Алгоритмы и структуры данных-1
Лекция 4

Дата: 25.09.2023

Программная инженерия, 2 курс
2023-2024 учебный год

Нестеров Р.А., PhD, ст. преподаватель
департамент программной инженерии ФКН

Еще раз об инварианте цикла

Еще раз об инварианте цикла

```
euclidean(X, Y)
A = X, B = Y
while A ≠ B
    if A > B
        A = A - B
    if B > A
        B = B - A
return A
```

Еще раз об инварианте цикла

```
euclidean(X, Y)
A = X, B = Y
while A ≠ B
    if A > B
        A = A - B
    if B > A
        B = B - A
return A
```

Цель – вычислить **НОД(X, Y)**

Еще раз об инварианте цикла

```
euclidean(X, Y)
A = X, B = Y
while A ≠ B
    if A > B
        A = A - B
    if B > A
        B = B - A
return A
```

Цель – вычислисть **НОД(X, Y)**

1. Переменные ***A*** и ***B*** уменьшаются на каждом шаге цикла в зависимости от их соотношения

Еще раз об инварианте цикла

```
euclidean(X, Y)
A = X, B = Y
while A ≠ B
    if A > B
        A = A - B
    if B > A
        B = B - A
return A
```

Цель – вычислить **НОД(X, Y)**

1. Переменные ***A*** и ***B*** уменьшаются на каждом шаге цикла в зависимости от их соотношения.
2. На выходе из цикла ***A* = *B***.

Еще раз об инварианте цикла

```
euclidean(X, Y)
A = X, B = Y
while A ≠ B
    if A > B
        A = A – B
    if B > A
        B = B – A
return A
```

Цель – вычислить **НОД(X, Y)**

1. Переменные ***A*** и ***B*** уменьшаются на каждом шаге цикла в зависимости от их соотношения.
2. На выходе из цикла ***A* = *B***.

Подойдет ли условие
НОД (X, Y) = НОД(A, B)?

Еще раз об инварианте цикла

```
euclidean(X, Y)
A = X, B = Y
while A ≠ B
    if A > B
        A = A - B
    if B > A
        B = B - A
return A
```

Цель – вычислить **НОД(X, Y)**

Подойдет ли условие
НОД (X, Y) = НОД(A, B)?

1. INIT – обуславливается начальными присваиваниями

Еще раз об инварианте цикла

```
euclidean(X, Y)
A = X, B = Y
while A ≠ B
    if A > B
        A = A - B
    if B > A
        B = B - A
return A
```

Цель – вычислить **НОД(X, Y)**

Подойдет ли условие
НОД (X, Y) = НОД(A, B)?

1. INIT – обуславливается начальными присваиваниями
2. CONT – свойства НОД(A, B) = НОД(B, A) и НОД(A + kB, B) = НОД(A, B)

Еще раз об инварианте цикла

```
euclidean(X, Y)
A = X, B = Y
while A ≠ B
    if A > B
        A = A - B
    if B > A
        B = B - A
return A
```

Цель – вычислить НОД(X, Y)

Подойдет ли условие
НОД (X, Y) = НОД(A, B)?

1. INIT – обуславливается начальными присваиваниями
2. CONT – свойства НОД(A, B) = НОД(B, A) и НОД(A + kB, B) = НОД(A, B)
3. EXIT – НОД(A, A) = A.

Еще раз об инварианте цикла

Поиск – все, что уже просмотрено на текущую итерацию, не содержит искомого элемента

Еще раз об инварианте цикла

Поиск – все, что уже просмотрено на текущую итерацию, не содержит искомого элемента

Сортировка – упорядоченная часть контейнера некоторым образом увеличивается

План

Решение рекуррентных соотношений.
Метод подстановки. Дерево рекурсии.

DaC-алгоритмы для решения задачи умножения квадратных матриц. Метод Штрассена.

Метод подстановки

Решение рекуррентного соотношения

Метод подстановки дает нам универсальный способ для отыскания решения (верхней границы сложности) рекуррентного соотношения

$$T(n) = T(n-1) + 1$$

Решение рекуррентного соотношения

Метод подстановки дает нам универсальный способ для отыскания решения (верхней границы сложности) рекуррентного соотношения

Гипотеза

$$T(n) = T(n-1) + 1 \qquad \boxed{T(n) = O(n^2)}$$

Решение рекуррентного соотношения

Метод подстановки дает нам универсальный способ для отыскания решения (верхней границы сложности) рекуррентного соотношения

Гипотеза

Проверка

$$T(n) = T(n-1) + 1 \quad \boxed{T(n) = O(n^2)} \rightarrow \boxed{\dots}$$

Решение рекуррентного соотношения

Метод подстановки дает нам универсальный способ для отыскания решения (верхней границы сложности) рекуррентного соотношения

Гипотеза

Проверка

$$T(n) = T(n-1) + 1 \quad \boxed{T(n) = O(n^2)} \longrightarrow \boxed{\ \ldots\ }$$

V

✕

Метод подстановки для SELCTION SORT

$$T(n) = T(n-1) + O(n)$$

Метод подстановки для SELCTION SORT

$$T(n) = T(n-1) + O(n)$$

Предположим, что $T(n) = O(n^2)$. Тогда $T(n) \leq cn^2$.

Метод подстановки для SELCTION SORT

$$T(n) = T(n-1) + O(n)$$

Предположим, что $T(n) = O(n^2)$. Тогда $T(n) \leq cn^2$.

Выполним подстановку: $T(n) \leq c(n-1)^2 + O(n)$.

Метод подстановки для SELCTION SORT

Выполним подстановку: $T(n) \leq c(n-1)^2 + O(n)$.

Проверяем:

$$T(n) \leq c(n-1)^2 + c_1 n$$

Метод подстановки для SELCTION SORT

Выполним подстановку: $T(n) \le c(n-1)^2 + O(n)$.

Проверяем:

$$T(n) \le c(n-1)^2 + c_1 n = cn^2 - 2cn + c + c_1 n$$

Метод подстановки для SELCTION SORT

Выполним подстановку: $T(n) \leq c(n-1)^2 + O(n)$.

Проверяем:

$$
\begin{aligned}
T(n) \leq c(n-1)^2 + c_1 n &= cn^2 - 2cn + c + c_1 n \\
&= cn^2 - (2cn - c - c_1 n)
\end{aligned}
$$

Метод подстановки для SELCTION SORT

Выполним подстановку: $T(n) \le c(n-1)^2 + O(n)$.

Проверяем:

$$T(n) \le c(n-1)^2 + c_1 n = cn^2 - 2cn + c + c_1 n$$
$$= cn^2 - (2cn - c - c_1 n) \le cn^2 \ ?$$

Метод подстановки для SELCTION SORT

Выполним подстановку: $T(n) \le c(n-1)^2 + O(n)$.

Проверяем:

$T(n) \le c(n-1)^2 + c_1 n = cn^2 - (2cn - c - c_1 n)$

$$2cn - c - c_1 n \ge 0$$

Метод подстановки для SELCTION SORT

Выполним подстановку: $T(n) \le c(n-1)^2 + O(n)$.

Проверяем:

$$T(n) \le c(n-1)^2 + c_1 n = cn^2 - (2cn - c - c_1 n)$$

$$2cn - c - c_1 n \ge 0$$
$$c_1 n \le c(2n-1)$$
$$c_1 \le c(2 - {}^1\!/_n)$$

Метод подстановки для SELCTION SORT

Выполним подстановку: $T(n) \le c(n-1)^2 + O(n)$.

Проверяем:

$T(n) \le c(n-1)^2 + c_1 n = cn^2 - (2cn - c - c_1 n)$

$$2cn - c - c_1 n \ge 0$$
$$c_1 n \le c(2n-1)$$
$$c_1 \le c(2 - {}^1/_n)$$

Учитывая, что $n \ge 1$, имеем $c_1 \le 2c$.

Метод подстановки для SELCTION SORT

Выполним подстановку: $T(n) \leq c(n-1)^2 + O(n)$.

Проверяем:

$$T(n) \leq c(n-1)^2 + c_1 n = cn^2 - (2cn - c - c_1 n)$$

Пусть $c = 2$ и $c_1 = 3$.

Тогда, $cn^2 - (2cn - c - c_1 n) = 2n^2 - (n-2)$

Метод подстановки для SELCTION SORT

Выполним подстановку: $T(n) \leq c(n-1)^2 + O(n)$.

Проверяем:

$T(n) \leq c(n-1)^2 + c_1 n = cn^2 - (2cn - c - c_1 n)$

Пусть $c = 2$ и $c_1 = 3$.

Тогда, $cn^2 - (2cn - c - c_1 n) = 2n^2 - (n-2) \leq 2n^2$?

Метод подстановки для SELCTION SORT

Выполним подстановку: $T(n) \leq c(n-1)^2 + O(n)$.

Проверяем:

$T(n) \leq c(n-1)^2 + c_1 n = cn^2 - (2cn - c - c_1 n)$

Пусть $c = 2$ и $c_1 = 3$.

Тогда, $cn^2 - (2cn - c - c_1 n) = 2n^2 - (n-2) \leq 2n^2$?

Начиная с $n_0 = 2$, неравенство справедливо.

Метод подстановки для SELCTION SORT

Выполним подстановку: $T(n) \le c(n-1)^2 + O(n)$.

Проверяем:

$$T(n) \le c(n-1)^2 + c_1 n = cn^2 - (2cn - c - c_1 n)$$

Пусть $c = 2$ и $c_1 = 3$.

Тогда, $cn^2 - (2cn - c - c_1 n) = 2n^2 - (n-2) \le 2n^2$?

Начиная с $n_0 = 2$, неравенство справедливо. ■

Метод подстановки для MERGE SORT

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

Метод подстановки для MERGE SORT

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

Предположим, что $T(n) = O(n)$. Тогда $T(n) \leq cn$.

Выполним подстановку: $T(n) \leq {}^{2cn}/_{2} + O(n)$.

Метод подстановки для MERGE SORT

Выполним подстановку: $T(n) \leq 2c\,{}^{n}/_{2} + O(n)$.

Проверяем:

$T(n) \leq cn + c_1 n$

Метод подстановки для MERGE SORT

Выполним подстановку: $T(n) \leq 2c\,{}^{n}/_{2} + O(n)$.

Проверяем:

$T(n) \leq cn + c_1 n \leq cn\ ?$

Метод подстановки для MERGE SORT

Выполним подстановку: $T(n) \leq 2c\,{}^{n}/_{2} + O(n)$.

Проверяем:

$T(n) \leq cn + c_1 n \leq cn\ ?$

Поскольку $c_1, n > 0$, имеем $cn + c_1 n > cn$.

Метод подстановки для MERGE SORT

Выполним подстановку: $T(n) \leq 2c\,{}^{n}/_{2} + O(n)$.

Проверяем:

$T(n) \leq cn + c_1 n \leq cn\,?$

Поскольку $c_1, n > 0$, имеем $cn + c_1 n > cn$.
Следовательно, изначальное предположение неверно!

