<u>Conclusion</u>: multiply two n-bit integers

* school method: $O(n^2)$
* Karatsuba (1960): $O(n^{\log 3})$
  - in practice: faster when $n \geq 500$
  - used in practice (cryptography)
* Schönhage and Strassen (1971):

  $O(n \log n \log\log n)$

  - based on Fast Fourier Transforms
  - in practice: faster than Karatsuba for very large n
  - used in practice
* Fürer (2007): $O(n \log n \cdot 2^{O(\log^* n)})$
  - theoretical algorithm.

Harvey, van der Hoeven, 2019: O(n log n)

We have solved recurrences using unfolding.

Here is an equivalent way to solve them: look at the <u>recursion tree</u>.

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ n + 2 \cdot T(n/2) & \text{if } n \geq 2 \end{cases}$$

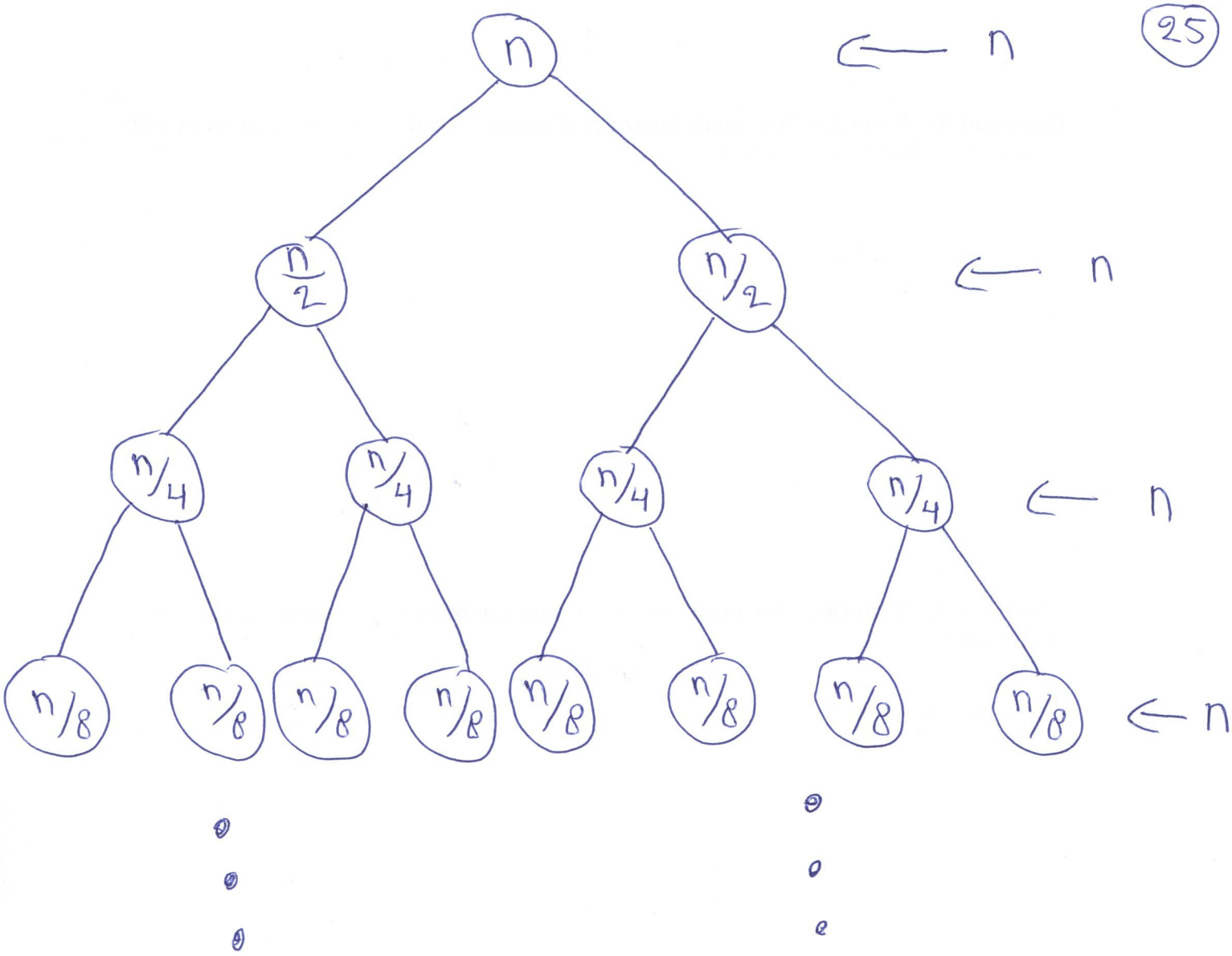
root level: n for the merge step

left child: $T(n/2)$

right child: $T(n/2)$

by repeating this, we get the following tree:

