

Complexity IBC028, Lecture 3

Herman Geuvers

Institute for Computing and Information Sciences
Radboud University Nijmegen

Version: spring 2026



Outline

Master Theorem

Examples of algorithms



Last week: Three techniques to prove complexity

1 Substitution Method:

Choose (guess) f and c (and N_0) and prove $T(n) \leq c f(n)$ (for $n > N_0$) by induction on n .

2 Recursion Tree method :

Method to estimate f . And then you still have to prove f is correct using (1)

3 Master theorem method :

General theorem for patterns of the shape

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

This week:

- Using the Master Theorem
- Examples of algorithms and applications of the Master Theorem

The 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^γ
- 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
 $\exists c < 1 \exists N \forall n > N (af(\frac{n}{b}) \leq cf(n))$.
 f is “relatively large” compared to n^γ



Typical Exercise with the Master Theorem

Exercise

Suppose

$$T(n) = 8T\left(\frac{n}{2}\right) + n$$

Determine the complexity class of T

We use the master theorem

- $a = 8$
- $b = 2$
- So: $\gamma = \log_2 8 = 3$
- $f(n) = n$, so $f(n) = \mathcal{O}(n^1)$ and $1 < 3$.
- So we are in case 1 and we conclude that $T = \Theta(n^3)$.



Example Karatsuba multiplication (I)

Multiplying two numbers X and Y of n digits in the “standard” way takes $\Theta(n^2)$ steps.

We can do it recursively (assume n is a power of 2):

$$\begin{array}{l}
 X = \overbrace{\boxed{a}}^{n/2} \overbrace{\boxed{b}}^{n/2} = a2^{n/2} + b \\
 Y = \underbrace{\boxed{c}}_{n/2} \underbrace{\boxed{d}}_{n/2} = c2^{n/2} + d
 \end{array}$$

$$XY = ac2^n + (ad + bc)2^{n/2} + bd$$

We do (recursively) 4 multiplications of numbers of size $\frac{n}{2}$, so

$$T(n) = 4T\left(\frac{n}{2}\right) + \mathcal{O}(n)$$

So (Master theorem, $a = 4, b = 2$, case (1)): $T(n) = \Theta(n^2)$.

But we can do better.

Example Karatsuba multiplication (II)

Karatsuba multiplication of two numbers X and Y of n digits.
(Assume n is a power of 2.)

$$X = \boxed{a} \boxed{b} = a2^{n/2} + b$$

$$Y = \boxed{c} \boxed{d} = c2^{n/2} + d$$

Define

$$X_1 = a + b \quad X_2 = c + d \quad X_3 = X_1 X_2$$

$$X_4 = ac \quad X_5 = bd \quad X_6 = X_3 - X_4 - X_5$$

$X_3 = ac + ad + bc + bd$ and $X_6 = ad + bc$, so XY can be obtained using 3 multiplications of numbers of size $n/2$:

$$T(n) = 3T\left(\frac{n}{2}\right) + \mathcal{O}(n)$$

MT, $a = 3, b = 2$, case (1): $T(n) = \Theta(n^{\log_2 3}) \approx \Theta(n^{1.584})$.

NB. In case n is not a power of 2, we add leading zeros.

(Exercise: check that this works and doesn't change the complexity class.)

Matrix multiplication

Multiplying two matrices $A \cdot B$ of size $n \times n$ (with n a power of 2).

- The “standard” algorithm takes $\Theta(n^3)$ steps.
- Any algorithm will be $\Omega(n^2)$.

Recursively: Split A and B into 4 submatrices of order $n/2$:

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \cdot \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

where

$$\begin{array}{ll} C_{11} = A_{11}B_{11} + A_{12}B_{21} & C_{12} = A_{11}B_{12} + A_{12}B_{22} \\ C_{21} = A_{21}B_{11} + A_{22}B_{21} & C_{22} = A_{21}B_{12} + A_{22}B_{22} \end{array}$$

All “administrative steps” are quadratic in n , so we have

$$T(n) = 8T\left(\frac{n}{2}\right) + \Theta(n^2).$$

Master Theorem ($a = 8, b = 2$, case (1)): $T(n) = \Theta(n^3)$.

Strassen Matrix multiplication (I)

Multiplying two matrices can be done in $\Theta(n^{2.8})$ ($\log_2 7 \approx 2.8$).
Split A and B into 4 submatrices of order $n/2$:

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \cdot \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

where

$$\begin{aligned} C_{11} &= A_{11}B_{11} + A_{12}B_{21} & C_{12} &= A_{11}B_{12} + A_{12}B_{22} \\ C_{21} &= A_{21}B_{11} + A_{22}B_{21} & C_{22} &= A_{21}B_{12} + A_{22}B_{22} \end{aligned}$$

We always have to compute each of the C_{ij} . Can we do that with less recursive calls?

