Динамическое программирование
06.12.2022
ИТМО ИС

Фибоначчи

```
int Fibonacci(int n):
  if n <= 1
    return 1
  a = Fibonacci(n - 1)
  b = Fibonacci(n - 2)
  return a + b
```

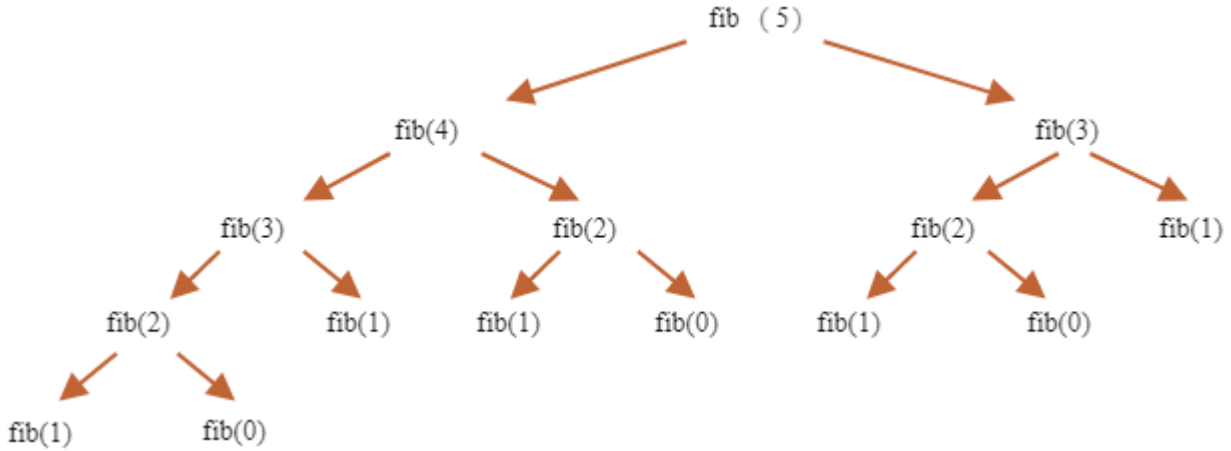
Что плохо в этом решении?

Фибоначчи

```
int Fibonacci(int n):
  if n <= 1
    return 1
  a = Fibonacci(n - 1)
  b = Fibonacci(n - 2)
  return a + b
```

Что плохо в этом решении?

Одно и то же значение вычисляется много раз



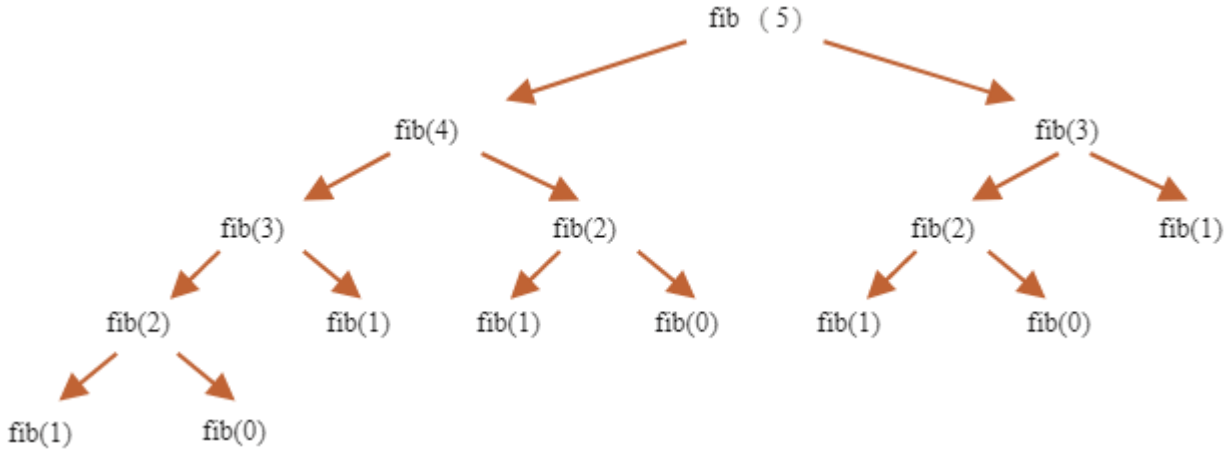
Фибоначчи

```
int Fibonacci(int n):
  if n <= 1
    return 1
  a = Fibonacci(n - 1)
  b = Fibonacci(n - 2)
  return a + b
```

Что плохо в этом решении?

Одно и то же значение вычисляется много раз

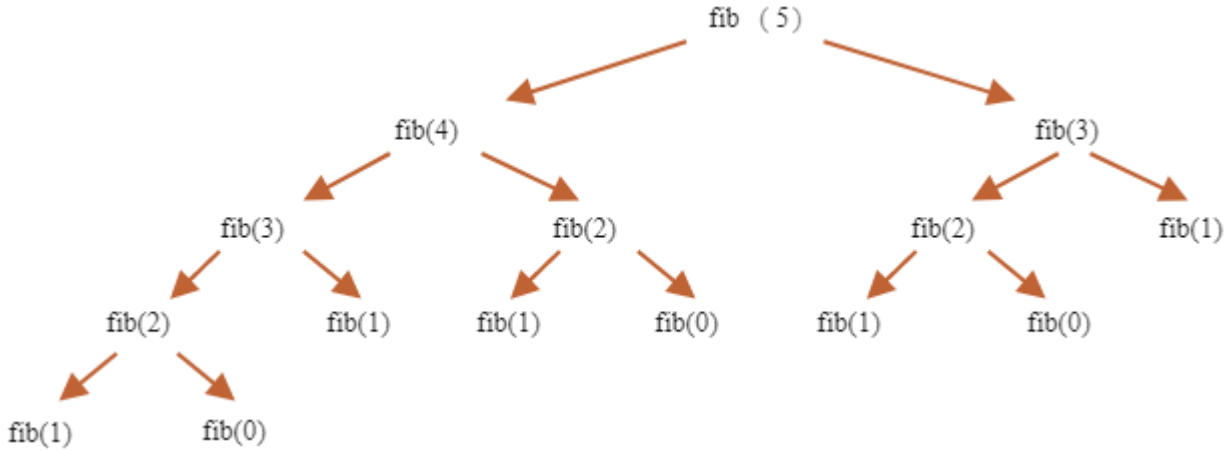
Какие идеи по исправлению?



Фибоначчи

```
int Fibonacci(int n):
  if n <= 1
    return 1
  a = Fibonacci(n - 1)
  b = Fibonacci(n - 2)
  return a + b
```

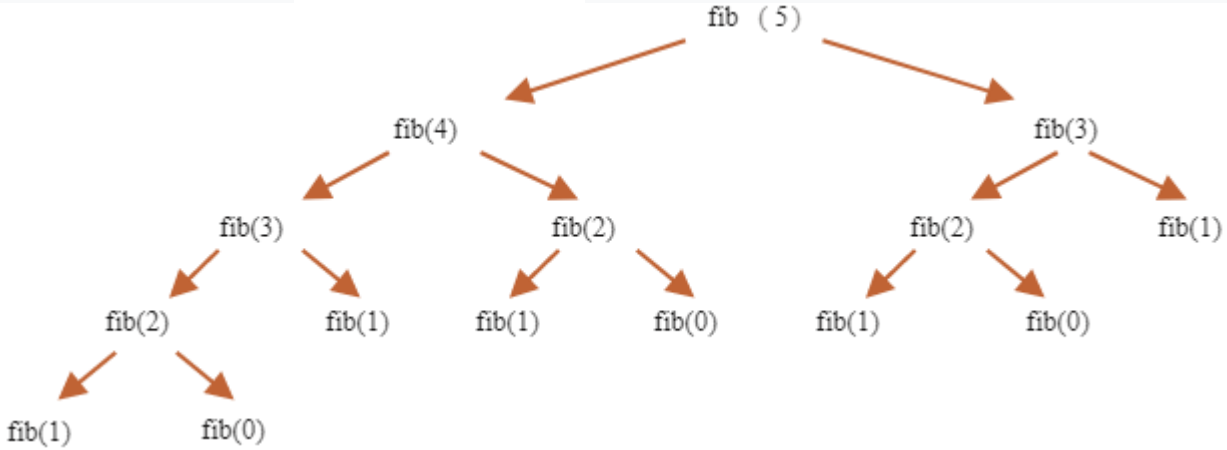
Давайте хранить уже посчитанные значения в массиве



Фибоначчи

```
int Fibonacci(int n):
  if n <= 1
    return 1
  a = Fibonacci(n - 1)
  b = Fibonacci(n - 2)
  return a + b
```

```
int Fibonacci(int n):
  if n <= 1
    return 1
  if fib[n] == -1 // проверка на то, не посчитали ли
    fib[n] = Fibonacci(n - 1) + Fibonacci(n - 2)
  return fib[n]
```



Пример динамики и мемоизации

Приведенный здесь пример показывает:

- мы можем считать маленькие части и приходить через них к ответу: f(n) = f(n-1) + f(n-2)
- мы можем запоминать ответы, чтобы быстрее их находить в дальнейшем

```
int Fibonacci(int n):
  if n <= 1
    return 1
  if fib[n] == -1 // проверка на то, не посчитали ли
    fib[n] = Fibonacci(n - 1) + Fibonacci(n - 2)
  return fib[n]
```

Динамическое программирование

Динамическое программирование — это когда у нас есть задача, которую непонятно как решать, и мы разбиваем ее на меньшие задачи, которые тоже непонятно как решать.

