Алгоритмы и структуры данных

Master method, алгоритм Штрассена

Divide and Conquer

The Master Method:
(a.k.a. основная теорема о рекуррентных соотношениях)

Если:

$$T(n) \leq a * T(\frac{n}{b}) + O(n^d)$$

Divide and Conquer

The Master Method:
(a.k.a. основная теорема о рекуррентных соотношениях)

Если:

$$T(n) \leq a * T(\frac{n}{b}) + O(n^d)$$

количество подзадач

размер подзадачи

сложность слияния

Divide and Conquer

The Master Method:
(a.k.a. основная теорема о рекуррентных соотношениях)

Если:

$$T(n) \le a * T(\frac{n}{b}) + O(n^d)$$

то,

$$T(n) = \begin{cases} O(n^d * log(n)), & if\ a = b^d \\ O(n^d), & if\ a < b^d \\ O(n^{log_b a}), & if\ a > b^d \end{cases}$$

Если:

$$T(n) \leq a * T(\frac{n}{b}) + O(n^d)$$

то,

$$T(n) = \begin{cases} O(n^d * log(n)), & if\ a = b^d \\ O(n^d), & if\ a < b^d \\ O(n^{log_b a}), & if\ a > b^d \end{cases}$$

Если:

$$T(n) \leq a * T(\tfrac{n}{b}) + O(n^d)$$

то,

$$T(n) = \begin{cases} O(n^d * \boxed{log(n)}), & if\ a = b^d \\ O(n^d), & if\ a < b^d \\ O(n^{\boxed{log_b a}}), & if\ a > b^d \end{cases}$$

Почему в одном случае основание логарифма нам важно, а в другом - нет?

Если:

$$T(n) \leq a * T(\frac{n}{b}) + O(n^d)$$

то,

$$T(n) = \begin{cases} O(n^d * \boxed{log(n)}), & if\ a = b^d \\ O(n^d), & if\ a < b^d \\ O(n^{\boxed{log_b a}}), & if\ a > b^d \end{cases}$$

Почему в одном случае основание логарифма нам важно, а в другом - нет?

Потому, что логарифмы с разными отличаются на константу (см. формулу перехода к новому основанию)

Если:

$$T(n) \leq a * T(\frac{n}{b}) + O(n^d)$$

то,

$$T(n) = \begin{cases} O(n^d * \boxed{log(n)}), & if\ a = b^d \\ O(n^d), & if\ a < b^d \\ O(n^{\boxed{log_b a}}), & if\ a > b^d \end{cases}$$

Почему в одном случае основание логарифма нам важно, а в другом - нет?

Потому, что логарифмы с разными отличаются на константу. Константа как коэффициент у n подавляется O-большим…

Если:

$$T(n) \leq a * T(\frac{n}{b}) + O(n^d)$$

то,

$$T(n) = \begin{cases} O(n^d * \boxed{log(n)}), & if\ a = b^d \\ O(n^d), & if\ a < b^d \\ O(n^{\boxed{log_b a}}), & if\ a > b^d \end{cases}$$

Почему в одном случае основание логарифма нам важно, а в другом - нет?

Потому, что логарифмы с разными отличаются на константу.
Константа как коэффициент у n подавляется O-большим…
Но константа в показателе степени полностью меняет картину!

Если:

$$T(n) \leq a * T(\frac{n}{b}) + O(n^d)$$

то,

$$T(n) = \begin{cases} O(n^d * log(n)), & if\ a = b^d \\ O(n^d), & if\ a < b^d \\ O(n^{log_b a}), & if\ a > b^d \end{cases}$$

Ограничения:

Если:

$$T(n) \leq a * T(\frac{n}{b}) + O(n^d)$$

то,

$$T(n) = \begin{cases} O(n^d * log(n)), & if\ a = b^d \\ O(n^d), & if\ a < b^d \\ O(n^{log_b a}), & if\ a > b^d \end{cases}$$

Ограничения:

1. $T(n) \leq constant$ для достаточно малых n

Если:

$$T(n) \le a * T(\frac{n}{b}) + O(n^d)$$

то,

$$T(n) = \begin{cases} O(n^d * log(n)), & if\ a = b^d \\ O(n^d), & if\ a < b^d \\ O(n^{log_b a}), & if\ a > b^d \end{cases}$$

Ограничения:

1. $T(n) \le constant$ для достаточно малых n

2. При рекурсивном переходе делим на равные подзадачи

Пусть рекурсия такова, что:

$$T(1) \leq c$$

$$T(n) \leq a * T(\frac{n}{b}) + c * n^d$$

(взяли константу c из определения O большого)

Пусть рекурсия такова, что:

$$T(1) \le c$$

$$T(n) \le a * T(\frac{n}{b}) + c * n^d$$

(взяли константу c из определения O большого)

и кроме того: пусть n - это степень b

Пусть рекурсия такова, что:

$$T(1) \leq c$$

$$T(n) \leq a * T(\tfrac{n}{b}) + c * n^d$$

(взяли константу c из определения O большого)

и кроме того: пусть n - это степень b

Получили чуть более грубое утверждение, но по смыслу не отличается от общего случая, лишь немного упрощает доказательство.

Пусть рекурсия такова, что:

$$T(1) \leq c$$

$$T(n) \leq a * T(\frac{n}{b}) + c * n^d$$

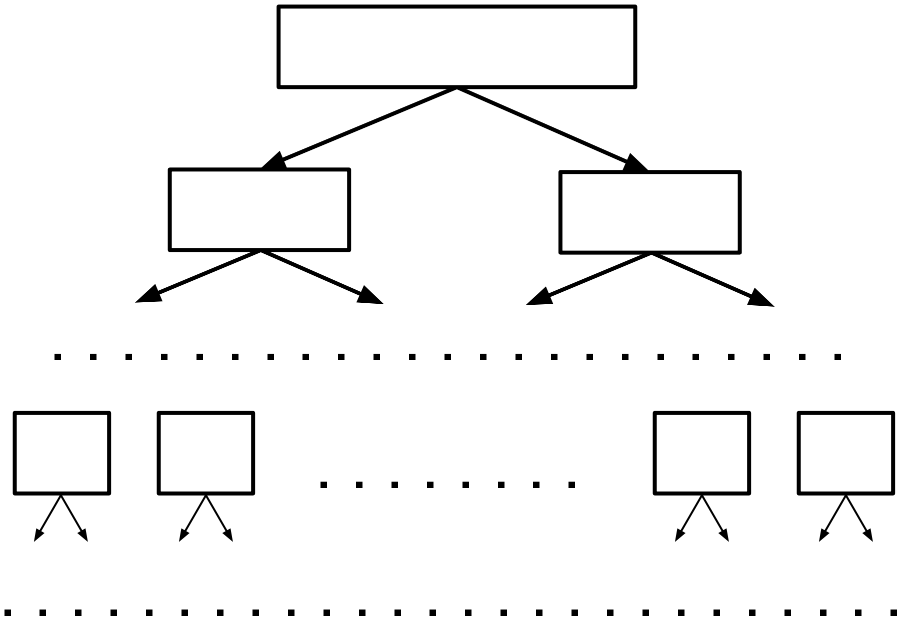
(взяли константу c из определения O большого)

и кроме того: пусть n - это степень b

Получили чуть более грубое утверждение, но по смыслу не отличается от общего случая, лишь немного упрощает доказательство.

А дальше просто будем обобщать доказательство сортировки слиянием!

Сортировка слиянием



уровень 0

уровень 1

уровень j

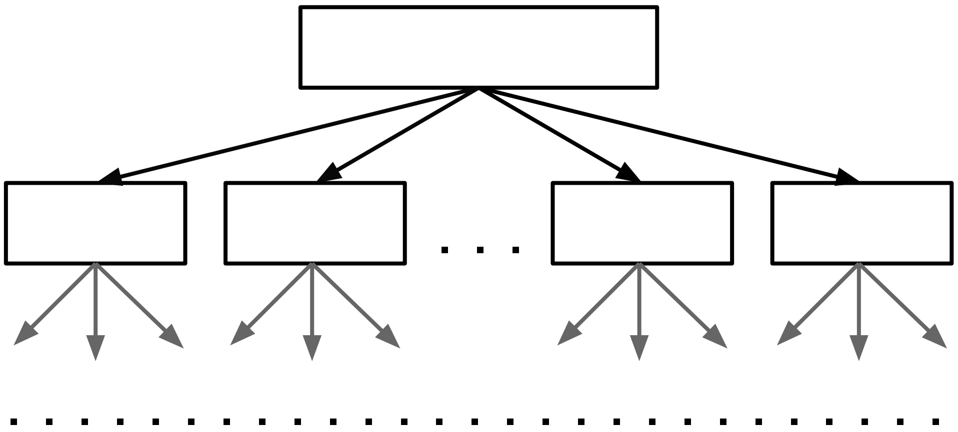
Сколько разных массивов на уровне j? $\quad 2^j$

Размер массива на уровне j? $\quad \frac{n}{2^j}$

Количество операций для слияния на уровне j?

