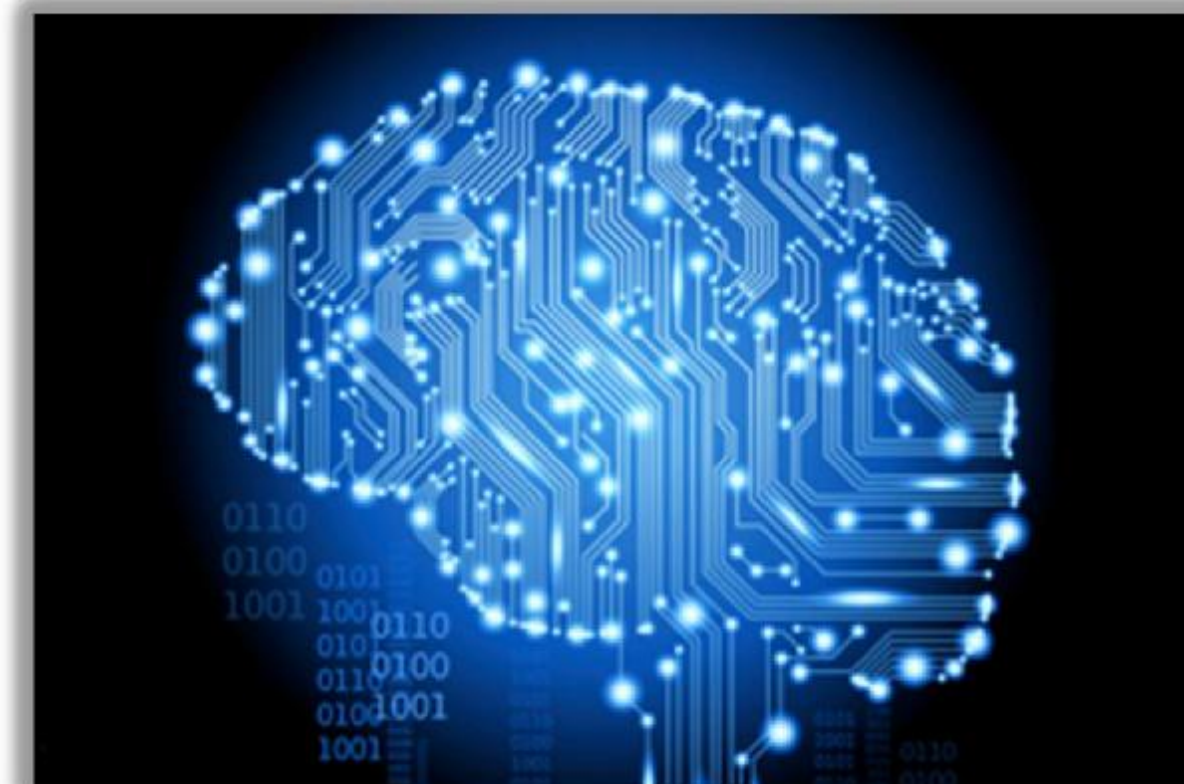


PISL 04

PISL04. Разделяй и властвуй

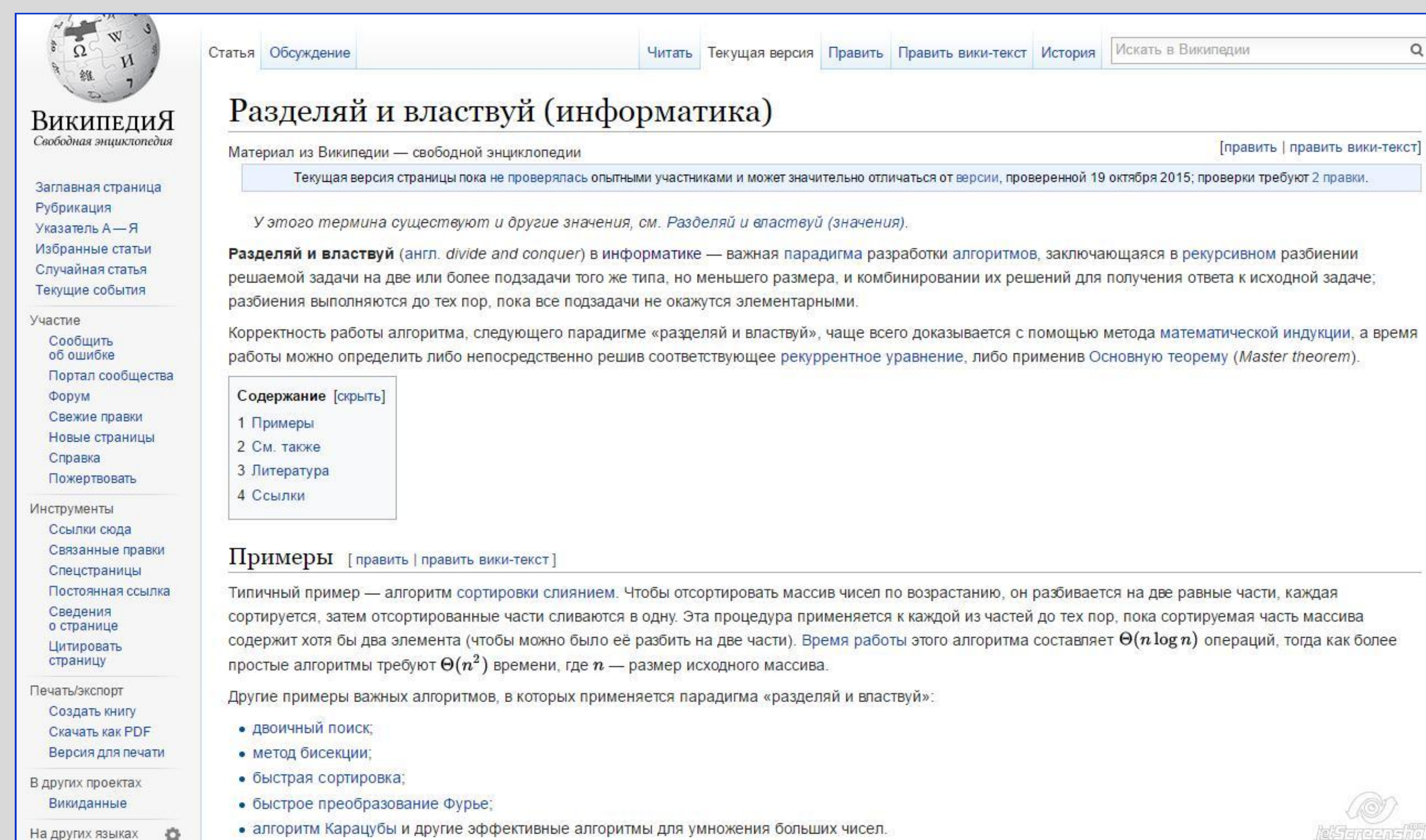


PISL04. Разделяй и властвуй.

- ⌘ Определение.
- ⌘ Принцип «разделяй и властвуй».
- ⌘ Операции сложения
- ⌘ Операции умножения
- ⌘ Умножение матриц.
- ⌘ Сортировка вставками
- ⌘ Сортировка слиянием
- ⌘ данных для приоритетных очередей, сортировок и т.д.
- ⌘ Задачи (A) (B) (C)

- ⌘ Материалы: <http://tinyurl.com/ei-pisl>
- ⌘ Github: <https://github.com/Khmelov/PISL2017-01-26>

PISL04. Разделяй и властвуй.



PISL04. Разделяй и властвуй.

«Разделяй и властвуй»

- Задача разбивается на несколько более простых подзадач.
- Подзадачи решаются рекурсивно.
- Из ответов для подзадач строится ответ для исходной подзадачи.

PISLO4. Разделяй и властвуй.

Поиск в неупорядоченном массиве

Поиск в неупорядоченном массиве

Вход: массив $A[1 \dots n]$, ключ k .

Выход: индекс i , такой что $A[i] = k$, или -1 , если такого i нет.

5

PISLO4. Разделяй и властвуй.

Поиск в неупорядоченном массиве

Поиск в неупорядоченном массиве

Вход: массив $A[1 \dots n]$, ключ k .

Выход: индекс i , такой что $A[i] = k$, или -1 , если такого i нет.

Функция $\text{LINEARSEARCH}(A[1 \dots n], k)$

для i от 1 до n :

если $A[i] = k$:

вернуть i

вернуть -1

6

PISLO4. Разделяй и властвуй.

Поиск в неупорядоченном массиве

Поиск в неупорядоченном массиве

Вход: массив $A[1 \dots n]$, ключ k .

Выход: индекс i , такой что $A[i] = k$, или -1 , если такого i нет.

Функция $\text{LINEARSEARCH}(A[1 \dots n], k)$

для i от 1 до n :

если $A[i] = k$:

вернуть i

вернуть -1

Время работы: $\Theta(n)$.

7

PISLO4. Разделяй и властвуй.

Поиск в упорядоченном массиве

Поиск в упорядоченном массиве

Вход: упорядоченный массив $A[1 \dots n]$
($A[1] \leq A[2] \leq \dots \leq A[n]$), ключ k .

Выход: индекс i , такой что $A[i] = k$, или -1 , если такого i нет.

