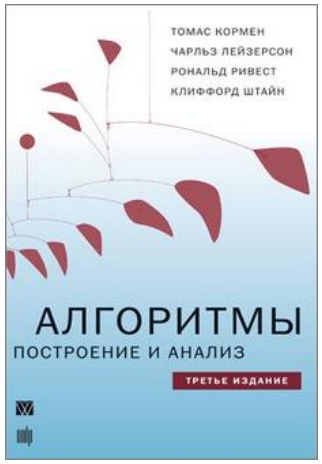
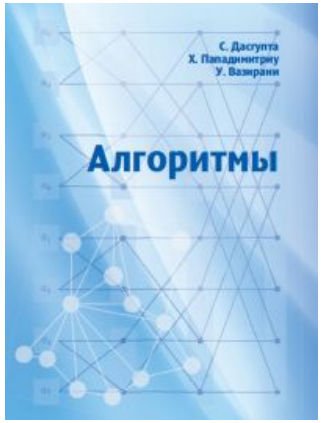
Адрианов Н.М.
Иванов А.Б.

Алгоритмы и структуры данных

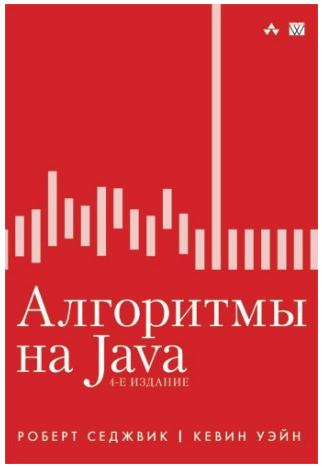
Введение. Сложность алгоритма



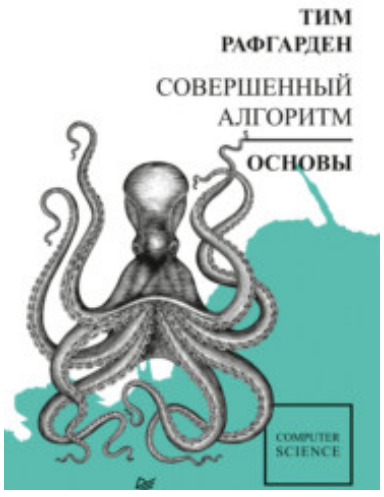
Т. Кормен, Ч. Лейзерсон,
Р. Ривест, К. Штайн
Алгоритмы. Построение и анализ
2-е издание, 2006
3-е издание, 2013

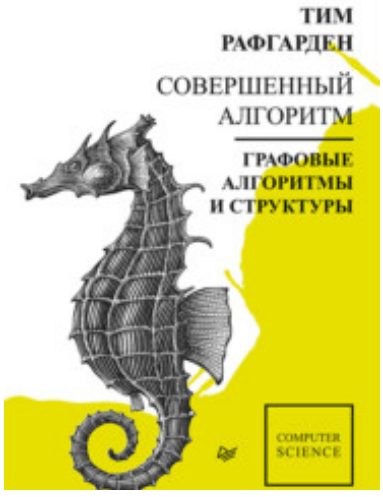


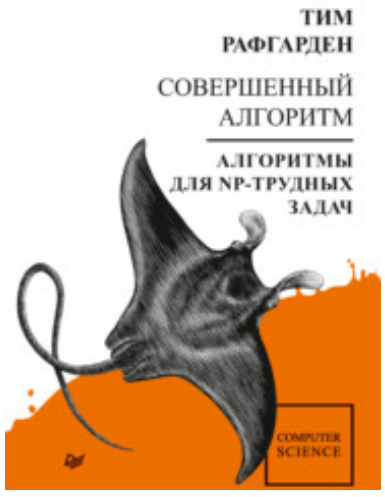
С. Дасгупта, Х. Пападимитриу,
У. Вазирани
Алгоритмы
2014

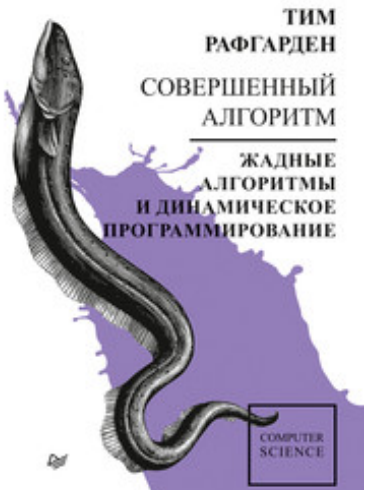


Р. Седжвик, К. Уэйн
Алгоритмы на Java
4-е издание, 2019









Т. Рафгарден

Совершенный алгоритм

Основы

Графовые алгоритмы и структуры данных

Жадные алгоритмы и динамическое программирование

Алгоритмы для NP-трудных задач

2020-2021

Что будет:

