

Сложность алгоритмов

Юрий Литвинов
yurii.litvinov@gmail.com

17.09.2019

Сложность

- ▶ Быстрее — не значит лучше!
 - ▶ Время работы программиста может быть дороже времени работы программы
 - ▶ Но может быть и наоборот, если программа часто используется
- ▶ Сложность
 - ▶ Вычислительная (время работы)
 - ▶ Емкостная (объём потребляемой памяти)
 - ▶ Они взаимосвязаны
- ▶ Вычислительную сложность непонятно, как измерять
 - ▶ Зависит от машины, на которой считаем
 - ▶ Зависит от объёма входных данных
 - ▶ Зависит от самих входных данных

Асимптотическая сложность

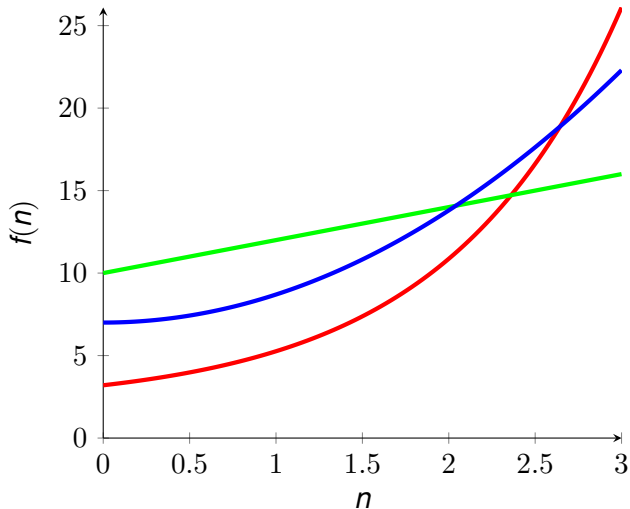
- ▶ О-символика

$$f(n) \in O(g(n))$$

$$\exists C > 0, n_0 : \forall n > n_0 \quad f(n) \leq C(g(n))$$

- ▶ То есть “ f принадлежит классу функций $O(g)$, если начиная с некоторого n_0 она ограничена сверху функцией g с точностью до некоторого наперёд заданного множителя”
- ▶ Или ещё более то есть — “ f растёт не быстрее g ”

Пример



Правила вычисления трудоёмкости

- ▶ Оператор S_1 выполняется за время $T_1(n)$, имеющее порядок $O(f(n))$, оператор S_2 — за время $T_2(n)$, имеющее порядок $O(g(n))$, тогда $S_1; S_2$ выполняется за время $O(\max(f(n), g(n)))$
 - ▶ Следствие: $O(n^2 + n) = O(n^2)$
- ▶ Если S_1 внутри себя порядка $O(f(n))$ раз вызывает S_2 , то итоговая трудоёмкость равна $O(f(n) * g(n))$
 - ▶ Это правило позволяет анализировать циклы

Пример: пузырьёк

```
for (int i = 0; i < n; ++i)           //  $O((n - 1) + \dots + 1) = O(n * (n - 1) / 2) = O(n^2)$ 
  for (int j = n; j > i; --j)         //  $O((n - i) * 1) = O(n - i)$ 
    if (a[j - 1] > a[j])              //  $O(1)$ 
      swap(a[j - 1], a[j]);           //  $O(1)$ 
```

- ▶ Бывают сортировки за $O(n * \log(n))$
- ▶ Бывает сортировка подсчётом, за $O(n)$
- ▶ Это важно, $\log_2(1000000) \sim 20$
 - ▶ Обратите внимание, $O(\log_2(n)) = O(\ln(n))$

Анализ рекурсивных алгоритмов

```
int recursiveFactorial(int a) {  
    if (a <= 1)  
        return 1;  
    else  
        return a * recursiveFactorial(a - 1);  
}
```

- ▶ $T(n) = c + T(n - 1)$ при $n > 1$, и d , при $n \leq 1$
- ▶ $T(n) = c + T(n - 1) = 2c + T(n - 2) = \dots = i * c + T(n - i) = \dots = (n - 1) * c + T(1) = (n - 1) * c + d$

Числа Фибоначчи

- ▶ $F_n = F_{n-2} + F_{n-1}, F_0 = 1, F_1 = 1$
- ▶ $F = 1, 1, 2, 3, 5, 8, 13, 21, \dots$
- ▶ Рекурсивное решение — ?
- ▶ Итеративное решение — ?
- ▶ Решение за $O(\log(n))$:

$$\begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n$$

- ▶ Формула Бине:

$$F_n = \frac{1}{\sqrt{5}} \left[\left(\frac{1 + \sqrt{5}}{2} \right)^n - \left(\frac{1 - \sqrt{5}}{2} \right)^n \right]$$

Пример времён вычисления в зависимости от трудоёмкости алгоритма

Сложность алгоритма	$n = 10$	$n = 10^3$	$n = 10^6$
$\log(n)$	1 с.	2 с.	5 с.
n	1 с.	1 мин. 40 с.	27 ч. 46 мин. 40 с.
n^2	1 с.	2 ч. 46 мин. 40 с.	~ 317 лет
2^n	1 с.	$\sim 40 \times 10^{21}$ лет	Очень много