(c) А.Кумок

Динамическое программирование

Динамическое программирование – математическая индукция на языке алгоритмов

Проведем некоторые аналогии:

Мат. Индукция	***Динамика***
База	*Инициализация (заполнение массива динамики базовыми значениями*
Переход	*Вычисление следующего значения, основываясь на предыдущих*

Динамическое программирование

План действий при решении задачи через ДП:

1. Понять, как разделить свою задачу на маленькие подзадачи
2. Выразить свое решение через решение подзадач
3. Выразить динамику своего решения формульно
4. Реализовать полученное решение, заведя массив, в котором хранятся ответы для подзадач
5. В полученном массиве динамики определить ячейку, где хранится итоговый ответ

Задача о рюкзаке

Задача о рюкзаке

N - количество предметов
W - вместимость рюкзака
w_i - вес i-ой вещи
p_i - стоимость i-ой вещи

Хотим собрать рюкзак большей стоимости с весом не более чем W

Иначе, хотим найти

b_i, i=1..N, b_i - 0/1:
sum(b_i * p_i) - max,
sum(b_i * w_i) <= W

Задача о рюкзаке

N - количество предметов, W - вместимость рюкзака, w_i - вес i-ой вещи, p_i - стоимость i-ой вещи
Хотим собрать рюкзак большей стоимости с весом не более чем W
Иначе хотим найти b_i, i=1..N, b_i - 0/1: sum(b_i * p_i) - max, sum(b_i * w_i) <= W

A(k,s) - максимальная стоимость предметов, которые можно уложить в рюкзак вместимости s, если можно использовать только первые k предметов

A(k,0)=0, A(0,s)=0

Задача о рюкзаке

1. k-ый предмет не попал в рюкзак => A(k, s) = A(k-1,s)
2. попал => A(k,s) = A(k-1, s-w_k) + p_k

Итого:

$$A(k,s) = \begin{cases} A(k-1,s), & b_k = 0 \\ A(k-1,s-w_k) + p_k, & b_k = 1 \end{cases}$$

То есть: $A(k,s) = \max(A(k-1,s), A(k-1,s-w_k) + p_k)$

Задача о рюкзаке

$W = 13, N = 5$

$w_1 = 3, p_1 = 1$

$w_2 = 4, p_2 = 6$

$w_3 = 5, p_3 = 4$

$w_4 = 8, p_4 = 7$

$w_5 = 9, p_5 = 6$

$$A(k,s) = \begin{cases} A(k-1,s), & b_k = 0 \\ A(k-1,s-w_k) + p_k, & b_k = 1 \end{cases}$$

То есть: $A(k,s) = \max(A(k-1,s), A(k-1,s-w_k) + p_k)$

можем брать первые ноль предметов

	1	2	3	4	5	6	7	8	9	10	11	12	13
k = 0	0	0	0	0	0	0	0	0	0	0	0	0	0
k = 1	0	0	1	1	1	1	1	1	1	1	1	1	1
k = 2	0	0	1	6	6	6	7	7	7	7	7	7	7
k = 3	0	0	1	6	6	6	7	7	10	10	10	11	11
k = 4	0	0	1	6	6	6	7	7	10	10	10	13	13
k = 5	0	0	1	6	6	6	7	7	10	10	10	13	13

Задача о рюкзаке

$W = 13, N = 5$

$w_1 = 3, p_1 = 1$

$w_2 = 4, p_2 = 6$

$w_3 = 5, p_3 = 4$

$w_4 = 8, p_4 = 7$

$w_5 = 9, p_5 = 6$

переход на вторую строку позволяет взять первый предмет

$$A(k,s) = \begin{cases} A(k-1,s), & b_k = 0 \\ A(k-1,s-w_k)+p_k, & b_k = 1 \end{cases}$$

То есть: $A(k,s) = \max(A(k-1,s), A(k-1,s-w_k)+p_k)$

начиная с вместимости 3 мы можем этот предмет взять

	1	2	3	4	5	6	7	8	9	10	11	12	13
k = 0	0	0	0	0	0	0	0	0	0	0	0	0	0
k = 1	0	0	1	1	1	1	1	1	1	1	1	1	1
k = 2	0	0	1	6	6	6	7	7	7	7	7	7	7
k = 3	0	0	1	6	6	6	7	7	10	10	10	11	11
k = 4	0	0	1	6	6	6	7	7	10	10	10	13	13
k = 5	0	0	1	6	6	6	7	7	10	10	10	13	13

Задача о рюкзаке

$W = 13, N = 5$

$w_1 = 3, p_1 = 1$

$w_2 = 4, p_2 = 6$

$w_3 = 5, p_3 = 4$

$w_4 = 8, p_4 = 7$

$w_5 = 9, p_5 = 6$

$$A(k,s) = \begin{cases} A(k-1,s), & b_k = 0 \\ A(k-1,s-w_k) + p_k, & b_k = 1 \end{cases}$$

То есть: $A(k,s) = \max(A(k-1,s), A(k-1,s-w_k) + p_k)$

какие предметы можем брать теперь?

	1	2	3	4	5	6	7	8	9	10	11	12	13
k = 0	0	0	0	0	0	0	0	0	0	0	0	0	0
k = 1	0	0	1	1	1	1	1	1	1	1	1	1	1
k = 2	0	0	1	6	6	6	7	7	7	7	7	7	7
k = 3	0	0	1	6	6	6	7	7	10	10	10	11	11
k = 4	0	0	1	6	6	6	7	7	10	10	10	13	13
k = 5	0	0	1	6	6	6	7	7	10	10	10	13	13

Задача о рюкзаке

$W = 13, N = 5$

$w_1 = 3, p_1 = 1$ ← этот

$w_2 = 4, p_2 = 6$ ← и этот!

$w_3 = 5, p_3 = 4$

$w_4 = 8, p_4 = 7$

$w_5 = 9, p_5 = 6$

$$A(k,s) = \begin{cases} A(k-1,s), & b_k = 0 \\ A(k-1,s-w_k) + p_k, & b_k = 1 \end{cases}$$

То есть: $A(k,s) = \max(A(k-1,s), A(k-1,s-w_k) + p_k)$

	1	2	3	4	5	6	7	8	9	10	11	12	13
k = 0	0	0	0	0	0	0	0	0	0	0	0	0	0
k = 1	0	0	1	1	1	1	1	1	1	1	1	1	1
k = 2	0	0	1	6	6	6	7	7	7	7	7	7	7
k = 3	0	0	1	6	6	6	7	7	10	10	10	11	11
k = 4	0	0	1	6	6	6	7	7	10	10	10	13	13
k = 5	0	0	1	6	6	6	7	7	10	10	10	13	13

Задача о рюкзаке

$W = 13, N = 5$

$w_1 = 3, p_1 = 1$ ← этот

$w_2 = 4, p_2 = 6$ ← и этот!

$w_3 = 5, p_3 = 4$

$w_4 = 8, p_4 = 7$

$w_5 = 9, p_5 = 6$

$$A(k,s) = \begin{cases} A(k-1,s), & b_k = 0 \\ A(k-1,s-w_k) + p_k, & b_k = 1 \end{cases}$$

То есть: $A(k,s) = \max(A(k-1,s), A(k-1,s-w_k) + p_k)$

выбираем максимальное значение из двух по формуле

	1	2	3	4	5	6	7	8	9	10	11	12	13
k = 0	0	0	0	0	0	0	0	0	0	0	0	0	0
k = 1	0	0	1	1	1	1	1	1	1	1	1	1	1
k = 2	0	0	1	6	6	6	7	7	7	7	7	7	7
k = 3	0	0	1	6	6	6	7	7	10	10	10	11	11
k = 4	0	0	1	6	6	6	7	7	10	10	10	13	13
k = 5	0	0	1	6	6	6	7	7	10	10	10	13	13

+6

Задача о рюкзаке

$W = 13, N = 5$

$w_1 = 3, p_1 = 1$ ← этот

$w_2 = 4, p_2 = 6$ ← и этот!

$w_3 = 5, p_3 = 4$

$w_4 = 8, p_4 = 7$

$w_5 = 9, p_5 = 6$

$$A(k,s) = \begin{cases} A(k-1,s), & b_k = 0 \\ A(k-1,s-w_k) + p_k, & b_k = 1 \end{cases}$$

То есть: $A(k,s) = \max(A(k-1,s), A(k-1,s-w_k) + p_k)$

выбираем максимальное значение из двух по формуле

	1	2	3	4	5	6	7	8	9	10	11	12	13
k = 0	0	0	0	0	0	0	0	0	0	0	0	0	0
k = 1	0	0	1	1	1	1	1	1	1	1	1	1	1
k = 2	0	0	1	6	6	6	7	7	7	7	7	7	7
k = 3	0	0	1	6	6	6	7	7	10	10	10	11	11
k = 4	0	0	1	6	6	6	7	7	10	10	10	13	13
k = 5	0	0	1	6	6	6	7	7	10	10	10	13	13

+6

так доходим до конца таблицы

Задача о рюкзаке

$W = 13, N = 5$

$w_1 = 3, p_1 = 1$

$w_2 = 4, p_2 = 6$

$w_3 = 5, p_3 = 4$

$w_4 = 8, p_4 = 7$

$w_5 = 9, p_5 = 6$

$$A(k,s) = \begin{cases} A(k-1,s), & b_k = 0 \\ A(k-1,s-w_k) + p_k, & b_k = 1 \end{cases}$$

То есть: $A(k,s) = \max(A(k-1,s), A(k-1,s-w_k) + p_k)$

	1	2	3	4	5	6	7	8	9	10	11	12	13
k = 0	0	0	0	0	0	0	0	0	0	0	0	0	0
k = 1	0	0	1	1	1	1	1	1	1	1	1	1	1
k = 2	0	0	1	6	6	6	7	7	7	7	7	7	7
k = 3	0	0	1	6	6	6	7	7	10	10	10	11	11
k = 4	0	0	1	6	6	6	7	7	10	10	10	13	13
k = 5	0	0	1	6	6	6	7	7	10	10	10	13	13

+6

Задача о рюкзаке

Код, соответствующий динамике

$$A(k,s) = \begin{cases} A(k-1,s), & b_k = 0 \\ A(k-1,s-w_k)+p_k, & b_k = 1 \end{cases}$$

То есть: $A(k,s) = \max(A(k-1,s), A(k-1,s-w_k)+p_k)$

```
for i = 0 to w
  A[0][i] = 0
for i = 0 to n
  A[i][0] = 0                                          //Первые элементы приравниваем к 0
for k = 1 to n
  for s = 1 to w                                       //Перебираем для каждого k все вместимости
    if s >= w[k]                                       //Если текущий предмет вмещается в рюкзак
      A[k][s] = max(A[k - 1][s], A[k - 1][s - w[k]] + p[k]) //Выбираем класть его или нет
    else
      A[k][s] = A[k - 1][s]                            //Иначе, не кладем
```

Задача о рюкзаке

$$A(k,s) = \begin{cases} A(k-1,s), & b_k = 0 \\ A(k-1,s-w_k) + p_k, & b_k = 1 \end{cases}$$

То есть: $A(k,s) = \max(A(k-1,s), A(k-1,s-w_k) + p_k)$

```
for i = 0 to w
  A[0][i] = 0
for i = 0 to n
  A[i][0] = 0                                          //Первые элементы приравниваем к 0
for k = 1 to n
  for s = 1 to w                                       //Перебираем для каждого k все вместимости
    if s >= w[k]                                       //Если текущий предмет вмещается в рюкзак
      A[k][s] = max(A[k - 1][s], A[k - 1][s - w[k]] + p[k]) //Выбираем класть его или нет
    else
      A[k][s] = A[k - 1][s]                            //Иначе, не кладем
```

Как найти элементы, которые будут включены в ответ?