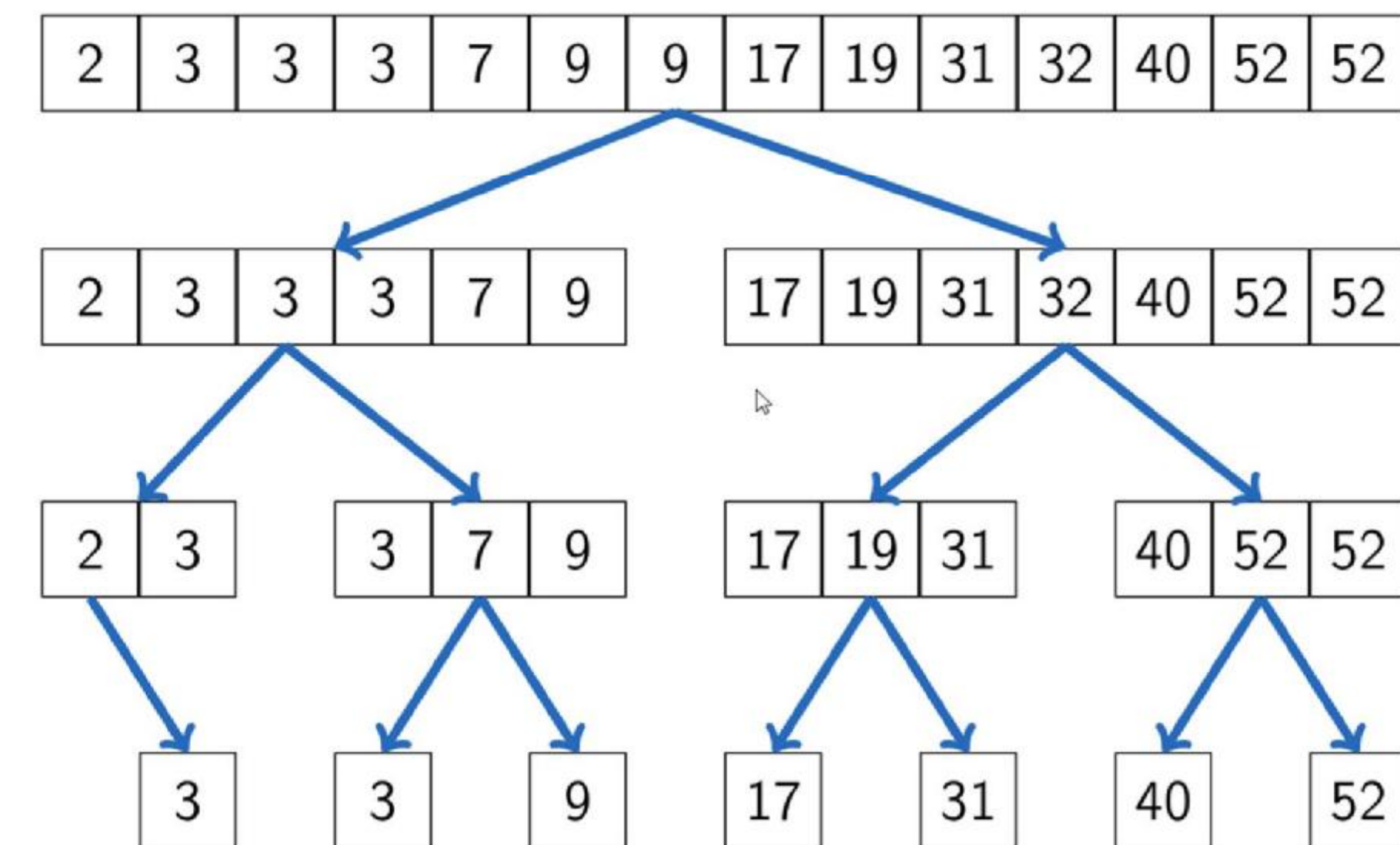
8

Поиск в упорядоченном массиве

2	3	3	3	7	9	9	17	19	31	32	40	52	52
---	---	---	---	---	---	---	----	----	----	----	----	----	----

9

Поиск в упорядоченном массиве



10

Двоичный поиск

Функция $\text{BINARYSEARCH}(A[1 \dots n], k)$

```

{A упорядочен}
 $\ell \leftarrow 1, r \leftarrow n$ 
пока  $\ell \leq r$ :
   $m \leftarrow \lfloor \frac{\ell+r}{2} \rfloor$ 
  если  $A[m] = k$ :
    вернуть  $m$ 
  иначе если  $A[m] > k$ :
     $r \leftarrow m - 1$ 
  иначе:
     $\ell \leftarrow m + 1$ 
вернуть  $-1$ 

```

11

Двоичный поиск

Функция $\text{BINARYSEARCH}(A[1 \dots n], k)$

```

{A упорядочен}
 $\ell \leftarrow 1, r \leftarrow n$ 
пока  $\ell \leq r$ :
   $m \leftarrow \lfloor \frac{\ell+r}{2} \rfloor$ 
  если  $A[m] = k$ :
    вернуть  $m$ 
  иначе если  $A[m] > k$ :
     $r \leftarrow m - 1$ 
  иначе:
     $\ell \leftarrow m + 1$ 
вернуть  $-1$ 

```

Время работы: $O(\log n)$.

12

PISLO4. Разделяй и властвуй.

Сложение в столбик: $O(n)$

$$\begin{array}{r}
 \text{перенос: } 1 \qquad \qquad \qquad 1 \quad 1 \quad 1 \\
 \begin{array}{ccccccc}
 1 & 1 & 0 & 1 & 0 & 1 & (53_2) \\
 1 & 0 & 0 & 0 & 1 & 1 & (35_2) \\
 \hline
 1 & 0 & 1 & 1 & 0 & 0 & 0 & (88_2)
 \end{array}
 \end{array}$$

PISLO4. Разделяй и властвуй.

Умножение в столбик: $O(n^2)$

$$\begin{array}{r}
 \begin{array}{ccccccc}
 & & & 1 & 1 & 0 & 1 & (13_2) \\
 \times & & & 1 & 0 & 1 & 1 & (11_2) \\
 \hline
 & & & & & & & \\
 & & & & 1 & 1 & 0 & 1 & (1101 \times 1) \\
 & & & 1 & 1 & 0 & 1 & & (1101 \times 1, \text{сдвинутое на } 1) \\
 & & 0 & 0 & 0 & 0 & & & (1101 \times 0, \text{сдвинутое на } 2) \\
 + & 1 & 1 & 0 & 1 & & & & (1101 \times 1, \text{сдвинутое на } 3) \\
 \hline
 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & (143_2)
 \end{array}
 \end{array}$$

PISLO4. Разделяй и властвуй.

Рекуррентная формула

$$y = \begin{cases} 2 \lfloor \frac{y}{2} \rfloor, & \text{если } y \text{ чётно} \\ 1 + 2 \lfloor \frac{y}{2} \rfloor, & \text{если } y \text{ нечётно} \end{cases}$$

PISLO4. Разделяй и властвуй.

Рекуррентная формула

$$y = \begin{cases} 2 \lfloor \frac{y}{2} \rfloor, & \text{если } y \text{ чётно} \\ 1 + 2 \lfloor \frac{y}{2} \rfloor, & \text{если } y \text{ нечётно} \end{cases}$$

$$x \cdot y = \begin{cases} 2(x \cdot \lfloor \frac{y}{2} \rfloor), & \text{если } y \text{ чётно} \\ x + 2(x \cdot \lfloor \frac{y}{2} \rfloor), & \text{если } y \text{ нечётно} \end{cases}$$

Алгоритм

Функция $\text{MULTIPLY}(x, y)$ {Вход: два n -битовых целых числа $x \geq 0$ и $y \geq 0$.}{Выход: xy .}если $y = 0$:

вернуть 0

 $z \leftarrow \text{MULTIPLY}(x, \lfloor y/2 \rfloor)$ если y чётно: вернуть $2z$

иначе:

 вернуть $x + 2z$

Алгоритм

Функция $\text{MULTIPLY}(x, y)$ {Вход: два n -битовых целых числа $x \geq 0$ и $y \geq 0$.}{Выход: xy .}если $y = 0$:

вернуть 0

 $z \leftarrow \text{MULTIPLY}(x, \lfloor y/2 \rfloor)$ если y чётно: вернуть $2z$

иначе:

 вернуть $x + 2z$

Время работы: $O(n^2)$ (число битов в записи y уменьшается на единицу с каждым рекурсивным вызовом).

Ещё одна рекуррентная формула

$$\begin{aligned}
 x &= \begin{bmatrix} x_L & x_R \end{bmatrix} = 2^{n/2}x_L + x_R \\
 y &= \begin{bmatrix} y_L & y_R \end{bmatrix} = 2^{n/2}y_L + y_R
 \end{aligned}$$

Ещё одна рекуррентная формула

$$\begin{aligned}
 x &= \begin{bmatrix} x_L & x_R \end{bmatrix} = 2^{n/2}x_L + x_R \\
 y &= \begin{bmatrix} y_L & y_R \end{bmatrix} = 2^{n/2}y_L + y_R
 \end{aligned}$$

$$\begin{aligned}
 xy &= (2^{n/2}x_L + x_R)(2^{n/2}y_L + y_R) \\
 &= 2^n x_L y_L + 2^{n/2}(x_L y_R + x_R y_L) + x_R y_R
 \end{aligned}$$

Ещё одна рекуррентная формула

$$\begin{aligned} x &= \begin{bmatrix} x_L & x_R \end{bmatrix} = 2^{n/2}x_L + x_R \\ y &= \begin{bmatrix} y_L & y_R \end{bmatrix} = 2^{n/2}y_L + y_R \end{aligned}$$

$$\begin{aligned} xy &= (2^{n/2}x_L + x_R)(2^{n/2}y_L + y_R) \\ &= 2^n x_L y_L + 2^{n/2}(x_L y_R + x_R y_L) + x_R y_R \end{aligned}$$

$$T(n) = 4T\left(\frac{n}{2}\right) + O(n)$$

21

Улучшенная рекуррентная формула

$$\blacksquare xy = 2^n x_L y_L + 2^{n/2}(x_L y_R + x_R y_L) + x_R y_R$$

22

Улучшенная рекуррентная формула

- $xy = 2^n x_L y_L + 2^{n/2}(x_L y_R + x_R y_L) + x_R y_R$
- вместо четырёх рекурсивных вызовов для вычисления $x_L y_L$, $x_L y_R$, $x_R y_L$ и $x_R y_R$, сделаем **три** для вычисления

$$x_L y_L, x_R y_R, \text{ и } (x_L + x_R)(y_L + y_R)$$

23

Улучшенная рекуррентная формула

- $xy = 2^n x_L y_L + 2^{n/2}(x_L y_R + x_R y_L) + x_R y_R$
- вместо четырёх рекурсивных вызовов для вычисления $x_L y_L$, $x_L y_R$, $x_R y_L$ и $x_R y_R$, сделаем **три** для вычисления

$$x_L y_L, x_R y_R, \text{ и } (x_L + x_R)(y_L + y_R)$$

- тогда

$$(x_L y_R + x_R y_L) = (x_L + x_R)(y_L + y_R) - x_L y_L - x_R y_R$$

24

Улучшенная рекуррентная формула

- $xy = 2^n x_L y_L + 2^{n/2}(x_L y_R + x_R y_L) + x_R y_R$
- вместо четырёх рекурсивных вызовов для вычисления $x_L y_L$, $x_L y_R$, $x_R y_L$ и $x_R y_R$, сделаем **три** для вычисления

$$x_L y_L, x_R y_R, \text{ и } (x_L + x_R)(y_L + y_R)$$

- тогда

$$(x_L y_R + x_R y_L) = (x_L + x_R)(y_L + y_R) - x_L y_L - x_R y_R$$

- соответствующее рекуррентное соотношение:

$$T(n) = 3T\left(\frac{n}{2}\right) + O(n)$$

25

Улучшенная рекуррентная формула

- $xy = 2^n x_L y_L + 2^{n/2}(x_L y_R + x_R y_L) + x_R y_R$
- вместо четырёх рекурсивных вызовов для вычисления $x_L y_L$, $x_L y_R$, $x_R y_L$ и $x_R y_R$, сделаем **три** для вычисления

$$x_L y_L, x_R y_R, \text{ и } (x_L + x_R)(y_L + y_R)$$

- тогда

$$(x_L y_R + x_R y_L) = (x_L + x_R)(y_L + y_R) - x_L y_L - x_R y_R$$

- соответствующее рекуррентное соотношение:

$$T(n) = 3T\left(\frac{n}{2}\right) + O(n)$$

- скоро покажем, что $T(n) = O(n^{1.59})$

26

Алгоритм Карацубы

Функция KARATSUBA(x, y)

{Вход: целые числа $x, y \geq 0$, в двоичной записи.}

{Выход: xy .}

$n \leftarrow \max(\text{размер } x, \text{ размер } y)$

если $n = 1$: вернуть xy

$x_L, x_R \leftarrow$ левые $\lceil n/2 \rceil$, правые $\lfloor n/2 \rfloor$ битов x

$y_L, y_R \leftarrow$ левые $\lceil n/2 \rceil$, правые $\lfloor n/2 \rfloor$ битов y

