

ХАРАКТЕРИСТИКА АЛГОРИТМИЧЕСКОЙ СЛОЖНОСТИ. ПОНЯТИЕ И СВОЙСТВА АЛГОРИТМИЧЕСКОЙ СЛОЖНОСТИ.



АЛГОРИТМИЧЕСКАЯ СЛОЖНОСТЬ

- связана с тем, насколько быстро или медленно работает конкретный алгоритм. Мы определяем сложность как числовую функцию $T(n)$ - время против входного размера n . Мы хотим определить время, затраченное алгоритмом, не завися от деталей реализации. Но $T(n)$ зависит от реализации. Данный алгоритм будет занимать различное количество времени на одних и тех же входах в зависимости от таких факторов, как: скорость процессора, набор команд, скорость диска, версия и тип компилятора и т. д. Обходной путь - **асимптотически** оценивать эффективность каждого алгоритма. Мы будем измерять время $T(n)$ как количество элементарных "шагов" (определенных любым образом), при условии, что каждый такой шаг занимает постоянное время.

АСИМПТОТИЧЕСКИЕ ОБОЗНАЧЕНИЯ

- Целью вычислительной сложности является классификация алгоритмов в соответствии с их производительностью. Мы представим функцию времени $T(n)$, используя **нотацию "big-O"** для выражения сложности времени выполнения алгоритма. Например, следующее утверждение

$$T(n) = O(n^2)$$

говорит, что алгоритм имеет квадратичную сложность по времени.

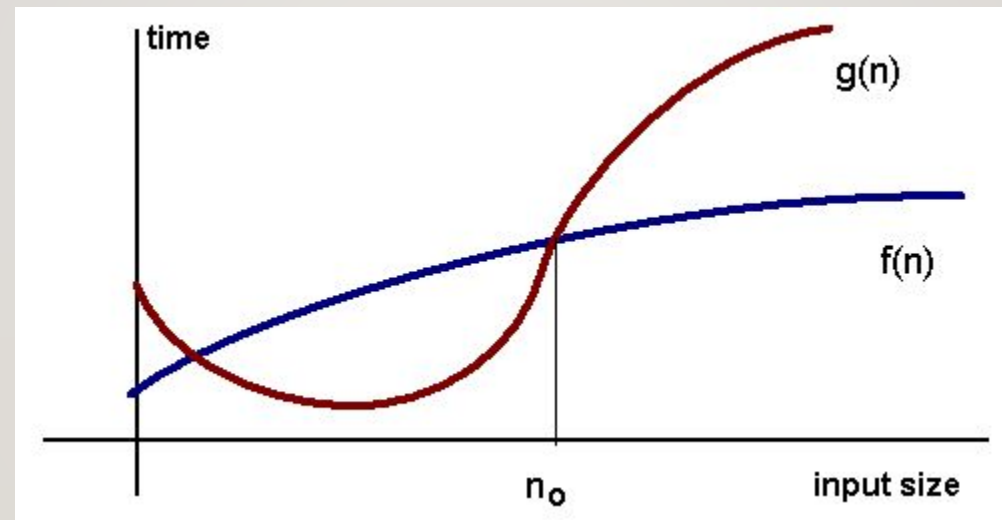
ОПРЕДЕЛЕНИЕ "BIG O"

- Для любых монотонных функций $f(n)$ и $g(n)$ от целых положительных чисел до целых положительных чисел говорят, что $f(n) = O(g(n))$, когда существуют такие константы $c > 0$ и $n_0 > 0$, так что

$$f(n) \leq c * g(n), \text{ для всех } n \geq n_0$$

- Интуитивно это означает, что функция $f(n)$ не растет быстрее, чем $g(n)$, или что функция $g(n)$ является верхней границей для $f(n)$ для всех достаточно больших $n \rightarrow \infty$

ГРАФИЧЕСКОЕ ПРЕДСТАВЛЕНИЕ ОТНОШЕНИЯ $F(N) = O(G(N))$:



ПРИМЕРЫ:

```
1 = O(n)
n = O(n*n)
log(n) = O(n)
2n + 1 = O(n)
```

"big-O" нотация не симметрична: $n = O(n^2)$ но $n^2 \neq O(n)$.

ПОСТОЯННОЕ ВРЕМЯ: $O(1)$

- Говорят, что алгоритм работает за постоянное время, если ему требуется одинаковое количество времени независимо от размера ввода. Примеры:
- массив: доступ к любому элементу
- стек фиксированного размера: методы `push` и `pop`
- очередь фиксированного размера: методы `enqueue` и `dequeue`

ЛИНЕЙНОЕ ВРЕМЯ: $O(N)$

- Говорят, что алгоритм работает за линейное время, если его выполнение по времени прямо пропорционально размеру ввода, то есть время увеличивается линейно с увеличением размера ввода.

Примеры:

- массив: линейный поиск, обход, нахождение минимума
- ArrayList: метод contains
- очередь: метод contains

ЛОГАРИФМИЧЕСКОЕ ВРЕМЯ: $O(\log N)$

- Говорят, что алгоритм работает за логарифмическое время, если его время выполнения пропорционально логарифму размера ввода.
Пример:
- бинарный поиск

КВАДРАТИЧНОЕ ВРЕМЯ: $O(N*N)$

- Говорят, что алгоритм работает за квадратичное время, если его время выполнения пропорционально квадрату размера ввода.

Примеры:

- сортировка по пузырькам, сортировка по выделению, сортировка по вставке

ОПРЕДЕЛЕНИЕ ПОНЯТИЯ "BIG OMEGA"

- Нам нужны обозначения для нижней границы. В этом случае используется заглавная омега Ω нотация. Мы говорим, что $f(n) = \Omega(g(n))$, когда существует постоянная c , для которой $f(n) \geq c * g(n)$ для всех достаточно больших n . Примеры:

$$n = \Omega(1)$$

$$n * n = \Omega(n)$$

$$n * n = \Omega(n \log(n))$$

$$2 * n + 1 = \Omega(n)$$

ОПРЕДЕЛЕНИЕ ПОНЯТИЯ "BIG THETA"

- Чтобы измерить сложность конкретного алгоритма, нужно найти верхнюю и нижнюю границы. В этом случае используется новая запись. Мы говорим, что $f(n) = \Theta(g(n))$ тогда и только тогда, когда $f(n) = O(g(n))$ и $f(n) = \Omega(g(n))$. Примеры:

$$2^n = \Theta(n)$$

$$n^2 + 2^n + 1 = \Theta(n^2)$$

АНАЛИЗ АЛГОРИТМОВ

- Наихудшая сложность алгоритма во время выполнения - это функция, определяемая максимальным количеством шагов, сделанных для любого экземпляра размера a .
- наилучшая сложность алгоритма во время выполнения - это функция, определяемая минимальным числом шагов, предпринимаемых для любого экземпляра размера a .
- средняя сложность времени выполнения алгоритма - это функция, определяемая средним числом шагов, предпринятых для любого экземпляра размера a .
- Амортизированная сложность алгоритма во время выполнения - это функция, определяемая последовательностью операций, применяемых к вводу размера a и усредняемой по времени.

РАССМОТРИМ АЛГОРИТМ ПОСЛЕДОВАТЕЛЬНОГО ПОИСКА В МАССИВЕ РАЗМЕРОМ N .

- Его наихудшая сложность во время выполнения - $O(n)$
- Его наилучшая сложность во время выполнения - $O(1)$
- Его средняя сложность во время выполнения составляет $O(n/2) = O(n)$