Strassen: yes, with 7 calls (in stead of 8) ... and a lot of additional administration.

Strassen Matrix multiplication (II)

Define

$$\begin{aligned} S_1 &= B_{12} - B_{22} & S_2 &= A_{11} + A_{12} & S_3 &= A_{21} + A_{22} \\ S_4 &= B_{21} - B_{11} & S_5 &= A_{11} + A_{22} & S_6 &= B_{11} + B_{22} \\ S_7 &= A_{12} - A_{22} & S_8 &= B_{21} + B_{22} & S_9 &= A_{11} - A_{21} \\ S_{10} &= B_{11} + B_{12} \end{aligned}$$

Define

$$\begin{aligned} P_1 &= A_{11} \cdot S_1 & P_2 &= S_2 \cdot B_{22} & P_3 &= S_3 \cdot B_{11} \\ P_4 &= A_{22} \cdot S_4 & P_5 &= S_5 \cdot S_6 & P_6 &= S_7 \cdot S_8 \\ P_7 &= S_9 \cdot S_{10} \end{aligned}$$

Then

$$\begin{aligned} C_{11} &= P_5 + P_4 - P_2 + P_6 & C_{12} &= P_1 + P_2 \\ C_{21} &= P_3 + P_4 & C_{22} &= P_5 + P_1 - P_3 - P_7 \end{aligned}$$

And note we have computed C_{11} , C_{12} , C_{21} and C_{22} using only 7 multiplications of size $n/2$.



Strassen Matrix multiplication (III)

- Strassen's algorithm does 7 recursive calls on a matrix of order $\frac{n}{2}$.
- All “administrative steps” are quadratic in n .

We now have

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

Master Theorem ($a = 7$, $b = 2$, $\log_2 7 \approx 2.8074$, case (1)):

$$T(n) = \Theta(n^{\log 7}) \approx \Theta(n^{2.8}).$$

- Later improvements have been made. e.g. to $\Theta(n^{2.3754})$, but Strassen is what is usually implemented.

Matrix inversion (I)

Compute the inverse A^{-1} of a matrix of size $n \times n$. First assume: $A^T = A$. (This is a serious limitation!) So we can write A as

$$\begin{pmatrix} B & C^T \\ C & D \end{pmatrix}$$

Let $S := D - CB^{-1}C^T$. Use standard matrix calculations to show:

$$A^{-1} = \begin{pmatrix} B^{-1} + B^{-1}C^T S^{-1}CB^{-1} & -B^{-1}C^T S^{-1} \\ -S^{-1}CB^{-1} & S^{-1} \end{pmatrix}$$

So we have expressed A^{-1} in terms of B^{-1} and S^{-1} and standard matrix operations. We find that

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

By Master Theorem ($a = b = 2$, $\gamma = \log_2 2 = 1 < \log 7$, case (3)):

$$T(n) = \Theta(n^{\log 7})$$

Check the regularity condition!

Matrix inversion (II)

If A is not symmetric we have (by simple matrix calculations)

$$A^{-1} = (A^T \cdot A)^{-1} \cdot A^T$$

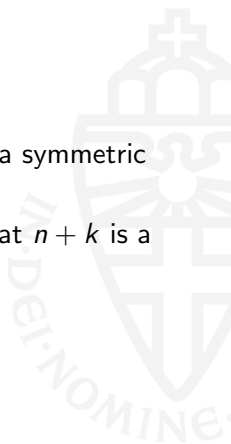
and (!) $A^T \cdot A$ is always symmetric.

Conclusion: A^{-1} can be computed from an inverse of a symmetric matrix + matrix multiplications, so it is in $\Theta(n^{\log 7})$.

In case n is not a power of 2, there is a $k < n$ such that $n + k$ is a power of 2. Apply the algorithm to

$$\begin{pmatrix} A & 0 \\ 0 & I_k \end{pmatrix}^{-1} = \begin{pmatrix} A^{-1} & 0 \\ 0 & I_k \end{pmatrix}$$

(Exercise: Show that this is still $\Theta(n^{\log 7})$.)



