OTUS

ОНЛАЙН-ОБРАЗОВАНИЕ

Теория пределов

Вычисление пределов функции.
Оценка сложности функции.

Лоскутов Антон Алексеевич

Меня хорошо слышно && видно?

Напишите в чат, если есть проблемы!

Ставьте + если все хорошо

Знакомство

- Заканчиваю механико-математический факультет МГУ им. Ломоносова
- Учился в Техносфере от Mail.Ru Group
- Являюсь ментором в Техносфере
- Работаю программистом-исследователем в Mail.Ru Group
- Веду лекции открытого курса mlcourse.ai

Правила вебинара

Активно участвуем

Задаем вопросы в чат

Off-topic обсуждаем в Slack

Вопросы вижу в чате, могу ответить не сразу

Цели занятия

После занятия вы сможете:

1 Вычислять различные пределы функций.

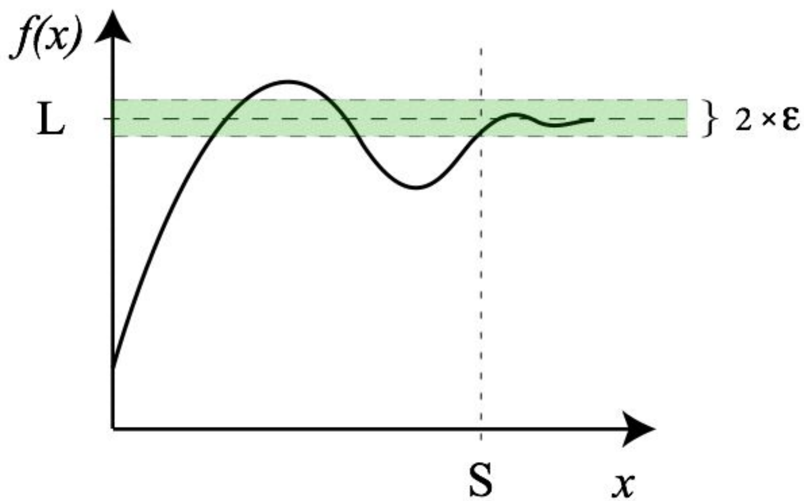
2 Разделять функции на некоторые классы эквивалентности.

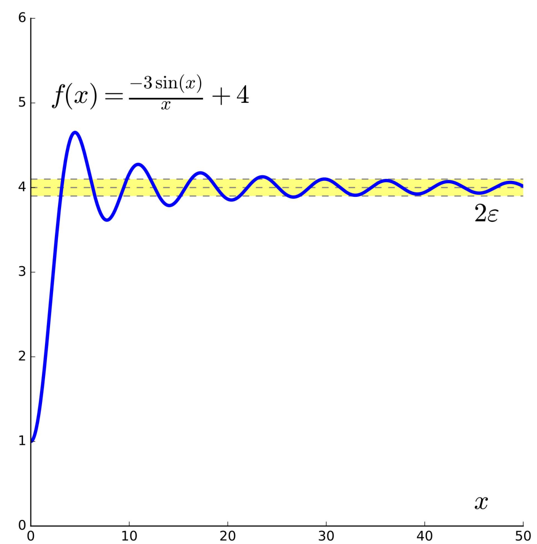
3 Сможете примерно вычислить время выполнения вашего кода.

Предел функции (повторение)

Определение (по Коши). Значение A называется пределом функции $f(x)$ в точке x_0, если выполняется равенство:

$$\lim_{x \to x_0} f(x) = A \Leftrightarrow \forall \varepsilon > 0 \; \exists \delta = \delta(\varepsilon) > 0 : \; \forall x \; 0 < |x - x_0| < \delta \Rightarrow |f(x) - A| < \varepsilon$$





Свойства предела функции (повторение)

1. $\left(\lim\limits_{x\to a} f(x) = A\right) \wedge \left(\lim\limits_{x\to a} g(x) = B\right) \Rightarrow \left(\lim\limits_{x\to a} \left[f(x) + g(x)\right] = A + B\right)$
2. $\left(\lim\limits_{x\to a} f(x) = A\right) \wedge \left(\lim\limits_{x\to a} g(x) = B\right) \Rightarrow \left(\lim\limits_{x\to a} \left[f(x) \cdot g(x)\right] = A \cdot B\right)$
3. $\left(\lim\limits_{x\to a} f(x) = A\right) \wedge \left(\lim\limits_{x\to a} g(x) = B \neq 0\right) \Rightarrow \left(\lim\limits_{x\to a} \left[\dfrac{f(x)}{g(x)}\right] = \dfrac{A}{B}\right)$

