

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, ...

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, ...

```
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, ...

```
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

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

$f = O(g)$ (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 имеют 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 $a[\text{mid}]$
 - $x = a[\text{mid}]?$ – нашли
 - $x < a[\text{mid}]?$ – ищем в нижней части массива
 - $x > a[\text{mid}]?$ – 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 программирования* 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 (сердину диапазона):

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

СОРТИРОВКА ПУЗЫРЬКОМ (BUBBLE SORT)

3	7	2	6	4	8	1	5
---	---	---	---	---	---	---	---

3	7	2	6	4	8	1	5
---	---	---	---	---	---	---	---

3	2	7	6	4	8	1	5
---	---	---	---	---	---	---	---

3	2	6	7	4	8	1	5
---	---	---	---	---	---	---	---

3	2	6	4	7	8	1	5
---	---	---	---	---	---	---	---

3	2	6	4	7	8	1	5
---	---	---	---	---	---	---	---

3	2	6	4	7	1	8	5
---	---	---	---	---	---	---	---

3	2	6	4	7	1	5	8
---	---	---	---	---	---	---	---

```
procedure BubbleSort( a[] ):  
  // a[] - массив от 0 до n-1  
  for i in [1 .. n-1]:  
    for j in [0 .. n-i- 1]:  
      if (a[j] > a[j+1]):  
        переставить a[j] и a[j+1]  
      end  
    end  
  end  
end  
end
```

СОРТИРОВКА ПУЗЫРЬКОМ (BUBBLE SORT)

3	7	2	6	4	8	1	5
---	---	---	---	---	---	---	---

3	7	2	6	4	8	1	5
---	---	---	---	---	---	---	---

3	2	7	6	4	8	1	5
---	---	---	---	---	---	---	---

3	2	6	7	4	8	1	5
---	---	---	---	---	---	---	---

3	2	6	4	7	8	1	5
---	---	---	---	---	---	---	---

3	2	6	4	7	8	1	5
---	---	---	---	---	---	---	---

3	2	6	4	7	1	8	5
---	---	---	---	---	---	---	---

3	2	6	4	7	1	5	8
---	---	---	---	---	---	---	---

```
procedure BubbleSort( a[] ):  
  // a[] - массив от 0 до n-1  
  for i in [1 .. n-1]:  
    for j in [0 .. n-i- 1]:  
      if (a[j] > a[j+1]):  
        переставить a[j] и a[j+1]  
      end  
    end  
  end  
end  
end
```

Анализ: количество сравнений

$$(n-1) + (n-2) + \dots + 1 = \frac{n(n-1)}{2}$$

$$O(n^2)$$

СОРТИРОВКА ВЫБОРОМ (SELECTION SORT)

3	7	2	6	4	8	1	5
---	---	---	---	---	---	---	---

3	7	2	6	4	5	1	8
---	---	---	---	---	---	---	---

3	1	2	6	4	5	7	8
---	---	---	---	---	---	---	---

3	1	2	5	4	6	7	8
---	---	---	---	---	---	---	---

3	1	2	4	5	6	7	8
---	---	---	---	---	---	---	---

3	1	2	4	5	6	7	8
---	---	---	---	---	---	---	---

2	1	3	4	5	6	7	8
---	---	---	---	---	---	---	---

1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---

```
procedure SelectionSort( a[] ):
  // a[] - массив от 0 до n-1
  for i in [1..n-1]:
    j = индекс максимального элемента в
        a[0..n-i]
    переставить a[j] и a[n-i]
  end
end
```

СОРТИРОВКА ВЫБОРОМ (SELECTION SORT)

3	7	2	6	4	8	1	5
---	---	---	---	---	---	---	---

3	7	2	6	4	5	1	8
---	---	---	---	---	---	---	---

3	1	2	6	4	5	7	8
---	---	---	---	---	---	---	---

3	1	2	5	4	6	7	8
---	---	---	---	---	---	---	---

3	1	2	4	5	6	7	8
---	---	---	---	---	---	---	---

3	1	2	4	5	6	7	8
---	---	---	---	---	---	---	---

2	1	3	4	5	6	7	8
---	---	---	---	---	---	---	---

1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---

```
procedure SelectionSort( a[] ):  
  // a[] - массив от 0 до n-1  
  for i in [1..n-1]:  
    j = индекс максимального элемента в  
      a[0..n-i]  
    переставить a[j] и a[n-i]  
  end  
end
```

Анализ: количество сравнений

$$(n-1) + (n-2) + \dots + 1 = \frac{n(n-1)}{2}$$

$$O(n^2)$$

СОРТИРОВКА ВСТАВКАМИ (INSERTION SORT)

3	7	2	6	4	8	1	5
---	---	---	---	---	---	---	---

3	7	2	6	4	8	1	5
---	---	---	---	---	---	---	---

3		7	6	4	8	1	5
---	--	---	---	---	---	---	---

	3	7	6	4	8	1	5
--	---	---	---	---	---	---	---

2	3	7	6	4	8	1	5
---	---	---	---	---	---	---	---

--	--	--	--	--	--	--	--

--	--	--	--	--	--	--	--

--	--	--	--	--	--	--	--

```
procedure InsertionSort( a[] ):
  // a[] - массив от 0 до n-1
  for i in [1..n-1]:
    j = i, t = a[j]
    while j > 0 && t < a[j-1]:
      a[j] = a[j-1]
      j = j-1
    end
    a[j] = t
  end
end
```

