Элементы теории алгоритмов

Элементы теории алгоритмов

§ 34. Уточнение понятия алгоритма

Зачем уточнять определение?

Алгоритм – точный набор инструкций для исполнителя.

? Всегда ли существует алгоритм?

Конструктивное доказательство: построить алгоритм.

? А если не удалось?

- задача о квадратуре круга
- задача о трисекции угла
- задача об удвоении куба
- вечный двигатель
- …

нестрогие понятия

? Как доказать, что алгоритма не существует?

Зачем уточнять определение?

Задача: алгоритм как математический объект.

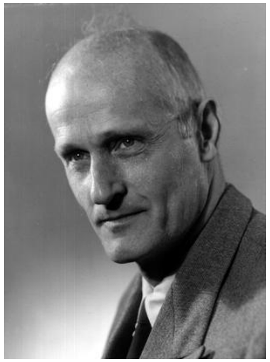
Теория алгоритмов (1930-е):

- доказательство алгоритмической неразрешимости задач
- анализ сложности алгоритмов
- сравнительная оценка качества алгоритмов

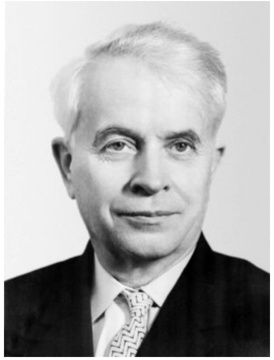
А. Тьюринг

Э. Пост

А. Чёрч



С. Клини



А. Марков

Что такое алгоритм?

Первые алгоритмы – правила арифметических действий:

- объекты – числа
- шаги – операции с однозначными числами

? Что считать шагом?

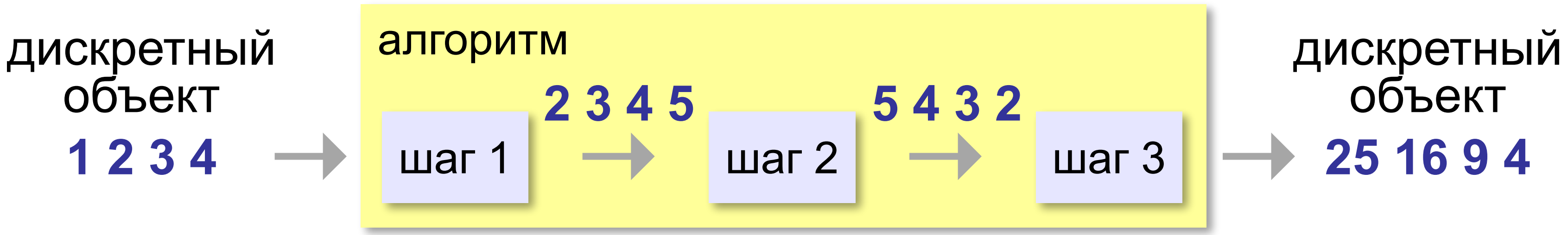
Все объекты можно закодировать как символьные строки:

! Можно рассматривать только алгоритмы обработки **строк**!

Из любого кода можно перевести в двоичный:

! Можно рассматривать только алгоритмы обработки **битовых** строк!

Как работает алгоритм?



- получает на вход дискретный объект
- в результате строит другой дискретный объект (или выдаёт сообщение об ошибке)
- обрабатывает объект по шагам
- на каждом шаге получается новый дискретный объект

Как работает алгоритм?

входное слово: **муха** → **алгоритм** → **слон** :выходное слово

! Любой алгоритм определяет функцию!

т.е. правило преобразования входа в выход

Функция не определена ⇔ алгоритм зацикливается или завершается аварийно.

```
ввод a, b
вывод a*sqrt(b)
```

→ ✗ b < 0

```
ввод a
нц пока да
кц
```

→ ✗ для всех **a**

Эквивалентные алгоритмы

Задают одну и ту же функцию:

```
если a < b то
  M:= a
иначе
  M:= b
все
```

↔

```
M:= b
если a < b то
  M:= a
все
```

Универсальные исполнители

Алгоритм привязан к исполнителю ⇒ идея: построить универсального исполнителя.

Для любого алгоритма для любого исполнителя можно построить эквивалентный алгоритм для **универсального исполнителя**.

- если есть алгоритм для универсального исполнителя, то задача разрешима
- если доказано, что нет алгоритма для универсального исполнителя, задача неразрешима

! Любой алгоритм может быть представлен как программа для универсального исполнителя!

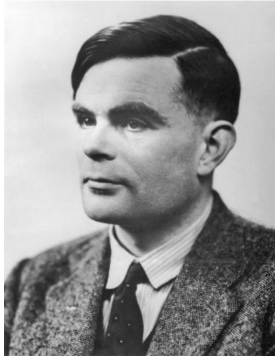
Универсальные исполнители

Алгоритм – это программа для универсального исполнителя.