Теорема (о 2 милиционерах). Если функция $y=f(x)$ такая, что $\varphi(x) \leqslant f(x) \leqslant \psi(x)$ для всех x в некоторой окрестности точки a, причем функции $\varphi(x)$ и $\psi(x)$ имеют одинаковый предел равный A при $x \to a$, то:

$$\lim_{x\to a} \varphi(x) = \lim_{x\to a} \psi(x) = A \Rightarrow \lim_{x\to a} f(x) = A.$$

Ряд Тейлора (повторение)

Определение. Многочленом Тейлора функции $f(x)$ вещественной переменной x, дифференцируемой k раз в точке a называется функция:

$$\sum_{n=0}^{k} \frac{f^{(n)}(a)}{n!}(x-a)^n = f(a) + f'(a)(x-a) + \frac{f^{(2)}(a)}{2!}(x-a)^2 + \ldots + \frac{f^{(k)}(a)}{k!}(x-a)^k$$

Определение. Если функция $f(x)$ бесконечно дифференцируема, то многочлен Тейлора называется рядом Тейлора.

Определение. Если $a = 0$, то ряд Тейлора называется рядом Маклорена.

Бином Ньютона

Определение. Формула для разложения целой неотрицательной степени суммы двух слагаемой, задаваемая формулой ниже, называется биномом Ньютона.

$$(a+b)^n = \sum_{k=0}^{n} \binom{n}{k} a^{n-k} b^k = \binom{n}{0} a^n + \binom{n}{1} a^{n-1} b + \cdots + \binom{n}{k} a^{n-k} b^k + \cdots + \binom{n}{n} b^n$$

Определение. Число сочетаний или биномиальный коэффициент выражается формулой:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} = C_n^k$$

Замечание. Формула бинома Ньютона является частным случаем разложениям в ряд Тейлора для уравнения:

$$(1+x)^r = \sum_{k=0}^{\infty} \binom{r}{k} x^k$$

Видеолекция с выводом формулы бинома Ньютона:
https://ru.coursera.org/lecture/kombinatorika-dlya-nachinayushchikh/dokazatiel-stvo-formuly-binoma-n-iutona-J6tn6

Вычисление пределов

Все пределы рассматриваем при $x \to 0$

$$\lim \frac{e^x - 1 - x}{x^2} = \lim \frac{1 + x + \frac{x^2}{2} - 1 - x}{x^2} = \frac{1}{2}$$

Вычисление пределов

Все пределы рассматриваем при $x \to 0$

$$\lim \frac{\sqrt{1+2tg(x)}-e^2+x^2}{arctg(x)-sin(x)} = \frac{\frac{2x^3}{3}}{\frac{x^3}{3}} = 2$$

1. $arctg(x) - sin(x) = (x + \frac{x^3}{6}) - (x - \frac{x^3}{6}) = \frac{x^3}{3}$

2. $\sqrt{1 + 2tg(x)} = 1 + \frac{2tg(x)}{2} - \frac{(2tg(x))^2}{8} + \frac{(2tg(x))^3}{16} = 1 + tg(x) - \frac{tg^2(x)}{2} + \frac{tg^3(x)}{2}$
$= 1 + (x + \frac{x^3}{3}) - \frac{x^2}{2} + \frac{x^3}{2} = 1 + x - \frac{x^2}{2} + \frac{5x^3}{6}$

3. $\sqrt{1 + 2tg(x)} - e^x + x^2 = 1 + x - \frac{x^2}{2} + \frac{5x^3}{6} - 1 - x - \frac{x^2}{2} - \frac{x^3}{6} + x^2 = \frac{2x^3}{3}$

Вычисление пределов

Все пределы рассматриваем при $x \to 0$

$$\lim \frac{3x^2+x}{ln(1-2x)} = \lim \frac{(3x^2+x)'}{(ln(1-2x)')}$$

$$= \lim \frac{6x+1}{\frac{-2}{1-2x}} = -\frac{1}{2}$$

Все пределы рассматриваем при $x \to 0$

Вычислите пределы:

$$\lim \frac{x+ln(1+x)}{e^{3x}-1} = \frac{2}{3}$$

$$\lim \frac{cos(x)-1+\frac{x^2}{2}}{x^4} = \frac{1}{24}$$

Ряд Тейлора

OTUS

Определение. Многочленом Тейлора функции $f(x)$ вещественной переменной x, дифференцируемой k раз в точке a называется функция:

$$\sum_{n=0}^{k} \frac{f^{(n)}(a)}{n!}(x-a)^n = f(a) + f'(a)(x-a) + \frac{f^{(2)}(a)}{2!}(x-a)^2 + \ldots + \frac{f^{(k)}(a)}{k!}(x-a)^k$$

Определение. Если функция $f(x)$ бесконечно дифференцируема, то многочлен Тейлора называется рядом Тейлора.

