Divide and Conquer-3

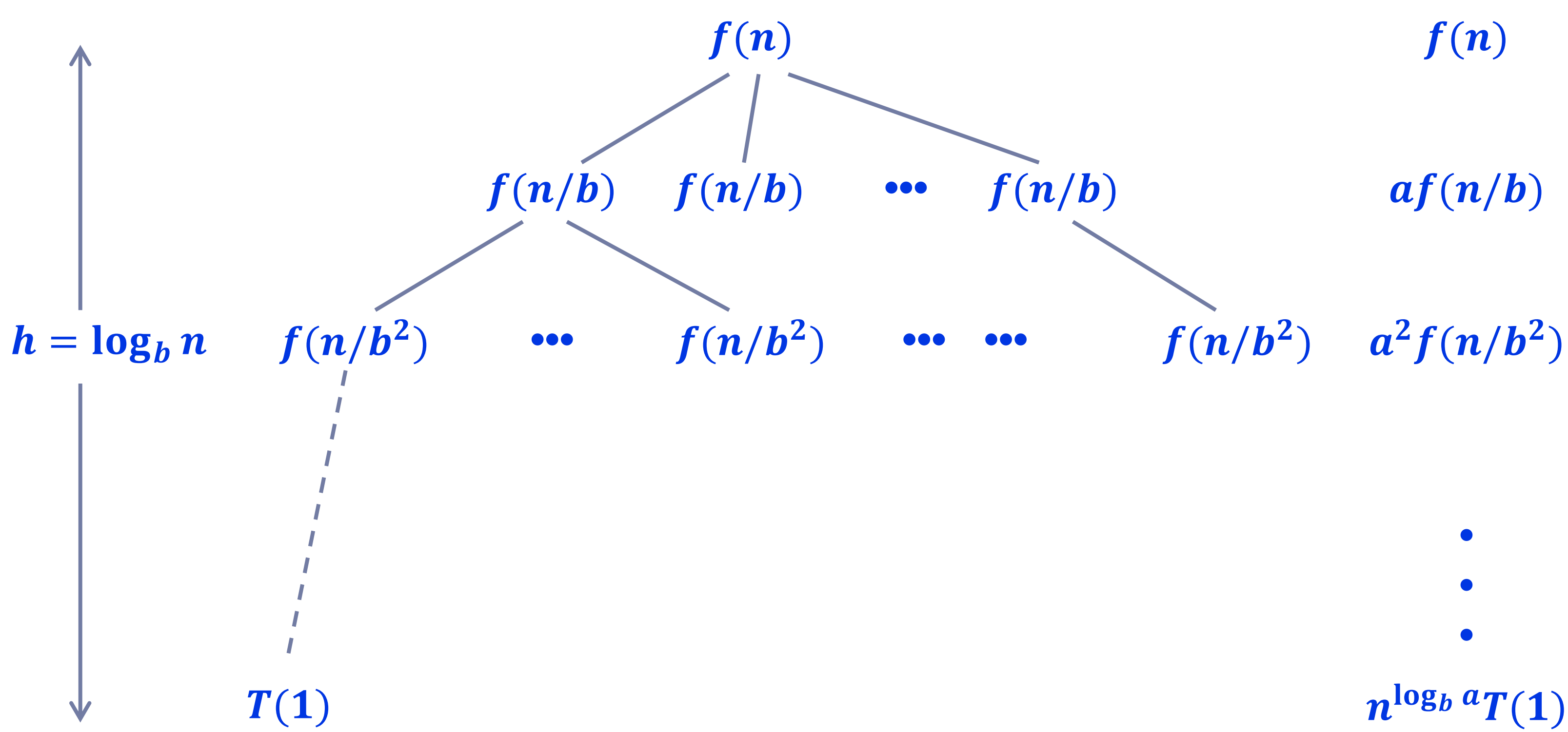
Department of Computer Science, Tsinghua University

The master method

The master method applies to recurrences of the form

$$T(n) = aT(n/b) + f(n)$$

where $a \geq 1, b \geq 1$ and $f(n)$ is asymptotically positive.