Модель вычислений:

- «*процессор*» (система команд и способ их выполнения)
- «*память*» (способ хранения данных)
- *язык программирования* (способ записи программ);
- *способ ввода* данных
- *способ вывода* результата

Универсальные исполнители

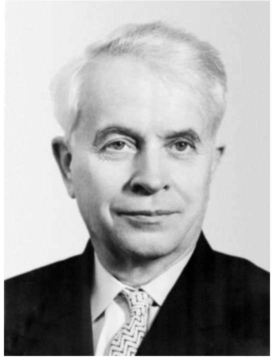


А. Тьюринг

машина Тьюринга

Э. Пост

машина Поста

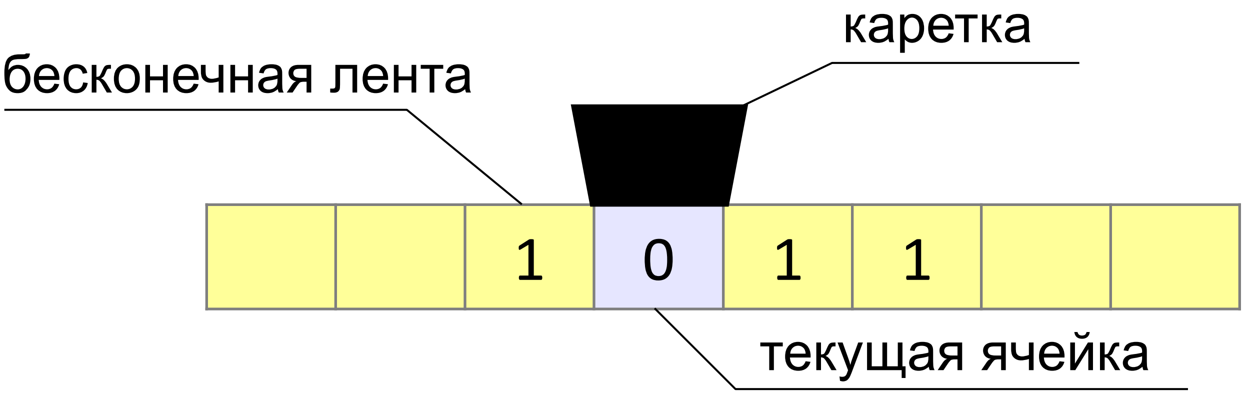


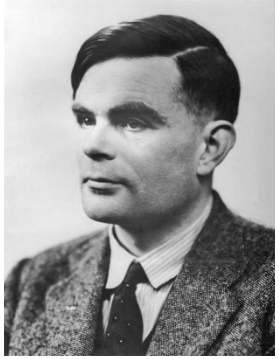
А. Марков

нормальные алгорифмы Маркова

! Все универсальные исполнители эквивалентны!

Машина Тьюринга




А. Тьюринг

- бесконечная лента («память»)
- каретка (запись и чтение)
- программируемый автомат («процессор»)

алфавит: $A = \{a_1, a_2, \ldots, a_N\}$ $A = \{0, 1, \square\}$

пробел

Что такое автомат?

Автомат – это устройство, работающее без участия человека.

Состояние – промежуточная задача, которую решает автомат.

$Q = \{q_1, q_2, \ldots, q_M\}$

начальное состояние

q_0 – остановка автомата

Программа для машины Тьюринга

Программа состоит из команд:

- записать символ a_i в текущую ячейку
- переместить каретку → ← • (не перемещать)
- перейти в состояние q_j

A = {0, 1, □}

$1 \rightarrow q_1$

- записать 1
- переместиться вправо
- перейти в состояние q_1

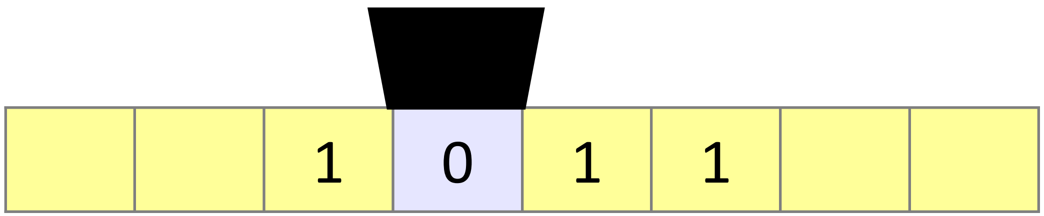
$0 \bullet q_0$

- записать 0
- не перемещать каретку
- останов (q_0)

Программа для машины Тьюринга

Задача. На ленте записано число в двоичной системе счисления. Каретка находится где-то над числом. Требуется увеличить число на единицу.

алфавит: $A = \{0, 1, \square\}$



состояния: q_1 – поиск правого конца слова

подзадачи

q_2 – увеличение числа на 1

Программа для машины Тьюринга

q_1 : поиск конца слова

- если 0, то →
- если 1, то →
- если □, то ← и **переход в q_2**

только изменения!

	q_1
0	→
1	→
□	← q_2

q_2 : увеличение числа на 1

- если 0, то записать 1 и стоп (q_0)
- если 1, то записать 0 и ←
- если □, то записать 1 и стоп (q_0)

	q_2
0	1 • q_0
1	0 ←
□	1 • q_0

? Как объединить две программы?