Метод подстановки для MERGE SORT

Выполним подстановку: $T(n) \le 2c\, {}^{n}\!/_{2} + O(n)$.

Проверяем:

$T(n) \le cn + c_1 n \le cn\ ?$

Поскольку $c_1, n > 0$, имеем $cn + c_1 n > cn$.
Следовательно, изначальное предположение неверно!

Требуется способ для поиска достоверной гипотезы…

Дерево рекурсии

Дерево рекурсии для MERGE SORT

Представляет собой развернутое представление внутреннего устройства рекуррентного соотношения

Каждая вершина дерева рекурсии определяет временную сложность решения подзадачи некоторого размера

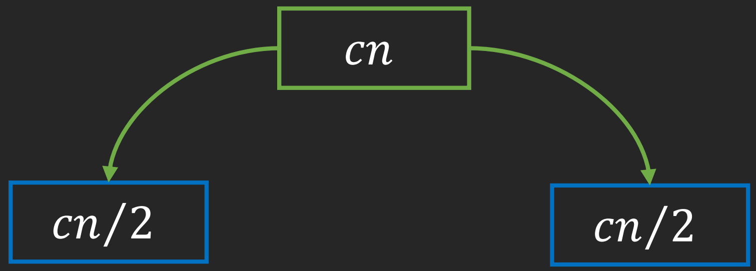
Дерево рекурсии для MERGE SORT

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

$$\boxed{T(n)}$$

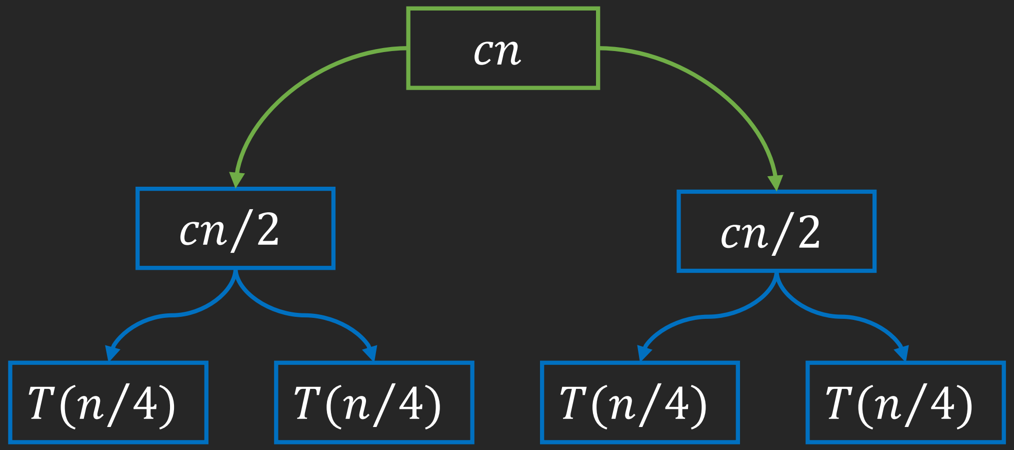
Дерево рекурсии для MERGE SORT

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



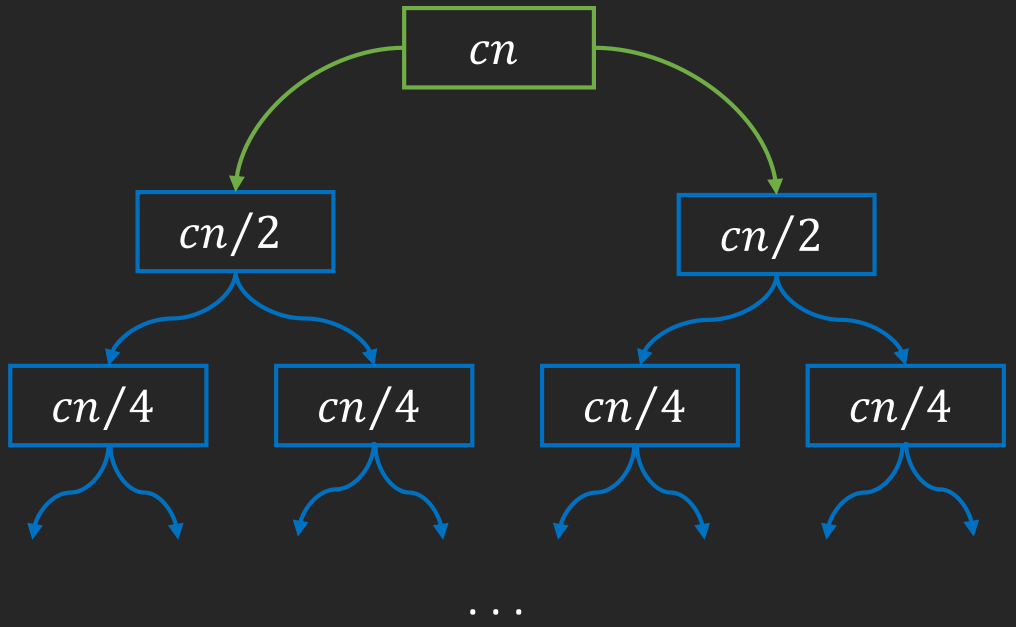
Дерево рекурсии для MERGE SORT

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



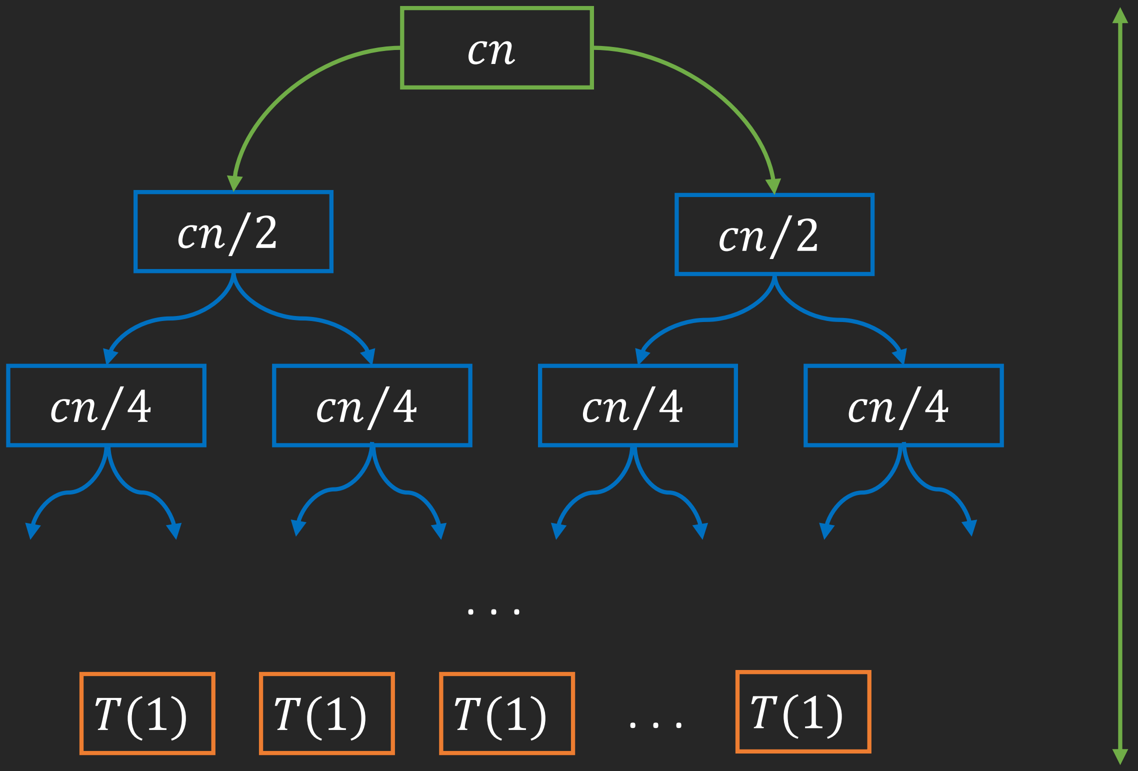
Дерево рекурсии для MERGE SORT

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



Дерево рекурсии для MERGE SORT

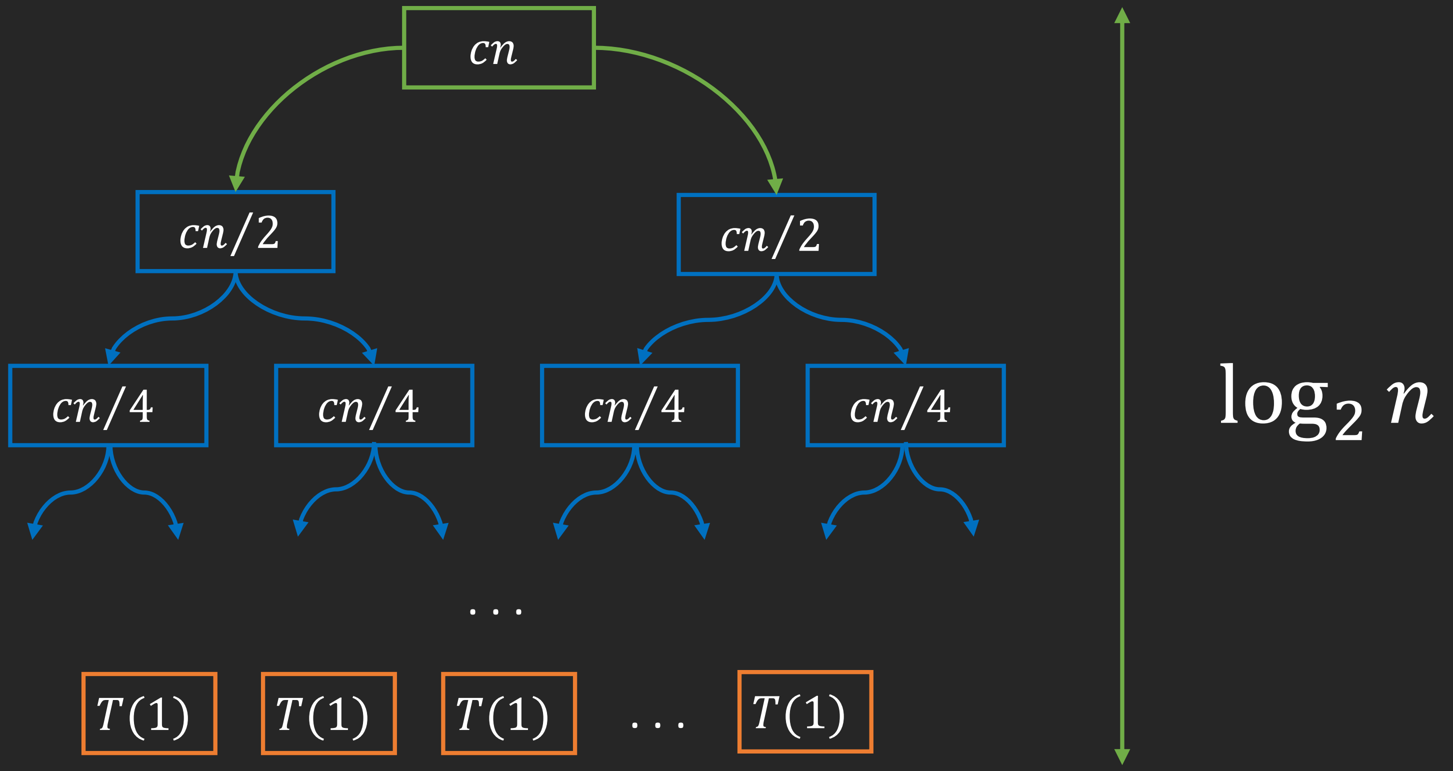
$$T(n) = 2T(n/2) + O(n)$$



Какова высота дерева?

