Алгоритмы и структуры данных

Бинарный поиск, анализ сложности алгоритмов

Задача

В массиве из уникальных элементов найти индекс элемент со значением x. Если такого нет, то вернуть -1.

Задача

В массиве из уникальных элементов найти индекс элемент со значением x. Если такого нет, то вернуть -1.

Input: array - массив уникальных чисел из N элементов. value - значение, которое нужно найти.

Output: i, если array[i] == value; -1 иначе.

Задача

В массиве из уникальных элементов найти индекс элемент со значением x. Если такого нет, то вернуть -1.

Input: array - массив уникальных чисел из N элементов. value - значение, которое нужно найти.

Output: i, если array[i] == value; -1 иначе.

Алгоритм: ?

Задача

В массиве из уникальных элементов найти индекс элемент со значением x. Если такого нет, то вернуть -1.

Input: array - массив уникальных чисел из N элементов. value - значение, которое нужно найти.

Output: i, если array[i] == value; -1 иначе.

Алгоритм: линейный поиск!

Линейный поиск

```
def find(array: List[int], x: int) -> int:
    for i in range(0, len(array)):
        if array[i] == x:
            return i
    return -1
```

Линейный поиск

```
int find(int arr[], int size, int x) {
    for (int i = 0; i < size; i++) {
        if (arr[i] == x) {
            return i;
        }
    }
    return -1;
}
```

Линейный поиск

```python
def find(array: List[int], x: int) -> int:
    for i in range(0, len(array)):
        if array[i] == x:
            return i
    return -1
```

Линейный поиск

```
def find(array: int[], x: int) -> int:
    for i in [0, len(array)):
        if array[i] == x:
            return i
    return -1
```

Псевдокод
(python-like)

Задача

В массиве из уникальных элементов найти индекс элемент со значением x. Если такого нет, то вернуть -1.

Input: array - массив уникальных чисел из N элементов. value - значение, которое нужно найти.

Output: i, если array[i] == value; -1 иначе.

Задача

В отсортированном по возрастанию массиве из уникальных элементов найти индекс элемент со значением x. Если такого нет, то вернуть -1.

1	2	7	13	35	37	90	93	99

Задача

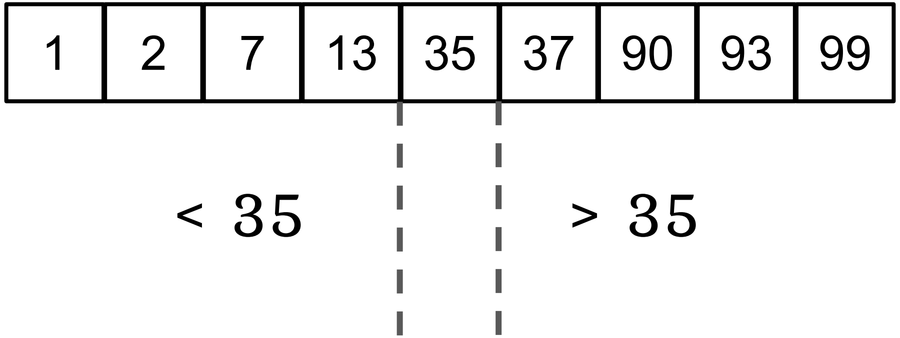
В отсортированном по возрастанию массиве из уникальных элементов найти индекс элемент со значением x. Если такого нет, то вернуть -1.

1	2	7	13	35	37	90	93	99

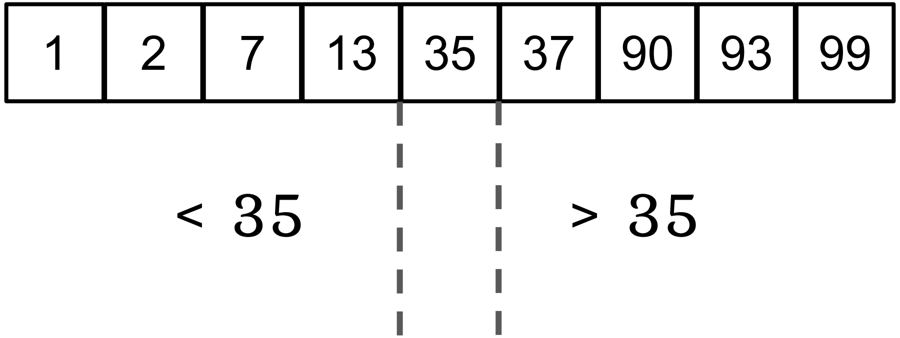
Линейный поиск прекрасно сработает.

Но можем ли мы лучше?

Бинарный поиск



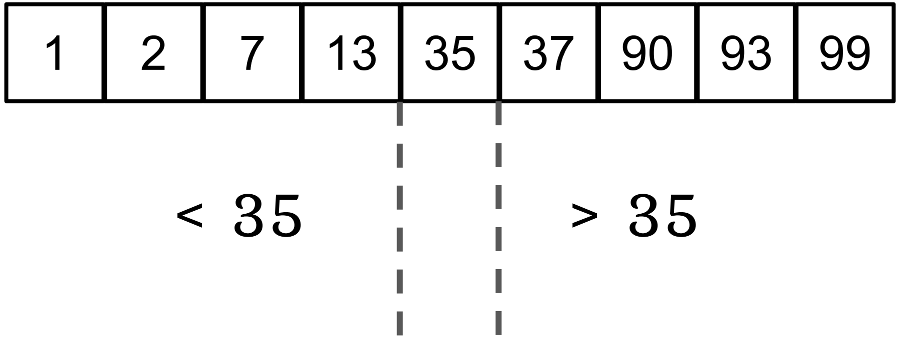
Бинарный поиск



Идея алгоритма:

1) Если элемент посередине* искомый => нашли ответ
2) Если искомый элемент меньше среднего* => искать слева
3) Если искомый элемент больше среднего* => искать справа

Бинарный поиск

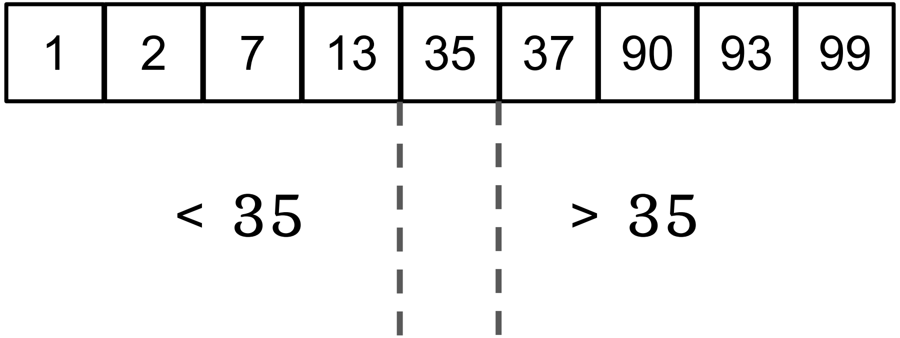


Идея алгоритма:

1) Если элемент посередине* искомый => нашли ответ
2) Если искомый элемент меньше среднего* => искать слева
3) Если искомый элемент больше среднего* => искать справа

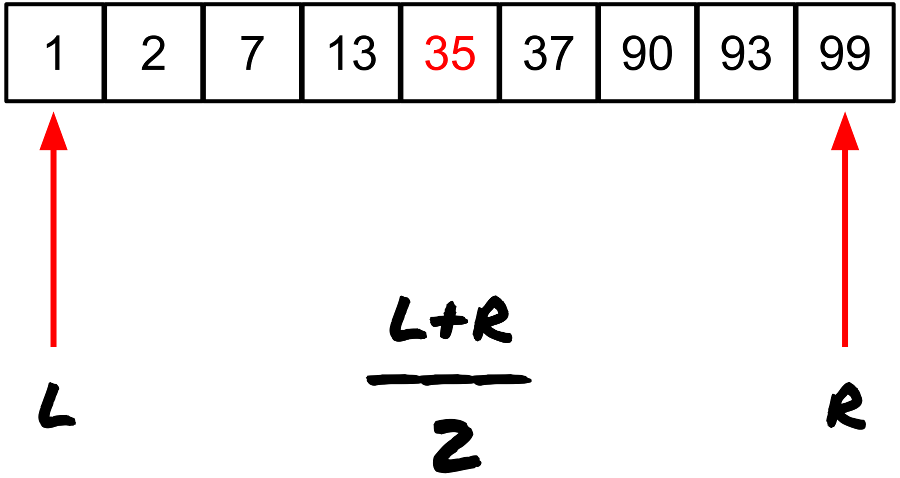
*средний элемент - элемент с индексом N/2