Программа для машины Тьюринга

	q_1
0	→
1	→
□	$\leftarrow q_2$

	q_2
0	1
1	0
□	$1 \bullet q_0$

! Связь подзадач через ячейку (□, q_1)!

Если алгоритмы А и Б можно запрограммировать на машине Тьюринга, то и любую их комбинацию тоже можно запрограммировать.

Тезис Чёрча-Тьюринга: Любой алгоритм (в интуитивном смысле этого слова) может быть представлен как программа для машины Тьюринга.

Программа для машины Тьюринга

начальное состояние | новая метка | переход | новое состояние

$$(\,0,\ q_1,\ \ 0,\ \ \rightarrow,\ \ q_1\,)$$
$$(\,1,\ q_1,\ \ 1,\ \ \rightarrow,\ \ q_1\,)$$
$$(\,\square,\ q_1,\ \ \square,\ \ \leftarrow,\ \ q_2\,)$$

$$(\,0,\ q_2,\ \ 1,\ \ \bullet,\ \ q_0\,)$$
$$(\,1,\ q_2,\ \ 0,\ \ \leftarrow,\ \ q_2\,)$$
$$(\,\square,\ q_2,\ \ 1,\ \ \bullet,\ \ q_0\,)$$

	q_1
0	$0 \rightarrow q_1$
1	
$\square$	

	q_2
0	$1 \bullet q_0$
1	$0 \leftarrow q_2$
$\square$	$1 \bullet q_0$

Программы для машины Тьюринга

	q_1
0	$\leftarrow$
1	$\leftarrow$
□	$\rightarrow q_0$

	q_1
0	$\rightarrow q_0$
1	$\rightarrow q_0$
□	$\leftarrow$

	q_1	q_2
0	q_2	□ $\leftarrow$
1	q_2	□ $\leftarrow$
□	$\leftarrow$	q_0

? Что делает программа?

? Когда зацикливается?

Программы для машины Тьюринга

Задача 1. Уменьшить двоичное число на 1.

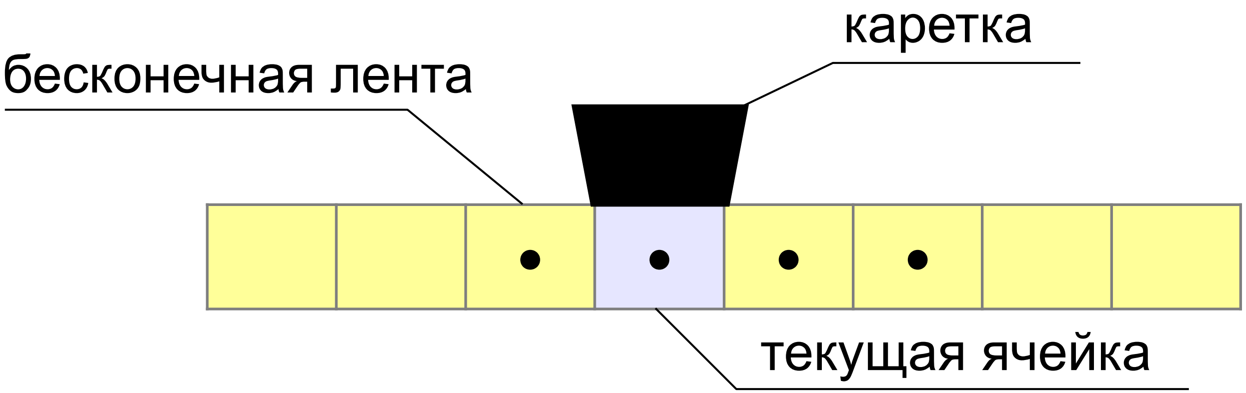
Задача 2. Увеличить на единицу число, записанное в десятичной системе счисления.

Задача 3. Уменьшить на единицу число, записанное в десятичной системе счисления.

Задача 4. Сложить два числа в двоичной системе, разделенные на ленте знаком «+».

Задача 5. Сложить два числа в десятичной системе, разделенные на ленте знаком «+».

Машина Поста



Э. Пост

! Алфавит сокращён до двух символов!

← переместить каретку на 1 ячейку влево

→ переместить каретку на 1 ячейку вправо

0 стереть метку в рабочей ячейке (записать **0**)

1 поставить метку в рабочей ячейке (записать **1**)

? n_0, n_1 если в рабочей ячейке нет метки, перейти к строке n_0, иначе перейти к строке n_1

стоп остановить машину

Программа для машины Поста

```
1. ←
2. ? 1,3
3. стоп
```

Что делает?

Зацикливается?

команда с переходом

← 3

строки нумеруются

сдвинуть каретку влево и перейти на строку 3

Как записывать числа?

		•	•	•	•		

4 в унарной системе!

Машины Поста и Тьюринга эквивалентны!

Программы для машины Поста

1	1
2	→
3	→ 1

1	←
2	? 3,4
3	1 1
4	стоп

1	? 2,3
2	1 4
3	→ 1
4	стоп

? Что делает программа?