$P_1 \leftarrow \text{KARATSUBA}(x_L, y_L)$

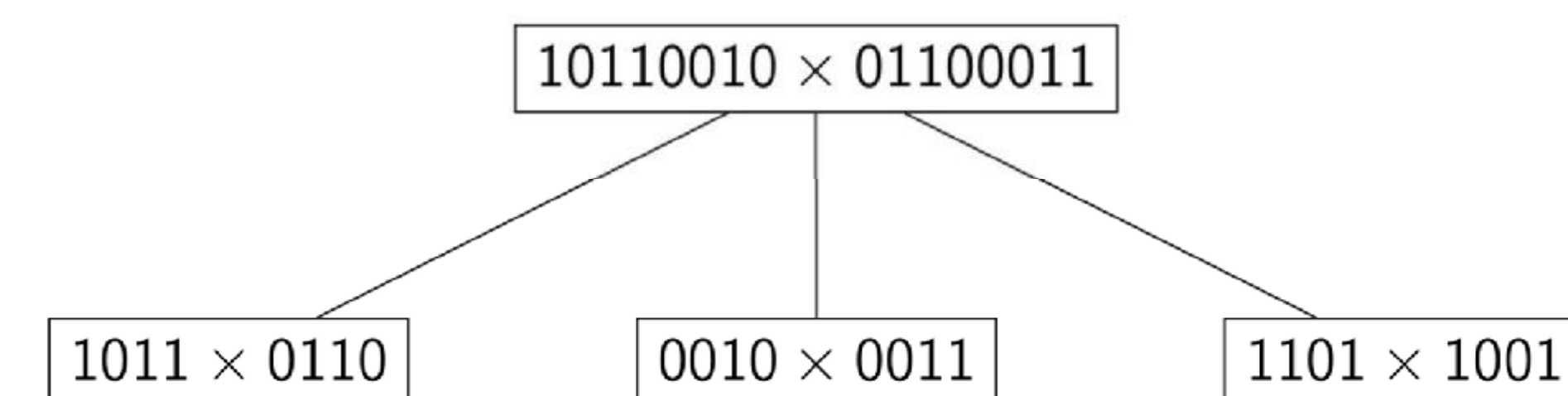
$P_2 \leftarrow \text{KARATSUBA}(x_R, y_R)$

$P_3 \leftarrow \text{KARATSUBA}(x_L + x_R, y_L + y_R)$

вернуть $P_1 \times 2^{2\lfloor n/2 \rfloor} + (P_3 - P_1 - P_2) \times 2^{\lfloor n/2 \rfloor} + P_2$

27

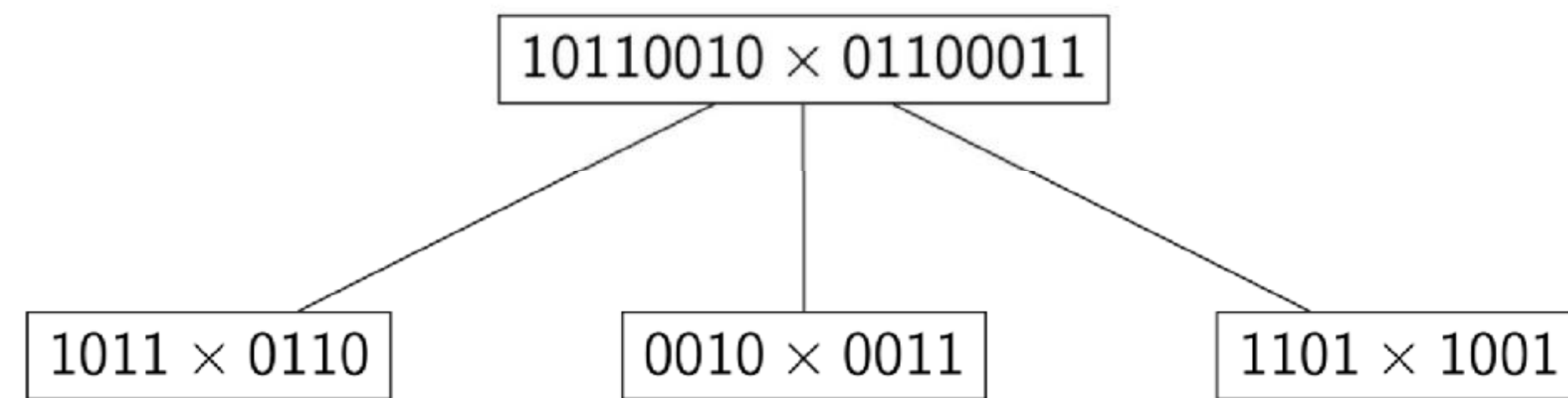
Дерево рекурсии



- $x_L = 1011$, $x_R = 0010$, $y_L = 0110$, $y_R = 0011$

28

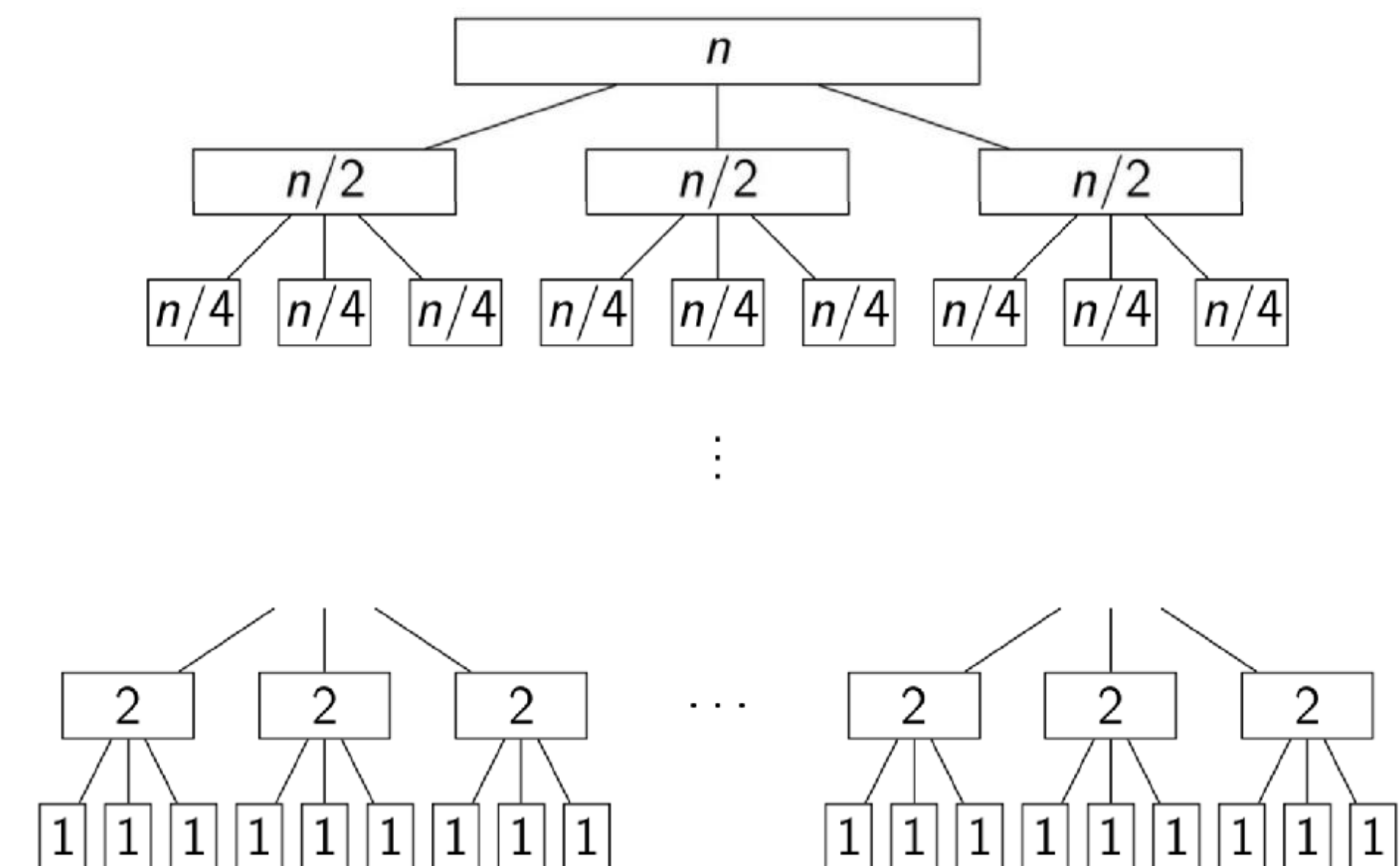
Дерево рекурсии



- $x_L = 1011$, $x_R = 0010$, $y_L = 0110$, $y_R = 0011$
- $x_L + x_R = 1101$, $y_L + y_R = 1001$