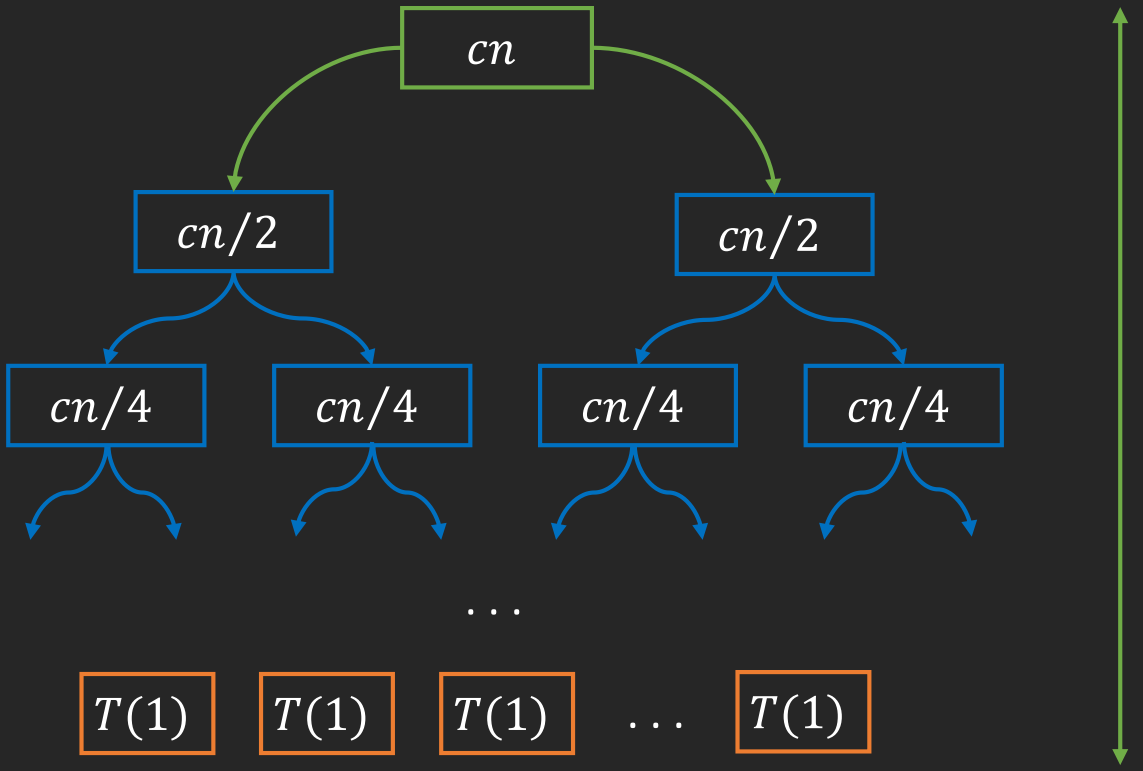
Дерево рекурсии для MERGE SORT

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



Дерево рекурсии для MERGE SORT

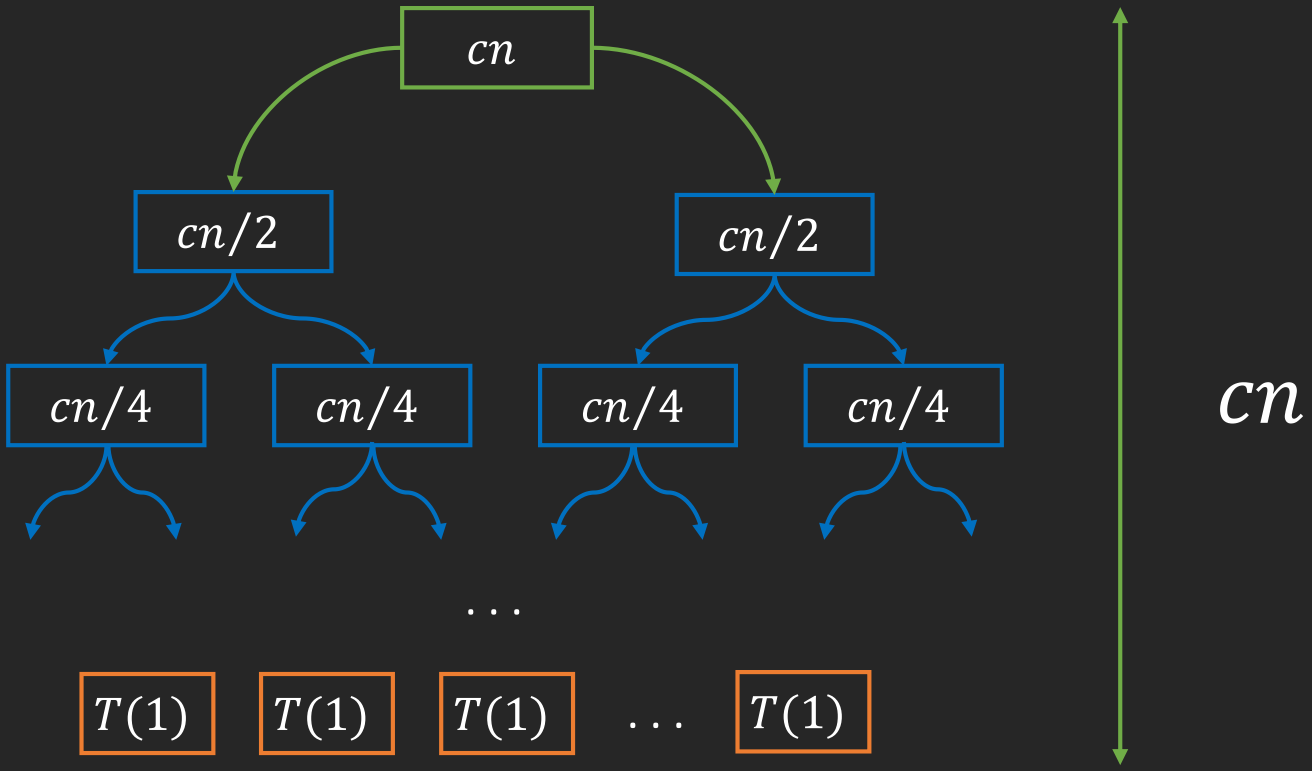
$$T(n) = 2T(n/2) + O(n)$$



Суммарная сложность каждого уровня дерева кроме последнего?