? При каких состояниях ленты?

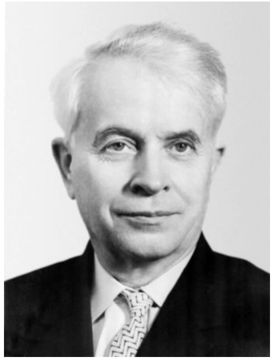
Программы для машины Поста

Задача 1. Напишите программу для машины Поста, которая увеличивает (уменьшает) число в единичной системе счисления на единицу. Каретка расположена слева от числа.

Задача 2. Напишите программу для машины Поста, которая складывает два числа в единичной системе счисления. Каретка расположена над пробелом, разделяющим эти числа на ленте.

Нормальные алгорифмы Маркова (НАМ)

НАМ – правила обработки символьных строк с помощью подстановок.



А. Марков

```
а  → н
ух → ло
м  → с
```

муха → **мухн** → **млон** → **слон**

```
→ 0
```

добавить 0 в начало строки

```
а → о.
```

заменить и стоп

? Корова → ?

Карова

Нормальные алгорифмы Маркова (НАМ)

Задача. Удалить из строки, состоящей из букв a и b, первый символ. Например, строка $abba$ должна бы преобразована в bba.

«Очевидное» решение:

~~a → .~~
~~b → .~~

? Что плохо?

~~b → .~~
~~a → .~~

Правильное решение:

маркер

```
*a → .
*b → .
   → *
```

abba	baab
*abba	*baab
bba	aab

! НАМ, машины Поста и Тьюринга эквивалентны!

Нормальные алгорифмы Маркова

алфавит: $A = \{0, 1\}$

```
0 → 00
1 → 11
```

```
*0 → 0*
*1 → 1*
*  →  =.
→ *
```

```
*0 → 00*
*1 → 11*
*  →   .
→ *
```

? Что делает НАМ?

Нормальные алгорифмы Маркова

Задача 1. Напишите НАМ, который «сортирует» цифры двоичного числа так, чтобы сначала стояли все нули, а потом – все единицы.

Задача 2. Напишите НАМ, который удаляет последний символ строки, состоящей из цифр 0 и 1. Какую операцию он выполняет, если рассматривать строку как двоичную запись числа?

Задача 3. Напишите НАМ, который умножает двоичное число на 2, добавляя 0 в конец записи числа.

Элементы теории алгоритмов

§ 35. Алгоритмически неразрешимые задачи

Вычислимые функции

входное слово

выходное слово

муха → **алгоритм** → **слон**

! Любой алгоритм определяет функцию!

т.е. правило преобразования входа в выход

Вычислимая функция – это функция, для вычисления которой существует алгоритм.

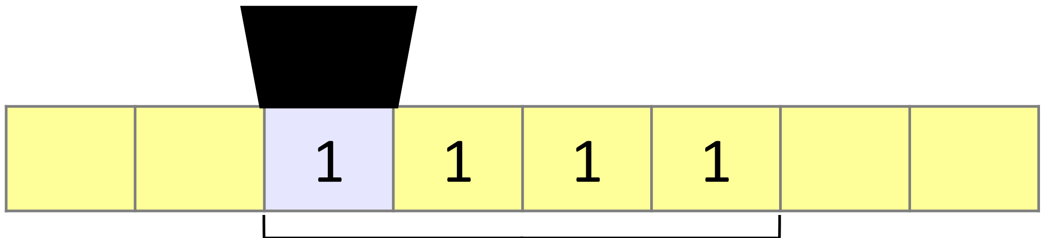
может задаваться разными алгоритмами:

```
а → 0
б → 1
```

```
б → 1
а → 0
```

Вычислимые функции

$$f(n)=\begin{cases}1, & \text{если } n \text{ – чётное}\\ 0, & \text{если } n \text{ – нечётное}\end{cases}$$



в унарной системе счисления

	q_1	q_2	q_3	q_4
1	$\rightarrow q_2$	$\rightarrow q_1$	$\leftarrow q_4$	$\square \leftarrow$
$\square$	$\leftarrow q_3$	$\leftarrow q_4$		q_0

q_1 – чётное число единиц

q_2 – нечётное число единиц

q_3 – оставить 1 единицу

q_4 – стереть все единицы

? Почему пусто?

Вычислимые функции

$$f(n)=\begin{cases}1, & \text{если } n \text{ – чётное}\\ 0, & \text{если } n \text{ – чётное}\end{cases}$$

```
11 → ""
1 → .
→ 1.
```

? Как написать НАМ?

Невычислимая функция (*В.А. Успенский*):

$$h(n)=\begin{cases}1, & \text{если в записи числа } \pi \text{ есть } n \text{ стоящих подряд девяток в окружении других цифр}\\ 0, & \text{если такой цепочки нет}\end{cases}$$

перебор 800 знаков:

$$h(n)=1 \;\text{ для } n=1,\,2,\,6.$$

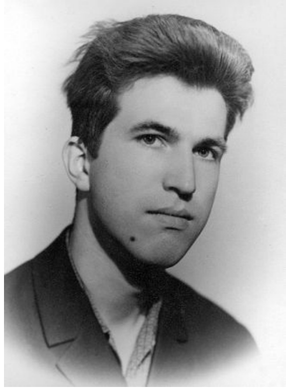
! Если $h(n)=0$, перебор не остановится!