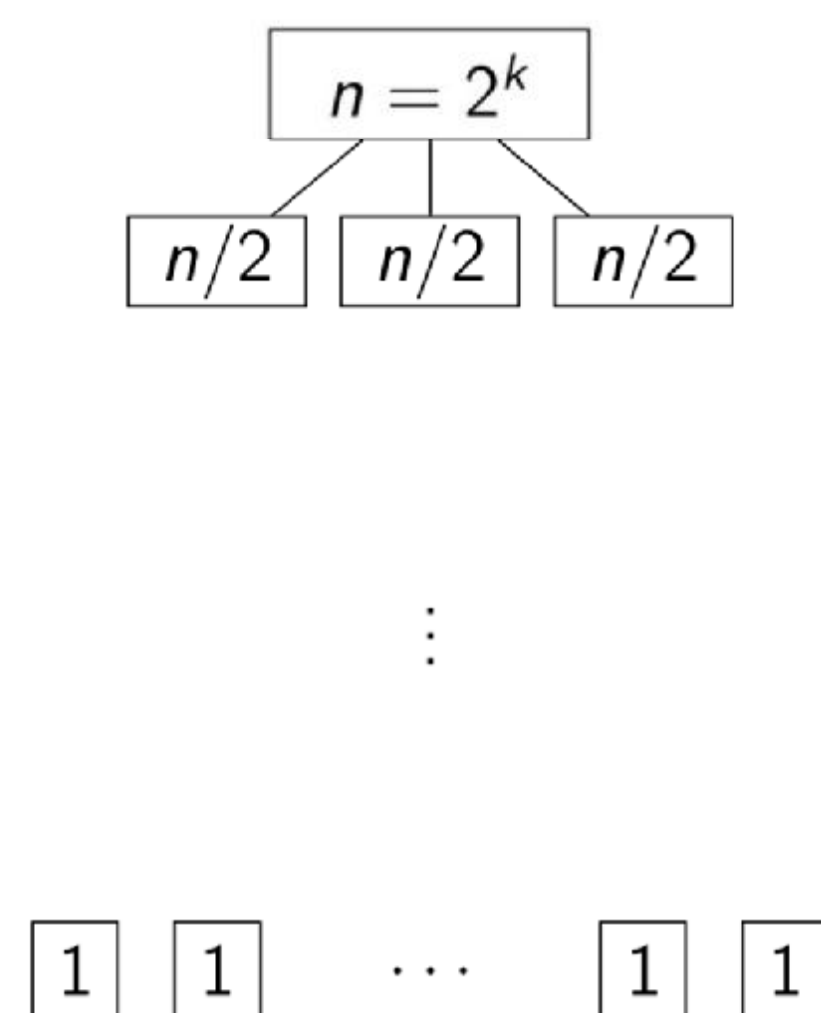
29

Дерево рекурсии



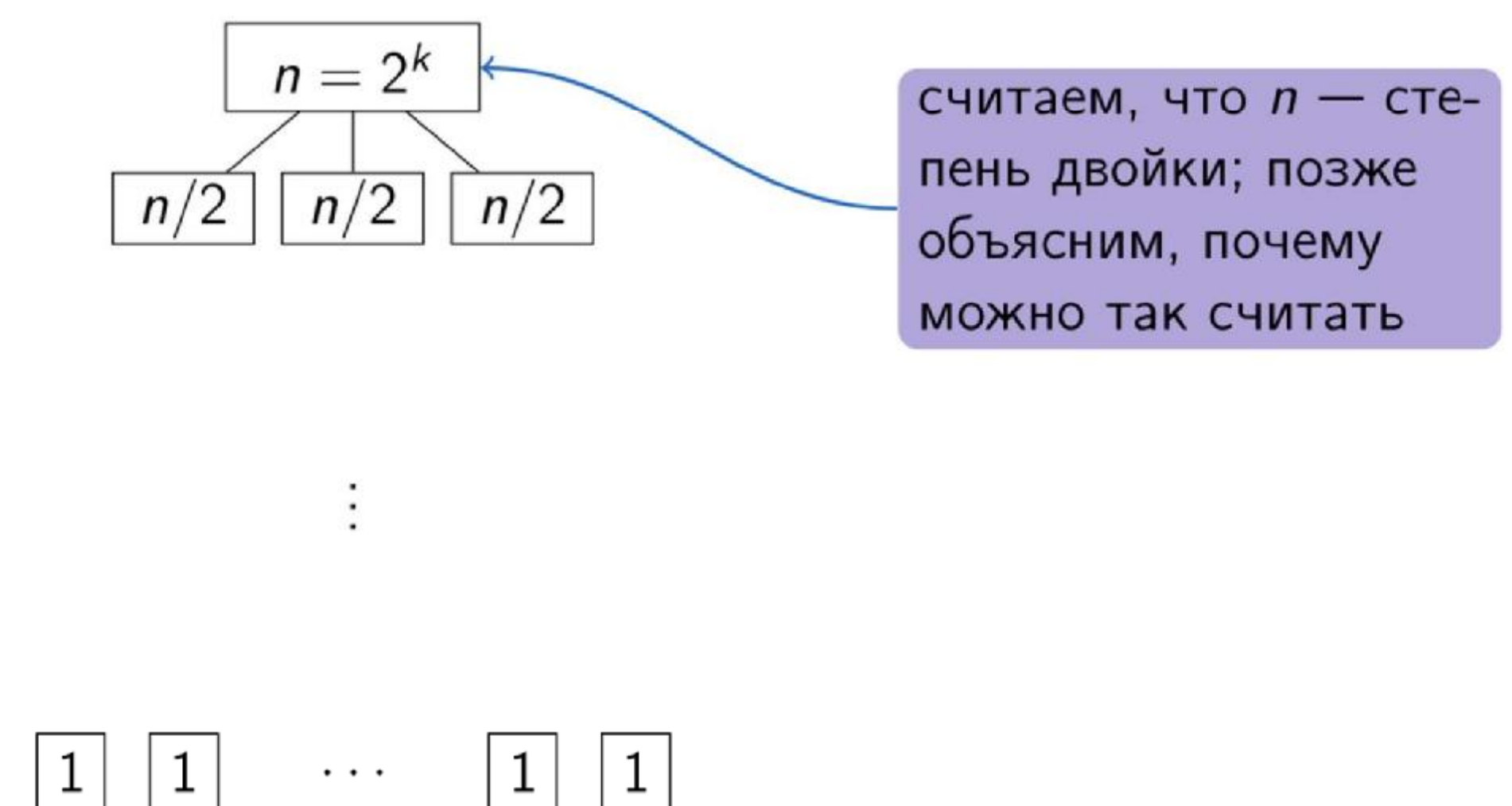
30

Оценка времени работы



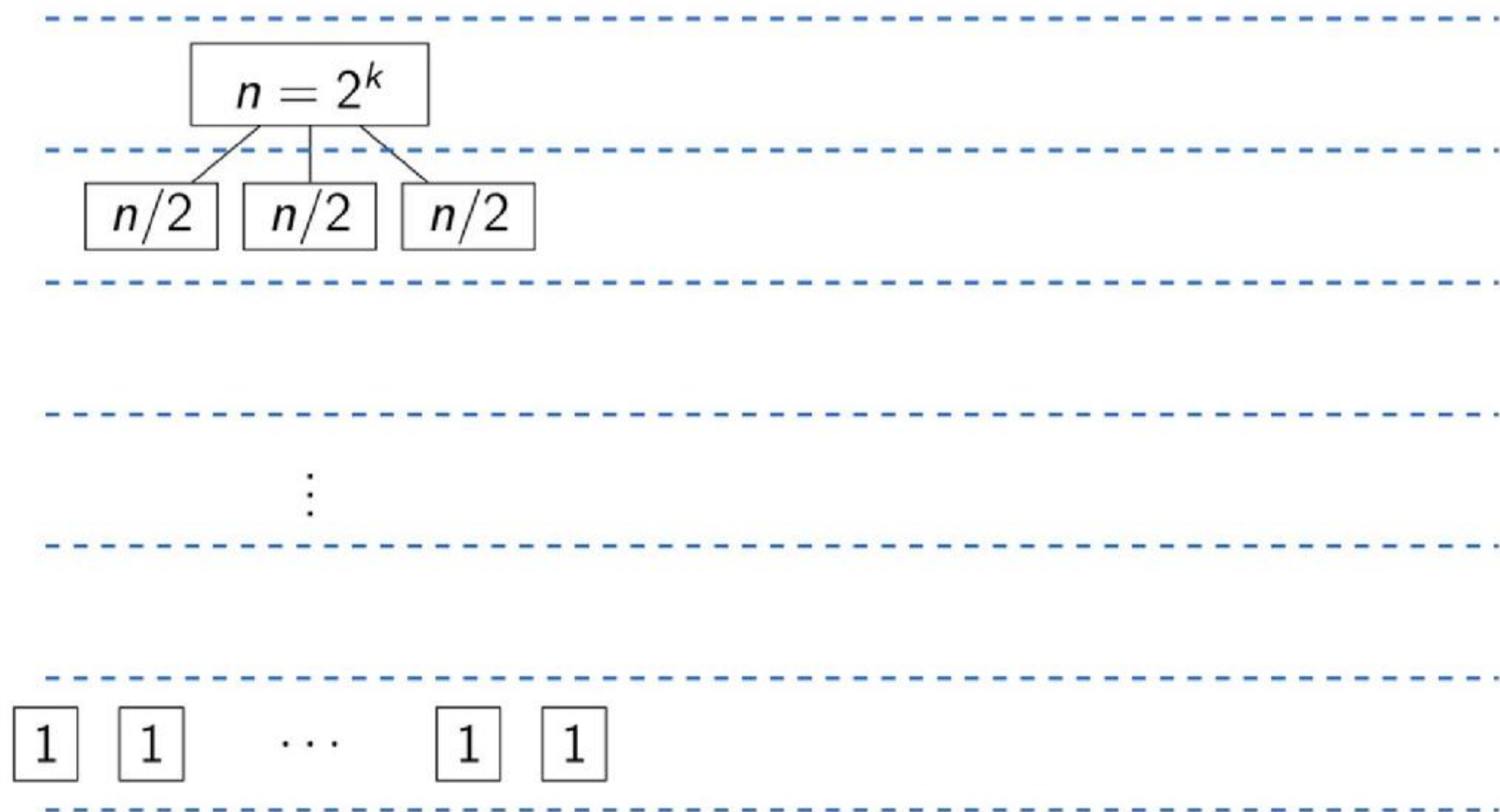
31

Оценка времени работы

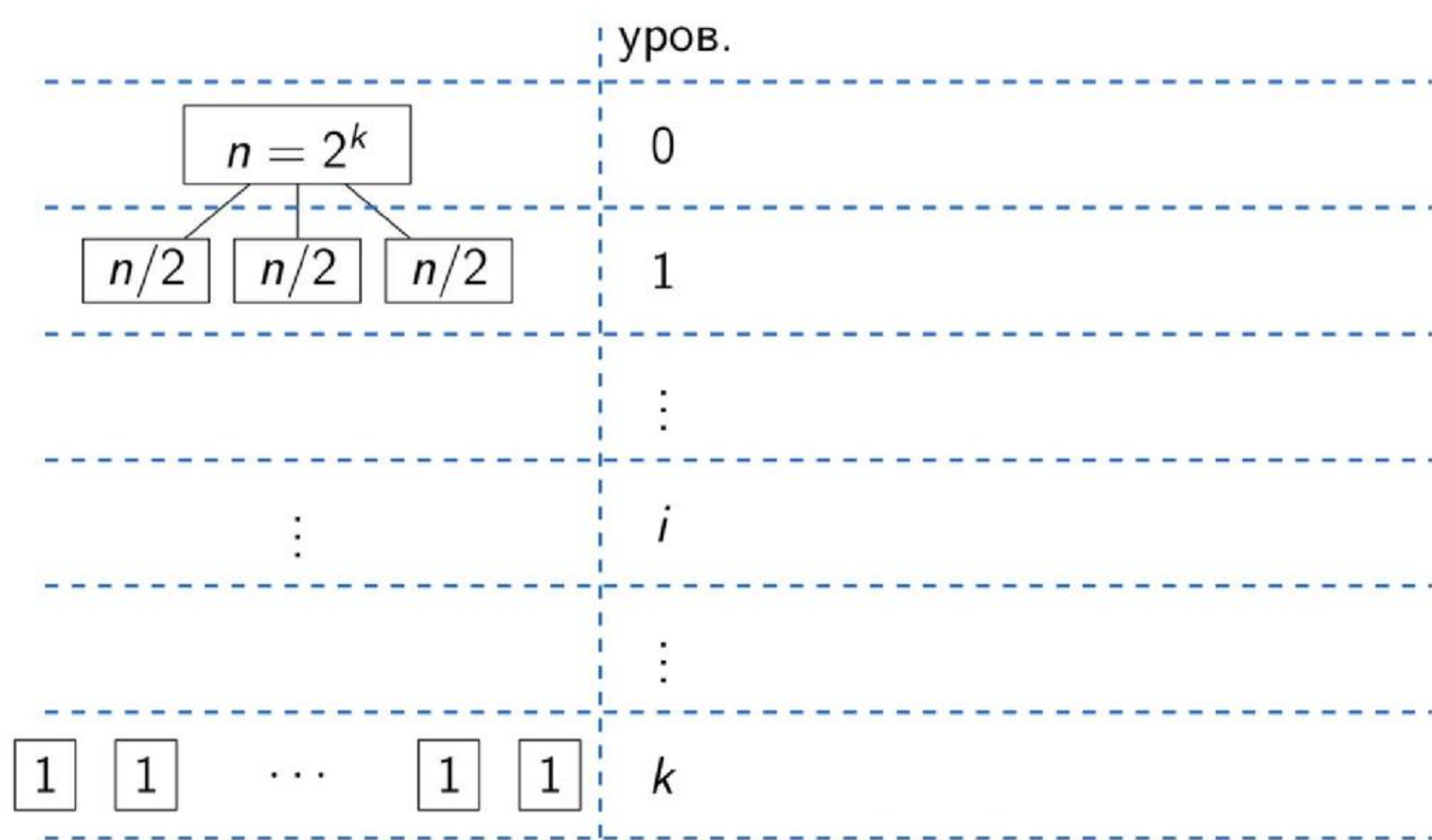


32

Оценка времени работы



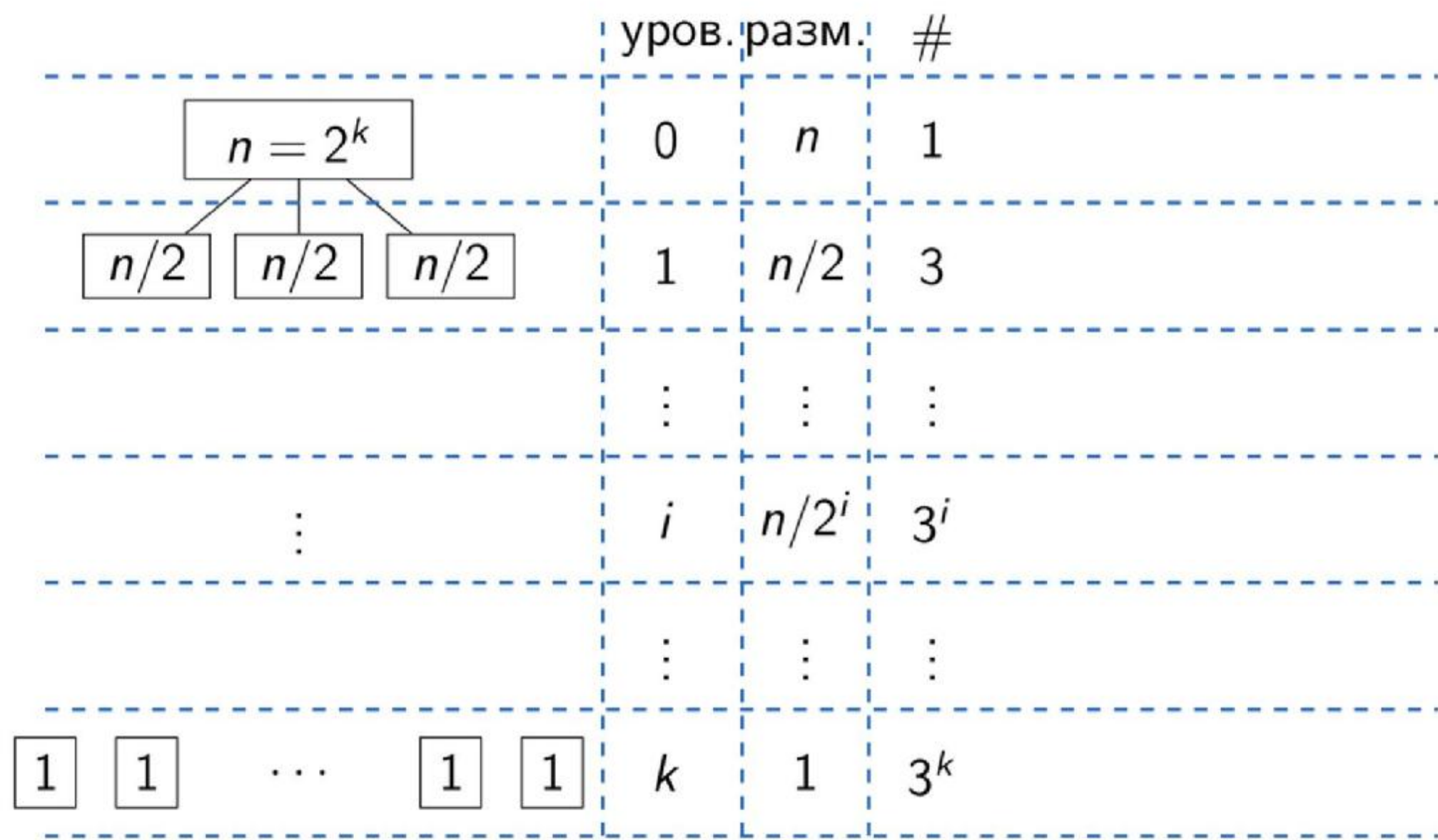
Оценка времени работы



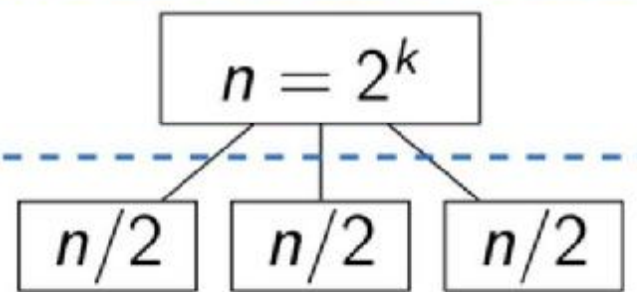
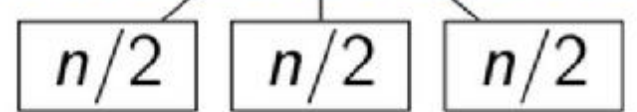

Оценка времени работы



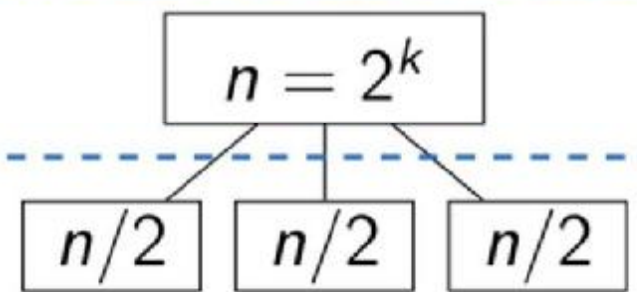
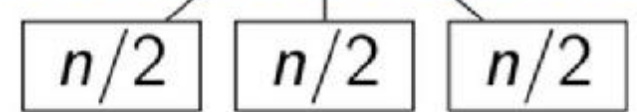

Оценка времени работы



Оценка времени работы

	уров.	разм.	#	работа
	0	n	1	cn
	1	n/2	3	3 · c · n/2
	⋮	⋮	⋮	⋮
⋮	i	n/2 ⁱ	3 ⁱ	3 ⁱ · c · n/2 ⁱ
	⋮	⋮	⋮	⋮
	k	1	3 ^k	3 ^k · c · n/2 ^k

Оценка времени работы

	уров.	разм.	#	работа
	0	n	1	cn
	1	n/2	3	3 · c · n/2
	⋮	⋮	⋮	⋮
⋮	i	n/2 ⁱ	3 ⁱ	3 ⁱ · c · n/2 ⁱ
	⋮	⋮	⋮	⋮
	k	1	3 ^k	3 ^k · c · n/2 ^k

$$\sum_{i=0}^k 3^i \cdot c \cdot n/2^i$$

Сумма геометрической прогрессии: формула

■ геометрическая прогрессия: $1 + c + c^2 + \dots + c^n$

Сумма геометрической прогрессии: формула

- геометрическая прогрессия: $1 + c + c^2 + \dots + c^n$
- если домножить на $(c - 1)$ и раскрыть скобки, почти всё сократится:

$$(c + c^2 + c^3 + \dots + c^{n+1}) - (1 + c + c^2 + \dots + c^n) = c^{n+1} - 1$$

Сумма геометрической
прогрессии: формула

- геометрическая прогрессия: $1 + c + c^2 + \dots + c^n$
- если домножить на $(c - 1)$ и раскрыть скобки, почти всё сократится:

$$(c + c^2 + c^3 + \dots + c^{n+1}) - (1 + c + c^2 + \dots + c^n) = c^{n+1} - 1$$

- поэтому при $c \neq 1$ верно равенство

$$1 + c + c^2 + \dots + c^n = \frac{c^{n+1} - 1}{c - 1}$$

Сумма геометрической
прогрессии: скорость роста

$$1 + c + c^2 + \dots + c^n = \begin{cases} \Theta(1) & \text{если } c < 1 \\ \Theta(n) & \text{если } c = 1 \\ \Theta(c^n) & \text{если } c > 1 \end{cases}$$

Сумма геометрической
прогрессии: скорость роста

$$1 + c + c^2 + \dots + c^n = \begin{cases} \Theta(1) & \text{если } c < 1 \\ \Theta(n) & \text{если } c = 1 \\ \Theta(c^n) & \text{если } c > 1 \end{cases}$$

- Если $c < 1$,