Определение. Если $a = 0$, то ряд Тейлора называется рядом Маклорена.

Определение. Ряд Тейлора можно записать в следующей форме:

$$f(x) = \sum_{k=o}^{n} \frac{f^{(n)}(x_0)}{k!}(x-x_0)^k + \boxed{o(x-x_0)^n}$$

Остаточный член в форме Пеано

Подробнее о многочлене Тейлора и остаточных членах: https://www.kantiana.ru/mathematics/umk/analis18.pdf

Нотация О-большое и о-малое

OTUS

Определение. f является О-большим от g при $x \to x_0$, если существует такая константа $C > 0$, что для всех x из некоторой окрестности точки x_0 имеет место неравенство:

$$|f(x)| \leqslant C|g(x)|$$

Определение. f является о-малым от g при $x \to x_0$, если для любого $\varepsilon > 0$ в проколотой окрестности точки x_0 имеет место неравенство:

$$|f(x)| < \varepsilon|g(x)|$$

Подробнее о нотации О-большое с примерами (англ):
https://www.freecodecamp.org/news/big-o-notation-why-it-matters-and-why-it-doesnt-1674cfa8a23c/

Эквивалентные функции

Определение. Назовем 2 функции асимптотически эквивалентными, если их отношение стремится к некоторой константе.

Определение. Назовем 2 функции асимптотически эквивалентными, если предел их отношения равен некоторой константе.

Обозначение. Введем нотацию О-большого. $f(n) = O(g(n))$ означает, что функция f асимптотически эквивалентна функции g.

Теорема. Если $\lim_{x \to a} \frac{|f(x)|}{|g(x)|}$ конечен, то $f(x) = O(g(x))$ при том же условии.

Эквивалентные функции

Определение. Назовем 2 функции асимптотически эквивалентными, если их отношение стремится к некоторой константе.

Определение. Назовем 2 функции асимптотически эквивалентными, если предел их отношения равен некоторой константе.

Обозначение. Введем нотацию О-большого. $f(n) = O(g(n))$ означает, что функция f асимптотически эквивалентна функции g.

Теорема. Если $\lim_{x\to a} \frac{|f(x)|}{|g(x)|}$ конечен, то $f(x) = O(g(x))$ при том же условии.

Пример. $x + x^2 = O(x)$ или $O(x^2)$ при $x \to 0$?

$$\lim_{x\to 0} \frac{x + x^2}{x} = \lim_{x\to 0} 1 + x = 1.$$

Подробнее о нотации О-большое с примерами: http://math-hse.info/a/2014-15/ling-la/o_O.pdf

Примеры нотации О-большого и о-малого

1. $N + 2 = O(N)$ — Верно
2. $2\,N = O(N)$ — Верно
3. $N = O(N\hat{}2)$ — Верно, но лучше так не писать
4. $N\hat{}2 = O(N)$ — Неверно
5. $100 = O(N)$ — Верно, но лучше так не писать
6. $N = O(1)$ — Неверно
7. $214\,N + 34 = O(N\hat{}2)$ — Верно, но лучше так не писать
8. $100 = O(1)$ — Верно
9. $100\hat{}12 = O(1)$ — Верно

Сложность алгоритмов

Можно рассматривать сложность кода алгоритмов как функцию от размера входящих параметров.

Если мы подаем на вход алгоритму массив из n элементов, то можем обозначить его вычислительную сложность как $f(n)$.

Обозначение. $f(n) = O(g(n))$ означает, что функция f асимптотически эквивалентна функции g.

Теперь мы можем разделить все функции на некоторых классы эквивалентности.

Классы эквивалентности

Линейная сложность O(n). Такой сложностью, например, обладает алгоритм поиска элемента в не отсортированном массиве.

Квадратичная сложность O(n^2). Такой сложностью, например, обладает пузырьковая сортировка.

Константная сложность O(1). Такой сложностью обладают некоторые операции над структурами данных.