$$T_j(n) \leq 2^j * (C * (\frac{n}{2^j}))$$

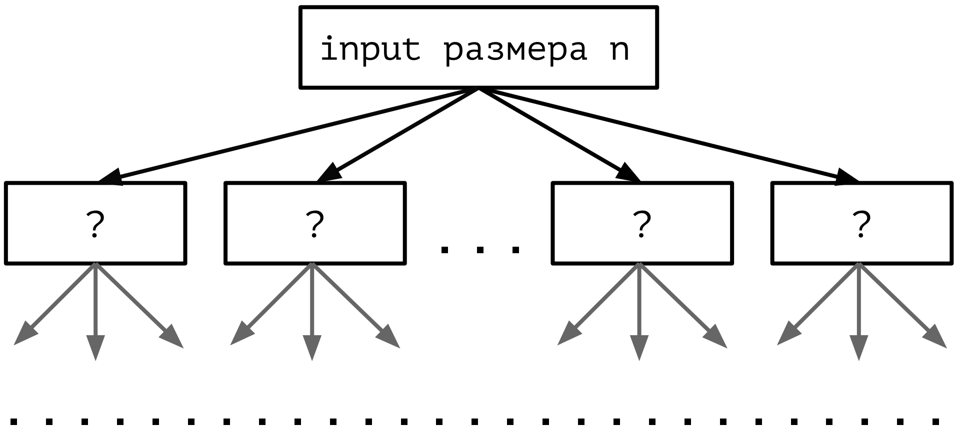
Обобщенный D&C алгоритм $T(n) \leq a * T(\frac{n}{b}) + c * n^d$



уровень 0

уровень 1

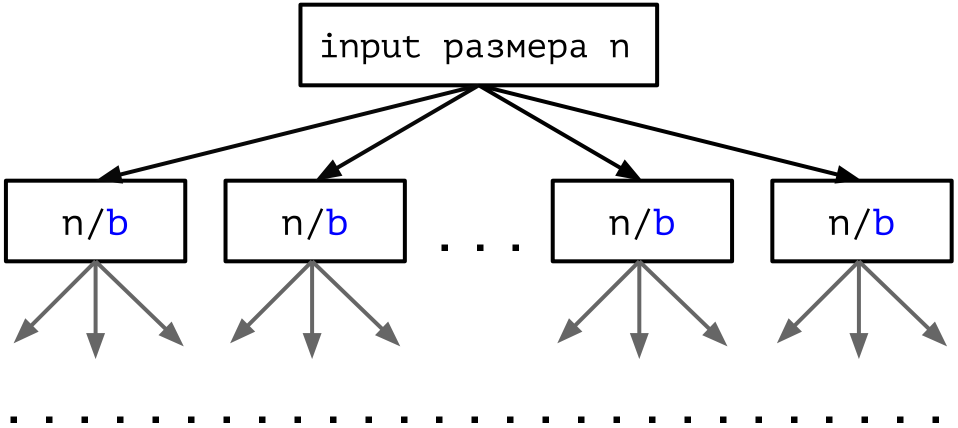
Обобщенный D&C алгоритм $T(n) \leq a * T(\frac{n}{b}) + c * n^d$



уровень 0

уровень 1

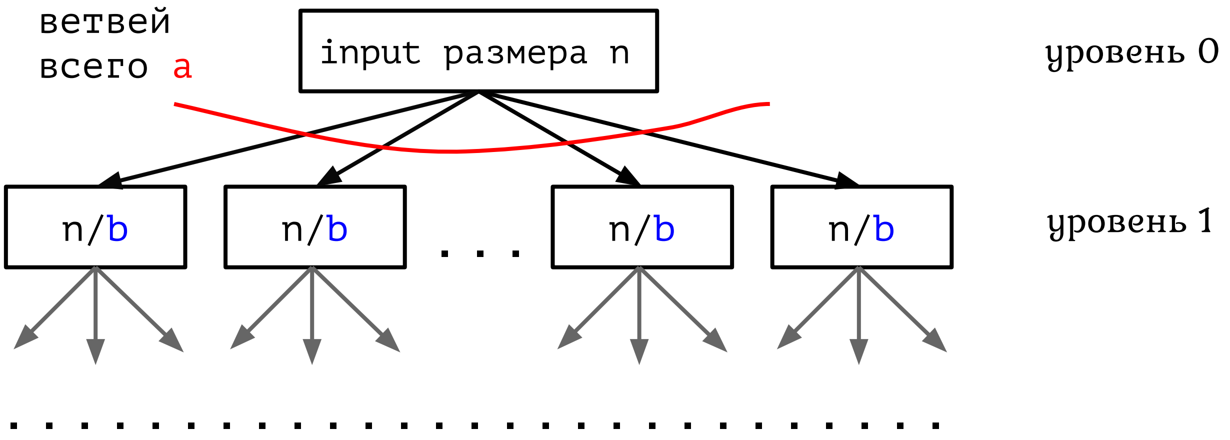
Обобщенный D&C алгоритм $T(n) \leq a * T(\frac{n}{b}) + c * n^d$