Бинарный поиск



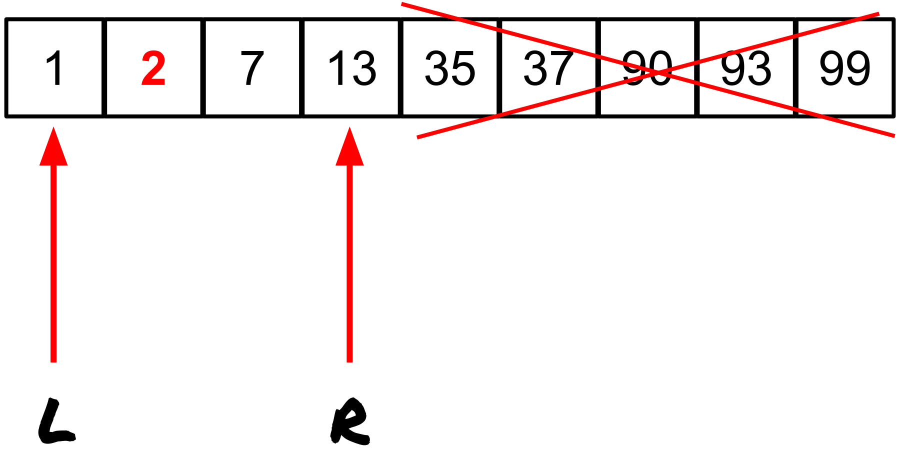
Пример: `найти индекс элемента 7`

Бинарный поиск



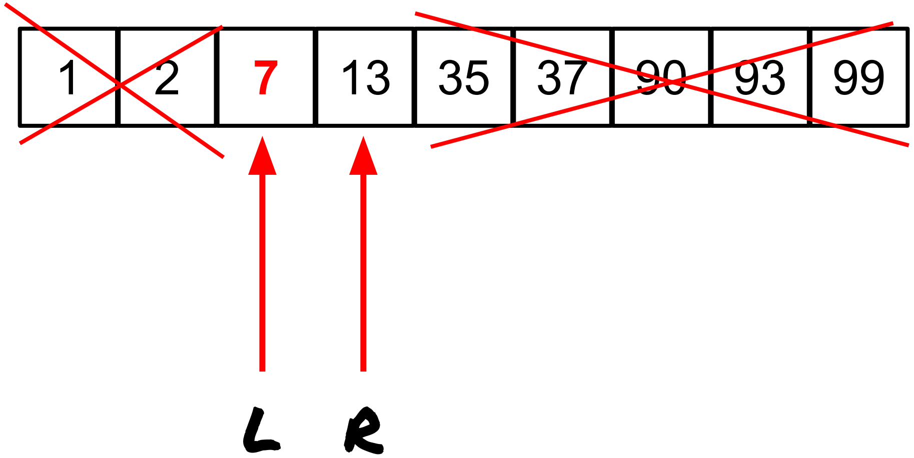
Пример: `найти индекс элемента 7`

Бинарный поиск



Пример: найти индекс элемента 7

Бинарный поиск



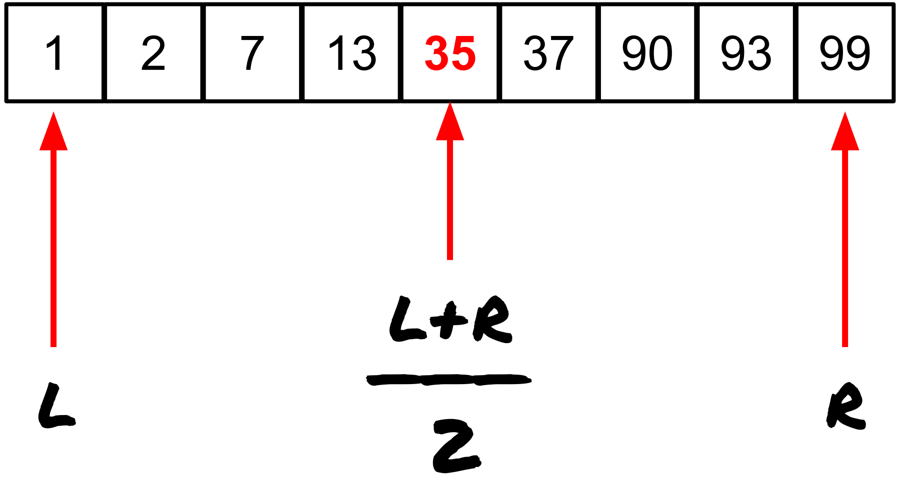
Пример: `найти индекс элемента 7`

Бинарный поиск

1	2	7	13	35	37	90	93	99

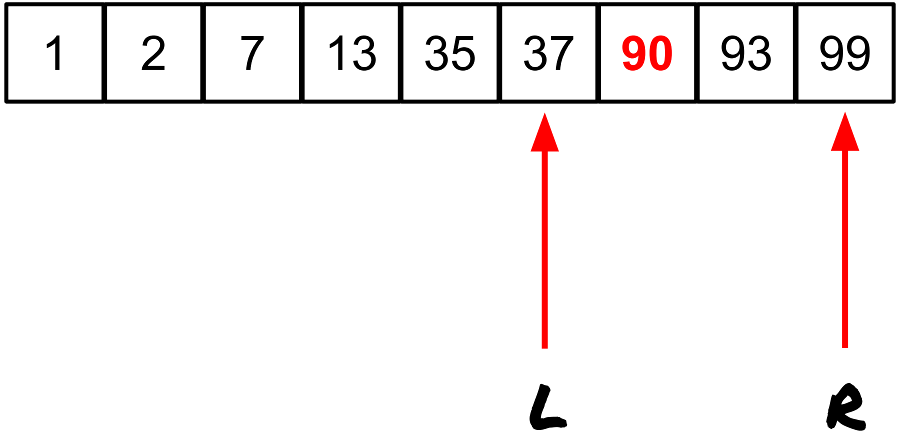
Пример: найти индекс элемента 137

Бинарный поиск



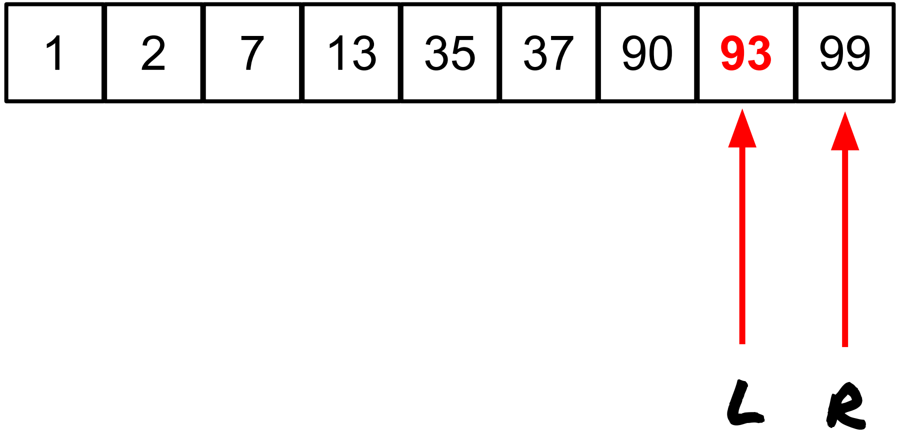
Пример: `найти индекс элемента 137`

Бинарный поиск



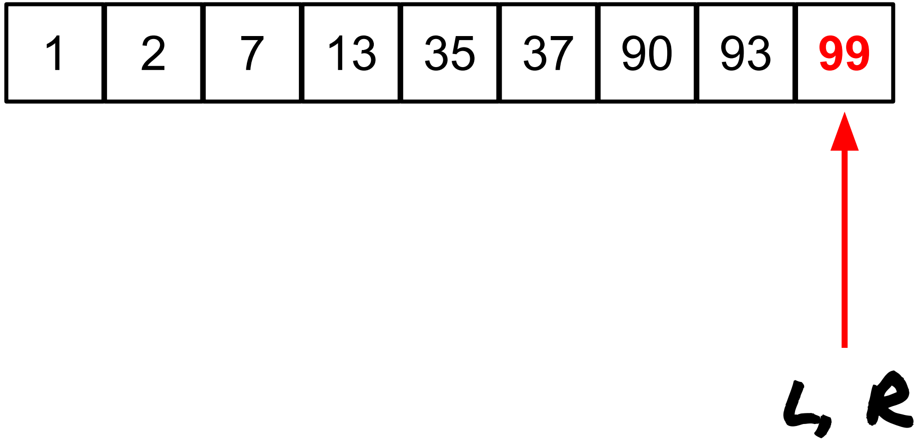
Пример: `найти индекс элемента 137`

Бинарный поиск



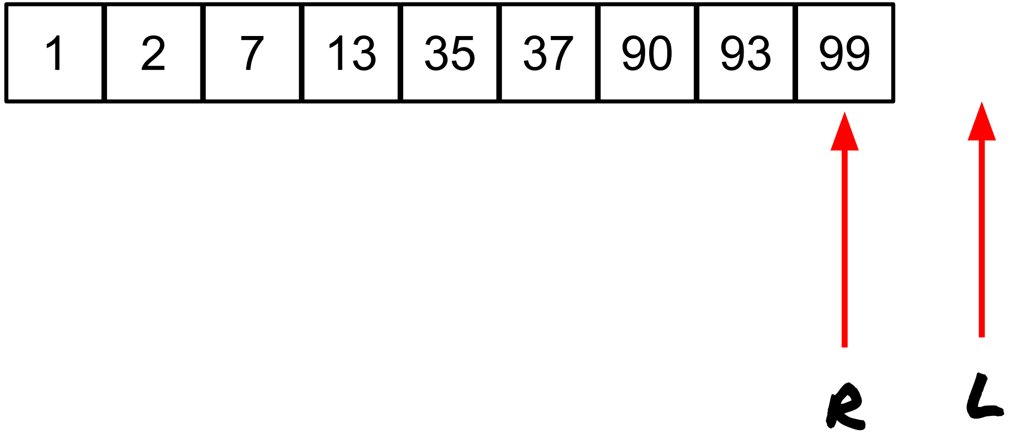
Пример: `найти индекс элемента 137`

Бинарный поиск



Пример: `найти индекс элемента 137`

Бинарный поиск



Пример: найти индекс элемента 137

Бинарный поиск

```
def binary_search(array: int[], l, r, x: int) -> int:
```

Бинарный поиск

```
def binary_search(array: int[], l, r, x: int) -> int:
    if l > r:
        return -1
```

Бинарный поиск

```
def binary_search(array: int[], l, r, x: int) -> int:
    if l > r:
        return -1

    pivot = (l + r) / 2
```

Бинарный поиск

```
def binary_search(array: int[], l, r, x: int) -> int:
    if l > r:
        return -1

    pivot = (l + r) / 2

    if array[pivot] == x:
        return pivot
```

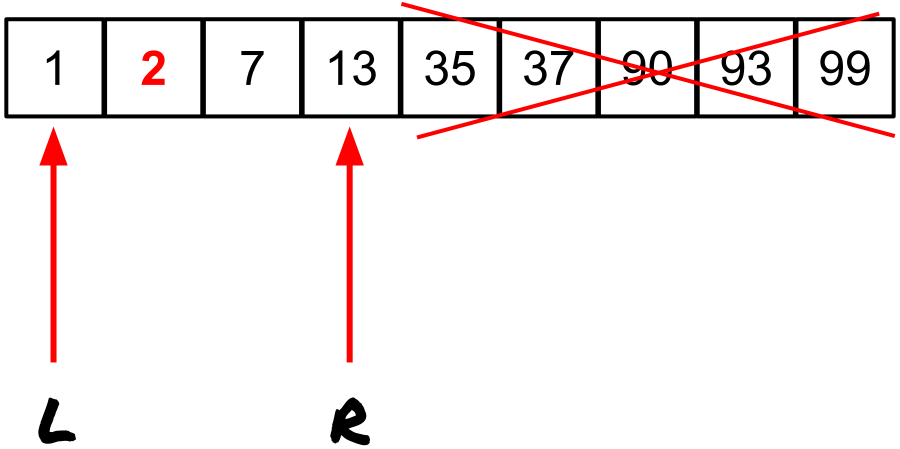
Бинарный поиск

```
def binary_search(array: int[], l, r, x: int) -> int:
    if l > r:
        return -1

    pivot = (l + r) / 2

    if array[pivot] == x:
        return pivot
    elif array[pivot] > x:
        return binary_search(array, l, pivot - 1, x)
```

Бинарный поиск



Пример: `найти индекс элемента 7`

Бинарный поиск

```
def binary_search(array: int[], l, r, x: int) -> int:
    if l > r:
        return -1

    pivot = (l + r) / 2

    if array[pivot] == x:
        return pivot
    elif array[pivot] > x:
        return binary_search(array, l, pivot - 1, x)
    else
        return binary_search(array, pivot + 1, r, x)
```

Бинарный поиск

```
def binary_search_impl(array: int[], l, r, x: int) -> int:
    ...

def binary_search(array: int[], x: int) -> int:
    binary_search_impl(array, …, …, x)
```

Бинарный поиск

```
def binary_search_impl(array: int[], l, r, x: int) -> int:
    ...

def binary_search(array: int[], x: int) -> int:
    binary_search_impl(array, 0, len(array) - 1, x)
```

Мини-задача #3 (1 балл)

Реализовать бинарный поиск без использования рекурсии.
Проверить свое решение на Leetcode:

https://leetcode.com/problems/binary-search/

Бинарный поиск

```
def binary_search_impl(array: int[], l, r, x: int) -> int:
    ...

def binary_search(array: int[], x: int) -> int:
    binary_search_impl(array, 0, len(array) - 1, x)
```

Ну и какой алгоритм лучше?
Линейный поиск или бинарный?

Бинарный поиск

```
def binary_search_impl(array: int[], l, r, x: int) -> int:
    ...

def binary_search(array: int[], x: int) -> int:
    binary_search_impl(array, 0, len(array) - 1, x)
```

Ну и какой алгоритм лучше?
Линейный поиск или бинарный?