Case 1:

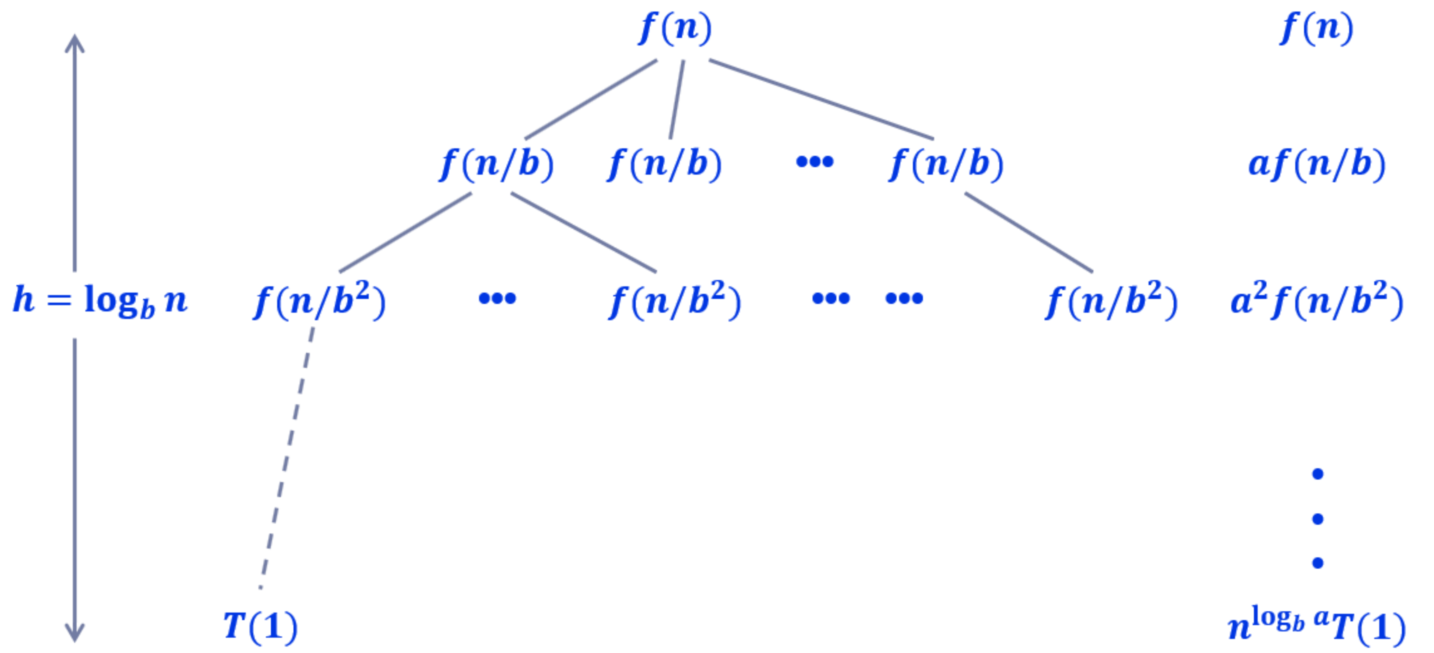
- Compare $f(n)$ with $n^{\log_b a}$
- Case 1. $f(n) = O(n^{\log_b a-\varepsilon})$ for some constant $\varepsilon > 0$
  - $f(n)$ grows polynomially slower than $n^{\log_b a}$ (by a n^{ε} factor)
  - Solution: $T(n) = \Theta(n^{\log_b a})$

- Ex. $T(n) = 4T(n/2) + n$

$$a = 4, b = 2 \Rightarrow n^{\log_b a} = n^2; f(n) = n$$

Case 1: $f(n) = O\left(n^{2-\varepsilon}\right)$ for $\varepsilon = 0.5$ $\quad \therefore T(n) = \Theta(n^2)$

Case 1:



Case 1

The weight increases geometrically from the root to the leaves. The leaves hold a constant fraction of the total weight $\Theta(n^{\log_b a})$

Case 2:

- Compare $f(n)$ with $n^{\log_b a}$
- Case 2. $f(n) = \Theta(n^{\log_b a})$
  - $f(n)$ and $n^{\log_b a}$ grow at similar rates.
  - Solution: $T(n) = \Theta(n^{\log_b a} \lg n)$

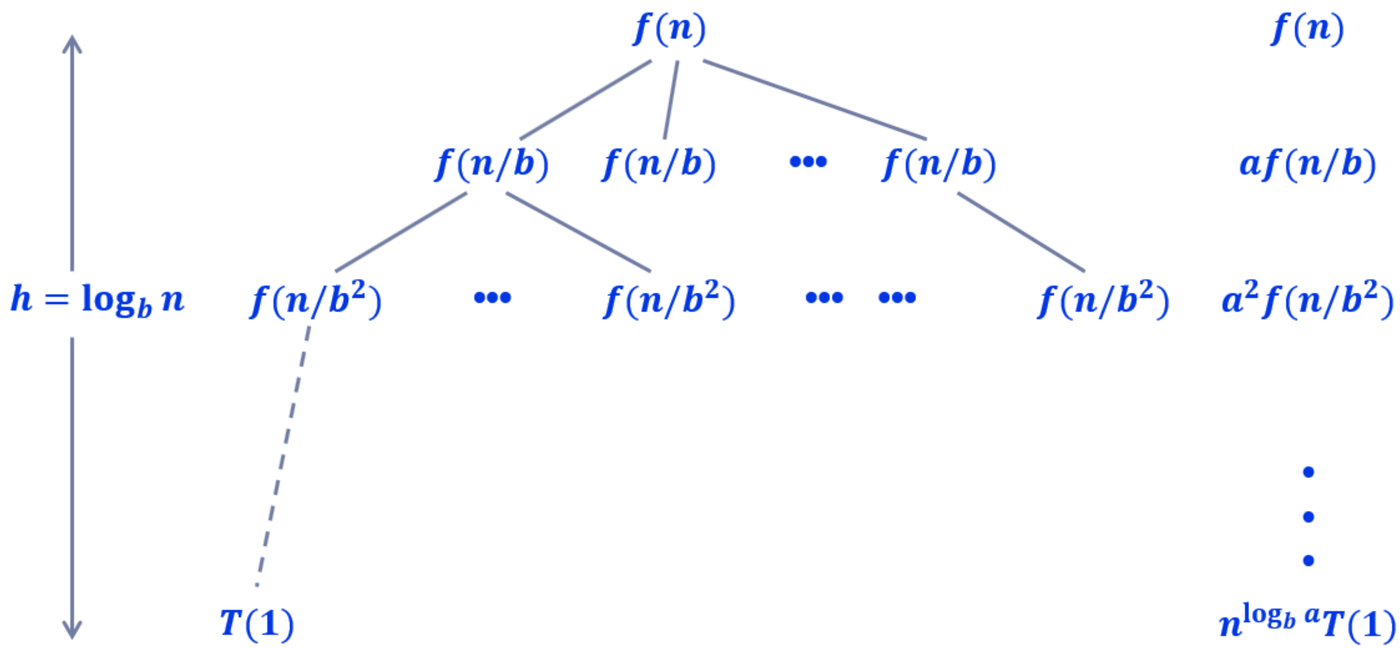
- Ex. $T(n) = 4T(n/2) + n^2$

$$a = 4, b = 2 \Rightarrow n^{\log_b a} = n^2; f(n) = n^2$$

Case 2: $f(n) = \Theta\left(n^2\right)$

$\therefore T(n) = \Theta(n^2 \lg n)$

Case 2:



Case 2

The weight is approximately the same on each of the $\log_b n + 1$ levels.