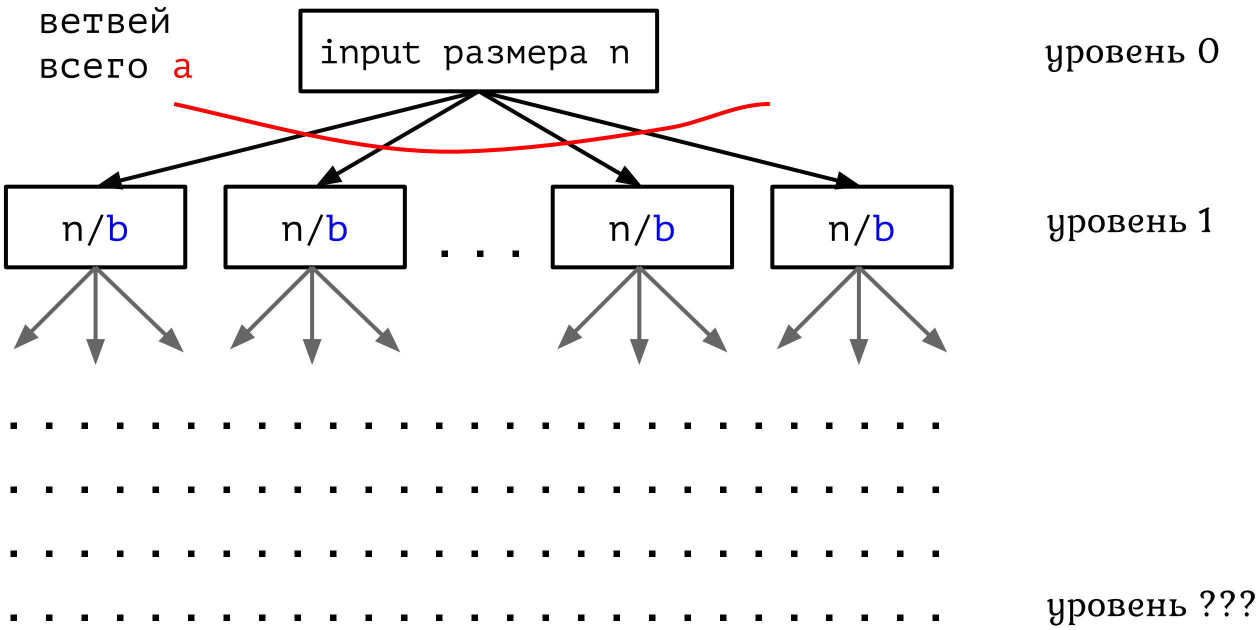
уровень 0

уровень 1

Обобщенный D&C алгоритм $T(n) \leq a * T(\frac{n}{b}) + c * n^d$

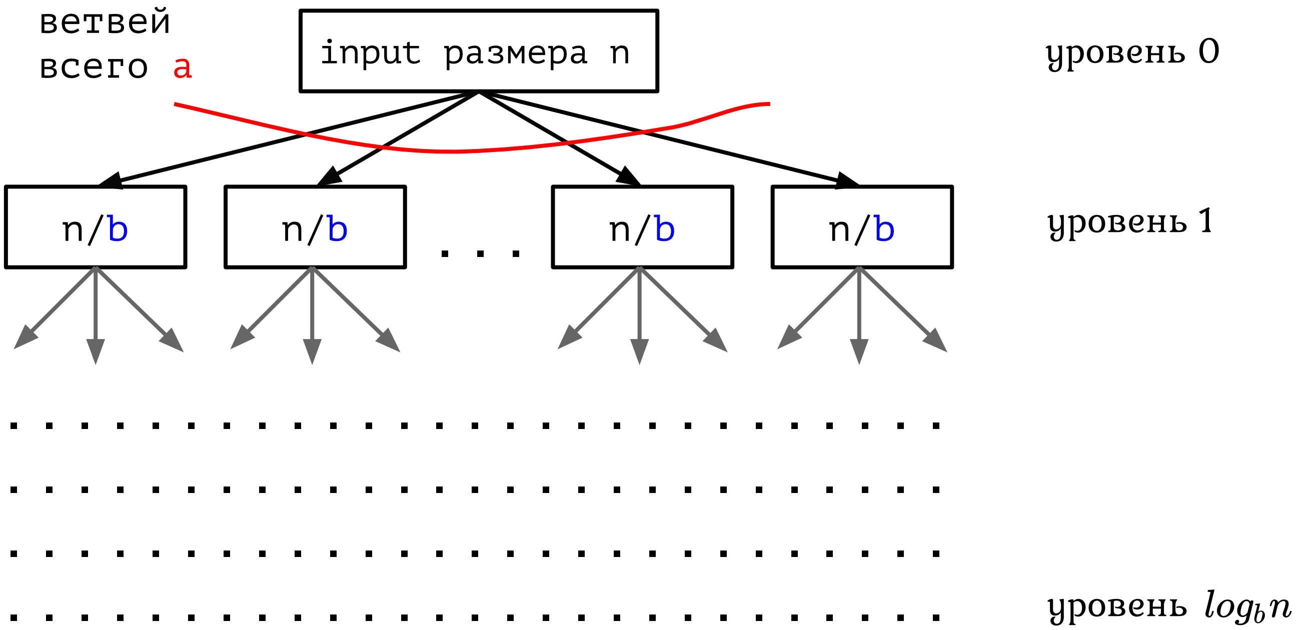


Обобщенный D&C алгоритм $T(n) \leq a * T(\frac{n}{b}) + c * n^d$

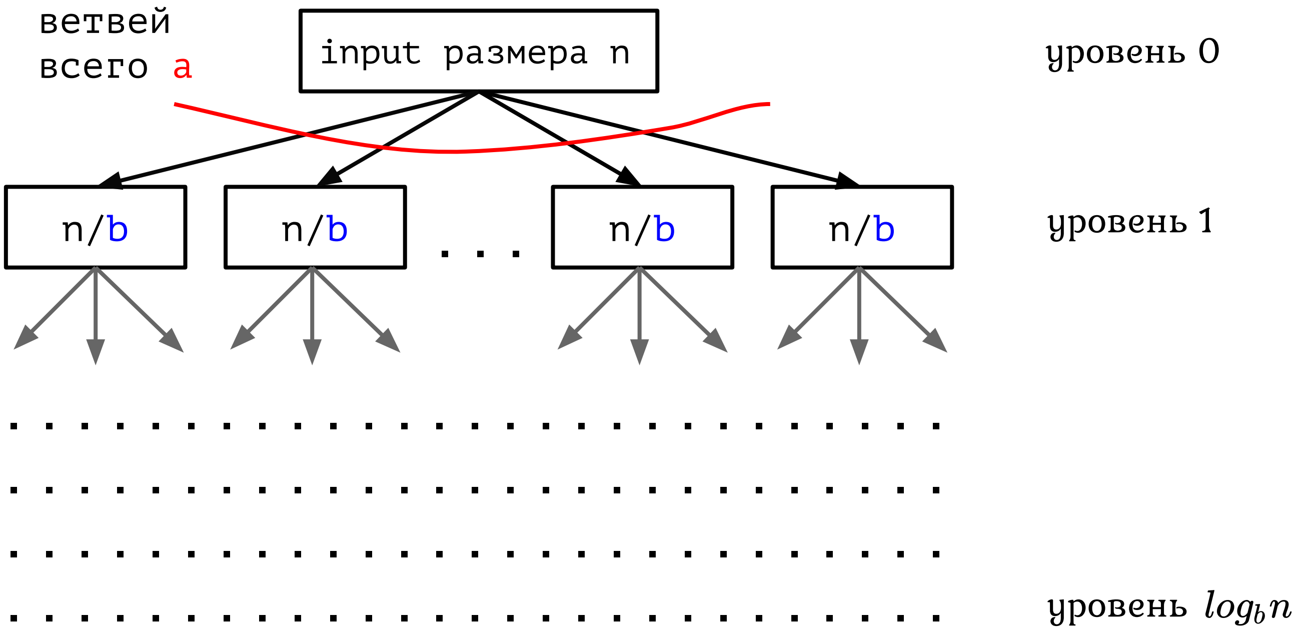


Обобщенный D&C алгоритм

$$T(n) \leq a * T(\frac{n}{b}) + c * n^d$$

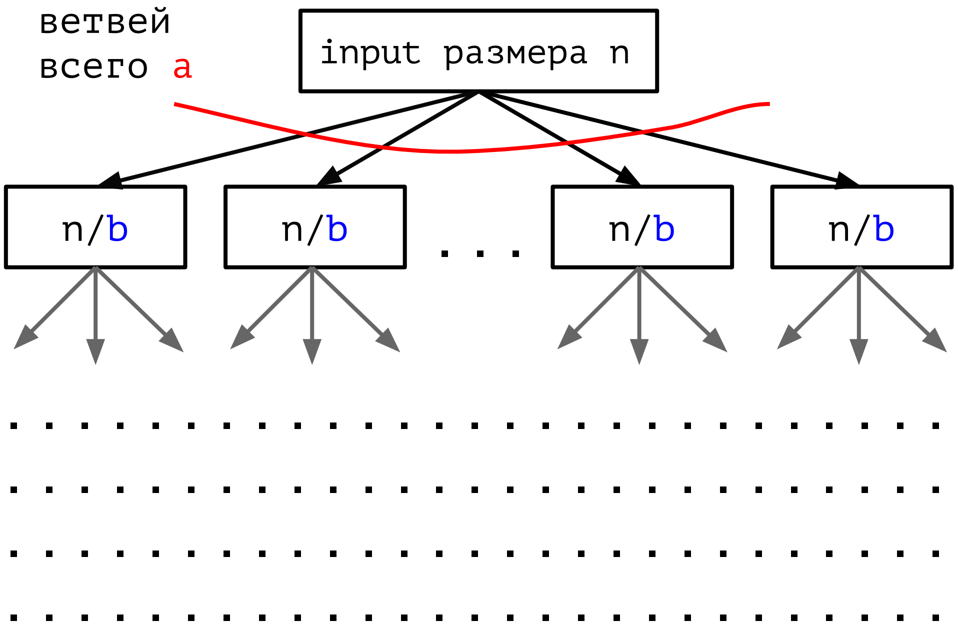


Обобщенный D&C алгоритм $T(n) \leq a * T(\frac{n}{b}) + c * n^d$



Сколько разных подзадач на уровне j?

Обобщенный D&C алгоритм $T(n) \leq a * T(\frac{n}{b}) + c * n^d$



уровень 0

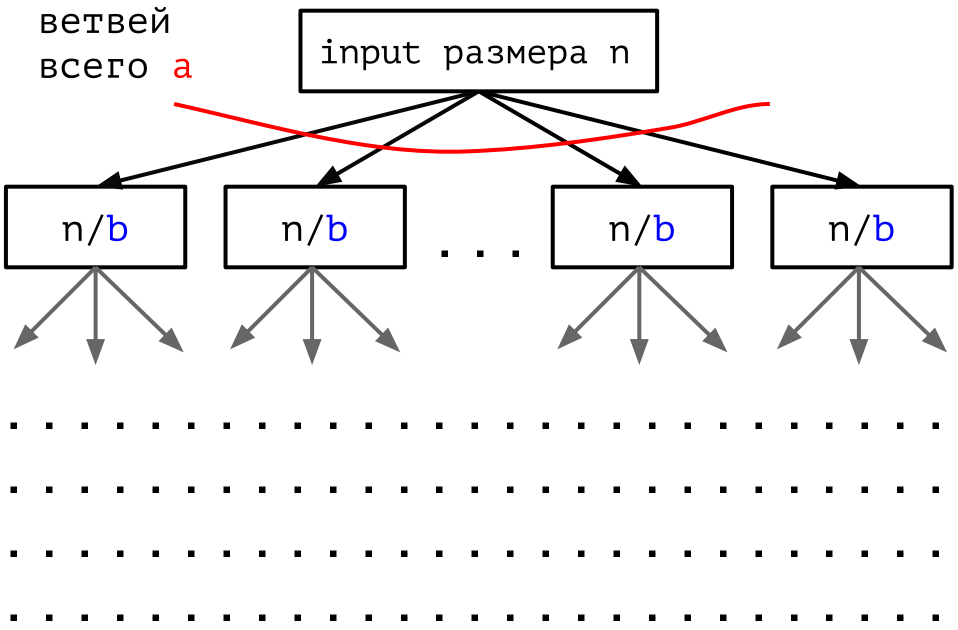
уровень 1

уровень $log_b n$

Сколько разных **подзадач** на уровне j?

$$a^j$$

Обобщенный D&C алгоритм $T(n) \leq a * T(\frac{n}{b}) + c * n^d$



уровень 0

уровень 1

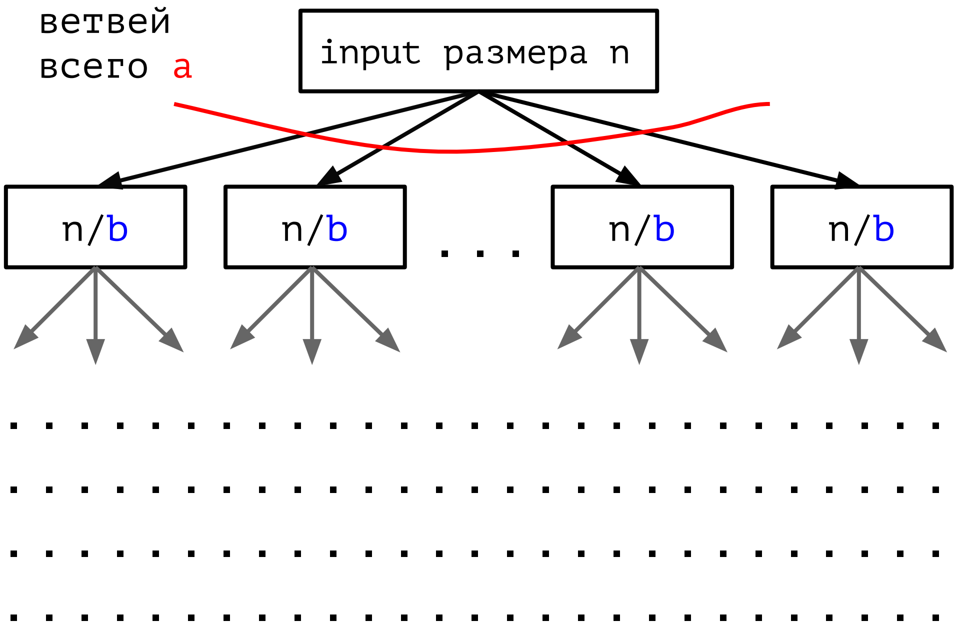
уровень $log_b n$

Сколько разных подзадач на уровне j?

a^j

Размер подзадачи на уровне j?