Опять оценим количество элементарных операций

Основные принципы оценки:

1. Оцениваем количество операций в зависимости от размера входных данных

Основные принципы оценки:

1. Оцениваем количество операций в зависимости от размера входных данных

2. Оцениваем в худшем случае, т.е. для входных данных, требующих наибольшее количество операций

Основные принципы оценки:

1. Оцениваем количество операций в зависимости от размера входных данных

2. Оцениваем в худшем* случае, т.е. для входных данных, требующих наибольшее количество операций

*на самом деле не всегда,иногда интересна оценка и в среднем

Линейный поиск

```
def find(array: int[], x: int) -> int:
    for i in [0, len(array)):
        if array[i] == x:
            return i
    return -1
```

Худший случай?

Линейный поиск

```
def find(array: int[], x: int) -> int:
    for i in [0, len(array)):
        if array[i] == x:
            return i
    return -1
```

Худший случай?

Элемента нет => пришлось пройти N итераций

Линейный поиск

```
def find(array: int[], x: int) -> int:
    for i in [0, len(array)):
        if array[i] == x:
            return i
    return -1
```

сравнение и инкремент – две операции

Худший случай?

Элемента нет => пришлось пройти N итераций

Линейный поиск

```
def find(array: int[], x: int) -> int:
    for i in [0, len(array)):
        if array[i] == x:
            return i
    return -1
```

сравнение и инкремент – две операции

Худший случай?

Элемента нет => пришлось пройти N итераций => 4*N + K операций

Бинарный поиск

```
def binary_search(array: int[], l, r, x: int) -> int:
    if l > r:
        return -1

    pivot = (l + r) / 2

    if array[pivot] == x:
        return pivot
    elif array[pivot] > x:
        return binary_search(array, l, pivot - 1, x)
    else
        return binary_search(array, pivot + 1, r, x)
```

Худший случай?

Бинарный поиск

```
def binary_search(array: int[], l, r, x: int) -> int:
    if l > r:
        return -1

    pivot = (l + r) / 2

    if array[pivot] == x:
        return pivot
    elif array[pivot] > x:
        return binary_search(array, l, pivot - 1, x)
    else
        return binary_search(array, pivot + 1, r, x)
```

Худший случай?

Элемента нет => [log2(N)] вызовов

Бинарный поиск

```
def binary_search(array: int[], l, r, x: int) -> int:
    if l > r:
        return -1

    pivot = (l + r) / 2

    if array[pivot] == x:
        return pivot
    elif array[pivot] > x:
        return binary_search(array, l, pivot - 1, x)
    else
        return binary_search(array, pivot + 1, r, x)
```

Худший случай?

Элемента нет => [log2(N)] вызовов

~10 операций на каждом вызове!

Бинарный поиск

```
def binary_search(array: int[], l, r, x: int) -> int:
    if l > r:
        return -1

    pivot = (l + r) / 2

    if array[pivot] == x:
        return pivot
    elif array[pivot] > x:
        return binary_search(array, l, pivot - 1, x)
    else
        return binary_search(array, pivot + 1, r, x)
```

Оценка на количество операций:

10*[log2(N)] + C

Бинарный поиск

Ну и какой алгоритм лучше?
Линейный поиск или бинарный?

Опять оценим количество элементарных операций

Бинарный поиск

Ну и какой алгоритм лучше?
Линейный поиск или бинарный?

Опять оценим количество элементарных операций

Что лучше: $4*N + K$ или $10*[\log2(N)] + C$?

Бинарный поиск

Ну и какой алгоритм лучше?
Линейный поиск или бинарный?

Опять оценим количество элементарных операций

Что лучше: 4*N + K или 10*[log2(N)] + C ?

Пора переходить к асимптотике

Анализ асимптотики

На самом деле нам не очень интересны коэффициенты и младшие степени в оценке

А почему?

Анализ асимптотики

На самом деле нам не очень интересны коэффициенты и младшие степени в оценке

А почему?

1. Они очень зависят от языка программирования, компилятора, архитектуры

Линейный поиск

```
def find(array: int[], x: int) -> int:
    for i in [0, len(array)):
        if array[i] == x:
            return i
    return -1
```

сравнение и инкремент – две операции

А вы уверены, что здесь 4 операции в цикле? 🤔

Худший случай?

Элемента нет => пришлось пройти N итераций => 4*N + K операций

Анализ асимптотики

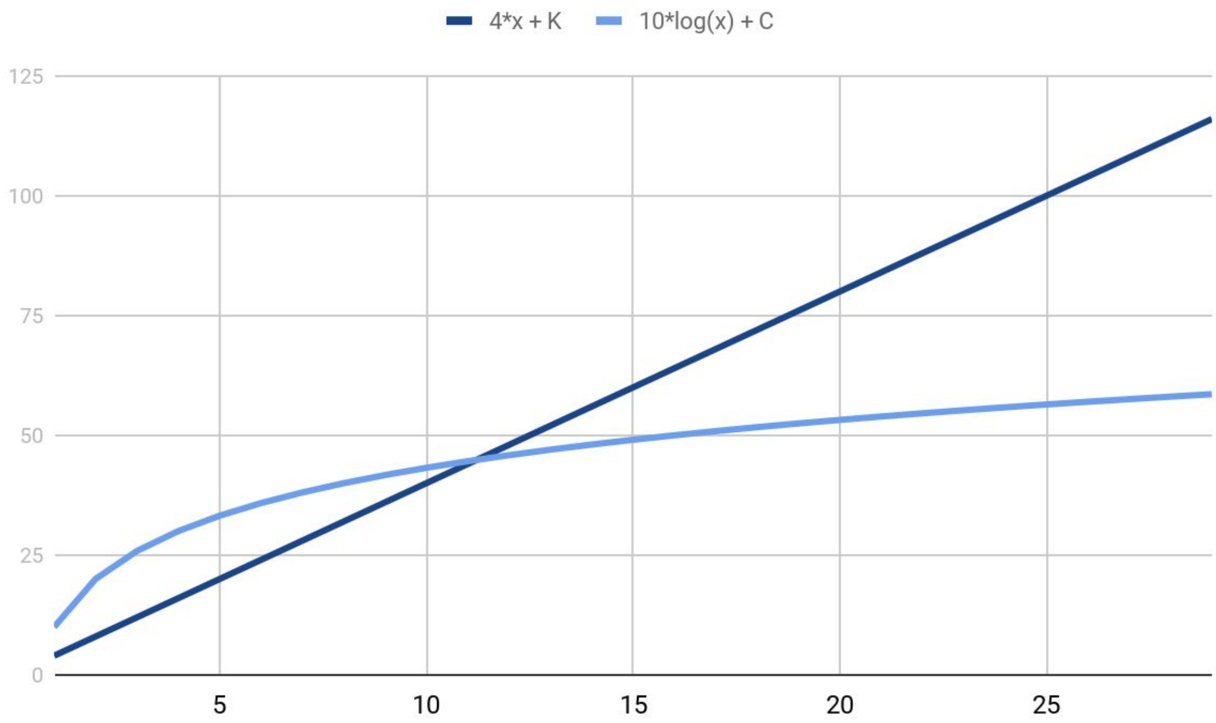
На самом деле нам не очень интересны коэффициенты и младшие степени в оценке

А почему?

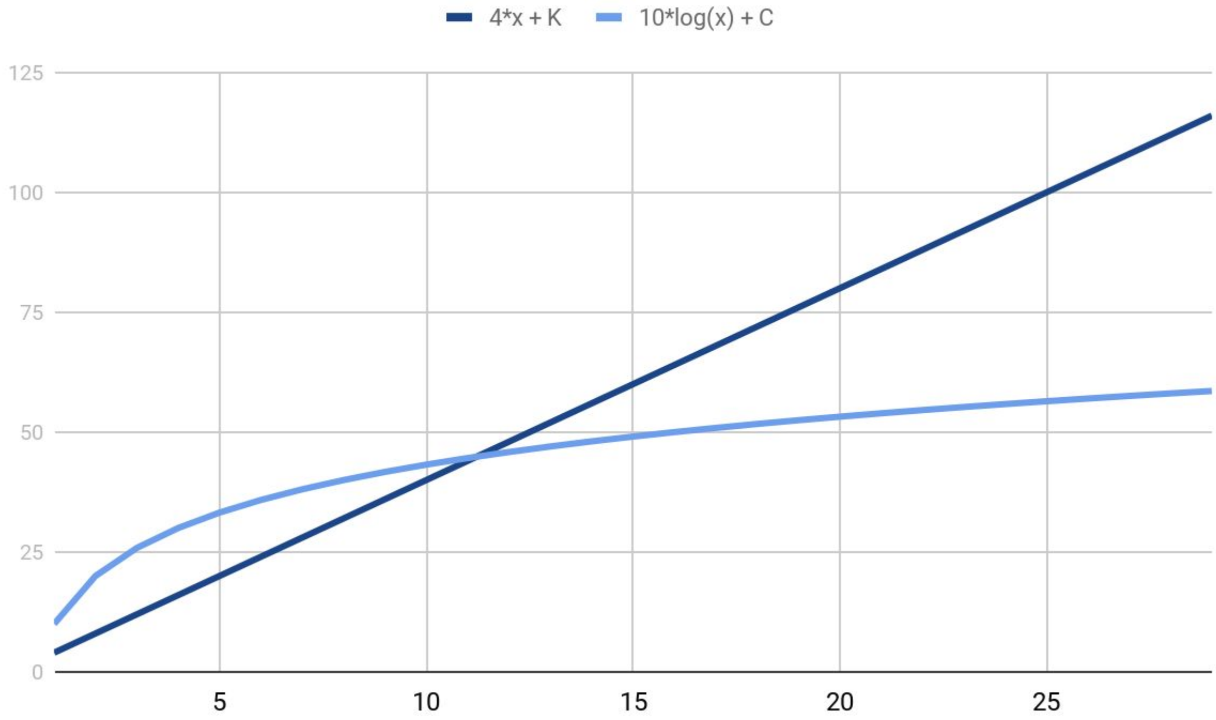
1. Они очень зависят от языка программирования, компилятора, архитектуры

2. Нам интереснее поведение алгоритма на больших входных данных

Анализ асимптотики



Анализ асимптотики



при достаточно больших x

4*x > 12*log(x)

для нас это самое главное

Основные принципы оценки:

1. Оцениваем количество операций в зависимости от размера входных данных

2. Оцениваем в худшем случае

3. Рассматриваем поведение функции для достаточно больших входных данных

Анализ асимптотики

Пусть есть функции `T(n)`, `f(n)`: `n = 1,2,3,..`

`T(n) = O(f(n))` ⇔

Анализ асимптотики

Пусть есть функции $T(n)$, $f(n)$: $n = 1,2,3,..$

$T(n) = O(f(n)) \Leftrightarrow$

$$\exists C,\ k > 0:\ \forall\ n \geq k \Rightarrow T(n) \leq C*f(n)$$

В нашем случае $T(n)$ - количество операций для входных данных размера n

Анализ асимптотики

$$T(n) = 4 * n^2 + 200$$

$$T(n) = O(f(n))$$

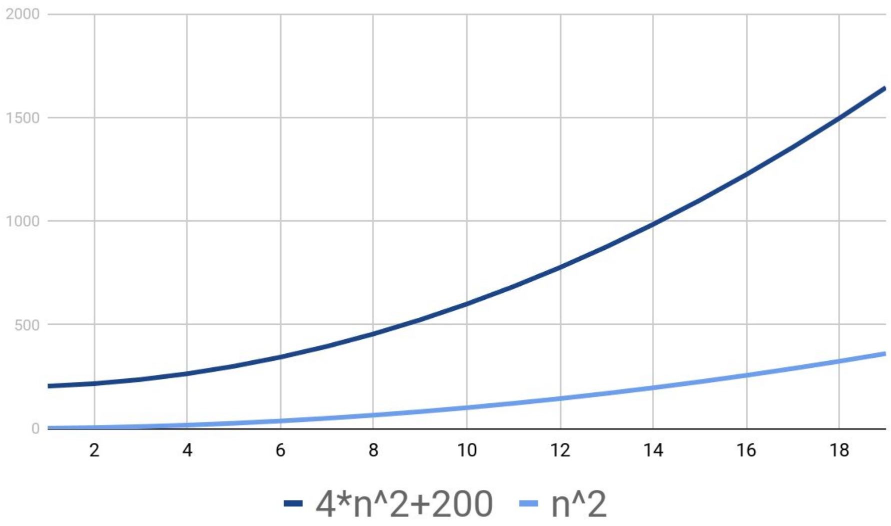
$$f(n) = ?$$

Анализ асимптотики

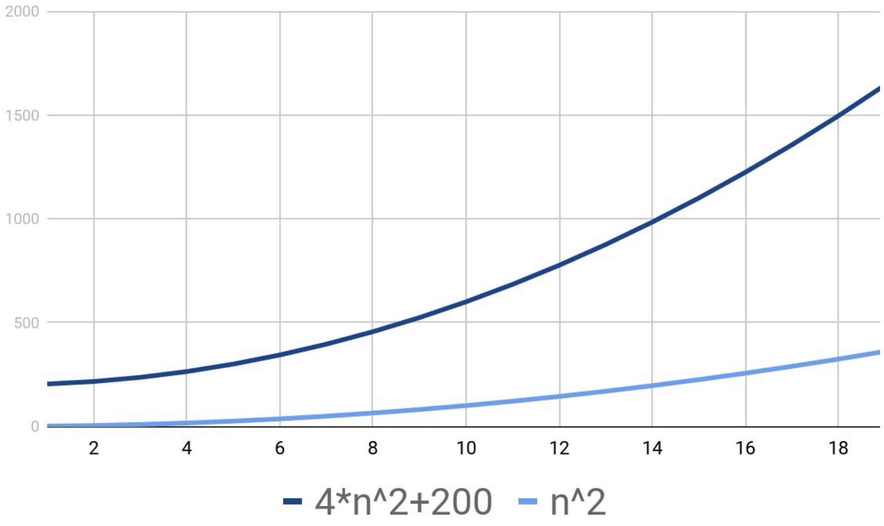
$$T(n) = 4 * n^2 + 200$$

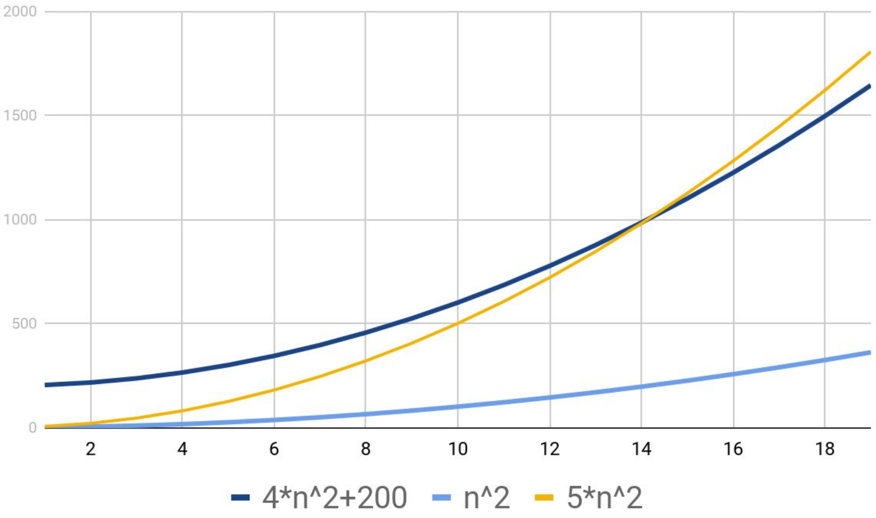
$$T(n) = O(f(n))$$

$$f(n) = ?$$



Анализ асимптотики





$$T(n) = 4 * n^2 + 200; \;\; T(n) = O(n^2)$$

Анализ асимптотики

Упражнение: доказать, что если

$$T(n) = a_k * n^k + a_{k-1} * n^{k-1} + \ldots + a_1 * n + a_0$$

то

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

Анализ асимптотики

$$T(n) = a_k * n^k + a_{k-1} * n^{k-1} + \dots + a_1 * n + a_0$$

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

Анализ асимптотики

T(n) = $a_k * n^k + a_{k-1} * n^{k-1} + \ldots + a_1 * n + a_0$

T(n) = O(f(n)) ⇔ ∃C, p > 0:
∀ n ≥ p ⇒ T(n) ≤ C*f(n)

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

Анализ асимптотики

T(n) = $a_k * n^k + a_{k-1} * n^{k-1} + \ldots + a_1 * n + a_0$

T(n) = O(f(n)) ⇔ ∃C, p > 0:
∀ n ≥ p ⇒ T(n) ≤ C*f(n)

Возьмем p = 1; C = $|a_k| + |a_{k-1}| + \ldots + |a_0|$

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

Анализ асимптотики

T(n) = $a_k * n^k + a_{k-1} * n^{k-1} + \ldots + a_1 * n + a_0$

T(n) = O(f(n)) ⇔ ∃C, p > 0:
∀ n ≥ p ⇒ T(n) ≤ C*f(n)

Возьмем p = 1; C = $|a_k| + |a_{k-1}| + \ldots + |a_0|$

T(n) ≤ $|a_k| * n^k + |a_{k-1}| * n^{k-1} + \ldots + |a_1| * n + |a_0|$

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

Анализ асимптотики

$$T(n) = a_k * n^k + a_{k-1} * n^{k-1} + \dots + a_1 * n + a_0$$

$T(n) = O(f(n)) \Leftrightarrow \exists C, p > 0:$
$\forall n \geq p \Rightarrow T(n) \leq C*f(n)$

Возьмем $p = 1$; $C = |a_k| + |a_{k-1}| + \dots + |a_0|$

$$T(n) \leq |a_k| * n^k + |a_{k-1}| * n^k + \dots + |a_1| * n^k + |a_0| * n^k$$

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

Анализ асимптотики

$$T(n) = a_k * n^k + a_{k-1} * n^{k-1} + \ldots + a_1 * n + a_0$$

$T(n) = O(f(n)) \Leftrightarrow \quad \exists C,\ p > 0:$

$\qquad \forall\ n \ge p \Rightarrow T(n) \le C*f(n)$

Возьмем $p = 1$; $C = |a_k| + |a_{k-1}| + \ldots + |a_0|$

$$T(n) \le |a_k| * n^k + |a_{k-1}| * n^k + \ldots + |a_1| * n^k + |a_0| * n^k$$

$$\le C * n^k$$

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

Анализ асимптотики

Упражнение: доказать, что если T(n) = n^k

то T(n) не является $O(n^{k-1})$

Бинарный поиск

Ну и какой алгоритм лучше?
Линейный поиск или бинарный?

Опять оценим количество элементарных операций

Что лучше: 4*N + K или 10*[log2(N)] + C ?

Пора переходить к асимптотике

Бинарный поиск

Ну и какой алгоритм лучше?
Линейный поиск или бинарный?

Опять оценим количество элементарных операций

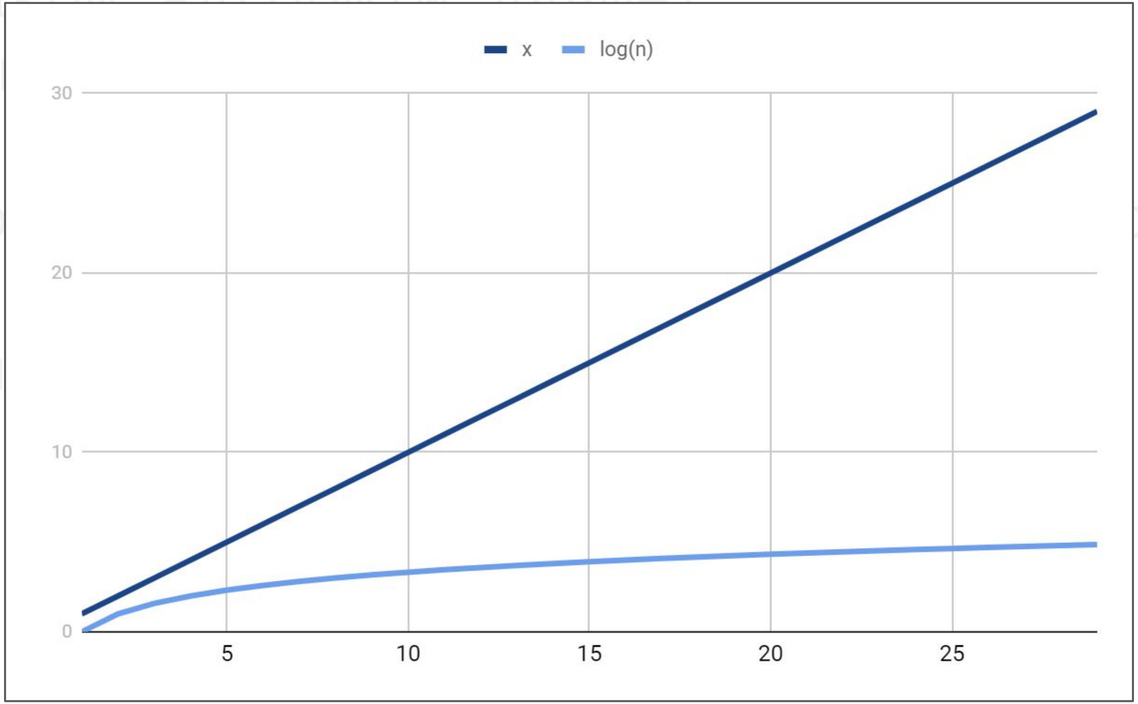
Что лучше: $O(N)$ или $O(\log2(N))$?

Бинарный поиск

Ну и какой алгоритм лучше?
Линейны

Опять о операций

Что луч



Бинарный поиск

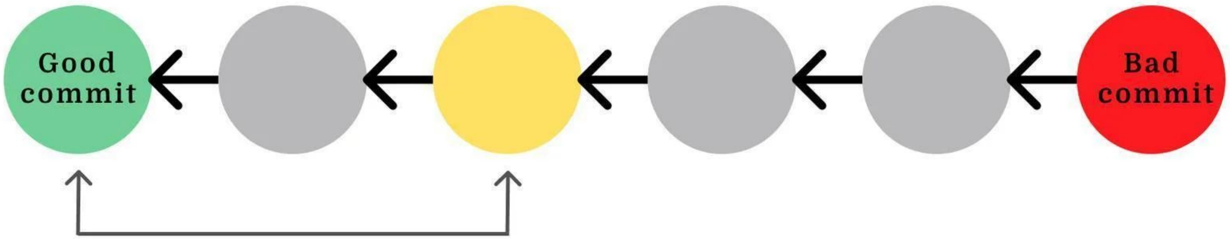
Ну и какой алгоритм лучше?
Линейный поиск или бинарный?

Опять оценим количество элементарных операций

Что лучше: O(N) или O(log2(N))? Конечно логарифм!

Бинарный поиск: практическое применение

GIT BISECT



Git helps to narrow down the bug by checking out a commit in the middle.