Задача о рюкзаке

```
function findAns(int k, int s)
  if A[k][s] == 0
    return
  if A[k - 1][s] == A[k][s]
    findAns(k - 1, s)
  else
    findAns(k - 1, s - w[k])
    ans.push(k)
```

$$A(k,s) = \begin{cases} A(k-1,s), & b_k = 0 \\ A(k-1,s-w_k) + p_k, & b_k = 1 \end{cases}$$

То есть: $A(k,s) = \max(A(k-1,s), A(k-1,s-w_k) + p_k)$

Рекурсия

Задача о рюкзаке. Восстановление

$W = 13, N = 5$

$w_1 = 3, p_1 = 1$

$w_2 = 4, p_2 = 6$

$w_3 = 5, p_3 = 4$

$w_4 = 8, p_4 = 7$

$w_5 = 9, p_5 = 6$

```
function findAns(int k, int s)
  if A[k][s] == 0
    return
  if A[k - 1][s] == A[k][s]
    findAns(k - 1, s)
  else
    findAns(k - 1, s - w[k])
    ans.push(k)
```

$$A(k,s) = \begin{cases} A(k-1,s), & b_k = 0 \\ A(k-1,s-w_k) + p_k, & b_k = 1 \end{cases}$$

То есть: $A(k,s) = \max(A(k-1,s), A(k-1,s-w_k) + p_k)$

	1	2	3	4	5	6	7	8	9	10	11	12	13
k = 0	0	0	0	0	0	0	0	0	0	0	0	0	0
k = 1	0	0	1	1	1	1	1	1	1	1	1	1	1
k = 2	0	0	1	6	6	6	7	7	7	7	7	7	7
k = 3	0	0	1	6	6	6	7	7	10	10	10	11	11
k = 4	0	0	1	6	6	6	7	7	10	10	10	13	13
k = 5	0	0	1	6	6	6	7	7	10	10	10	13	13

13

5-ый предмет не входит в ответ

6 + 6 = 12

Задача о рюкзаке. Восстановление

$W = 13, N = 5$

$w_1 = 3, p_1 = 1$

$w_2 = 4, p_2 = 6$

$w_3 = 5, p_3 = 4$

$w_4 = 8, p_4 = 7$

$w_5 = 9, p_5 = 6$

```
function findAns(int k, int s)
  if A[k][s] == 0
    return
  if A[k - 1][s] == A[k][s]
    findAns(k - 1, s)
  else
    findAns(k - 1, s - w[k])
    ans.push(k)
```

$$A(k,s) = \begin{cases} A(k-1,s), & b_k = 0 \\ A(k-1,s-w_k) + p_k, & b_k = 1 \end{cases}$$

То есть: $A(k,s) = \max(A(k-1,s), A(k-1,s-w_k) + p_k)$

	1	2	3	4	5	6	7	8	9	10	11	12	13
k = 0	0	0	0	0	0	0	0	0	0	0	0	0	0
k = 1	0	0	1	1	1	1	1	1	1	1	1	1	1
k = 2	0	0	1	6	6	6	7	7	7	7	7	7	7
k = 3	0	0	1	6	6	6	7	7	10	10	10	11	11
k = 4	0	0	1	6	6	6	7	7	10	10	10	13	13
k = 5	0	0	1	6	6	6	7	7	10	10	10	13	13

4-ый предмет входит в ответ! 6 + 7 = 13

Задача о рюкзаке. Восстановление

$W = 13, N = 5$

$w_1 = 3, p_1 = 1$

$w_2 = 4, p_2 = 6$

$w_3 = 5, p_3 = 4$

$w_4 = 8, p_4 = 7$

$w_5 = 9, p_5 = 6$

```
function findAns(int k, int s)
  if A[k][s] == 0
    return
  if A[k - 1][s] == A[k][s]
    findAns(k - 1, s)
  else
    findAns(k - 1, s - w[k])
    ans.push(k)
```

$$A(k,s) = \begin{cases} A(k-1,s), & b_k = 0 \\ A(k-1,s-w_k) + p_k, & b_k = 1 \end{cases}$$

То есть: $A(k,s) = \max(A(k-1,s), A(k-1,s-w_k) + p_k)$

	1	2	3	4	5	6	7	8	9	10	11	12	13
k = 0	0	0	0	0	0	0	0	0	0	0	0	0	0
k = 1	0	0	1	1	1	1	1	1	1	1	1	1	1
k = 2	0	0	1	6	6	6	7	7	7	7	7	7	7
k = 3	0	0	1	6	6	6	7	7	10	10	10	11	11
k = 4	0	0	1	6	6	6	7	7	10	10	10	13	13
k = 5	0	0	1	6	6	6	7	7	10	10	10	13	13

0+4=4

3-ий предмет не берем

Задача о рюкзаке. Восстановление

$W = 13, N = 5$

$w_1 = 3, p_1 = 1$

$w_2 = 4, p_2 = 6$

$w_3 = 5, p_3 = 4$

$w_4 = 8, p_4 = 7$

$w_5 = 9, p_5 = 6$

```
function findAns(int k, int s)
  if A[k][s] == 0
    return
  if A[k - 1][s] == A[k][s]
    findAns(k - 1, s)
  else
    findAns(k - 1, s - w[k])
    ans.push(k)
```

$$A(k,s) = \begin{cases} A(k-1,s), & b_k = 0 \\ A(k-1,s-w_k) + p_k, & b_k = 1 \end{cases}$$

То есть: $A(k,s) = \max(A(k-1,s), A(k-1,s-w_k) + p_k)$

	1	2	3	4	5	6	7	8	9	10	11	12	13
k = 0	0	0	0	0	0	0	0	0	0	0	0	0	0
k = 1	0	0	1	1	1	1	1	1	1	1	1	1	1
k = 2	0	0	1	6	6	6	7	7	7	7	7	7	7
k = 3	0	0	1	6	6	6	7	7	10	10	10	11	11
k = 4	0	0	1	6	6	6	7	7	10	10	10	13	13
k = 5	0	0	1	6	6	6	7	7	10	10	10	13	13

0+6

Второй предмет берем! Дальше ничего не берем.

Задача о рюкзаке. Асимптотика

$W = 13, N = 5$

$w_1 = 3, p_1 = 1$

$w_2 = 4, p_2 = 6$

$w_3 = 5, p_3 = 4$

$w_4 = 8, p_4 = 7$

$w_5 = 9, p_5 = 6$

По времени: ?

$$A(k,s) = \begin{cases} A(k-1,s), & b_k = 0 \\ A(k-1,s-w_k) + p_k, & b_k = 1 \end{cases}$$

То есть: $A(k,s) = \max(A(k-1,s), A(k-1,s-w_k) + p_k)$

	1	2	3	4	5	6	7	8	9	10	11	12	13
k = 0	0	0	0	0	0	0	0	0	0	0	0	0	0
k = 1	0	0	1	1	1	1	1	1	1	1	1	1	1
k = 2	0	0	1	6	6	6	7	7	7	7	7	7	7
k = 3	0	0	1	6	6	6	7	7	10	10	10	11	11
k = 4	0	0	1	6	6	6	7	7	10	10	10	13	13
k = 5	0	0	1	6	6	6	7	7	10	10	10	13	13

Задача о рюкзаке. Асимптотика

$W = 13, N = 5$

$w_1 = 3, p_1 = 1$

$w_2 = 4, p_2 = 6$

$w_3 = 5, p_3 = 4$

$w_4 = 8, p_4 = 7$

$w_5 = 9, p_5 = 6$

По времени: O(WN)
По памяти: ?

$$A(k,s) = \begin{cases} A(k-1,s), & b_k = 0 \\ A(k-1,s-w_k) + p_k, & b_k = 1 \end{cases}$$

То есть: $A(k,s) = \max(A(k-1,s), A(k-1,s-w_k) + p_k)$

	1	2	3	4	5	6	7	8	9	10	11	12	13
k = 0	0	0	0	0	0	0	0	0	0	0	0	0	0
k = 1	0	0	1	1	1	1	1	1	1	1	1	1	1
k = 2	0	0	1	6	6	6	7	7	7	7	7	7	7
k = 3	0	0	1	6	6	6	7	7	10	10	10	11	11
k = 4	0	0	1	6	6	6	7	7	10	10	10	13	13
k = 5	0	0	1	6	6	6	7	7	10	10	10	13	13

Задача о рюкзаке. Асимптотика

$W = 13, N = 5$

$w_1 = 3, p_1 = 1$

$w_2 = 4, p_2 = 6$

$w_3 = 5, p_3 = 4$

$w_4 = 8, p_4 = 7$

$w_5 = 9, p_5 = 6$

По времени: O(WN)
По памяти: O(WN)

$$A(k,s) = \begin{cases} A(k-1,s), & b_k = 0 \\ A(k-1,s-w_k) + p_k, & b_k = 1 \end{cases}$$

То есть: $A(k,s) = \max(A(k-1,s), A(k-1,s-w_k) + p_k)$

	1	2	3	4	5	6	7	8	9	10	11	12	13
k = 0	0	0	0	0	0	0	0	0	0	0	0	0	0
k = 1	0	0	1	1	1	1	1	1	1	1	1	1	1
k = 2	0	0	1	6	6	6	7	7	7	7	7	7	7
k = 3	0	0	1	6	6	6	7	7	10	10	10	11	11
k = 4	0	0	1	6	6	6	7	7	10	10	10	13	13
k = 5	0	0	1	6	6	6	7	7	10	10	10	13	13

Наибольшая возрастающая подпоследовательность

5 4 **1 2** 5 **3** 6 7 10 9 3 **4 5** 2 1 5 **7**

Задача:

Дан массив из n чисел: $a[0..n-1]$. Требуется найти в этой последовательности строго возрастающую подпоследовательность наибольшей длины.

Определение:

Наибольшая возрастающая подпоследовательность (НВП) (англ. *Longest increasing subsequence, LIS*) строки x длины n — это последовательность $x[i_1] < x[i_2] < \cdots < x[i_k]$ символов строки x таких, что $i_1 < i_2 < \cdots < i_k, 1 \leqslant i_j \leqslant n$, причем k — наибольшее из возможных.

Наибольшая возрастающая подпоследовательность

5	4	**1**	**2**	5	**3**	**6**	**7**

d_i – что будем хранить в массиве динамики?

