

# Complexity IBC028, Lecture 2

H. Geuvers

Institute for Computing and Information Sciences  
Radboud University Nijmegen

Version: spring 2026



# Outline

Recursion tree method

The Master Theorem



# Techniques to prove $T(n) = \mathcal{O}(g(n))$ [or $T(n) = \Omega(g(n))$ or $T(n) = \Theta(g(n))$ ]

There are basically three techniques

- 1 **Substitution Method:** For given  $g$ ,  
Choose  $c > 0$  (and  $N$ ) and prove (by induction on  $n$ )

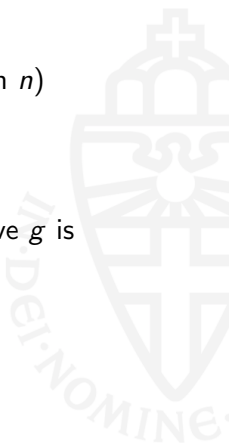
$$T(n) \leq c g(n) \quad (\text{for all } n > N)$$

- 2 **Recursion Tree method :**  
Method to find  $g$ . And then you still have to prove  $g$  is correct using (1)

- 3 **Master theorem method :**  
General theorem for patterns of the shape

$$T(n) = aT\left(\frac{n}{b}\right) + f(n).$$

Actually: casting the heuristic method of (2) into a general theorem.



# Substitution method

Last week (MergeSort):

## THEOREM

If  $T(n) \leq 2T(\lfloor \frac{n}{2} \rfloor) + \Theta(n)$ , then

$$T(n) \in \mathcal{O}(n \log n).$$

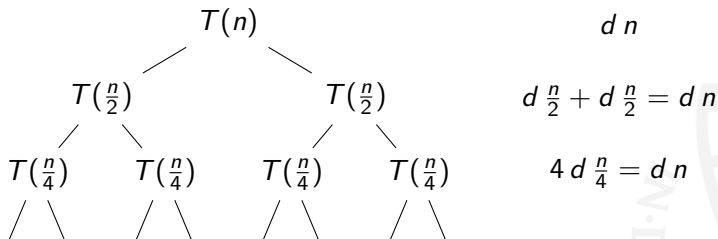
In fact, the  $n \log n$  was an educated guess, which we then proved by induction.

But how do we make an “educated guess” ...how do we find the  $n \log n$ ?

Answer: Make a **recursion tree**!

# Recursion Tree method (I)

EXAMPLE.  $T(n) = 2T(\frac{n}{2}) + d n.$



- The height is  $\log n$ , so there are  $\log n + 1$  layers
- Per layer:  $d n$  cost contribution
- Bottom: #leaves =  $2^{\log n} = n$ ; cost per leaf  $\Theta(1)$ .
- Total cost:  $d n \log n + n \Theta(1)$
- So we conjecture:  $T(n) = \Theta(n \log n)$

## Some computation rules with log

For exponent:  $(b^x)^y = b^{x \cdot y}$  and  $b^x b^y = b^{x+y}$ .

By definition:

$$\log_b x = y \iff b^y = x \quad \text{and so } b^{\log_b x} = x$$

Rules for log

$\log_b(x \cdot y) = \log_b x + \log_b y$	$\log_b(x^k) = k \log_b x$
$\log_b\left(\frac{x}{y}\right) = \log_b x - \log_b y$	$\log_b\left(\frac{1}{x}\right) = -\log_b x$

Changing base:

$$\log_a x = \log_a b \cdot \log_b x \quad \text{and so} \quad \log_a f(n) = \log_a b \cdot \log_b f(n)$$

$$x^{\log_c y} = y^{\log_c x} \quad \text{and so} \quad x^{\log_c f(n)} = f(n)^{\log_c x}$$

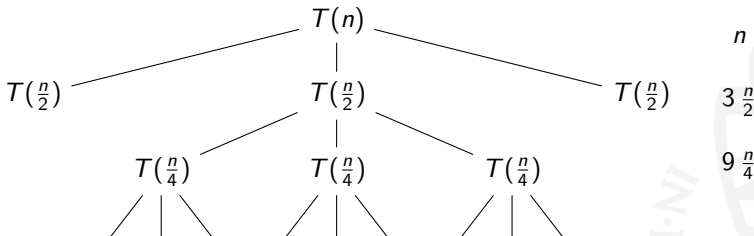
Addition/substraction under log:

$$\log(x-1) \geq \log x - 1 \quad \log x + 1 \geq \log(x+1) \quad \text{for } x \geq 2$$



# Recursion Tree method (II)

Question. Given  $T(n) = 3T(\lfloor \frac{n}{2} \rfloor) + n$ , find  $f$  with  $T(n) = \Theta(f(n))$ .



