Сортировки и О-нотация

Шовкопляс Григорий

Введение в алгоритмы и структуры данных

Перед началом использования

Правила курса

- Около 16 занятий
    - Лекция
    - Практика + ДЗ
- Экзамена не будет
- Оценка ставится в зависимости от процента сдачи домашних заданий
    - 50% — 3
    - 65% — 4
    - 80% — 5

Правила приема домашних заданий

- Контест на codeforces.com
- Около 4-6 задач
- Сдача задач:
  - Прислать ссылку на посылку в тестирующей системе на портале в соответствующей форме.
  - Проверяем код на адекватность (кодстайл, реализован нужный алгоритм и т.п.)
  - На ссылку ответ: зачтено или замечания
  - Soft deadline — среда 23:55
    - Гарантируется, что в течение четверга будет проверено
  - Hard deadline — пятница 23:55
    - После засчитываться не будет
- **Списывание и копирование чужого кода запрещено!**

Общая информация по сдаче задач

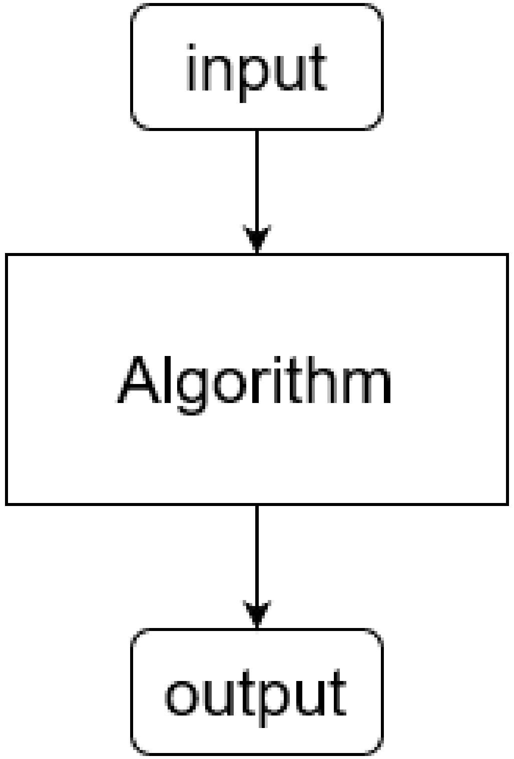
- Задача считается сданной, в тот момент, когда соответствующая ссылка отправлена
- Если есть одобрительная причина сдвинуть дедлайны, пишите об этом, мы готовы и хотим идти на встречу в таких ситуациях
- Для связи: Discord

- *NEW: Будет возможность сдать некоторые темы экстерном*

Алгоритмы и как их оценивать

Что такое алгоритм?

