bound(l, m + 1, x)
  else
    return upper_bound(m + 1, r, x)
```

# Левое и правое вхождение

- То есть нужно каждый раз писать копипасту?
- Нет, можно выразить одно вхождение через другое
- upper_bound(x) = lower_bound(x + 1)
- Для целых чисел

# Инвариант

- У нас был инвариант $a[l] \leq x < a[r]$
- Какие проблемы?
- Другие инварианты:
  - $a[l] \leq x \leq a[r]$
  - $a[l] < x \leq a[r]$
    - “Положим” $a[-1] = -\infty, a[n] = \infty$
  - $a[l] < x < a[r]$
- Тогда, какой инвариант самый удобный?

# Левое вхождение

Инвариант: a[l] < x <= a[r]!

- Инвариант выполняется по умолчанию
- Никогда не обращаемся к a[-1] или a[n]
- Запускаем как lower_bound(-1, n, x)

```
lower_bound(l, r, x)
  if l == r - 1
    return r
  m = (l + r) / 2
  if x <= a[m]
    return lower_bound(l, m, x)
  else
    return lower_bound(m, r, x)
```

# Левое вхождение

Нерекурсивный вариант

```
lower_bound(x)
  l = -1
  r = n
  while l < r - 1
    m = (l + r) / 2
    if x <= a[m]
      r = m
    else
      l = m
  return r
```

# Число вхождений

- Как узнать сколько раз элемент X встречается в массиве?
- $\forall i \in [lower\_bound(x); upper\_bound(x))\ a[i] = x$
- cnt(x) = upper_bound(x) – lower_bound(x)

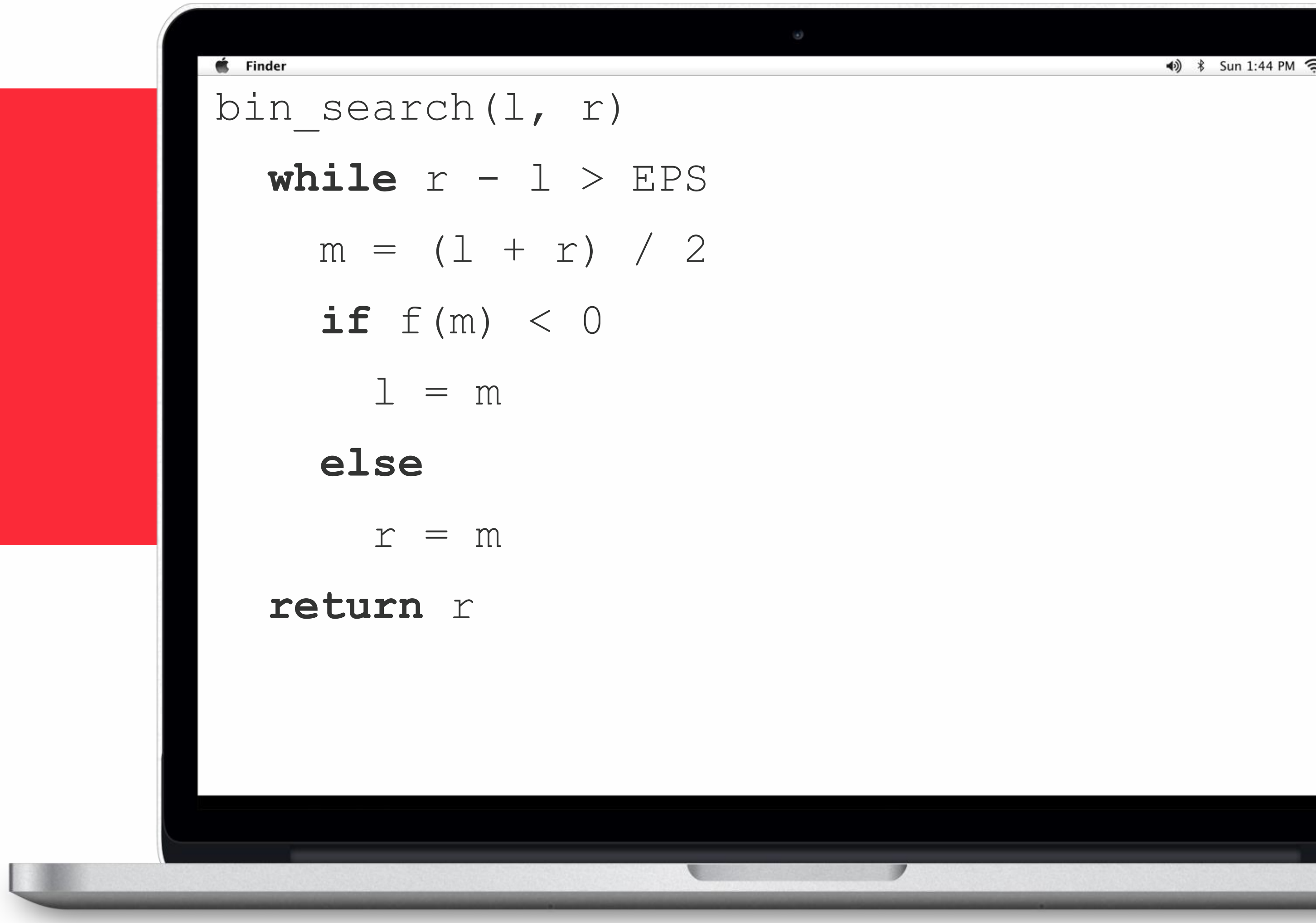

# Вещественный двоичный поиск

# Вещественный двоичный поиск

- Хотим найти такой x, что f(x) = 0
- Какие условия накладываются на функцию?
  - Функция монотонная (можно нестрого)
  - $f(l) \leq 0$ и $f(r) \geq 0$

# Вещественный двоичный поиск

Сразу псевдокод