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**

Наибольшая возрастающая подпоследовательность

5	4	**1**	**2**	5	**3**	**6**	**7**

d_i – массив длин НВП, заканчивающихся в позиции i

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
1	1	1	2	3	3	4	5

Наибольшая возрастающая подпоследовательность

d_i – массив длин НВП, заканчивающихся в позиции i

исходный массив

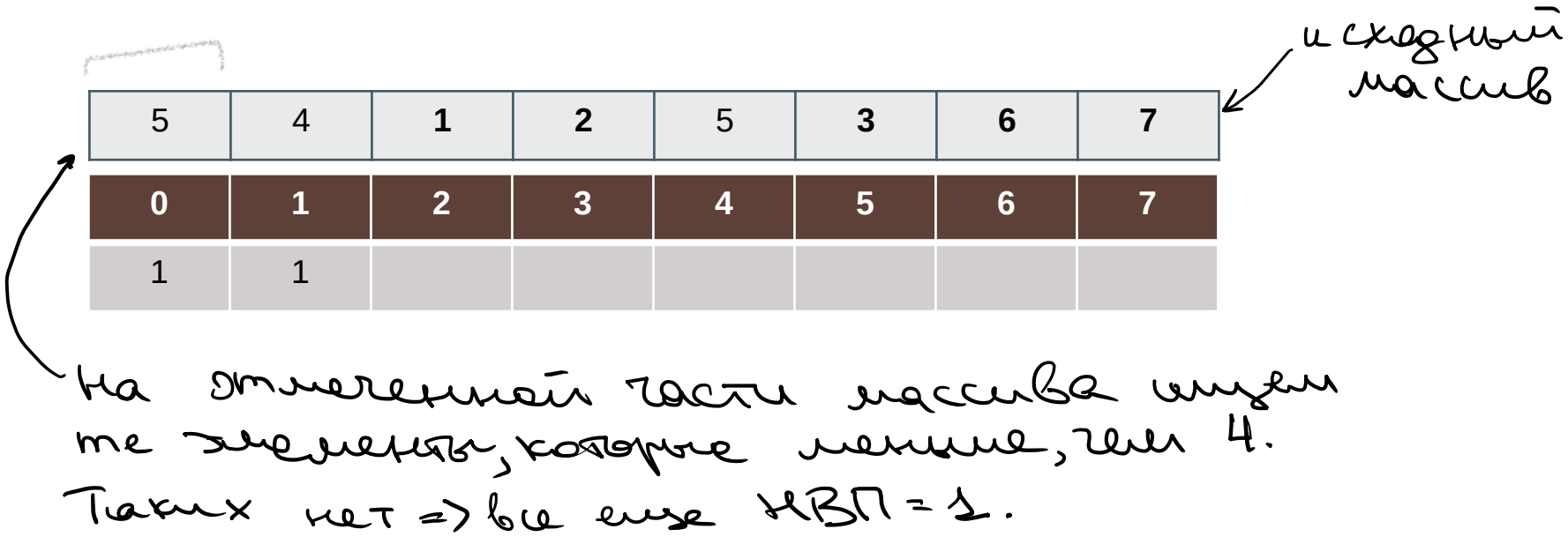
5	4	1	2	5	3	6	7

0	1	2	3	4	5	6	7
1							

Инициализируем динамику тем, что НВП на массиве размером 1 равна 1.

Наибольшая возрастающая подпоследовательность

d_i – массив длин НВП, заканчивающихся в позиции i



Наибольшая возрастающая подпоследовательность

d_i – массив длин НВП, заканчивающихся в позиции i

исходный массив

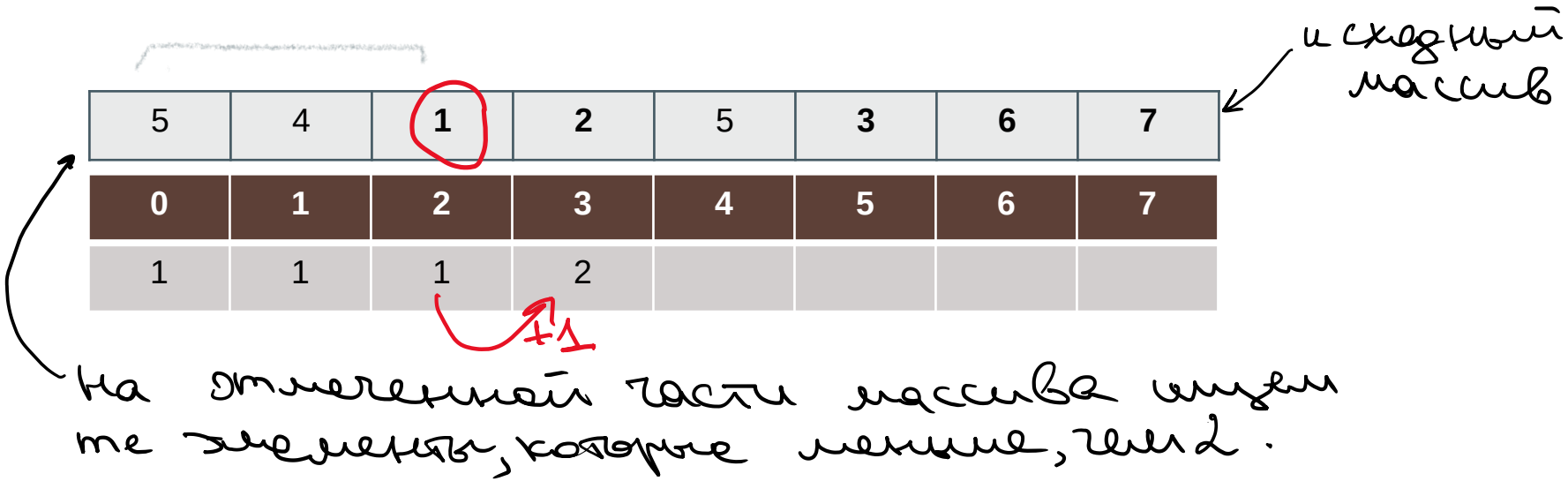
5	4	1	2	5	3	6	7

0	1	2	3	4	5	6	7
1	1	1					

На отмеченной части массива ищем те элементы, которые меньше, чем 1. Таких нет => во всех НВП = 1.

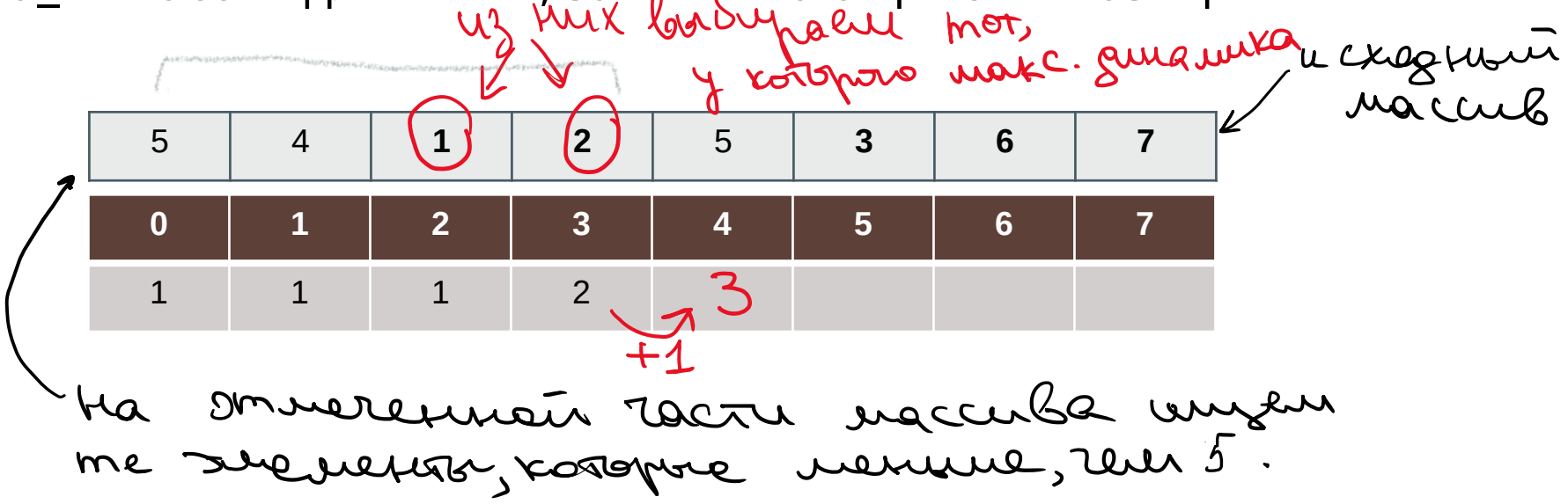
Наибольшая возрастающая подпоследовательность

d_i – массив длин НВП, заканчивающихся в позиции i



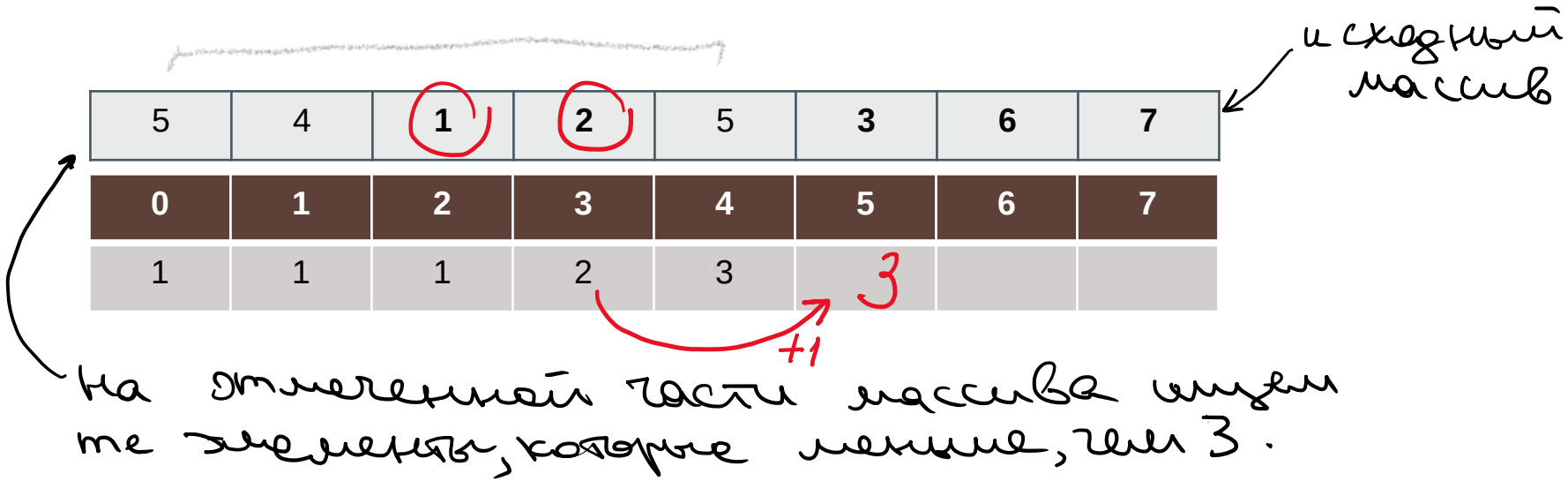
Наибольшая возрастающая подпоследовательность

