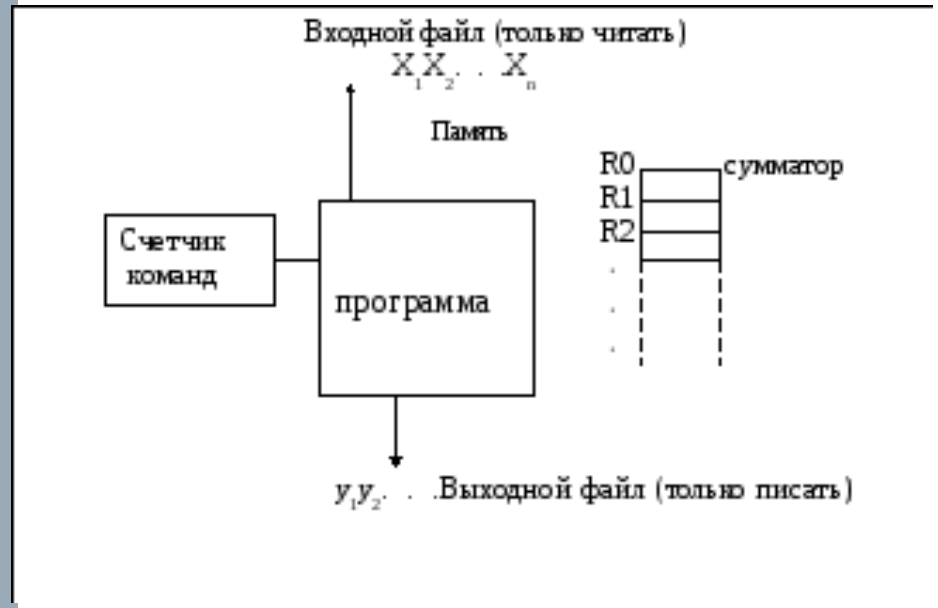
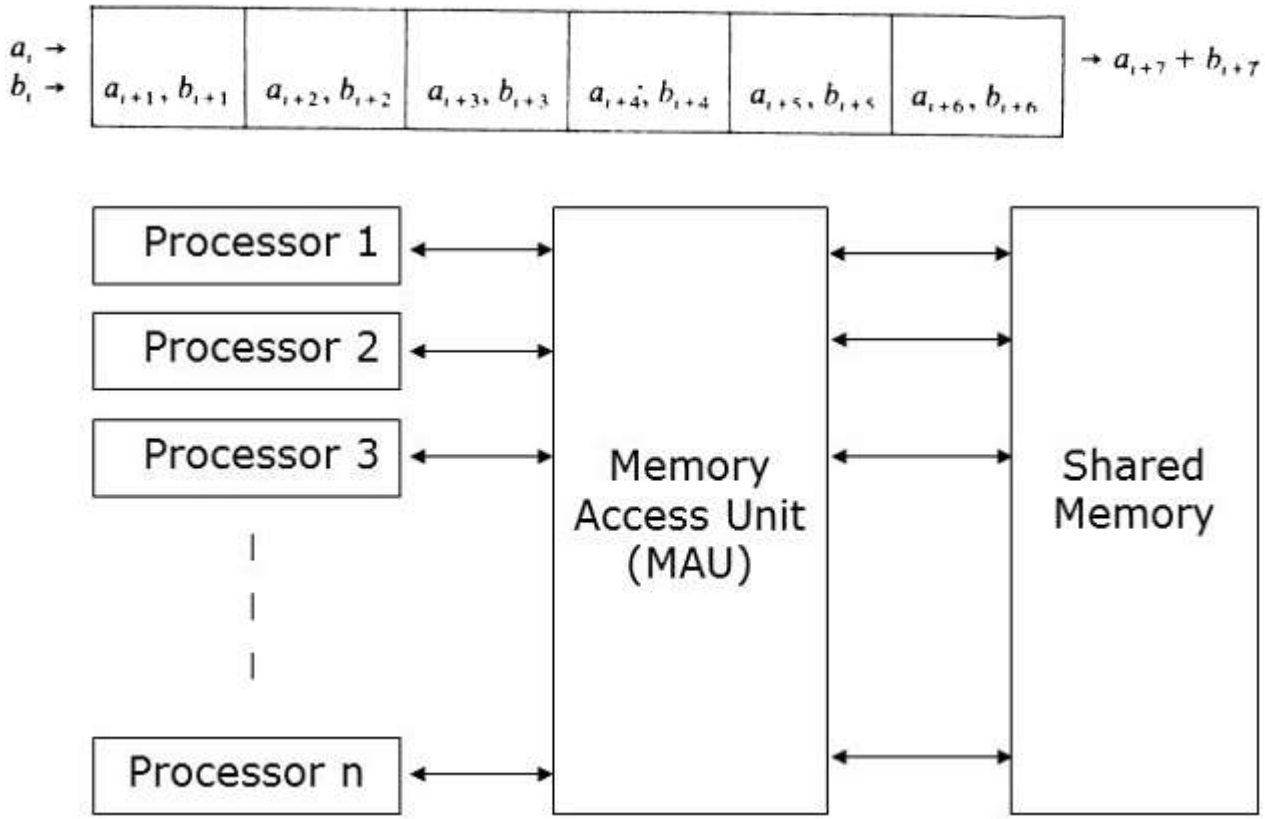
Анализ алгоритмов