Обобщенный D&C алгоритм $T(n) \le a * T(\frac{n}{b}) + c * n^d$



уровень 0

уровень 1

уровень $log_b n$

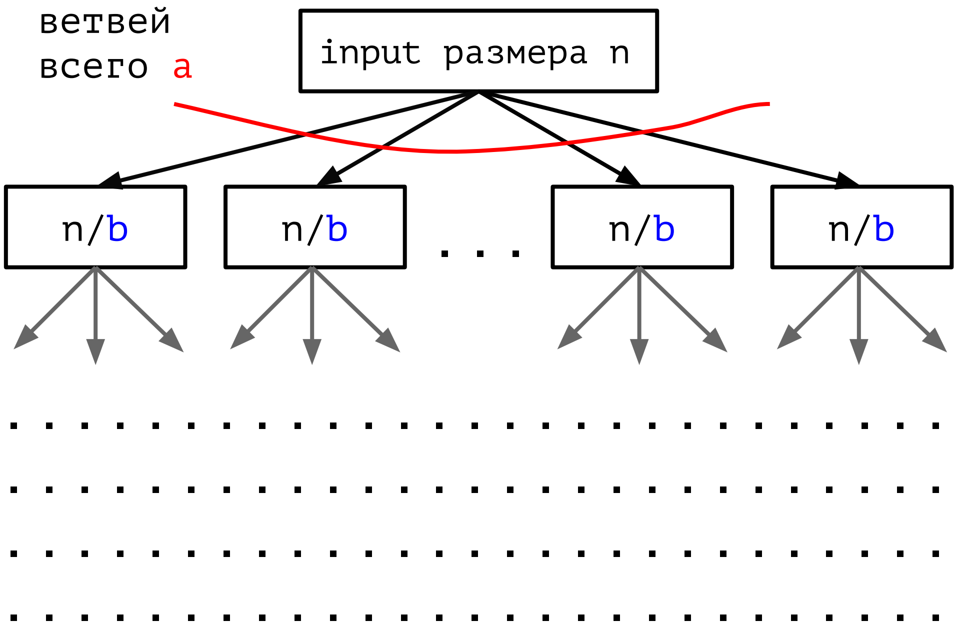
Сколько разных **подзадач** на уровне j?

$$a^j$$

Размер **подзадачи** на уровне j?

$$\frac{n}{b^j}$$

Обобщенный D&C алгоритм $T(n) \le a * T(\frac{n}{b}) + c * n^d$



уровень 0

уровень 1

уровень $log_b n$

Сколько разных подзадач на уровне j?

$$a^j$$

Размер подзадачи на уровне j?

$$\frac{n}{b^j}$$

Количество работы на уровне j?

Обобщенный D&C алгоритм $T(n) \leq a * T(\frac{n}{b}) + c * n^d$

Общее количество работы на уровне j
(без учета работы в рекурсивных вызовах)

Обобщенный D&C алгоритм $T(n) \leq a * T(\frac{n}{b}) + c * n^d$

Общее количество работы на уровне j
(без учета работы в рекурсивных вызовах)

$$\leq \qquad \frac{n}{b^j}$$

Такого размера была каждая
подзадача на уровне j

Обобщенный D&C алгоритм $T(n) \leq a * T(\frac{n}{b}) + \boxed{c * n^d}$

Общее количество работы на уровне j
(без учета работы в рекурсивных вызовах)

$$\leq \qquad c\left(\frac{n}{b^j}\right)^d$$

Столько мы потратили на
"слияние" этой подзадачи

Обобщенный D&C алгоритм $T(n) \leq a * T(\frac{n}{b}) + \boxed{c * n^d}$

Общее количество работы на уровне j
(без учета работы в рекурсивных вызовах)

$$\leq a^j * c(\frac{n}{b^j})^d$$

Столько было
подзадач на
уровне j

Столько мы потратили на
"слияние" этой подзадачи

Обобщенный D&C алгоритм $T(n) \le a * T(\frac{n}{b}) + c * n^d$

Общее количество работы на уровне j
(без учета работы в рекурсивных вызовах)

$$\le a^j * c(\frac{n}{b^j})^d = c * n^d(\frac{a}{b^d})^j$$

Обобщенный D&C алгоритм $T(n) \le a * T(\frac{n}{b}) + c * n^d$

Тогда общее количество работы на всех уровнях $j = 0, 1, 2, \ldots, log_b n$

$$\le c * n^d \sum_{j=0}^{log_b n} (\frac{a}{b^d})^j$$

Обобщенный D&C алгоритм $T(n) \leq a * T(\frac{n}{b}) + c * n^d$

Тогда общее количество работы на всех уровнях $j = 0, 1, 2, \ldots, log_b n$

$$\leq c * n^d \sum_{j=0}^{log_b n} \left(\frac{a}{b^d}\right)^j$$

Осталось проанализировать эту сумму!

Но сначала поговорим об её интерпретации для алгоритмов.

Обобщенный D&C алгоритм $T(n) \le a * T(\frac{n}{b}) + c * n^d$

$$c * n^d \left(\frac{a}{b^d}\right)^j$$

Как об этом думать?

Обобщенный D&C алгоритм $T(n) \leq a * T(\frac{n}{b}) + c * n^d$

$$c * n^d \left(\frac{a}{b^d}\right)^j$$

Как об этом думать?

a — скорость появления подзадач

Обобщенный D&C алгоритм $T(n) \leq a * T(\frac{n}{b}) + c * n^d$

$$c * n^d \left(\frac{a}{b^d}\right)^j$$

Как об этом думать?

a — скорость появления подзадач

b^d — ???

Обобщенный D&C алгоритм $T(n) \leq a * T(\frac{n}{b}) + c * n^d$

$$c * n^d \left(\frac{a}{b^d}\right)^j$$

Как об этом думать?

a — скорость появления подзадач

b^d — скорость уменьшения работы

Обобщенный D&C алгоритм $T(n) \leq a * T(\frac{n}{b}) + c * n^d$

$$c * n^d \left(\frac{a}{b^d}\right)^j$$

Как об этом думать?

1. $a = b^d$, тогда количество работы на каждом уровне одинаковое

Обобщенный D&C алгоритм $T(n) \leq a * T(\frac{n}{b}) + c * n^d$

$$c * n^d \left(\frac{a}{b^d}\right)^j$$

Как об этом думать?

1. $a = b^d$, тогда количество работы на каждом уровне одинаковое => оценим, как в merge sort

Обобщенный D&C алгоритм $T(n) \le a * T(\frac{n}{b}) + c * n^d$

$$c * n^d \left(\frac{a}{b^d}\right)^j$$

Как об этом думать?

1. $a = b^d$, тогда количество работы на каждом уровне одинаковое => оценим, как в merge sort

2. $a < b^d$, тогда количество работы падает на каждом уровне

Обобщенный D&C алгоритм $T(n) \le a * T(\frac{n}{b}) + c * n^d$

$$c * n^d \left(\frac{a}{b^d}\right)^j$$

Как об этом думать?

1. $a = b^d$, тогда количество работы на каждом уровне одинаковое => оценим, как в merge sort

2. $a < b^d$, тогда количество работы падает на каждом уровне => все ограничивается количеством работы на первом уровне (это сколько?)

Обобщенный D&C алгоритм $T(n) \le a * T(\frac{n}{b}) + c * n^d$

$$c * n^d \left(\frac{a}{b^d}\right)^j$$

Как об этом думать?

1. $a = b^d$, тогда количество работы на каждом уровне одинаковое => оценим, как в merge sort

2. $a < b^d$, тогда количество работы падает на каждом уровне => все ограничивается количеством работы на первом уровне, т.е. $O(n^d)$

$$c * n^d \left(\frac{a}{b^d}\right)^j$$

Как об этом думать?

1. $a = b^d$, тогда количество работы на каждом уровне одинаковое => оценим, как в merge sort

2. $a < b^d$, тогда количество работы падает на каждом уровне => все ограничивается количеством работы на первом уровне, т.е. $O(n^d)$

3. $a > b^d$?

$$c * n^d \left(\frac{a}{b^d}\right)^j$$

Как об этом думать?

1. $a = b^d$, тогда количество работы на каждом уровне одинаковое => оценим, как в merge sort

2. $a < b^d$, тогда количество работы падает на каждом уровне => все ограничивается количеством работы на первом уровне, т.е. $O(n^d)$

3. $a > b^d$, тогда количество работы только растет на каждом уровне

$$c * n^d \left(\frac{a}{b^d}\right)^j$$

Как об этом думать?

1. $a = b^d$, тогда количество работы на каждом уровне одинаковое => оценим, как в merge sort

2. $a < b^d$, тогда количество работы падает на каждом уровне => все ограничивается количеством работы на первом уровне, т.е. $O(n^d)$

3. $a > b^d$, тогда количество работы только растет на каждом уровне. Тогда все ограничивается работой в листьях рекурсивного дерева, т.е. $O(\#\text{листьев})$

$$c * n^d \left(\frac{a}{b^d}\right)^j$$

Это была интуиция, теперь докажем строго

Как об этом думать?