d_i – массив длин НВП, заканчивающихся в позиции i



Наибольшая возрастающая подпоследовательность

d_i – массив длин НВП, заканчивающихся в позиции i

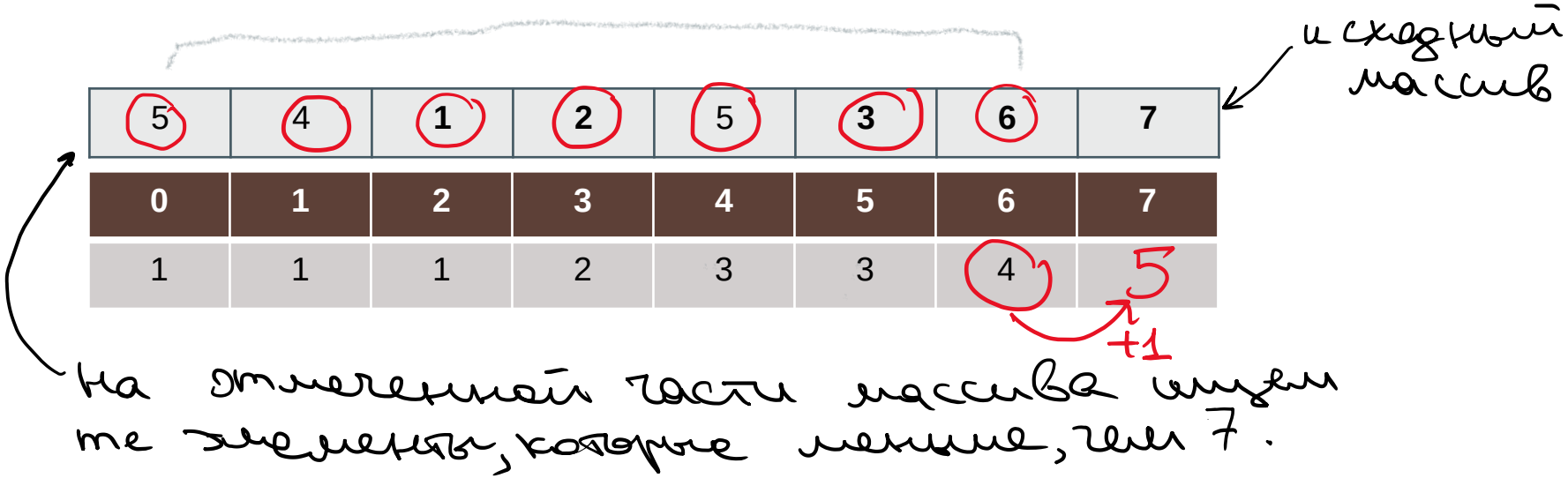


Наибольшая возрастающая подпоследовательность

d_i – массив длин НВП, заканчивающихся в позиции i

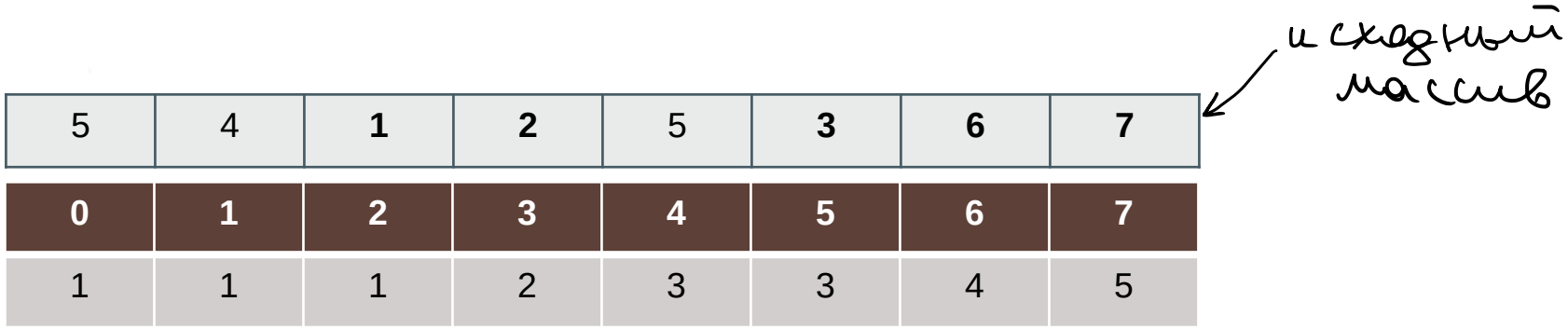
Наибольшая возрастающая подпоследовательность

d_i – массив длин НВП, заканчивающихся в позиции i



Наибольшая возрастающая подпоследовательность

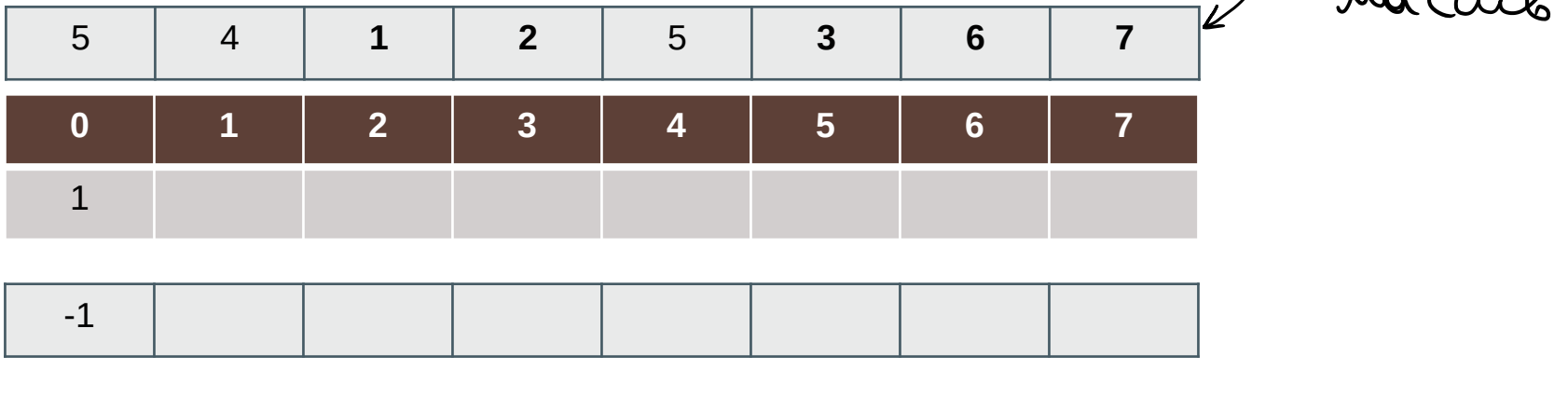
Как восстановить ответ?



Наибольшая возрастающая подпоследовательность

Как восстановить ответ?

Хранить массив prev, где каждый элемент – индекс, по которому мы пришли в текущий индекс



Наибольшая возрастающая подпоследовательность

Как восстановить ответ?

Хранить массив prev, где каждый элемент – индекс, по которому мы пришли в текущий индекс

и входной массив

5	4	1	2	5	3	6	7

0	1	2	3	4	5	6	7
1	1						

prev

-1	-1						

Наибольшая возрастающая подпоследовательность

Как восстановить ответ?

Хранить массив prev, где каждый элемент – индекс, по которому мы пришли в текущий индекс

и входной массив

5	4	**1**	**2**	5	**3**	**6**	**7**

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
1	1	1					

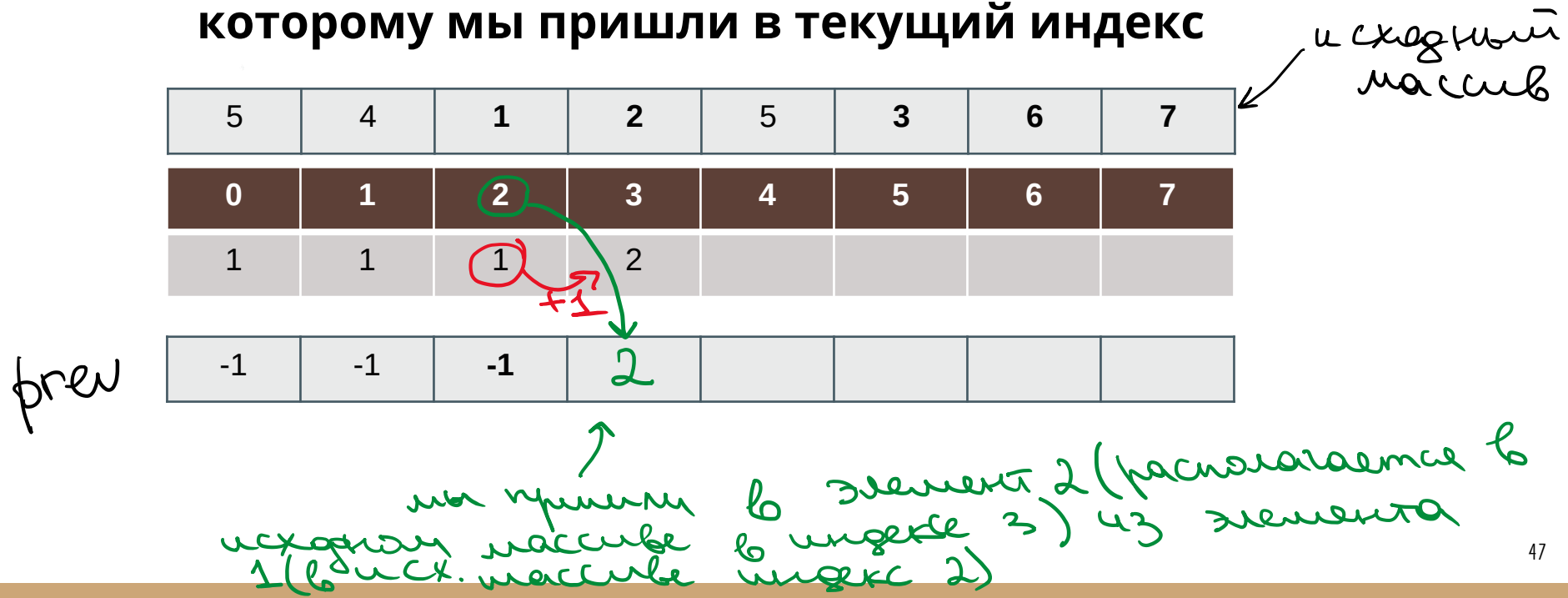
prev

-1	-1	**-1**					

Наибольшая возрастающая подпоследовательность

Как восстановить ответ?