Основные моменты

Модель вычислений

- RAM - модель
- Модель конвейера
- Параллельная модель





Любая модель дает возможность изучать и анализировать алгоритмы, не прибегая к использованию конкретного языка программирования или компьютерной платформы

RAM - модель

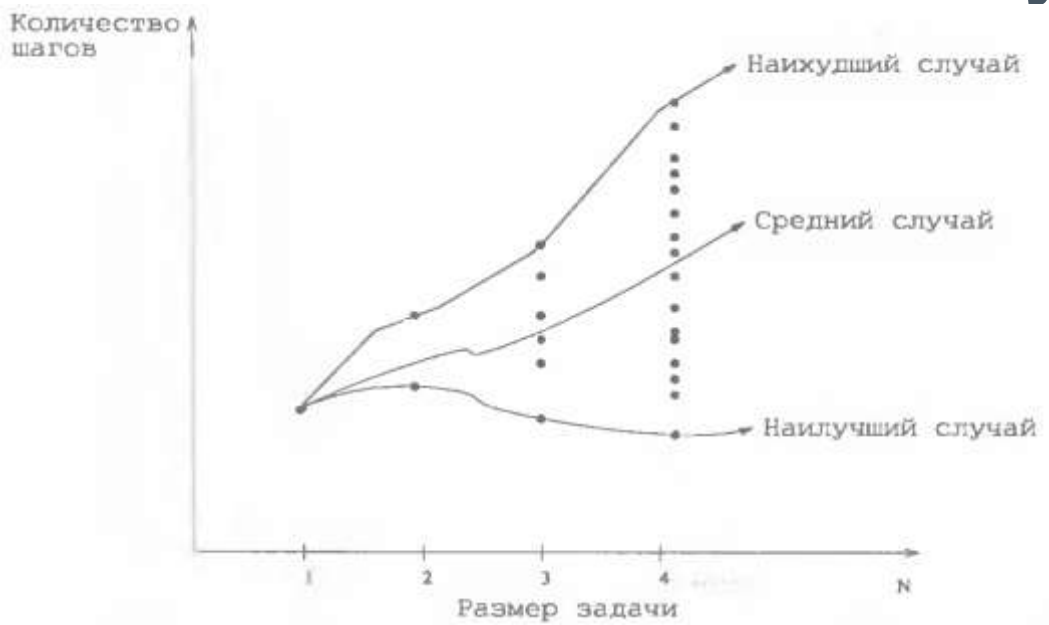
- Для исполнения любой простой операции требуется ровно один временной шаг
- Циклы и подпрограммы не считаются простыми операциями, а состоят из нескольких простых операций
- Каждое обращение к памяти занимает один временной шаг
- Компьютер обладает неограниченным объемом оперативной памяти

Анализ сложности алгоритма

Как работает алгоритм со *всеми* экземплярами задачи

Наилучший случай

Наихудший случай



Средний случай

Пусть D_A - множество конкретных проблем данной задачи, заданное в формальной системе.
Пусть $D \in D_A$ – задание конкретной проблемы и $|D| = N$
Обозначим это подмножество через D_N: $D_N = \{D \in D_N : |D| = N\}$

Обозначим мощность множества D_N через $M_{DN} \Rightarrow M_{DN} = |D_N|$

Наилучший, **наихудший**, средний случай

› *Сложность алгоритма в наихудшем случае* $\hat{F_a}(N)$ – это функция, определяемая максимальным количеством шагов, требуемых для обработки любого входного экземпляра размером *N*.

$$\hat{F_a}(N) = \max_{D \in D_N} \{F_a(D)\}$$

Наилучший, наихудший, средний случай

› *Сложность алгоритма в наилучшем случае* $F_a(N)$ – это функция, определяемая минимальным количеством шагов, требуемых для обработки любого входного экземпляра размером *N*.

$$F_a(N) = \min_{D \in D_N} \{F_a(D)\}$$

Наилучший, наихудший, **средний** случай

› *Сложность алгоритма в среднем случае* $\overline{F}_a(N)$ – это функция, определяемая средним количеством шагов, требуемых для обработки всех экземпляра размером *N*.

$$\overline{F}_a(N) = \frac{1}{M_{D_N}} \sum_{D \in D_N} \{F_a(D)\}$$

Классификация алгоритмов по виду функции трудоемкости

- Количественно-зависимые по трудоемкости алгоритмы.
- Параметрически-зависимые по трудоемкости алгоритмы.
- Количественно-параметрические по трудоемкости алгоритмы
  - Порядково-зависимые по трудоемкости

Количественно-зависимые по трудоемкости алгоритмы

Алгоритмы, функция трудоемкости которых зависит только от размерности конкретного входа, и не зависит от конкретных значений

$$F_a(D) = F_a(|D|) = F_a(N)$$

Пример – алгоритмы для стандартных операций с массивами и матрицами: умножение, сложение и т.д.

