Divide & Conquer (I)

Jordi Cortadella and Jordi Petit
Department of Computer Science

Divide-and-conquer algorithms

- Strategy:
  - Divide the problem into smaller subproblems of the same type of problem
  - Solve the subproblems recursively
  - Combine the answers to solve the original problem

- The work is done in three places:
  - In partitioning the problem into subproblems
  - In solving the basic cases at the tail of the recursion
  - In merging the answers of the subproblems to obtain the solution of the original problem

Conventional product of polynomials

Example:

$$P(x) = 2x^3 + x^2 - 4$$

$$Q(x) = x^2 - 2x + 3$$

$$(P \cdot Q)(x) = 2x^5 + (-4 + 1)x^4 + (6 - 2)x^3 + 8x - 12$$

$$(P \cdot Q)(x) = 2x^5 - 3x^4 + 4x^3 + 8x - 12$$

Conventional product of polynomials

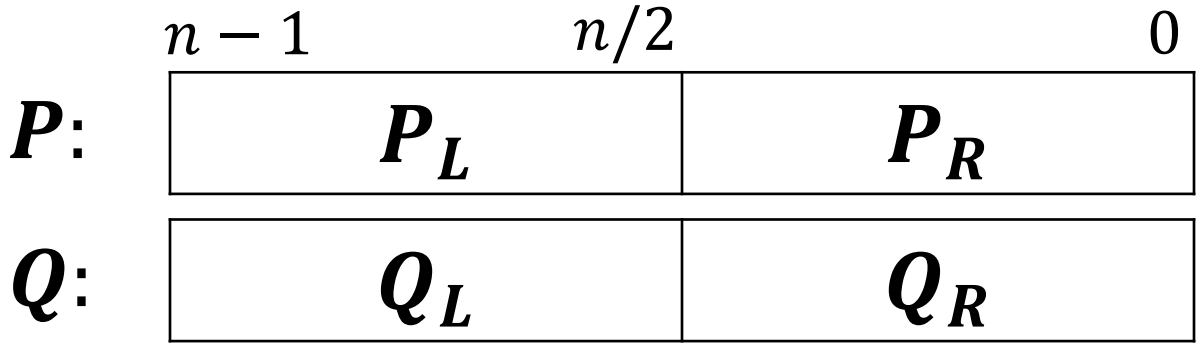
```
function PolynomialProduct(P, Q)
    // P and Q are vectors of coefficients.
    // Returns R = P × Q.
    // degree(P) = size(P)-1, degree(Q) = size(Q)-1.
    // degree(R) = degree(P)+degree(Q).

    R = vector with size(P)+size(Q)-1 zeros;

    for each P_i
      for each Q_j
          R_{i+j} = R_{i+j} + P_i · Q_j

    return R
```

Complexity analysis:

- Multiplication of polynomials of degree n: $O(n^2)$
- Addition of polynomials of degree n: $O(n)$

Product of polynomials: Divide&Conquer

Assume that we have two polynomials with n coefficients (degree $n-1$)



$$P(x)\cdot Q(x) = P_L(x)\cdot Q_L(x)\cdot x^n + (P_R(x)\cdot Q_L(x) + P_L(x)\cdot Q_R(x))\cdot x^{n/2} + P_R(x)\cdot Q_R(x)$$

$$T(n) = 4\cdot T(n/2) + \mathrm{O}(n) = \mathrm{O}(n^2)$$ ← Shown later

Product of complex numbers

- The product of two complex numbers requires four multiplications:

$$(a + bi)(c + di) = ac - bd + (bc + ad)i$$

- Carl Friedrich Gauss (1777-1855) noticed that it can be done with just three: ac, bd and $(a + b)(c + d)$

$$bc + ad = (a + b)(c + d) - ac - bd$$

- A similar observation applies for polynomial multiplication.