$$1 < \frac{c^{n+1} - 1}{c - 1} = \frac{1 - c^{n+1}}{1 - c} < \frac{1}{1 - c}$$

Сумма геометрической
прогрессии: скорость роста

$$1 + c + c^2 + \dots + c^n = \begin{cases} \Theta(1) & \text{если } c < 1 \\ \Theta(n) & \text{если } c = 1 \\ \Theta(c^n) & \text{если } c > 1 \end{cases}$$

- Если $c < 1$,

$$1 < \frac{c^{n+1} - 1}{c - 1} = \frac{1 - c^{n+1}}{1 - c} < \frac{1}{1 - c}$$

- Если $c > 1$,

$$c^n < \frac{c^{n+1} - 1}{c - 1} < \frac{c}{c - 1} c^n$$

PISLO4. Разделяй и властвуй.

Оценка на время работы

$$\sum_{i=0}^k 3^i \cdot c \cdot n/2^i = cn \cdot \sum_{i=0}^k \left(\frac{3}{2}\right)^i$$

45

Кафедра экономической информатики Бгуир 2017

PISLO4. Разделяй и властвуй.

Оценка на время работы

$$\begin{aligned} \sum_{i=0}^k 3^i \cdot c \cdot n/2^i &= cn \cdot \sum_{i=0}^k \left(\frac{3}{2}\right)^i \\ &= cn \sum_{i=0}^{\log_2 n} \left(\frac{3}{2}\right)^i \end{aligned}$$

46

Кафедра экономической информатики Бгуир 2017

PISLO4. Разделяй и властвуй.

Оценка на время работы

$$\begin{aligned} \sum_{i=0}^k 3^i \cdot c \cdot n/2^i &= cn \cdot \sum_{i=0}^k \left(\frac{3}{2}\right)^i \\ &= cn \sum_{i=0}^{\log_2 n} \left(\frac{3}{2}\right)^i \\ &= cn \cdot \Theta\left(\frac{3^{\log_2 n}}{2^{\log_2 n}}\right) \end{aligned}$$

47

Кафедра экономической информатики Бгуир 2017

PISLO4. Разделяй и властвуй.

Оценка на время работы

$$\begin{aligned} \sum_{i=0}^k 3^i \cdot c \cdot n/2^i &= cn \cdot \sum_{i=0}^k \left(\frac{3}{2}\right)^i \\ &= cn \sum_{i=0}^{\log_2 n} \left(\frac{3}{2}\right)^i \\ &= cn \cdot \Theta\left(\frac{3^{\log_2 n}}{2^{\log_2 n}}\right) \\ &= \Theta(n^{\log_2 3}) \end{aligned}$$

48

Кафедра экономической информатики Бгуир 2017

Оценка на время работы

$$\begin{aligned} \sum_{i=0}^k 3^i \cdot c \cdot n/2^i &= cn \cdot \sum_{i=0}^k \left(\frac{3}{2}\right)^i \\ &= cn \sum_{i=0}^{\log_2 n} \left(\frac{3}{2}\right)^i \\ &= cn \cdot \Theta\left(\frac{3^{\log_2 n}}{2^{\log_2 n}}\right) \\ &= \Theta(n^{\log_2 3}) \\ &= \Theta(n^{1.584\dots}) \end{aligned}$$

49

Почему можно считать, что
 $n = 2^k$?