Параметрически-зависимые по трудоемкости алгоритмы

Алгоритмы, трудоемкость которых определяется не размерностью входа, а конкретными значениями в обрабатываемой памяти

$$F_a(D) = F_a(d_1, \dots, d_n) = F_a(P_1, \dots, P_m), m \leq n$$

Пример – алгоритмы вычисления стандартных функций с заданной точностью.

Вычисление x^k последовательным умножением $F_a(x, k) = F_a(k)$

Вычисление функции с помощью ряда с точностью ξ

$$e^x = \sum \frac{x^n}{n!}, \text{с точностью } \xi \quad \Longrightarrow \quad F_a = F_a(x, \xi)$$

Количественно-параметрические по трудоемкости алгоритмы

Алгоритмы, трудоемкость которых зависит как от количества данных на входе, так и от значений входных данных

$$F_a(D) = F_a(|D|, P_1, \dots, P_m) = F_a(N, P_1, \dots, P_m)$$

Порядко-зависимые по трудоемкости алгоритмы

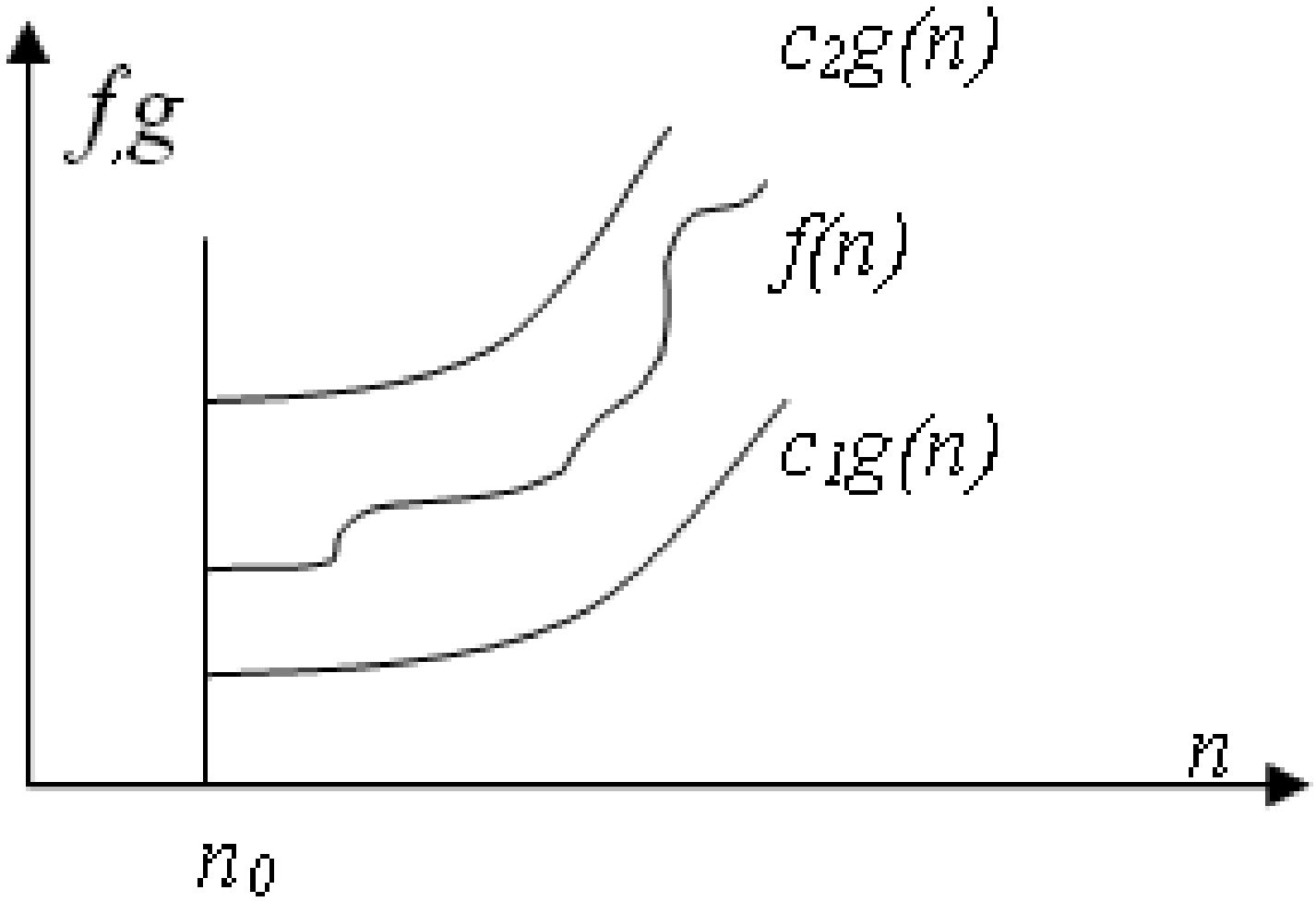
$D: (d_1, \dots, d_n), |D| = N$

Определим $D_p = \{(d_1, \dots, d_n)\}$ – множество всех упорядоченных наборов из $d_1, \dots, d_n, \left|D_p\right| = n!$