```
bin_search(l, r)
  while r - l > EPS
    m = (l + r) / 2
    if f(m) < 0
      l = m
    else
      r = m
  return r
```

# Вещественный двоичный поиск

Можно искать не только корень функции, но и любое значение, если $f(l) \leq y$ и $f(r) \geq y$

```
bin_search(y, l, r)
  while r - l > EPS
    m = (l + r) / 2
    if f(m) < y
      l = m
    else
      r = m
  return r
```

# Вещественный двоичный поиск

На самом деле while зло

Пишем всегда for!

```
bin_search(y, l, r)
  while r - l > EPS
  for i = 0 to ITN
    m = (l + r) / 2
    if f(m) < y
      l = m
    else
      r = m
  return r
```

# Вещественный двоичный поиск

- За сколько работает?
  - $O(\log_2 \frac{r-l}{EPS})$
- А что делать, если условие $f(l) \leq y$ и $f(r) \geq y$ не выполняется?
- Или мы не знаем никаких l и r вообще?
- `l = -1; while(f(l) > y) l *= 2`
- `r = 1; while(f(r) < y) r *= 2`

- Если функция убывает, то наоборот

# Двоичный поиск по ответу

# Пример задачи

- Есть аллея длины L
- Фонарь в точке X освещает промежуток (X-R; X+R)