$T(n) = \Theta(n^{\log_b a} \lg n)$

Case 3:

- Compare $f(n)$ with $n^{\log_b a}$
- Case 3. $f(n) = \Omega(n^{\log_b a+\varepsilon})$ for some constant $\varepsilon > 0$
  - $f(n)$ grows polynomially faster than $n^{\log_b a}$ (by an n^{ε} factor)
  - and $f(n)$ satisfies the **regularity condition** that

    $$af(n/b) \leq cf(n) \text{ for some constant } c < 1$$
  - Solution: $T(n) = \Theta(f(n))$

- Ex. $T(n) = 4T(n/2) + n^3$

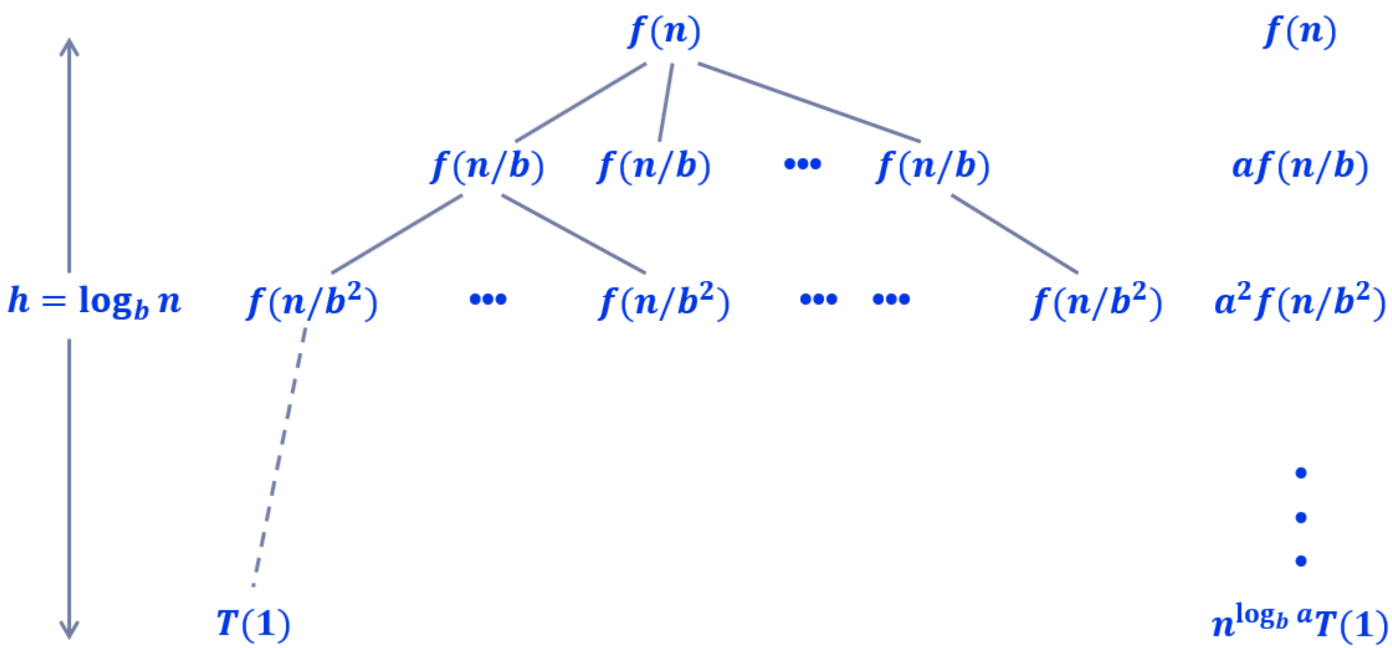
$$a = 4, b = 2 \Rightarrow n^{log_b a} = n^2; f(n) = n^3$$

Case 3: $f(n) = \Omega\left(n^{2+\varepsilon}\right)\ for\, \varepsilon = 0.5$

$4(n/2)^3 \leq cn^3$ (reg. cond.) $holds\ when\ c = 1/2$

$\therefore T(n) = \Theta(n^3)$

Case 3:



Case 3

The weight decreases geometrically from the root to the leaves. The root holds a constant fraction of the total weight $\Theta\big(f(n)\big)$

One more example

- Ex. $T(n) = 2T(n/2) + n\lg n$

  $a = 2, b = 2 \Rightarrow n^{\log_b a} = n; f(n) = n\lg n$

Master method does not apply

In particular, the ratio $f(n)/n = \lg n$ is not asymptotically greater than n^{ε} for any positive constant ε