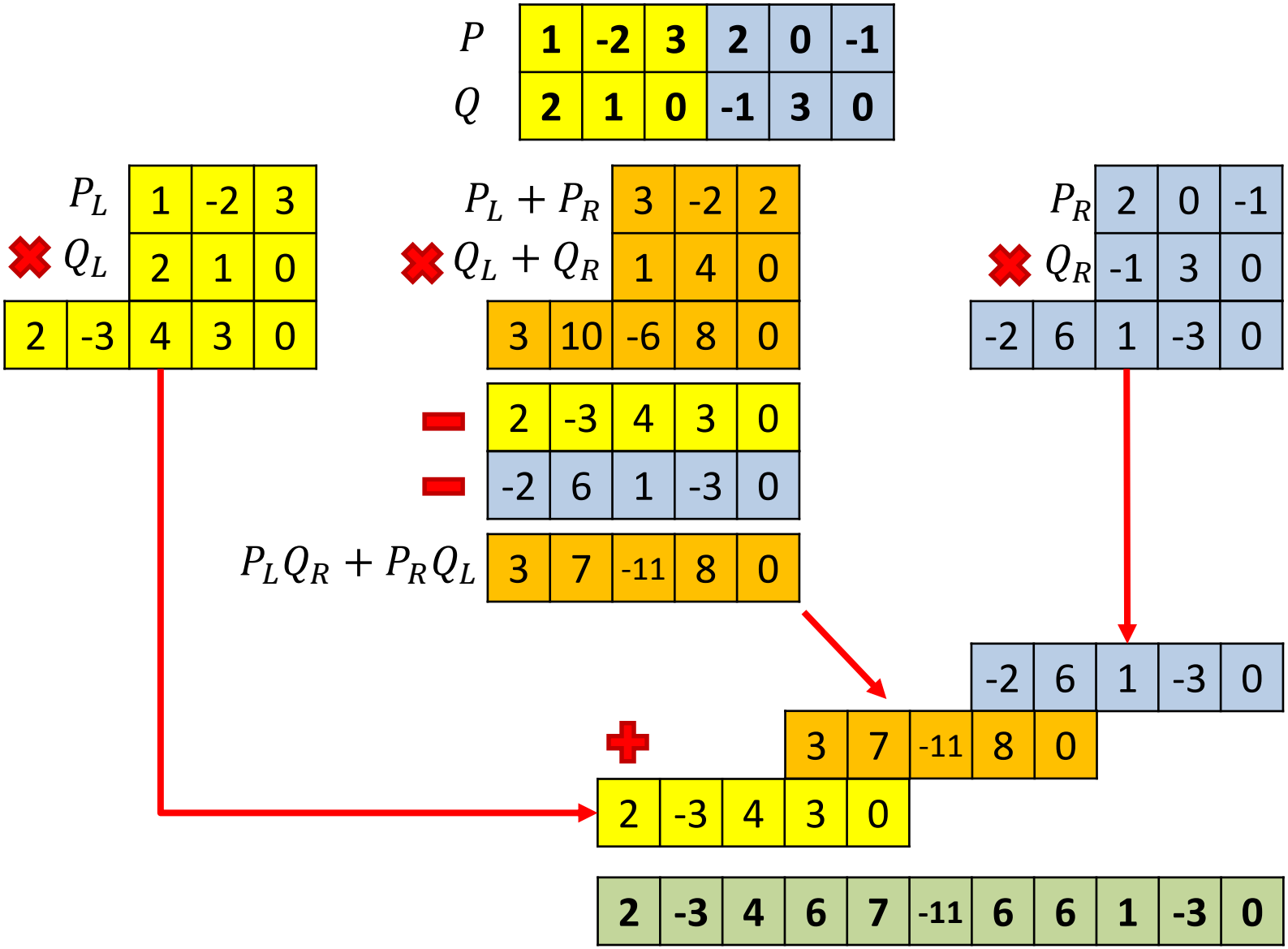
Product of polynomials with Gauss's trick

$$\begin{aligned} R_1 &= P_L Q_L \\ R_2 &= P_R Q_R \\ R_3 &= (P_L + P_R)(Q_L + Q_R) \end{aligned}$$

