# Lecture 2

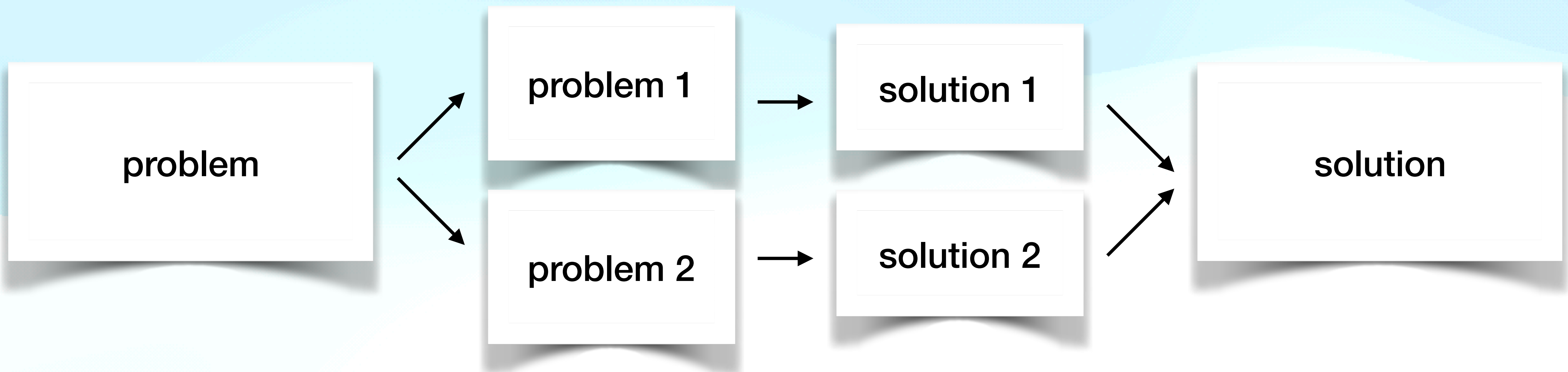
Divide and conquer

# Today's plan

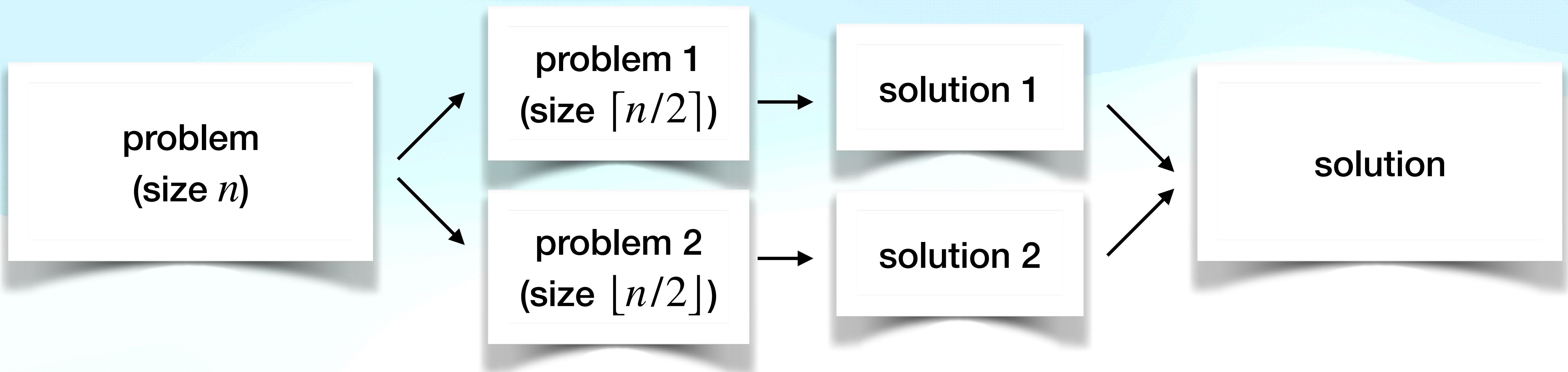
1. Divide and conquer
2. Analysis of recursive algorithms
3. Master theorem
4. Fast Fourier transform

# Divide and conquer

# Divide and conquer



# Divide and conquer




**Algorithm(problem, $n$)**

solution 1 = Algorithm(problem, $n/2$)
solution 2 = Algorithm(problem, $n/2$)
solution = solution 1 + solution 2
**return** solution

$$\boxed{\begin{aligned} T(1) &= const \\ T(n) &= T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + f(n) \end{aligned}}$$

# Merge sort (tri fusion)

**Task:** sort a list $L$ of $n$ integers

**Divide-and-conquer:** partition the list into two halves, sort each half recursively, and then *merge* the two sorted lists.

# Merge sort

Partition the elements into two groups, sort each group to obtain a sorted list, and then interleave the lists (animation)

```
MergeSort(L, low, high)
if (low < high) then
    middle = (low+high)/2
    L1 = MergeSort(L,low,middle)
    L2 = MergeSort(L,middle+1,high)
    return Merge(L1,L2)
```

To sort the list L, we call MergeSort(L, 1, n).

# Merge sort

- The efficiency of mergesort depends upon how efficiently we merge the two sorted halves $L_1$, $L_2$.
- If $L_1$ or $L_2$ is empty, we simply copy the other list to the output.
- Otherwise let $x$ be the minimum of the head elements in $L_1$ and $L_2$. We pop $x$ and push it to the output, and apply the procedure to the new lists $L_1$ and $L_2$ recursively.
- $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + O(n)$. Later we will show that $T(n) = O(n \log n)$.

# Analysis of recursive algorithms

M.C. Esher. Circle limit III.

# Recurrences

$$\boxed{\begin{aligned} T(1) &= const \\ T(n) &= T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + f(n) \end{aligned}}$$

**How to find the asymptotic of $T(n)$?**

- Substitution method
- Recursion-tree method
- Master method

**Remark:** The constant in the first equation does not change the asymptotic. For this reason, the first equation is often omitted.

# Substitution method

- Guess the asymptotic of $T(n)$
- Show that the answer is correct via the mathematical induction

**Example: Merge sort**

$$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + a \cdot n$$

no recipe

Our guess: $T(n) \leq c \cdot n \log_2 n$ for all $n \geq 2$

$$T(2) \leq 2c \Leftrightarrow c \geq T(2)/2,\ T(3) \leq 3c \log_2 3 \Leftrightarrow c \geq T(3)/3 \log_2 3, \ldots,$$
$$T(6) \leq 6c \log_2 6 \Leftrightarrow c \geq T(6)/6 \log_2 6$$

# Substitution method

$$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + a \cdot n$$

$$T(n) \leq c(\frac{n}{2}+1)\log_2(\frac{n}{2}+1) + c\frac{n}{2}\log_2\frac{n}{2} + a \cdot n \leq cn\log_2\frac{n}{2} + c\log_2\frac{n}{2} + c(\frac{n}{2}+1) + a \cdot n$$

(As $\log_2(\frac{n}{2}+1) \leq \log_2(\frac{n}{2} \cdot 2) \leq \log_2\frac{n}{2} + 1$ for $n \geq 4$)

$$T(n) \leq c \cdot n(\log_2 n - 1) + c\log_2\frac{n}{2} + c(\frac{n}{2}+1) + a \cdot n =$$

$$= c \cdot n\log_2 n + a \cdot n + c \cdot \log_2 n - c\frac{n}{2}$$

# Substitution method

$$T(n) = c \cdot n \log_2 n + a \cdot n + c \cdot \log_2 n - c\frac{n}{2}$$

For all $n \geq 6$, $c \log_2 n - cn/2 \leq -cn/20$. Choosing $c \geq \max\{20a, T(2)/2, \ldots, T(6)/6 \log_2 6\}$, we finally obtain

$$T(n) = c \cdot n \log_2 n + a \cdot n + c \cdot \log_2 n - c \cdot (\frac{n}{2} - 1) \leq$$