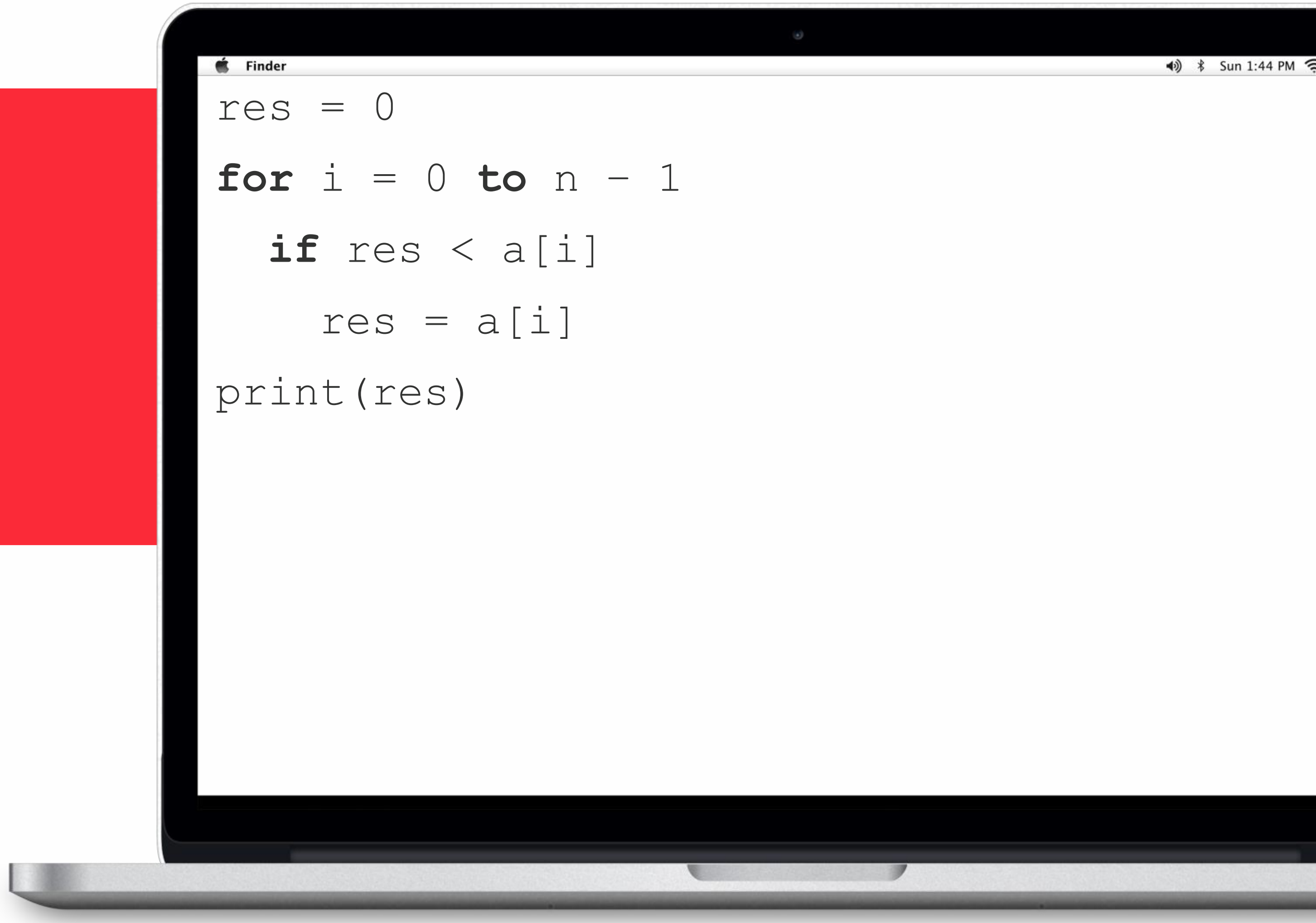
- «Некоторая последовательность действий, приводящая к некоторому результату»
- Опустим многопоточность, откуда берутся входные данные и подобные усложнения



Пример алгоритма

Поиск максимума в массиве натуральных чисел

input: Массив a длины n, состоящий из натуральных чисел



Как оценивать алгоритм?

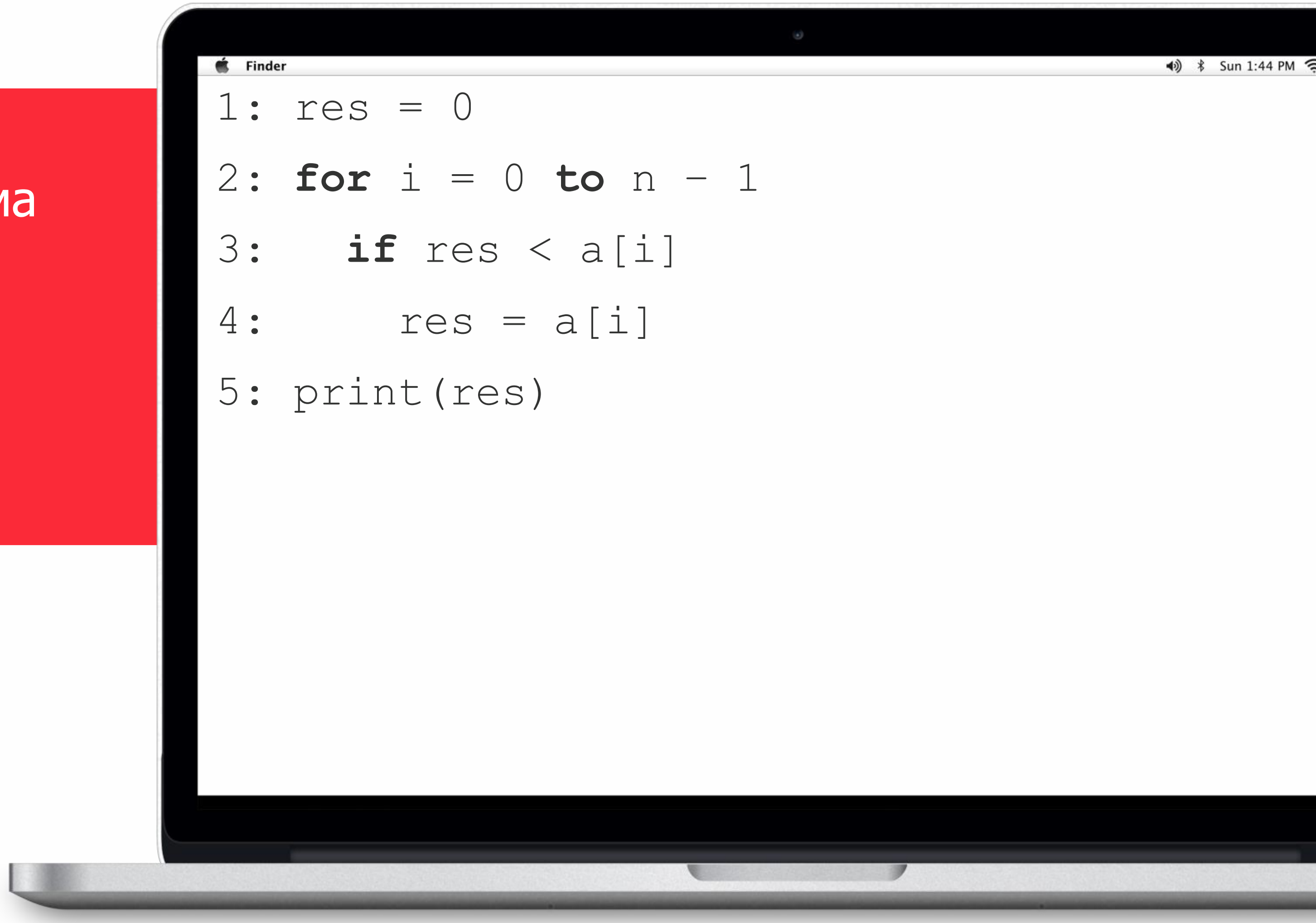
- Время работы
  - Также полезно оценивать память, а иногда и другие параметры
- В секундах плохо
- Введем абстрактную модель
  - RAM-модель