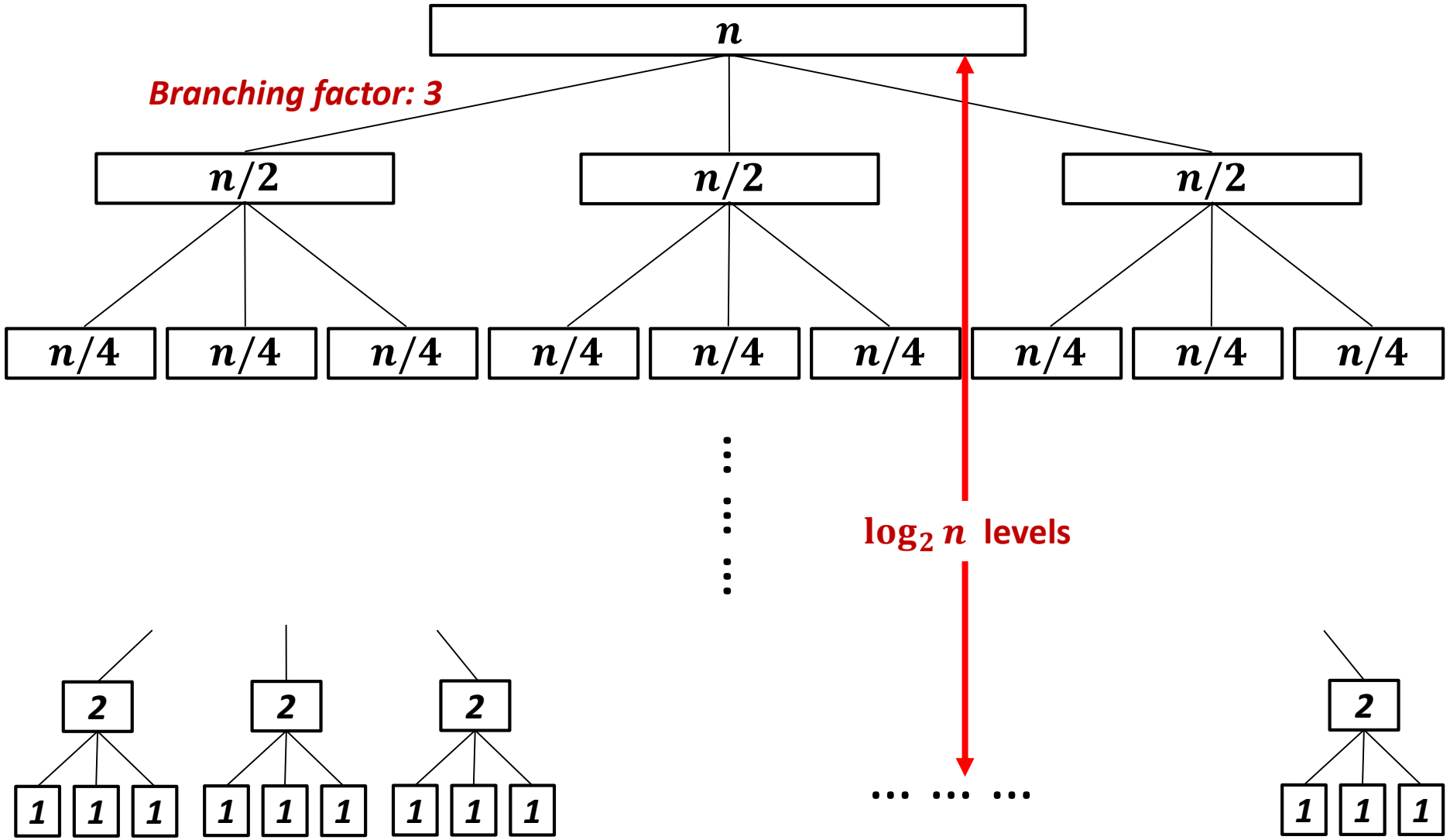
$$PQ = \underbrace{P_L Q_L}_{R_1} x^n + \underbrace{(P_R Q_L + P_L Q_R)}_{R_3 - R_1 - R_2} x^{n/2} + \underbrace{P_R Q_R}_{R_2}$$

$$T(n) = 3T(n/2) + \mathrm{O}(n)$$

Polynomial multiplication: recursive step



Pattern of recursive calls



Useful reminders

- Sum of geometric series with ratio r:

$$S = a + ar + ar^2 + ar^3 + \cdots + ar^{n-1}$$

$$S = a\left(\frac{1 - r^n}{1 - r}\right) = \frac{a}{1 - r} + \frac{r}{r - 1}ar^{n-1}$$