$$\leq cn \log_2 n + a \cdot n - c \cdot n/20 \leq cn \log_2 n$$

# Recursion-tree method

$$T(n) = 3T(\lfloor n/3 \rfloor) + n^2$$

# Recursion-tree method

Simplification:
$T(n) = 3T(n/3) + n^2$
$n = 3^k$

$$T(n) = 3T(\lfloor n/3 \rfloor) + n^2$$

# Recursion-tree method

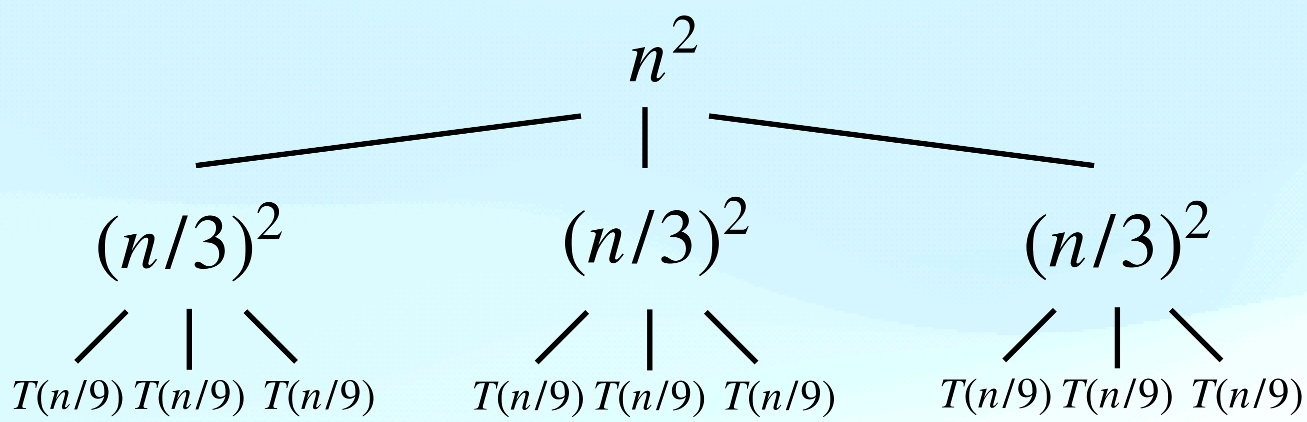
$$n^2$$

$$T(n/3) \qquad T(n/3) \qquad T(n/3)$$

**Simplification:**
$T(n) = 3T(n/3) + n^2$
$n = 3^k$

$$T(n) = 3T(\lfloor n/3 \rfloor) + n^2$$

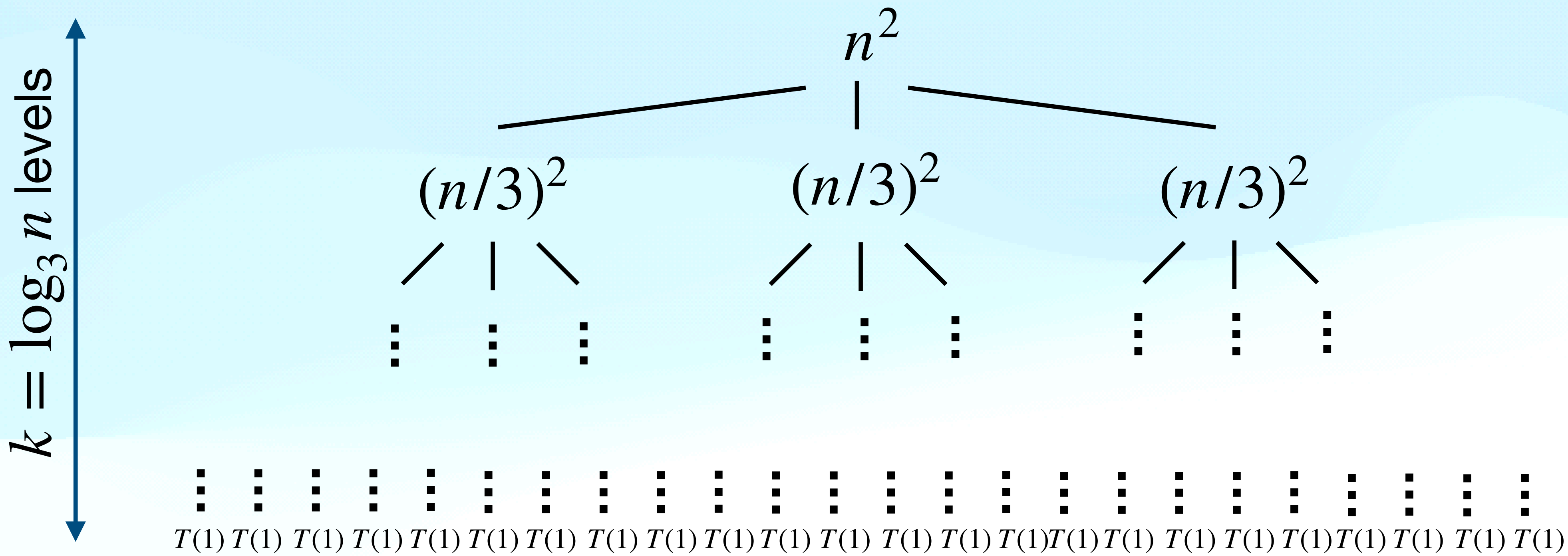
# Recursion-tree method




**Simplification:**
$T(n) = 3T(n/3) + n^2$
$n = 3^k$

$$T(n) = 3T(\lfloor n/3 \rfloor) + n^2$$

# Recursion-tree method




$3^k = n$ leaves

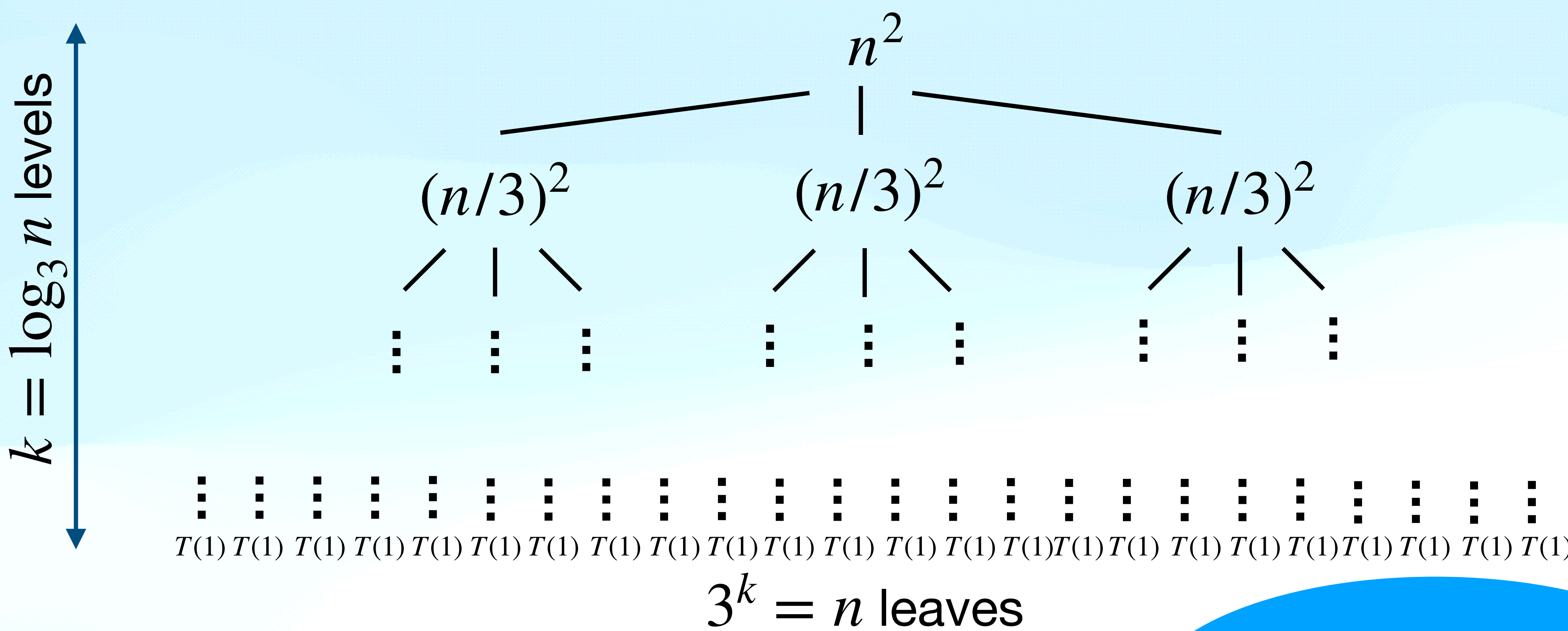
**Simplification:**

$T(n) = 3T(n/3) + n^2$

$n = 3^k$

$$T(n) = 3T(\lfloor n/3 \rfloor) + n^2$$

# Recursion-tree method




We now must verify the answer

$$T(n) = 3T(\lfloor n/3 \rfloor) + n^2$$

$$T(n) = \sum_{k=0}^{\log_3 n - 1} 3^k \cdot (n/3^k)^2 + T(1) \cdot n = n^2 \sum_{k=0}^{\log_3 n - 1} 1/3^k + O(n) = O(n^2)$$

# Master theorem (théorème sur les récurrences de partition)

# Master theorem

$T(n) = aT(n/b) + f(n)$, where $a \geq 1$, $b > 1$ are integers, $f(n)$ - asymptotically positive.

Define $r := \log_b a$.

1. If $f(n) = O(n^{r-\varepsilon})$ for some $\varepsilon > 0$, then $T(n) = \Theta(n^r)$.

2. If $f(n) = \Theta(n^r)$, then $T(n) = \Theta(n^r \log n)$.

3. If $f(n) = \Omega(n^{r+\varepsilon})$ for some $\varepsilon > 0$ and $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large $n$ (<u>regularity condition</u>), then $T(n) = \Theta(f(n))$.

**Example.** $T(n) = 7T(n/2) + \Theta(n^2)$

$f(n) = O(n^{\log_2 7 - \varepsilon})$, therefore $T(n) = \Theta(n^{\log_2 7}) = O(n^{2.8704})$

# Master theorem

$T(n) = aT(n/b) + f(n)$, where $a \geq 1, b > 1$ are integers, $f(n)$ - asymptotically positive.

Define $r := \log_b a$.

$n/b$ can be $\lceil n/b \rceil$, or $\lfloor n/b \rfloor$

1. If $f(n) = O(n^{r-\varepsilon})$ for some $\varepsilon > 0$, then $T(n) = \Theta(n^r)$.
2. If $f(n) = \Theta(n^r)$, then $T(n) = \Theta(n^r \log n)$.
3. If $f(n) = \Omega(n^{r+\varepsilon})$ for some $\varepsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large $n$, then $T(n) = \Theta(f(n))$.

**Example.** $T(n) = 7T(n/2) + \Theta(n^2)$

$f(n) = O(n^{\log_2 7-\varepsilon})$, therefore $T(n) = \Theta(n^{\log_2 7}) = O(n^{2.8704})$

# Master theorem

$T(n) = aT(n/b) + f(n)$, where $a \geq 1$, $b > 1$ are integers, $f(n)$ - asymptotically positive.

Define $r := \log_b a$.

1. If $f(n) = O(n^{r-\varepsilon})$ for some $\varepsilon > 0$, then $T(n) = \Theta(n^r)$.

2. If $f(n) = \Theta(n^r)$, then $T(n) = \Theta(n^r \log n)$.

3. If $f(n) = \Omega(n^{r+\varepsilon})$ for some $\varepsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large $n$, then $T(n) = \Theta(f(n))$.

**NB!** Master theorem does not cover all possible cases for $f(n)$.

# Master theorem

**Our plan.**

1. Analyse the recurrence $T(n) = aT(n/b) + f(n)$ assuming that T is defined over reals, not integers (“continuos” Master theorem, no floors or ceilings in the recurrence)