- Height is  $\log n$ , so  $3^{\log n} = n^{\log 3}$  leaves, contributing  $\Theta(n^{\log 3})$ .
- At layer  $i$  we have  $3^i \frac{n}{2^i}$  contribution.
- Total:  $\sum_{i=0}^{\log n} \left(\frac{3}{2}\right)^i n = n \frac{\left(\frac{3}{2}\right)^{\log n + 1} - 1}{\frac{3}{2} - 1} = 2 \frac{3}{2} n \left(\frac{3}{2}\right)^{\log n} - 2n = c \cdot 3^{\log n} - 2n$  (for some  $c > 0$ ).
- As  $3^{\log n} = n^{\log 3}$ , we conjecture:  $T(n) = \Theta(n^{\log 3})$ .

## Substitution method

Given  $T(n) = 3T(\lfloor \frac{n}{2} \rfloor) + n$ . We prove:  $T(n) = \mathcal{O}(n^{\log 3})$ .

Proof. We need to prove  $T(n) \leq cn^{\log 3}$  for appropriately chosen  $c$  (for all  $n > N$  for some appropriately chosen  $N$ )

$$\begin{aligned} T(n) &= 3T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + n \\ &\stackrel{IH}{\leq} 3c\left(\frac{n}{2}\right)^{\log 3} + n \\ &= \frac{3c n^{\log 3}}{2^{\log 3}} + n = cn^{\log 3} + n \stackrel{??}{\leq} cn^{\log 3} \end{aligned}$$

The induction fails, so we add a linear factor:  $T(n) \leq cn^{\log 3} + dn$ . We notice that it works for  $d = -2$ , because we have

$$T(n) = 3T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + n \stackrel{IH}{\leq} 3\left(c\left(\frac{n}{2}\right)^{\log 3} - 2\frac{n}{2}\right) + n = cn^{\log 3} - 3n + n = cn^{\log 3} - 2n$$

We choose  $N = 1$  and we choose  $c$  large enough such that  $T(n) \leq cn^{\log 3} - 2n$  for small  $n$ .

# Computing the median of an unsorted list

Problem: Given an unsorted list of elements, compute the **median**.  
(Median of  $A$  = element that has half of the elements of  $A$  below it and the other half above it.)

Possible solution:

- First sort the list  $A$ , with  $|A| = n$ .
- Then take the  $\lfloor \frac{n}{2} \rfloor$ -th element

This takes  $\mathcal{O}(n \log n)$  time.

But it can be done in linear time!

The algorithm is more general: for  $A$  a list and  $k$  a number,

$M(A, k) :=$  the  $k$ -th element of the sorted version of  $A$ .

Then the median of  $A$  is  $M(A, \lfloor \frac{|A|}{2} \rfloor)$ .



# Computing the median of a list in linear time (I)

$M(A, k) :=$  the  $k$ -th element of the sorted version of  $A$ .

Let  $n = |A|$ . For purpose of exposition, we assume  $n = 5p$  for some  $p$ . (If  $n < 5p$  add 0s to get  $n = 5p$ .)

- ① Split  $A$  randomly in  $\frac{n}{5}$  groups of 5 elements
- ② Determine the median of each group of 5 elements.
- ③ Determine recursively the median of these  $\frac{n}{5}$  medians, say  $m$
- ④ Count the number of elements in  $A$  that are  $\leq m$ , say  $\ell$ .
  - If  $\ell = k$ , we are done and  $m$  is the output.
  - If  $\ell > k$ , then  $m$  is larger than the number we are looking for, so we continue recursively with  $M(A \setminus A_{\text{high}}, k)$
  - If  $\ell < k$ , then  $m$  is smaller than the number we are looking for, so we continue recursively with  $M(A \setminus A_{\text{low}}, k - |A_{\text{low}}|)$ .
  - Until  $n$  is “very small”, say  $n \leq 10$ , then compute the  $k$ -th element directly

**Q.** What exactly are  $A_{\text{high}}$  and  $A_{\text{low}}$  and how large are they?

# Computing the median of a list in linear time (II)

- ① Split  $A$  randomly in  $\frac{n}{5}$  groups of 5 elements
- ② Determine the median of each group of 5 elements.
- ③ Determine recursively the median of these  $\frac{n}{5}$  medians, say  $m$
- ④ Count the number of elements in  $A$  that are  $\leq m$ , say  $\ell$ .
  - If  $\ell = k$ , we are done and  $m$  is the output.
  - If  $\ell > k$ , then  $m$  is larger than the number we are looking for, so we continue recursively with  $M(A \setminus A_{\text{high}}, k)$
  - If  $\ell < k$ , then  $m$  is smaller than the number we are looking for, so we continue recursively with  $M(A \setminus A_{\text{low}}, k - 3 \lceil \frac{n}{10} \rceil)$ .
  - Until  $n$  is “very small”, say  $n \leq 10$ , then compute the  $k$ -th element directly

Complexity:

$$T(n) = T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right) + \Theta(n).$$

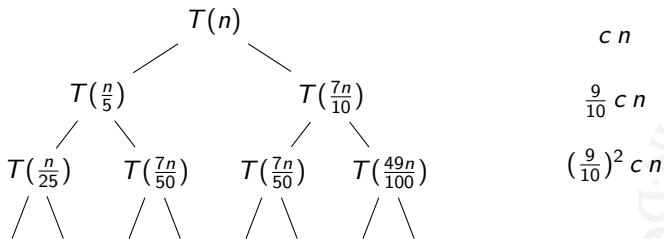
Note that steps (1), (2) and the first part of (4) are linear in  $n$ .