Summary

- Compare $f(n)$ with $n^{\log_b a}$
- 1. $f(n) = O(n^{\log_b a-\varepsilon})$ for some constant $\varepsilon > 0$
  - $f(n)$ grows polynomially slower than $n^{\log_b a}$ (by a n^{ε} factor)
  - Solution: $T(n) = \Theta(n^{\log_b a})$
- 2. $f(n) = \Theta(n^{\log_b a})$
  - $f(n)$ and $n^{\log_b a}$ grow at similar rates.
  - Solution: $T(n) = \Theta(n^{\log_b a} \lg n)$
- 3. $f(n) = \Omega(n^{\log_b a+\varepsilon})$ for some constant $\varepsilon > 0$
  - $f(n)$ grows polynomially faster than $n^{\log_b a}$ (by an n^{ε} factor)
  - and $f(n)$ satisfies the **regularity condition** that
    $af(n/b) \le cf(n)$ for some constant $c < 1$
  - Solution: $T(n) = \Theta(f(n))$

In-class Exercise

- True or False: if a divide-and-conquer algorithm A runs in $T(n) = aT(n/b) + f(n)$
  - 1) $T(n) = \Omega\left(n^{\log_b a}\right)$.
  - 2) if $a > 1$, the running time of A can be bounded by a logarithmic function (i.e., $T(n) = O(\lg n)$).
  - 3) When $a < b$ and $f(n) = \theta(n)$, then $T(n) = \theta(n)$.

Matrix Multiplication

$$\left.\begin{aligned} A = [a_{ij}], B &= [b_{ij}] \\ C = [c_{ij}] &= A \cdot B \end{aligned}\right\} \; i, j = 1,2,\cdots,n$$

$$\begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nn} \end{bmatrix}$$

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

Standard Algorithm

```
for i = 1 to n
  for j = 1 to n
      c_ij = 0
      for k = 1 to n
      c_ij = c_ij + a_ik × b_kj
```

Running time $= \Theta(n^3)$

IDEA:

$n \times n$ matrix = 2×2 matrix of $(n/2) \times (n/2)$ submatrices:

$$\begin{bmatrix} r & s \\ t & u \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

$$C = A \times B$$

$$\left.\begin{aligned} r &= ae + bg \\ s &= af + bh \\ t &= ce + dg \\ u &= cf + dh \end{aligned}\right\}$$

8 mults of $(n/2) \times (n/2)$ **submatrices**

4 adds of $(n/2) \times (n/2)$ **submatrices**

$$\boldsymbol{T(n) = 8T(n/2) + \theta\left(n^2\right) = \theta(n^3)}$$

Strassen's idea (1969)

Reduce **8 mults to 7 mults**

$$
\begin{aligned}
P_1 &= a(f-h) \\
P_2 &= (a+b)h \\
P_3 &= (c+d)e \\
P_4 &= d(g-e) \\
P_5 &= (a+d)(e+h) \\
P_6 &= (b-d)(g+h) \\
P_7 &= (a-c)(e+f)
\end{aligned}
$$

$$
\begin{aligned}
r &= P_5 + P_4 - P_2 + P_6 \\
s &= P_1 + P_2 \\
t &= P_3 + P_4 \\
u &= P_5 + P_1 - P_3 - P_7
\end{aligned}
$$

Strassen's Algorithm

1. ***Divide:*** Partition *A* and *B* into $(n/2) \times (n/2)$ submatrices and form terms to be multiplied.
2. ***Conquer:*** Perform 7 multiplications of $(n/2) \times (n/2)$ submatrices recursively.
3. ***Combine:*** Form *C* using + and – on $(n/2) \times (n/2)$ submatrices.

$$T(n) = 7T\left(\frac{n}{2}\right) + \Theta(n^2) = \Theta(n^{2.81})$$

Chapter Notes for Strassen's algorithm in practice.

Summary

- Master Method: three cases depending on the outcome of comparing $f(n)$ with $n^{\log_b a}$.
- There are cases where the Master Method does not apply.
- Estimating efficiency using the Master Method can provide hints to design efficient D&C algorithms.