- Оцениваем число элементарных операций RAM-модели
  - get, арифметические операции…

Оценка времени работы алгоритма

$T(n)$ – время работы алгоритма на входе размера n в худшем случае.

$$T(n) = 5n + 2 = O(n)$$




```
res = 0
for i = 0 to n - 1
  if res < a[i]
    res = a[i]
print(res)
```

O-нотация

- T(n) = O(g(n))
  - $\exists n_0, C = const : \forall n > n_0 \quad T(n) \le C \cdot f(n)$
- 5n + 2 = O(n)
  - $C = 10, n_0 = 2$
- T(n) = Ω(g(n))
  - $\exists n_0, C = const : \forall n > n_0 \quad T(n) \ge C \cdot f(n)$

Сортировки

Сортировка вставками

- На k-ой итерации считаем, что первые k элементов отсортированы
- Ищем место для k+1-го
- После n итераций все ОК

Сортировка вставками

Псевдокод

```
for i = 1 to n-1
  j = i
  while j > 0 and a[j - 1] > a[j]
    swap(a[j - 1], a[j])
    j--
```

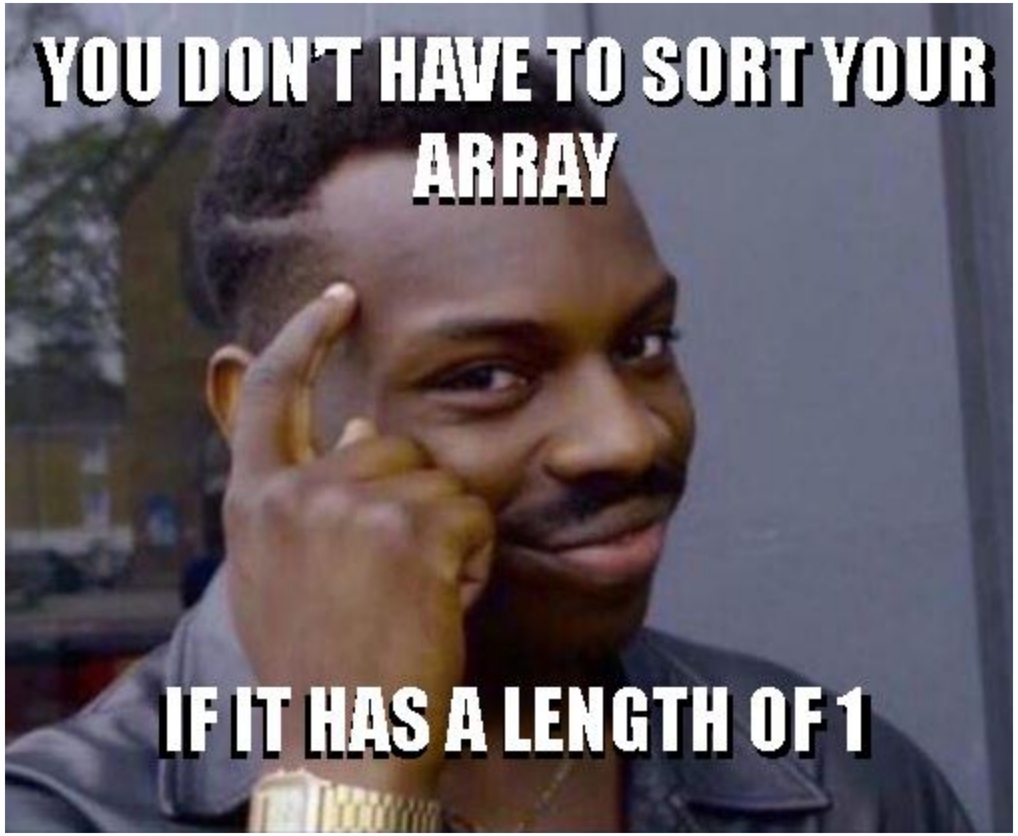