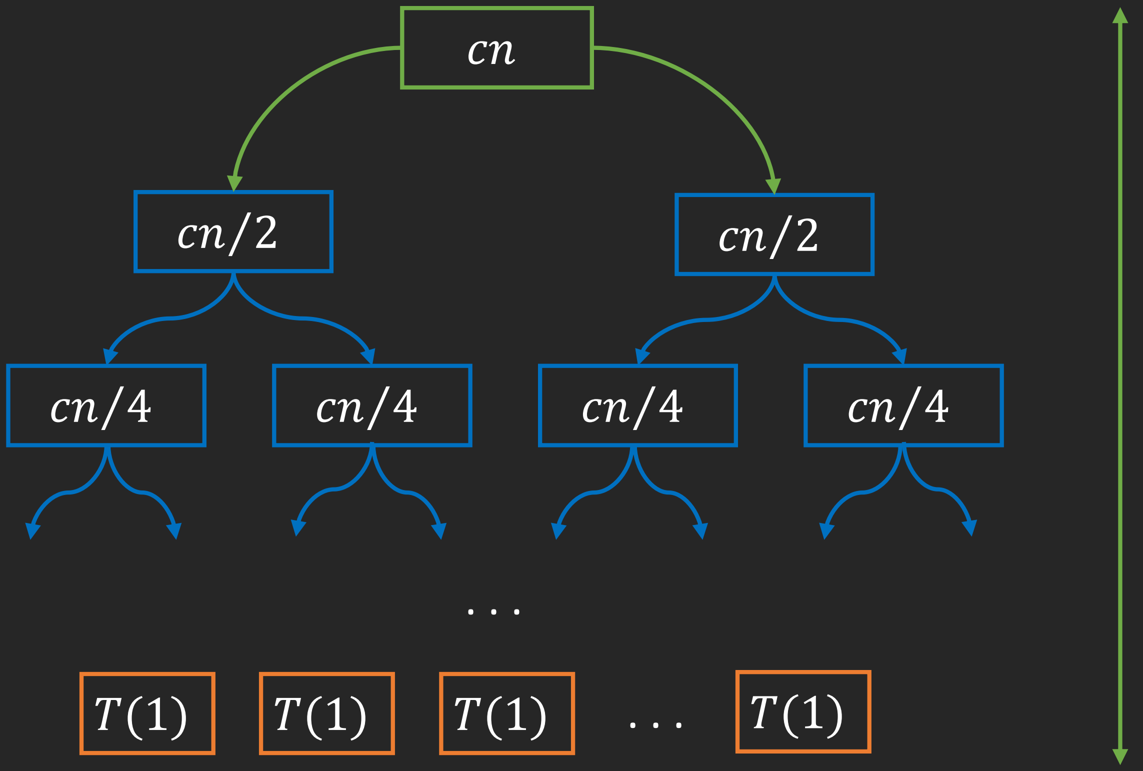
Дерево рекурсии для MERGE SORT

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



Дерево рекурсии для MERGE SORT

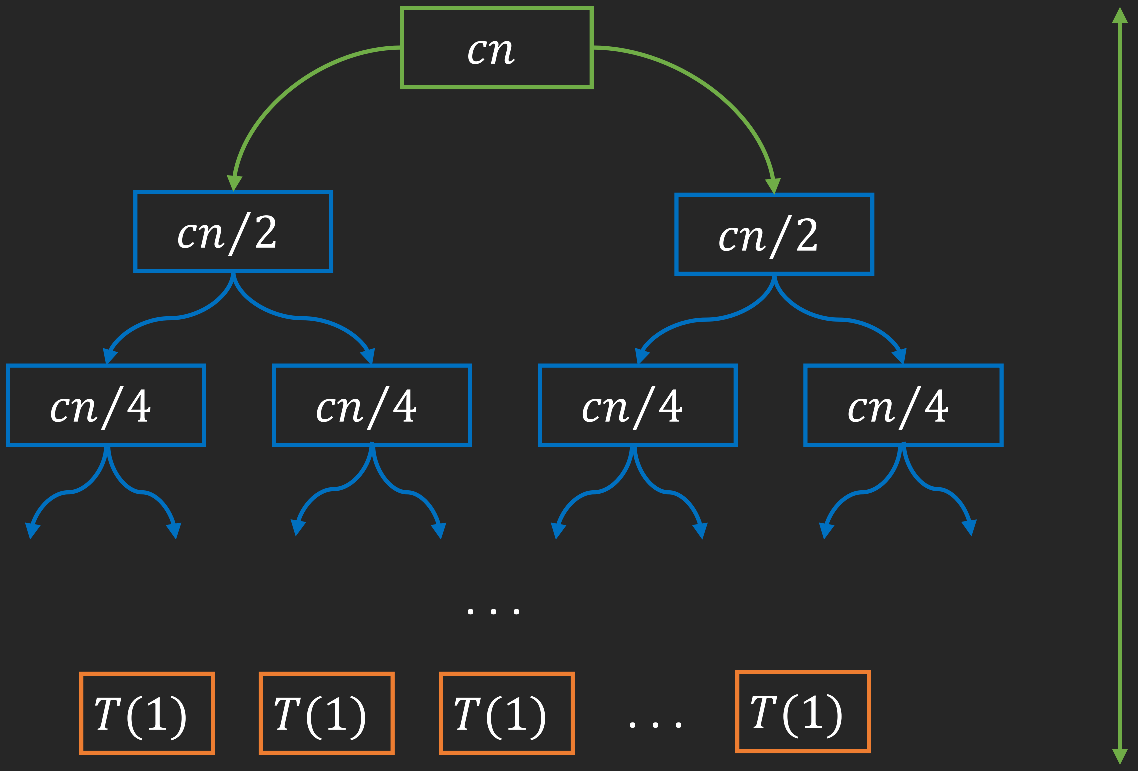
$$T(n) = 2T(n/2) + O(n)$$



Суммарная сложность последнего уровня?

Дерево рекурсии для MERGE SORT

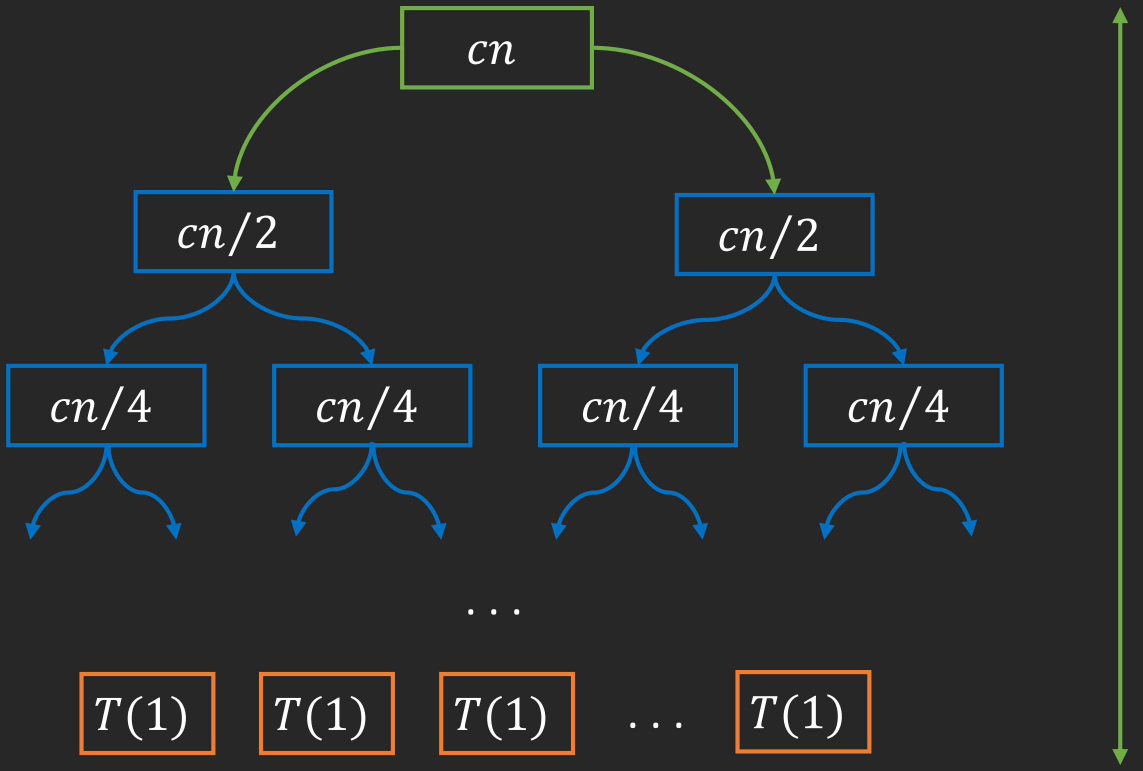
$$T(n) = 2T(n/2) + O(n)$$



$$O(1) \cdot n = O(n)$$

Дерево рекурсии для MERGE SORT

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

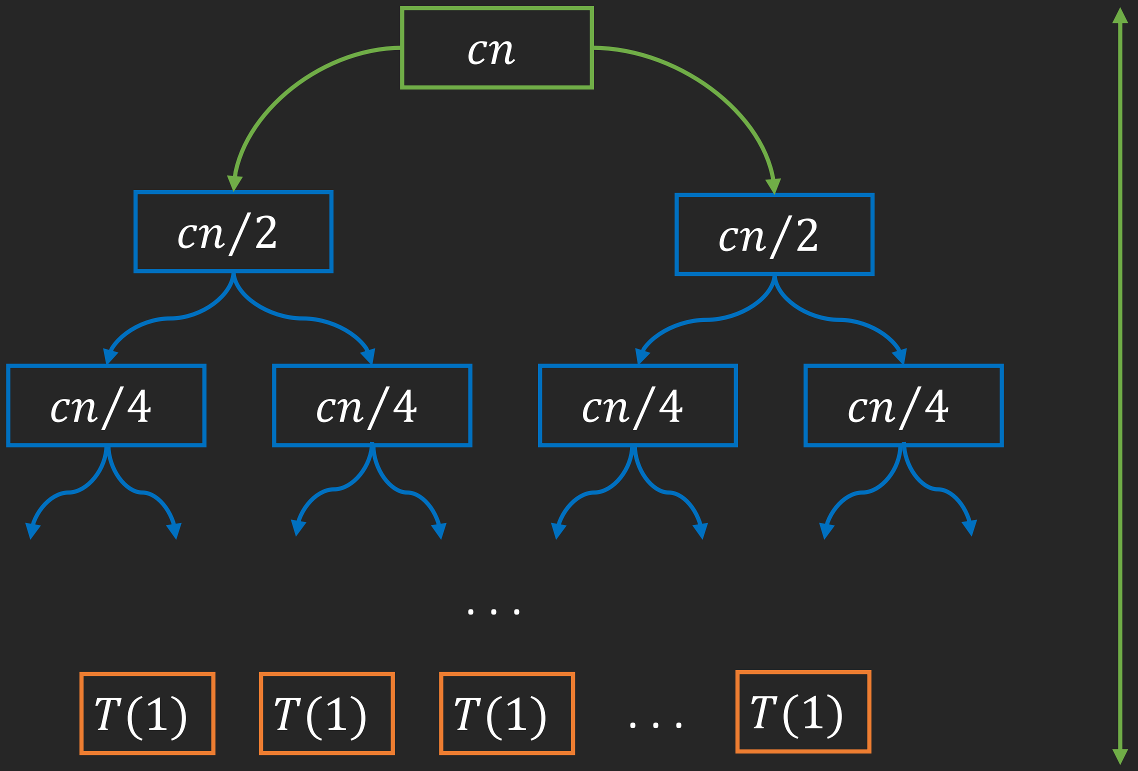


Общая сумма сложностей

$$cn \log_2 n + O(n)$$

Дерево рекурсии для MERGE SORT

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

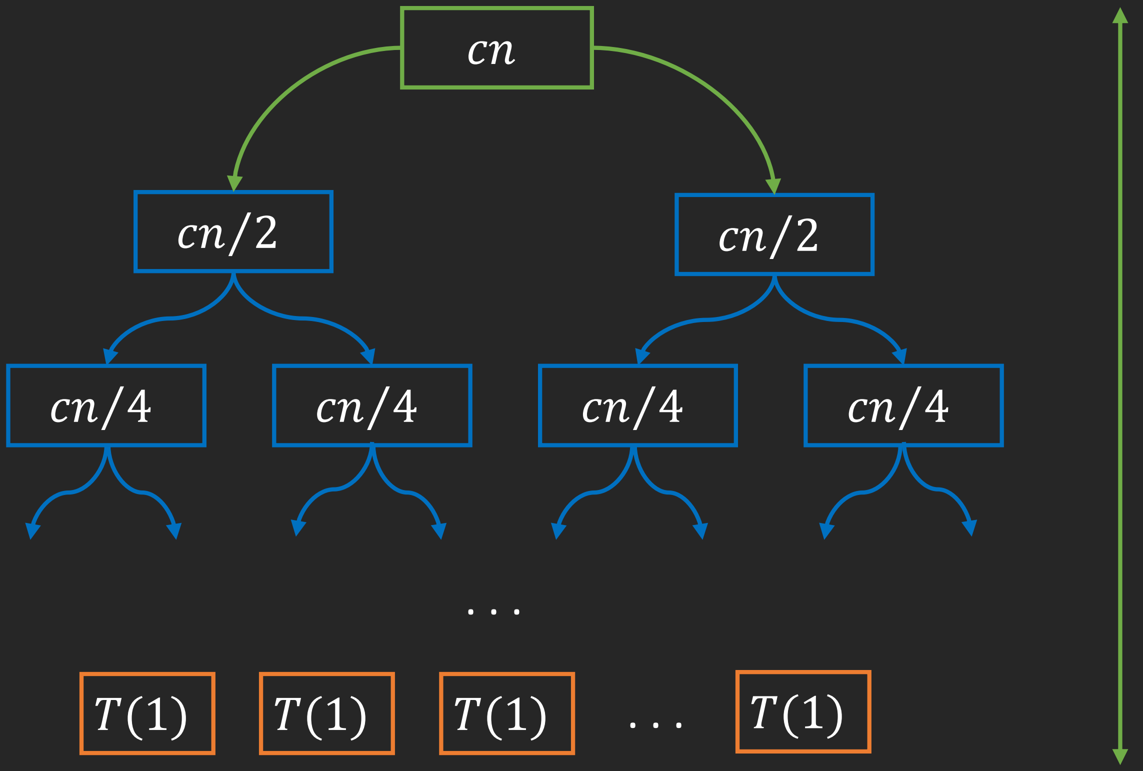


Общая сумма сложностей

$$cn \log_2 n + O(n)$$
$$= O(n \log n)$$

Дерево рекурсии для MERGE SORT

$$T(n) = 2T(n/2) + O(n) = O(n \log n)?$$



Общая сумма сложностей

$$cn \log_2 n + O(n)$$
$$= O(n \log n)$$

Дерево рекурсии для MERGE SORT

$$T(n) = 2T(n/2) + O(n) = O(n \log n)?$$

Выполним подстановку: $T(n) \le {}^{2dn}/_{2} \cdot \log {}^{n}/_{2} + cn$.

