ЛЕКЦИЯ 09
O-НОТАЦИЯ

АЛГОРИТМИЗАЦИЯ И ПРОГРАММИРОВАНИЕ

СНОВА НАИВНЫЙ АЛГОРИТМ

Как оцените время работы этой функции:

```
bool is_prime(int n) {
    if (n < 2) return false;
    for (int i = 2; i * i <= n; ++i)
        if (n % i == 0) return false;
    return true;
}
```

СНОВА НАИВНЫЙ АЛГОРИТМ

Как оцените время работы этой функции:

```
bool is_prime(int n) {                         \\ t1
    if (n < 2) return false;                   \\ t2
    for (int i = 2; i * i <= n; ++i)           \\ sqrt(n)*t3
        if (n % i == 0) return false;          \\ sqrt(n)*t4
    return true;
}
```

ОБСУЖДЕНИЕ

Итак, время работы наивного алгоритма составит $k_1 + k_2\sqrt{n}$

Здесь n – проверяемое нами на простоту число

От чего зависят значения k_1 и k_2?

ОБСУЖДЕНИЕ

Итак, время работы наивного алгоритма составит $k_1 + k_2\sqrt{n}$

Здесь n – проверяемое нами на простоту число

От чего зависят значения k_1 и k_2?

- Быстродействие компьютера (ноутбук vs суперкомпьютер)
- Архитектура микропроцессора (насколько дорогой branch и детали реализации арифметики/логики)
- Качество компилятора и линковщика (насколько оптимизирован код)

На практике мы часто не знаем и не можем знать большую часть этих параметров

Но мы всегда знаем наш **главный параметр $\boldsymbol{n}$**!

ОБСУЖДЕНИЕ

Итак, время работы наивного алгоритма составит $k_1 + k_2\sqrt{n}$

Здесь n – проверяемое нами на простоту число

От чего зависят значения k_1 и k_2?

- Быстродействие компьютера (ноутбук vs суперкомпьютер)
- Архитектура микропроцессора (насколько дорогой branch и детали реализации арифметики/логики)
- Качество компилятора и линковщика (насколько оптимизирован код)

На практике мы часто не знаем и не можем знать большую часть этих параметров

Но мы всегда знаем наш **главный параметр $\boldsymbol{n}$**!

СНОВА НАИВНЫЙ АЛГОРИТМ

Для примера оценим выполнение наивного алгоритма нахождение чисел Фибоначчи:

```
unsigned long long fib(int n) {
    if (n == 0) return 0ull;
    if (n <= 2) return 1ull;
    return fib(n - 1) + fib (n - 2);
}
```

СНОВА НАИВНЫЙ АЛГОРИТМ

Для примера оценим выполнение наивного алгоритма нахождение чисел Фибоначчи:

```
unsigned long long fib(int n) {
    if (n == 0) return 0ull;             // t_a φ^n
    if (n <= 2) return 1ull;             // t_a φ^n
    return fib(n - 1) + fib (n - 2);     // (t_b + t_c) φ^n
}
```

СНОВА НАИВНЫЙ АЛГОРИТМ

Для примера оценим выполнение наивного алгоритма нахождение чисел Фибоначчи:

```
unsigned long long fib(int n) {
    if (n == 0) return 0ull;                 // tₐϕⁿ
    if (n <= 2) return 1ull;                 // tₐϕⁿ
    return fib(n - 1) + fib (n - 2);         // (t_b + t_c)ϕⁿ
}
```

Имеем ровно $\frac{1}{\sqrt{5}}\phi^n$ вызовов функции и общее время $k_3\phi^n$

Первая мысль при сравнении $k_1 + k_2 n$ и $k_3\phi^n$: а есть ли вообще разница какие значения имеют k_1, k_2, k_3?