2. Prove “discrete” Master theorem

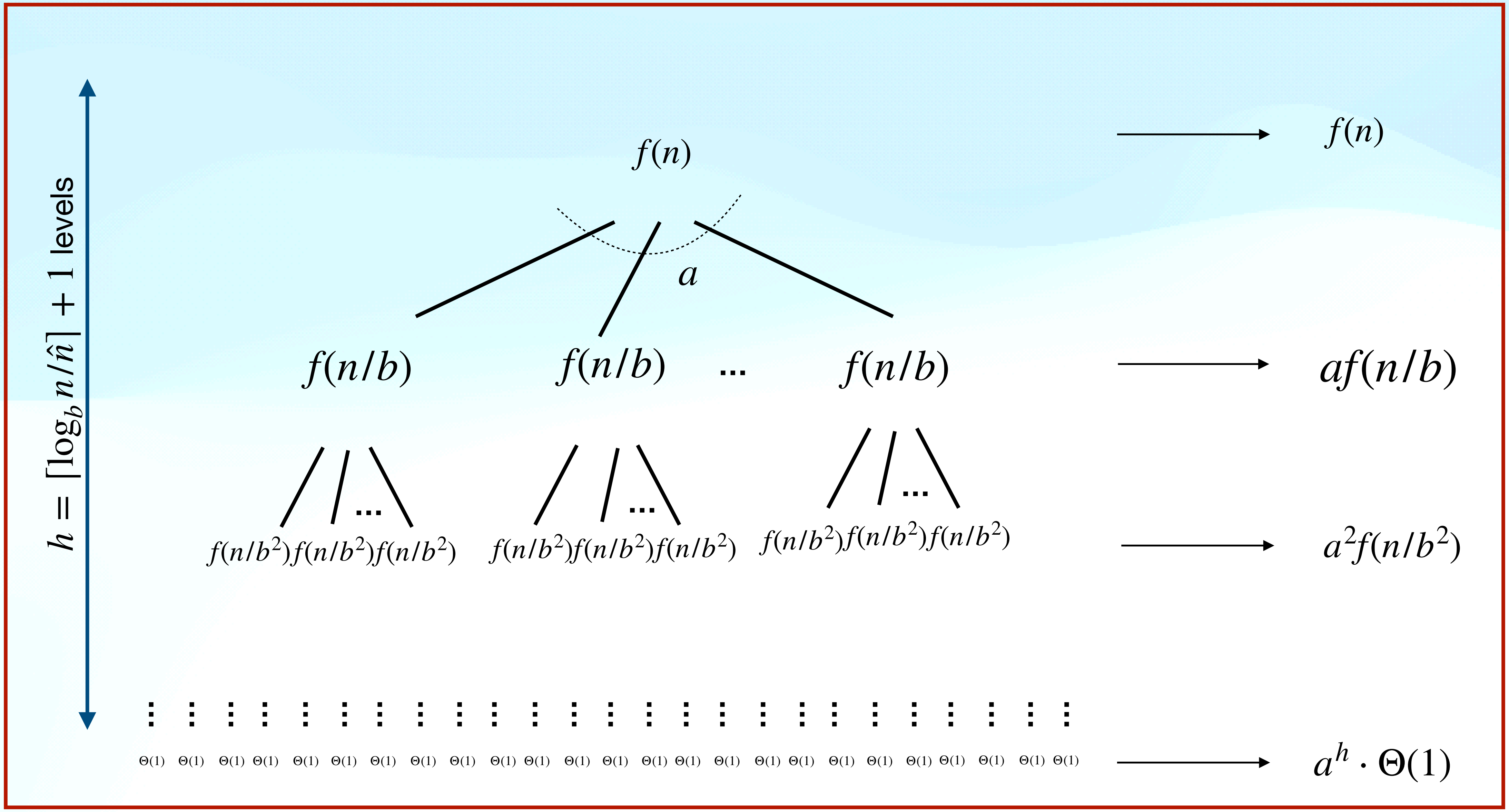
# Master theorem

**Lemma.** Define

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \leq \hat{n} \\ aT(n/b) + f(n) & \text{if } n > \hat{n} \end{cases}$$

Then $T(n) = \Theta(n^{\log_b a}) + \sum_{k=0}^{\lceil \log_b n/\hat{n} \rceil - 1} a^k f(n/b^k)$.

# Master theorem




$$T(n) \le f(n) + af(n/b) + \ldots + a^{h-1}f(n/b^{h-1}) + a^h \cdot \Theta(1) = \Theta(n^{\log_b a}) + \sum_{k=0}^{\lceil \log_b n/\hat{n} \rceil - 1} a^k f(n/b^k)$$

# Master theorem

$q = \lceil \log_b n/\hat{n} \rceil - 1$

**Lemma.** Let $g(n) = \Theta(n^{\log_b a}) + \sum_{k=0}^{q} a^k f(n/b^k)$.

1. If $f(n) = O(n^{\log_b a - \varepsilon})$, then $g(n) = \Theta(n^{\log_b a})$;

2. If $f(n) = \Theta(n^{\log_b a})$, then $g(n) = \Theta(n^{\log_b a} \log n)$;

3. If $f(n) = \Omega(n^{\log_b a + \varepsilon})$ and $af(n/b) \le cf(n)$ for some constant $c < 1$ and for all sufficiently large $n$, then $g(n) = \Theta(f(n))$.

# Master theorem

**Case 1.** $f(n) = O(n^{\log_b a-\varepsilon})$; $g(n) = \Theta(n^{\log_b a}) + \sum_{k=0}^{q} a^k f(n/b^k)$

$q = \lceil \log_b n/\hat{n} \rceil - 1$

$$g(n) = \Theta(n^{\log_b a}) + \sum_{k=0}^{q} a^k f(n/b^k) = \Theta(n^{\log_b a}) + O\left(\sum_{k=0}^{q} a^k (n/b^k)^{\log_b a-\varepsilon}\right)$$

$$\sum_{k=0}^{q} a^k (n/b^k)^{\log_b a-\varepsilon} = n^{\log_b a-\varepsilon} \sum_{k=0}^{q} (ab^\varepsilon / b^{\log_b a})^k =$$

$$n^{\log_b a-\varepsilon} \sum_{k=0}^{\lceil \log_b n/\hat{n} \rceil - 1} (b^\varepsilon)^k = O(n^{\log_b a-\varepsilon}\left(\frac{b^{\varepsilon \log_b n/\hat{n}} - 1}{b^\varepsilon - 1}\right)) = O(n^{\log_b a-\varepsilon}(n/\hat{n})^\varepsilon)$$

Therefore, $g(n) = \Theta(n^{\log_b a})$.

# Master theorem

$q = \lceil \log_b n/\hat{n} \rceil - 1$

**Case 2.** $f(n) = \Theta(n^{\log_b a})$; $g(n) = \Theta(n^{\log_b a}) + \sum_{k=0}^{q} a^k f(n/b^k)$

$$g(n) = \Theta(n^{\log_b a}) + \sum_{k=0}^{q} a^k f(n/b^k) = \Theta(n^{\log_b a}) + \Theta\left(\sum_{k=0}^{q} a^k (n/b^k)^{\log_b a}\right)$$

$$\sum_{k=0}^{q} a^k (n/b^k)^{\log_b a} = n^{\log_b a} \sum_{k=0}^{q} (a/b^{\log_b a})^k = n^{\log_b a} \sum_{k=0}^{\lceil \log_b n/\hat{n} \rceil - 1} 1 = n^{\log_b a} \cdot \Theta(\log_b n/\hat{n})$$

Therefore, $g(n) = \Theta(n^{\log_b a} \log n)$.

# Master theorem

**Case 3.** $f(n) = \Omega(n^{\log_b a+\varepsilon})$, $af(n/b) \leq cf(n)$ **for** $c < 1$,

$$g(n) = \Theta(n^{\log_b a}) + \sum_{k=0}^{q} a^k f(n/b^k)$$

By induction, $a^k f(n/b^k) \leq c^k f(n)$. Hence,

$$\sum_{k=0}^{q} a^k f(n/b^k) \leq \sum_{k=0}^{q} c^k f(n) = f(n) \sum_{k=0}^{q} c^k = \Theta(f(n))$$

(lower bound: obvious, upper bound: geom. series with $c < 1$)

Therefore, $g(n) = \Theta(f(n))$.

# Back to our plan

- We showed the continuous Master theorem
- We now must show that the continuous variant implies the discrete one, where the domain of T are natural numbers and each T(n/b) must be either $T(\lfloor n/b \rfloor)$ or $T(\lceil n/b \rceil)$
- We follow William Kuszmaul, Charles E. Leiserson "Floors and ceilings in divide-and-conquer recurrences", Symposium on Simplicity in Algorithms 2021

# Why not to follow CLRS textbook?

Floors and Ceilings in Divide-and-Conquer Recurrences*

William Kuszmaul
Charles E. Leiserson

MIT CSAIL
{kuszmaul,cel}@mit.edu

**Abstract**

The master theorem is a core tool for algorithm analysis. Many applications use the discrete version of the theorem, in which floors and ceilings may appear within the recursion. Several of the known proofs of the discrete master theorem include substantial errors, however, and other known proofs employ sophisticated mathematics. We present an elementary and approachable proof that applies generally to Akra-Bazzi-style recurrences.

include the claim that the theorem holds in the presence of floors and ceilings.

To distinguish the two situations, we call the master theorem without floors and ceilings the ***continuous master theorem***[1] and the master theorem with floors and ceilings the ***discrete master theorem***. When we speak only of the master theorem, we mean the discrete master theorem, but we usually include the term "discrete" in this paper for clarity in distinguishing the two cases.

proved the theorem for exact powers of $b$. Cormen, Leiserson, and Rivest [5, Section 4.3] presented the discrete master theorem, extending Bentley, Haken, and Saxe's earlier treatment to include floors and ceilings, but their proof is at best a sketch, not a rigorous argument, and it leaves key issues unaddressed. These problems have persisted through two subsequent editions [6, 7] with the additional coauthor Stein.

# Why not to follow CLRS textbook?

- Aho, Hopcroft, Ullman offered one of the first treatments of divide-and-conquer recurrences, giving three cases for recurrences of the form T(n) = aT (n/b) + cn (1974)
- Bentley, Haken, and Saxe introduced the master theorem in modern form, but proved it for $n = b^k$ only (1980)
- CLRS extended the proof to the discrete version, but gave only a sketch of the proof (1990)
- Akra and Bazzi considered $T(n) = \sum_{i=1}^{t} a_i T(n/b_i) + f(n)$ (1998)
- Leighton simplifies the proof of Akra and Bazzi and extends is to the discrete version (1996)
- Campbell spots several flows in the proof of Leighton and devotes more than 300 pages to carefully correct the issues (2020)
- More generalizations by Drmota and Szpankowski(2013), Roura (2001), Yap (2011)

# Definitions

**Discrete recurrences**

$$T(n) = f(n) + \sum_{i \in S} a_i T(\lfloor n/b_i \rfloor) + \sum_{i \notin S} a_i T(\lceil n/b_i \rceil)$$

$$a_i \in \mathbb{R}^+, b_i \in \mathbb{R}^+, n \geq \hat{n}$$

For $1 \leq n < \hat{n}$, there exist $c_1, c_2$: $c_1 \leq T(n) \leq c_2$

**Polynomial-growth condition**

$\exists \hat{n} > 0$ such that $\forall \Phi \geq 1 \, \exists d > 1 : d^{-1} f(n) \leq f(\varphi n) \leq d f(n)$

for all $1 \leq \varphi \leq \Phi$ and $n \geq \hat{n}$

# 6 technical slides ahead!

# KEEP CALM AND CARRY ON

**Lemma 1.** For $\beta > 1, n \in \mathbb{N}$ let $L = \Pi_{i=1}^{n}(1 - \frac{1}{\beta^i + 1})^2$, $U = \Pi_{i=1}^{n}(1 + \frac{1}{\beta^i - 1})^2$

We have $L = \Omega(1)$ and $U = O(1)$.

**Proof.**

$$\beta > 1 \Rightarrow \frac{1}{\beta^i} < \frac{1}{\beta^i - 1} \Rightarrow 1/L = \Pi_{1=1}^{n}(1 + \frac{1}{\beta^i})^2 < \Pi_{i=1}^{n}(1 + \frac{1}{\beta^i - 1})^2 = U$$

$$U = \Pi_{i=1}^{n}(1 + \frac{1}{\beta^i - 1})^2 \leq \Pi_{i=1}^{\infty}(1 + \frac{1}{\beta^i - 1})^2 \leq \Pi_{i=1}^{\infty}(e^{1/(\beta^i - 1)})^2 =$$

(Here we use $1 + 1/x \leq e^{1/x}$ for $x \neq 0$)

$$= \exp(\sum_{i=1}^{\infty} \frac{2}{\beta^i - 1}) \leq \exp(\sum_{i=1}^{\infty} \frac{4}{\beta^i}) + O(1) = O(1)$$

**Lemma 2.** Let $\beta > 1$; $\beta_i \geq \beta, 1 \leq i \leq k$; $B := \Pi_{i=1}^{k} \beta_i$

There exists $c = c(\beta) > 0$ such that for all $n_1, n_2, \ldots, n_k$ where $n_i > \max(\beta, 1 + 1/(\sqrt{\beta} - 1))$ and $\lfloor n_{i-1}/\beta_i \rfloor \leq n_i \leq \lceil n_{i-1}/\beta_i \rceil$, we have $c^{-1/4}(n_0/B) \leq n_k \leq c^{1/4}(n_0/B)$.