1. $a = b^d$, тогда количество работы на каждом уровне одинаковое => оценим, как в merge sort

2. $a < b^d$, тогда количество работы падает на каждом уровне => все ограничивается количеством работы на первом уровне, т.е. $O(n^d)$

3. $a > b^d$, тогда количество работы только растет на каждом уровне. Тогда все ограничивается работой в листьях рекурсивного дерева, т.е. $O(\#\text{листьев})$

Первый случай для $T(n) \leq a * T(\frac{n}{b}) + c * n^d$

Общее количество работы $\leq c * n^d \sum_{j=0}^{log_b n} (\frac{a}{b^d})^j$ $(*)$

Первый случай для $T(n) \le a * T(\frac{n}{b}) + c * n^d$

Общее количество работы $\le c * n^d \sum_{j=0}^{log_b n} (\frac{a}{b^d})^j \quad (*)$

Если $a = b^d$, то $(*) = c * n^d \sum_{0}^{log_b n} (1^j) =$

Первый случай для $T(n) \le a * T(\frac{n}{b}) + c * n^d$

Общее количество работы $\le c * n^d \sum_{j=0}^{log_b n} (\frac{a}{b^d})^j \quad (*)$

Если $a = b^d$, то $(*) = c * n^d \sum_0^{log_b n} (1^j) =$

$$= c * n^d (log_b n + 1) =$$

Первый случай для $T(n) \leq a * T(\frac{n}{b}) + c * n^d$

Общее количество работы $\leq c * n^d \sum_{j=0}^{log_b n} (\frac{a}{b^d})^j \quad (*)$

Если $a = b^d$, то $(*) = c * n^d \sum_0^{log_b n} (1^j) =$

$$= c * n^d (log_b n + 1) = O(n^d * logn)$$

Первый случай доказан

Если:

$$T(n) \leq a * T(\frac{n}{b}) + O(n^d)$$

то,

$$T(n) = \begin{cases} O(n^d * log(n)), & if\ a = b^d \\ O(n^d), & if\ a < b^d \\ O(n^{log_b a}), & if\ a > b^d \end{cases}$$ ✔

Ограничения:

1. $T(n) \leq constant$ для достаточно малых n
2. При рекурсивном переходе делим на равные подзадачи

Школьные факты про суммы

Пусть $r \geq 0, r \neq 1$

Школьные факты про суммы

Пусть $r \geq 0, r \neq 1$

Тогда $1 + r + r^2 + r^3 + \cdots + r^k = \frac{r^{k+1} - 1}{r - 1}$ $(**)$

сумма геометрической прогрессии

Школьные факты про суммы

Пусть $r \geq 0, r \neq 1$

Тогда $1 + r + r^2 + r^3 + \cdots + r^k = \frac{r^{k+1} - 1}{r - 1}$ $(**)$ сумма геометрической прогрессии

Если $r < 1$, то $(**) \leq \frac{1}{1-r}$

Школьные факты про суммы

Пусть $r \geq 0, r \neq 1$

Тогда $1 + r + r^2 + r^3 + \cdots + r^k = \frac{r^{k+1} - 1}{r - 1}$ $(**)$ сумма геометрической прогрессии

Если $r < 1$, то $(**) \leq \frac{1}{1-r}$

Если $r > 1$, то $(**) \leq r^k(1 - \frac{1}{r-1})$

Школьные факты про суммы

Пусть $r \geq 0, r \neq 1$

Тогда $1 + r + r^2 + r^3 + \cdots + r^k = \frac{r^{k+1} - 1}{r - 1}$ $(**)$ сумма геометрической прогрессии

Если $r < 1$, то $(**) \leq \boxed{\frac{1}{1-r}}$

Если $r > 1$, то $(**) \leq r^k \boxed{(1 - \frac{1}{r-1})}$

Это константы не зависящие от k!

Второй случай для $T(n) \leq a * T(\frac{n}{b}) + c * n^d$

Общее количество работы $\leq c * n^d \sum_{j=0}^{log_b n} (\frac{a}{b^d})^j$ $(*)$

Если $a < b^d$

Второй случай для $T(n) \leq a * T(\frac{n}{b}) + c * n^d$

Общее количество работы $\leq c * n^d \sum_{j=0}^{log_b n} (\frac{a}{b^d})^j \quad (*)$

Если $a < b^d$, то $(\frac{a}{b^d}) < 1$

Второй случай для $T(n) \le a * T(\frac{n}{b}) + c * n^d$

Общее количество работы $\le c * n^d \sum_{j=0}^{log_b n} (\frac{a}{b^d})^j$ $(*)$

Если $a < b^d$, то $(\frac{a}{b^d}) < 1$ и тогда $\sum_0^{log_b n} (\frac{a}{b^d})^j \le constant$

Второй случай для $T(n) \leq a * T(\frac{n}{b}) + c * n^d$

Общее количество работы $\leq c * n^d \sum_{j=0}^{log_b n} (\frac{a}{b^d})^j \quad (*)$

Если $a < b^d$, то $(\frac{a}{b^d}) < 1$ и тогда $\sum_0^{log_b n} (\frac{a}{b^d})^j \leq constant$

А значит $(*) \leq c * n^d * constant$

Второй случай для $T(n) \le a * T(\frac{n}{b}) + c * n^d$

Общее количество работы $\le c * n^d \sum_{j=0}^{log_b n} (\frac{a}{b^d})^j \quad (*)$

Если $a < b^d$, то $(\frac{a}{b^d}) < 1$ и тогда $\sum_{0}^{log_b n} (\frac{a}{b^d})^j \le constant$

А значит $(*) \le c * n^d * constant = O(n^d)$

Второй случай доказан!

Если:

$$T(n) \leq a * T(\frac{n}{b}) + O(n^d)$$

то,

$$T(n) = \begin{cases} O(n^d * log(n)), & if\ a = b^d \\ O(n^d), & if\ a < b^d \\ O(n^{log_b a}), & if\ a > b^d \end{cases}$$

Ограничения:

1. $T(n) \leq constant$ для достаточно малых n

2. При рекурсивном переходе делим на равные подзадачи

Третий случай для $T(n) \leq a * T(\frac{n}{b}) + c * n^d$

Общее
количество
работы

$$\leq c * n^d \sum_{j=0}^{log_b n} (\frac{a}{b^d})^j \quad (*)$$

Если $a > b^d$

Школьные факты про суммы

Пусть $r \geq 0, r \neq 1$

Тогда $1 + r + r^2 + r^3 + \cdots + r^k = \frac{r^{k+1} - 1}{r - 1}$ $(**)$

сумма геометрической прогрессии

Если $r < 1$, то $(**) \leq \boxed{\frac{1}{1-r}}$

Если $r > 1$, то $(**) \leq \boxed{r^k}\boxed{(1 - \frac{1}{r-1})}$

Это константы не зависящие от k!

Третий случай для $T(n) \leq a * T(\frac{n}{b}) + c * n^d$

Общее количество работы $\leq c * n^d \sum_{j=0}^{log_b n} (\frac{a}{b^d})^j \quad (*)$

Если $a > b^d$, то $(*) \leq c * n^d * (\frac{a}{b^d})^{log_b n} * constant$

Третий случай для $T(n) \leq a * T(\frac{n}{b}) + c * n^d$

Общее количество работы $\leq c * n^d \sum_{j=0}^{log_b n} (\frac{a}{b^d})^j \quad (*)$

Если $a > b^d$, то $(*) = O(n^d * (\frac{a}{b^d})^{log_b n})$

Третий случай для $T(n) \le a * T(\frac{n}{b}) + c * n^d$

Общее количество работы $\le c * n^d \sum_{j=0}^{log_b n} (\frac{a}{b^d})^j \quad (*)$

Если $a > b^d$, то $(*) = O(n^d * (\frac{a}{b^d})^{log_b n})$

Заметим, что $b^{-d*log_b n} = (b^{log_b n})^{-d} = n^{-d}$

Третий случай для $T(n) \le a * T(\frac{n}{b}) + c * n^d$

Общее количество работы $\le c * n^d \sum_{j=0}^{log_b n} (\frac{a}{b^d})^j \quad (*)$

Если $a > b^d$, то $(*) = O(\cancel{n^d} * (\frac{a}{b^d})^{log_b n})$

Заметим, что $b^{-d*log_b n} = (b^{log_b n})^{-d} = \cancel{n^{-d}}$

Третий случай для $T(n) \le a * T(\frac{n}{b}) + c * n^d$

Общее количество работы $\le c * n^d \sum_{j=0}^{log_b n} (\frac{a}{b^d})^j \quad (*)$

Если $a > b^d$, то $(*) = O(\cancel{n^d} * (\frac{a}{b^d})^{log_b n}) = O(a^{log_b n})$

Заметим, что $b^{-d*log_b n} = (b^{log_b n})^{-d} = \cancel{n^{-d}}$

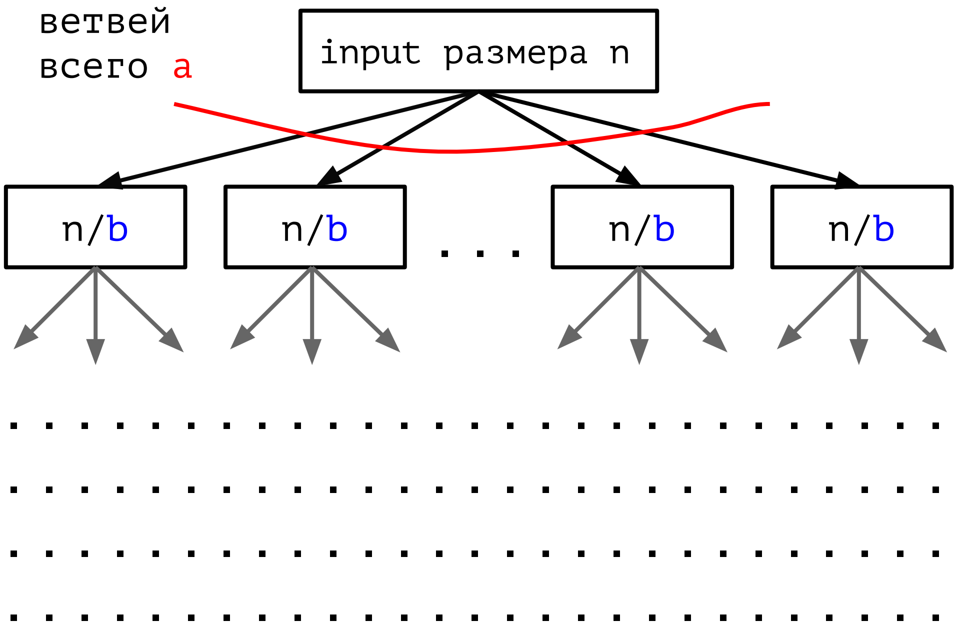
Третий случай для $T(n) \le a * T(\frac{n}{b}) + c * n^d$

Общее количество работы $\le c * n^d \sum_{j=0}^{log_b n} (\frac{a}{b^d})^j \quad (*)$

Если $a > b^d$, то $(*) = O(n^d * (\frac{a}{b^d})^{log_b n}) = O(a^{log_b n})$

Кстати, а что такое $a^{log_b n}$ в терминах дерева рекурсии?