- Logarithms:

$$\log_b n = \log_b a \cdot \log_a n$$

$$a^{\log_b n} = a^{(\log_a n)(\log_b a)} = n^{\log_b a}$$

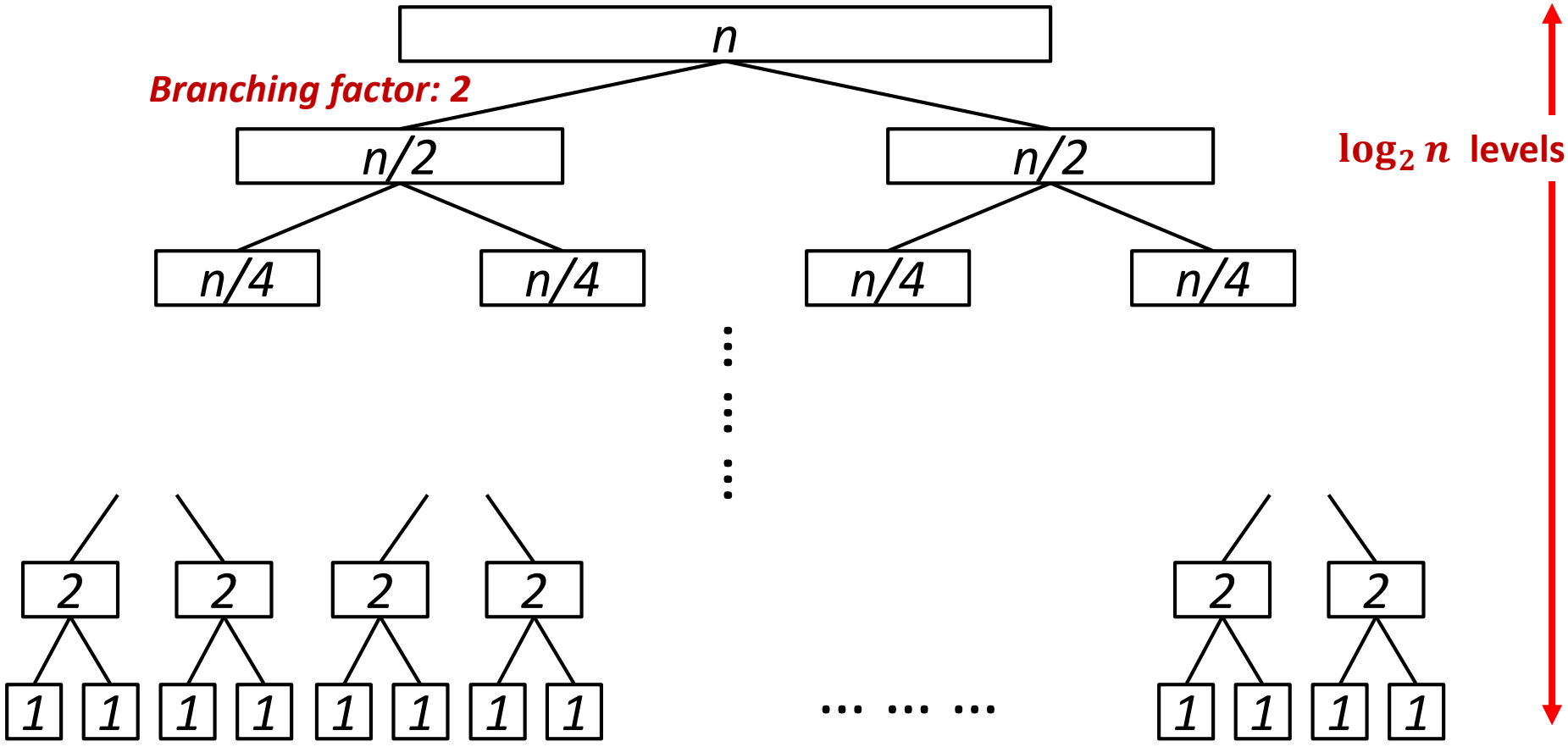
Complexity analysis

- The time spent at level k is

$$3^k \cdot \mathrm{O}\left(\frac{n}{2^k}\right) = \left(\frac{3}{2}\right)^k \cdot \mathrm{O}(n)$$

- For $k = 0$, runtime is $\mathrm{O}(n)$.
- For $k = \log_2 n$, runtime is $\mathrm{O}\left(3^{\log_2 n}\right)$, which is equal to $\mathrm{O}\left(n^{\log_2 3}\right)$.
- The runtime per level increases geometrically by a factor of 3/2 per level. The sum of any increasing geometric series is, within a constant factor, simply the last term of the series.
- Therefore, the complexity is $\mathrm{O}(n^{1.59})$.