**Proof.** Let $\rho_i := \dfrac{n_i}{n_{i-1}/\beta_i}$.

$$(n_0/B)\Pi_{i=1}^{k}\rho_i = \frac{n_0 \Pi_{i=1}^{k}\rho_i}{\Pi_{i=1}^{k}\beta_i} = n_0 \Pi_{i=1}^{k} \frac{\rho_i}{\beta_i} = n_0 \Pi_{i=1}^{k} \frac{n_i}{n_{i-1}} = n_k$$

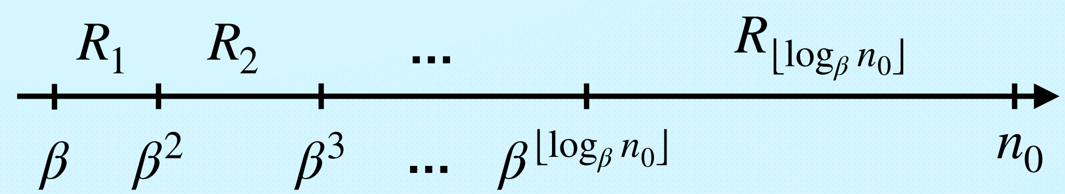
It is enough to show that $c^{-1/4} \leq \Pi_{i=1}^{k}\rho_i \leq c^{1/4}$ for some $c = c(\beta)$

$$n_{i-1}/\beta_i - 1 \leq \lfloor n_{i-1}/\beta_i \rfloor \leq n_i \leq \lceil n_{i-1}/\beta_i \rceil \leq n_{i-1}/\beta_i + 1 \Rightarrow$$

$$n_i - 1 \leq n_{i-1}/\beta_i \leq n_i + 1 \Rightarrow \boxed{\underbrace{\frac{n_i}{n_i + 1}}_{1 - \frac{1}{n_i + 1}} \leq \rho_i \leq \underbrace{\frac{n_i}{n_i - 1}}_{1 + \frac{1}{n_i - 1}} \; (*)}$$

**Proof of Lemma 2 (continued).**

$$\boxed{\underbrace{\frac{n_i}{n_i+1}}_{1-\frac{1}{n_i+1}} \le \rho_i \le \underbrace{\frac{n_i}{n_i-1}}_{1+\frac{1}{n_i-1}} \ (*)}$$




From $(*)$: $\rho_i \le 1 + \dfrac{1}{n_i - 1} \le 1 + \dfrac{1}{1/(\sqrt{\beta}-1)} = \sqrt{\beta}$

$n_{i+2} = \dfrac{n_i \rho_{i+1} \rho_{i+2}}{\beta_{i+1}\beta_{i+2}} \le n_i/\beta \Rightarrow$ every range $R_j$ contains at most two $n_i$'s

From $(*)$ again: $n_i \in R_j \Rightarrow 1 - \dfrac{1}{\beta^j + 1} \le \rho_i \le 1 + \dfrac{1}{\beta^j - 1} (n_i > \beta^j)$

Therefore, $\Pi_{i=1}^{k} \rho_i = \Pi_{j=1}^{\lfloor \log_\beta n_0 \rfloor} (\Pi_{n_i \in R_j} \rho_i) \le \Pi_{j=1}^{\lfloor \log_\beta n_0 \rfloor} (1 + \dfrac{1}{\beta^j - 1})^2 \le c^{1/4}$ **(Lemma 1)**

$$\Pi_{i=1}^{k} \rho_i \ge \Pi_{j=1}^{\lfloor \log_\beta n_0 \rfloor} (1 - \frac{1}{\beta^j + 1})^2 \ge c^{-1/4} \text{ (Lemma 1)}$$

**Lemma 3.** $\beta_{\min}, \beta_{\max} > 1$. Assume that for all $1 \le i \le k$, $\beta_{\min} \le \beta_i \le \beta_{\max}$, and let $B = \Pi_i \beta_i$.

There exists $c = c(\beta_{\min}, \beta_{\max})$ such that for any $n_1, n_2, \ldots, n_k$ with $n_0 \ge cB$ and $\lfloor n_{i-1}/\beta_i \rfloor \le n_i \le \lceil n_{i-1}/\beta_i \rceil$, we have $c^{-1}(n_0/B) \le n_k \le c(n_0/B)$.

**Proof.**

Let $c = c(\beta_{\min})$ be the constant from Lemma 3. W.l.o.g. $\sqrt{c} > \max\{\frac{1}{\sqrt{\beta_{\min}} - 1} + 1, \beta_{\min}\}$ $(*)$ and $c^{1/4} > 2\beta_i$

If $n_j \ge \sqrt{c}$ for all $j$, then **Lemma 3** follows from **Lemma 2** and $(*)$. Let $j$ be the smallest value such that $n_j \le \sqrt{c}$. We have $j \ge 1$ as $n_0 \ge cB \ge \sqrt{c}$.

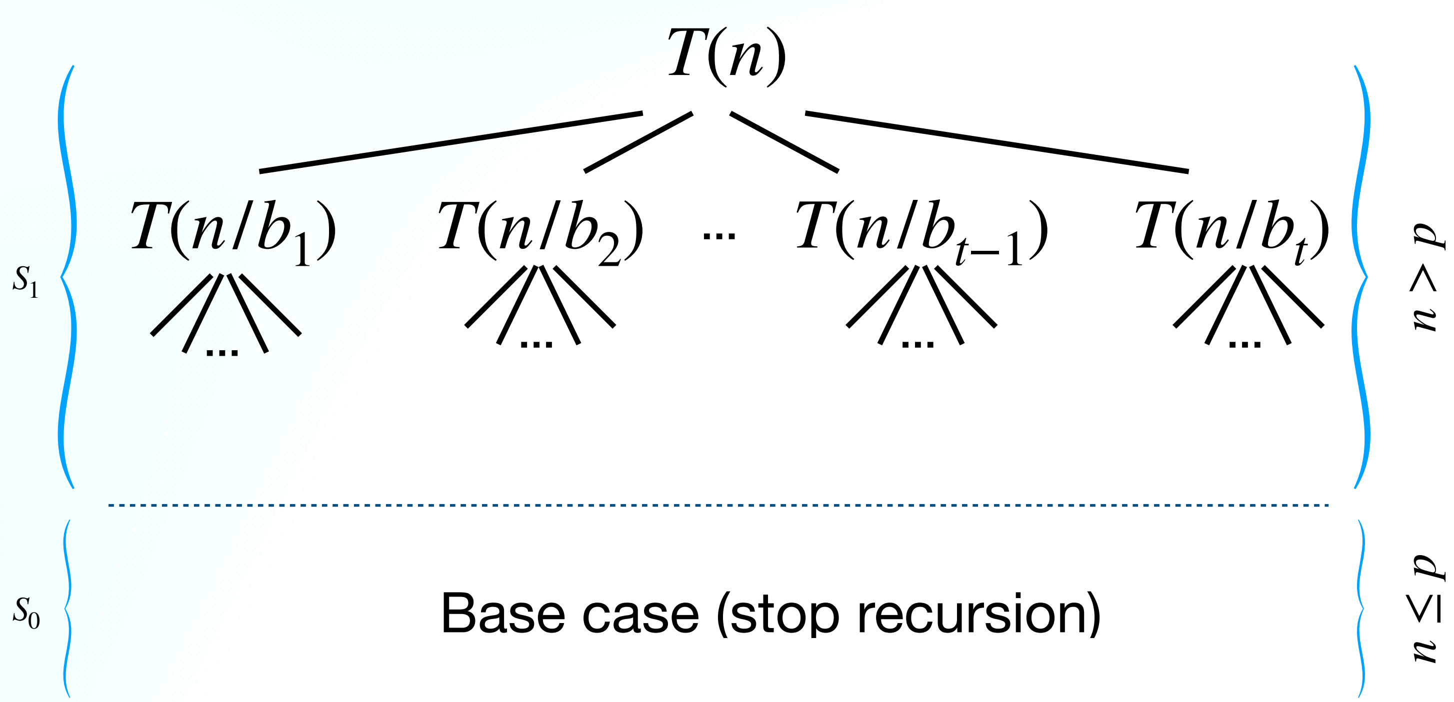
- If $j = 1$, then $n_{j-1} = n_0 \ge c^{-1/4}(n_0/B)$ (trivial).
- If $j > 1$, we apply **Lemma 2** to $\beta_1, \beta_2, \ldots, \beta_{j-1}$ and $n_0, n_1, \ldots, n_{j-1}$ and $\beta = \beta_{\min}$ (all conditions are satisfied) to obtain that $n_{j-1} \ge c^{-1/4}(n_0/B)$

In both cases, $n_{j-1} \ge c^{-1/4}(n_0/B) \ge c^{3/4}$. Therefore, $n_j \ge \lfloor \underbrace{n_{j-1}}_{\ge c^{1/4} > 2\beta_j} / \beta_j \rfloor \ge n_{j-1}/(2\beta_j) > n_{j-1}/c^{1/4} \ge \sqrt{c}$

**Lemma 4.** Let $a_1, a_2, \ldots, a_t > 0$ and $b_1, b_2, \ldots, b_t > 1$ be constants, $f(n) : \mathbb{R}^+ \to \mathbb{R}^+$ which satisfies the polynomial-growth condition. Consider $T(n) = f(n) + \sum_{i=1}^{t} a_i T(n/b_i)$ defined for $n \in \mathbb{R}^+$ ( $*$ ). Assume that $T'(n)$ defined on $\mathbb{N}$ also satisfies ( $*$ ), but each $n/b_i$ is replaced with $\lfloor n/b_i \rfloor$ or $\lceil n/b_i \rceil$. Then $T'(n) = \Theta(T(n))$.

**Proof.**

Let $c$ be the constant from Lemma 3 for $\beta_{\min} = \min_i b_i$ and $\beta_{\max} = \max_i b_i$. Let $\hat{n}$ be a sufficiently large constant. Define $p := \max\{\hat{n}, c \cdot \max_i b_i\}$. For $T(n)$, the base case is $n \leq p$.