Обобщенный D&C алгоритм $T(n) \leq a * T(\frac{n}{b}) + c * n^d$



уровень 0

уровень 1

уровень $log_b n$

Сколько разных подзадач на уровне j?

$$a^j$$

Размер подзадачи на уровне j?

$$\frac{n}{b^j}$$

Третий случай для $T(n) \leq a * T(\frac{n}{b}) + c * n^d$

Общее количество работы $\leq c * n^d \sum_{j=0}^{log_b n} (\frac{a}{b^d})^j \quad (*)$

Если $a > b^d$, то $(*) = O(n^d * (\frac{a}{b^d})^{log_b n}) = O(a^{log_b n})$

Кстати, а что такое $a^{log_b n}$ в терминах дерева рекурсии?

Это количество листьев! Как мы и предполагали.

Если:

$$T(n) \leq a * T(\frac{n}{b}) + O(n^d)$$

то,

$$T(n) = \begin{cases} O(n^d * log(n)), & if\ a = b^d \\ O(n^d), & if\ a < b^d \\ O(n^{log_b a}), & if\ a > b^d \end{cases}$$

✔ ✔ ???

Ограничения:

1. $T(n) \leq constant$ для достаточно малых n

2. При рекурсивном переходе делим на равные подзадачи

Третий случай для $T(n) \leq a * T(\frac{n}{b}) + c * n^d$

Общее количество работы $\leq c * n^d \sum_{j=0}^{log_b n} (\frac{a}{b^d})^j \quad (*)$

Если $a > b^d$, то $(*) = O(a^{log_b n})$

Вспомним, что $a^{log_b n} = n^{log_b a}$

Третий случай для $T(n) \leq a * T(\frac{n}{b}) + c * n^d$

Общее количество работы $\leq c * n^d \sum_{j=0}^{log_b n} (\frac{a}{b^d})^j \quad (*)$

Если $a > b^d$, то $(*) = O(a^{log_b n}) = O(n^{log_b a})$

Вспомним, что $a^{log_b n} = n^{log_b a}$

И третий случай доказан! $\square$

Если:

$$T(n) \leq a * T(\frac{n}{b}) + O(n^d)$$

то,

$$T(n) = \begin{cases} O(n^d * log(n)), & if\ a = b^d \\ O(n^d), & if\ a < b^d \\ O(n^{log_b a}), & if\ a > b^d \end{cases}$$

Ограничения:

1. $T(n) \leq constant$ для достаточно малых n

2. При рекурсивном переходе делим на равные подзадачи

Если:

$$T(n) \le a * T(\frac{n}{b}) + O(n^d)$$

то,

$$T(n) = \begin{cases} O(n^d * log(n)), & if\ a = b^d \\ O(n^d), & if\ a < b^d \\ O(n^{log_b a}), & if\ a > b^d \end{cases}$$

Замечания по доказательству:

1. Если в условии заменить O(...) на Θ(...), то и в результате получим Θ(...)

Если:

$$T(n) \leq a * T(\frac{n}{b}) + O(n^d)$$

то,

$$T(n) = \begin{cases} O(n^d * log(n)), & if\ a = b^d \\ O(n^d), & if\ a < b^d \\ O(n^{log_b a}), & if\ a > b^d \end{cases}$$

Замечания по доказательству:

1. Если в условии заменить O(...) на Θ(...), то и в результате получим Θ(...)

2. Есть формулировки, где вместо n^d используется $f(n)$

$$T(n) \leq a * T(\frac{n}{b}) + O(n^d) \quad \Rightarrow \quad T(n) = \begin{cases} O(n^d * log(n)), & if\ a = b^d \\ O(n^d), & if\ a < b^d \\ O(n^{log_b a}), & if\ a > b^d \end{cases}$$

$$T(n) \leq a * T(\frac{n}{b}) + O(n^d) \quad \Rightarrow \quad T(n) = \begin{cases} O(n^d * log(n)), & if\ a = b^d \\ O(n^d), & if\ a < b^d \\ O(n^{log_b a}), & if\ a > b^d \end{cases}$$

Пример #1:

Сортировка слиянием:

a = ?
b = ?
d = ?

$$T(n) \leq a * T(\frac{n}{b}) + O(n^d) \quad \Rightarrow \quad T(n) = \begin{cases} O(n^d * log(n)), & if\ a = b^d \\ O(n^d), & if\ a < b^d \\ O(n^{log_b a}), & if\ a > b^d \end{cases}$$

Пример #1:

Сортировка слиянием:

a = 2
b = 2
d = 1

$$T(n) \leq a * T(\frac{n}{b}) + O(n^d) \quad \Rightarrow \quad T(n) = \begin{cases} \boxed{O(n^d * log(n)), \ if\ a = b^d} \\ O(n^d), \quad if\ a < b^d \\ O(n^{log_b a}), \quad if\ a > b^d \end{cases}$$

Пример #1:

Сортировка слиянием:

a = 2
b = 2
d = 1

$$\Rightarrow \quad O(n * log(n))$$

$$T(n) \leq a * T(\frac{n}{b}) + O(n^d) \quad \Rightarrow \quad T(n) = \begin{cases} O(n^d * log(n)), & if\ a = b^d \\ O(n^d), & if\ a < b^d \\ O(n^{log_b a}), & if\ a > b^d \end{cases}$$

Пример #1.5:

Сортировка слиянием, но слияние работает за квадрат (плохая реализация):

a = ?
b = ?
d = ?

$$T(n) \le a * T(\frac{n}{b}) + O(n^d) \quad \Rightarrow \quad T(n) = \begin{cases} O(n^d * log(n)), & if\ a = b^d \\ O(n^d), & if\ a < b^d \\ O(n^{log_b a}), & if\ a > b^d \end{cases}$$

Пример #1.5:

Сортировка слиянием, но слияние работает за квадрат (плохая реализация):

a = 2
b = 2
d = 2

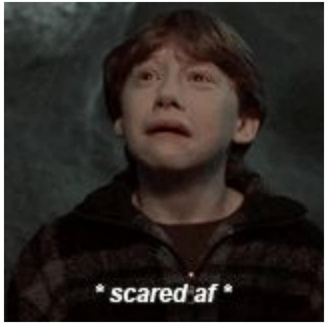
$$T(n) \le a * T(\tfrac{n}{b}) + O(n^d) \quad \Rightarrow \quad T(n) = \begin{cases} O(n^d * log(n)), & if\ a = b^d \\ \boxed{O(n^d), \quad if\ a < b^d} \\ O(n^{log_b a}), & if\ a > b^d \end{cases}$$

Пример #1.5:

Сортировка слиянием, но слияние работает за квадрат (плохая реализация):

a = 2
b = 2
d = 2

$\Rightarrow O(n^2)$

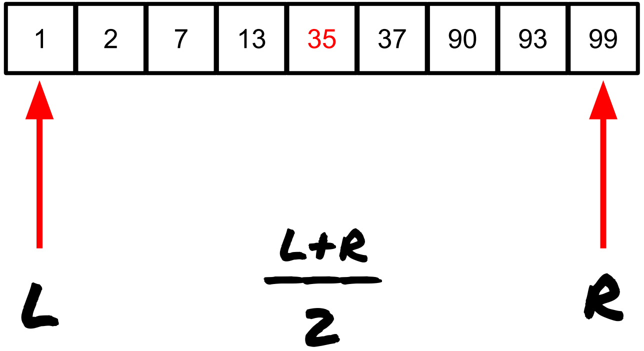


$$T(n) \leq a * T(\tfrac{n}{b}) + O(n^d) \quad \Rightarrow \quad T(n) = \begin{cases} O(n^d * log(n)), & if\; a = b^d \\ O(n^d), & if\; a < b^d \\ O(n^{log_b a}), & if\; a > b^d \end{cases}$$

Пример #2:

Бинарный поиск:

a = ?
b = ?
d = ?



$$T(n) \le a * T(\tfrac{n}{b}) + O(n^d) \quad \Rightarrow \quad T(n) = \begin{cases} O(n^d * log(n)), & if\ a = b^d \\ O(n^d), & if\ a < b^d \\ O(n^{log_b a}), & if\ a > b^d \end{cases}$$

Пример #2:

Бинарный поиск:

a = 1
b = 2
d = 0

1	2	7	13	35	37	90	93	99

L $\qquad \frac{L+R}{2} \qquad$ R

$$T(n) \leq a * T(\tfrac{n}{b}) + O(n^d) \quad \Rightarrow \quad T(n) = \begin{cases} \boxed{O(n^d * log(n)), \ if \ a = b^d} \\ O(n^d), \quad if \ a < b^d \\ O(n^{log_b a}), \quad if \ a > b^d \end{cases}$$

Пример #2:

Бинарный поиск:

a = 1
b = 2
d = 0

$$\Rightarrow O(log(n))$$

1	2	7	13	35	37	90	93	99

L

$\frac{L+R}{2}$

R

$$T(n) \le a * T(\frac{n}{b}) + O(n^d) \quad \Rightarrow \quad T(n) = \begin{cases} O(n^d * log(n)), & if\ a = b^d \\ O(n^d), & if\ a < b^d \\ O(n^{log_b a}), & if\ a > b^d \end{cases}$$

Пример #3:

Алгоритм Карацубы:

a = ?
b = ?
d = ?