Сортировка вставками

- Почему работает?
- Сколько работает?
    - 1, 2, 3, … n
    - n, n-1, … 2, 1
- $T(n) = O(n^2)$

Как применять оценку на практике

- $1\text{ГГц} = 10^9 c^{-1}$
- 10^9 операций в секунду
  - каких операций?
- $n = O(n^2)$ следовательно при $n = 10^5$; $n^2 = 10^{10}$
- За секунду вряд ли работает

- **Оценка времени примерная!**



Сортировка слиянием

Сортировка слиянием

- Пусть мы умеем за быстро сливать два отсортированных массива в один
- Разделяй и властвуй
    - Поделим массив на две части
    - Рекурсивно их отсортируем
    - Сольем в один
    - Done!

Слияние двух массивов

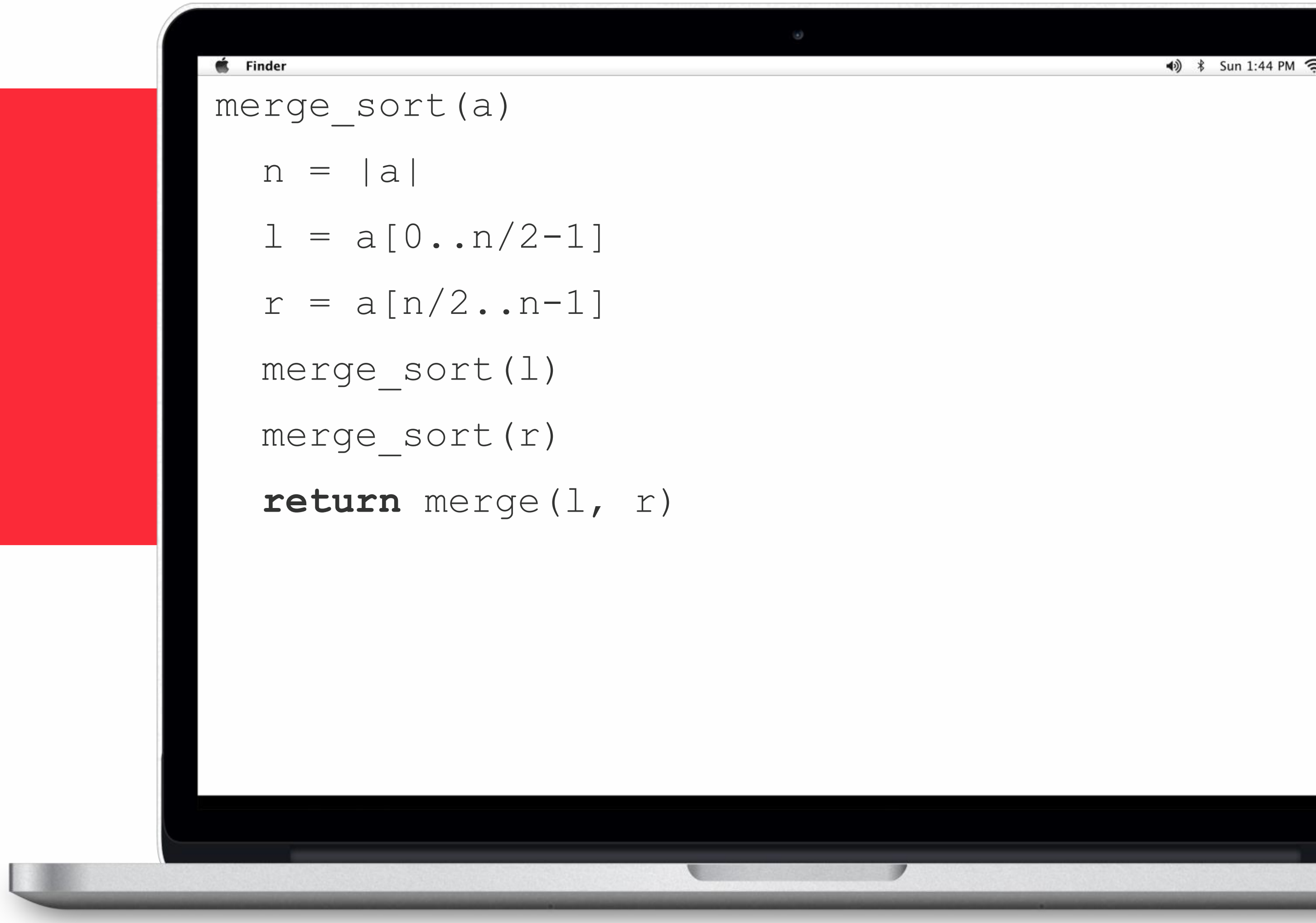
Псевдокод

```
merge(a, b)
  n = |a|
  m = |b|
  i = 0
  j = 0
  while i + j < n + m
    if j == m or (i < n and a[i] < b[j])
      c[i + j] = a[i]
      i++
    else
      c[i + j] = b[j]
      j++
  return c
```