Алгоритмически неразрешимые задачи

Алгоритмически неразрешимая задача – это задача, соответствующая невычислимой функции.

⇒ общего решения задачи нет, его бесполезно искать!

10-я проблема Гильберта (1900): найти метод, который позволяет определить, имеет ли заданное алгебраическое уравнение с целыми коэффициентами решение в целых числах.



Ю.В. Матиясевич

$x^2 + y^3 + 2 = 0$ ⇒ (5;–3) и (–5;–3)

! 1970: общего алгоритма **нет**!

Алгоритмически неразрешимые задачи

Г.В. Лейбниц, XVII в.: разработать алгоритм, позволяющий установить, можно ли вывести формулу Б из формулы А в рамках заданной системы аксиом («***проблема распознавания выводимости***»).

! 1936: в общем виде задача **неразрешима**!

А. Чёрч

\+ удалось получить отрицательные результаты

Алгоритмически неразрешимые задачи

Проблема останова: по тексту любой программы P и ее входным данным X определяет, завершается ли программа P при входе X за конечное число шагов или зацикливается.

Проблема эквивалентности: по двум заданным алгоритмам определить, будут ли они выдавать одинаковые результаты для любых допустимых исходных данных.

! Невозможно полностью автоматизировать отладку программ!

Элементы теории алгоритмов

§ 36. Сложность вычислений

Что такое сложность вычислений?

Задачи теории алгоритмов:
- существует ли алгоритм решения задачи?
- можно ли им воспользоваться?

Шахматы:
- алгоритм существует (конечное число позиций)
- полный перебор нереален

Требования к алгоритму:
- быстродействие
- минимальный расход памяти

временнáя сложность

пространственная сложность

! Обычно эти требования противоречивы!

Временна́я сложность

T – количество элементарных операций универсального исполнителя (компьютера)

! T зависит от размера входных данных n!

Временная сложность алгоритма – функция $T(n)$.

Задача 1. Вычислить сумму первых трёх элементов массива (при $n \geq 3$).

```
Sum:= A[1] + A[2] + A[3]
```

$T(n) = 3$

2 сложения + запись в память

Задача 2. Вычислить сумму всех элементов массива.

```
Sum:= 0
нц для i от 1 до n
  Sum:= Sum + A[i]
кц
```

$T(n) = 2n + 1$

n сложений, $n+1$ операций записи

Временна́я сложность

Задача 3. Отсортировать все элементы массива по возрастанию методом выбора.

```
нц для i от 1 до n-1
  nMin:= i;
  нц для j от i+1 до n
    если A[i] < A[nMin] то nMin:= j все
  кц
  если nMin <> i то
    c:= A[i]; A[i]:= A[nMin]; A[nMin]:= c
  все
кц
```

Число сравнений: $T_c(n) = (n-1) + (n-2) + \ldots + 2 + 1 = \frac{n(n-1)}{2} = \frac{1}{2}n^2 - \frac{1}{2}n$

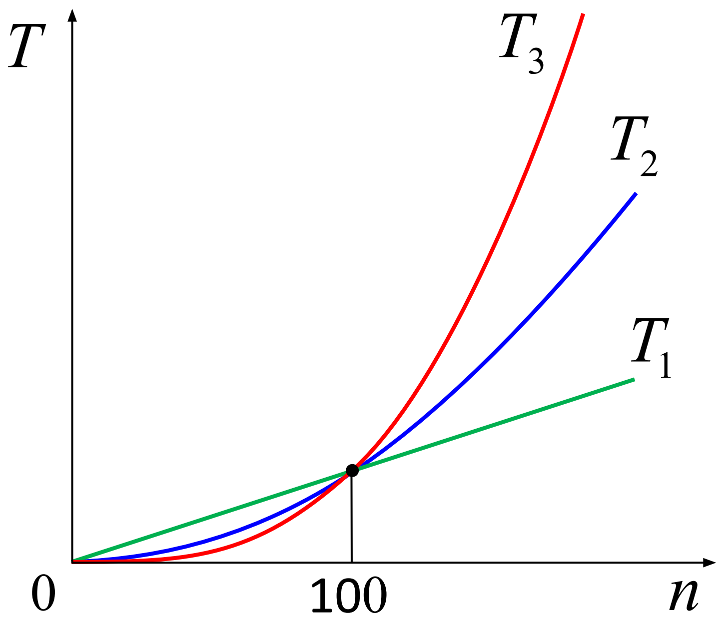
Число перестановок: $T_p(n) \le n-1$

зависит от данных

Сравнение алгоритмов по сложности

$T_1(n) = 10000 \cdot n$ $T_2(n) = 100 \cdot n^2$ $T_3(n) = n^3$

Какой алгоритм выбрать?



