More on
Divide and Conquer

The divide-and-conquer design paradigm

1. ***Divide*** the problem (instance) into subproblems.
2. ***Conquer*** the subproblems by solving them recursively.
3. ***Combine*** subproblem solutions.

Example: merge sort

1. ***Divide:*** Trivial.
2. ***Conquer:*** Recursively sort 2 subarrays.
3. ***Combine:*** Linear-time merge.

$$T(n) = 2\,T(n/2) + O(n)$$

 # subproblems

subproblem size

work dividing and combining

Master theorem:

$$T(n) = a\,T(n/b) + f(n)$$

CASE 1: 若$f(n) = O(n^{\log_b a-\varepsilon})$，則 $T(n) = \Theta(n^{\log_b a})$

CASE 2: 若 $f(n) = \Theta(n^{\log_b a})$，則 $T(n) = \Theta(n^{\log_b a} \lg n)$

CASE 3: 若$f(n) = \Omega(n^{\log_b a+\varepsilon})$，且$af(n/b) \le cf(n)$，則 $T(n) = \Theta(f(n))$

Merge sort: $a = 2$, $b = 2 \Rightarrow n^{\log_b a} = n$

$\Rightarrow$ CASE 2 $T(n) = \Theta(n \lg n)$.

Binary search

Find an element in a sorted array:

1. Divide: Check middle element.

2. Conquer: Recursively search 1 subarray.

3. Combine: Trivial.

Example: Find 9

3	5	7	8	9	12	15

Binary search

Find an element in a sorted array:

1. Divide: Check middle element.

2. Conquer: Recursively search 1 subarray.

3. Combine: Trivial.

Example: Find 9

3	5	7	8	9	12	15

Binary search

Find an element in a sorted array:

1. Divide: Check middle element.

2. Conquer: Recursively search 1 subarray.

3. Combine: Trivial.

Example: Find 9

3 5 7 8 $\boxed{9\ \ 12\ \ 15}$

Binary search

Find an element in a sorted array:

1. Divide: Check middle element.

2. Conquer: Recursively search 1 subarray.

3. Combine: Trivial.

Example: Find 9

3 5 7 8 9 12 15

Binary search

Find an element in a sorted array:

1. Divide: Check middle element.

2. Conquer: Recursively search 1 subarray.

3. Combine: Trivial.

Example: Find 9

3 5 7 8 $\boxed{9}$ 12 15

Binary search

Find an element in a sorted array:

1. Divide: Check middle element.

2. Conquer: Recursively search 1 subarray.

3. Combine: Trivial.

Example: Find 9

3 5 7 8 9 12 15

Recurrence for binary search

$$T(n) = 1\ T(n/2) + \Theta(1)$$

 # subproblems

subproblem size

work dividing and combining

$a=1,\ b=2,\ n^{\log_b a} = n^0 \Rightarrow$ CASE 2

$\Rightarrow T(n) = \Theta(\lg n)$.

Powering a number

Problem: Compute a^n, where $n \in \mathbb{N}$.

Naive algorithm: $\Theta(n)$.

Divide-and-conquer algorithm:

$$a^n = \begin{cases} a^{n/2} \cdot a^{n/2} & \text{if } n \text{ is even;} \\ a^{(n-1)/2} \cdot a^{(n-1)/2} \cdot a & \text{if } n \text{ is odd.} \end{cases}$$

$$T(n) = T(n/2) + \Theta(1) \;\Rightarrow\; T(n) = \Theta(\lg n)\,.$$

Fibonacci numbers

Recursive definition:

$$F_n = \begin{cases} 0 & \text{if } n = 0; \\ 1 & \text{if } n = 1; \\ F_{n-1} + F_{n-2} & \text{if } n \geq 2. \end{cases}$$

0 1 1 2 3 5 8 13 21 34 …

Naive recursive algorithm: $\Omega(\varphi^n)$ (exponential time), where $\phi = (1 + \sqrt{5})/2$ is the ***golden ratio***.

Computing Fibonacci numbers

Naive recursive squaring:

$F_n = \phi^n/\sqrt{5}$ rounded to the nearest integer. 5

- Recursive squaring: $\Theta(\lg n)$ time.
- This method is unreliable, since floating-point arithmetic is prone to round-off errors.

Bottom-up:

- Compute $F_0, F_1, F_2, \ldots, F_n$ in order, forming each number by summing the two previous.
- Running time: $\Theta(n)$.