Если $F_a\left({}_iD_p\right) \neq F_a\left({}_jD_p\right)$, где ${}_iD_p,\ {}_jD_p \in D_p$, то алгоритм порядко-зависимый по трудоемкости

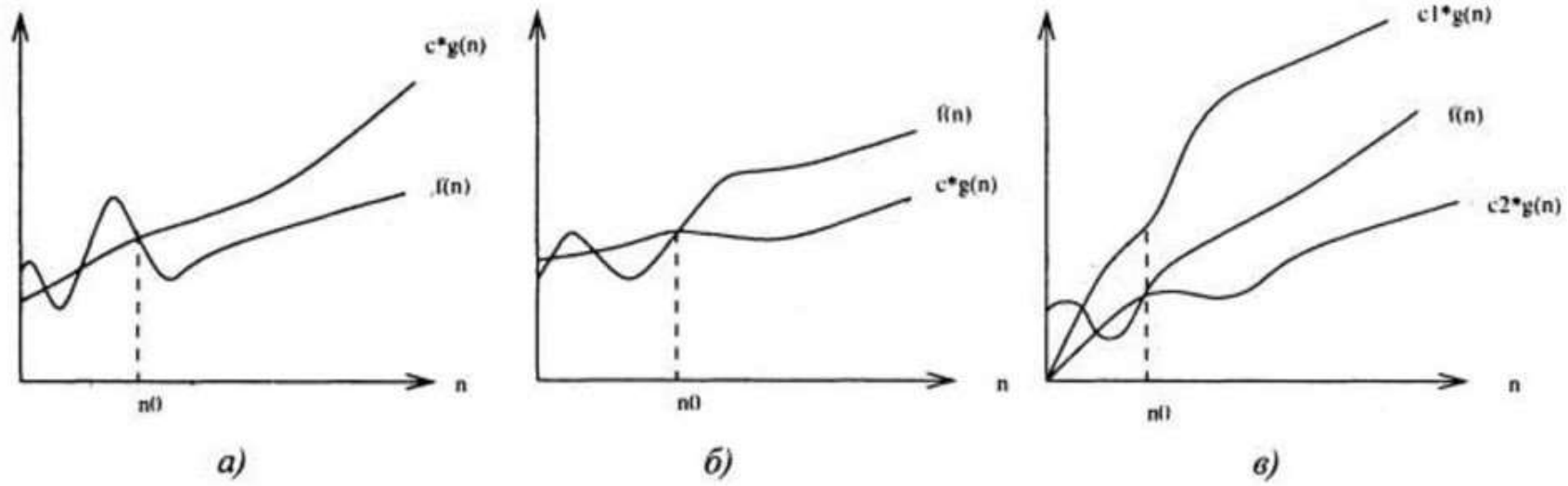
Пример – алгоритмы сортировки, поиска максимума и минимума

Асимптотические обозначения



Асимптотические обозначения

- $f(n) = O(g(n))$ означает, что функция $f(n)$ *ограничена сверху* функцией $c \cdot g(n)$. Иными словами, существует такая константа c, для которой $f(n) \le c \cdot g(n)$ при достаточно большом значении n (т. е. $n \ge n_0$ для некоторой константы n_0);
- $f(n) = \Omega(g(n))$ означает, что функция $f(n)$ *ограничена снизу* функцией $c \cdot g(n)$. Иными словами, существует такая константа c, для которой $f(n) \ge c \cdot g(n)$ для всех $n \ge n_0$;
- $f(n) = \Theta(g(n))$ означает, что функция $f(n)$ *ограничена сверху* функцией $c_1 \cdot g(n)$, а *снизу* функцией $c_2 \cdot g(n)$ для всех $n \ge n_0$. Иными словами, существуют константы c_1 и c_2, для которых $f(n) \le c_1 \cdot g(n)$ и $f(n) \ge c_2 \cdot g(n)$. Следовательно, функция $g(n)$ дает нам хорошие ограничения для функции $f(n)$.