СНОВА НАИВНЫЙ АЛГОРИТМ

Для примера оценим выполнение наивного алгоритма нахождение чисел Фибоначчи:

```
unsigned long long fib(int n) {
    if (n == 0) return 0ull;                // t_a ϕ^n
    if (n <= 2) return 1ull;                // t_a ϕ^n
    return fib(n - 1) + fib (n - 2);        // (t_b + t_c) ϕ^n
}
```

Имеем ровно $\frac{1}{\sqrt{5}}\phi^n$ вызовов функции и общее время $k_3\phi^n$

Первая мысль при сравнении $k_1 + k_2 n$ и $k_3\phi^n$: а есть ли вообще разница какие значения имеют k_1, k_2, k_3?

При достаточно большом n всегда $k_1 + k_2 n < k_3\phi^n$

О-НОТАЦИЯ

Базовая интуиция, что при достаточно большом n, выполняется:

$$k_6 < k_4 + k_5 \log(n) < k_1 + k_2 n < k_3 1.61^n$$

Получает свое развитие в О-нотации

$$f(n) = O\big(g(n)\big) \leftrightarrow \exists k, M | \forall n > k, M \cdot g(n) \geq |f(n)|$$

О-НОТАЦИЯ

Базовая интуиция, что при достаточно большом n, выполняется:

$$k_6 < k_4 + k_5 \log(n) < k_1 + k_2 n < k_3 1.61^n$$

Получает свое развитие в O-нотации

$$f(n) = O\big(g(n)\big) \leftrightarrow \exists k, M | \forall n > k, M \cdot g(n) \geq |f(n)|$$

Например $2x^3 + 16x - 1 = O(x^3)$

O-нотация не слишком строгая, поэтому эта же функция равна $O(x^4)$

О-НОТАЦИЯ

Базовая интуиция, что при достаточно большом n, выполняется:

$$k_6 < k_4 + k_5 \log(n) < k_1 + k_2 n < k_3 1.61^n$$

Получает свое развитие в О-нотации

$$f(n) = O\big(g(n)\big) \leftrightarrow \exists k, M | \forall n > k, M \cdot g(n) \geq |f(n)|$$

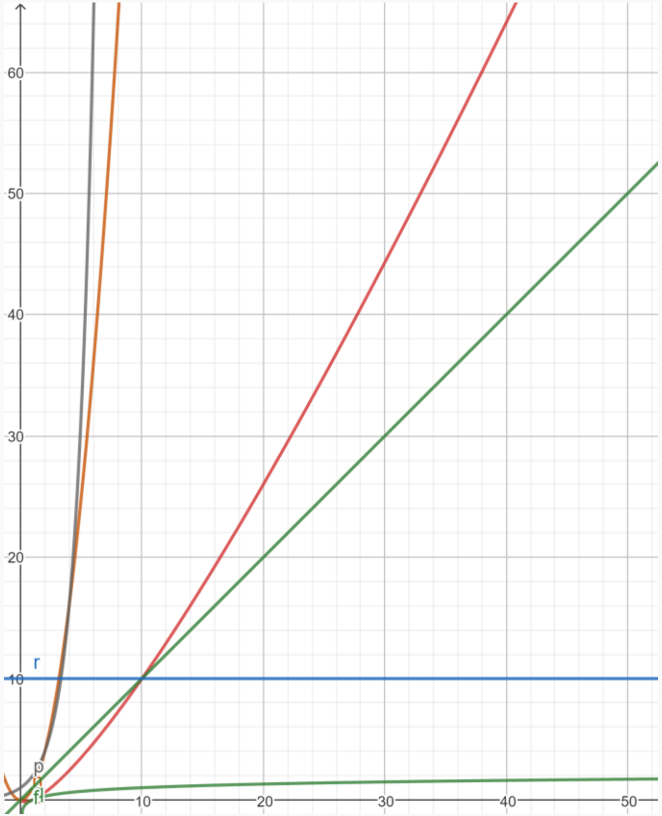
Например $2x^3 + 16x - 1 = O(x^3)$

О-нотация не слишком строгая, поэтому эта же функция равна $O(x^4)$

Говорят, что О-нотация отражает **асимптотику** зависимости **ресурса** (например времени работы алгоритма) от **главного параметра** в задаче (например номера числа Фибоначчи).