The bug can be either in the left or right half of the commit range.

https://git-scm.com/docs/git-bisect

Мини-задача #4 (2 балла)

Самостоятельно реализовать команду git bisect.
Программа (скрипт) должна принимать на вход:

1. Путь до директории с git репозиторием
2. Два хэша коммитов, задающих диапазон поиска
3. Команду, которую нужно выполнить для проверки ошибок на определенном коммите (если команда возвращает error code отличный от 0, то коммит - плохой)

Протестировать решение на специально созданном репозитории с историей коммитов.

КВИЗ!

Сложность алгоритма?

```
def max(matrix: int[][]) -> int:
    max = matrix[0][0]
    for i in [0, len(matrix)):
        for j in [0, len(matrix[i])):
            if matrix[i][j] > max:
                max = matrix[i][j]
    return max
```

Сложность для входной матрицы N*M?

Сложность алгоритма?

```
def max(matrix: int[][]) -> int:
    max = matrix[0][0]
    for i in [0, len(matrix)):
        for j in [0, len(matrix[i])):
            if matrix[i][j] > max:
                max = matrix[i][j]
    return max
```

Сложность для входной матрицы N*M? O(N*M)

Сложность алгоритма?

```
def max(matrix: int[][]) -> int:
    max = matrix[0][0]
    for i in [0, len(matrix)):
        for j in [0, len(matrix[i])):
            if matrix[i][j] > max:
                max = matrix[i][j]
                if max > 10:
                    return max
    return max
```

Сложность алгоритма?

```
def max(matrix: int[][]) -> int:
    max = matrix[0][0]
    for i in [0, len(matrix)):
        for j in [0, len(matrix[i])):
            if matrix[i][j] > max:
                max = matrix[i][j]
                if max > 10:
                    return max
    return max
```

снова O(N*M) (в худшем ведь!)

Сложность алгоритма?

```
def multiply(a, b: int[][]) -> int[][]:
    res = int[len(a)][len(b[0])]
    for i in [0, len(a)):
        for j in [0, len(b[0])):
            for k in [0, len(b)):
                res[i][j] += a[i][k] * b[k][j]
    return res
```

Сложность для входных матриц N*M и M*P?

Сложность алгоритма?

```
def multiply(a, b: int[][]) -> int[][]:
    res = int[len(a)][len(b[0])]
    for i in [0, len(a)):
        for j in [0, len(b[0])):
            for k in [0, len(b)):
                res[i][j] += a[i][k] * b[k][j]
    return res
```

Сложность для входных матриц N*M и M*P? O(N*M*P)

Сложность алгоритма?

Сложность умножения двух
чисел из N цифр в столбик?

Сложность алгоритма?

Сложность умножения двух чисел из N цифр в столбик?

$$O(N^2)$$

Сложность алгоритма?

Сложность умножения двух чисел из N цифр в столбик?

$O(N^2)$

Сложность умножения двух чисел из N цифр методом Карацубы?

Все еще "скоро узнаете"

Сложность алгоритма?

Сложность алгоритма поиска в глубину
в графе из N вершин и M ребер?

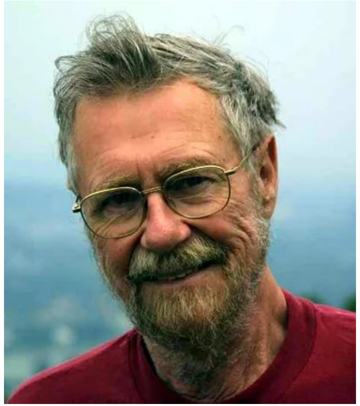
Сложность алгоритма?

Сложность алгоритма поиска в глубину
в графе из N вершин и M ребер?

$O(N + M)$ т.е. линейное!

Сложность алгоритма?

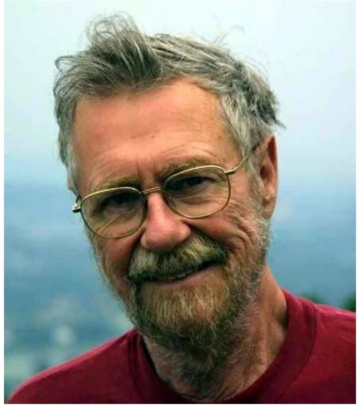
Сложность алгоритма Дейкстры поиска кратчайших путей в графе из N вершин и M ребер?



Сложность алгоритма?

Сложность алгоритма Дейкстры поиска кратчайших путей в графе из N вершин и M ребер?

В наивной реализации: $O(N^2)$



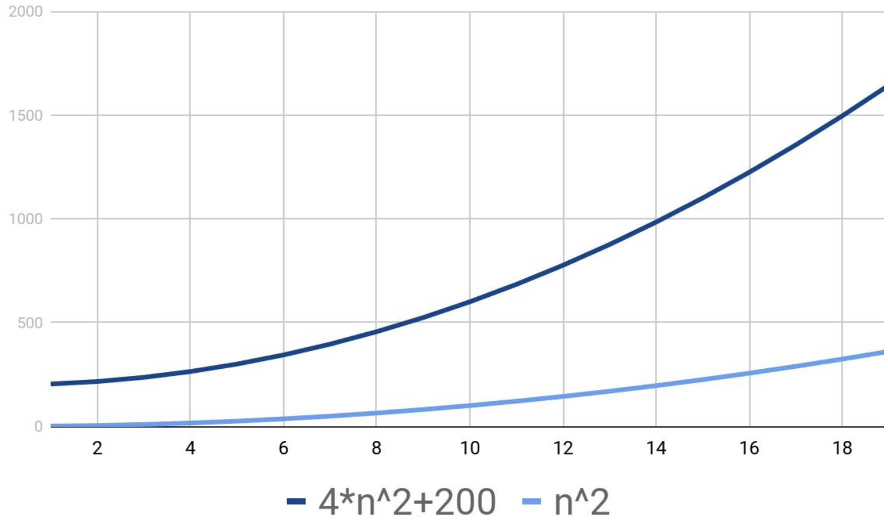
Сложность алгоритма?

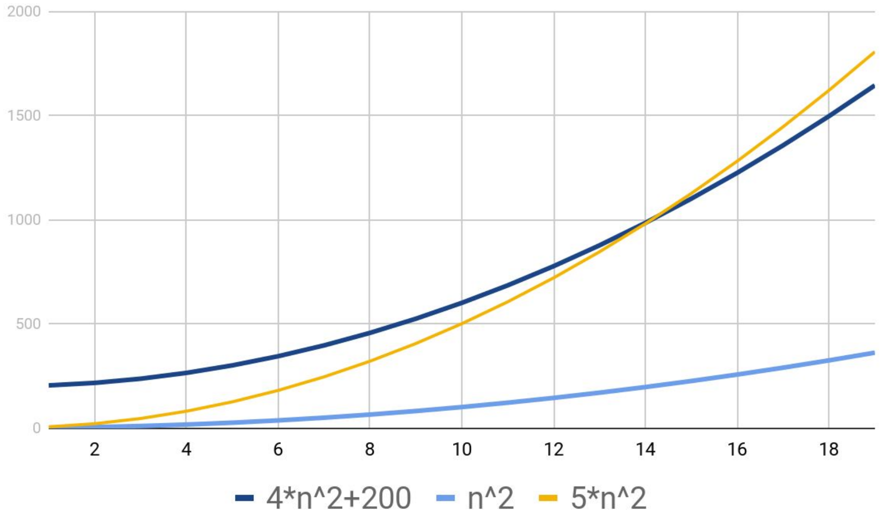
Сложность алгоритма Дейкстры поиска кратчайших путей в графе из N вершин и M ребер?

В наивной реализации: $O(N^2)$

Но можно достичь и: $O((N + M) * logN))$

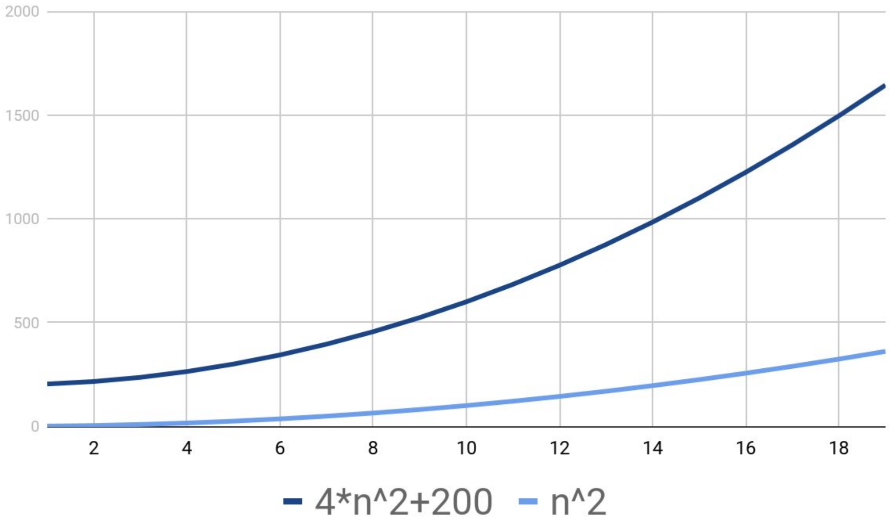
Анализ асимптотики





$$T(n) = 4 * n^2 + 200; \;\; T(n) = O(n^2)$$

Анализ асимптотики



$$T(n) = 4 * n^2 + 200;$$

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

А верно ли, что:

$$T(n) = O(n^3)?$$

Анализ асимптотики

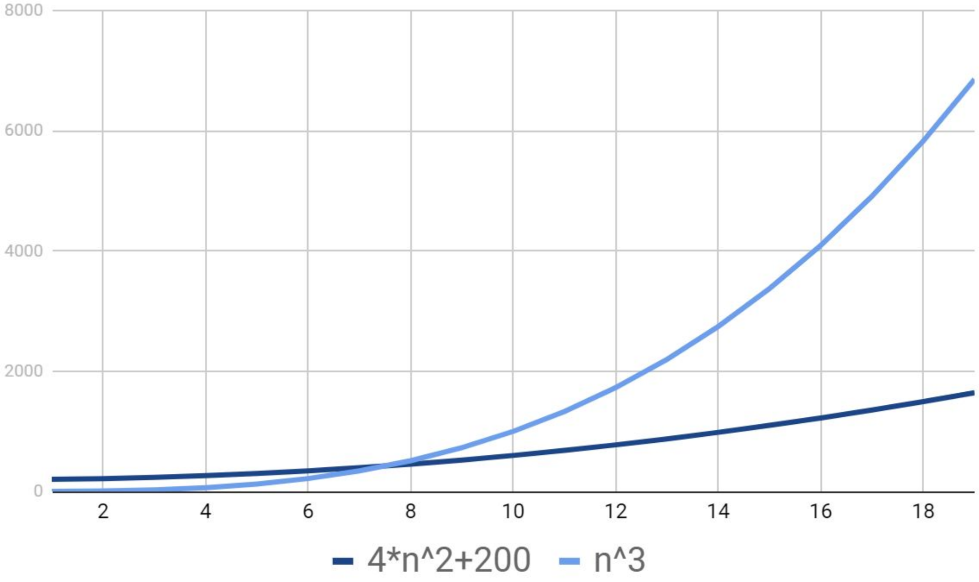
Пусть есть функции $T(n)$, $f(n)$: $n = 1,2,3,..$