СОРТИРОВКА ВСТАВКАМИ (INSERTION SORT)

3	7	2	6	4	8	1	5
---	---	---	---	---	---	---	---

3	7	2	6	4	8	1	5
---	---	---	---	---	---	---	---

3		7	6	4	8	1	5
---	--	---	---	---	---	---	---

	3	7	6	4	8	1	5
--	---	---	---	---	---	---	---

2	3	7	6	4	8	1	5
---	---	---	---	---	---	---	---

--	--	--	--	--	--	--	--

--	--	--	--	--	--	--	--

--	--	--	--	--	--	--	--

```
procedure InsertionSort( a[] ):
  // a[] - массив от 0 до n-1
  for i in [1..n-1]:
    j = i, t = a[j]
    while j > 0 && t < a[j-1]:
      a[j] = a[j-1]
      j = j-1
    end
    a[j] = t
  end
end
```

В лучшем случае сравнений

$$1 + 1 + \dots + 1 = n - 1$$

в худшем случае

$$1 + 2 + \dots + (n - 1) = \frac{n(n - 1)}{2}$$

СОРТИРОВКА СЛИЯНИЕМ (MERGE SORT)

«РАЗДЕЛЯЙ И ВЛАСТВУЙ» (DIVIDE AND CONQUER)

- Разделим массив на 2 части размера $n/2$
- Отсортируем обе части (2 рекурсивных вызова)
- Выполним процедуру слияния: объединяем отсортированные части таким образом, чтобы получить полностью отсортированный массив

СОРТИРОВКА СЛИЯНИЕМ

Ключевая процедура –
слияние (merge)

Особенность: слияние требует дополнительный массив `aux` (auxiliary - вспомогательный)

Выделяем его сразу, чтобы не делать этого при каждом вызове

```
procedure MergeSort( a[], aux[],
                    low, high ):
    if (high <= low):
        return

    mid = low + (high - low) / 2

    MergeSort(a, aux, low, mid)
    MergeSort(a, aux, mid + 1, high)

    Merge(a, aux, low, mid, high)
end

procedure MergeSort( a[] ):
    // a[] - массив от 0 до n-1
    aux = создать дополнительный
        массив [0..n-1]
    MergeSort(a, aux, 0, n-1)
end
```

СОРТИРОВКА СЛИЯНИЕМ

3	7	2	6	4	8	1	5
---	---	---	---	---	---	---	---

--	--	--	--	--	--	--	--

--	--	--	--	--	--	--	--

```
procedure Merge( a[], aux[],
                 low, mid, high ):
  for k in [low..high]:
    aux[k] = a[k]

  i = low, j = mid + 1
  for k in [low..high]:
    if i > mid:
      a[k] = aux[j++]
    else if j > high:
      a[k] = aux[i++]
    else if aux[j] < aux[i]:
      a[k] = aux[j++]
    else:
      a[k] = aux[i++]
  end
end
```

СОРТИРОВКА СЛИЯНИЕМ: СЛОЖНОСТЬ

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$

СОРТИРОВКА СЛИЯНИЕМ: СЛОЖНОСТЬ

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$

$$T(n) = O(n \log n)$$

РАЗДЕЛЯЙ И ВЛАВСТВУЙ: ОСНОВНОЙ МЕТОД*

Теорема. Если сложность алгоритма определяется рекуррентным соотношением

$$T(n) = aT\left(\frac{n}{b}\right) + O(n^d),$$

то

$$T(n) = \begin{cases} O(n^d \log n), & \text{если } a = b^d \\ O(n^d), & \text{если } a < b^d \\ O(n^{\log_b a}), & \text{если } a > b^d. \end{cases}$$

НУЖНЫ ЛИ БЫСТРЫЕ АЛГОРИТМЫ?

Так ли важно – $O(n^2)$ или $O(n \log n)$?

Предположим: РС выполняет 10^8 операций в секунду,
гипотетический суперкомпьютер – 10^{12} операций.

	n	n^2	$n \log n$
10^3	10^3	10^6	10^4
PC:		0 сек	0 сек
Supercomputer:		0 сек	0 сек
10^6	10^6	10^{12}	$2 \cdot 10^7$
PC:		2.8 часа	0.2 сек
Supercomputer:		1 сек	0 сек
10^9	10^9	10^{18}	$3 \cdot 10^{10}$
PC:		317 лет	5мин
Supercomputer:		12 суток	0 сек

ЗАКОН МУРА

Если бы авиапромышленность в последние 25 лет развивалась столь же стремительно, как компьютерная техника, то сейчас самолёт Boeing 767 стоил бы 500\$ и совершал облёт земного шара за 20 минут, затрачивая при этом пять галлонов топлива.

Scientific American, 1983

ЗАКОН МУРА - СЛЕДСТВИЕ

Можно ли можно пренебречь эффективностью алгоритмов, так как производительность компьютеров растет, в соответствии с законом Мура, экспоненциально?