Subproblem $q$ at level $j$:
$q = < q(1), q(2), \ldots, q(j) >$
$q(i) \in \{1,2,\ldots,t\}, |q| = j$
$q(i)$ - choice of the recursion brunch at level $i$
$n_q = n/\Pi_{i=1}^{|q|} b_{q(i)}$

**Proof.**

$$T(n) = \sum_{q \in S_1} f(n_q)\Pi_{i=1}^{|q|} a_{q(i)} + \sum_{q \in S_0} T(n_q)\Pi_{i=1}^{|q|} a_{q(i)} + f(n) = \sum_{q \in S_1} f(n_q)\Pi_{i=1}^{|q|} a_{q(i)} + \Theta(\sum_{q \in S_0} \Pi_{i=1}^{|q|} a_{q(i)}) + f(n)$$

When computing $T'(n)$ for a subproblem $q$:

$$\lfloor \frac{n'_{<q(1),q(2),\ldots,q(j-1)>}}{q(j)} \rfloor \le n'_q \le \lceil \frac{n'_{<q(1),q(2),\ldots,q(j-1)>}}{q(j)} \rceil$$

$$T'(n) = \sum_{q \in S_1} f(n'_q)\Pi_{i=1}^{|q|} a_{q(i)} + \sum_{q \in S_0} T'(n'_q)\Pi_{i=1}^{|q|} a_{q(i)} + f(n) \; (*)$$

As $n_q > p$ for $q \in S_1$, $n_q > p/\max_i b_i \ge c$ for all $q \in S$. By Lemma 3 with $\beta_i = b_{q(i)}$, for all $q$ we have $n'_q = \Theta(n_q)$. It follows that $\exists \Phi > 1$ such that $n'_q \in [\Phi^{-1} n_q, \Phi n_q]$. Therefore, $n'_q \ge n_q/\Phi \ge \hat{n}/\Phi$ and we can choose $\hat{n}$ so that $(*)$ is defined correctly.

By the polynomial-growth condition, $f(n'_q) = \Theta(f(n_q))$ for all $q \in S$. For $q \in S_0$, $n'_q = \Theta(1)$ and therefore $T'(n'_q) = \Theta(1)$. It follows:

$$T'(n') = \sum_{q \in S_1} \Theta(f(n_q))\Pi_{i=1}^{|q|} a_{q(i)} + \Theta(\sum_{q \in S_0} \Pi_{i=1}^{|q|} a_{q(i)}) + f(n) = \Theta(T(n))$$

**Proof.**

$$T(n) = \sum_{q\in S_1} f(n_q)\Pi_{i=1}^{|q|} a_{q(i)} + \sum_{q\in S_0} T(n_q)\Pi_{i=1}^{|q|} a_{q(i)} + f(n) = \sum_{q\in S_1} f(n_q)\Pi_{i=1}^{|q|} a_{q(i)} + \Theta(\sum_{q\in S_0} \Pi_{i=1}^{|q|} a_{q(i)}) + f(n)$$

When computing $T'(n)$ for a subproblem $q$:

$$\lfloor \frac{n'_{<q(1),q(2),\ldots,q(j-1)>}}{q(j)} \rfloor \leq n'_q \leq \lceil \frac{n'_{<q(1),q(2),\ldots,q(j-1)>}}{q(j)} \rceil$$

$$T'(n) = \sum_{q\in S_1} f(n'_q)\Pi_{i=1}^{|q|} a_{q(i)} + \sum_{q\in S_0} T'(n'_q)\Pi_{i=1}^{|q|} a_{q(i)} + f(n) \ (*)$$

As $n_q > p$ for $q \in S_1$, $n_q > p/\max_i b_i \geq c$ for all $q \in S$. By Lemma 3 with $\beta_i = b_{q(i)}$, for all $q$ we have $n'_q = \Theta(n_q)$, and hence $\exists \Phi > 1$ such that $n'_q \in [\Phi^{-1} n_q, \Phi n_q]$. Therefore, $n'_q \geq n_q/\Phi \geq \hat{n}/\Phi$ and we can choose $\hat{n}$ so that $(*)$ is defined correctly.

By the polynomial-growth condition, $f(n'_q) = \Theta(f(n_q))$ for all $q \in S$. For $q \in S_0$, $n'_q = \Theta(1)$ and therefore $T'(n'_q) = \Theta(1)$. It follows:

$$T'(n') = \sum_{q\in S_1} \Theta(f(n_q))\Pi_{i=1}^{|q|} a_{q(i)} + \Theta(\sum_{q\in S_0} \Pi_{i=1}^{|q|} a_{q(i)}) + f(n) = \Theta(T(n))$$

# Discrete Master theorem

$T(n) = a_1 T(\lfloor n/b \rfloor) + a_2 T(\lceil n/b \rceil) + f(n)$, where
$a := a_1 + a_2 \geq 1$, $b > 1$, $f(n)$ - asymptotically positive.

Define $r := \log_b a$.

**Case 1.** If $f(n) = O(n^{r-\varepsilon})$ for some $\varepsilon > 0$, then $T(n) = \Theta(n^r)$.

**Case 2.** If $f(n) = \Theta(n^r)$, then $T(n) = \Theta(n^r \log n)$.

**Case 3.** If $f(n) = \Omega(n^{r+\varepsilon})$ for some $\varepsilon > 0$, and if
$a_1 f(\lfloor n/b \rfloor) + a_2 f(\lceil n/b \rceil) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large $n$, then $T(n) = \Theta(f(n))$.

# Discrete Master theorem

**Case 1.**

**Fact.** Replacing $f(n)$ with a function $f'(n)$ satisfying $f'(n) \leq f(n)$ (resp. $f'(n) \geq f(n)$) for all $n$ in the domain of $f$ does not increase (resp. decrease) $T(n)$.

Let $f(n) = O(n^c)$ for $c < \log_b a$. Then as a "bigger" function consider $f'(n) = r(n^c + 1)$ for $r$ big enough. By Lemma 4 and the continuos Master theorem, $T(n) = O(n^{\log_b a})$.

As a "smaller" function, consider $f'(n) = 0$. By Lemma 4 and the continuos Master theorem, $T(n) = \Omega(n^{\log_b a})$.

**Exercise.** Both bigger and smaller functions satisfy the polynomial growth condition.

**Case 2.** Analogous.

# Discrete Master theorem

**Case 3.**

$T(n) \geq f(n)$ and hence $T(n) = \Omega(f(n))$. It remains to show that $T(n) = O(f(n))$.

**Regularity condition:** $a_1 f(\lfloor n/b \rfloor) + a_2 f(\lceil n/b \rceil) \leq cf(n)$ for some $c < 1$ and all $n \geq p$.

For all $n < p$, there exists $s \geq 1$: $T(n) \leq sf(n)$. We show by induction that for all $n \in \mathbb{N}$, $T(n) \leq qf(n)$ for $q = s/(1-c)$.

- Base case: $n < p$ - by the choice of $s$
- Suppose that $n \geq p$ and the claim holds for all smaller $n$

$$T(n) = a_1 T(\lfloor n/b \rfloor) + a_2 T(\lceil n/b \rceil) + f(n) \leq a_1 q f(\lfloor n/b \rfloor) + a_2 q f(\lceil n/b \rceil) + f(n) \leq$$

$$\leq qcf(n) + f(n) = (\frac{sc}{1-c} + 1)f(n) = \frac{s - \overbrace{(1-c)s + 1 - c}^{\leq 0}}{1-c} f(n) \leq qf(n)$$

# Application: **Merge sort**

$T(1) = const$

$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + c \cdot n$

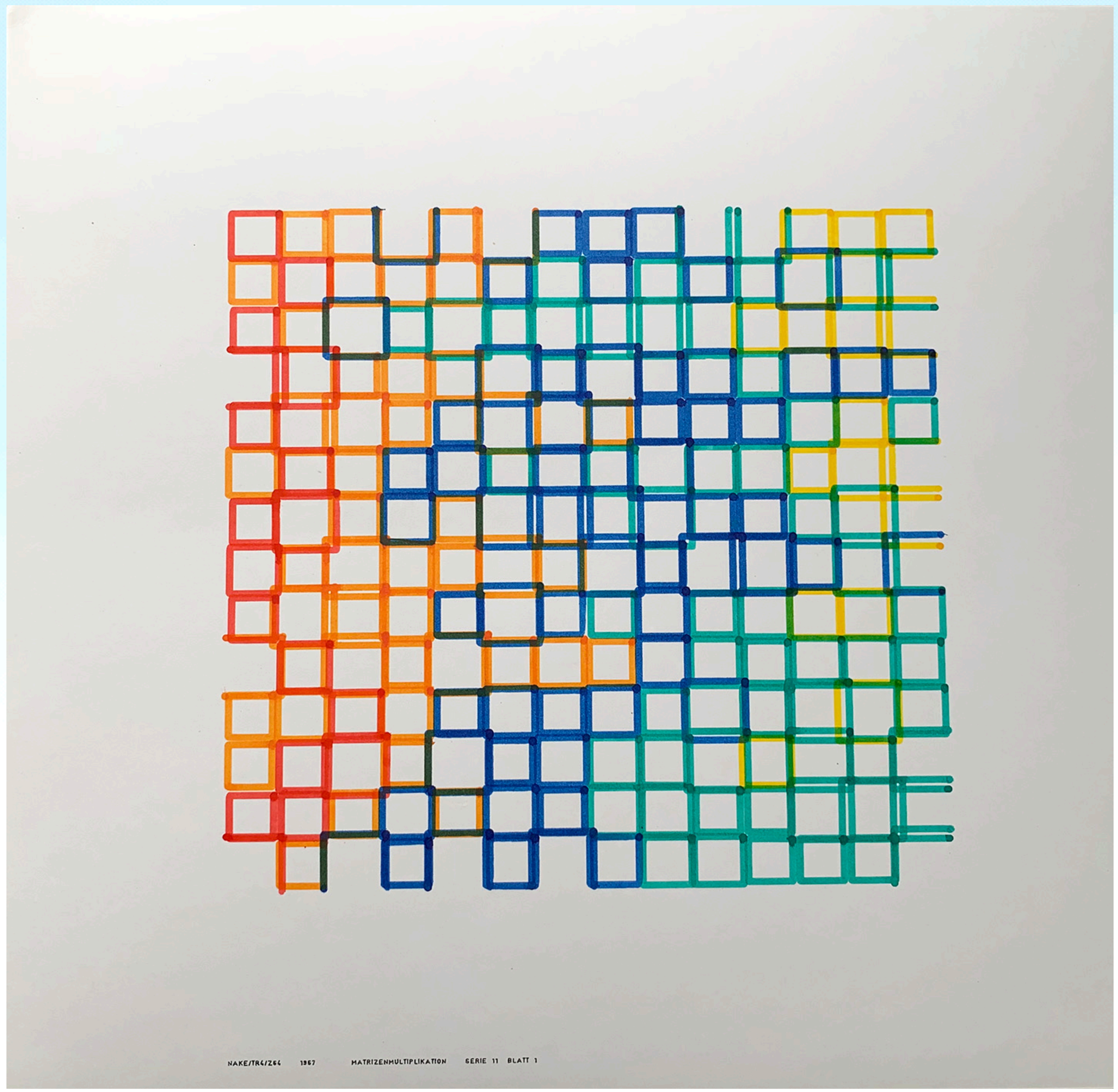
We have $a = 2$, $b = 2$, and $f(n) = \Theta(n^{\log_b a})$. This is case 2 of Master theorem!

Hence, $T(n) = O(n \log n)$.