Алгоритм Карацубы

a b
12 34
56 78
c d

Идея: $x * y = 10^N ac + 10^{\frac{N}{2}}(ad + bc) + bd$

Шаг 1: рекурсивно вычисляем ac

Шаг 2: рекурсивно вычисляем bd

Шаг 3: рекурсивно вычисляем $(a + b)(c + d) = \cancel{ac} + ad + bc + \cancel{bd}$

Шаг 4: ad + bc = (3) - (1) - (2)

Шаг 5: Суммируем со сдвигами, получаем ответ!

$$T(n) \leq a * T(\frac{n}{b}) + O(n^d) \quad \Rightarrow \quad T(n) = \begin{cases} O(n^d * log(n)), & if\ a = b^d \\ O(n^d), & if\ a < b^d \\ O(n^{log_b a}), & if\ a > b^d \end{cases}$$

Пример #3:

Алгоритм Карацубы:

a = ?
b = ?
d = ?

$$T(n) \le a * T(\frac{n}{b}) + O(n^d) \quad \Rightarrow \quad T(n) = \begin{cases} O(n^d * log(n)), & if\ a = b^d \\ O(n^d), & if\ a < b^d \\ O(n^{log_b a}), & if\ a > b^d \end{cases}$$

Пример #3:

Алгоритм Карацубы:

a = 3
b = 2
d = 1

$$T(n) \leq a * T(\frac{n}{b}) + O(n^d) \quad \Rightarrow \quad T(n) = \begin{cases} O(n^d * log(n)), & if\ a = b^d \\ O(n^d), & if\ a < b^d \\ \boxed{O(n^{log_b a}), \quad if\ a > b^d} \end{cases}$$

Пример #3:

Алгоритм Карацубы:

a = 3
b = 2
d = 1

$$\Rightarrow O(n^{log_2 3})$$

$$T(n) \leq a * T(\frac{n}{b}) + O(n^d) \quad \Rightarrow \quad T(n) = \begin{cases} O(n^d * log(n)), & if\ a = b^d \\ O(n^d), & if\ a < b^d \\ \boxed{O(n^{log_b a}), \quad if\ a > b^d} \end{cases}$$

Пример #3:

Алгоритм Карацубы:

a = 3
b = 2
d = 1

$$\Rightarrow O(n^{log_2 3}) \approx O(n^{1.59})$$

Алгоритм Карацубы

Но что с количеством элементарных операций? 🤔

И какой алгоритм лучше: в столбик или Карацубы?

Алгоритм Карацубы

Но что с количеством элементарных операций? 🤔

И какой алгоритм лучше: в столбик или Карацубы?

$O(n^2)$ $O(n^{1.59})$

Алгоритм Карацубы

Замечания:

1. Числа могут быть разной длины, алгоритм продолжает работать (подумайте, как это учесть в коде?)

2. Если бы рекурсивных вызовов было 4, а не 3, то алгоритм стал бы значительно хуже умножения в столбик!

$$T(n) \leq a * T(\frac{n}{b}) + O(n^d) \quad \Rightarrow \quad T(n) = \begin{cases} O(n^d * log(n)), & if\ a = b^d \\ O(n^d), & if\ a < b^d \\ O(n^{log_b a}), & if\ a > b^d \end{cases}$$

Пример #3.5:

Карацуба, но с 4 рекурсивными вызовами:

a = ?
b = ?
d = ?

$$T(n) \leq a * T(\frac{n}{b}) + O(n^d) \quad \Rightarrow \quad T(n) = \begin{cases} O(n^d * log(n)), & if\ a = b^d \\ O(n^d), & if\ a < b^d \\ \boxed{O(n^{log_b a}), \quad if\ a > b^d} \end{cases}$$

Пример #3.5:

Карацуба, но с 4 рекурсивными вызовами:

a = 4
b = 2
d = 1

$$T(n) \le a * T(\frac{n}{b}) + O(n^d) \quad \Rightarrow \quad T(n) = \begin{cases} O(n^d * log(n)), & if\ a = b^d \\ O(n^d), & if\ a < b^d \\ \boxed{O(n^{log_b a}), \quad if\ a > b^d} \end{cases}$$

Пример #3.5:

Карацуба, но с 4 рекурсивными вызовами:

a = 4
b = 2 $\Rightarrow O(n^{log_2 4})$
d = 1

$$T(n) \le a * T(\tfrac{n}{b}) + O(n^d) \quad \Rightarrow \quad T(n) = \begin{cases} O(n^d * log(n)), & if\ a = b^d \\ O(n^d), & if\ a < b^d \\ \boxed{O(n^{log_b a}), \quad if\ a > b^d} \end{cases}$$

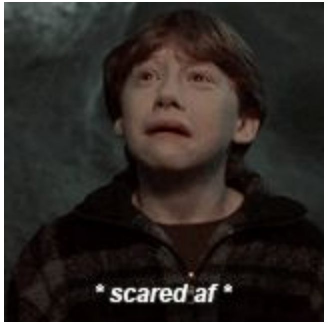
Пример #3.5:

Карацуба, но с 4 рекурсивными вызовами:

a = 4
b = 2
d = 1

$$\Rightarrow \quad O(n^{log_2 4}) = O(n^2)$$

(при этом константы
хуже, чем в столбик)



$$T(n) \leq a * T(\frac{n}{b}) + O(n^d) \quad \Rightarrow \quad T(n) = \begin{cases} O(n^d * log(n)), & if \; a = b^d \\ O(n^d), & if \; a < b^d \\ O(n^{log_b a}), & if \; a > b^d \end{cases}$$

Пример #4:

Рекурсивный алгоритм умножения матриц:

Умножение матриц

Умножение матриц

```
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)

Умножение матриц

```
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*N и N*N?

Умножение матриц

```
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*N и N*N? $O(N^3)$

Быстрое умножение матриц

$$X = \left(\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right)$$

$$Y = \left(\begin{array}{c|c} E & F \\ \hline G & H \end{array}\right)$$

Если A, B, C, D, E, F, G, H - блоки размера N/2 * N/2, то что можно сказать про X*Y?

Быстрое умножение матриц

$$X = \left(\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right)$$

$$Y = \left(\begin{array}{c|c} E & F \\ \hline G & H \end{array}\right)$$

$$X*Y = \left(\begin{array}{c|c} AE+BG & AF+BH \\ \hline CE+DG & CF+DH \end{array}\right)$$

Быстрое умножение матриц

Напрашивается рекурсивный алгоритм!

$$X = \left(\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right)$$

$$Y = \left(\begin{array}{c|c} E & F \\ \hline G & H \end{array}\right)$$

$$X*Y = \left(\begin{array}{c|c} AE+BG & AF+BH \\ \hline CE+DG & CF+DH \end{array}\right)$$

Быстрое умножение матриц

$$X = \left(\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right)$$

$$Y = \left(\begin{array}{c|c} E & F \\ \hline G & H \end{array}\right)$$

Напрашивается рекурсивный алгоритм!

$$X*Y = \left(\begin{array}{c|c} AE+BG & AF+BH \\ \hline CE+DG & CF+DH \end{array}\right)$$

Сколько подзадач?
Размер подзадачи?
Сложность "слияния"?

Быстрое умножение матриц

$$X = \left(\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right)$$

$$Y = \left(\begin{array}{c|c} E & F \\ \hline G & H \end{array}\right)$$

Напрашивается рекурсивный алгоритм!

$$X*Y = \left(\begin{array}{c|c} AE+BG & AF+BH \\ \hline CE+DG & CF+DH \end{array}\right)$$

Сколько подзадач? **8**
Размер подзадачи?
Сложность "слияния"?

Быстрое умножение матриц

$$X = \left(\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right)$$

$$Y = \left(\begin{array}{c|c} E & F \\ \hline G & H \end{array}\right)$$

Напрашивается рекурсивный алгоритм!

$$X*Y = \left(\begin{array}{c|c} AE+BG & AF+BH \\ \hline CE+DG & CF+DH \end{array}\right)$$

Сколько подзадач? **8**

Размер подзадачи? N/2

Сложность "слияния"?

Быстрое умножение матриц

$$X = \left(\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right)$$

$$Y = \left(\begin{array}{c|c} E & F \\ \hline G & H \end{array}\right)$$

Напрашивается рекурсивный алгоритм!

$$X*Y = \left(\begin{array}{c|c} AE+BG & AF+BH \\ \hline CE+DG & CF+DH \end{array}\right)$$