A popular recursion tree



Example: efficient sorting algorithms.

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

Algorithms may differ on the amount of work done at each level: $O(n^c)$

Examples

Algorithm	Branch	c	Runtime equation
Power (x^y)	1	0	$T(y) = T(y/2) + \mathrm{O}(1)$
Binary search	1	0	$T(n) = T(n/2) + \mathrm{O}(1)$
Merge sort	2	1	$T(n) = 2 \cdot T(n/2) + \mathrm{O}(n)$
Polynomial product	4	1	$T(n) = 4 \cdot T(n/2) + \mathrm{O}(n)$
Polynomial product (Gauss)	3	1	$T(n) = 3 \cdot T(n/2) + \mathrm{O}(n)$

Master theorem

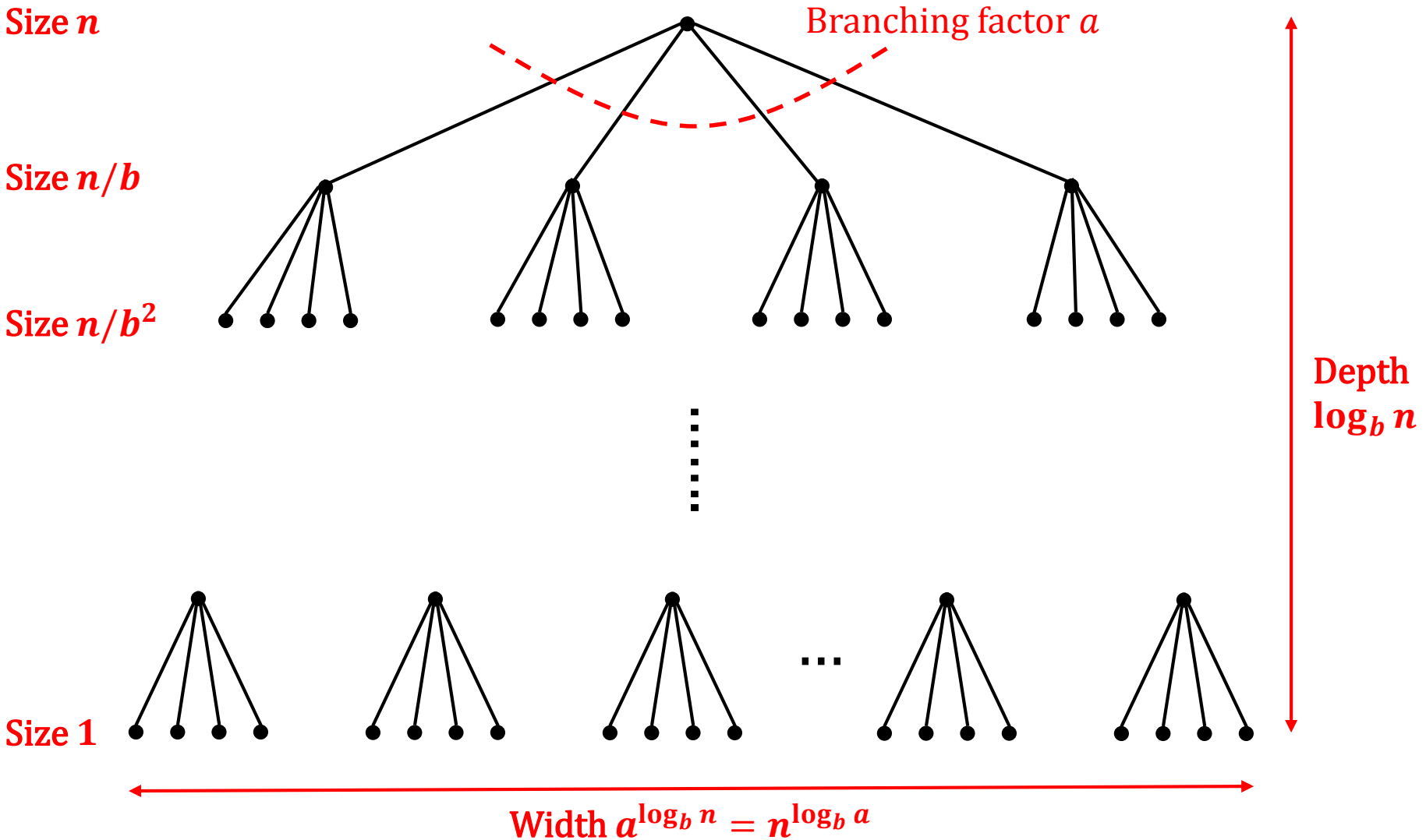
- Typical pattern for Divide&Conquer algorithms:
  - Split the problem into a subproblems of size n/b
  - Solve each subproblem recursively
  - Combine the answers in $\mathrm{O}(n^c)$ time