# Open question

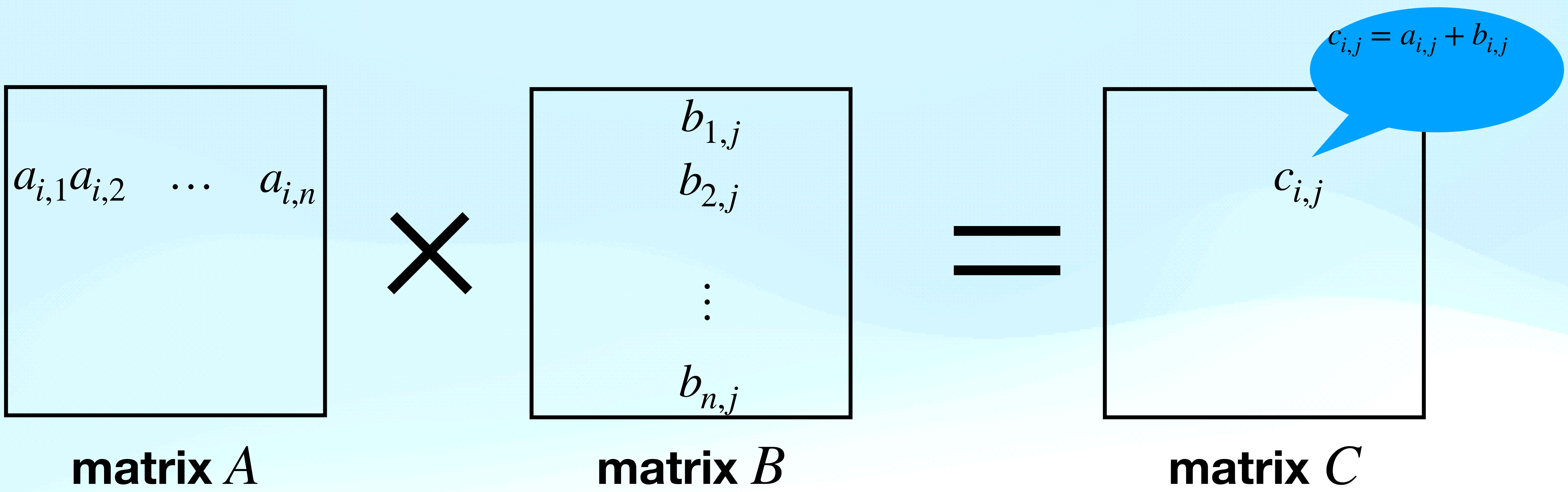
Can the techniques from the paper of Kuszmaul and Leiserston be used to show Lemma 4 for recurrences where each term $T(n/b_i)$ can be replaced with $T(n/b_i + h_i(n))$, and $|h_i(n)| \leq n/\lg^{1+\varepsilon} n$ for some $\varepsilon > 0$?

# Matrix multiplication



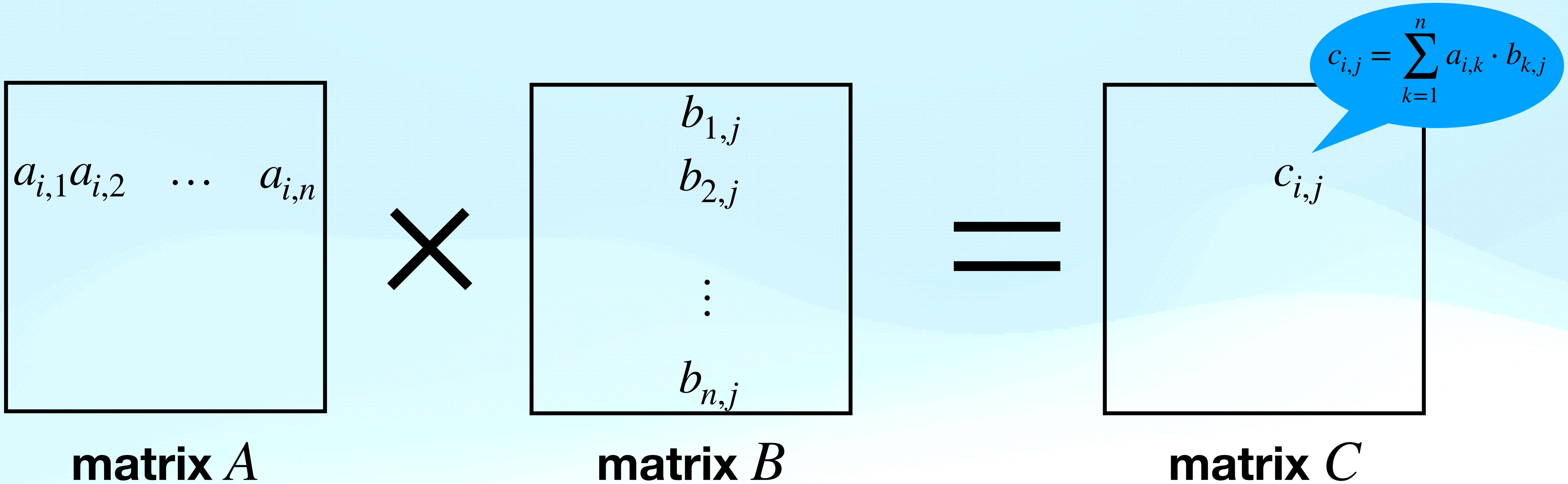

F.Nake, Matrizenmultiplikation Serie 11 Blatt 1

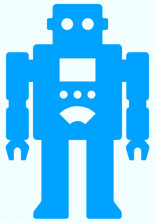
# Matrix sum




- $n^2$ matrix entries to compute
- Each can be computed in $O(1)$ time, takes $O(n^2)$ time in total

# Matrix multiplication



- $n^2$ matrix entries to compute
- Each can be computed in $O(n)$ time, takes $O(n^3)$ time in total
- Essential in many fields, including deep learning
- Current best: $O(n^{2.3728596})$-time algorithm, see Josh Alman, Virginia Vassilevska Williams, "A Refined Laser Method and Faster Matrix Multiplication", SODA 2020
- Open problem: Can we do better?

# Strassen's algorithm

$$\begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix} \times \begin{bmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{bmatrix} = \begin{bmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{bmatrix}$$

**matrix A** **matrix B** **matrix C**

For simplicity, assume $n = 2^k$

**First attempt (doesn't work!)**

$$C_{1,1} = A_{1,1} \cdot B_{1,1} + A_{1,2} \cdot B_{2,1} \qquad C_{1,2} = A_{1,1} \cdot B_{1,2} + A_{1,2} \cdot B_{2,2}$$

$$C_{2,1} = A_{2,1} \cdot B_{1,1} + A_{2,2} \cdot B_{2,1} \qquad C_{2,2} = A_{2,1} \cdot B_{1,2} + A_{2,2} \cdot B_{2,2}$$

# Strassen's algorithm

$$\underbrace{\begin{array}{c|c} A_{1,1} & A_{1,2} \\ \hline A_{2,1} & A_{2,2} \end{array}}_{\textbf{matrix A}} \times \underbrace{\begin{array}{c|c} B_{1,1} & B_{1,2} \\ \hline B_{2,1} & B_{2,2} \end{array}}_{\textbf{matrix B}} = \underbrace{\begin{array}{c|c} C_{1,1} & C_{1,2} \\ \hline C_{2,1} & C_{2,2} \end{array}}_{\textbf{matrix C}}$$

**MatrixMult(A, B, n)**

**if** $n = 1$ **do**

$\quad c_{1,1} \leftarrow a_{1,1} \cdot b_{1,1}$

**else**

$\quad$ Partition $A, B, C$

$\quad C_{1,1} = \text{MatrixMult}(A_{1,1}, B_{1,1}, n/2) + \text{MatrixMult}(A_{1,2}, B_{2,1}, n/2)$

$\quad C_{1,2} = \text{MatrixMult}(A_{1,1}, B_{1,2}, n/2) + \text{MatrixMult}(A_{1,2}, B_{2,2}, n/2)$

$\quad C_{2,1} = \text{MatrixMult}(A_{2,1}, B_{1,1}, n/2) + \text{MatrixMult}(A_{2,1}, B_{2,1}, n/2)$

$\quad C_{2,2} = \text{MatrixMult}(A_{2,1}, B_{1,2}, n/2) + \text{MatrixMult}(A_{2,2}, B_{2,2}, n/2)$

**return** $C$

$$T(n) = 8T(n/2) + \Theta(n^2)$$

**Exercise.** $T(n) = \Theta(n^3)$

# Strassen's algorithm

$$\underset{\textbf{matrix A}}{\begin{array}{c|c} A_{1,1} & A_{1,2} \\ \hline A_{2,1} & A_{2,2} \end{array}} \times \underset{\textbf{matrix B}}{\begin{array}{c|c} B_{1,1} & B_{1,2} \\ \hline B_{2,1} & B_{2,2} \end{array}} = \underset{\textbf{matrix C}}{\begin{array}{c|c} C_{1,1} & C_{1,2} \\ \hline C_{2,1} & C_{2,2} \end{array}}$$

**Second attempt**

$$M_1 = (A_{1,1} + A_{2,2})(B_{1,1} + B_{2,2})$$
$$M_2 = (A_{2,1} + A_{2,2}) \cdot B_{1,1}$$
$$M_3 = A_{1,1} \cdot (B_{1,2} - B_{2,2})$$
$$M_4 = A_{2,2} \cdot (B_{2,1} - B_{1,1})$$
$$M_5 = (A_{1,1} + A_{1,2}) \cdot B_{2,2}$$
$$M_6 = (A_{2,1} - A_{1,1})(B_{1,1} + B_{1,2})$$
$$M_7 = (A_{1,2} - A_{2,2})(B_{2,1} + B_{2,2})$$

$$C_{1,1} = A_{1,1} \cdot B_{1,1} + A_{1,2} \cdot B_{2,1} = M_1 + M_4 - M_5 + M_7$$
$$C_{1,2} = A_{1,1} \cdot B_{1,2} + A_{1,2} \cdot B_{2,2} = M_3 + M_5$$
$$C_{2,1} = A_{2,1} \cdot B_{1,1} + A_{2,2} \cdot B_{2,1} = M_2 + M_4$$
$$C_{2,2} = A_{2,1} \cdot B_{2,1} + A_{2,2} \cdot B_{2,2} = M_1 - M_2 + M_3 + M_6$$

$$\boxed{\begin{aligned} T(n) &= 7T(n/2) + \Theta(n^2) \\ T(n) &= O(n^{2.8074}) \end{aligned}}$$

# Strassen's algorithm

$$\begin{array}{|c|c|} \hline A_{1,1} & A_{1,2} \\ \hline A_{2,1} & A_{2,2} \\ \hline \end{array} \times \begin{array}{|c|c|} \hline B_{1,1} & B_{1,2} \\ \hline B_{2,1} & B_{2,2} \\ \hline \end{array} = \begin{array}{|c|c|} \hline C_{1,1} & C_{1,2} \\ \hline C_{2,1} & C_{2,2} \\ \hline \end{array}$$

**matrix A** **matrix B** **matrix C**

## Second attempt

$M_1 = (A_{1,1} + A_{2,2})(B_{1,1} + B_{2,2})$

$M_2 = (A_{2,1} + A_{2,2}) \cdot B_{1,1}$

$M_3 = A_{1}$ …

$M_4 =$ …

$M_5 = (A_{1,1}$ …

$M_6 = (A_{2,1} - A_{1,1})(B_{1,1} + B_{1,2})$

$M_7 = (A_{1,2} - A_{2,2})(B_{2,1} + B_{2,2})$

$C_{1,1} = A_{1,1} \cdot B_{1,1} + A_{1,2} \cdot B_{2,1} = M_1 + M_4 - M_5 + M_7$