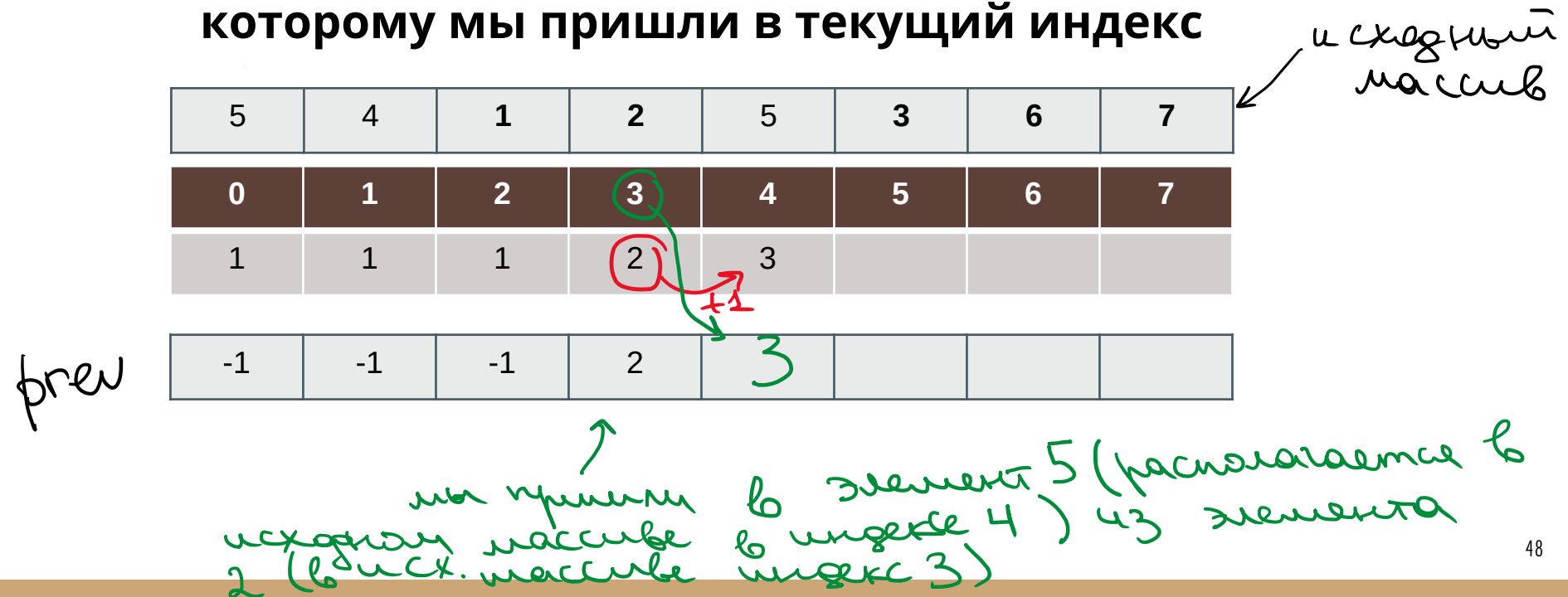
Хранить массив prev, где каждый элемент – индекс, по которому мы пришли в текущий индекс



Наибольшая возрастающая подпоследовательность

Как восстановить ответ?

Хранить массив prev, где каждый элемент – индекс, по которому мы пришли в текущий индекс



Наибольшая возрастающая подпоследовательность

Как восстановить ответ?

Хранить массив prev, где каждый элемент – индекс, по которому мы пришли в текущий индекс

исходный массив

5	4	1	2	5	3	6	7

0	1	2	3	4	5	6	7
1	1	1	2	3	3		

+1

prev

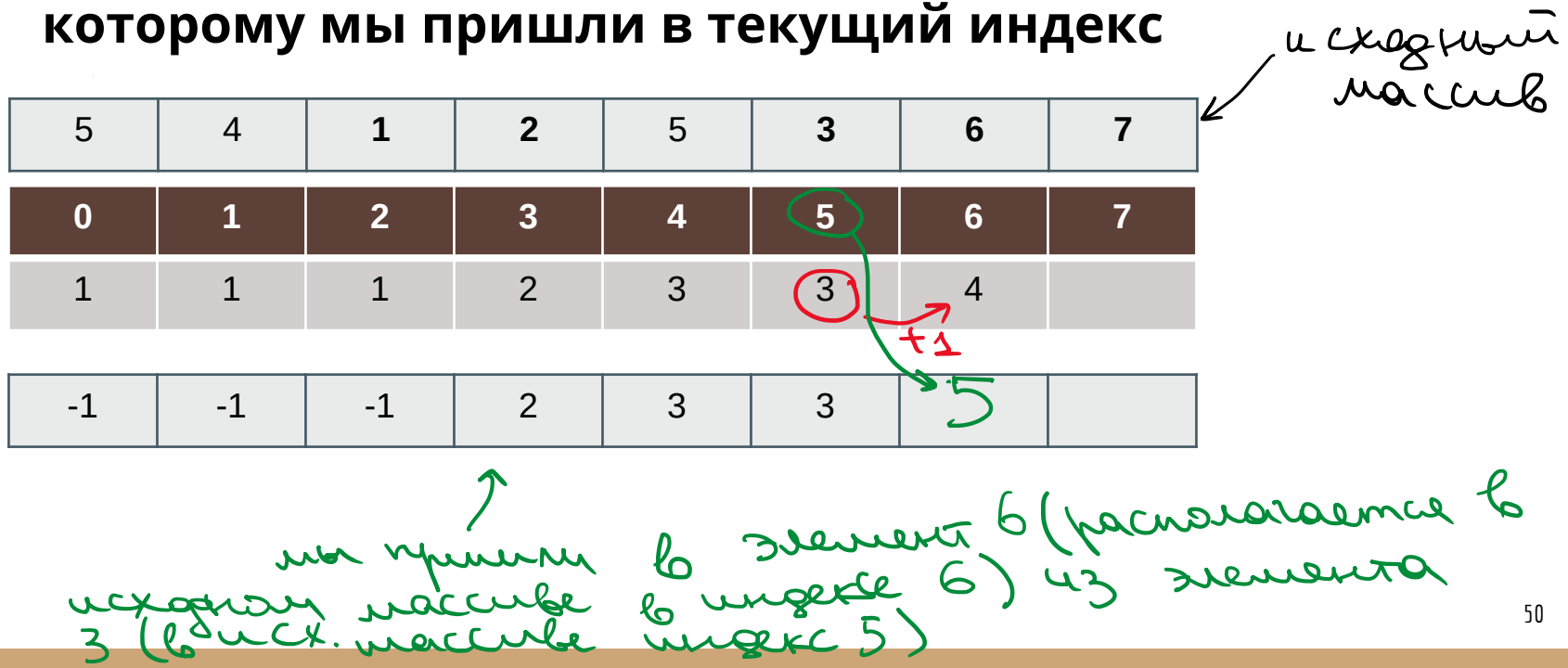
-1	-1	-1	2	3			

мы пришли в элемент 3 (расположенный в исходном массиве в индексе 5) из элемента 2 (в исх. массиве индекс 3)

Наибольшая возрастающая подпоследовательность

Как восстановить ответ?

Хранить массив prev, где каждый элемент – индекс, по которому мы пришли в текущий индекс



Наибольшая возрастающая подпоследовательность

Как восстановить ответ?

Хранить массив prev, где каждый элемент – индекс, по которому мы пришли в текущий индекс

исходный массив

5	4	1	2	5	3	6	7

0	1	2	3	4	5	6	7
1	1	1	2	3	3	4	5

+1

prev

-1	-1	-1	2	3	3	5	6

мы пришли в элемент 7 (расположенный в исходном массиве в индексе 7) из элемента 6 (в исх. массиве индекс 6)

Наибольшая возрастающая подпоследовательность

Как восстановить ответ?

Хранить массив prev, где каждый элемент – индекс, по которому мы пришли в текущий индекс

и сходный массив

5	4	**1**	**2**	5	**3**	**6**	**7**

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
1	1	1	2	3	3	4	5

prev

-1	-1	-1	2	3	3	5	6

Наибольшая возрастающая подпоследовательность

Как восстановить ответ?

Теперь умеем ходить по массиву prev и выводить все элементы из финальной последовательности

и исходный массив

5	4	**1**	**2**	5	**3**	**6**	**7**

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
1	1	1	2	3	3	4	5

prev

-1	-1	-1	2	3	3	5	6

Наибольшая возрастающая подпоследовательность

Как восстановить ответ?

Теперь умеем ходить по массиву prev и выводить все элементы из финальной последовательности

и исходный массив

5	4	**1**	**2**	5	**3**	**6**	**7**

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
1	1	1	2	3	3	4	5

prev

-1	-1	-1	2	3	3	5	6

За сколько работает такой алгоритм?