Дерево рекурсии для MERGE SORT

$$T(n) = 2T(n/2) + O(n) = O(n \log n)?$$

Выполним подстановку: $T(n) \le dn \cdot \log {}^{n}\!/_{2} + cn$.

Проверяем:

$$T(n) \le dn \log {}^{n}\!/_{2} + cn = dn(\log n - \log 2) + cn$$

Дерево рекурсии для MERGE SORT

$$T(n) = 2T(n/2) + O(n) = O(n \log n)?$$

Выполним подстановку: $T(n) \le dn \cdot \log {}^{n}/_{2} + cn$.

Проверяем:

$$T(n) \le dn \log {}^{n}/_{2} + cn = dn(\log n - \log 2) + cn$$
$$= dn\,(\log n\ - 1) + cn$$

Дерево рекурсии для MERGE SORT

$$T(n) = 2T(n/2) + O(n) = O(n \log n)?$$

Выполним подстановку: $T(n) \le dn \cdot \log {}^{n}/_{2} + cn$.

Проверяем:

$$T(n) \le dn \log {}^{n}/_{2} + cn = dn \log n - (d - c)n$$

Дерево рекурсии для MERGE SORT

$$T(n) = 2T(n/2) + O(n) = O(n\log n)?$$

Выполним подстановку: $T(n) \le dn \cdot \log {}^{n}/_{2} + cn$.

Проверяем:

$$T(n) \le dn\log {}^{n}/_{2} + cn = dn\log n - (d-c)n$$

Дерево рекурсии для MERGE SORT

$$T(n) = 2T(n/2) + O(n) = O(n \log n)?$$

Выполним подстановку: $T(n) \leq dn \cdot \log {}^{n}/_{2} + cn$.

Проверяем:

$$T(n) \leq dn \log {}^{n}/_{2} + cn = dn \log n - (d - c)n$$

$$(d - c)n \geq 0$$

Дерево рекурсии для MERGE SORT

$$T(n) = 2T(n/2) + O(n) = O(n \log n)?$$

Выполним подстановку: $T(n) \le dn \cdot \log {}^{n}/_{2} + cn$.

Проверяем:

$$T(n) \le dn \log {}^{n}/_{2} + cn = dn \log n - (d - c)n$$

$$(d - c)n \ge 0$$

Следовательно, $d \ge c$.

Дерево рекурсии для MERGE SORT

$$T(n) = 2T(n/2) + O(n) = O(n\log n)?$$

Выполним подстановку: $T(n) \le dn \cdot \log {}^{n}/_{2} + cn$.

Проверяем:

$$T(n) \le dn\log {}^{n}/_{2} + cn = dn\log n - (d-c)n$$

$$(d-c)n \ge 0$$

Следовательно, $d \ge c$. ■

DaC-алгоритмы для умножения квадратных матриц

Наивный алгоритм умножения матриц

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \cdot \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} =$$

$$= \begin{pmatrix} a_{11} \cdot b_{11} + a_{12} \cdot b_{21} & a_{11} \cdot b_{12} + a_{12} \cdot b_{22} \\ a_{21} \cdot b_{11} + a_{22} \cdot b_{21} & a_{21} \cdot b_{12} + a_{22} \cdot b_{22} \end{pmatrix}$$

Наивный алгоритм умножения матриц

```
multiply(A, B)
n = A.rows
C – матрица размера n×n
for i = 1 to n
    for j = 1 to n
        c_ij = 0
        for k = 1 to n
            c_ij += a_ik · b_kj
```

Наивный алгоритм умножения матриц

```
multiply(A, B)
n = A.rows
C – матрица размера n×n
for i = 1 to n
    for j = 1 to n
        c_ij = 0
        for k = 1 to n
            c_ij += a_ik · b_kj
```

$O(n^3)$

DaC-алгоритм умножения матриц ver. 1

DIVIDE Каждая из матриц-операндов размера $n \times n$ разбивается на 4 матрицы $n/2 \times n/2$

DaC-алгоритм умножения матриц ver. 1

DIVIDE Каждая из матриц-операндов размера $n \times n$ разбивается на 4 подматрицы $n/2 \times n/2$

CONQUER Рекурсивно умножаем получившиеся подматрицы

DaC-алгоритм умножения матриц ver. 1

DIVIDE Каждая из матриц-операндов размера $n \times n$ разбивается на 4 подматрицы $n/2 \times n/2$

CONQUER Рекурсивно умножаем получившиеся подматрицы

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \cdot \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

```
multiplyDaC(A, B)
n = A.rows
C – матрица размера n×n
if n = 1
    c11 = a11 · b11
else
    разбить A, B, C на подматрицы n/2×n/2
    C11 = multiplyDaC(A11, B11) + multiplyDaC(A12, B21)
    C12 = multiplyDaC(A11, B12) + multiplyDaC(A12, B22)
    C21 = multiplyDaC(A21, B11) + multiplyDaC(A22, B21)
    C22 = multiplyDaC(A21, B12) + multiplyDaC(A22, B22)
```

multiplyDaC(A, B)

n = $A.rows$

C – матрица размера $n \times n$

if n = 1

    $c_{11} = a_{11} \cdot b_{11}$

else

    разбить A, B, C на подматрицы $n/2 \times n/2$

    $C_{11} =$ multiplyDaC(A_{11}, B_{11}) + multiplyDaC(A_{12}, B_{21})

    $C_{12} =$ multiplyDaC(A_{11}, B_{12}) + multiplyDaC(A_{12}, B_{22})

    $C_{21} =$ multiplyDaC(A_{21}, B_{11}) + multiplyDaC(A_{22}, B_{21})

    $C_{22} =$ multiplyDaC(A_{21}, B_{12}) + multiplyDaC(A_{22}, B_{22})

$T(n) = \ldots$

```
multiplyDaC(A, B)
n = A.rows
C – матрица размера n×n
if n = 1
    c11 = a11 · b11
else
    разбить A, B, C на подматрицы n/2×n/2
    C11 = multiplyDaC(A11, B11) + multiplyDaC(A12, B21)
    C12 = multiplyDaC(A11, B12) + multiplyDaC(A12, B22)
    C21 = multiplyDaC(A21, B11) + multiplyDaC(A22, B21)
    C22 = multiplyDaC(A21, B12) + multiplyDaC(A22, B22)
```

$T(n) = \ldots$

$O(1)$

```
multiplyDaC(A, B)
n = A.rows
C – матрица размера n×n
if n = 1
    c11 = a11 · b11
else
    разбить A, B, C на подматрицы n/2×n/2
    C11 = multiplyDaC(A11, B11) + multiplyDaC(A12, B21)
    C12 = multiplyDaC(A11, B12) + multiplyDaC(A12, B22)
    C21 = multiplyDaC(A21, B11) + multiplyDaC(A22, B21)
    C22 = multiplyDaC(A21, B12) + multiplyDaC(A22, B22)
```

$T(n) = \ldots$

$O(1)$

$8T(n/2) + \ldots$

Сложение выделенных подматриц

Каждая из четырех подматриц содержит $n^2/4$ элементов

Сложение выделенных подматриц

Каждая из четырех подматриц содержит $n^2/4$ элементов

Четыре нерекурсивных сложения выполняются соответственно за $O(n^2)$

```
multiplyDaC(A, B)
n = A.rows
C – матрица размера n×n
if n = 1
    c11 = a11 · b11
else
    разбить A, B, C на подматрицы n/2×n/2
    C11 = multiplyDaC(A11, B11) + multiplyDaC(A12, B21)
    C12 = multiplyDaC(A11, B12) + multiplyDaC(A12, B22)
    C21 = multiplyDaC(A21, B11) + multiplyDaC(A22, B21)
    C22 = multiplyDaC(A21, B12) + multiplyDaC(A22, B22)
```

$T(n) = \ldots$

$O(1)$

$8 \cdot T(n/2) + O(n^2)$

```
multiplyDaC(A, B)
n = A.rows
C – матрица размера n×n
if n = 1
    c11 = a11 · b11
else
    разбить A, B, C на подматрицы n/2×n/2
    C11 = multiplyDaC(A11, B11) + multiplyDaC(A12, B21)
    C12 = multiplyDaC(A11, B12) + multiplyDaC(A12, B22)
    C21 = multiplyDaC(A21, B11) + multiplyDaC(A22, B21)
    C22 = multiplyDaC(A21, B12) + multiplyDaC(A22, B22)
```

$T(n) = \ldots$

$O(1)$

???

$8 \cdot T(n/2)$
$+ O(n^2)$

Варианты разбиения матриц

Создание 12 матриц $n/2 \times n/2$ путем копирования элементов из исходных матриц $O(n^2)$

Варианты разбиения матриц

Создание 12 матриц $n/2 \times n/2$ путем копирования элементов из исходных матриц $O(n^2)$

Использование диапазонов индексов строк и столбцов в качестве представления для подматриц $O(1)$

Варианты разбиения матриц

Создание 12 матриц $n/2 \times n/2$ путем копирования элементов из исходных матриц $O(n^2)$

Использование диапазонов индексов строк и столбцов в качестве представления для подматриц $O(1)$

Итоговая временная сложность не зависит от выбора способа разбиения

```
multiplyDaC(A, B)
n = A.rows
C – матрица размера n×n
if n = 1
    c11 = a11 · b11
else
    разбить A, B, C на подматрицы n/2×n/2
    C11 = multiplyDaC(A11, B11) + multiplyDaC(A12, B21)
    C12 = multiplyDaC(A11, B12) + multiplyDaC(A12, B22)
    C21 = multiplyDaC(A21, B11) + multiplyDaC(A22, B21)
    C22 = multiplyDaC(A21, B12) + multiplyDaC(A22, B22)
```