Суперполиномиальная сложность O(2^n). Такой сложностью не обладают правильно написанные стандартные алгоритмы. Если получили такую сложность, то стоит задуматься как его ускорить.

Примеры и сложности многих операций: https://habr.com/ru/post/188010/

Константная сложность

Константная сложность O(1). Такой сложностью обладают некоторые операции над структурами данных.

```
In [10]: a = 1

In [11]: arr[5] = 123

In [12]: b = a * (arr[5] + arr[0]) ** 2
```

Классы эквивалентности

OTUS

Линейная сложность O(n). Такой сложностью, например, обладает алгоритм поиска элемента в не отсортированном массиве.

Квадратичная сложность O(n^2). Такой сложностью, например, обладает пузырьковая сортировка.

In [24]:
```
def bubble_sort(array):

    # Генерируем длину массива
    N = len(array)

    # Проходимся по всем элементам списка
    for i in range(N-1):

        # Проходимся по всем элементам списка после i-го
        for j in range(N-i-1):

            # Если не выполняется сравнение, то меняем местами элементы
            if array[j] > array[j+1]:
                array[j], array[j+1] = array[j+1], array[j]
    return array
```

In [25]:
```
bubble_sort(list(reversed(range(5))))
```

Out[25]:
```
[0, 1, 2, 3, 4]
```

In [26]:
```
5 in list(range(10))
```

Out[26]:
```
True
```

Что делать с рекурсией?

Рекурсия

Простая рекурсия

В случае простой рекурсии сложность программы в общем случае можно оценить как $O(n\, f(n))$, где $O(f(n))$ - сложность рекурсивной функции.

Логарифмическая сложность

Логарифмическая сложность O(log(n)). Такой сложностью обладает бинарный поиск. То есть поиск элемента в отсортированном массиве. Рекурсивный алгоритм с приемлемой сложностью.

```
In [15]: def binary_search(array, n):

             # Генерируем индекс середины массива
             mid = int(len(array) / 2)

             # Проверяем не на середине ли наш искомый элемент
             if n == array[mid]:
                 return mid

             # Если нет, то рекурсивно ищем в левой стороне массива
             elif n > array[mid]:
                 return mid + binary_search(array[mid:], n)

             # Иначе рекурсивно ищем в правой стороне массива
             else:
                 return binary_search(array[:mid], n)

In [16]: binary_search([1,2,3,4,5,6,7], 3)

Out[16]: 2
```

Простая рекурсия

В случае простой рекурсии сложность программы в общем случае можно оценить как $O(n * f(n))$, где $O(f(n))$ - сложность рекурсивной функции.

Многократная рекурсия

В случае многократной рекурсии сложность программы в общем случае можно оценить как $O(k\hat{}n * f(n))$, где $O(f(n))$ - сложность рекурсивной функции и k - число вызовов рекурсии внутри функции.

Пример.

```
In [10]: def fibonacci(n):
             if n in [1, 2]:
                 return 1
             return fibonacci(n-1) + fibonacci(n-2)
```

$$O(2^n)$$

Рекурсия

Простая рекурсия

В случае простой рекурсии сложность программы в общем случае можно оценить как $O(n * f(n))$, где $O(f(n))$ - сложность рекурсивной функции.

Многократная рекурсия

В случае многократной рекурсии сложность программы в общем случае можно оценить как $O(k\hat{}n * f(n))$, где $O(f(n))$ - сложность рекурсивной функции и k - число вызовов рекурсии внутри функции.

Вывод

Рекурсия может быть полезна и аккуратна, если применять ее правильно, иначе могут быть большие проблемы.

Примеры

OTUS

Найдите сложность данных алгоритмов:

1.
```
In [6]: s = 0
        for x in range(n):
            s += x
```

$O(n)$

2.
```
In [8]: def factorial(n):
            if not n:
                return 1
            return factorial(n-1) * n

In [10]: factorial(n)
```

$O(n)$

3.*
```
In [36]: s = x = 0
         while n:
             x += 1
             s += factorial(x)
             n //= 2
```