О-НОТАЦИЯ

	n	$n \log n$	n^2	2^n
10	1	1	1	1
50	1	1	1	13 days
10^6	1	1	15 min	∞
10^{10}	10 sec	2 min	3 years	∞
10^{16}	2 h	28 h	∞	∞



СНОВА НАИВНЫЙ АЛГОРИТМ

Простейший способ проверить число на простоту:

```
bool is_prime(int n) {
    if (n < 2) return false;
    for (int i = 2; i * i <= n; ++i)
        if (n % i == 0) return false;
    return true;
}
```

Асимптотическая сложность $O(\sqrt{n})$ и, кажется, его можно улучшить

УЛУЧШИМ НАИВНЫЙ АЛГОРИТМ

Используем тот факт, что четные всегда не простые, кроме 2:

```
bool is_prime(int n) {
    if (n == 2) return true;
    if ((n < 2) || (n % 2 == 0)) return false;
    for (int i = 3; i * i <= n; i+=2)
        if (n % i == 0) return false;
    return true;
}
```

Этот алгоритм **вдвое** быстрее предыдущего. Но какая у него асимптотика?

ЕЩЕ УЛУЧШИМ НАИВНЫЙ АЛГОРИТМ

Используем тот факт, что простые числа имеют вид $6k \pm 1$:

```
bool is_prime(int n) {
    if ((n == 2)||(n == 3)) return true;
    if ((n < 2)||(n % 2 == 0)||(n % 3 == 0)) return false;
    for (int i = 5; i * i <= n; i+=6)
        if ((n % i == 0)||(n % (i + 2) == 0) return false;
    return true;
}
```

Этот алгоритм **втрое** быстрее наивного. Но какая у него асимптотика?

СУТЬ АСИМПТОТИКИ

Асимптотическая сложность не измеряет время выполнения задачи

Она измеряет то, как **изменяется** время выполнения при изменении входных данных

ОБСУЖДЕНИЕ

До сих пор мы говорили только об одном ресурсе – **времени**. Но бывают и другие ресурсы. Ваши предложения?

ОБСУЖДЕНИЕ

До сих пор мы говорили только об одном ресурсе – **времени**.

Но бывают и другие ресурсы. Ваши предложения?

- **Память**
- Объем пересылаемых данных
- Сложность разработки в человеко-часах
- Стоимость лицензий для подключаемых библиотек
- Энергопотребление компьютера

Память второй по важности ресурс после времени

Разумеется, память – это ресурс, только в языках с явным управлением памятью. К счастью для всех нас, язык C++ именно такой.

ОБСУЖДЕНИЕ

Вам надо часто искать N-ое простое число для $0 < N < M$
Оцените асимптотику решета Эратосфена по памяти

Для N-ого просто числа есть математическая верхняя граница

$$\pi(n) \leq n(\log n + \log\log n), n \geq 20$$

ОБСУЖДЕНИЕ

Вам надо часто искать N-ое простое число для $0 < N < M$
Оцените асимптотику решета Эратосфена по памяти

Для N-ого просто числа есть математическая верхняя граница

$$\pi(n) \leq n(\log n + \log\log n), n \geq 20$$

Значит асимптотика по памяти поиска N-ого просто числа при помощи решета Эратосфена составляет $O(n \log n)$

УПРАЖНЕНИЯ

Оцените асимптотику следующих выражений

- $n + \log n + \sin n$
- $n^2 - 180n + 12$
- $5^{\log_2 n} + n^2\sqrt{n}$
- $n^{100} + 1.1^n$
- $n^3 - 1.1^{\sqrt{n}}$