- Running time: $T(n) = a \cdot T(n/b) + \mathrm{O}(n^c)$

- Master theorem:

$$T(n) = \begin{cases} \mathrm{O}(n^c) & \text{if } c > \log_b a \quad (a < b^c) \\ \mathrm{O}(n^c \log n) & \text{if } c = \log_b a \quad (a = b^c) \\ \mathrm{O}\left(n^{\log_b a}\right) & \text{if } c < \log_b a \quad (a > b^c) \end{cases}$$

Master theorem: recursion tree



Master theorem: proof

- For simplicity, assume n is a power of b.
- The base case is reached after $\log_b n$ levels.
- The kth level of the tree has a^k subproblems of size n/b^k.
- The total work done at level k is:

$$a^k \times O\left(\frac{n}{b^k}\right)^c = O(n^c) \times \left(\frac{a}{b^c}\right)^k$$

- As k goes from 0 (the root) to $\log_b n$ (the leaves), these numbers form a geometric series with ratio a/b^c. We need to find the sum of such a series.

$$T(n) = O(n^c) \cdot \underbrace{\left(1 + \frac{a}{b^c} + \frac{a^2}{b^{2c}} + \frac{a^3}{b^{3c}} + \cdots + \frac{a^{\log_b n}}{b^{(\log_b n)c}}\right)}_{\log_b n \text{ terms}}$$