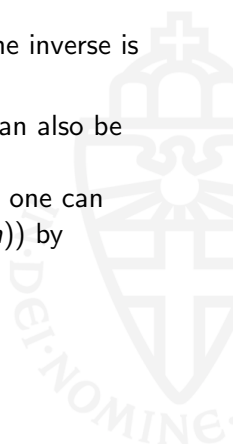
Matrix calculations

Some remarks

- If matrix multiplication is $\Theta(f(n))$, then taking the inverse is also $\Theta(f(n))$.
- One obtains “inverse” from “multiplication”. It can also be done the other way around:

Given an algorithm for matrix inverse in $\Theta(f(n))$, one can obtain a matrix multiplication algorithm in $\Theta(f(n))$ by

$$\begin{pmatrix} I & A & 0 \\ 0 & I & B \\ 0 & 0 & I \end{pmatrix}^{-1} = \begin{pmatrix} I & -A & AB \\ 0 & I & -B \\ 0 & 0 & I \end{pmatrix}$$

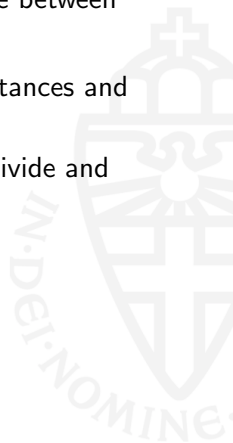




Minimum distance between points

Given n points in \mathbb{R}^2 , determine the minimum distance between two points.

- Can be done in $\Theta(n^2)$ steps. (Compute all n^2 distances and determine the minimum.)
- It can also be done in $\Theta(n \log n)$. We present a divide and conquer algorithm MinDist.
- Let a set of points P of size n in \mathbb{R}^2 be given. We first do some preprocessing:
 - sort P according to the x -coordinate: R_x ,
 - sort P according to the y -coordinate: R_y ,
 - Cost: $\Theta(n \log n)$.



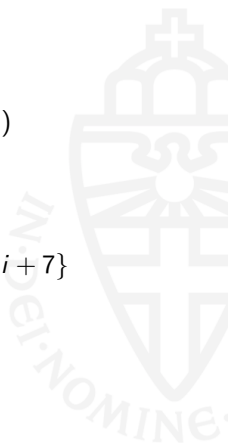
MinDist algorithm (I)

MinDist(P) =

```
if # $P \leq 3$  then compare the distances ( $\leq 3$ )
else  $P = P_1 \cup P_2, \quad x_\ell, \quad \#P_i \leq \left\lceil \frac{\#P}{2} \right\rceil,$ 
 $\forall (x, y) \in P_1 (x \leq x_\ell), \forall (x, y) \in P_2 (x \geq x_\ell)$ 
 $\delta_1 := \text{MinDist}(P_1), \delta_2 := \text{MinDist}(P_2),$ 
 $\delta := \min(\delta_1, \delta_2)$ 
consider  $P' := \{(x, y) \mid x_\ell - \delta < x < x_\ell + \delta\},$ 
using  $R_y$  order  $P'$  into  $[(x_1, y_1), \dots, (x_m, y_m)]$ 
determine, for every  $i \leq m,$ 
 $\varepsilon_i := \min\{d((x_i, y_i), (x_j, y_j)) \mid i < j \leq i + 7\}$ 
 $\delta_3 := \min\{\varepsilon_i \mid 1 \leq i \leq m\}$ 
return  $\min(\delta_3, \delta)$ 
```

MinDist algorithm (II)

- $P = P_1 \cup P_2, \quad x_\ell, \quad \#P_i \leq \left\lceil \frac{\#P}{2} \right\rceil$
- $\forall (x, y) \in P_1 (x \leq x_\ell), \quad \forall (x, y) \in P_2 (x \geq x_\ell)$
- $\delta_1 := \text{MinDist}(P_1), \quad \delta_2 := \text{MinDist}(P_2), \quad \delta := \min(\delta_1, \delta_2)$
- $P' := \{(x, y) \mid x_\ell - \delta < x < x_\ell + \delta\}$
- $P' = [(x_1, y_1), \dots, (x_m, y_m)]$
- For every $i \leq m, \quad \varepsilon_i := \min\{d((x_i, y_i), (x_j, y_j) \mid i < j \leq i + 7)\}$
- $\delta_3 := \min\{\varepsilon_i \mid 1 \leq i \leq m\}$
- **return** $\min(\delta_3, \delta)$



MinDist algorithm (III)

The complexity of MinDist, given a set of points P with $\#P = n$, $T(n)$ is:

$$T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n).$$

This is a well-known equation (but one can also use the Master Theorem):

$$T(n) = \Theta(n \log n).$$

Together with the preprocessing work (of creating the R_x and R_y orderings), this yields an algorithm of complexity $\Theta(n \log n)$.