$T(n) = O(f(n)) \Leftrightarrow$

$$\exists C,\ k > 0:\ \forall\ n \geq k \Rightarrow T(n) \leq C*f(n)$$

В нашем случае $T(n)$ - количество операций для входных данных размера n

Анализ асимптотики



$$T(n) = 4 * n^2 + 200;$$

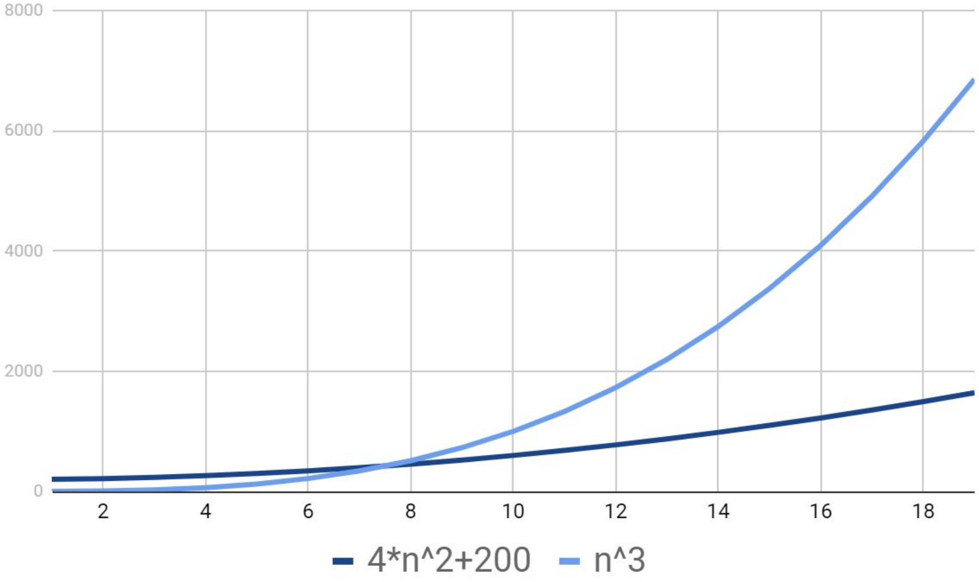
$$T(n) = O(n^2)$$

А верно ли, что:

$$T(n) = O(n^3)?$$

Конечно да!

Анализ асимптотики



$$T(n) = 4 * n^2 + 200;$$

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

А верно ли, что:

$$T(n) = O(n^3)?$$

Конечно да! Так почему мы тогда говорим только про оценку сверху?

Анализ асимптотики

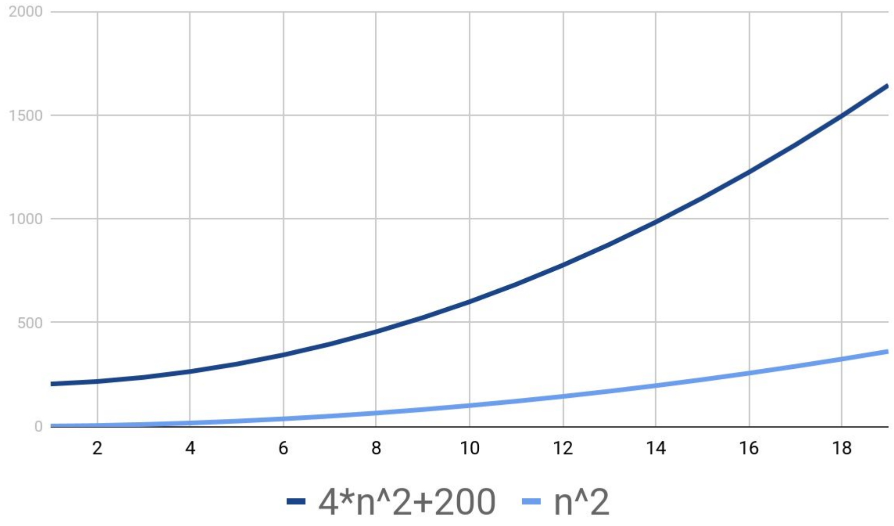
Пусть есть функции $T(n)$, $f(n)$: $n = 1,2,3,..$

$T(n) = \Omega(f(n)) \Leftrightarrow$

$$\exists C, k > 0: \forall n \geq k \Rightarrow T(n) \geq C*f(n)$$

В нашем случае $T(n)$ - количество операций для входных данных размера n

Анализ асимптотики

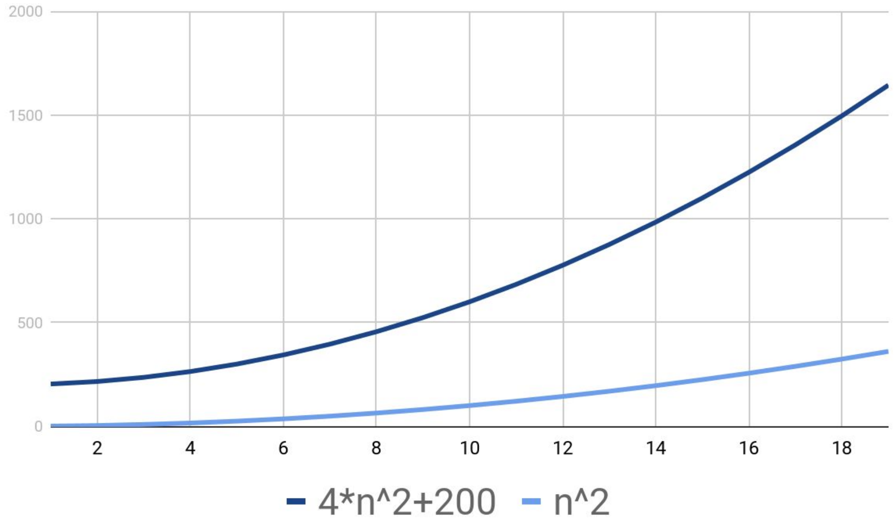


$$T(n) = 4 * n^2 + 200;$$

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

$$T(n) = \Omega(?)$$

Анализ асимптотики



$$T(n) = 4 * n^2 + 200;$$

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

$$T(n) = \Omega(n^2)$$

Анализ асимптотики

Пусть есть функции $T(n)$, $f(n)$: $n = 1,2,3,..$

$$T(n) = \Theta(f(n)) \Leftrightarrow T(n) = O(f(n)) \text{ и } T(n) = \Omega(f(n))$$

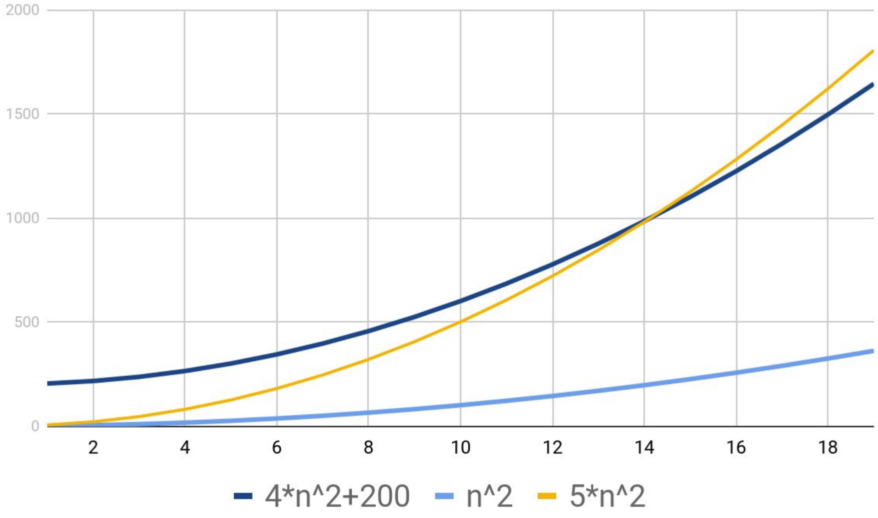
Анализ асимптотики

Пусть есть функции $T(n)$, $f(n)$: $n = 1,2,3,..$

$T(n) = \Theta(f(n)) \Leftrightarrow T(n) = O(f(n))$ и $T(n) = \Omega(f(n))$

Эквивалентное определение: $\exists\, c1,\ c2,\ k > 0:\ \forall\ n \ge k \Rightarrow c1 \cdot f(n) \le T(n) \le c2 \cdot f(n)$

Анализ асимптотики



$$T(n) = 4 * n^2 + 200;$$

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

$$T(n) = \Omega(n^2)$$

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

Анализ асимптотики

Найти Θ оценку для алгоритма - это очень здорово, так мы получаем полное понимание поведения алгоритма на больших данных.

Анализ асимптотики

Найти Θ оценку для алгоритма - это очень здорово, так мы получаем полное понимание поведения алгоритма на больших данных.

Но это сложно, чаще есть (несовпадающие) оценки снизу и сверху - $\Omega(...)$ и $O(...)$

Анализ асимптотики

Найти Θ оценку для алгоритма - это очень здорово, так мы получаем полное понимание поведения алгоритма на больших данных.

Но это сложно, чаще есть (несовпадающие) оценки снизу и сверху - $\Omega(...)$ и $O(...)$

В практических же целях важнее именно оценка сверху, поэтому обычно говорят про $O(...)$

Замечания про асимптотику

1. Все, что обсуждали, верно и для памяти, потребляемой алгоритмом (берем как функцию от размера входных данных, смотрим на асимптотику)

Замечания про асимптотику

1. Все, что обсуждали, верно и для памяти, потребляемой алгоритмом (берем как функцию от размера входных данных, смотрим на асимптотику)

2. Константы и младшие степени на практике могут быть важны! Тогда одной асимптотики не хватает для анализа

"On the basis of the issues discussed here, I propose that members of SIGACT, and editors of computer science and mathematics journals, adopt the O, Ω, Θ notations as defined above, unless a better alternative can be found reasonably soon".

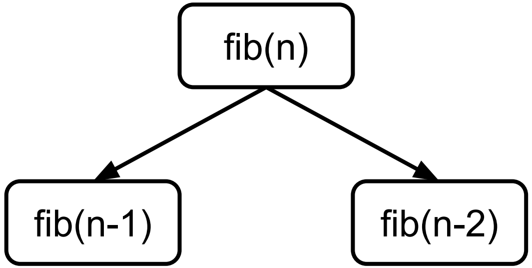
-D.E. Knuth, "Big Omicron and Big Omega and Big Theta", SIGACT News, 1976

Сложность алгоритма?

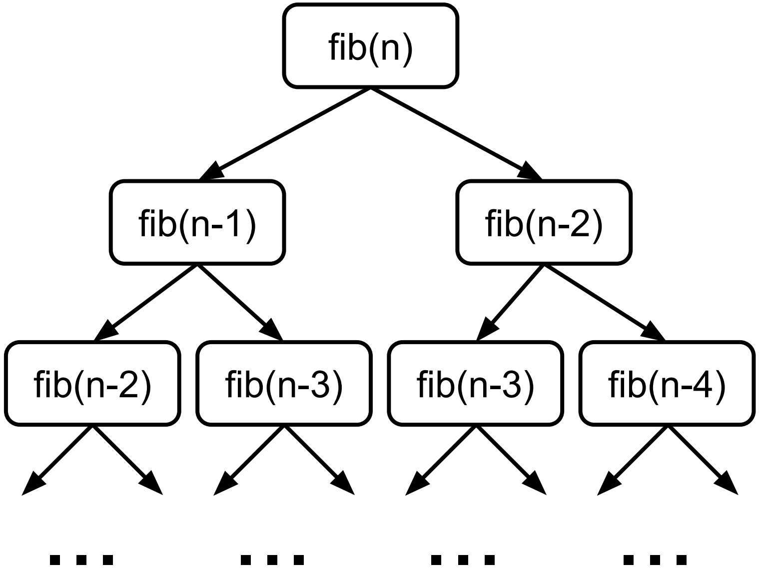
```
def fib(n: int) -> int:
    if n == 0 or n == 1:
        return n
    return fib(n - 1) + fib(n - 2)
```

Сложность вычисление N-ого числа Фибоначчи?