Наибольшая возрастающая подпоследовательность

Время работы - O(n^2)

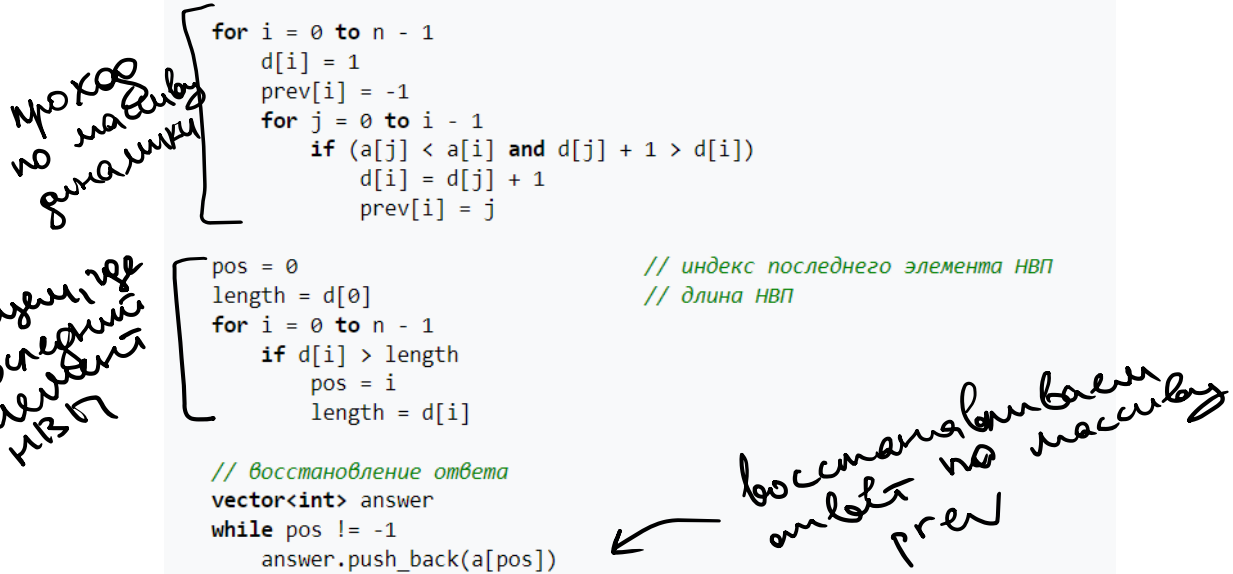
```
vector<int> findLIS(vector<int> a):
    int n = a.size                          //размер исходной последовательности
    int prev[0..n - 1]
    int d[0..n - 1]

    for i = 0 to n - 1
        d[i] = 1
        prev[i] = -1
        for j = 0 to i - 1
            if (a[j] < a[i] and d[j] + 1 > d[i])
                d[i] = d[j] + 1
                prev[i] = j

    pos = 0                                 // индекс последнего элемента НВП
    length = d[0]                           // длина НВП
    for i = 0 to n - 1
        if d[i] > length
            pos = i
            length = d[i]

    // восстановление ответа
    vector<int> answer
    while pos != -1
        answer.push_back(a[pos])
        pos = prev[pos]
    reverse(answer)

    return answer
```



Наибольшая возрастающая подпоследовательность

Время работы - O(n^2)

Как сделать быстрее?

```
vector<int> findLIS(vector<int> a):
    int n = a.size                          //размер исходной последовательности
    int prev[0..n - 1]
    int d[0..n - 1]

    for i = 0 to n - 1
        d[i] = 1
        prev[i] = -1
        for j = 0 to i - 1
            if (a[j] < a[i] and d[j] + 1 > d[i])
                d[i] = d[j] + 1
                prev[i] = j

    pos = 0                                 // индекс последнего элемента НВП
    length = d[0]                           // длина НВП
    for i = 0 to n - 1
        if d[i] > length
            pos = i
            length = d[i]

    // восстановление ответа
    vector<int> answer
    while pos != -1
        answer.push_back(a[pos])
        pos = prev[pos]
    reverse(answer)

    return answer
```

Наибольшая возрастающая подпоследовательность

Время работы - O(n^2)

Как сделать быстрее?

Поменять динамику

```
vector<int> findLIS(vector<int> a):
    int n = a.size                          //размер исходной последовательности
    int prev[0..n - 1]
    int d[0..n - 1]

    for i = 0 to n - 1
        d[i] = 1
        prev[i] = -1
        for j = 0 to i - 1
            if (a[j] < a[i] and d[j] + 1 > d[i])
                d[i] = d[j] + 1
                prev[i] = j

    pos = 0                                 // индекс последнего элемента НВП
    length = d[0]                           // длина НВП
    for i = 0 to n - 1
        if d[i] > length
            pos = i
            length = d[i]

    // восстановление ответа
    vector<int> answer
    while pos != -1
        answer.push_back(a[pos])
        pos = prev[pos]
    reverse(answer)

    return answer
```

Наибольшая возрастающая подпоследовательность

5	4	**1**	**2**	5	**3**	**6**	**7**

d_i - ?

Наибольшая возрастающая подпоследовательность

5	4	**1**	**2**	5	**3**	**6**	**7**

d_i - массив чисел, на которые кончаются НВП длины i
если таких чисел несколько - min из них

Наибольшая возрастающая подпоследовательность

d_i - массив чисел, на которые кончаются НВП длины i
если таких чисел несколько - min из них

5	4	1	2	5	3	6	7

0	1	2	3	4	5	6	7
x	5	∞	∞	∞	∞	∞	∞

При инициализации будем говорить, что нулевых последовательностей нет, а 5 дает последовательность длины 1.

Наибольшая возрастающая подпоследовательность

d_i - массив чисел, на которые кончаются НВП длины i
если таких чисел несколько - min из них

5	4	1	2	5	3	6	7

0	1	2	3	4	5	6	7
x	4	∞	∞	∞	∞	∞	∞

Тут можем только уменьшить макс. число для длины 1.

Наибольшая возрастающая подпоследовательность

d_i - массив чисел, на которые кончаются НВП длины i
если таких чисел несколько - min из них

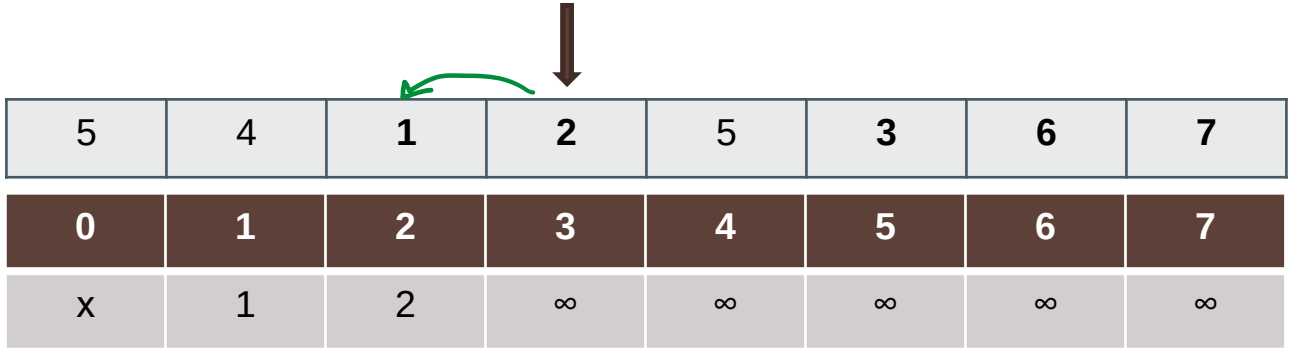
5	4	**1**	**2**	5	**3**	**6**	**7**

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
x	1	∞	∞	∞	∞	∞	∞

То же самое.

Наибольшая возрастающая подпоследовательность

d_i - массив чисел, на которые кончаются НВП длины i
если таких чисел несколько - min из них



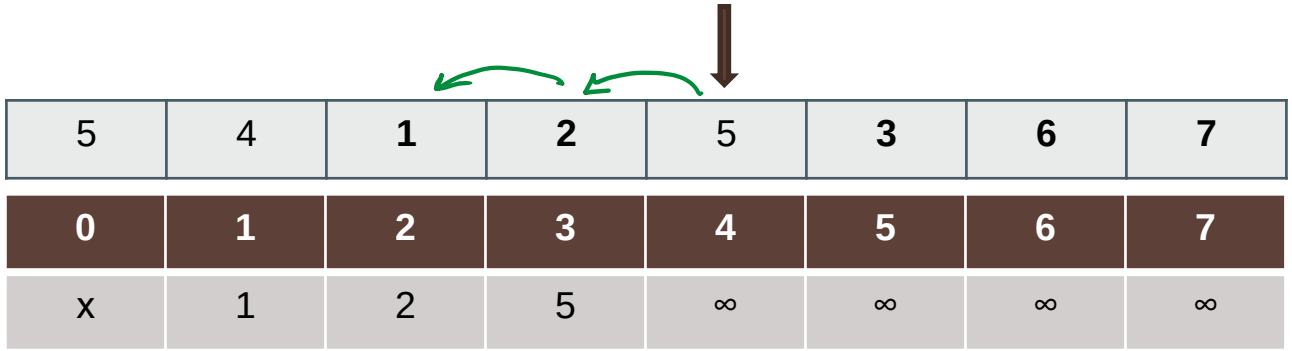
5	4	**1**	**2**	5	**3**	**6**	**7**

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
x	1	2	∞	∞	∞	∞	∞

Находим, что есть подпосл. длины 1, которая кончается на 1, значит можем приписать к ней 2.

Наибольшая возрастающая подпоследовательность

d_i - массив чисел, на которые кончаются НВП длины i
если таких чисел несколько - min из них



5	4	**1**	**2**	5	**3**	**6**	**7**

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
x	1	2	5	∞	∞	∞	∞

Находим, что есть послед. длины 2, которая кончается на 2, значит можем приписать к ней 5.

Наибольшая возрастающая подпоследовательность

d_i - массив чисел, на которые кончаются НВП длины i
если таких чисел несколько - min из них

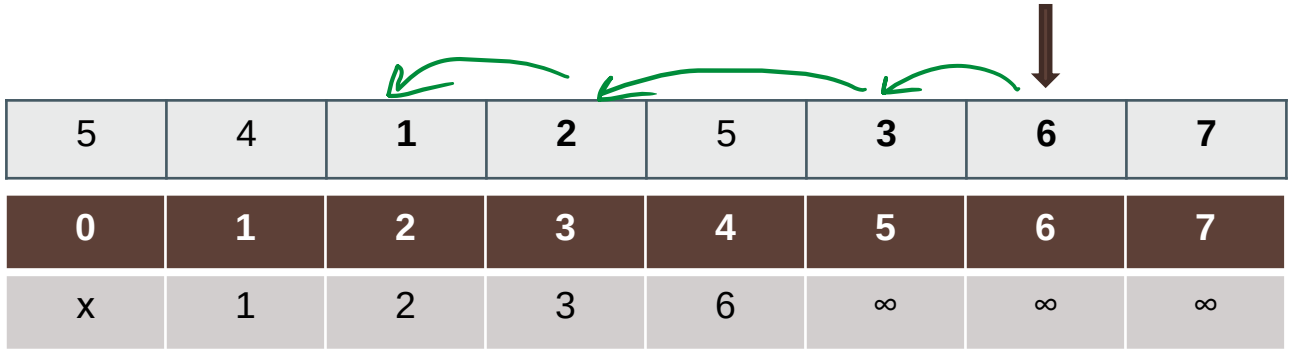
5	4	**1**	**2**	5	**3**	**6**	**7**

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
x	1	2	3	∞	∞	∞	∞

Видим что макс. число, которое меньше 3 в нашей динамике — 2, значит можем сделать посл. длины 3, которая содержит 1, 2, 3

Наибольшая возрастающая подпоследовательность

d_i - массив чисел, на которые кончаются НВП длины i
если таких чисел несколько - min из них



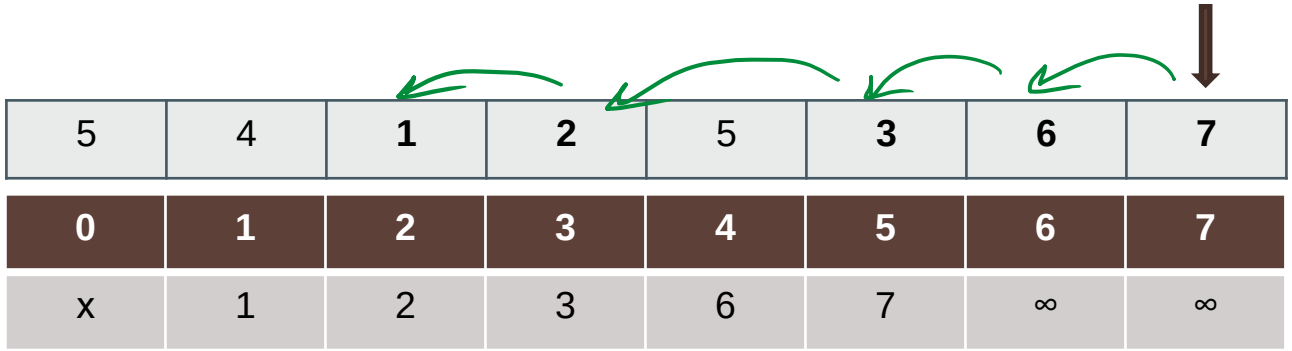
5	4	**1**	**2**	5	**3**	**6**	**7**

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
x	1	2	3	6	∞	∞	∞

Находим, что есть посл. длины 3, которая кончается на 3, значит можем приписать к ней 6.