$T(n) = \ldots$

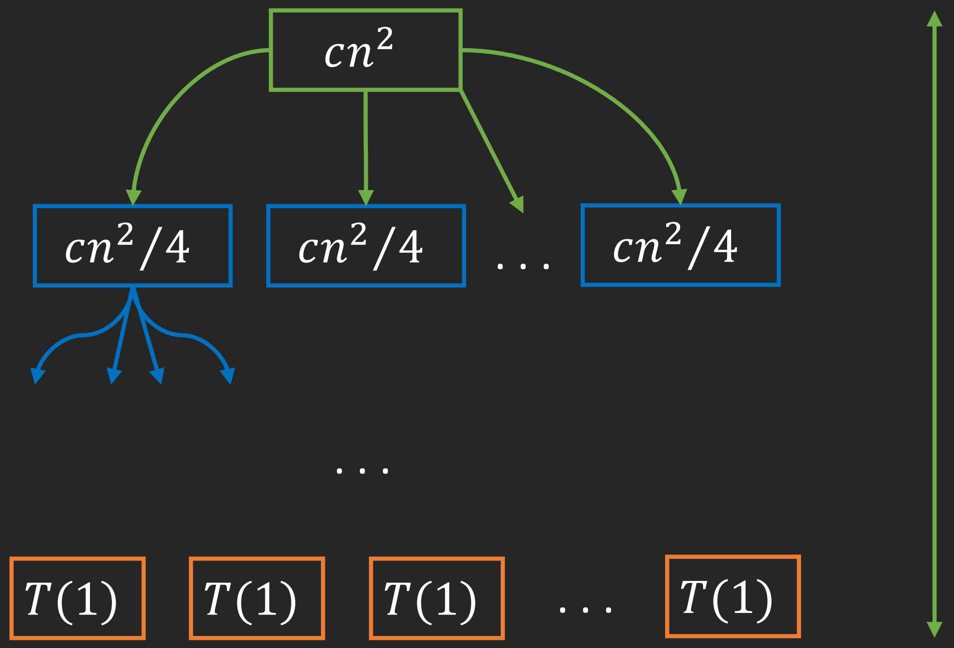
$O(1)$

$O(n^2)$

$8 \cdot T(n/2)$
$+ O(n^2)$

DaC-алгоритм умножения матриц ver. 1

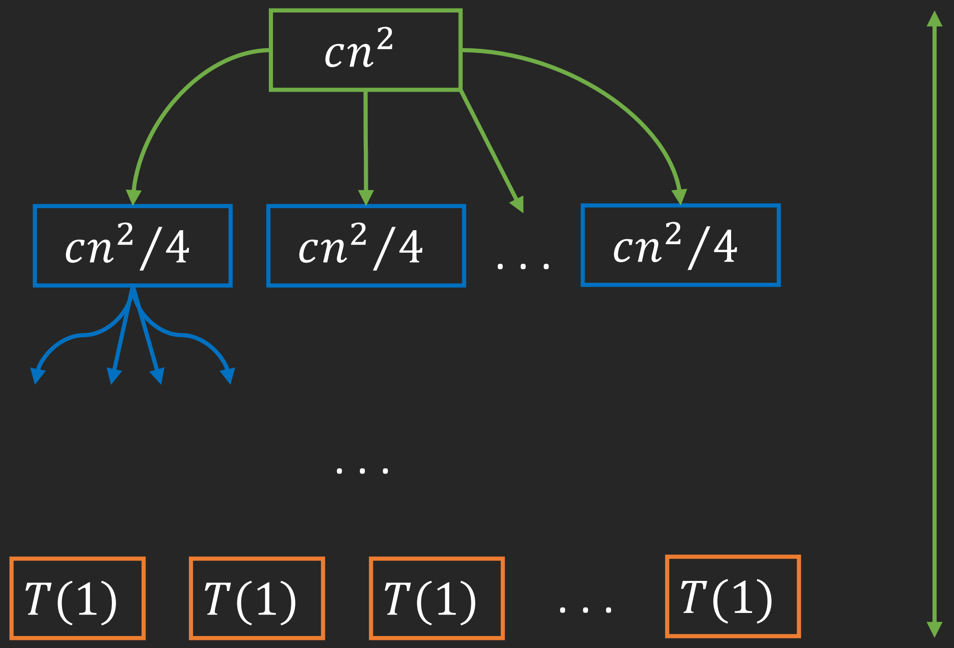
$$T(n) = \begin{cases} O(1) & \text{при } n = 1 \\ 8 \cdot T(n/2) + O(n^2) & \text{при } n > 1 \end{cases}$$



Высота дерева - $\log_2 n$

DaC-алгоритм умножения матриц ver. 1

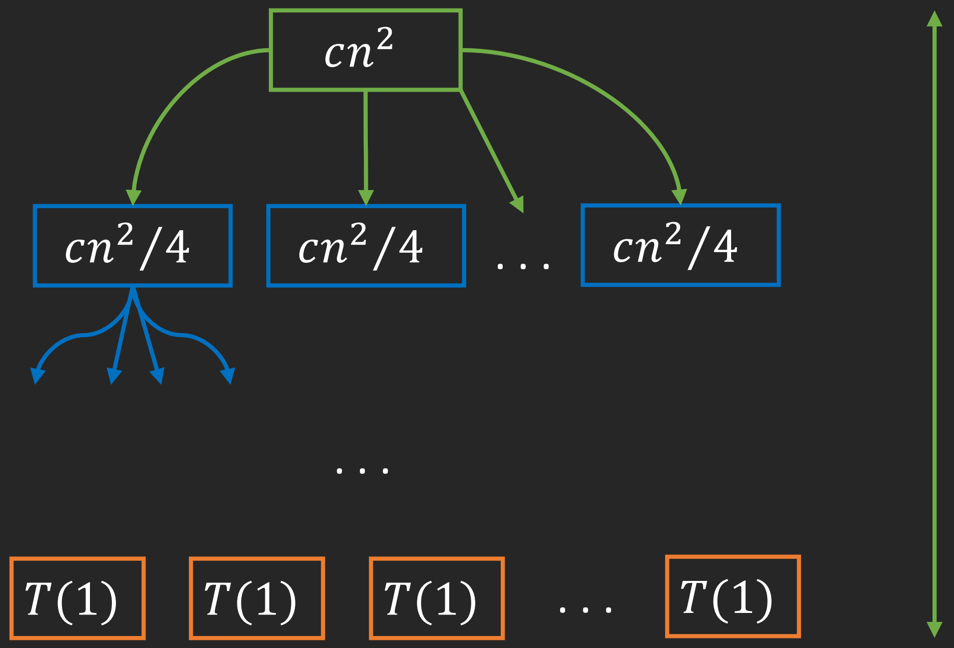
$$T(n) = \begin{cases} O(1) & \text{при } n = 1 \\ 8 \cdot T(n/2) + O(n^2) & \text{при } n > 1 \end{cases}$$



На уровне i - 8^i задач

DaC-алгоритм умножения матриц ver. 1

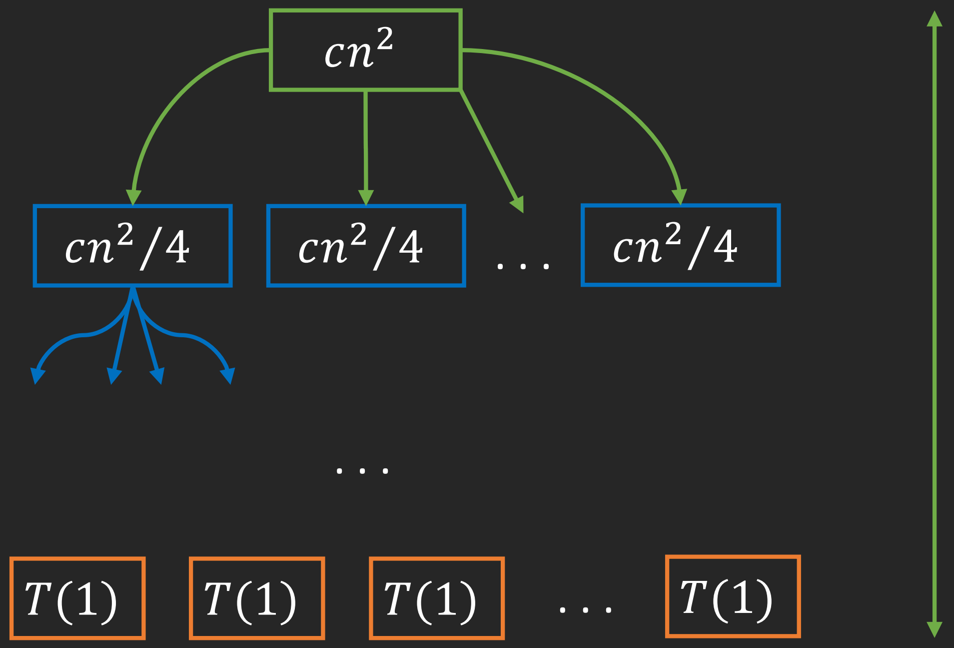
$$T(n) = \begin{cases} O(1) & \text{при } n = 1 \\ 8 \cdot T(n/2) + O(n^2) & \text{при } n > 1 \end{cases}$$



На уровне i - 8^i задач с затратами ${cn^2}/{4^i}$

DaC-алгоритм умножения матриц ver. 1

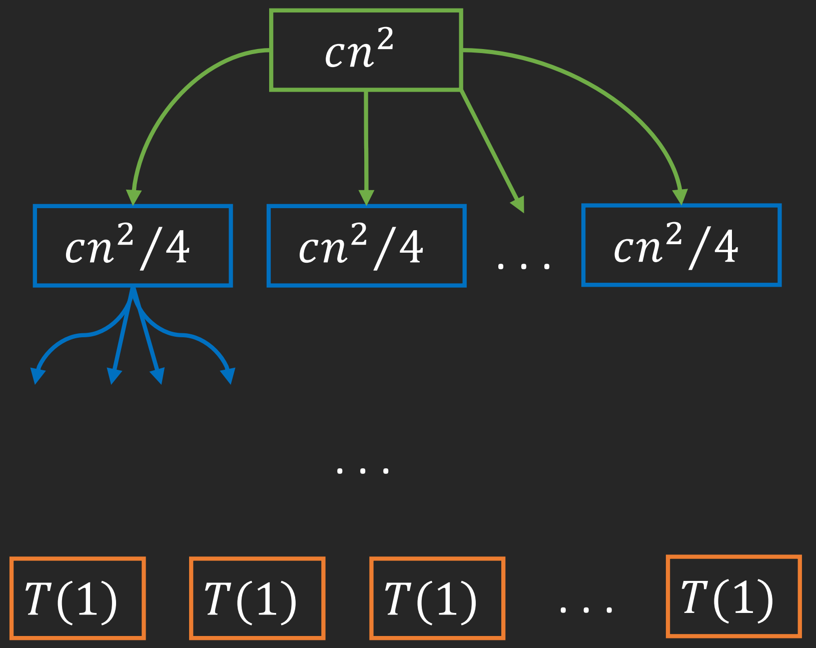
$$T(n) = \begin{cases} O(1) & \text{при } n = 1 \\ 8 \cdot T(n/2) + O(n^2) & \text{при } n > 1 \end{cases}$$