```
Сколько подзадач?     8
Размер подзадачи?     N/2
Сложность "слияния"? O(N^2)
```

$$T(n) \leq a * T(\frac{n}{b}) + O(n^d) \quad \Longrightarrow \quad T(n) = \begin{cases} O(n^d * log(n)), & if\ a = b^d \\ O(n^d), & if\ a < b^d \\ O(n^{log_b a}), & if\ a > b^d \end{cases}$$

Пример #4:

Рекурсивный алгоритм умножения матриц:

a = 8
b = 2
d = 2

$$T(n) \le a * T(\frac{n}{b}) + O(n^d) \quad \Rightarrow \quad T(n) = \begin{cases} O(n^d * log(n)), & if\ a = b^d \\ O(n^d), & if\ a < b^d \\ \boxed{O(n^{log_b a}), \quad if\ a > b^d} \end{cases}$$

Пример #4:

Рекурсивный алгоритм умножения матриц:

$$\begin{matrix} a = 8 \\ b = 2 \\ d = 2 \end{matrix} \Rightarrow O(n^{log_2 8}) = O(n^3)$$

Алгоритм Штрассена

$$X = \left(\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right)$$

$$Y = \left(\begin{array}{c|c} E & F \\ \hline G & H \end{array}\right)$$

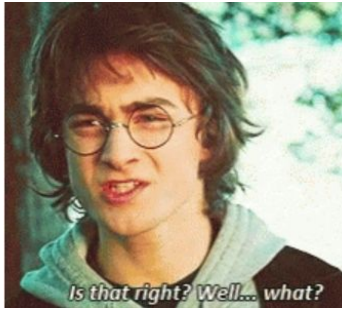
Алгоритм Штрассена

$$X = \left(\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right)$$

$$Y = \left(\begin{array}{c|c} E & F \\ \hline G & H \end{array}\right)$$

```
P1 = A(F-H)
P2 = (A+B)H
P3 = (C+D)E
P4 = D(G-E)
P5 = (A+D)(E+H)
P6 = (B-D)(G+H)
P7 = (A-C)(E+F)
```

Алгоритм Штрассена

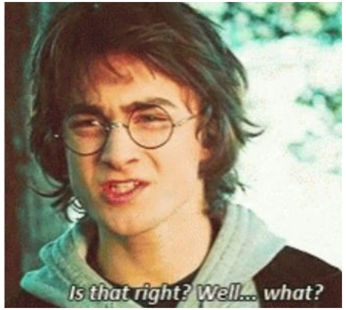


$$X = \left(\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right)$$

$$Y = \left(\begin{array}{c|c} E & F \\ \hline G & H \end{array}\right)$$

P1 = A(F-H)
P2 = (A+B)H
P3 = (C+D)E
P4 = D(G-E)
P5 = (A+D)(E+H)
P6 = (B-D)(G+H)
P7 = (A-C)(E+F)

Алгоритм Штрассена



P1 = A(F-H)
P2 = (A+B)H
P3 = (C+D)E
P4 = D(G-E)
P5 = (A+D)(E+H)
P6 = (B-D)(G+H)
P7 = (A-C)(E+F)

Q1 = P5 + P4 - P2 + P6
Q2 = P1 + P2
Q3 = P3 + P4
Q4 = P1 + P5 - P3 - P7

Алгоритм Штрассена

$$X = \left(\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right)$$

$$Y = \left(\begin{array}{c|c} E & F \\ \hline G & H \end{array}\right)$$

Тогда верно следующее:

$$X*Y = \left(\begin{array}{c|c} Q1 & Q2 \\ \hline Q3 & Q4 \end{array}\right)$$

Алгоритм Штрассена

$$X*Y = \left(\begin{array}{c|c} \boxed{AE+BG} & AF+BH \\ \hline CE+DG & CF+DH \end{array}\right)$$

$$X*Y = \left(\begin{array}{c|c} \boxed{Q1} & Q2 \\ \hline Q3 & Q4 \end{array}\right)$$

Алгоритм Штрассена

$$X*Y = \left(\begin{array}{c|c} \boxed{AE+BG} & AF+BH \\ \hline CE+DG & CF+DH \end{array}\right) \qquad X*Y = \left(\begin{array}{c|c} \boxed{Q1} & Q2 \\ \hline Q3 & Q4 \end{array}\right)$$

$$Q1 = P5 + P4 - P2 + P6 =$$

Алгоритм Штрассена

$$X*Y = \left(\begin{array}{c|c} \boxed{AE+BG} & AF+BH \\ \hline CE+DG & CF+DH \end{array}\right) \qquad X*Y = \left(\begin{array}{c|c} \boxed{Q1} & Q2 \\ \hline Q3 & Q4 \end{array}\right)$$

$$Q1 = P5 + P4 - P2 + P6 =$$
$$= AE + AH + DE + DH + DG - DE - AH -$$
$$BH + BG + BH - DG - DH$$

Алгоритм Штрассена

$$X*Y = \left(\begin{array}{c|c} \boxed{AE+BG} & AF+BH \\ \hline CE+DG & CF+DH \end{array}\right)$$

$$X*Y = \left(\begin{array}{c|c} \boxed{Q1} & Q2 \\ \hline Q3 & Q4 \end{array}\right)$$

$$Q1 = P5 + P4 - P2 + P6 =$$
$$= AE + \cancel{AH} + \cancel{DE} + \cancel{DH} + \cancel{DG} - \cancel{DE} - \cancel{AH} -$$
$$\cancel{BH} + BG + \cancel{BH} - \cancel{DG} - \cancel{DH}$$

Алгоритм Штрассена

$$X*Y = \left(\begin{array}{c|c} \boxed{AE+BG} & AF+BH \\ \hline CE+DG & CF+DH \end{array}\right) \qquad X*Y = \left(\begin{array}{c|c} \boxed{Q1} & Q2 \\ \hline Q3 & Q4 \end{array}\right)$$

$$Q1 = P5 + P4 - P2 + P6 =$$

$$= AE + \cancel{AH} + \cancel{DE} + \cancel{DH} + \cancel{DG} - \cancel{DE} - \cancel{AH} -$$

$$\cancel{BH} + BG + \cancel{BH} - \cancel{DG} - \cancel{DH} = AE + BG$$

Алгоритм Штрассена

1. Делаем 7 рекурсивных вызовов для подсчета P

Алгоритм Штрассена

1. Делаем 7 рекурсивных вызовов для подсчета P
2. Размер каждой подзадачи N/2

Алгоритм Штрассена

1. Делаем 7 рекурсивных вызовов для подсчета P

2. Размер каждой подзадачи N/2

3. Соединяем результат все еще за N^2

$$T(n) \leq a * T(\frac{n}{b}) + O(n^d) \quad \Rightarrow \quad T(n) = \begin{cases} O(n^d * log(n)), & if\ a = b^d \\ O(n^d), & if\ a < b^d \\ O(n^{log_b a}), & if\ a > b^d \end{cases}$$

Пример #4:

Алгоритм Штрассена умножения матриц:

a = 7
b = 2
d = 2

$$T(n) \leq a * T(\frac{n}{b}) + O(n^d) \quad \Rightarrow \quad T(n) = \begin{cases} O(n^d * log(n)), & if\ a = b^d \\ O(n^d), & if\ a < b^d \\ \boxed{O(n^{log_b a}), \quad if\ a > b^d} \end{cases}$$

Пример #4:

Алгоритм Штрассена умножения матриц:

a = 7
b = 2
d = 2

$$\Rightarrow O(n^{log_2 7}) \approx O(n^{2.81})$$

$$T(n) \le a * T(\frac{n}{b}) + O(n^d) \quad \Rightarrow \quad T(n) = \begin{cases} O(n^d * log(n)), & if\ a = b^d \\ O(n^d), & if\ a < b^d \\ \boxed{O(n^{log_b a}), \quad if\ a > b^d} \end{cases}$$

Пример #4:

Алгоритм Штрассена умножения матриц:

Лучше, чем классический алгоритм!

a = 7
b = 2
d = 2

$$\Rightarrow O(n^{log_2 7}) \approx O(n^{2.81})$$

Takeaways

- Master Method - как близкий к универсальному метод оценки сложности D&C алгоритмов

Takeaways

- Master Method - как близкий к универсальному метод оценки сложности D&C алгоритмов

- Доказательство сводится к рассмотрению трех случаев изменения работы на очередном уровне дерева рекурсии

Takeaways

- Master Method - как близкий к универсальному метод оценки сложности D&C алгоритмов
- Доказательство сводится к рассмотрению трех случаев изменения работы на очередном уровне дерева рекурсии
- Карацуба все-таки лучше (по асимптотике) умножения в столбик

Takeaways

- Master Method - как близкий к универсальному метод оценки сложности D&C алгоритмов
- Доказательство сводится к рассмотрению трех случаев изменения работы на очередном уровне дерева рекурсии
- Карацуба все-таки лучше (по асимптотике) умножения в столбик
- Алгоритм Штрассена для $O(n^{2.81})$ умножения матриц

Мини-задача #9 (1-3 балла)

Реализовать три алгоритма умножения квадратных матриц: классический, через 8 рекурсивных вызовов и алгоритм Штрассена. [1 балл]

Мини-задача #9 (1-3 балла)

Реализовать три алгоритма умножения квадратных матриц: классический, через 8 рекурсивных вызовов и алгоритм Штрассена. [1 балл]

Кроме того, реализовать тестовый стенд, принимающий на вход набор алгоритмов и набор входных данных, а возвращающий подробный отчет, сколько времени заняла работа каждого алгоритма на каждом примере входных данных. [1 балл]

Данные записывать в отформатированную таблицу (см. 9 мини задачу из курса Python)

Мини-задача #9 (1-3 балла)

Реализовать три алгоритма умножения квадратных матриц: классический, через 8 рекурсивных вызовов и алгоритм Штрассена. [1 балл]

Кроме того, реализовать тестовый стенд, принимающий на вход набор алгоритмов и набор входных данных, а возвращающий подробный отчет, сколько времени заняла работа каждого алгоритма на каждом примере входных данных. [1 балл]

Продемонстрировать с помощью стенда, начиная с какого порядка данных Штрассен начинает выигрывать [1 балл]

Мини-задача #9 (1-3 балла)

Замечание:

Запускать каждый алгоритм на одних и тех же данных стоит по несколько раз. По результатам этих запусков необходимо находить выборочное среднее, стандартное отклонение и среднее геометрическое, и выводить именно их комбинацию в качестве результата.

Это повысит правдоподобность результатов, снизив влияние внешних факторов на ваше измерение.