- Требуется поставить n фонарей так, чтобы осветить всю аллею и мощность фонарей была минимальна
- Если внимательно присмотреться, тут есть функция!

# Пример задачи-2

- На прямой расположены стойла (в конкретных точках, $n$ штук)
- Необходимо расставить $K$ коров так, чтобы минимальное расстояние между коровами было как можно больше

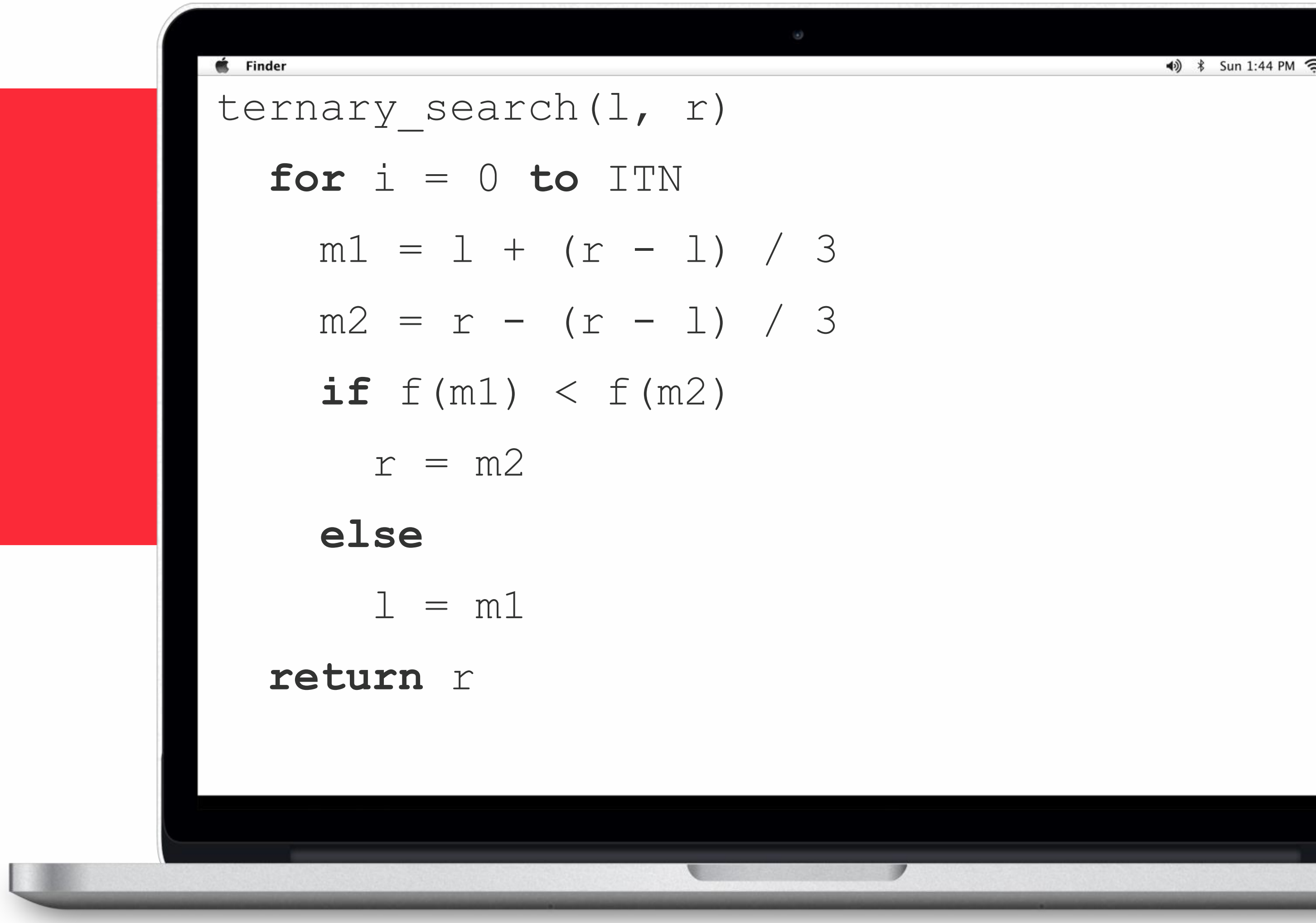
# Троичный поиск

# Троичный поиск

- Пусть есть унимодальная функция
- Можно найти ее экстремум
- Раз в названии три, будем делить на три части!

# Троичный поиск

Сразу псевдокод



```
ternary_search(l, r)
  for i = 0 to ITN
    m1 = l + (r - l) / 3
    m2 = r - (r - l) / 3
    if f(m1) < f(m2)
      r = m2
    else
      l = m1
  return r
```

# Троичный поиск

- За сколько работает?
  - $O(\log_{\frac{3}{2}}\frac{r-l}{EPS})$
- Можно ли улучшить основание логарифма?
  - $m_1 = \frac{l+r}{2} - \varepsilon;\ m_2 = \frac{l+r}{2} + \varepsilon$
- А если функция очень тяжело вычисляема?
  - Сечение Фибоначчи
  - Золотое сечение

# Десерт

# Интерполяционный поиск

- А как мы ищем слово в словаре?
- Не двоичным же поиском

Все!