УПРАЖНЕНИЯ

Оцените асимптотику следующих выражений

- $n + \log n + \sin n$ $\qquad O(n)$
- $n^2 - 180n + 12$ $\qquad O(n^2)$
- $5^{\log_2 n} + n^2\sqrt{n}$ $\qquad O(5^{\log_2 n})$
- $n^{100} + 1.1^n$ $\qquad O(1.1^n)$
- $n^3 - 1.1^{\sqrt{n}}$ $\qquad O(1.1^{\sqrt{n}})$

УПРАЖНЕНИЯ

Оцените асимптотику следующих выражений

- $n + \log n + \sin n$ $\quad O(n)$
- $n^2 - 180n + 12$ $\quad O(n^2)$
- $5^{\log_2 n} + n^2\sqrt{n}$ $\quad O(n^{2,5})$
- $n^{100} + 1.1^n$ $\quad O(1.1^n)$
- $n^3 - 1.1^{\sqrt{n}}$ $\quad O(1.1^{\sqrt{n}})$

Расположите выражения в порядке возрастания асимптотики

3^n	$n\log_2 n$	$\log_4 n$	n	$2^{\log_5 n}$	n^2	$\sqrt{n}$	2^{2n}

УПРАЖНЕНИЯ

Оцените асимптотику следующих выражений

- $n + \log n + \sin n$ $O(n)$
- $n^2 - 180n + 12$ $O(n^2)$
- $5^{\log_2 n} + n^2\sqrt{n}$ $O(n^{2,5})$
- $n^{100} + 1.1^n$ $O(1.1^n)$
- $n^3 - 1.1^{\sqrt{n}}$ $O(1.1^{\sqrt{n}})$

Расположите выражения в порядке возрастания асимптотики

3^n	$n\log_2 n$	$\log_4 n$	n	$2^{\log_5 n}$	n^2	$\sqrt{n}$	2^{2n}
8	5	1	4	2	6	3	7

НАИМЕНЬШЕЕ ОБЩЕЕ КРАТНОЕ

Число 2520 является наименьшим числом, которое делится без остатка на числа от 2 до 10.

Задача состоит в том, чтобы найти наименьшее число, которое делится без остатка на все числа от 2 до N.

Вам предлагаю наивный алгоритм: *идти от числа N вверх и каждое число проверять, делится ли оно каждое из чисел от 2 до N*

Оцените асимптотику наивного алгоритма

Подумайте, можно ли использовать алгоритм Евклида, чтобы улучшить это решение?

Математический инсайт:

$$lcm(a,b) = \frac{ab}{\gcd(a,b)}, lcm(a,b,c) = lcm(lcm(a,b),c)$$

ДОМАШНЕЕ ЗАДАНИЕ

1. Вам дано число 12385 и требуется найти наименьший куб больше этого числа. Результатом будет 13824. В другом случае дано число 1245678. Требуется найти 5-ую степень некоторого числа, которая будет наименьшей превышающей данной число. Результатом будет 1419857. Напишите функцию, которая принимает число N и степень pow, и находит наименьшее число, возведенное в степень pow, которое больше N.
2. Напишите функцию, которая принимает число N и возвращает количество различных делителей этого числа. (Для 4 это 3 (1, 2, 4), для 5 это 2 (1, 5))

РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА

1. Дорохова Т.Ю., Основы алгоритмизации и программирования : учебное пособие для СПО / Т.Ю. Дорохова, И.Е. Ильина. – Саратов, Москва : Профобразование, Ай, Пи Ар Медиа, 2022. – 139 с.
2. Кудинов Ю.И., Основы алгоритмизации и программирования : учебное пособие для СПО / Ю.И. Кудинов, А.Ю. Келина. – 2-е изд. – Липецк, Саратов: Липецкий государственный технический университет, Профообразование, 2020. – 71 с.
3. Дональд Кнут, Искусство программирования. Том 1. Основные алгоритмы / Ю.В. Козаченко. - 3-е изд – Москва, Санкт-Петербург: ВИЛЬЯМС, 2018. – 721 с.