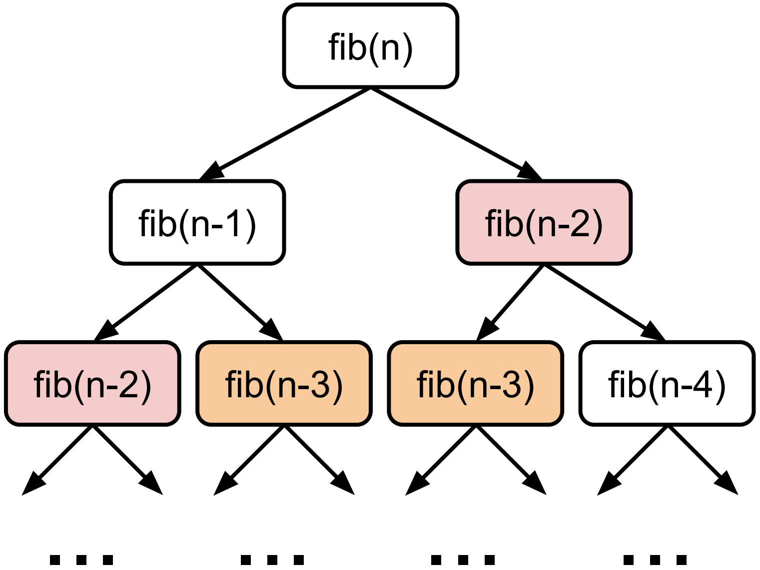
Сложность алгоритма?



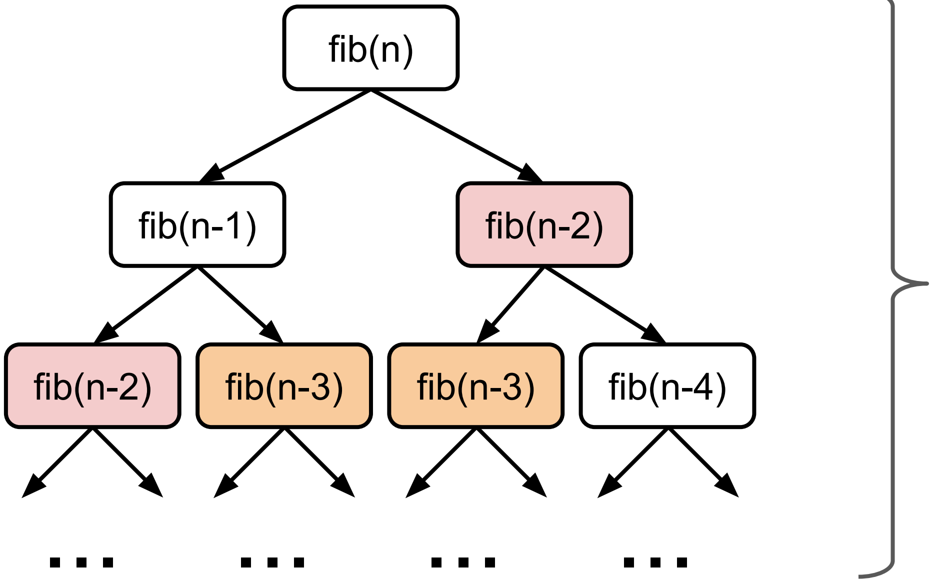
Сложность алгоритма?



Сложность алгоритма?



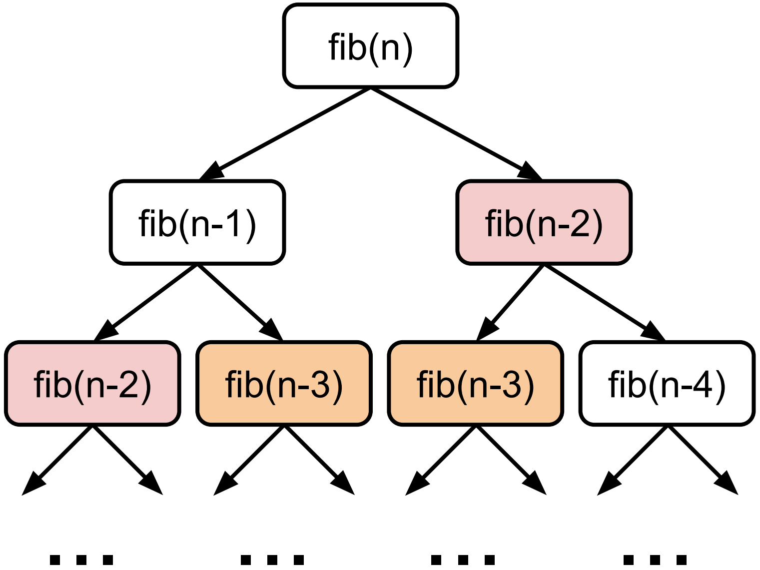
Сложность алгоритма?



n уровней,

на каждом* в два раза больше вызовов

Сложность алгоритма?

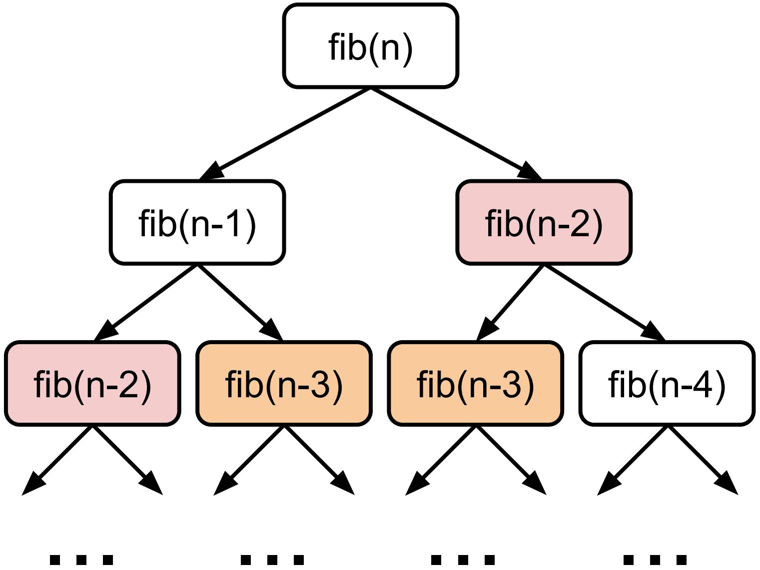


n уровней,

на каждом* в два раза больше вызовов

на последнем уровне $\approx 2^n$

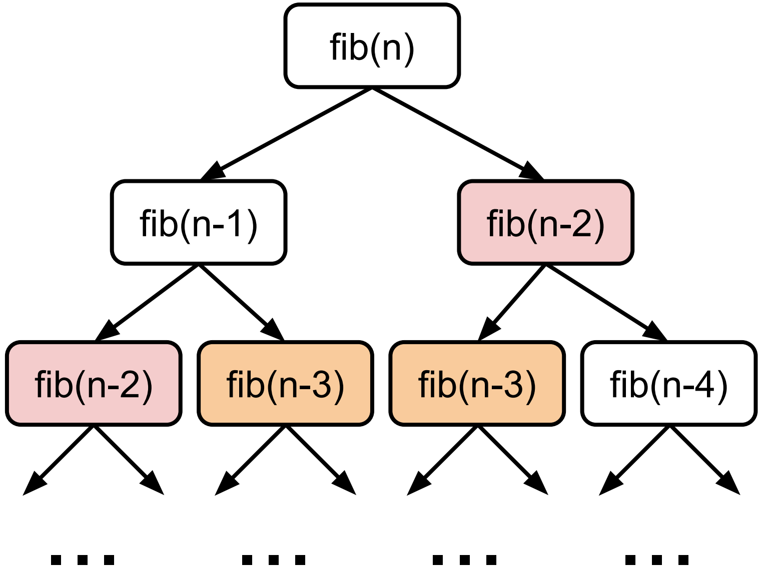
Сложность алгоритма?



$$T(n) \leq C * (1 + 2 + 4 + \ldots + 2^n) \leq C * 2^n$$

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

Сложность алгоритма?



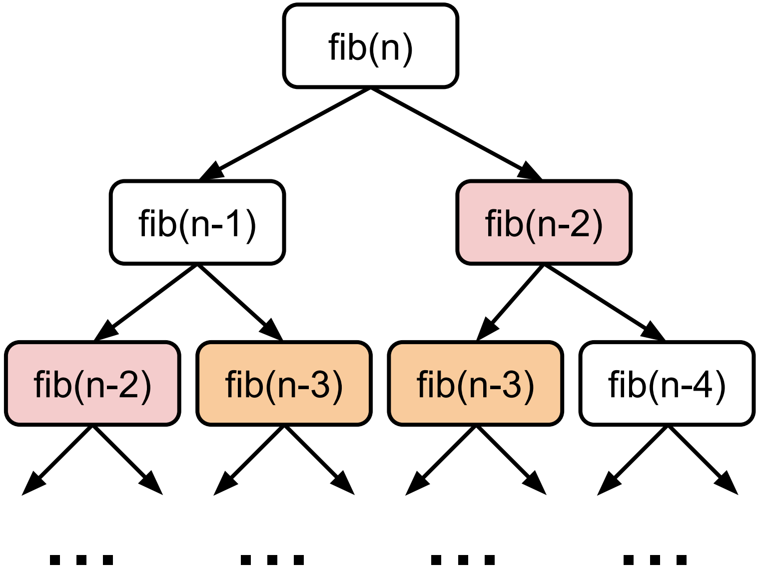
$$T(n) \leq C * (1 + 2 + 4 + \ldots + 2^n) \leq C * 2^n$$

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

На самом деле:

$$T(n) = O(p^n), p = \frac{1+\sqrt{5}}{2}$$

Сложность алгоритма?



$$T(n) \leq C * (1 + 2 + 4 + \ldots + 2^n) \leq C * 2^n$$

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

На самом деле:

$$T(n) = O(p^n), p = \frac{1+\sqrt{5}}{2}$$

Сложность алгоритма?

```python
def fib2(n: int) -> int:
    if n == 0 or n == 1: return n
    f0, f1 = 0, 1

    for i in [1, n):
        res = f0 + f1
        f0 = f1
        f1 = res

   return res
```

Сложность алгоритма?

```
def fib2(n: int) -> int:
    if n == 0 or n == 1: return n
    f0, f1 = 0, 1

    for i in [1, n):
        res = f0 + f1
        f0 = f1
        f1 = res

   return res
```

Сложность: O(N)

Экспоненциальная сложность

А когда такая сложность может быть вполне подходящей?

Экспоненциальная сложность

Задача: написать функцию, которая печатает все подмножества множества {1, 2,... N}.

Экспоненциальная сложность

Задача: написать функцию, которая печатает все подмножества множества {1, 2,... N}.

Сколько всего таких подмножеств?

Экспоненциальная сложность

Задача: написать функцию, которая печатает все подмножества множества {1, 2,... N}.

Сколько всего таких подмножеств?
Как раз 2^N!

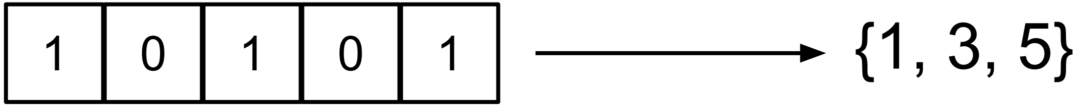
Экспоненциальная сложность

Задача: написать функцию, которая печатает все подмножества множества {1, 2,... N}.