- «Разделяй и властвуй»
- Жадные алгоритмы
- Динамическое программирование
- Структуры данных

- Умножение (числа, многочлены, матрицы)
- Сортировки
- Графы
- Строки

Числа Фибоначчи

$$F_0 = 0, \quad F_1 = 1, \quad F_n = F_{n-1} + F_{n-2}$$

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, \ldots$$

Числа Фибоначчи

$$F_0 = 0, \quad F_1 = 1, \quad F_n = F_{n-1} + F_{n-2}$$

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, \ldots$$

```
procedure Fib(n):
  if n = 0:
    return 0
  if n = 1:
    return 1

  return Fib(n-1) + Fib(n-2)
end
```

Числа Фибоначчи

$$F_0 = 0, \quad F_1 = 1, \quad F_n = F_{n-1} + F_{n-2}$$

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, \ldots$$

```
procedure Fib(n):
  if n = 0:
    return 0
  if n = 1:
    return 1

  return Fib(n-1) + Fib(n-2)
end
```

$\sim F_n \approx 2^{0.694n}$

```
procedure Fib(n):
  if n = 0:
    return 0
  if n = 1:
    return 1

  (a,b) = (0,1)
  for i in [2..n]:
    (a,b) = (b,a+b)
  return b
end
```

$\sim n$

Анализ алгоритма

1. Правильно ли работает алгоритм?

2. Какова сложность (время работы в зависимости от n)?

3. Существует ли более быстрый алгоритм?

Сложность в худшем случае и в среднем

Ω_n – множество всех допустимых наборов входных данных размера n.

$T(\omega)$ – сложность (количество операций) алгоритма A на входных данных $\omega \in \Omega_n$.

Сложность **в худшем случае**:

$$T(n) = \max_{\omega \in \Omega_n} T(\omega)$$

Сложность **в среднем** (average):

$$T_{avg}(n) = \frac{1}{|\Omega_n|} \sum_{\omega \in \Omega_n} T(\omega)$$

O-нотация

$$f,\, g : \mathbb{N} \to \mathbb{R}_+ = \{x \in \mathbb{R} \,|\, x > 0\}$$

$f = O(g)$ (f растет не быстрее g),
если существует $c \in \mathbb{R}_+$, что $f(n) \leq c \cdot g(n)$.

$f = \Omega(g)$ (f растет не медленнее g),
если существует $c \in \mathbb{R}_+$, что $f(n) \geq c \cdot g(n)$.

$f = \Theta(g)$ (f и g имееют одинаковый порядок роста),
если $f = O(g)$ и $g = O(f)$.

Сортировка массива

Различная сложность алгоритмов решения одной и той же задачи хорошо иллюстрируется алгоритмами сортировки. Начнем с них и мы.

А зачем вообще нужно сортировать массивы?

Поиск элемента в массиве

Дано: массив `a[]`, элемент `x`
Требуется: определить, содержится ли `x` в `a[]`

Решение: перебрать все элементы `a[]`, сравнивая их с `x`
Сложность: $O(n)$

Бинарный поиск

Дано: **отсортированный** массив `a[]`, элемент `x`
Требуется: определить, содержится ли `x` в `a[]`

- Делим массив пополам
- Сравниваем `x` с центральным элементом `a[mid]`
  - `x` $=$ `a[mid]`? – нашли
  - `x` $<$ `a[mid]`? – ищем в нижней части массива
  - `x` $>$ `a[mid]`? – ищем в верхней части массива

Бинарный поиск

Пусть $n \leq 2^k$ $(k = \lceil \log n \rceil)$

Итераций: $k + 1$

На каждой итерации: $O(1)$

Итого: $O(\log n)$

```
procedure BinarySearch( a[], x ):
  // a[] - an array from 0 to n-1
  low = 0
  high = n-1
  while low <= high:
    mid = low + (high - low) / 2

    if a[mid] < x:
      low = mid + 1
    else if a[mid] > x:
      high = mid - 1
    else
      return mid
  end
  return -1
end
```

Бинарный поиск – простой алгоритм?..

Donald Knuth (*Искусство программирования*):
бинарный поиск был впервые опубликован в 1946 году, но только в 1962 году была опубликована первая версия без багов.

Jon Bentley (*Жемчужины программированияб 1986*):
только 10% профессиональных программистов смогли реализовать бинарный поиск без ошибок.

Joshua Bloch (2006):
образцовый код в книге Бентли также содержит ошибку (также ошибка была в реализации для языка Java – и оставалась незамеченной почти 10 лет)
http://googleresearch.blogspot.ru/2006/06/extra-extra-read-all-about-it-nearly.html

Бинарный поиск – простой алгоритм?..

```
if (l>u)
    p=-1, break
```

Оператор break завершает внутренний цикл, к которому он относится. Следующее выражение вычисляет значение m (середину диапазона):

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