(value in each node: time for the merge step of the corresponding recursive call)

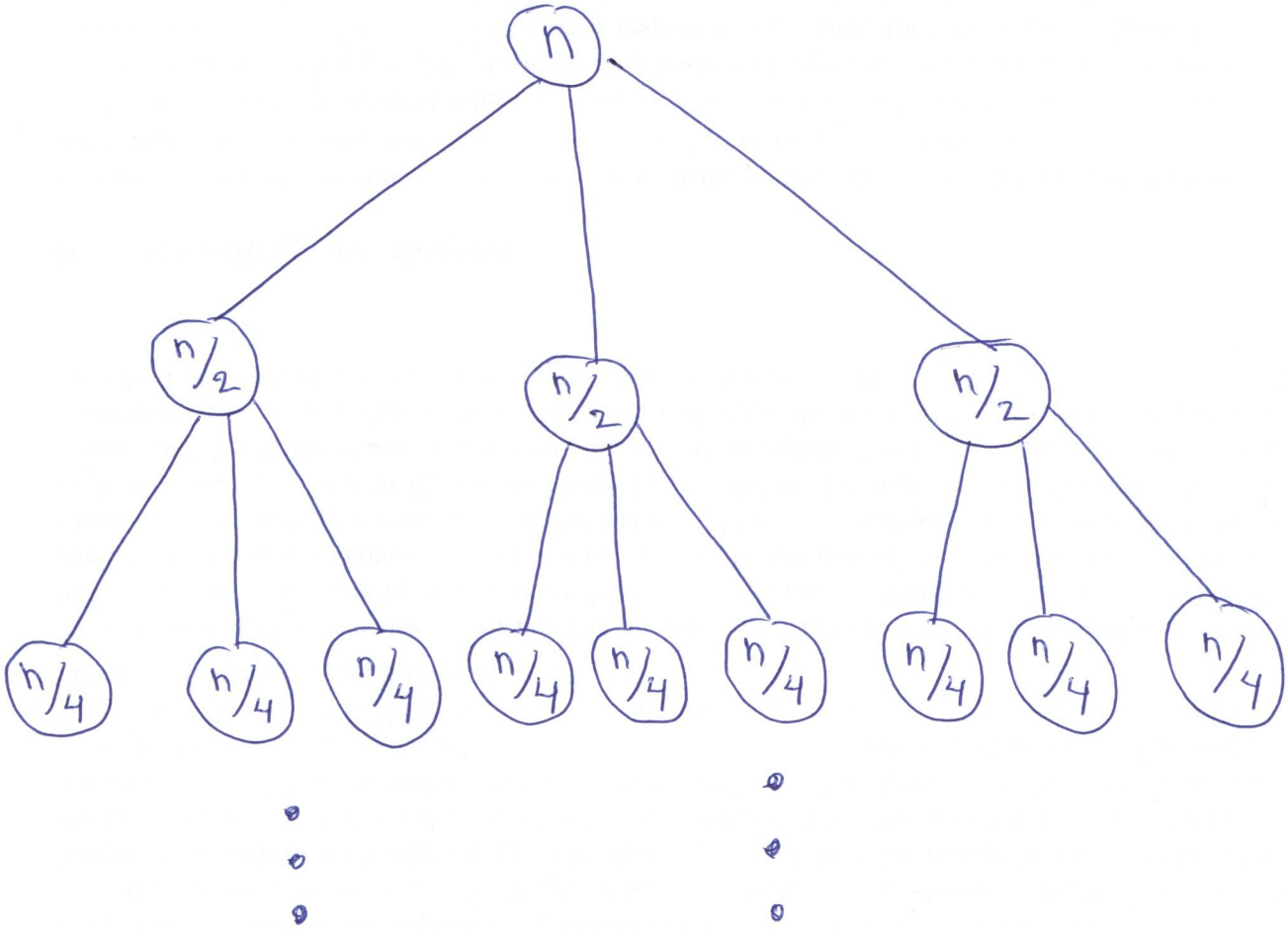


At each level : n

number of levels = recursion depth = $O(\log n)$

$\therefore T(n) = O(n \log n)$

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ n + 3 \cdot T(n/2) & \text{if } n \geq 2 \end{cases}$$



