cost of multiplying 2 $n \times n$ matrices where $n = 2^k$:

$$\begin{array}{ccc} A & B & C \end{array}$$

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

recurrence

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

book has $\Theta(n^2)$ since they count additions.

master thm : $T(n) = 8T(\frac{n}{2}) + 1$

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

<u>case 1</u> : Pick any ε s.t. $0 < \varepsilon \leq 3$

$$1 = n^0 = O(n^{\log_2 8 - \varepsilon}) \quad \checkmark$$

so $T(n) = \Theta(n^3)$.

No improvement over std. iterative algorithm :

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

In 1969, Volker Strassen discovered a way to do this with only 7 matrix multiplications.

recurrence

$$T(n) = 7T\left(\frac{n}{2}\right) + 1$$

book has $\Theta(n^2)$ here.

master thm:

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

<u>case 1</u>: pick $\varepsilon > 0$ and $\varepsilon \leq \log_2 7$

$$n^0 = O\left(n^{\log_2 7 - \varepsilon}\right) \checkmark$$

$$\therefore T(n) = \Theta\left(n^{\log_2 7}\right) = \Theta\left(n^{2.8073\ldots}\right)$$

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

what if n is <u>not</u> an exact power of 2? Idea: Pad A, B with 0's until its size is N, an exact power of 2.

<u>Exercise</u>

Let N be the smallest power of 2 greater than or equal to N (i.e. $N = 2^{\lceil \lg n \rceil}$.) define

$$\underset{N\times N}{A'} = \begin{array}{cc} n & N-n \\ \left(\begin{array}{c|c} \underset{n\times n}{A} & 0 \\ \hline 0 & 0 \end{array}\right) & \end{array} \qquad \underset{N\times N}{B'} = \begin{array}{cc} n & N-n \\ \left(\begin{array}{c|c} \underset{n\times n}{B} & 0 \\ \hline 0 & 0 \end{array}\right) & \end{array}$$

where A', B' are $N\times N$.

show that: $N^{\lg_2(7)} = \Theta(n^{\lg(7)})$.

Exercise

show how to do this even if A and B are not square!!

$$\underset{p\times q}{A}\cdot\underset{q\times r}{B}=\underset{p\times r}{C}$$

chap 9:

new problem: find both min and max of an array, return the pair (min, max).

$$A[1 \ldots\ldots\ldots\ldots n]$$
$$= (A_1, A_2, A_3, \ldots, A_n)$$

Iterative solution: cost $= 2(n-1) = 2n-2$
$= \Theta(n)$

define

(basic op = comp)

<u>cost</u>

<u>min(x, y)</u>

```
return (x < y ? x : y)        1
```

<u>max(x, y)</u>

```
return (x < y ? y : x)        1
```

<u>MinMax(A, p, r)</u> (pre: $p \le r$)

```
if p = r
    return (A[p], A[p])
else
    q = ⌊(p+r)/2⌋
    (m1, M1) = MinMax(A, p, q)
    (m2, M2) = MinMax(A, q+1, r)
    return (min(m1, m2), max(M1, M2))
```

Exercise:

Prove correctness (induction on $m = r - p + 1$)

Runtime:

$$T(n) = \begin{cases} 0 & n = 1 \checkmark \\ T(\lceil \frac{n}{2} \rceil) + T(\lfloor \frac{n}{2} \rfloor) + 2 & n \geq 2 \checkmark \end{cases}$$

exact solution: $T(n) = 2n - 2 \sim 2n$

check: $RHS = T(\lceil \frac{n}{2} \rceil) + T(\lfloor \frac{n}{2} \rfloor) + 2$

$$= \left(2\lceil \tfrac{n}{2} \rceil - 2\right) + \left(2\lfloor \tfrac{n}{2} \rfloor - \cancel{2}\right) + \cancel{2}$$

$$= 2\left(\lceil \tfrac{n}{2} \rceil + \lfloor \tfrac{n}{2} \rfloor\right) - 2$$

$$= 2n - 2$$

$$= T(n) = LHS.$$

Exercise

give a D&C algorithm that does this with exactly: $\lceil \frac{3n}{2} \rceil - 2$ comparisons, i.e. cost $\sim \frac{3}{2} n$

$$\underbrace{A_1 \; A_2} \; \underbrace{A_3 \; A_4} \; \cdots \; A_{n-1} \; A_n$$

sec. 9.2 : The selection Problem

Defn
The i^{th} order-statistic of an array $A[1 \ldots n]$ (consisting of distinct elements) is the the i^{th} smallest element. equivalently: if A is sorted, it is the element $A[i]$.

equivalently: The i^{th} order statistic is the unique element in $A[1 \ldots n]$ that is greater than exactly $(i-1)$ other elements. $(1 \le i \le n)$

$i = 1$ gives minimum
$i = n$ gives maximum.

also: greater than or equal to exactly i elements.

Problem

Given $A[1 \ldots n]$, distinct elements, and $1 \le i \le n$, determine the i^{th} order statistic.

<u>solution</u>

sort the array, and return A[i]

cost of sort = $\Omega(n \log n)$

can find i^{th} O.S. without sorting

<u>RandSelect(A, p, r, i)</u> {Pre: $1 \le i \le r-p+1$
Pre: distinct ele.

```
if p = r
    return A[p]
q = RandPartition(A, p, r)
k = q - p + 1        // length of A[p...q]
if k = i
    return A[q]
else if i < k
    return RandSelect(A, p, q-1, i)
else   // k < i
    return RandSelect(A, q+1, r, i-k)
```

Recall RandPartition(A, p, r) does

$$A[p \cdots (q-1)] < A[q] < A[(q+1) \cdots r]$$

<u>Exercise</u>: Prove correctness.

RandSelect() is similar

- QuickSort()
- BinarySearch()

next time: Runtime.

answer: $\Theta(n)$ in avg. case.

- <u>Exercise</u>: Write non-random version
- <u>Exercise</u>: analyze runtime of non-random version in worst case.