Master theorem: proof

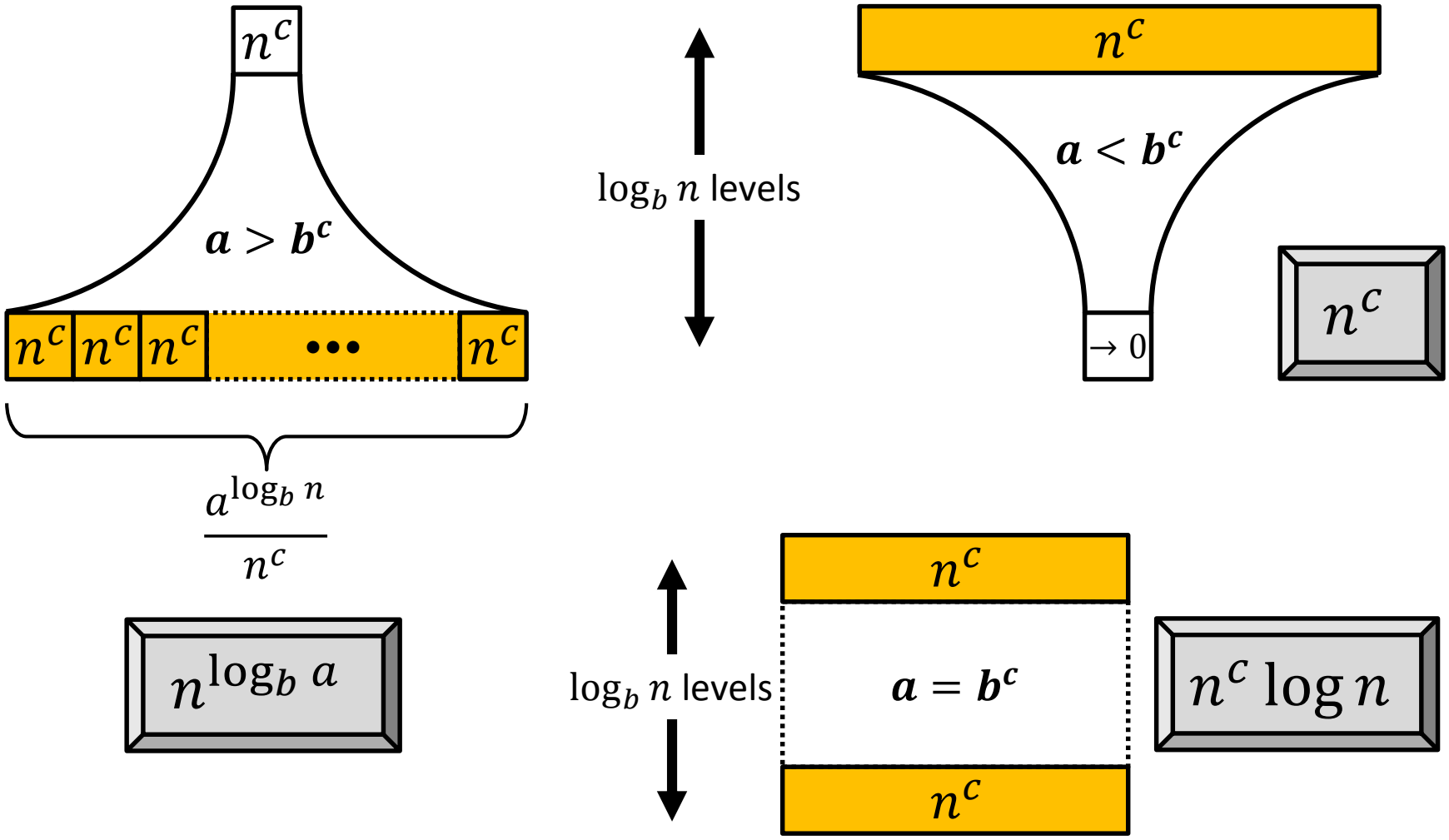
- Case $a/b^c < 1$. Decreasing series. The sum is dominated by the first term ($k = 0$): $\mathrm{O}(n^c)$.

- Case $a/b^c > 1$. Increasing series. The sum is dominated by the last term ($k = \log_b n$):

$$n^c \left(\frac{a}{b^c}\right)^{\log_b n} = n^c \left(\frac{a^{\log_b n}}{(b^{\log_b n})^c}\right) = a^{\log_b n} = n^{\log_b a}$$

- Case $a/b^c = 1$. We have $\mathrm{O}(\log n)$ terms all equal to $\mathrm{O}(n^c)$.

Master theorem: visual proof



Master theorem: examples

Running time: $T(n) = a \cdot T(n/b) + \mathrm{O}(n^c)$

$$T(n) = \begin{cases} \mathrm{O}(n^c) & \text{if } a < b^c \\ \mathrm{O}(n^c \log n) & \text{if } a = b^c \\ \mathrm{O}\left(n^{\log_b a}\right) & \text{if } a > b^c \end{cases}$$

Algorithm	a	c	Runtime equation	Complexity
Power (x^y)	1	0	$T(y) = T(y/2) + \mathrm{O}(1)$	$\mathrm{O}(\log y)$
Binary search	1	0	$T(n) = T(n/2) + \mathrm{O}(1)$	$\mathrm{O}(\log n)$
Merge sort	2	1	$T(n) = 2 \cdot T(n/2) + \mathrm{O}(n)$	$\mathrm{O}(n \log n)$
Polynomial product	4	1	$T(n) = 4 \cdot T(n/2) + \mathrm{O}(n)$	$\mathrm{O}(n^2)$
Polynomial product (Gauss)	3	1	$T(n) = 3 \cdot T(n/2) + \mathrm{O}(n)$	$\mathrm{O}(n^{\log_2 3})$

$b = 2$ for all the examples