Assume $n = 2^k$.

level 0 (root) : 1 call, size n

level 1 : 3 calls, size $n/2$

level 2 : 3^2 calls, size $n/2^2$

level i : 3^i calls, size $n/2^i$

level k: 3^k calls, size $\frac{n}{2^k} = 1$

$$\therefore T(n) = \sum_{i=0}^{k} 3^i \cdot n/2^i$$

Same as on the bottom of page 21.

Master Theorem

$$T(n) = \begin{cases} 1 & \text{if } n = 1, \\ a \cdot T(n/b) + n^d & \text{if } n \geq 2. \end{cases}$$

$a \geq 1,\ b > 1,\ d \geq 0.$

if $d > \log_b a$: $T(n) = O(n^d)$

if $d = \log_b a$: $T(n) = O(n^d \log n)$

if $d < \log_b a$: $T(n) = O(n^{\log_b a})$

Merge Sort: $a = 2,\ b = 2,\ d = 1$: $\log_b a = d$

$\therefore T(n) = O(n \log n)$

Karatsuba: $a = 3,\ b = 2,\ d = 1$: $\log_b a > d$

$\therefore T(n) = O(n^{\log_b a}) = O(n^{\log 3})$.

Binary Search: $a = 1,\ b = 2,\ d = 0$: $\log_b a = d$

$\therefore T(n) = O(\log n)$

Matrix Multiplication

<u>Input</u>: $n \times n$ matrices X and Y

<u>Output</u>: product $Z = XY$, which is an $n \times n$ matrix.

Z_{ij} = entry in row i and column j of Z

= dot product of row i of X and column j of Y

$$= \sum_{k=1}^{n} X_{ik} Y_{kj}$$

* each entry of Z can be computed in $O(n)$ time
* Z has n^2 entries

$\therefore$ Z can be computed in $O(n^3)$ time.

Is there a faster algorithm?

yes, using divide-and-conquer.

Assume n is a power of 2.

Divide both X and Y into 4 $\frac{n}{2} \times \frac{n}{2}$ matrices:

$$X = \begin{pmatrix} A & B \\ C & D \end{pmatrix}, \quad Y = \begin{pmatrix} E & F \\ G & H \end{pmatrix}$$

Then:

$$XY = \begin{pmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{pmatrix}$$

To multiply two $n \times n$ matrices:

* 8 times: recursively multiply two $\frac{n}{2} \times \frac{n}{2}$ matrices
* 4 times: add two $\frac{n}{2} \times \frac{n}{2}$ matrices

$T(n)$ = time to multiply two $n \times n$ matrices.

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 8 \cdot T\left(\frac{n}{2}\right) + O(n^2) & \text{if } n \geq 2 \end{cases}$$

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

Strassen (1969):

* 7 recursive calls
* 18 times: add two $\frac{n}{2} \times \frac{n}{2}$ matrices

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 7 \cdot T\left(\frac{n}{2}\right) + O(n^2) & \text{if } n \geq 2 \end{cases}$$

$$\therefore T(n) = O(n^{\log 7}), \quad \log 7 \approx 2.81$$

$$P_1 = A(F-H) \qquad P_5 = (A+D)(E+H)$$

$$P_2 = (A+B)H \qquad P_6 = (B-D)(G+H)$$

$$P_3 = (C+D)E \qquad P_7 = (A-C)(E+F)$$

$$P_4 = D(G-E)$$

Then:

$$XY = \begin{pmatrix} P_5 + P_4 - P_2 + P_6 & P_1 + P_2 \\ P_3 + P_4 & P_1 + P_5 - P_3 - P_7 \end{pmatrix}$$

World record:

Coppersmith and Winograd (1990):

$$O(n^{2.376})$$

Breaking News

Alman and Vassilevska-Williams

2.37286

Symposium on Discrete Algorithms

January 2021

Lower bound: $\Omega(n^2)$.