при $n < 100$:

$$T_3(n) < T_2(n) < T_1(n)$$

при $n > 100$:

$$T_3(n) > T_2(n) > T_1(n)$$

Нужно знать размер данных!

Асимптотическая сложность

Асимптотическая сложность – это скорость роста количества операций при больших значениях n.

линейная

постоянная

сложность $O(n)$ $\Leftrightarrow$ $T(n) \le c \cdot n$ для $n \ge n_0$

сумма элементов массива:

$$T(n) = 2 \cdot n - 1 \le 2 \cdot n \quad \text{для} \quad n \ge 1 \quad \Rightarrow$$

квадратичная

сложность $O(n^2)$ $\Leftrightarrow$ $T(n) \le c \cdot n^2$ для $n \ge n_0$

сортировка методом выбора:

$$T_c(n) = \frac{1}{2}n^2 - \frac{1}{2}n \le \frac{1}{2}n^2 \quad \text{для} \quad n \ge 0 \quad \Rightarrow \quad O(n^2)$$

Асимптотическая сложность

кубичная

сложность $O(n^3)$ ⇔ $T(n) \le c \cdot n^3$ для $n \ge n_0$

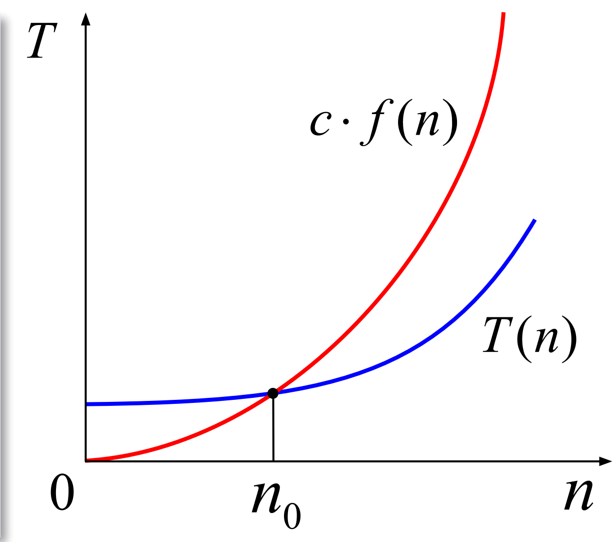
сложность $O(2^n)$

сложность $O(n!)$

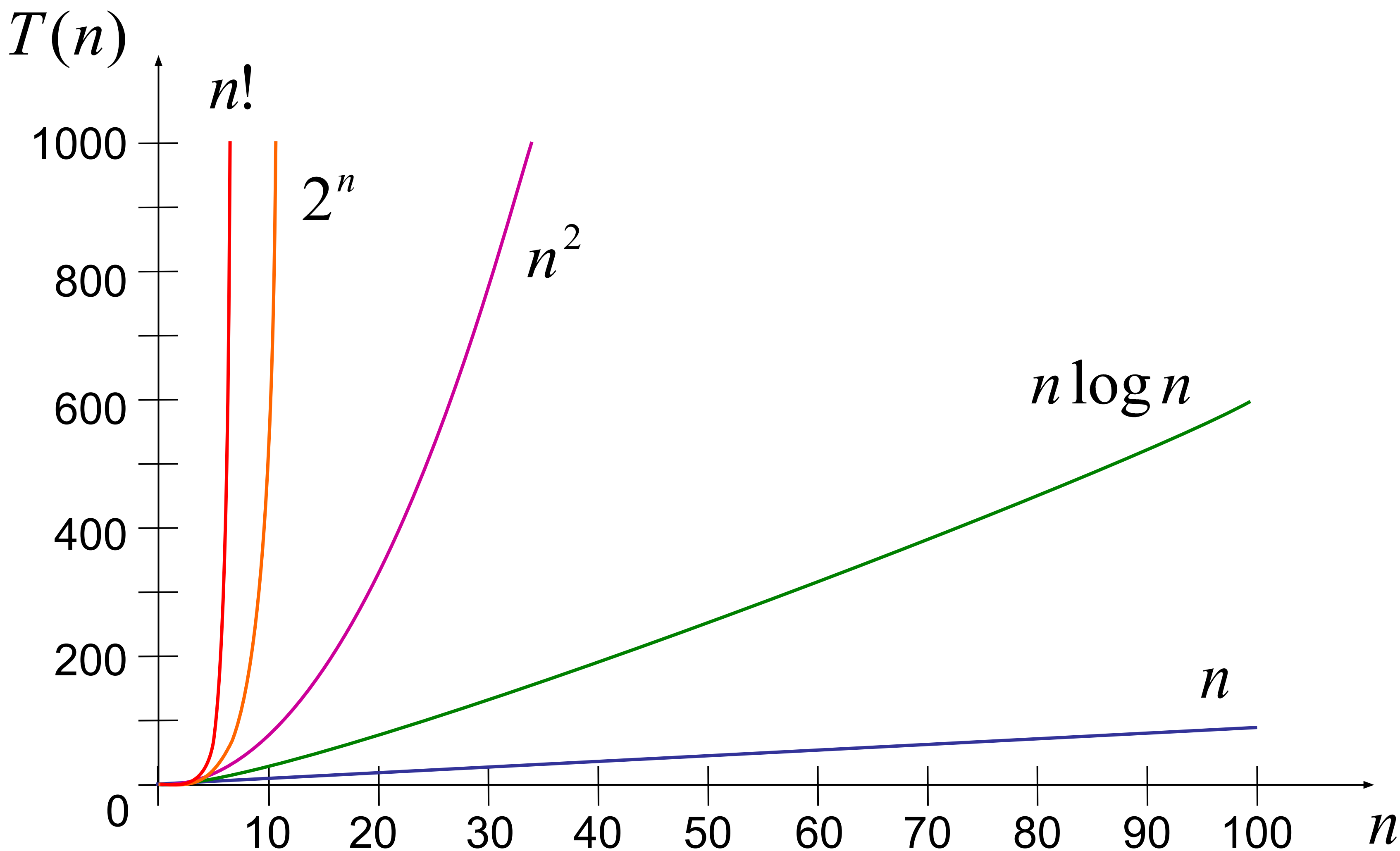
задачи оптимизации, полный перебор вариантов