В отрезке $[n, 2n]$ всегда найдётся степень двойки.

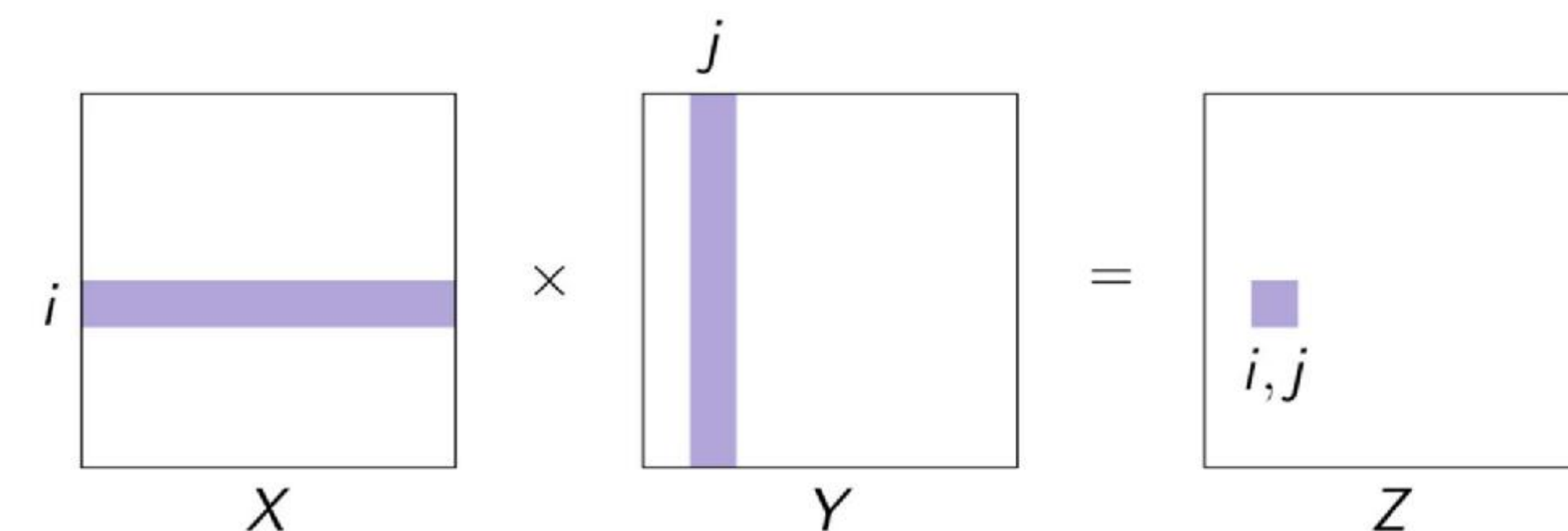
50

Заключение

- сложение двух n -битовых чисел: $O(n)$

51

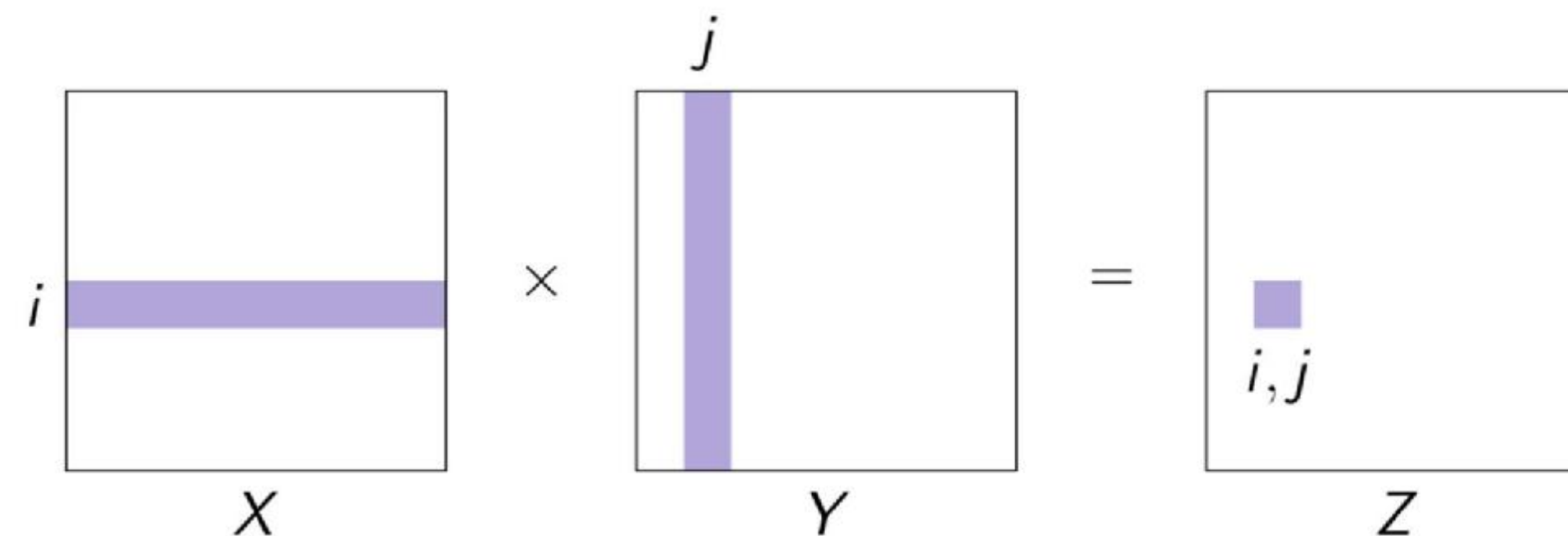
Умножение матриц



$$Z[i,j] = \sum_{k=1}^n X[i,k] \cdot Y[k,j]$$

52

Умножение матриц



$$Z[i, j] = \sum_{k=1}^n X[i, k] \cdot Y[k, j]$$

Время работы наивного алгоритма: $O(n^3)$.

53

Умножение матриц размера 2×2

$$X = \begin{bmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \end{bmatrix} \quad Y = \begin{bmatrix} Y_{11} & Y_{12} \\ Y_{21} & Y_{22} \end{bmatrix}$$

$$XY = \begin{bmatrix} X_{11}Y_{11} + X_{12}Y_{21} & X_{11}Y_{21} + X_{12}Y_{22} \\ X_{21}Y_{11} + X_{22}Y_{21} & X_{21}Y_{21} + X_{22}Y_{22} \end{bmatrix}$$

54

Умножение матриц размера 2×2

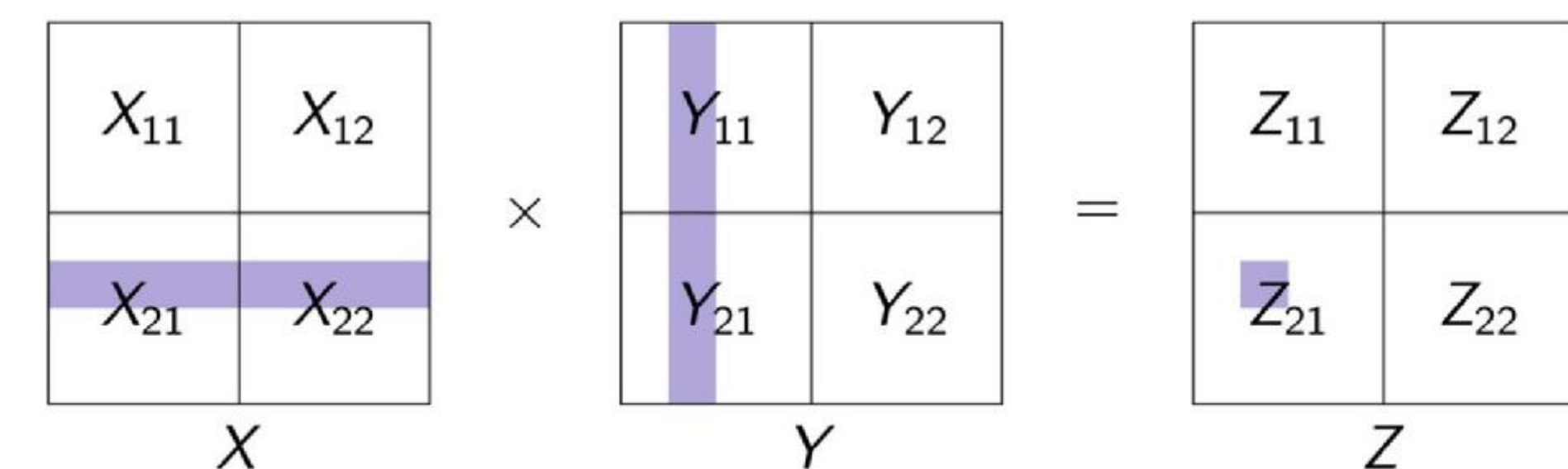
$$X = \begin{bmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \end{bmatrix} \quad Y = \begin{bmatrix} Y_{11} & Y_{12} \\ Y_{21} & Y_{22} \end{bmatrix}$$

$$XY = \begin{bmatrix} X_{11}Y_{11} + X_{12}Y_{21} & X_{11}Y_{21} + X_{12}Y_{22} \\ X_{21}Y_{11} + X_{22}Y_{21} & X_{21}Y_{21} + X_{22}Y_{22} \end{bmatrix}$$

Данная формула верна и в случае, когда X и Y — это матрицы размера $n \times n$, а X_{ij} , Y_{ij} — это их подматрицы размера $n/2 \times n/2$!

55

Визуальное пояснение



$$Z_{21} = X_{21}Y_{11} + X_{22}Y_{21}$$

56

Рекурсивный алгоритм со
временем работы $O(n^3)$

$$XY = \begin{bmatrix} X_{11}Y_{11} + X_{12}Y_{21} & X_{11}Y_{21} + X_{12}Y_{22} \\ X_{21}Y_{11} + X_{22}Y_{21} & X_{21}Y_{21} + X_{22}Y_{22} \end{bmatrix}$$

$$T(n) = 8T(n/2) + O(n^2)$$

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

Алгоритм Штрассена

$$XY = \begin{bmatrix} X_{11}Y_{11} + X_{12}Y_{21} & X_{11}Y_{12} + X_{12}Y_{22} \\ X_{21}Y_{11} + X_{22}Y_{21} & X_{21}Y_{12} + X_{22}Y_{22} \end{bmatrix}$$

$$XY = \begin{bmatrix} P_5 + P_4 - P_2 + P_6 & P_1 + P_2 \\ P_3 + P_4 & P_1 + P_5 - P_3 - P_7 \end{bmatrix}$$

$$\begin{aligned} P_1 &= X_{11}(Y_{21} - Y_{22}) & P_5 &= (X_{11} + X_{22})(Y_{11} + Y_{22}) \\ P_2 &= (X_{11} + X_{12})Y_{22} & P_6 &= (X_{12} - X_{22})(Y_{21} + Y_{22}) \\ P_3 &= (X_{21} + X_{22})Y_{11} & P_7 &= (X_{11} - X_{21})(Y_{11} + Y_{12}) \\ P_4 &= X_{22}(Y_{21} - Y_{11}) \end{aligned}$$

Лемма

Время работы алгоритма Штрассена есть
 $O(n^{\log_2 7}) = O(n^{2.807...})$.