# Computing the median of a list in linear time (III)

$$T(n) \leq T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right) + cn \text{ for some } c.$$

To find the complexity class of  $T$  we can make a recursion tree.



- The height is between  $\log_5 n$  and  $\log_{\frac{10}{7}} n$ , which is on average below  $\log_2 n$ , so an upperbound for the number of leaves is  $2^{\log_2 n} = n^{\log_2 2} = n$ .
- The layers:  $\sum_{i=0}^{??} \left(\frac{9}{10}\right)^i cn \leq \sum_{i=0}^{\infty} \left(\frac{9}{10}\right)^i cn = cn \sum_{i=0}^{\infty} \left(\frac{9}{10}\right)^i = 10cn$
- Conjecture  $T(n) \leq 10cn$ .

## Computing the median of a list in linear time (IV)

$$T(n) \leq T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right) + cn.$$

From the recursion tree method we conjecture that  $T(n) \leq 10cn$ .

Proof by induction on  $n$ 

- For small  $n$ , it is correct. (Possibly need to choose a larger  $c$ .)
- For larger  $n$ :

$$\begin{aligned} T(n) &\leq T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right) + cn \\ &\stackrel{\text{IH}}{\leq} 10c\left(\frac{n}{5}\right) + 10c\left(\frac{7n}{10}\right) + cn \\ &= 2cn + 7cn + cn \\ &= 10cn \end{aligned}$$

So  $T(n) = \mathcal{O}(n)$ , and so  $M$  is linear in the length of the input list.

# Master Theorem

## THEOREM

Suppose  $a \geq 1$  and  $b > 1$  and we abbreviate  $\gamma := \log_b a$ .

$$T(n) = aT\left(\frac{n}{b}\right) + f(n).$$

Then

- 1  $T(n) = \Theta(n^\gamma)$  if  $f(n) = \mathcal{O}(n^d)$  for some  $d < \gamma$ .  
 $f$  is “relatively small” compared to  $n^\gamma$
- 2  $T(n) = \Theta(n^\gamma \log n)$  if  $f(n) = \Theta(n^\gamma)$ .  
 E.g. the Mergesort case
- 3  $T(n) = \Theta(f(n))$  if  $f(n) = \Omega(n^d)$  for some  $d > \gamma$  and  
 $\exists c \in (0, 1) \exists N \forall n > N (af(\frac{n}{b}) \leq cf(n))$ .  
 $f$  is “relatively large” compared to  $n^\gamma$

# Using the Master Theorem (I)

$$T(n) = 9T\left(\frac{n}{3}\right) + n$$

## THEOREM (with $\gamma = \log_b a$ )

- 1  $T(n) = \Theta(n^\gamma)$  if  $f(n) = \mathcal{O}(n^d)$  for some  $d < \gamma$ .
- 2  $T(n) = \Theta(n^\gamma \log n)$  if  $f(n) = \Theta(n^\gamma)$ .
- 3  $T(n) = \Theta(f(n))$  if  $f(n) = \Omega(n^d)$  for some  $d > \gamma$  and  
 $\exists c \in (0, 1) \exists N \forall n > N (af(\frac{n}{b}) \leq cf(n))$ .

Now,  $a = 9$  and  $b = 3$ , so  $\gamma = \log_b a = \log_3 9 = 2$ .

Also  $f(n) = n = \mathcal{O}(n) = \mathcal{O}(n^1)$  and  $1 < 2 = \gamma$ .

So case (1) of the Master Theorem applies and we have

$$T(n) = \Theta(n^2).$$

# Using the Master Theorem (II)

## THEOREM (with $\gamma = \log_b a$ )

- 1  $T(n) = \Theta(n^\gamma)$  if  $f(n) = \mathcal{O}(n^d)$  for some  $d < \gamma$ .
- 2  $T(n) = \Theta(n^\gamma \log n)$  if  $f(n) = \Theta(n^\gamma)$ .
- 3  $T(n) = \Theta(f(n))$  if  $f(n) = \Omega(n^d)$  for some  $d > \gamma$  and  
 $\exists c \in (0, 1) \exists N \forall n > N (af(\frac{n}{b}) \leq cf(n))$ .

$$T(n) = 9T\left(\frac{n}{4}\right) + n^2.$$

Now,  $a = 9$  and  $b = 4$ , so  $\gamma = \log_b a = \log_4 9 \approx 1.584$ .

Also  $f(n) = n^2 = \Omega(n^2)$  and  $2 > \gamma$ .

So case (3) of the Master Theorem applies and we have

$$T(n) = \Theta(n^2).$$

We need an extra check:

$$\exists c \in (0, 1) \exists N \forall n \geq N (af(\frac{n}{b}) \leq cf(n))??$$

That is:  $9\left(\frac{n}{4}\right)^2 \leq cn^2$ , so take  $c := \frac{9}{16}$  and this is ok.