$$\sum_{i=0}^{\log_2 n - 1} \frac{8^i}{4^i} cn^2 + 8^{\log_2 n} O(1)$$

DaC-алгоритм умножения матриц ver. 1

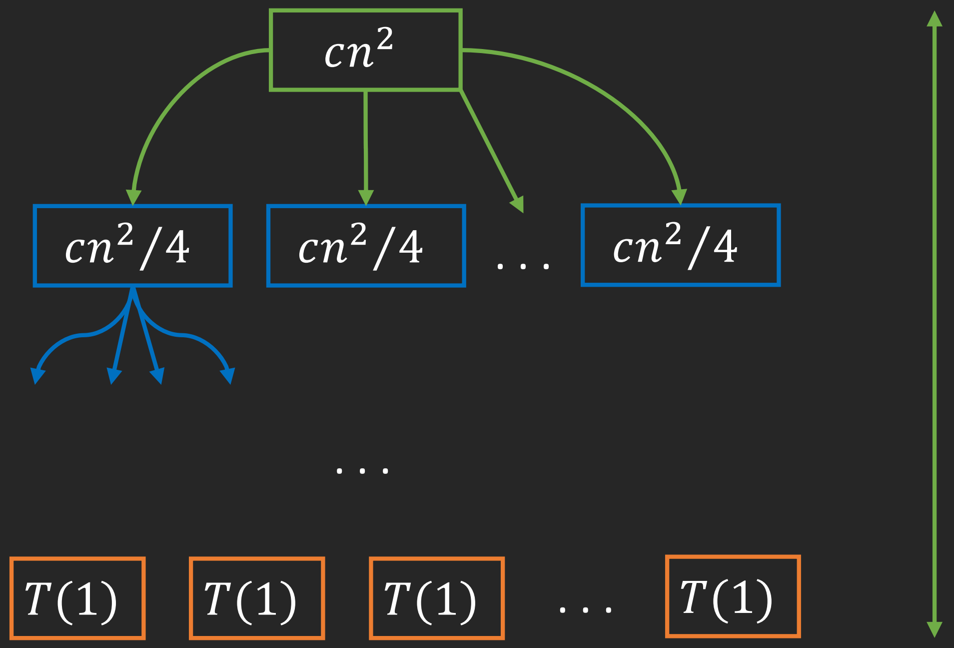
$$T(n) = \begin{cases} O(1) & \text{при } n = 1 \\ 8 \cdot T(n/2) + O(n^2) & \text{при } n > 1 \end{cases}$$



$$\sum_{i=0}^{\log_2 n - 1} 2^i cn^2 + n^3 \cdot O(1)$$

DaC-алгоритм умножения матриц ver. 1

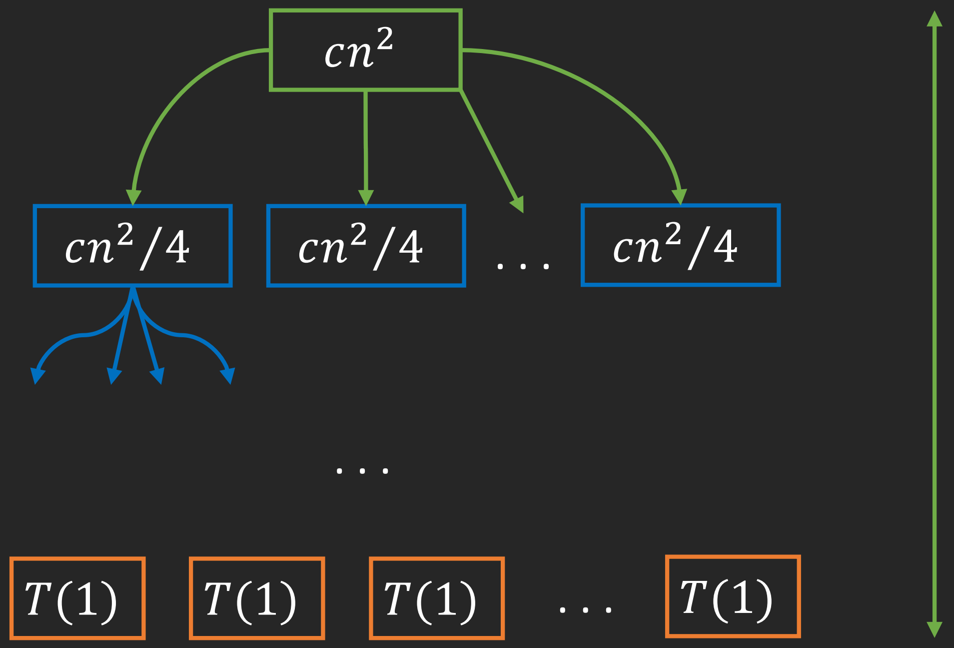
$$T(n) = \begin{cases} O(1) & \text{при } n = 1 \\ 8 \cdot T(n/2) + O(n^2) & \text{при } n > 1 \end{cases}$$



$$\sum_{i=0}^{\log_2 n - 1} 2^i cn^2 + O(n^3)$$

DaC-алгоритм умножения матриц ver. 1

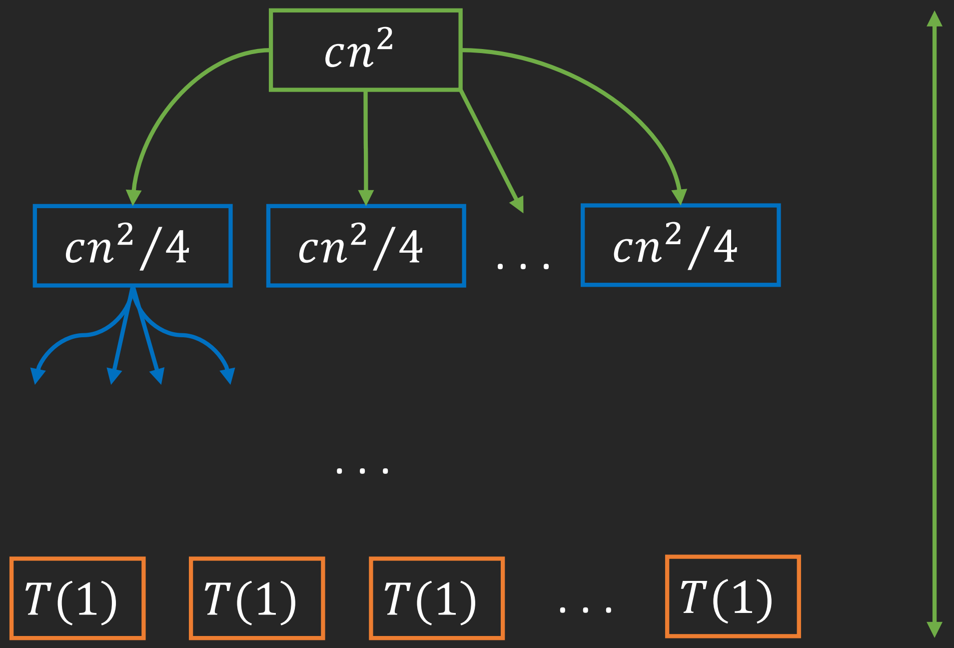
$$T(n) = \begin{cases} O(1) & \text{при } n = 1 \\ 8 \cdot T(n/2) + O(n^2) & \text{при } n > 1 \end{cases}$$



$$\sum_{i=0}^{\log_2 n - 1} 2^i c n^2$$

DaC-алгоритм умножения матриц ver. 1

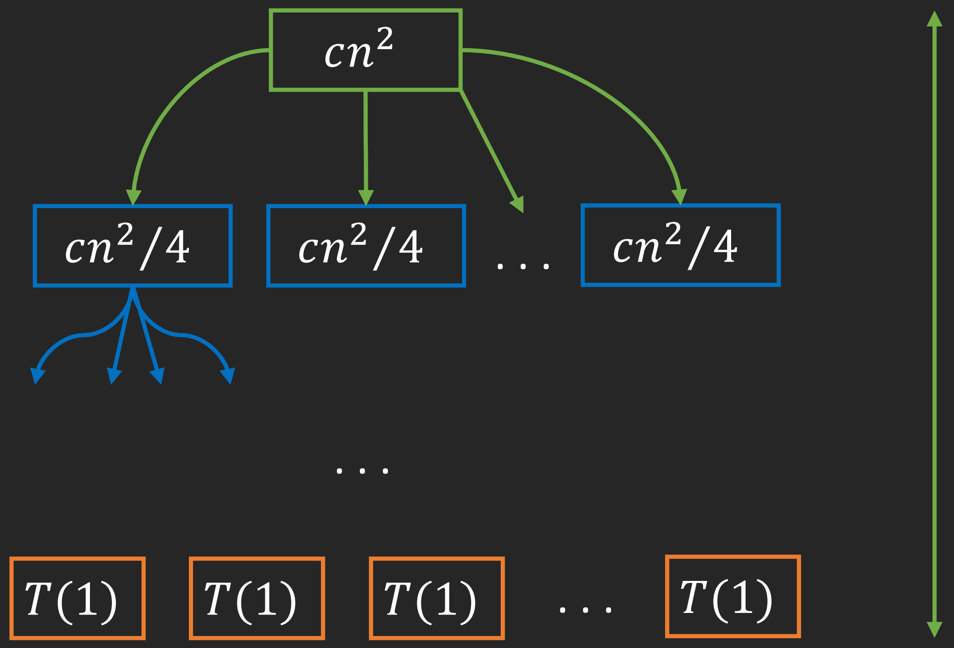
$$T(n) = \begin{cases} O(1) & \text{при } n = 1 \\ 8 \cdot T(n/2) + O(n^2) & \text{при } n > 1 \end{cases}$$



$$\sum_{i=0}^{\log_2 n - 1} 2^i cn^2 = cn^2(1 + 2 + 4 + \cdots + $$
$$+ 2^{\log_2 n - 1})$$

DaC-алгоритм умножения матриц ver. 1

$$T(n) = \begin{cases} O(1) & \text{при } n = 1 \\ 8 \cdot T(n/2) + O(n^2) & \text{при } n > 1 \end{cases}$$