Доказательство

- Рекуррентное соотношение: $T(n) = 7T(n/2) + O(n^2)$.
- В дереве рекурсии будет $\log_2 n$ уровней. На i -м уровне будет 7^i задач размера $n/2^i$, на каждую из которых уйдёт время $c(n/2^i)^2$.

$$\sum_{i=0}^{\log_2 n} 7^i c \left(\frac{n}{2^i}\right)^2 = cn^2 \sum_{i=0}^{\log_2 n} \frac{7^i}{4^i} = n^2 \cdot \Theta\left(\frac{7^{\log_2 n}}{4^{\log_2 n}}\right) = \Theta(n^{\log_2 7})$$

□

Постановка задачи

Сортировка

Вход: массив $A[1 \dots n]$.

Выход: перестановка $A'[1 \dots n]$ элементов массива $A[1 \dots n]$, в которой элементы упорядочены по неубыванию: $A'[1] \leq A'[2] \leq \dots \leq A'[n]$.

Замечания

- Алгоритм имеет доступ к оракулу сравнения. Считаем, что сравнение занимает константное время.
- Если $A = A'$, то алгоритм сортирует **на месте**.

Сортировка вставками

Процедура INSERTIONSORT($A[1 \dots n]$)

для i от 2 до n :
 $j \leftarrow i$
 пока $j > 1$ и $A[j] < A[j - 1]$:
 обменять $A[j]$ и $A[j - 1]$
 $j \leftarrow j - 1$

Корректность и время работы

- Корректность следует из выполнения следующего инварианта: после i итераций внешнего цикла подмассив $A[1 \dots i]$ упорядочен (по неубыванию).
- Время работы: $1 + 2 + \dots + (n - 1) = \Theta(n^2)$.
- Массив сортируется на месте.

61

Сумма арифметической прогрессии

Лемма

$$1 + 2 + \dots + n = \frac{n(n + 1)}{2}$$

62

Сумма арифметической прогрессии

Лемма

$$1 + 2 + \dots + n = \frac{n(n + 1)}{2}$$

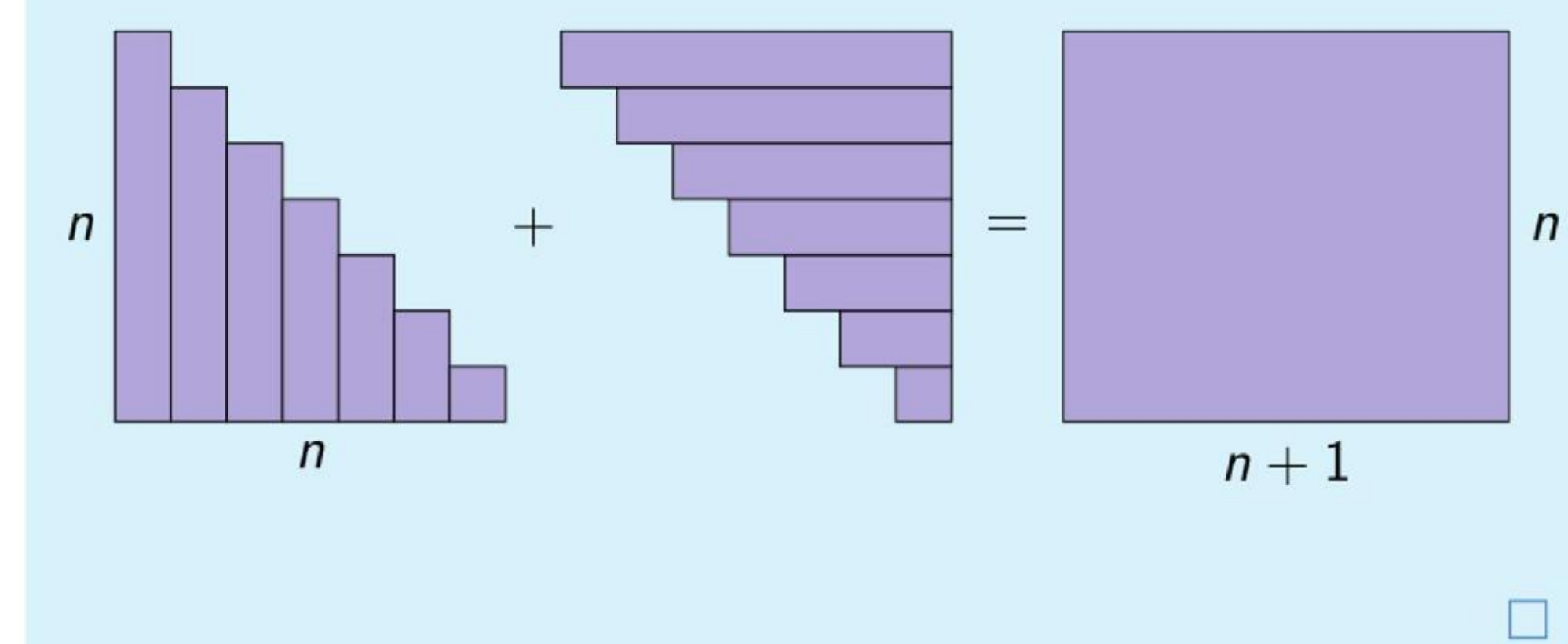
Доказательство

$$\begin{array}{cccccc} 1 & 2 & 3 & \dots & n-1 & n \\ n & n-1 & n-2 & \dots & 2 & 1 \\ \hline n+1 & n+1 & n+1 & \dots & n+1 & n+1 \end{array} = n(n + 1) \quad \square$$

63

Альтернативное доказательство

Доказательство



64

Сортировка слиянием

Процедура MERGESORT(A, ℓ, r)

если $\ell < r$:
 $m \leftarrow \lfloor \frac{\ell+r}{2} \rfloor$
 MERGE(MERGESORT(A, ℓ, m), MERGESORT($A, m+1, r$))

65

Сортировка слиянием

Процедура MERGESORT(A, ℓ, r)

если $\ell < r$:
 $m \leftarrow \lfloor \frac{\ell+r}{2} \rfloor$
 MERGE(MERGESORT(A, ℓ, m), MERGESORT($A, m+1, r$))

Процедура MERGE

- Сликает два упорядоченных массива в один.
- Работает за линейное время от суммы размеров данных двух массивов: первым элементом результирующего массива будет меньший из первых элементов данных массивов, оставшаяся часть может быть заполнена рекурсивно.

66

Время работы

Лемма

Время работы алгоритма сортировки слиянием — $O(n \log n)$.

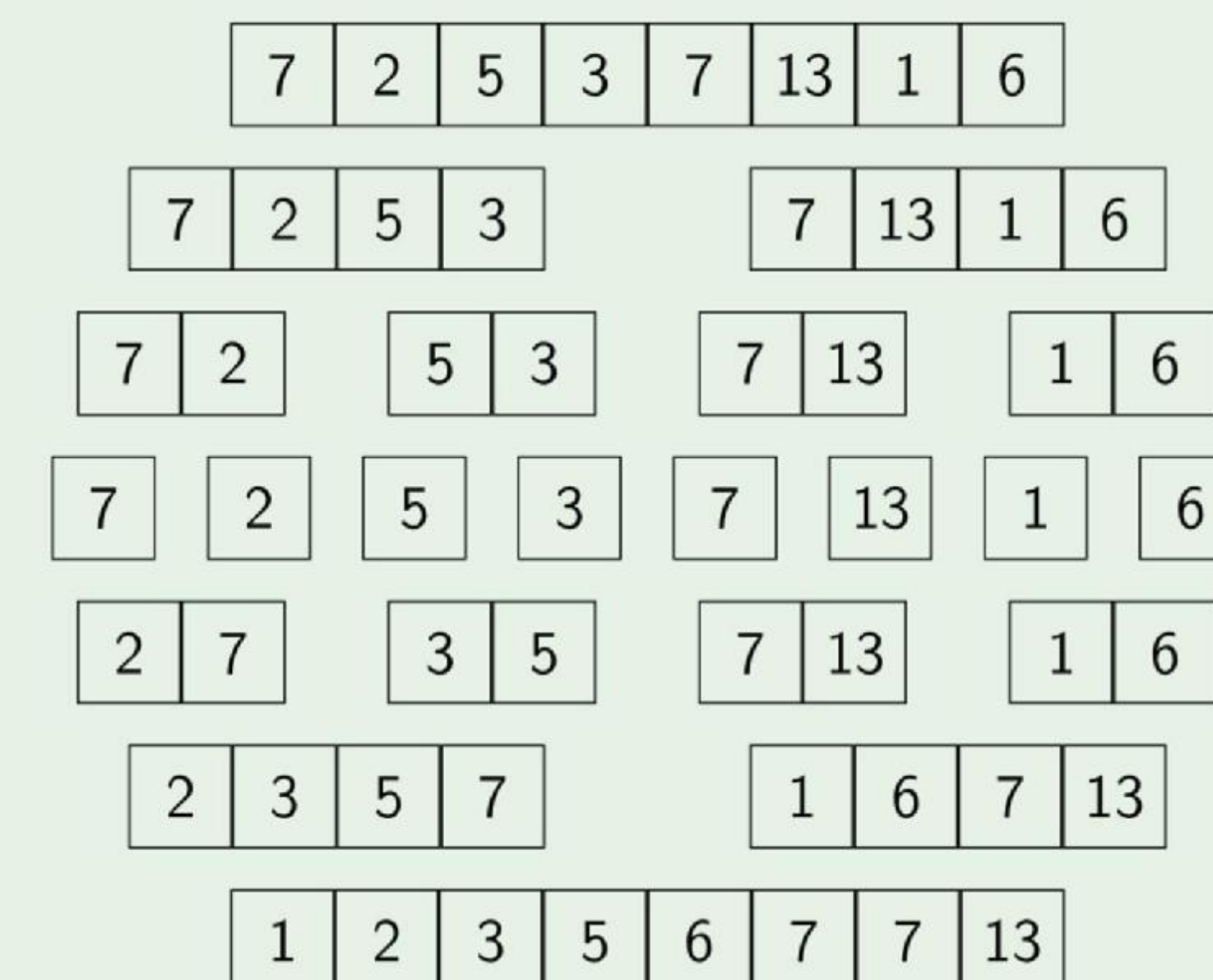
Доказательство

- Рекуррентное соотношение на время работы:
 $T(n) = 2T(n/2) + O(n)$.
- В дереве рекурсии будет $\log_2 n$ уровней. Суммарный размер задач на каждом уровне — n , работа на каждом уровне — $O(n)$.