Сколько всего таких подмножеств?
Как раз 2^N!

Как решать?

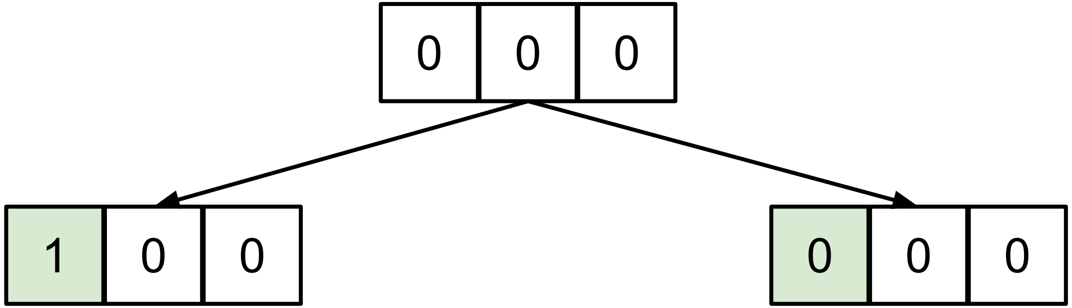
Экспоненциальная сложность



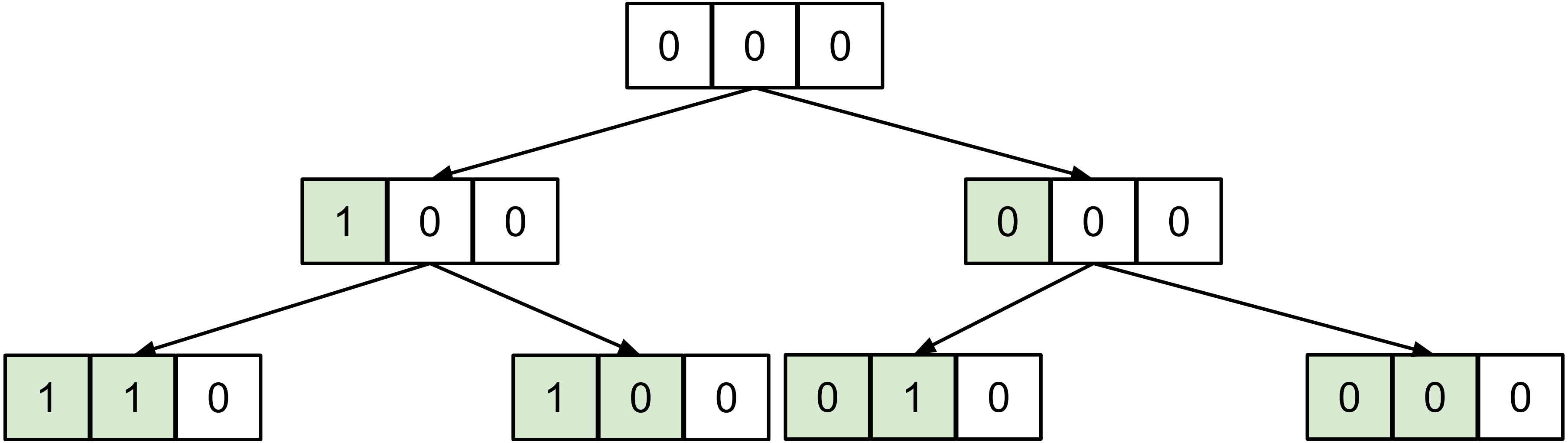
Экспоненциальная сложность

0	0	0

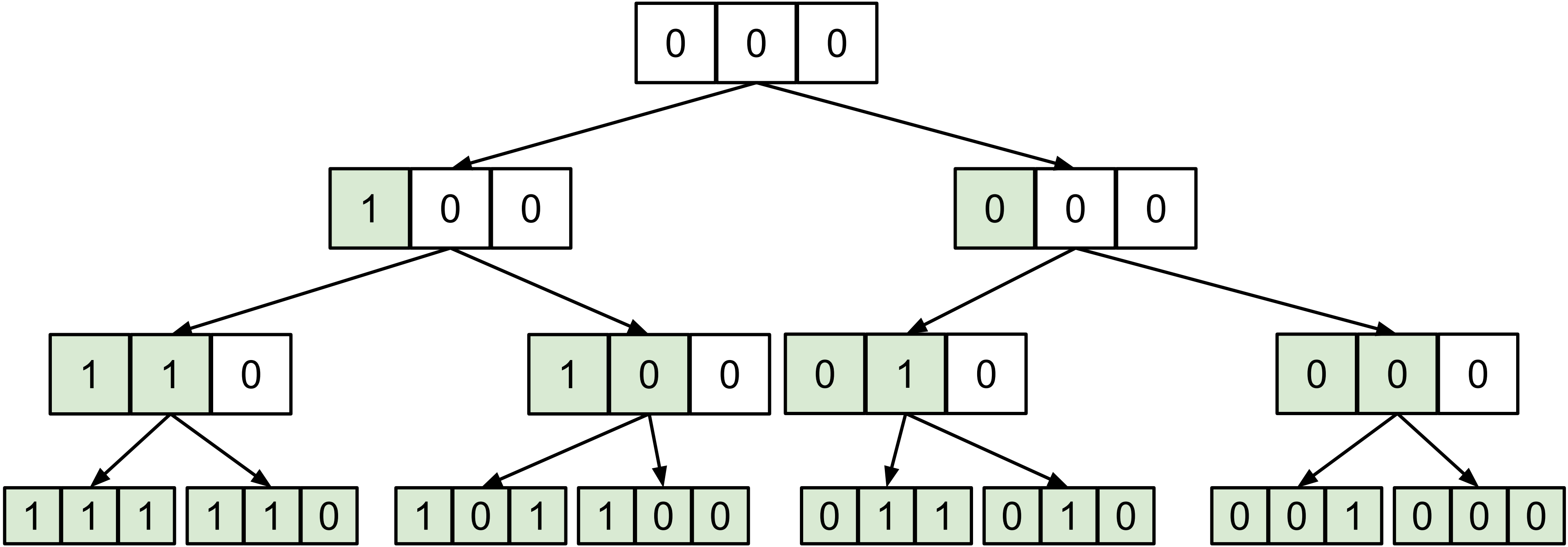
Экспоненциальная сложность



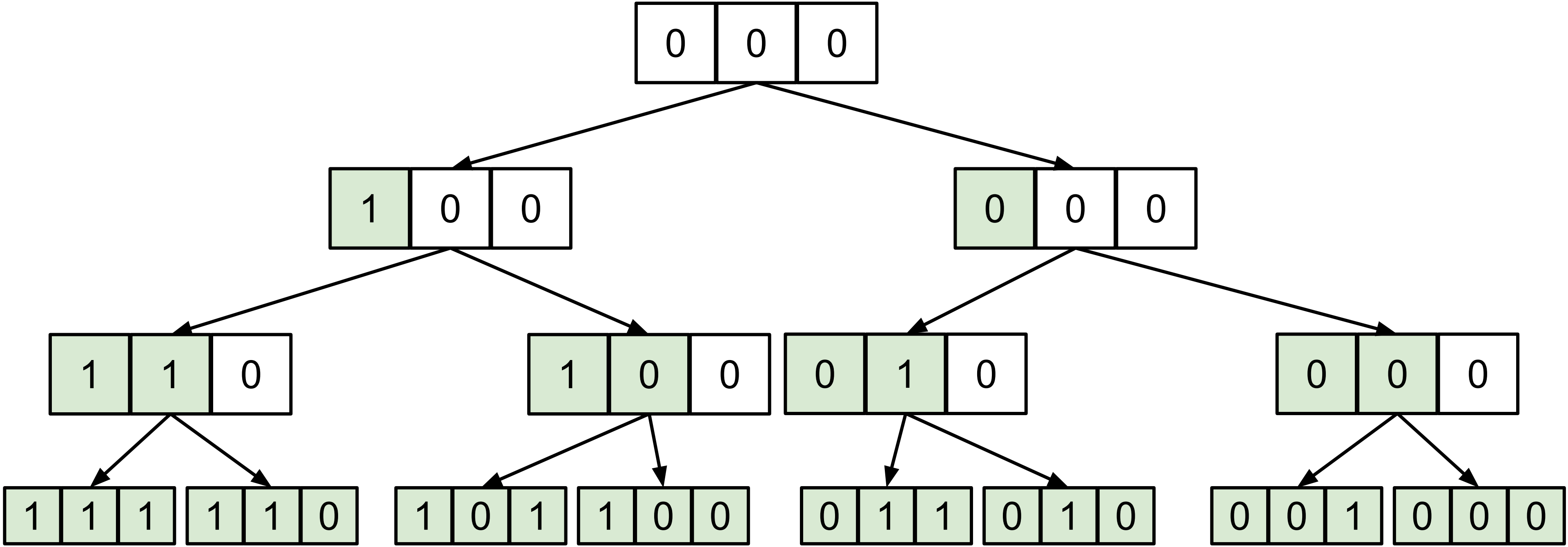
Экспоненциальная сложность



Экспоненциальная сложность



Экспоненциальная сложность



Экспоненциальная сложность

```
def calc_set(array: int[], index: int):
```

Экспоненциальная сложность

```
def calc_set(array: int[], index: int):
    ...
    array[index] = 0
    calc_set(array, index + 1)

    array[index] = 1
    calc_set(array, index + 1)
```

Экспоненциальная сложность

```
def calc_set(array: int[], index: int):

    if index == len(array):
        print_set(array)

    array[index] = 0
    calc_set(array, index + 1)

    array[index] = 1
    calc_set(array, index + 1)
```

Сложность?

Экспоненциальная сложность

```
def calc_set(array: int[], index: int):

    if index == len(array):
        print_set(array)

    array[index] = 0
    calc_set(array, index + 1)

    array[index] = 1
    calc_set(array, index + 1)
```

Сложность?

$$O(2^N)$$

Экспоненциальная сложность

```
def calc_set(array: int[], index: int):

    if index == len(array):
        print_set(array)

    array[index] = 0
    calc_set(array, index + 1)

    array[index] = 1
    calc_set(array, index + 1)
```

Сложность?

$$O(2^N)$$

И это правильно!

Takeaways

- Главные принципы оценки сложности алгоритмов:

Takeaways

- Главные принципы оценки сложности алгоритмов:
  - худший случай
  - функция от размера входных данных,
  - асимптотика

Takeaways

- Главные принципы оценки сложности алгоритмов:
  - худший случай
  - функция от размера входных данных,
  - асимптотика

- $O(...)$, $\Omega(...)$, $\Theta(...)$

Takeaways

- Главные принципы оценки сложности алгоритмов:
  - худший случай
  - функция от размера входных данных,
  - асимптотика
- $O(...)$, $\Omega(...)$, $\Theta(...)$
- Бинарный поиск в теории и на практике
- Экспоненциальная сложность

Мини-задача #3 (1 балл)

Реализовать бинарный поиск без использования рекурсии.
Проверить свое решение на Leetcode:

https://leetcode.com/problems/binary-search/

Мини-задача #4 (2 балла)

Самостоятельно реализовать команду git bisect.
Программа (скрипт) должна принимать на вход:

1. Путь до директории с git репозиторием
2. Два хэша коммитов, задающих диапазон поиска
3. Команду, которую нужно выполнить для проверки ошибок на определенном коммите (если команда возвращает error code отличный от 0, то коммит - плохой)

Протестировать решение на специально созданном репозитории с историей коммитов.