Сортировка слиянием

Псевдокод

Хм.. чего-то не хватает...



Сортировка слиянием

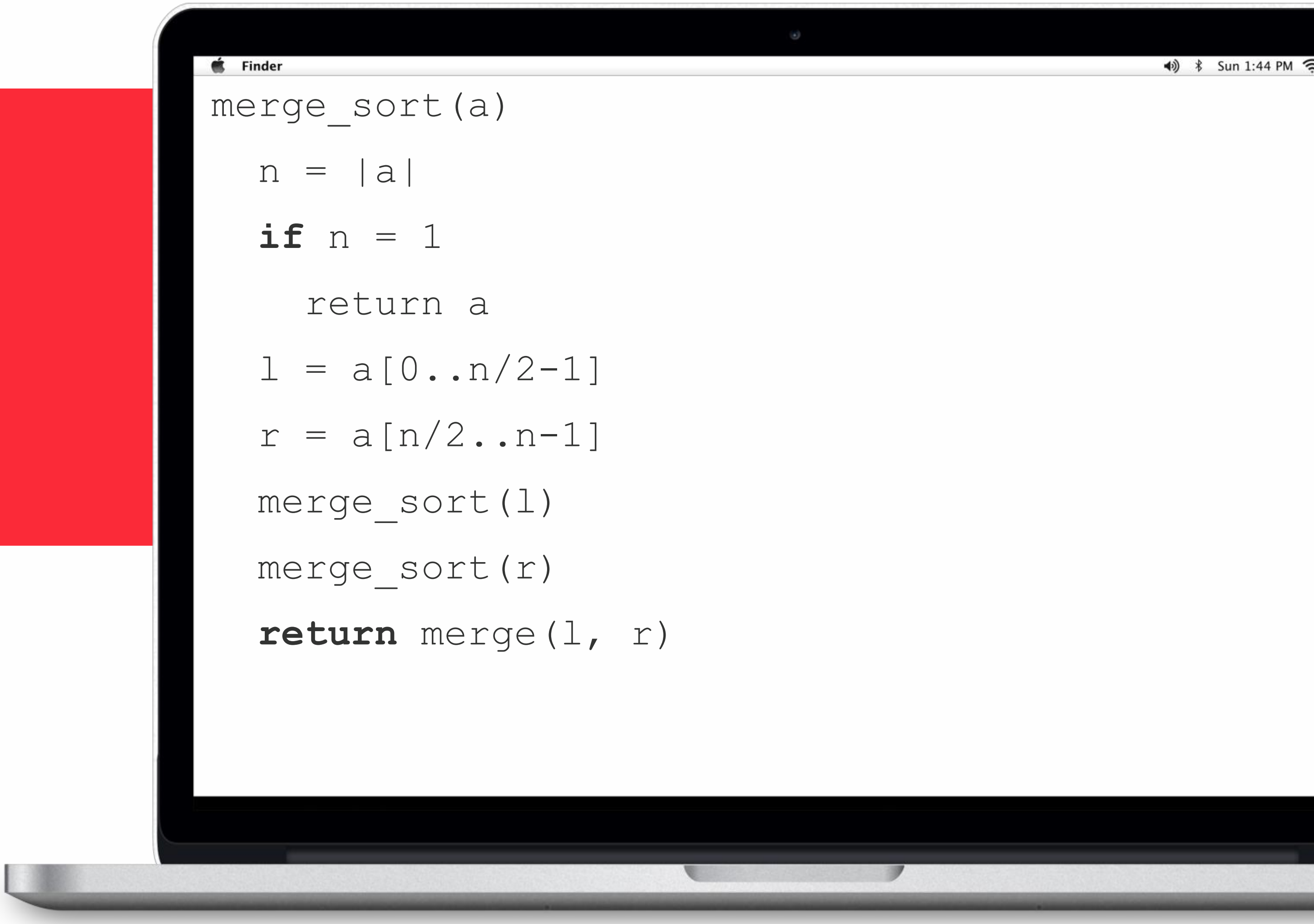
Терминальное условие!

```
merge_sort(a)
  n = |a|
  if n = 1
    return a
  l = a[0..n/2-1]
  r = a[n/2..n-1]
  merge_sort(l)
  merge_sort(r)
  return merge(l, r)
```