Алгоритм имеет **асимптотическую сложность** $O\,(\,f\,(n)\,)$, если найдется такая постоянная c, что начиная с некоторого $n = n_0$ выполняется условие

$$T(n) \le c \cdot f\,(n)$$



Асимптотическая сложность



Алгоритмы поиска

Линейный поиск

```
nX:= 0
нц для i от 1 до n
  если A[i] = X то
    nX:= i
    выход
  все
кц
```

? Сложность?

сложность $O(n)$

Алгоритмы поиска

Двоичный поиск

```
L:= 1; R:= n + 1
нц пока L < R - 1
  c:= div(L + R, 2)
  если X < A[c] то
    R:= c
  иначе
    L:= c
  все
кц
```

$\log_2 n$

$n = 2^m$

? Сколько шагов?

$T(n) = m + 1$

$T(n) = \log_2 n + 1$

сложность $O(\log n)$

основание роли не играет

$$\log_a n = \frac{1}{\log_b a} \cdot \log_b n$$

? Какой алгоритм поиска лучше?

Алгоритмы сортировки

Метод «пузырька»

```
нц для i от 1 до n-1
  нц для j от n-1 до i шаг -1
    если A[j] > A[j+1] то
      c:= A[j]; A[j]:= A[j+1]; A[j+1]:= c;
    все
  кц
кц
```

сравнений: $T_c(n) = (n-1) + (n-2) + \ldots + 2 + 1 = \dfrac{n(n-1)}{2} = \dfrac{1}{2}n^2 - \dfrac{1}{2}n$

присваиваний при перестановках:

$$T_p(n) = 3 \cdot \frac{n(n-1)}{2} = \frac{3}{2}n^2 - \frac{3}{2}n$$

сложность $O(n^2)$

Алгоритмы сортировки

Сортировка подсчётом

```
цел C[1:MAX]
нц для i от 1 до MAX
  C[i]:=0
кц
```

обнулить массив счётчиков

```
нц для i от 1 до n
  C[A[i]]:=C[A[i]]+1
кц
```

n

подсчитать, сколько каких чисел

```
k:=1
нц для i от 1 до MAX
  нц для j от 1 до C[i]
    A[k]:=i
    k:=k+1
  кц
кц
```

n

заполнить массив заново

! Все значения [1,MAX]!

сложность $O(n)$

? За счёт чего?

Алгоритмы сортировки

! При использовании операций «сравнить» и «переставить» сложность не может быть меньше $O(n \cdot \log n)$!

Сортировка слиянием (*Merge sort*) $O(n \cdot \log n)$

Пирамидальная сортировка (*Heap sort*) $O(n \cdot \log n)$

Быстрая сортировка (*Quick sort*)

в среднем $O(n \cdot \log n)$

в худшем случае $O(n^2)$

Элементы теории алгоритмов

§ 37. Доказательство правильности программ

Как доказать правильность программы?

Тестирование – проверка работы программы с помощью набора тестовых данных, для которых известен правильный результат.

? Может ли тестирование доказать правильность?

«*Отладка может показать лишь наличие ошибок и никогда их отсутствие*».

Э.В. Дейкстра

Поиск максимума из трёх:

```
если a > b то M:=a иначе M:=b все
если b > c то M:=b иначе M:=c все
```

Тесты проходят: (a,b,c) = (1,2,3), (1,3,2), (2,1,3) и (2,3,1)

Тесты не проходят: (3,1,2), (3,2,1)

Доказательное программирование

требования к входным данным → **алгоритм** → требования к результату

вход: **a** $= a_0$, **b** $= b_0$

```
b:=a + b
a:=b - a
b:=b - a
```

b $= a_0 + b_0$

a $= a_0 + b_0 - a_0 = b_0$

b $= a_0 + b_0 - b_0 = a_0$

выход: **a** $= b_0$, **b** $= a_0$

доказательство

```
если a > b то M:=a иначе M:=b все
если b > c то M:=b иначе M:=c все
```

$$M = \begin{cases} b, & b > c \\ c, & c \geq b \end{cases}$$

алгоритм неверный!

