CSE 102 11-3-22

Sec 4.2 (P.75-83) in CLRS

Strassen's algorithm

consider 2 $n \times n$ sq. matrices A, B and multiplying them

$$C = A \cdot B$$

Recall

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} \quad \begin{cases} \text{for} \\ 1 \le i \le n \\ 1 \le j \le n \end{cases}$$

Run time:

Basic OP.: multiplication of floating Point numbers,

so algorithm runs in time $\Theta(n^3)$.

Divide & Conquer

case: n is exact power of 2.

$$A = \left(\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right)$$

size = $\frac{n}{2} \times \frac{n}{2}$

$$B = \left(\begin{array}{c|c} B_{11} & B_{12} \\ \hline B_{21} & B_{22} \end{array}\right)$$

$$C_{11} = A_{11} \cdot B_{11} + A_{12} \cdot B_{21}$$

$$C_{12} = A_{11} \cdot B_{12} + A_{12} \cdot B_{22}$$

$$C_{21} = A_{21} \cdot B_{11} + A_{22} \cdot B_{21}$$

$$C_{22} = A_{21} \cdot B_{12} + A_{22} \cdot B_{22}$$

let

$$C = \left(\begin{array}{c|c} C_{11} & C_{12} \\ \hline C_{21} & C_{22} \end{array}\right)$$

Exercises

- Prove this is equivalent to ordinary Matrix multiplication.
- write Pseudo-code (P. 77 of text).

let $T(n)$ = # of Basic ops necessary on matrices of size $n \times n$

observe

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

↑ const. cost of dividing and combining. Book has $\Theta(n^2)$ here since they're also counting additions.

Apply Master Theorem to

$$T(n) = 8T\left(\frac{n}{2}\right) + 1$$

Comp: $n^{\log_2 8} = n^3$ to $1 = n^0$

case 1: $T(n) = \Theta(n^3)$

no improvement !!!

In 1969 Volker Strassen:

figured out how to do this

with only 7 submatrix multiplications

we get new recurrence:

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

$\Rightarrow$ 0 by Master thm

Compare: $n^{\log_2 7}$ to $n^0 = 1$

↑ winner

Case 1:

$$T(n) = \Theta\left(n^{\log_2 7}\right)$$

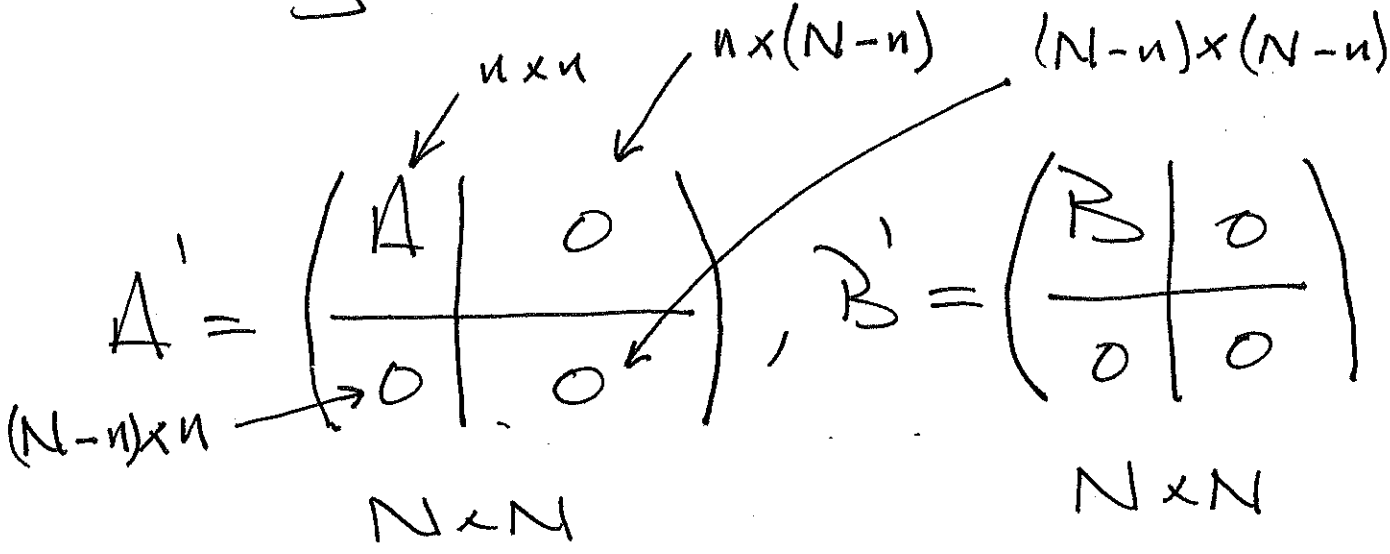
$$= \Theta\left(n^{2.8073\ldots}\right)$$

$$= O\left(n^3\right)$$

Better than original.