Сортировка слиянием

Оценка времени работы:

- Графически
- Рекуррентно
    - «В лоб»
    - Математическая индукция



Лирическое отступление:

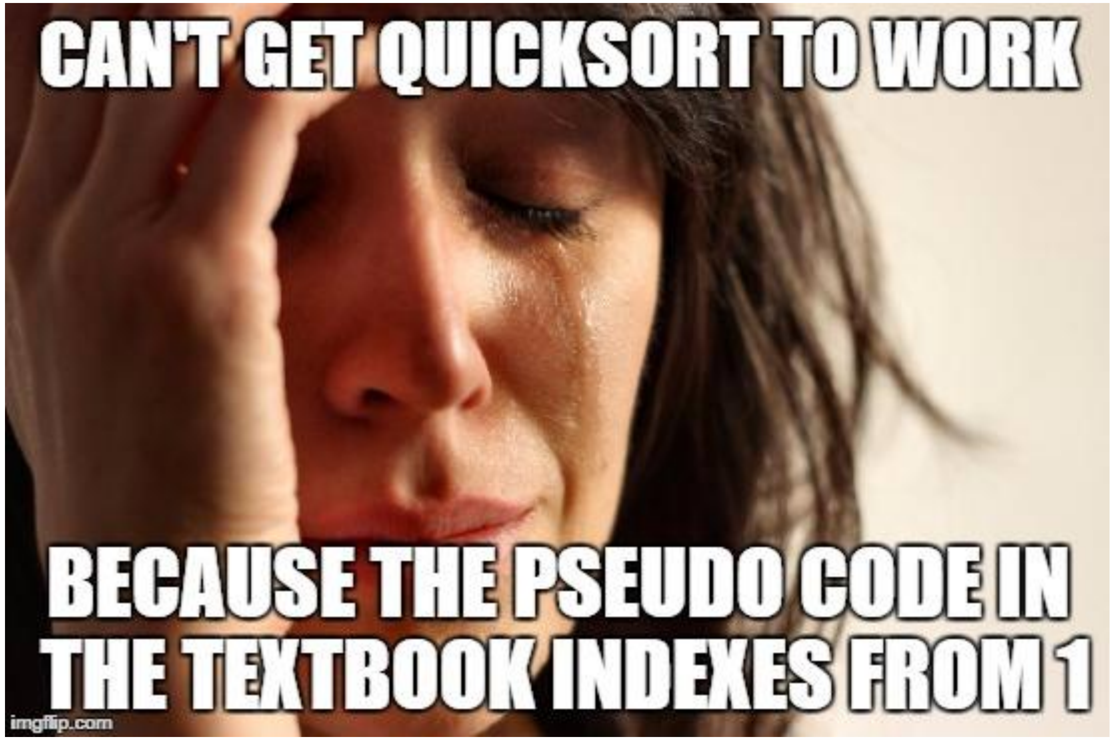
мастер-теорема

Мастер-теорема

- Пусть имеется рекуррентное соотношение:
  - $T(n) = \begin{cases} aT\left(\frac{n}{b}\right) + O(n^c), n > 1 \\ O(1), \quad n = 1 \end{cases}$
  - $a \in \mathbb{N}, b \in \mathbb{R}, b > 1, c \in \mathbb{R}^+$
- Тогда асимптотическое решение имеет вид:
  - Если $c > \log_b a$, то $T(n) = O(n^c)$
  - Если $c = \log_b a$, то $T(n) = O(n^c log n)$
  - Если $c < \log_b a$, то $T(n) = O(n^{\log_b a})$

Свойства сортировки слиянием

- Требуется дополнительная память
- Устойчивость (стабильность)
- Работает всегда за O(nlogn)



Быстрая сортировка

Быстрая сортировка

- *Пусть все числа в массиве различны*
- Можем разделить на две части относительно некоторого числа x
    - < x
    - > x
- Отсортируем их рекурсивно
- Все ок?