$C_{1,2} = A_{1,1} \cdot B_{1,2} + A_{1,2} \cdot B_{2,2} = M_3 + M_5$

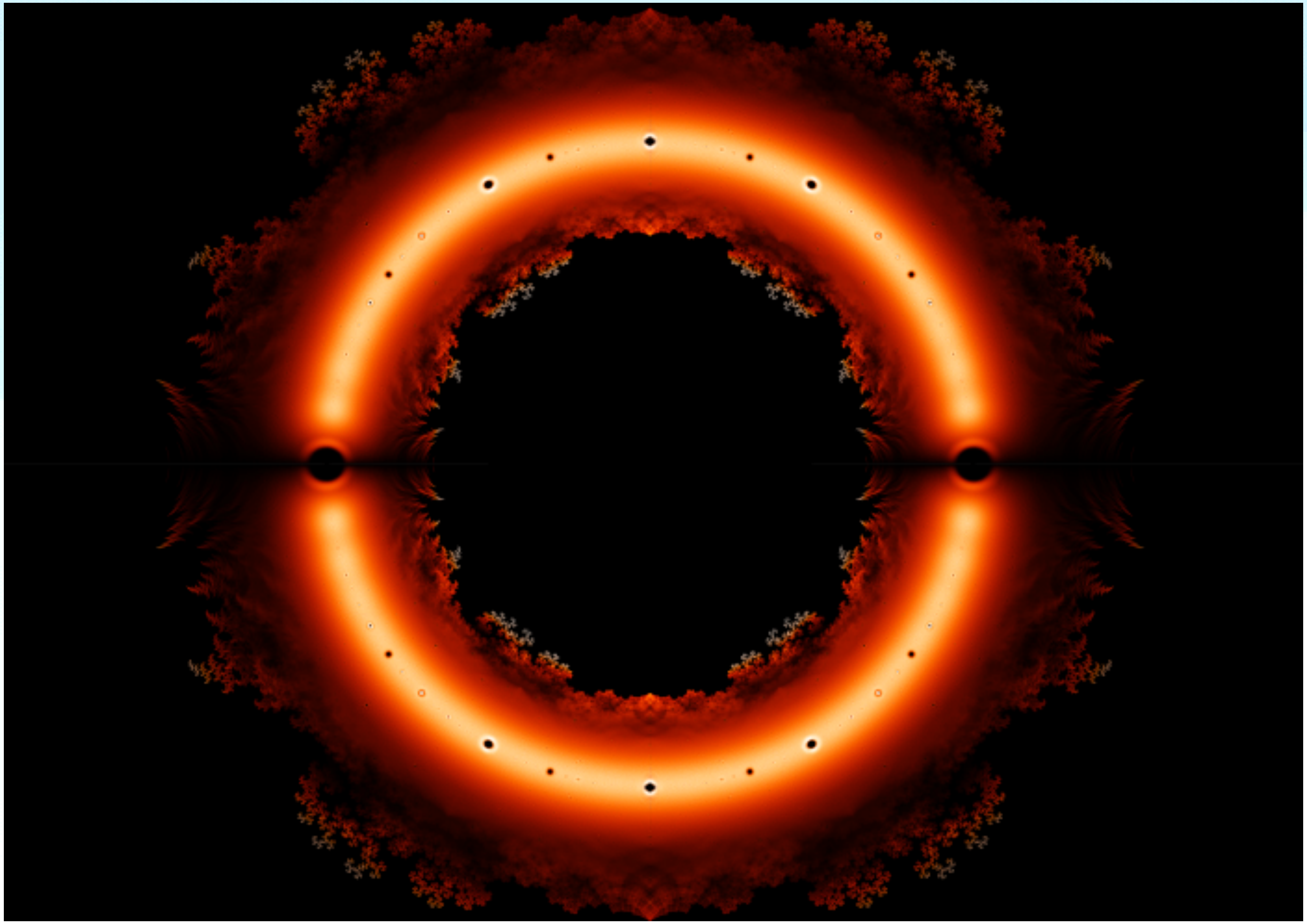
$C_{2,1} = A_{2,1} \cdot B_{1,1} + A_{2,2} \cdot B_{2,1} = M_2 + M_4$

$C_{2,2} = A_{2,1} \cdot B_{2,1} + A_{2,2} \cdot B_{2,2} = M_1 - M_2 + M_3 + M_6$

We assumed $n = 2^k$.
Show that it's OK.

$$\boxed{\begin{array}{l} T(n) = 7T(n/2) + \Theta(n^2) \\ T(n) = O(n^{2.8074}) \end{array}}$$

# Fast multiplication of polynomials





Sam Derbyshire. Roots of polynomials with $\pm 1$ coefficients.

# Fast multiplication of polynomials

Consider $P(x) = \sum_{i=0}^{n-1} a_i x^i$, $Q(x) = \sum_{i=0}^{n-1} b_i x^i$

We can compute $R_1(x) = P(x) + Q(x) = \sum_{i=0}^{n-1} (a_i + b_i)x^i$ in $O(n)$ time

Horner's rule allows to evaluate $P(x_0)$ in $O(n)$ time
$P(x_0) = (\ldots(((a_{n-1})x_0 + a_{n-2})x_0 + a_{n-3})x_0 + \ldots + a_1)x_0 + a_0$

Computing $R_2(x) = P(x) \cdot Q(x) = \sum_{k=0}^{2n-2} \sum_{i+j=k} (a_i \cdot b_j)x^k$ requires more time

Naively: $O(n^2)$ time

**Fast Fourier Transform:** $O(n \log n)$ time

# Representing polynomials

Polynomials can be represented in two ways:

1. Coefficient representation: $P(x) = \sum_{i=0}^{n-1} a_i x^i$

   - Horner's rule allows to evaluate $P(x_0)$ in $O(n)$ time
     $P(x_0) = (\ldots(((a_{n-1})x_0 + a_{n-2})x_0 + a_{n-3})x_0 + \ldots + a_1)x_0 + a_0$

   - The sum of two polynomials can be computed in $O(n)$ time

2. Point-value representation: $\{(x_0, y_0), (x_1, y_1), \ldots, (x_{n-1}, y_{n-1})\}$ such that $P(x_i) = y_i$

# Fast multiplication of polynomials

1. Coefficient representation: $P(x) = \sum_{i=0}^{n-1} a_i x^i$

2. Point-value representation: $\{(x_0, y_0), (x_1, y_1), \ldots, (x_{n-1}, y_{n-1})\}$ such that $P(x_i) = y_i$

- **Theorem.** For any set $\{(x_0, y_0), (x_1, y_1), \ldots, (x_{n-1}, y_{n-1})\}$, where $x_i \neq x_j$, there exists a unique polynomial of degree $< n$ such that $P(x_i) = y_i$ for all $i = 0,\ldots, n-1$. (Proof on the next slide!)

- Given point-value representations $\{(x_0, y_0), (x_1, y_1), \ldots, (x_{n-1}, y_{n-1})\}$ of $P(x)$ and $\{(x_0, y'_0), (x_1, y'_1), \ldots, (x_{n-1}, y'_{n-1})\}$ of $P(x)$ and of $Q(x)$, one can compute the point-value representation of $P(x) + Q(x)$ or $P(x) \cdot Q(x)$ in $O(n)$ time

# Fast multiplication of polynomials

**Theorem.** For any set $\{(x_0, y_0), (x_1, y_1), \ldots, (x_{n-1}, y_{n-1})\}$, where $x_i \neq x_j$, there exists a unique polynomial of degree $< n$ such that $P(x_i) = y_i$ for all $i = 0, \ldots, n-1$.

**Proof.**

Let $P(x) = \sum_{i=0}^{n-1} a_i x^i$. We can represent the condition $P(x_i) = y_i$ for all $i = 0, \ldots, n-1$ in the matrix form (subtlety: $x_i, y_j \in \mathbb{C}$):

Vandermonde matrix,
determinant $= \Pi_{0 \leq i < j \leq n-1}(x_j - x_i)$

$$\begin{pmatrix} 1 & x_0 & x_0^2 & \ldots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \ldots & x_1^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \ldots & x_{n-1}^{n-1} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{pmatrix}$$

# Fast multiplication of polynomials

coefficient representation

$a_{n-1}, \ldots, a_1, a_0$
$b_{n-1}, \ldots, b_1, b_0$

naive multiplication

$O(n^2)$ time

$c_{2n-2}, \ldots, c_1, c_0$

FFT

$\Theta(n \log n)$ time

inverse FFT

$\Theta(n \log n)$ time

point-value representation

$P(w_{2n}^0), Q(w_{2n}^0)$
$P(w_{2n}^1), Q(w_{2n}^1)$
$\vdots$
$P(w_{2n}^{2n-1}), Q(w_{2n}^{2n-1})$

point-wise mult.

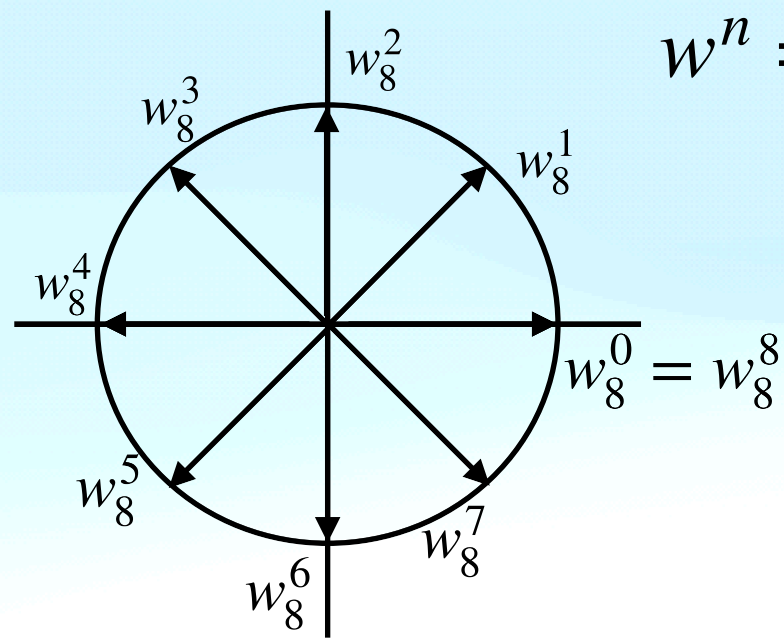
$\Theta(n)$ time

$R(w_{2n}^0)$
$R(w_{2n}^1)$
$\vdots$
$R(w_{2n}^{2n-1})$

$w_{2n}$ - $(2n)$-th complex root of unity

# Complex roots of unity

$$w^n = 1$$




$$w_n^k = e^{2\pi i k/n} = \cos 2\pi k/n + i \sin 2\pi k/n$$

**Halving property:**

$$\{(w_{2n}^0)^2, (w_{2n}^1)^2, \ldots, (w_{2n}^{2n-1})^2\} = \{w_n^0, w_n^1, \ldots, w_n^{n-1}\}$$