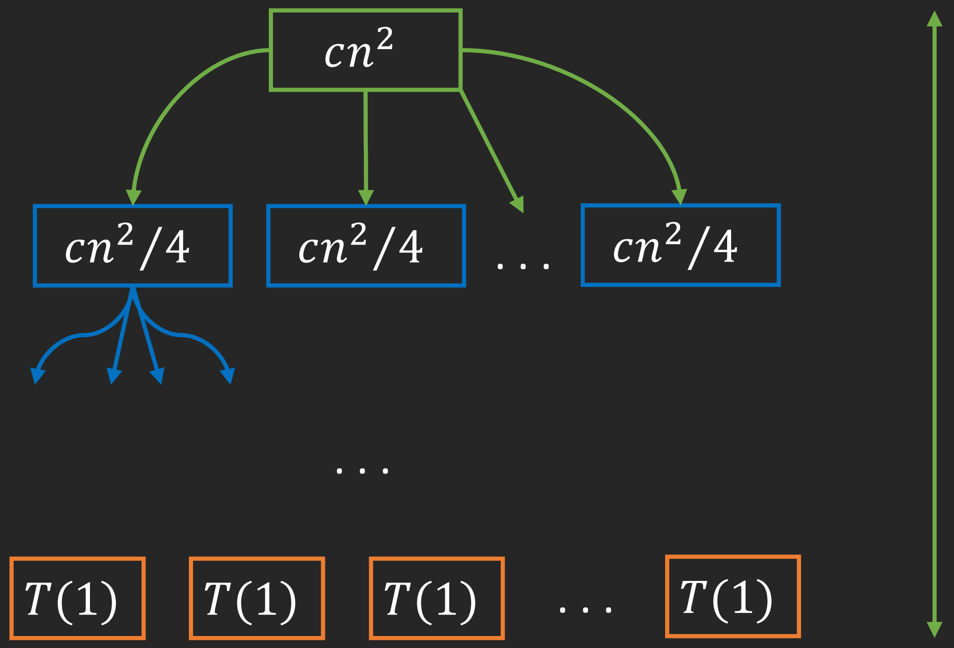


$$\sum_{i=0}^{\log_2 n-1} 2^i cn^2 \le cn^2(1 + 2 + 4 + \cdots +$$

$$+2^{\log_2 n})$$

DaC-алгоритм умножения матриц ver. 1

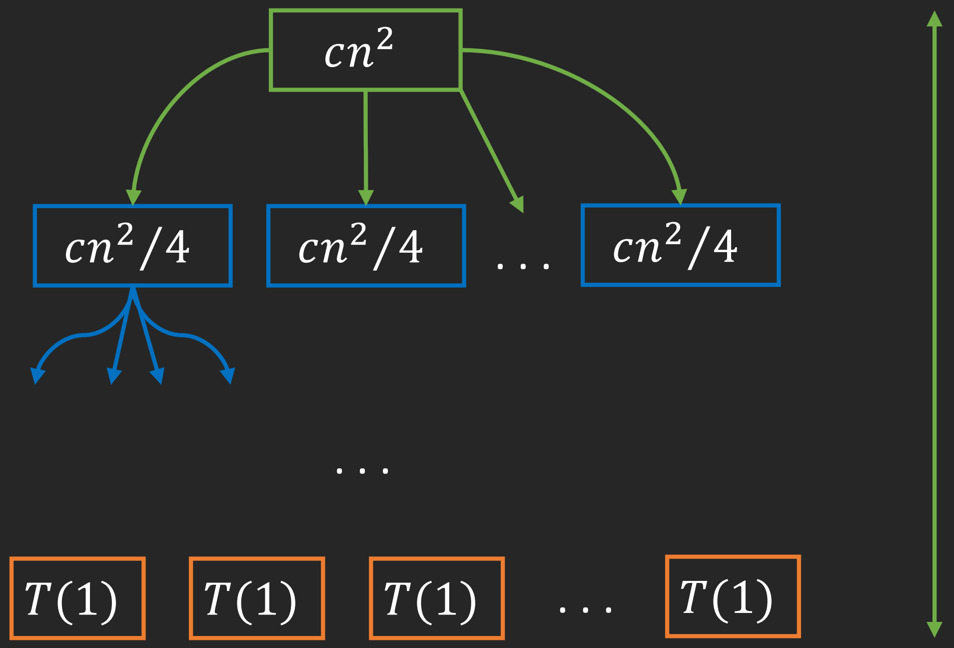
$$T(n) = \begin{cases} O(1) & \text{при } n = 1 \\ 8 \cdot T(n/2) + O(n^2) & \text{при } n > 1 \end{cases}$$



$$\sum_{i=0}^{\log_2 n - 1} 2^i cn^2 \le cn^2(1 + 2 + 4 + \cdots + n)$$

DaC-алгоритм умножения матриц ver. 1

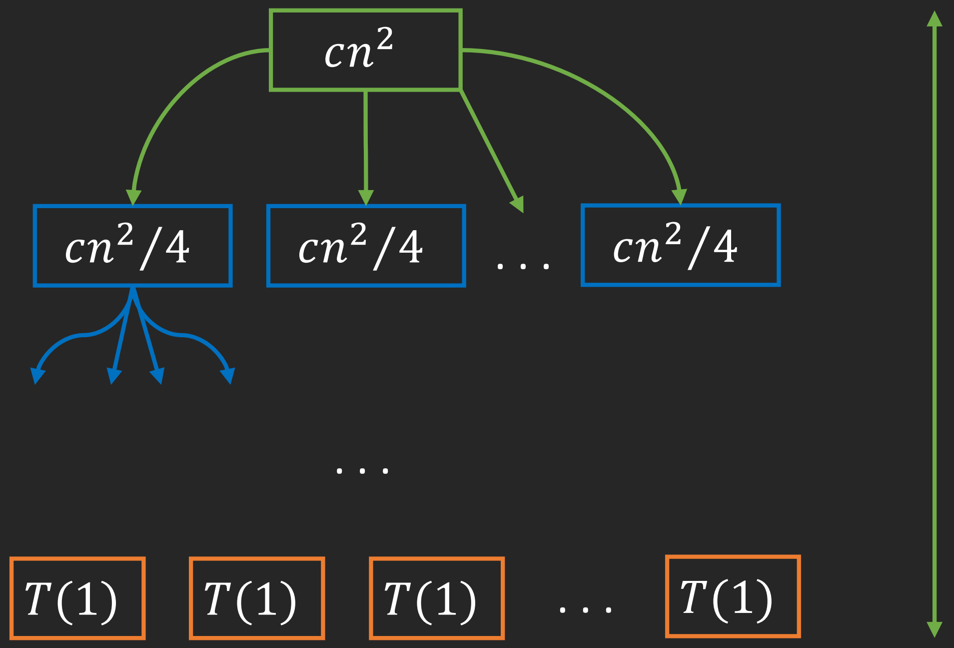
$$T(n) = \begin{cases} O(1) & \text{при } n = 1 \\ 8 \cdot T(n/2) + O(n^2) & \text{при } n > 1 \end{cases}$$



$$\sum_{i=0}^{\log_2 n - 1} 2^i c n^2 \leq c n^2 (2 \cdot n - 1)$$

DaC-алгоритм умножения матриц ver. 1

$$T(n) = \begin{cases} O(1) & \text{при } n = 1 \\ 8 \cdot T(n/2) + O(n^2) & \text{при } n > 1 \end{cases}$$

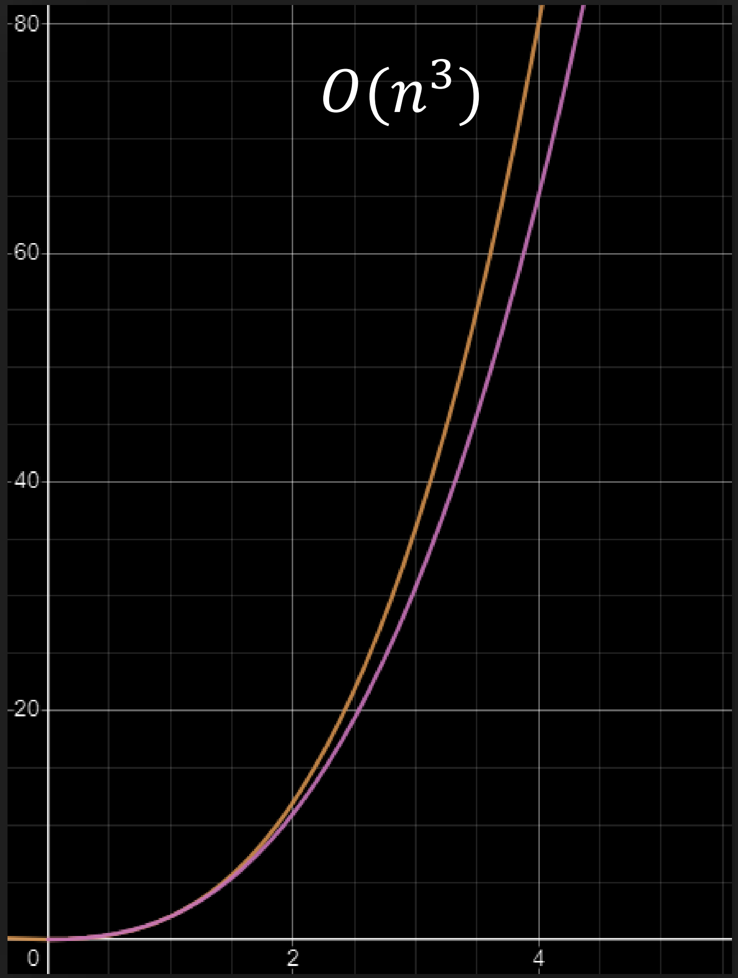


$$\sum_{i=0}^{\log_2 n - 1} 2^i c n^2 = O(n^3)$$

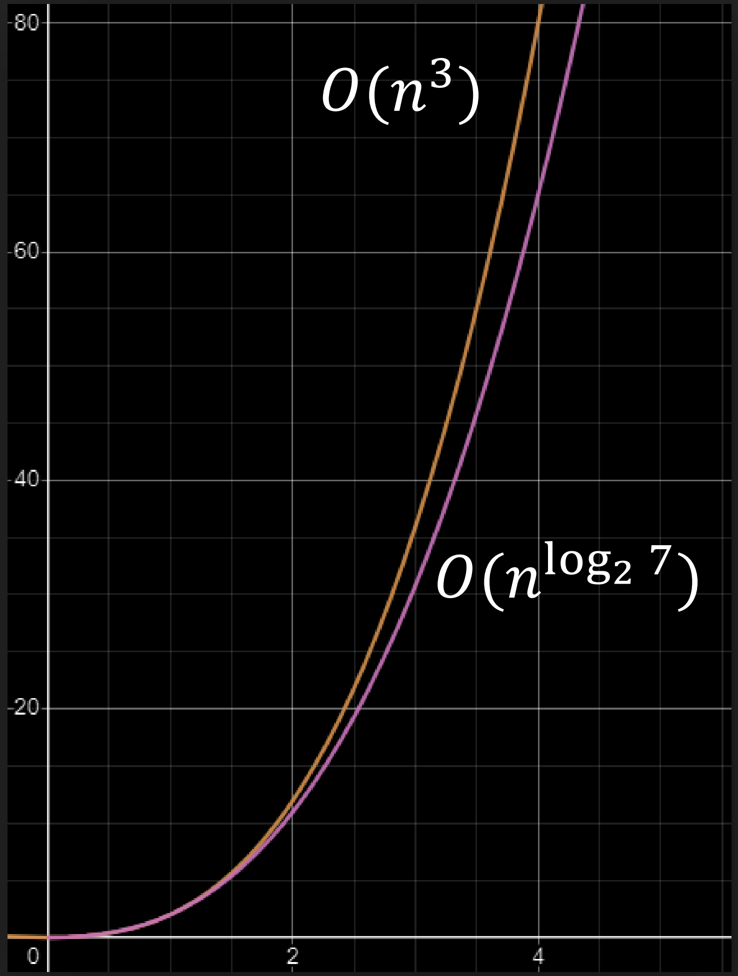
DaC-алгоритм умножения матриц ver. 1

Наивный DaC-подход к умножению квадратных матриц не дает преимущества по сравнению с обычным алгоритмом

DaC-алгоритм умножения матриц ver. 2



DaC-алгоритм умножения матриц ver. 2



Алгоритм Штрассена

$$T(n) = 7T(n/2) + O(n^2)$$

Recap

Метод подстановки и дерево рекурсии
для решения рекуррентных соотношений

Решение задачи умножения квадратных матриц

Teaser – Лекция 5

Основная теорема о рекуррентных соотношениях (master-теорема)

Метод Штрассена для умножения квадратных матриц