Разделение массива

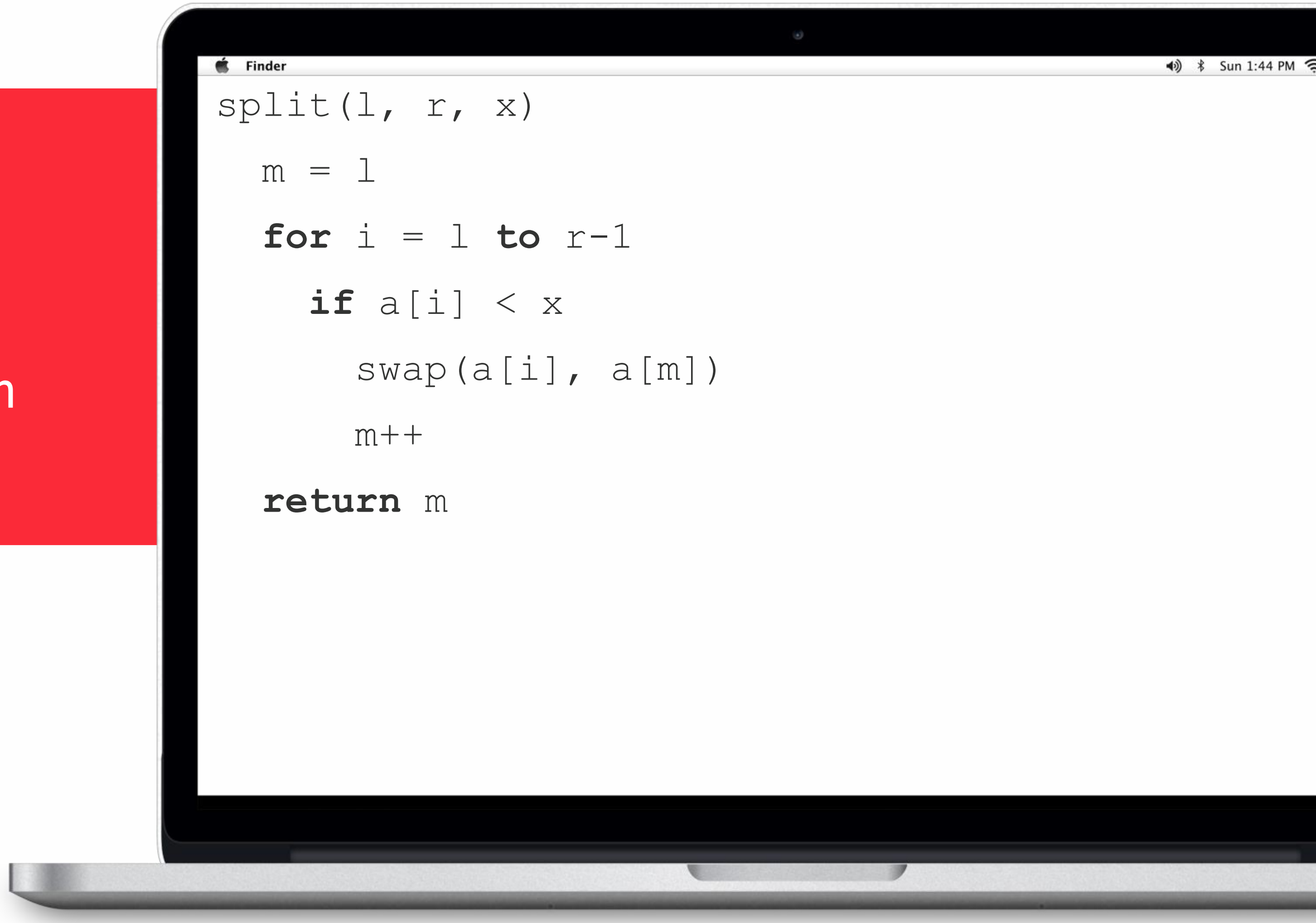
Псевдокод

```
split(a, x)
  l = []
  r = []
  for i = 0 to n-1
    if a[i] < x
      l.add(a[i])
    else
      r.add(a[i])
  return (l, r)
```