$3n^2 - 100n + 6 = O(n^2)$, т. к. выбрано $c = 3$ и $3n^2 > 3n^2 - 100n + 6$;

$3n^2 - 100n + 6 = O(n^3)$, т. к. выбрано $c = 1$ и $n^3 > 3n^2 - 100n + 6$ при $n > 3$;

$3n^2 - 100n + 6 \neq O(n)$, т. к. для любого значения c выбрано $cn < 3n^2$ при $n > c$;

$3n^2 - 100n + 6 = \Omega(n^2)$, т. к. выбрано $c = 2$ и $2n^2 < 3n^2 - 100n + 6$ при $n > 100$;

$3n^2 - 100n + 6 \neq \Omega(n^3)$, т. к. выбрано $c = 3$ и $3n^2 - 100n + 6 < n^3$ при $n > 3$;

$3n^2 - 100n + 6 = \Omega(n)$, т. к. для любого значения c выбрано $cn < 3n^2 - 100n + 6$ при $n > 100c$;

$3n^2 - 100n + 6 = \Theta(n^2)$, т. к. применимо как O, так и Ω;

$3n^2 - 100n + 6 \neq \Theta(n^3)$, т. к. применимо только O;

$3n^2 - 100n + 6 \neq \Theta(n)$, т. к. применимо только Ω.

Скорость роста и отношения доминирования

$n/f(n)$	$\lg n$	n	$n\lg n$	n^2	2^n	$n!$
10	0,003 мкс	0,01 мкс	0,033 мкс	0,1 мкс	1 мкс	3,63 мс
20	0,004 мкс	0,02 мкс	0,086 мкс	0,4 мкс	1 мс	77,1 лет
30	0,005 мкс	0,03 мкс	0,147 мкс	0,9 мкс	1 с	8.4×10^{15} лет
40	0,005 мкс	0,04 мкс	0,213 мкс	1,6 мкс	18,3 мин	
50	0,006 мкс	0,05 мкс	0,282 мкс	2,5 мкс	13 дней	
100	0,007 мкс	0,1 мкс	0,644 мкс	10 мкс	4×10^{13} лет	
1 000	0,010 мкс	1,00 мкс	9,966 мкс	1 мс		
10 000	0,013 мкс	10 мкс	130 мкс	100 мс		
100 000	0,017 мкс	0,10 мс	1,67 мс	10 с		
1 000 000	0,020 мкс	1 мс	19,93 мс	16,7 мин		
10 000 000	0,023 мкс	0,01 с	0,23 с	1,16 дней		
100 000 000	0,027 мкс	0,10 с	2,66 с	115,7 дней		
1 000 000 000	0,030 мкс	1 с	29,90 с	31,7 лет		

Семейства производительности

- Константная: $O(1)$
- Логарифмическая: $O(\log n)$
- Сублинейная: $O(n^d)$ $d<1$
- Линейная: $O(n)$
- Суперлинейная (линейно-логарифмическая): $O(n \cdot \log n)$
- Квадратичная: $O(n^2)$
- Кубическая: $O(n^3)$
- Экспоненциальная: $O(c^n)$
- Факториальная: $O(n!)$

Работа с асимптотическими обозначениями

Сумма двух функций определяется доминантной функцией

$$O(f(n)) + O(g(n)) = O(\max(f(n),(g(n)))$$
$$\Omega(f(n)) + \Omega(g(n)) = \Omega(\max(f(n),(g(n)))$$
$$\Theta(f(n)) + \Theta(g(n)) = \Theta(\max(f(n),(g(n)))$$

$$2n^3 + 3n^2 + n + 5 = O(n^3)$$

$$O(cf(n)) \to O(f(n))$$
$$\Omega(cf(n)) \to \Omega(f(n))$$
$$\Theta(cf(n)) \to \Theta(f(n))$$

$$O(f(n))^*O(g(n)) \to O(f(n)^*g(n))$$
$$\Omega(f(n))^*\Omega(g(n)) \to \Omega(f(n)^*g(n))$$
$$\Theta(f(n))^*\Theta(g(n)) \to \Theta(f(n)^*g(n))$$