Наибольшая возрастающая подпоследовательность

d_i - массив чисел, на которые кончаются НВП длины i
если таких чисел несколько - min из них



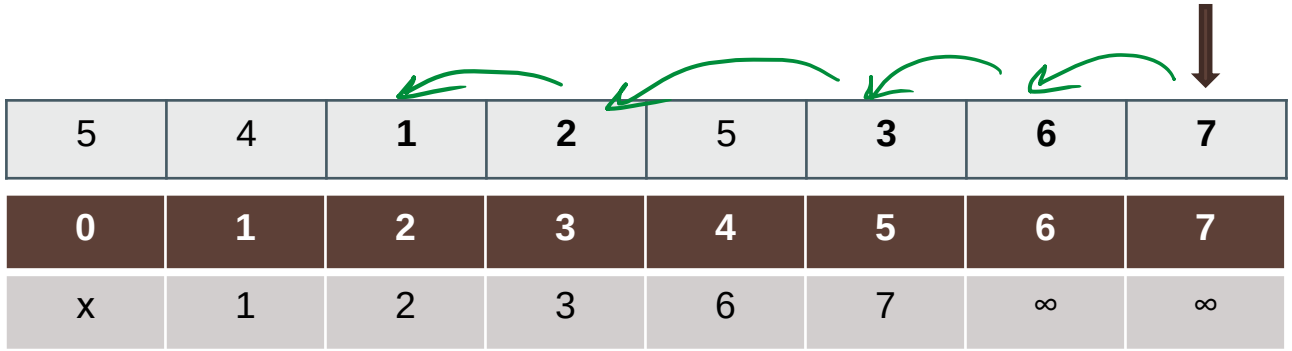
5	4	**1**	**2**	5	**3**	**6**	**7**

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
x	1	2	3	6	7	∞	∞

Находим, что есть посл. длины 4, которая кончается на 6, значит дойдем пути к ней 7.

Наибольшая возрастающая подпоследовательность

d_i - массив чисел, на которые кончаются НВП длины i
если таких чисел несколько - min из них



5	4	**1**	**2**	5	**3**	**6**	**7**

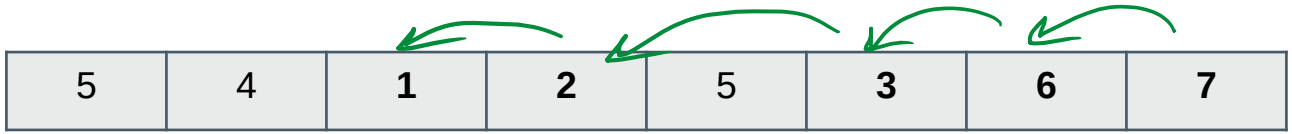
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
x	1	2	3	6	7	∞	∞

ПОЧЕМУ ЭТО БЫСТРЕЕ?

Наибольшая возрастающая подпоследовательность

Чем это быстрее?

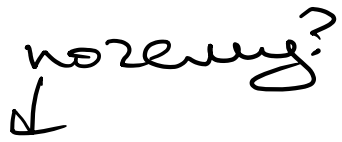
d[i-1] <= d[i], обновляем только один элемент



5	4	**1**	**2**	5	**3**	**6**	**7**

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
x	1	2	3	6	7	∞	∞

Наибольшая возрастающая подпоследовательность



Чем это быстрее?

d[i-1] <= d[i], обновляем только один элемент

5	4	**1**	**2**	5	**3**	**6**	**7**

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
x	1	2	3	6	7	∞	∞

Наибольшая возрастающая подпоследовательность

потому что всегда выбираем минимальный элемент из возможных

Чем это быстрее?

d[i-1] <= d[i], обновляем только один элемент

5	4	1	2	5	3	6	7

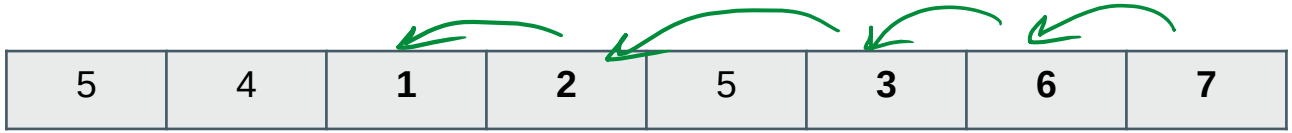
0	1	2	3	4	5	6	7
x	1	2	3	6	7	∞	∞

Наибольшая возрастающая подпоследовательность

Чем это быстрее?

d[i-1] <= d[i], обновляем только один элемент

Можем использовать бинпоиск для поиска самого правого элемента меньше нашего



5	4	**1**	**2**	5	**3**	**6**	**7**

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
x	1	2	3	6	7	∞	∞

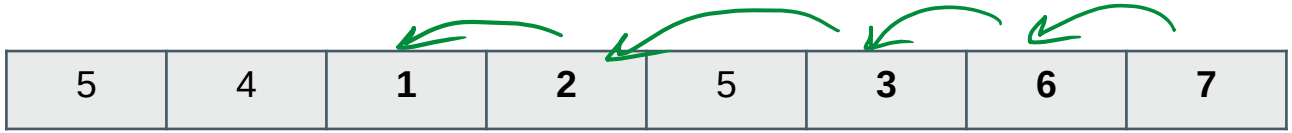
Наибольшая возрастающая подпоследовательность

Чем это быстрее?

d[i-1] <= d[i], обновляем только один элемент

Можем использовать бинпоиск для поиска самого правого элемента меньше нашего

Время работы: ???



5	4	**1**	**2**	5	**3**	**6**	**7**

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
x	1	2	3	6	7	∞	∞

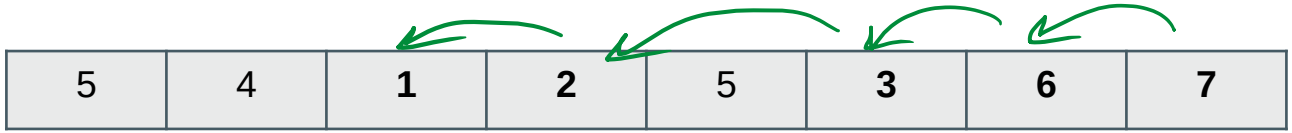
Наибольшая возрастающая подпоследовательность

Чем это быстрее?

d[i-1] <= d[i], обновляем только один элемент

Можем использовать бинпоиск для поиска самого правого элемента меньше нашего

Время работы: O(NlogN)



5	4	**1**	**2**	5	**3**	**6**	**7**

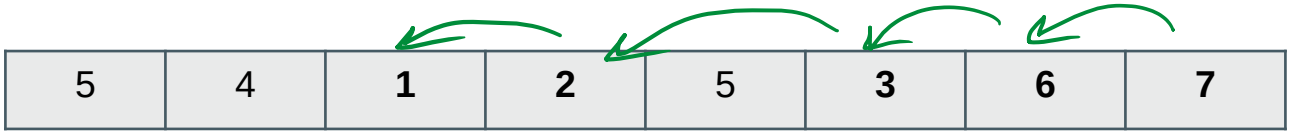
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
x	1	2	3	6	7	∞	∞

Наибольшая возрастающая подпоследовательность

Чем это быстрее?

d[i-1] <= d[i], обновляем только один элемент

Как восстановить ответ?



5	4	**1**	**2**	5	**3**	**6**	**7**

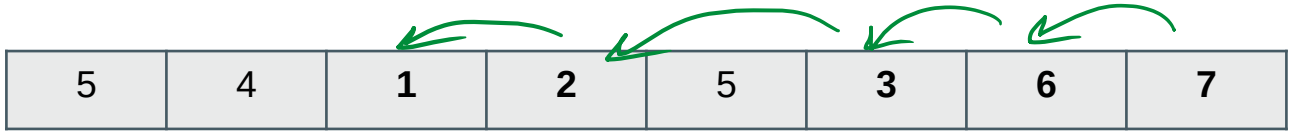
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
x	1	2	3	6	7	∞	∞

Наибольшая возрастающая подпоследовательность

Чем это быстрее?

d[i-1] <= d[i], обновляем только один элемент

Восстановление ответа по тому же принципу, что и в первом решении



5	4	**1**	**2**	5	**3**	**6**	**7**

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
x	1	2	3	6	7	∞	∞

-1	-1	-1	2	3	3	4	5

prev

Наибольшая возрастающая подпоследовательность

```
vector<int> findLIS(vector<int> a):
    int n = a.size                          //размер исходной последовательности
    int d[0..n]
    int pos[0..n]
    int prev[0..n - 1]
    length = 0

    pos[0] = -1
    d[0] = -∞
    for i = 1 to n
        d[i] = ∞
    for i = 0 to n - 1
        j = binary_search(d, a[i])
        if (d[j - 1] < a[i] and a[i] < d[j])
            d[j] = a[i]
            pos[j] = i
            prev[i] = pos[j - 1]
            length = max(length, j)

    // восстановление ответа
    vector<int> answer
    p = pos[length]
    while p != -1
        answer.push_back(a[p])
        p = prev[p]
    reverse(answer)

    return answer
```