Алгоритм Евклида

```
a:=m; b:=n
нц пока b<>0
  r:=mod(a,b)
  a:=b; b:=r
кц
вывод a
```

$НОД(a,b)=НОД(m,n)$

$a=b\cdot p+r$

$НОД(a,b)=НОД(b,r)=НОД(m,n)$

$b=0$

$a=НОД(a,0)=НОД(a,b)=НОД(m,n)$

После любого шага цикла:

$НОД(a,b)=НОД(m,n)$ **инвариант цикла**

Инвариант цикла

Инвариант цикла – это соотношение между значениями переменных, которое остается справедливым после завершения любого шага цикла.

«*Программиста бьют по рукам, если он посмеет написать оператор цикла, не найдя перед этим его инварианта*».

А.П. Ершов

Инвариант цикла

Сумма элементов массива:

```
Sum:= 0
нц для i от 1 до n
  Sum:= Sum + A[i]
кц
```

Sum = A[1]+A[2]+ … + A[i]

Поиск в массиве:

```
Min:=A[1]
нц для i от 2 до n
  если A[i] < Min то
    Min:= A[i]
  все
кц
```

Min = min(A[1],A[2],…, A[i])

Инвариант цикла

Сортировка методом «пузырька»:

```
нц для i от 1 до n-1
  нц для j от n-1 до i шаг -1
    если A[j] > A[j+1] то
      c:= A[j]; A[j]:= A[j+1]; A[j+1]:= c;
    все
  кц
кц
```

i-й элемент по порядку стоит в позиции от i до j

i первых элементов установлены на свои места

Быстрое возведение в степень $a^n = ?$

Задача – построить цикл с помощью инварианта.

Правила:

$$a^k = a^{k-1} \cdot a$$

при нечётных k

$$a^k = \left(a^2\right)^{k/2}$$

при чётных k

$$a^7 = a^6 \cdot [a]$$

$$a^n = \boxed{b^k} \cdot \boxed{p}$$

⇐ **инвариант**

! Задача – свести k к нулю!

$$k = 0 \;\Rightarrow\; a^n = p$$

Быстрое возведение в степень

```
b:=a; k:=n; p:=1
нц пока k <> 0
  если mod(k,2) = 0 то
    k:=div(k,2)
    b:=b*b
  иначе
    k:=k-1
    p:=b*p
  все
кц
вывод p
```

$$a^n = b^k \cdot p$$

$$a^k = \left(a^2\right)^{k/2}$$

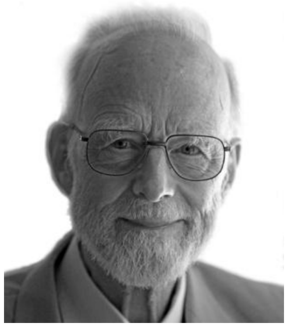
$$a^k = a^{k-1} \cdot a$$

$$a^n = b^k \cdot p$$

? Как можно изменить начальные условия?

Доказательное программирование

Спецификация – точная и полная формулировка задачи, содержащая информацию, необходимую для построения алгоритма её решения.



Ч.Э. Хоар

программа

{Q} S {R}

предусловие постусловие

«*Если выполнение программы **S** началось в состоянии, удовлетворяющем **Q**, то гарантируется, что оно завершится через конечное время в состоянии, удовлетворяющем **R***».

Спецификации алгоритмов

Алгоритм Евклида:

Q: $m \geq n > 0$

R: $a = НОД\,(m,\,n)$

Суммирование элементов массива:

Q: $n > 0$

R: $Sum = \sum_{i=1}^{n} A[i] = A[1] + A[2] + \ldots + A[n]$

Корректная программа – это программа, соответствующая спецификации.

Надёжная программа – это программа, которая корректна и, кроме того, не завершается аварийно при недопустимых входных данных.

Доказательное программирование

Правила преобразования:

- если {Q}S{P} и $P \Rightarrow R$, то {Q}S{R}
- если $R \Rightarrow Q$ и {Q}S{P}, то {R}S{P}
- если {Q}S_1{P} и {P}S_2{R}, то {Q}S_1S_2{R}
- …

Верификация – доказательство правильности готовых программ (сложно!).

! Проще сразу доказывать правильность отдельных блоков (циклов, процедур)!

Конец фильма

ПОЛЯКОВ Константин Юрьевич
д.т.н., учитель информатики
ГБОУ СОШ № 163, г. Санкт-Петербург
kpolyakov@mail.ru

ЕРЕМИН Евгений Александрович
к.ф.-м.н., доцент кафедры мультимедийной дидактики и ИТО ПГГПУ, г. Пермь
eremin@pspu.ac.ru

Источники иллюстраций

1. en.wikipedia.org
2. ru.wikipedia.org
3. иллюстрации художников издательства «Бином»
4. авторские материалы