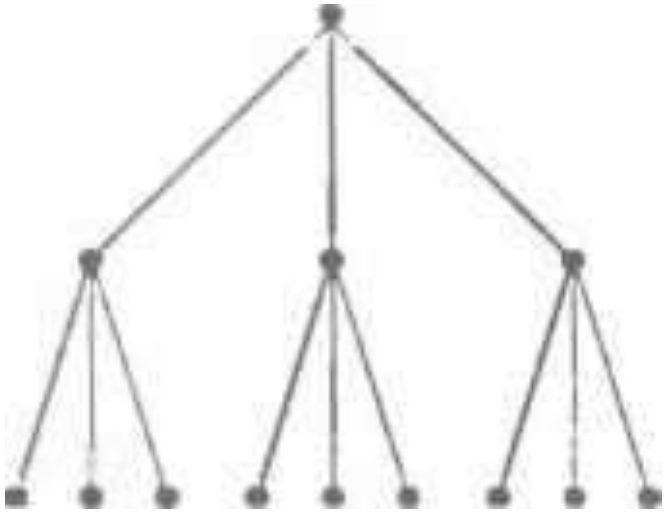
Логарифмы и их применение

Логарифм – это функция, обратная показательной

Двоичный (бинарный) поиск – классический алгоритм поиска элемента в отсортированном массиве (векторе), использующий дробление массива пополам.

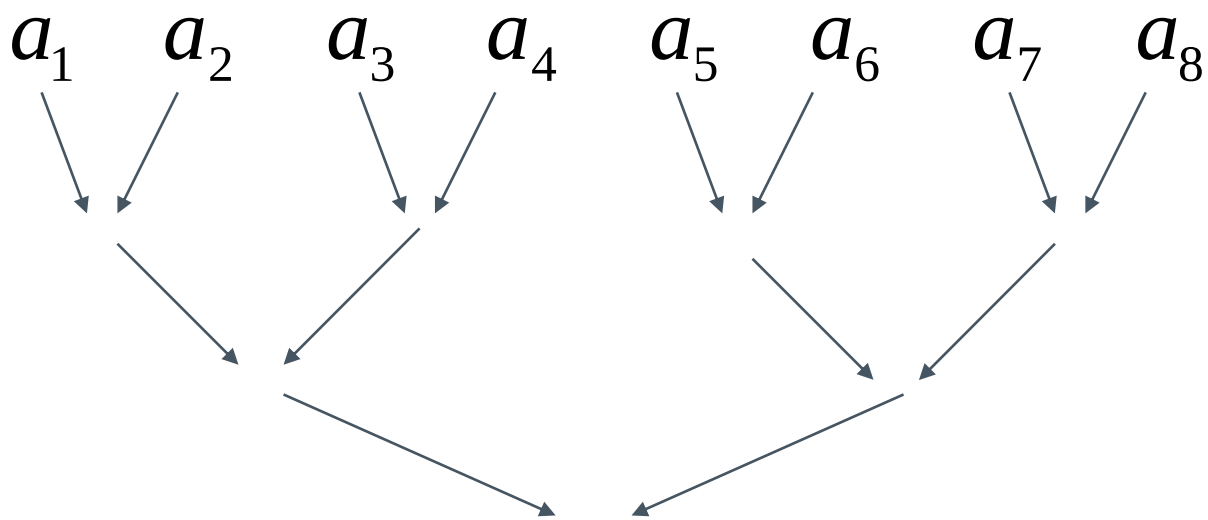
Деревья – d потомков, h – высота

d=2 – бинарное дерево



Параллельный алгоритм

$$a_1 + a_2 + \ldots + a_n = \sum a_i$$



$$\frac{1}{q}\left(\frac{n}{2}+\frac{n}{4}+\ldots+1\right)=\frac{2^q-1}{q}=\frac{n-1}{\log_2 n}=O\left(\frac{n}{\log_2 n}\right)$$

Быстрое возведение в степень

Нужно точно вычислить a^n для достаточно большого *n*.

$$(a \times a \times \cdots \times a) \quad n-1 \text{ раз}$$

Другой способ

$$n = \lfloor n/2 \rfloor + \lceil n/2 \rceil$$

$a^n = \left(a^{n/2}\right)^2 - n$ четное; $a^n = a\left(a^{\lfloor n/2 \rfloor}\right)^2 - n$ нечетное

```
function power(a, n)
    if (n = 0) return(1)
    x = power (a, ⌊n/2⌋)
    if (n is even) then return(x²)
        else return(a × x²)
```

O(log n) операций

Логарифмы и система уголовного судопроизводства

Понесенные убытки	Повышение уровня наказания
(A) 2 000 долларов или меньше	Уровень не повышается
(B) Свыше 2 000 долларов	Повысить на один уровень
(C) Свыше 5 000 долларов	Повысить на два уровня
(D) Свыше 10 000 долларов	Повысить на три уровня
(E) Свыше 20 000 долларов	Повысить на четыре уровня
(F) Свыше 40 000 долларов	Повысить на пять уровней
(G) Свыше 70 000 долларов	Повысить на шесть уровней
(H) Свыше 120 000 долларов	Повысить на семь уровней
(I) Свыше 200 000 долларов	Повысить на восемь уровней
(J) Свыше 350 000 долларов	Повысить на девять уровней
(K) Свыше 500 000 долларов	Повысить на десять уровней
(L) Свыше 800 000 долларов	Повысить на одиннадцать уровней
(M) Свыше 1 500 000 долларов	Повысить на двенадцать уровней
(N) Свыше 2 500 000 долларов	Повысить на тринадцать уровней
(O) Свыше 5 000 000 долларов	Повысить на четырнадцать уровней
(P) Свыше 10 000 000 долларов	Повысить на пятнадцать уровней
(Q) Свыше 20 000 000 долларов	Повысить на шестнадцать уровней
(R) Свыше 40 000 000 долларов	Повысить на семнадцать уровней
(S) Свыше 5 000 000 долларов	Повысить на восемнадцать уровней