Наоборот, закон Мура увеличивает важность эффективных алгоритмов!

Вместе с ростом производительности растут объемы памяти и размеры входных данных, которые хочется обрабатывать.

НИЖНЯЯ ОЦЕНКА СЛОЖНОСТИ СОРТИРОВКИ

Теорема. Любой детерминированный алгоритм сортировки *сравнением* имеет сложность в *худшем* случае $\Omega(n \log n)$.

Количество перестановок в массиве из n элементов = $n!$

Работу алгоритма на различных входных данных можно представить в виде бинарного дерева. Каждое ветвление – сравнение элементов массива. Если при любых входных данных количество сравнений не больше S , то глубина дерева не больше S , а количество конечных узлов – не больше 2^S .

$$2^S \geq n! \quad \Rightarrow \quad S = \Omega(n \log n)$$

Формула Стирлинга: $n! \sim \sqrt{2\pi n} \left(\frac{n}{e}\right)^n$

или грубая оценка $n! > \left[\frac{n}{2}\right]^{\left[\frac{n}{2}\right]}$

СВОЙСТВА АЛГОРИТМОВ СОРТИРОВКИ

- Сложность
- Необходимая память
- Устойчивость

Сортировка слиянием:

- $O(n \log n)$
- $\geq N$
- Устойчивый

БЫСТРАЯ СОРТИРОВКА (QUICKSORT)



Tony Hoare в 1959 году в СССР

Интерес сэра Тони Хоара к компьютерным вычислениям проснулся в начале пятидесятых годов, когда он изучал философию (наряду с латинским и греческим) в Оксфордском университете, под руководством Джона Лукаса. Во время своей службы в Королевском военно-морском флоте изучал русский язык. В 1959 году, будучи аспирантом Московского государственного университета, он изучал машинный перевод языков и теорию вероятностей, в школе А.Н. Колмогорова. Для эффективного поиска слов в словаре, он разработал известный алгоритм «быстрой сортировки».

<http://theoryandpractice.ru/presenters/16008-toni-khoar>

БЫСТРАЯ СОРТИРОВКА (QUICKSORT)

3	7	2	6	4	8	1	5
---	---	---	---	---	---	---	---

--	--	--	--	--	--	--	--

--	--	--	--	--	--	--	--

--	--	--	--	--	--	--	--

--	--	--	--	--	--	--	--

--	--	--	--	--	--	--	--

--	--	--	--	--	--	--	--

--	--	--	--	--	--	--	--

```
procedure QuickSort(a[], left, right):  
  // a[] - массив от 0 до n-1  
  pivot = a[left]  
  i = left + 1  
  for j in [left + 1 .. right]:  
    if a[j] < pivot:  
      переставить a[j] и a[i]  
      i++  
    end  
  end  
  переставить a[left] и a[i-1]  
  
  QuickSort(a, left, i - 2)  
  QuickSort(a, i, right)  
end
```

БЫСТРАЯ СОРТИРОВКА (КОРРЕКТНОСТЬ)

БЫСТРАЯ СОРТИРОВКА (QUICKSORT)

Сложность? Выбор опорного элемента?

- В худшем случае: $O(n^2)$
- В лучшем случае (опорный элемент - медиана):

$$T(n) = 2T\left(\left\lceil \frac{n}{2} \right\rceil\right) + O(n) \implies T(n) = O(n \log n)$$

- В «плохом» случае - опорный элемент делит массив в соотношении 99/100: $O(n \log n)$

БЫСТРАЯ СОРТИРОВКА (QUICKSORT)

Теорема. Сложность быстрой сортировки «в среднем» – $O(n \log n)$ (при случайном выборе опорного элемента).

- Пространство элементарных событий $\Omega = \{\text{все возможные последовательности опорных элементов}\}$
- Случайная величина $X(\omega) = \text{"количество выполняемых сравнений при последовательности опорных элементов } \omega \text{"}$
- Наша цель:

$$M[X] = O(n \log n)$$

БЫСТРАЯ СОРТИРОВКА (QUICKSORT)

- z_i - i -ая порядковая статистика
- $X_{ij}(\omega)$ - количество сравнений z_i и z_j
- По определению

$$X(\omega) = \sum_{i < j} X_{ij}(\omega)$$

- Из линейности математического ожидания

$$M[X] = \sum_{i < j} M[X_{ij}]$$

- Или

$$M[X] = \sum_{i < j} P[X_{ij} = 1]$$

БЫСТРАЯ СОРТИРОВКА (QUICKSORT)

$$P[X_{ij} = 1] = \frac{2}{j - i + 1}$$

$$M[X] = \sum_{i < j} P[X_{ij} = 1]$$

↓

$$\begin{aligned} M[X] &= \sum_{i < j} P[X_{ij} = 1] = \sum_{i < j} \frac{2}{j - i + 1} = \\ &= \sum_{i=1}^{n-1} \sum_{j=i+1}^n \frac{2}{j - i + 1} \leq 2n \sum_{k=2}^n \frac{1}{k} \leq 2n \ln n \end{aligned}$$