What if n is not an exact Power of 2?

Pad A and B with 0's to get



with N an exact Pow. of 2

$$C' = \left(\begin{array}{c|c} C & 0 \\ \hline 0 & 0 \end{array}\right)$$

Exercise:

let N be smallest pow of 2 that is greater than or eq. to n. show

$$N^{\log_2(7)} = \Theta\left(n^{\lg(7)}\right)$$

claim: $N = 2^{\lceil \lg(n) \rceil}$

Dynamic Programming

Problem: Compute Binomial Coefficients

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} \qquad 0 \le k \le n$$

Pascal's identity

$$\binom{n}{k} = \begin{cases} 1 & \text{if } k=0,\ k=n \\ \binom{n-1}{k-1} + \binom{n-1}{k} & \text{if } 0<k<n \end{cases}$$

```
                1
              1   1
            1   2   1
          1   3   3   1
        1   4   6   4   1
```

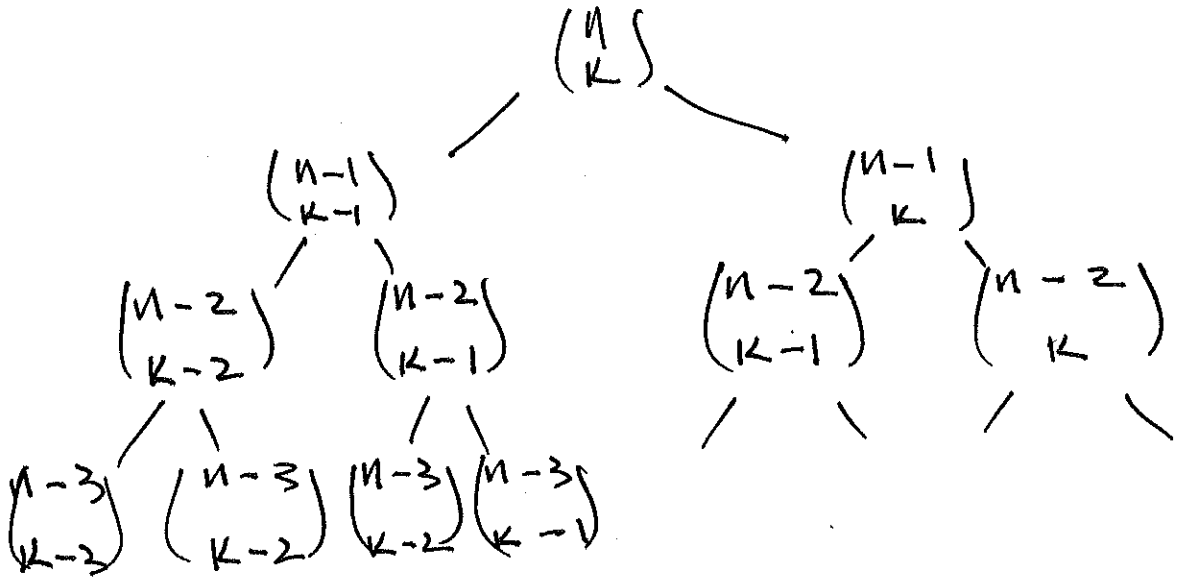
Divide & Conquer soln

BinCoeff (n, k)

1. if k=0 or k=n
2. return 1
3. else
4. return BinCoeff(n-1, k-1) + BinCoeff(n-1, k)

Runtime: if $n = 2k$

$$\binom{n}{k} = \Theta\left(\frac{2^n}{\sqrt{n}}\right) = \Theta\left(\frac{4^k}{\sqrt{k}}\right)$$



Dynamic Programming

construct a table containing solutions to subinstances previously computed. Do this iteratively

Memoization

same thing but recursive.

DynBinCoef(n, k)

1. C[0] = 1
2. C[1 ... k] = (0, 0, ..., 0)
3. for i = 1 to n
4. for i = k down to 1
5. C[i] = C[i-1] + C[i]
6. return C[k]

Exercise

improve this by simplifying.

- don't add zeros
- don't compute unneeded values

Runtime: $\Theta(nK)$.

Problem: Coin changing

Suppose we have n types of coins of values $(d_1, d_2, \ldots, d_n)$ with $d_i \geq 1$ a pos. integer. Also have an amount N that we wish to pay. We have an unlimited supply in each denomination.

Two questions:

- what is the least number of coins necessary to pay ammount N ?
- Exactly which coins, i.e. how many of each type are to be disbursed to pay N with fewest coins ?