Рекомендуемые наказания в федеральных судах США за преступления финансового мошенничества

Теория сложности вычислений и сложностные классы задач

- Теоретический предел трудоемкости задачи

Оценка трудоемкости алгоритма в худшем случае $F_a(D_a) = O\big(g(D_a)\big)$

Есть ли функциональный нижний предел для $g(D_a)$?

Функциональный теоретический нижний предел трудоемкости задачи в худшем случае

$$F_{thlim} = \min\{\Theta(\hat{F_a}(D))\}$$

Тогда любой алгоритм

$$\hat{F_a}(D) = \Omega(F_{thlim})$$

1. Задача поиска максимума в массиве A=(a_1,…,a_n) - $F_{thlim} = \Theta(n)$
2. Задача умножения матриц $F_{thlim} = \Theta\big(n^2\big)$ (реально $\Theta\big(n^{2,34}\big)$)

Сложностные классы задач

- Класс P (задачи с полиномиальной сложностью)
- Класс NP (полиномиально проверяемые задачи)
- Проблема P=NP
- Класс NPC (NP – полные задачи)

Работы Кобмена (Alan Cobham, 1964) и
Эдмондсона (Jack Edmonds, 1965)

Класс P

Задача называется полиномиальной ,т.е. относится к классу P, если ∃ k=const и алгоритм, решающий задачу с

$$F_a(n) = O\left(\mathrm{n}^{\mathrm{k}}\right),$$

где n – длина входа алгоритма в битах n=|D|

Преимущества:

› для большинства задач из класса P константа k<6

› класс P инвариантен по модели вычислений (для широкого класса моделей)

› класс P обладает свойством естественной замкнутости

Класс NP

Дано: N чисел – A = (a_1,…,a_n) и число V

Задача: Найти вектор X = (x_1,…,x_n), $x_i \in \{0,1\}$: $\sum a_i x_i = V$

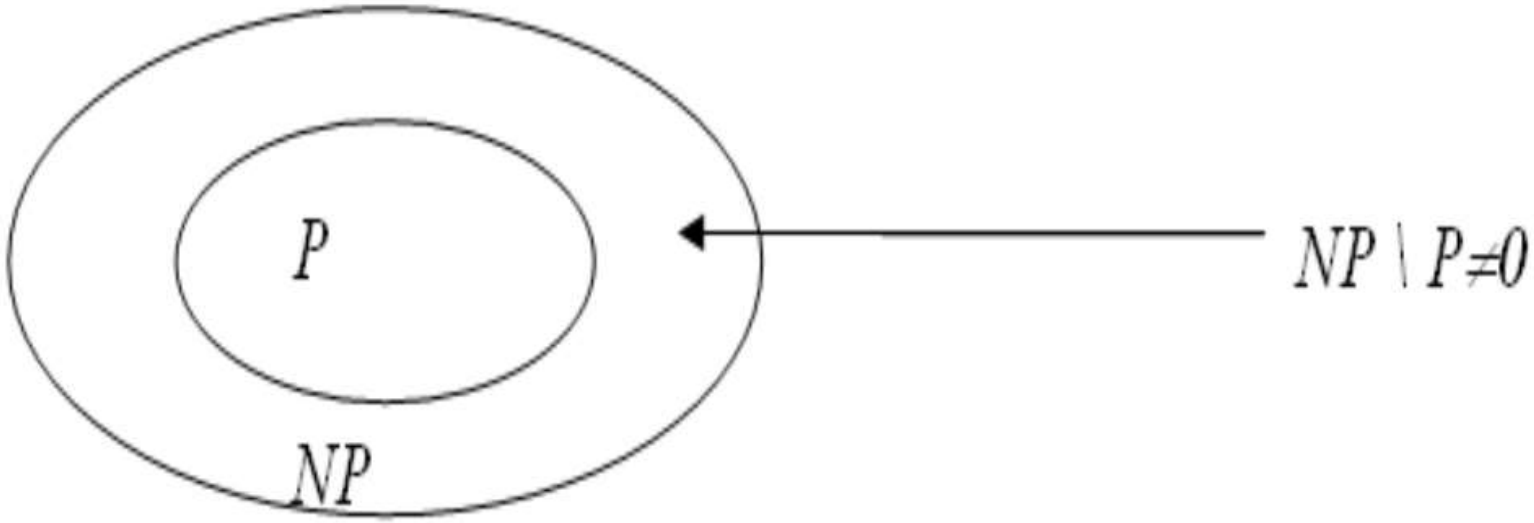
Если есть алгоритм нахождения X, то проверка правильности результата м.б. выполнена с полиномиальной сложностью: требует не более $\Theta(N)$ операций