Разделение массива

Откажемся от дополнительной памяти

Инвариант: элементы от l до m меньше x



Быстрая сортировка

Псевдокод

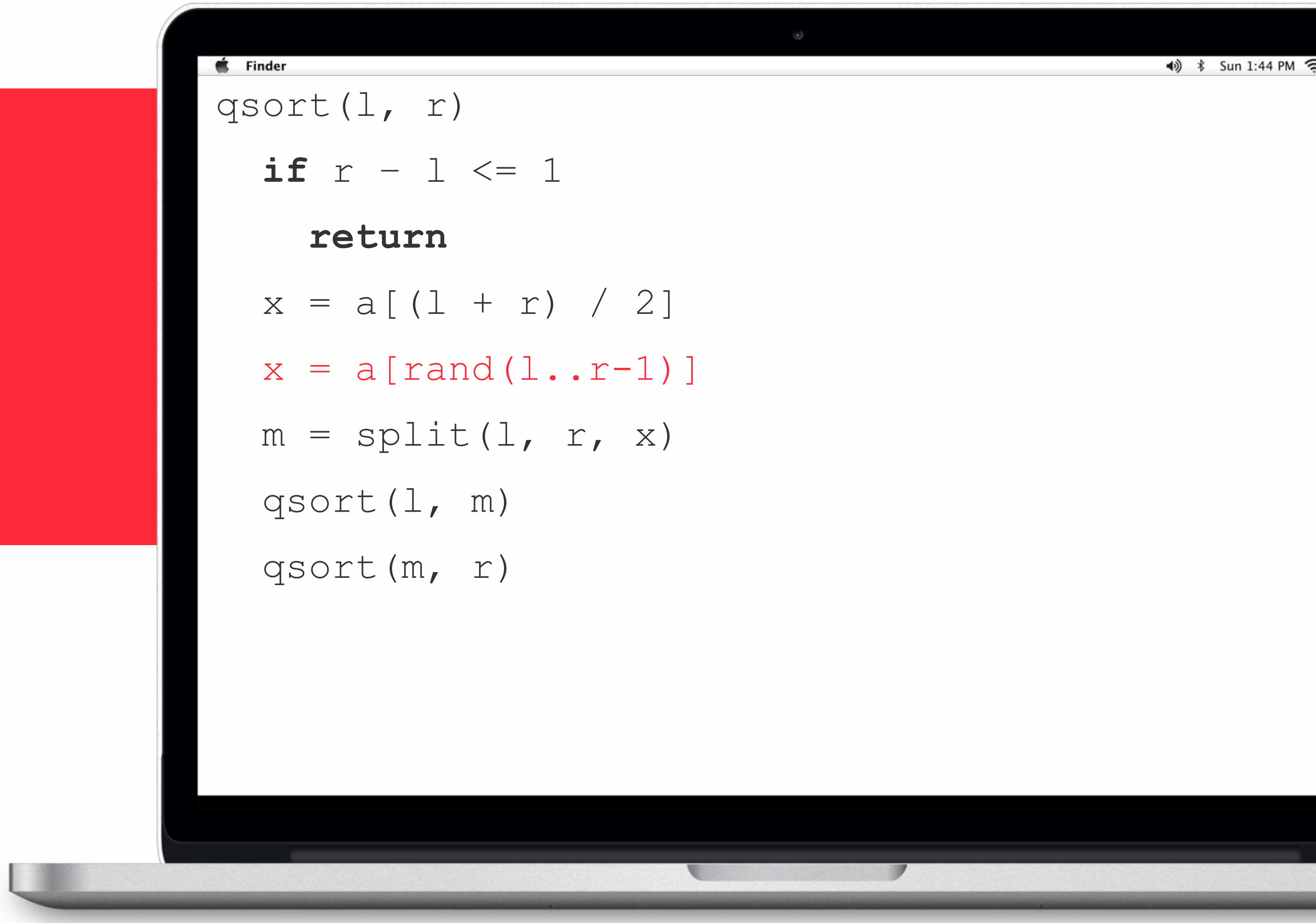
```
qsort(l, r)
  if r - l <= 1
    return
  x = a[(l + r) / 2]
  m = split(l, r, x)
  qsort(l, m)
  qsort(m, r)
```

Оценка времени работы

- В худшем случае O(n^2)
  - Почему называется быстрой?
- Среднее время работы O($nlogn$)
  - Почему?
- Дополним код

Быстрая сортировка

Псевдокод



Свойства быстрой сортировки

- Не требуется дополнительная память
- Не устойчива
- В худшем случае O(n^2)
- На практике работает быстрее сортировки слиянием (при адекватной реализации)

- Кстати, дополнительно нужно рассмотреть случай, если могут быть равные элементы

Все!