**Cancellation property:** $w_{dn}^{dk} = w_n^k$

**Summation property:** $\sum_{j=0}^{n-1} (w_n^k)^j = 0$ for all $k \neq 0 \pmod{n}$

# Complex roots of unity

$w_n^k = e^{2\pi ik/n} = \cos 2\pi k/n + i \sin 2\pi k/n$

**Cancellation:** $w_{dn}^{dk} = w_n^k$

$w_{dn}^{dk} = e^{2\pi i(dk/dn)} = e^{2\pi i(k/n)} = w_n^k$

**Summation:** $\sum_{j=0}^{n-1} (w_n^k)^j = 0$ **for all** $k \neq 0 \pmod{n}$

$$\sum_{j=0}^{n-1} (w_n^k)^j = \frac{(w_n^k)^n - 1}{w_n^k - 1} = \frac{(w_n^n)^k - 1}{w_n^k - 1} = 0$$

**Halving:** $\{(w_{2n}^0)^2, (w_{2n}^1)^2, \ldots, (w_{2n}^{2n-1})^2\} = \{w_n^0, w_n^1, \ldots, w_n^{n-1}\}$

$$(w_{2n}^j)^2 = w_{2n}^{2j} = w_n^j = \begin{cases} w_n^j, & j \leq n-1 \\ w_n^{j-n}, & j \geq n \end{cases}$$

**If** $j \geq n$, $w_n^j = e^{2\pi i(j/n)} = e^{2\pi i(1+(j-n)/n)} = \underbrace{e^{2\pi i}}_{=1} \cdot e^{2\pi i(j-n)/n} = w_n^{j-n}$

# Fast Fourier Transform

$P(x) = \sum_{i=0}^{i=n-1} a_i x^i \rightarrow$ discrete Fourier transform $\{P(w_n^0), P(w_n^1), \ldots, P(w_n^n)\}$

Assumption: $n = 2^j$

$$P(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \ldots + a_1 x + a_0$$

$P_{odd}(x) = a_{n-1}x^{n/2-1} + a_{n-3}x^{n/2-2} + \ldots + a_1$ $\quad$ $P_{even}(x) = a_{n-2}x^{n/2-1} + a_{n-4}x^{n/2-1} + \ldots + a_0$

1. $P(x) = xP_{odd}(x^2) + P_{even}(x^2)$

2. We evaluate $P_{odd}(x)$ and $P_{even}(x)$ at $(w_n^0)^2, (w_n^1)^2, \ldots, (w_n^{n-1})^2$ recursively (by the halving property, $\{(w_n^0)^2, (w_n^1)^2, \ldots, (w_n^{n-1})^2\} = \{(w_{n/2}^0), (w_{n/2}^1), \ldots, (w_{n/2}^{n/2-1})\}$)

3. And combine the results to compute $\{P(w_n^0), P(w_n^1), \ldots, P(w_n^n)\}$

# Fast Fourier Transform

$P(x) = \sum_{i=0}^{i=n-1} a_i x^i \rightarrow$ discrete Fourier transform $\{P(w_n^0),$

$T(n) = 2T(n/2) + \Theta(n)$

Assumption: $n = 2^j$

$$P(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \ldots + a_1 x + a_0$$

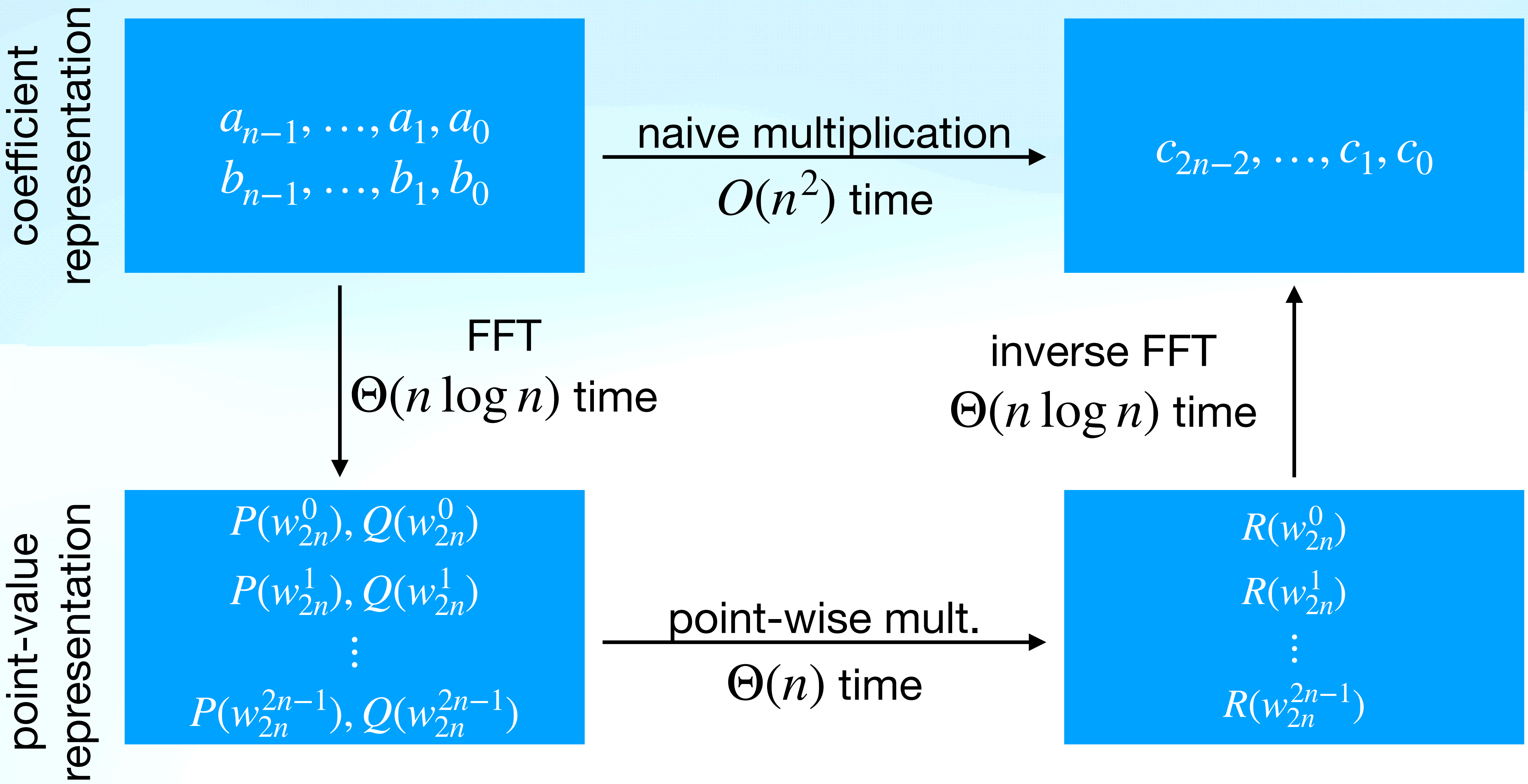
$P_{odd}(x) = a_{n-1}x^{n/2-1} + a_{n-3}x^{n/2-2} + \ldots + a_1$ $\quad$ $P_{even}(x) = a_{n-2}x^{n/2-1} + a_{n-4}x^{n/2-1} + \ldots + a_0$

1. $P(x) = xP_{odd}(x^2) + P_{even}(x^2)$

2. We evaluate $P_{odd}(x)$ and $P_{even}(x)$ at $(w_n^0)^2, (w_n^1)^2, \ldots, (w_n^{n-1})^2$ recursively (by the halving property, $\{(w_n^0)^2, (w_n^1)^2, \ldots, (w_n^{n-1})^2\} = \{(w_{n/2}^0), (w_{n/2}^1), \ldots, (w_{n/2}^{n/2-1})\}$)

3. And sum the results to obtain $\{P(w_n^0), P(w_n^1), \ldots, P(w_n^n)\}$

# Fast multiplication of polynomials




$w_{2n}$ - $(2n)$-th complex root of unity

# Inverse fast Fourier Transform

Discrete Fourier transform $\{P(w_n^0), P(w_n^1), \ldots, P(w_n^n)\} \ \rightarrow P(x) = \sum_{i=0}^{i=n-1} a_i x^i$

$$\begin{pmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & w_n & w_n^2 & \ldots & w_n^{n-1} \\ 1 & w_n^2 & w_n^4 & \ldots & w_n^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & w_n^{n-1} & w_n^{(n-1)2} & \ldots & w_n^{(n-1)(n-1)} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} P(w_n^0) \\ P(w_n^1) \\ \vdots \\ P(w_n^{n-1}) \end{pmatrix}$$

$V_n$

# Inverse fast Fourier Transform

Discrete Fourier transform $\{P(w_n^0), P(w_n^1), \ldots, P(w_n^n)\} \rightarrow P(x) = \sum_{i=0}^{i=n-1} a_i x^i$

$$\begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{pmatrix} = V_n^{-1} \times \begin{pmatrix} P(w_n^0) \\ P(w_n^1) \\ \vdots \\ P(w_n^{n-1}) \end{pmatrix}$$

# Inverse fast Fourier Transform

Discrete Fourier transform $\{P(w_n^0), P(w_n^1), \ldots, P(w_n^n)\} \;\rightarrow P(x) = \sum_{i=0}^{i=n-1} a_i x^i$

$$\begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{pmatrix} = V_n^{-1} \times \begin{pmatrix} P(w_n^0) \\ P(w_n^1) \\ \vdots \\ P(w_n^{n-1}) \end{pmatrix}$$

**Theorem.** $V_n^{-1}[j,k] = w_n^{-kj}/n$ (Proof on the next slide!)

**Corollary.** $a_j = \dfrac{1}{n}\sum_{k=0}^{n-1} P(w_n^k) w_n^{-kj}$; in other words, $a_j = Q(w_n^{-j}) = Q(w_n^{n-j})$, where $Q(z) = y_k z^k$ and $y_k = P(w_n^k)$. Hence, we can compute $a_j$ by the fast Fourier transform.

# Inverse fast Fourier Transform

**Theorem.** $V_n^{-1}[j,k] = w_n^{-kj}/n$

**Proof.**

$$(V_n^{-1}V_n)[j,j'] = \sum_{k=0}^{n-1}(V_n^{-1})[j,k](V_n)[k,j'] = \sum_{k=0}^{n-1}(w_n^{-kj}/n)(w_n^{kj'}) = \sum_{k=0}^{n-1}(w_n^{k(j'-j)}/n)$$

If $j' = j$, the sum equals $1$. Otherwise, the sum equals $0$ by the summation property.

**This lecture: Divide and conquer**

- Divide-and-conquer
- Merge sort
- Analysis of recursive algorithms
- Fast matrix multiplication
- Fast multiplication of polynomials

**Next lecture: Hashing**

- Chained hash tables
- Designing hash functions
- Open addressing
- Cuckoo hashing
- Rolling hash functions