□

67

Пример



68

Итеративная сортировка слиянием

Функция $\text{ITERATIVEMERGESORT}(A[1 \dots n])$

```

Q ← [] {пустая очередь}
для i от 1 до n:
    PUSHBACK(Q, [A[i]])
пока |Q| > 1:
    PUSHBACK(Q, MERGE(POPFONT(Q), POPFONT(Q)))
вернуть POPFONT(Q)
    
```

69

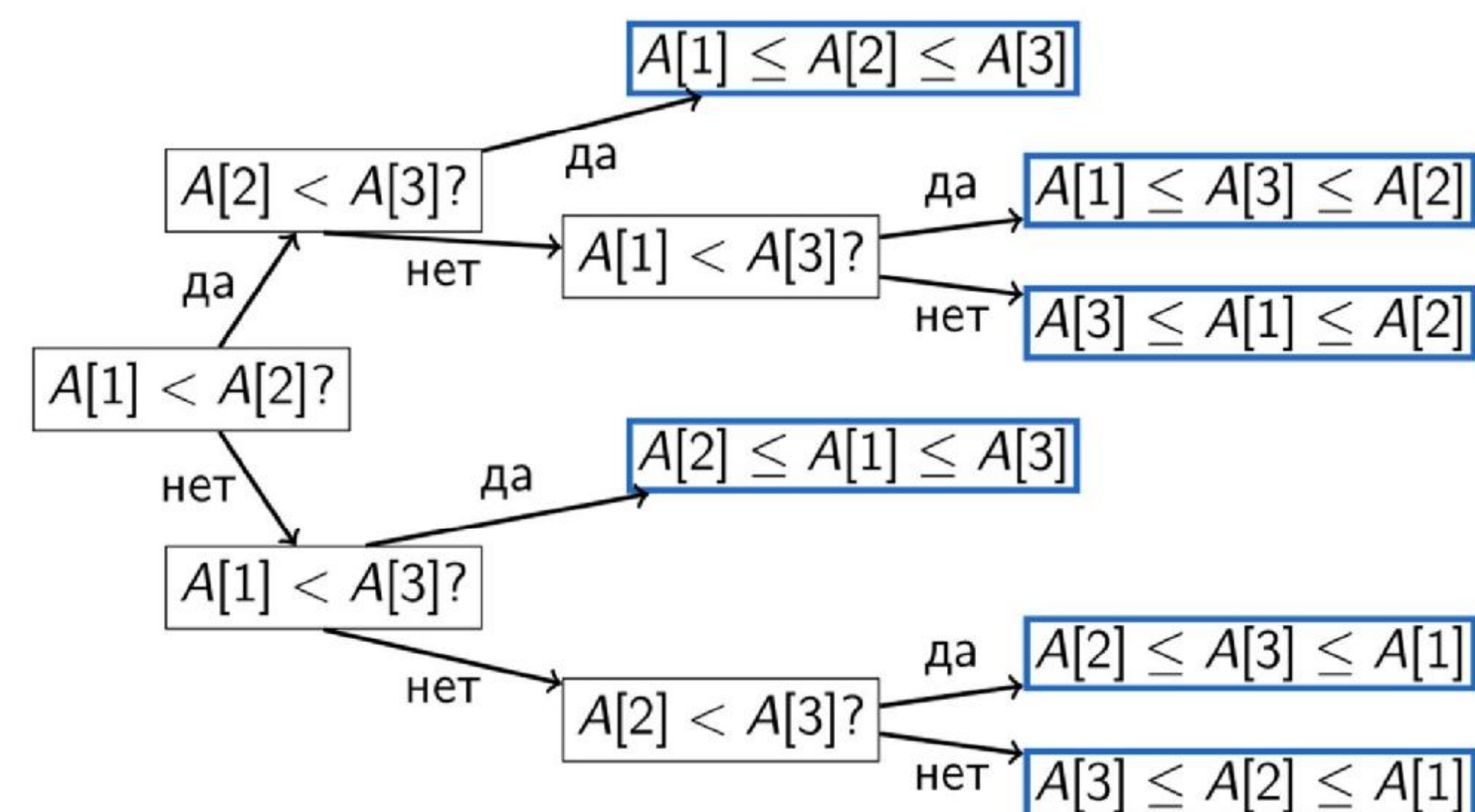
Нижняя оценка для сортировки сравнениями

Лемма

Любой корректный алгоритм сортировки, основанный на сравнениях элементов, делает $\Omega(n \log n)$ сравнений в худшем случае на массиве размера n .

70

Дерево сравнений



71

Доказательство

- число листьев ℓ в дереве $\geq n!$ (число перестановок)

72

Доказательство

- число листьев ℓ в дереве $\geq n!$ (число перестановок)
- время работы алгоритма (число сделанных сравнений) в худшем случае не меньше глубины дерева d

Доказательство

- число листьев ℓ в дереве $\geq n!$ (число перестановок)
- время работы алгоритма (число сделанных сравнений) в худшем случае не меньше глубины дерева d
- $d \geq \log_2 \ell$ (или, что то же самое, $2^d \geq \ell$)

Доказательство

- число листьев ℓ в дереве $\geq n!$ (число перестановок)
- время работы алгоритма (число сделанных сравнений) в худшем случае не меньше глубины дерева d
- $d \geq \log_2 \ell$ (или, что то же самое, $2^d \geq \ell$)
- таким образом, время работы не меньше

$$\log_2(n!) = \Omega(n \log n)$$

Доказательство

- число листьев ℓ в дереве $\geq n!$ (число перестановок)
- время работы алгоритма (число сделанных сравнений) в худшем случае не меньше глубины дерева d
- $d \geq \log_2 \ell$ (или, что то же самое, $2^d \geq \ell$)
- таким образом, время работы не меньше

$$\log_2(n!) = \Omega(n \log n)$$

- формула Стирлинга: $n! \sim \sqrt{2\pi n} \left(\frac{n}{e}\right)^n$

Доказательство

- число листьев ℓ в дереве $\geq n!$ (число перестановок)
- время работы алгоритма (число сделанных сравнений) в худшем случае не меньше глубины дерева d
- $d \geq \log_2 \ell$ (или, что то же самое, $2^d \geq \ell$)
- таким образом, время работы не меньше

$$\log_2(n!) = \Omega(n \log n)$$

- формула Стирлинга: $n! \sim \sqrt{2\pi n} \left(\frac{n}{e}\right)^n$
- $n! > \left(\frac{n}{2}\right)^{n/2}$, поэтому $\log(n!) = \Omega(n \log n)$ □

Заключение

- Наивный алгоритм сортировки имеет время работы $O(n^2)$.
- Алгоритм сортировки слиянием, основанный на методе «разделяй и властвуй», работает за время $O(n \log n)$. Использует $O(n)$ дополнительной памяти (для слияния массивов).
- Любой алгоритм, сортирующий сравнениями, в худшем случае делает $\Omega(n \log n)$ сравнений. Поэтому алгоритм сортировки слиянием асимптотически оптимален.

Задание А. Бинарный поиск

- it
- a_dubatovka
- a_khmlov
- lesson01
- lesson02
- lesson03
 - A_Huffman
 - B_Huffman
 - C_HeapMax
 - dataHuffman.txt
 - encodeHuffman.txt
 - heapData.txt
 - Lesson3Test
- lesson04
 - A_BinaryFind
 - B_MergeSort
 - C_GetInversions
 - dataA.txt
 - dataB.txt
 - dataC.txt
 - Lesson4Test
- alesnax
- apilipenka
- astro_emelya
- belash_ea
- chatovich
- du4
- ededovich.lesson1

```

7
8  /*
9  В первой строке источника данных даны:
10     - целое число 1<=n<=100000 (размер массива)
11     - сам массив A[1...n] из n различных натуральных чисел,
12       не превышающих 10Е9, в порядке возрастания,
13  Во второй строке
14     - целое число 1<=k<=10000 (сколько чисел нужно найти)
15     - k натуральных чисел b1,...,bk не превышающих 10Е9 (сами числа)
16  Для каждого i от 1 до kk необходимо вывести индекс 1<=j<=n,
17    для которого A[j]=bi, или -1, если такого j нет.
18
19  Sample Input:
20  5 1 5 8 12 13
21  5 8 1 23 1 11
22
23  Sample Output:
24  3 1 -1 1 -1
25
26  (!) Обратите внимание на смещение начала индекса массивов JAVA относительно условий задачи
27  */
28
29  public class A_BinaryFind {
30      int[] findIndex(InputStream stream) throws FileNotFoundException {
31          //подготовка к чтению данных

```

Задание Б. Сортировка слиянием

- lesson01
- lesson02
 - A_VideoRegistrator
 - B_Scheduler
 - C_GreedyKnapsack
 - greedyKnapsack.txt
 - Lesson2Test
- lesson03
 - A_Huffman
 - B_Huffman
 - C_HeapMax
 - dataHuffman.txt
 - encodeHuffman.txt
 - heapData.txt
 - Lesson3Test
- lesson04
 - A_BinaryFind
 - B_MergeSort
 - C_GetInversions
 - dataA.txt
 - dataB.txt
 - dataC.txt
 - Lesson4Test
- alesnax
- apilipenka
- astro_emelya
- belash_ea
- chatovich
- du4
- ededovich.lesson1

```

1  package by.it.a_khmlov.lesson04;
2
3  import java.io.FileInputStream;
4  import java.io.FileNotFoundException;
5  import java.io.InputStream;
6  import java.util.Scanner;
7
8  /*
9  Реализуйте сортировку слиянием для одномерного массива.
10  Сложность алгоритма должна быть не хуже, чем O(n log n)
11
12  Первая строка содержит число 1<=n<=10000,
13  вторая - массив A[1...n], содержащий натуральные числа, не превосходящие 10Е9.
14  Необходимо отсортировать полученный массив.
15
16  Sample Input:
17  5
18  2 3 9 2 9
19  Sample Output:
20  2 2 3 9 9
21
22  */
23  public class B_MergeSort {
24
25      int[] getMergeSort(InputStream stream) throws FileNotFoundException {
26          //подготовка к чтению данных

```


Задание С. Сортировка слиянием 2.

lesson02

- A_VideoRegistrar
- B_Scheduler
- C_GreedyKnapsack
- greedyKnapsack.txt
- Lesson2Test

lesson03

- A_Huffman
- B_Huffman
- C_HeapMax
- dataHuffman.txt
- encodeHuffman.txt
- heapData.txt
- Lesson3Test

lesson04

- A_BinaryFind
- B_MergeSort
- C_GetInversions
- dataA.txt
- dataB.txt
- dataC.txt
- Lesson4Test

alesnax

apilipenka

astro_emelya

belash_ea

chatovich

du4

ededovich.lesson1

grishkevich

hustlestar

jahstreet

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

/*

Рассчитать число инверсий одномерного массива.

Сложность алгоритма должна быть не хуже, чем $O(n \log n)$

Первая строка содержит число $1 \leq n \leq 10000$,

вторая – массив $A[1..n]$, содержащий натуральные числа, не превосходящие 10^9 .

Необходимо посчитать число пар индексов $1 \leq i < j \leq n$, для которых $A[i] > A[j]$.

(Такая пара элементов называется инверсией массива.

Количество инверсий в массиве является в некотором смысле

его мерой неупорядоченности: например, в упорядоченном по неубыванию

массиве инверсий нет вообще, а в массиве, упорядоченном по убыванию,

инверсию образуют каждые (т.е. любые) два элемента.

)

Sample Input:

5

2 3 9 2 9

Sample Output:

2

Головоломка (т.е. не обязательно).

Попробуйте обеспечить скорость лучше, чем $O(n \log n)$ за счет многопоточности.

Докажите рост производительности замерами времени.

Большой тестовый массив можно прочитать свой или сгенерировать его программно.

*/