$O(n*log(n))$

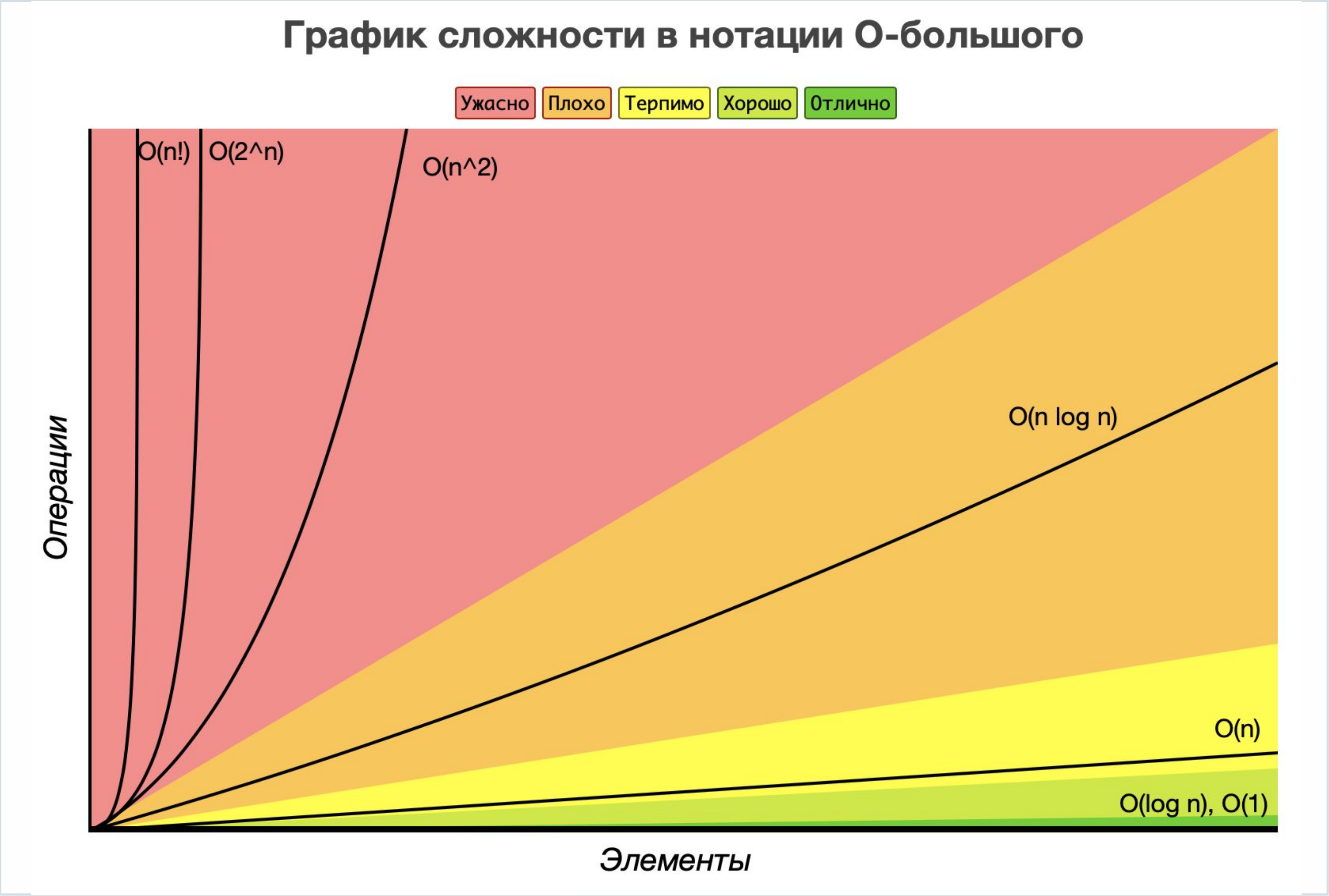
Время исполнения программы наглядно

OTUS

	10	20	30	40	50	60
O(n)	0.00001 сек	0.00002 сек	0.00003 сек	0.00004 сек	0.00005 сек	0.00006 сек
O(n^2)	0.0001 сек	0.0004 сек	0.0009 сек	0.0016 сек	0.0025 сек	0.0036 сек
O(n^3)	0.001 сек	0.008 сек	0.027 сек	0.064 сек	0.125 сек	0.216 сек
O(n^5)	0.1 сек	3.2 сек	24.3 сек	1.7 мин	5.2 мин	13 мин
O(2^n)	0.0001 сек	1 сек	17.9 мин	12.7 дней	35.7 веков	366 веков
O(3^n)	0.059 сек	58 мин	6.5 лет	3855 веков	2*10^8 веков	10^13 веков

Время исполнения программы наглядно

OTUS



Есть вопросы или замечания?

Напишите в чат свои вопросы и замечания!

Ставьте + если все понятно

Антон Лоскутов

Slack: @LoskutovAnton

Спасибо за внимание!