The Master Theorem does not always apply (I)

$$T(n) = 2T\left(\frac{n}{2}\right) + n \log n.$$

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

Then, with $\gamma := \log_b a$

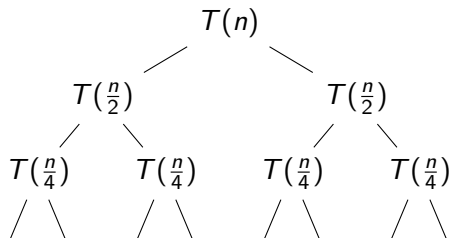
- 1 $T(n) = \Theta(n^\gamma)$ if $f(n) = \mathcal{O}(n^d)$ for some $d < \gamma$.
- 2 $T(n) = \Theta(n^\gamma \log n)$ if $f(n) = \Theta(n^\gamma)$.
- 3 $T(n) = \Theta(f(n))$ if $f(n) = \Omega(n^d)$ for some $d > \gamma$ and $\exists c < 1 \exists N \forall n > N (af(\frac{n}{b}) \leq cf(n))$.



The Master Theorem does not always apply (II)

$$T(n) = 2T\left(\frac{n}{2}\right) + n \log n.$$

We apply the recursion tree method



$$n \log n$$

$$2 \frac{n}{2} \log \frac{n}{2} = n \log \frac{n}{2}$$

$$4 \frac{n}{4} \log \frac{n}{4} = n \log \frac{n}{4}$$

So we have

$$T(n) = \sum_{i=0}^{\log n} n \log \frac{n}{2^i} + 2^{\log n} \Theta(1) = \sum_{i=0}^{\log n} n \log \frac{n}{2^i} + cn$$

Example continued

$$T(n) = 2T\left(\frac{n}{2}\right) + n \log n.$$

Using the recursion tree method, we have found

$$T(n) = \sum_{i=0}^{\log n} n \log \frac{n}{2^i} + cn$$

and by some further computation:

$$\begin{aligned} \sum_{i=0}^{\log n} n \log \frac{n}{2^i} &= n \sum_{i=0}^{\log n} (\log n - i) \\ &= n(\log^2 n - \sum_{i=0}^{\log n} i) \\ &= n(\log^2 n - \frac{1}{2}(\log n(\log n + 1))) \end{aligned}$$

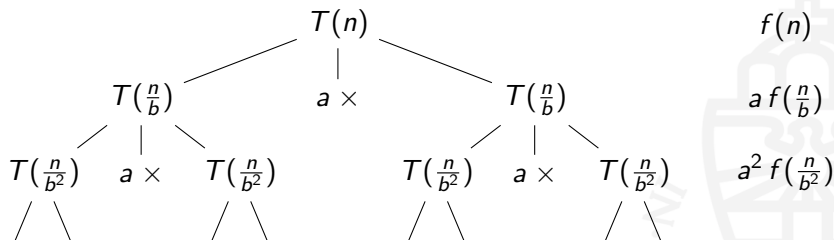
so we conclude (and this can be proven correct):

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



Proof sketch of the Master Theorem (I)

Suppose $a \geq 1$ and $b > 1$ and $T(n) = aT(\frac{n}{b}) + f(n)$. Then



- There are $\log_b n$ layers
- There are $a^{\log_b n}$ ($= n^{\log_b a}$) leaves

$$T(n) = \Theta(n^{\log_b a}) + \sum_{j=0}^{\log_b n - 1} a^j f\left(\frac{n}{b^j}\right).$$

Proof sketch of the Master Theorem (II)

Let $T(n) = aT(\frac{n}{b}) + f(n)$ (with $a \geq 1, b > 1$) and let $\gamma := \log_b a$.

$$T(n) = \Theta(n^\gamma) + \sum_{j=0}^{\log_b n-1} a^j f\left(\frac{n}{b^j}\right).$$

- 1 If $f(n) = \mathcal{O}(n^d)$ for some $d < \gamma$, then $T(n) = \Theta(n^\gamma)$
- 2 If $f(n) = \Theta(n^\gamma)$, then $T(n) = \Theta(n^\gamma \log n)$.
- 3 If $f(n) = \Omega(n^d)$ for some $d > \gamma$,
and $\exists c < 1 \exists N_0 \forall n > N_0 (af(\frac{n}{b}) \leq cf(n))$, then $T(n) = \Theta(f(n))$.