Формально: $\forall\ D \in D_A$, $|D|=n$ поставим в соответствие сертификат $S \in S_A$: $|S|=O(n^l)$ и алгоритм $A_S=A_S(D,S)$, такой что он выдает «1», если решение правильно, и «0» ,если решение не верно. Тогда задача принадлежит классу NP, если $F(A_S)=O(n^m)$.

Проблема P=NP

Основная проблема теории сложности

Можно ли все задачи, решение которых проверяется с полиномиальной сложностью, решить за полиномиальное время?



На сегодня отсутствует теоретические доказательства как совпадения, так и не совпадения этих классов.

Предположение – класс P является подмножеством класса NP

Класс NP-полных задач

Понятие сводимости:

Если есть Задача 1 (P1) и решающий эту задачу алгоритм, а для Задачи 2 (P2) алгоритм неизвестен, то если мы можем переформулировать P2 в терминах P1, то мы решаем P2.

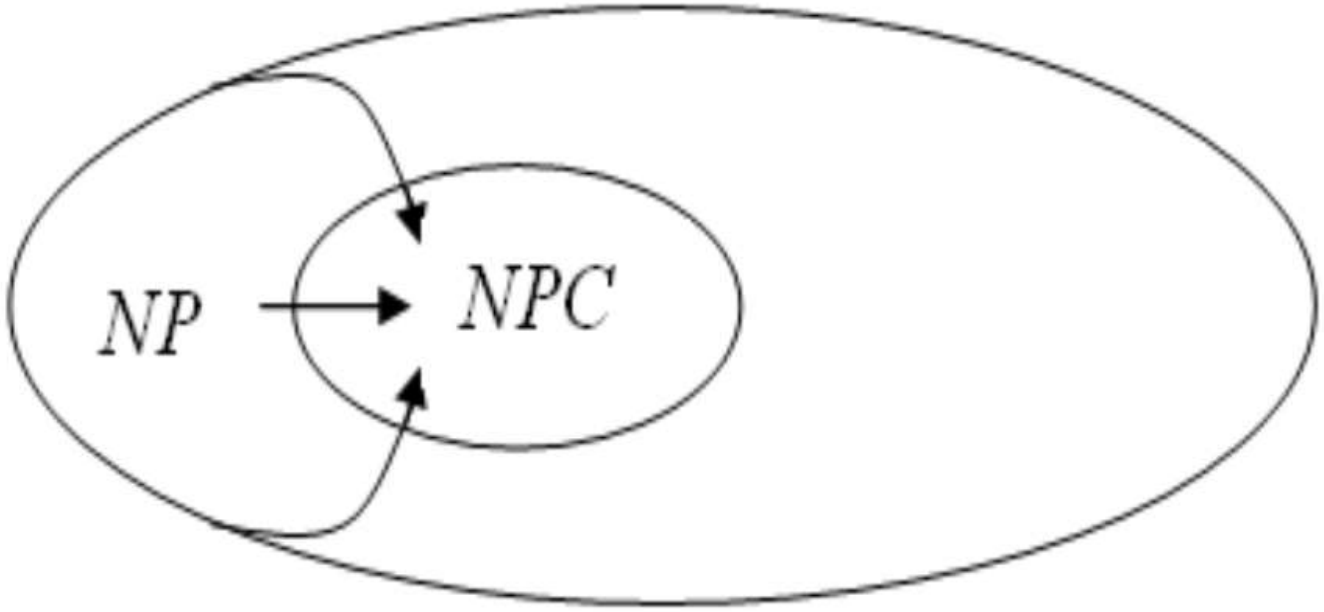
Если P1 задана множеством конкретных проблем D_{A1}, а P2 – множеством D_{A2} и $\exists\ f_s$ (алгоритм), сводящий конкретную постановку P2 (d_{A2}) к конкретной постановке P1 (d_{A1}) : $f_S(d_{A2} \in D_{A2}) = d_{A1} \in D_{A1}$, то P2 сводима к P1

Если при этом $F_a(f_S) = O(n^k)$, т.е. принадлежит классу P, то

P1 полиномиально сводится к P2

Условия класса NPC

- Задача принадлежит классу NP
- К данной задачи должны полиномиально сводиться все задачи из класса NP



Соотношение классов P, NP, NPC