Recursive squaring

Theorem: $$\begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n .$$

Algorithm: Recursive squaring.

Time = $\Theta(\lg n)$.

Proof of theorem. (Induction on n.)

Base ($n = 1$): $$\begin{bmatrix} F_2 & F_1 \\ F_1 & F_0 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^1 .$$

Recursive squaring

Inductive step ($n \geq 2$):

$$\begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix} = \begin{bmatrix} F_n & F_{n-1} \\ F_{n-1} & F_{n-2} \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n-1} \cdot \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n}$$

■

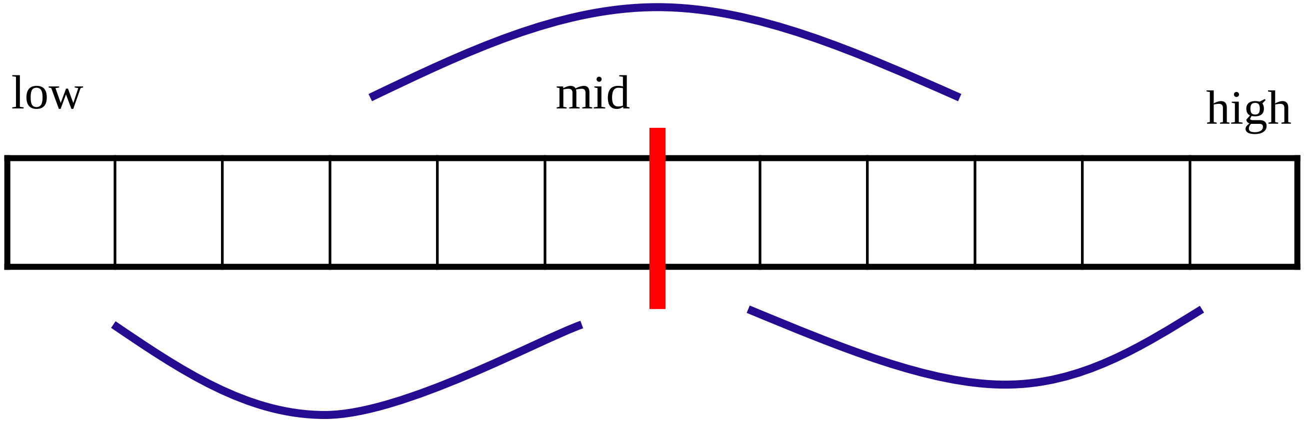
Maximum subarray problem

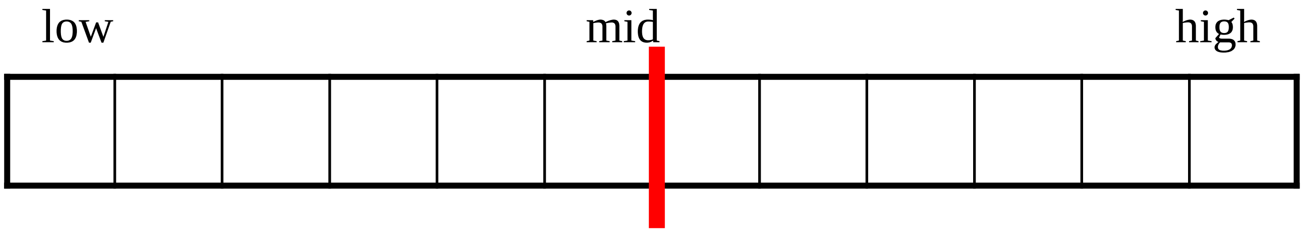
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
13	*-3*	*-25*	*20*	*-3*	*-16*	*-23*	*18*	*20*	*-7*	*12*	*-5*	*-22*	*15*	*-4*	*7*

Input: **An Array of numbers**

Output: **A subarray with the maximum sum**

Observation:





- **Subproblem:** Find a maximum subarray of A[*low* .. *high*]. In original call, *low* =1, *high* = n.
- **Divide** the subarray into two subarrays. Find the midpoint *mid* of the subarrays, and consider the subarrays A[*low* ..*mid*] And A[*mid* +1..*high*]
- **Conquer** by finding a maximum subarrays of A[*low* ..*mid*] and A[*mid*+1..*high*].
- **Combine** by finding a maximum subarray that crosses the midpoint, and using the best solution out of the three.

```
FIND-MAX-CROSSING-SUBARRAY(A, low, mid,  high)
// Find a maximum subarray of the form A[i..mid].
leftsum =-∞; sum= 0;
for i = mid downto low
   sum =sum + A[i] ;
    if sum > leftsum
          leftsum = sum
          maxleft = i
// Find a maximum subarray of the form A[mid + 1.. j].
rightsum =- ∞; sum= 0;
for j = mid + 1 to high
   sum = sum + A[j]
    if sum > rightsum
         rightsum = sum
         maxright = j
// Return the indices and the sum of the two subarrays.
return (maxleft, maxright, leftsum +rightsum)
```

```
FIND-MAXIMUM-SUBARRAY(A, low, high)
if high == low
   return (low, high, A[low]) // base case: only one element
else mid = ⌊(low +high)/2⌋
  (leftlow, lefthigh, leftsum) =
        FIND-MAXIMUM-SUBARRAY(A, low, mid)
  (rightlow, righthigh, rightsum) =
        FIND-MAXIMUM-SUBARRAY(A, mid + 1, high)
  (crosslow, crosshigh, cross-sum) =
        FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
if leftsum >= rightsum and leftsum >= crosssum
        return (leftlow, lefthigh, leftsum)
elseif rightsum >= leftsum and rightsum >= crosssum
        return (rightlow, righthigh, rightsum)
else return (crosslow, crosshigh, crosssum)
```

$$\boxed{T(n) = 2T(n/2) + \Theta(n)}$$

$$\boxed{T(n) = \Theta(n \log n)}$$

Initial call: FIND-MAXIMUM-SUBARRAY(A, 1, n)

Matrix multiplication

Input: $A = [a_{ij}],\ B = [b_{ij}].$

Output: $C = [c_{ij}] = A \cdot B.$

$$\left.\begin{aligned} &A = [a_{ij}],\ B = [b_{ij}]. \\ &C = [c_{ij}] = A \cdot B. \end{aligned}\right\} \quad i, j = 1, 2, \ldots, n.$$

$$\begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \cdot \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nn} \end{bmatrix}$$

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

Standard algorithm

for $i \leftarrow 1$ **to** n

do for $j \leftarrow 1$ **to** n

do $c_{ij} \leftarrow 0$

for $k \leftarrow 1$ **to** n

do $c_{ij} \leftarrow c_{ij} + a_{ik} \cdot b_{kj}$

Running time = $\Theta(n^3)$

Divide-and-conquer algorithm

IDEA:

$n \times n$ matrix = 2×2 matrix of $(n/2) \times (n/2)$ submatrices:

$$\left[\begin{array}{c|c} r & s \\ \hline t & u \end{array}\right] = \left[\begin{array}{c|c} a & b \\ \hline c & d \end{array}\right] \cdot \left[\begin{array}{c|c} e & f \\ \hline g & h \end{array}\right]$$

$$C \quad = \quad A \quad \cdot \quad B$$

$$\begin{cases} r = ae + bg \\ s = af + bh \\ t = ce + dg \\ u = cf + dh \end{cases}$$

8 mults of $(n/2) \times (n/2)$ submatrices

4 adds of $(n/2) \times (n/2)$ submatrices

Analysis of D&C algorithm

$$T(n) = 8\, T(n/2) + \Theta(n^2)$$

 # subproblems

subproblem size

work dividing and combining

$a=8,\ b=2,\ n^{\log_b a} = n^3 \Rightarrow$ CASE 1 $\Rightarrow T(n) = \Theta(n^3)$.

No better than the ordinary algorithm.

Strassen's idea

- Multiply 2×2 matrices with only 7 recursive

$P1 = a \cdot (f - h)$

$P2 = (a + b) \cdot h$

$P3 = (c + d) \cdot e$

$P4 = d \cdot (g - e)$

$P5 = (a + d) \cdot (e + h)$

$P6 = (b - d) \cdot (g + h)$

$P7 = (a - c) \cdot (e + f)$

$r = P5 + P4 - P2 + P6$

$s = P1 + P2$

$t = P3 + P4$

$u = P5 + P1 - P3 - P7$

7 mults, 18 adds/subs.

Note: No reliance on commutativity of mult!

Strassen's idea

$$\begin{cases} r = ae + bg \\ s = af + bh \\ t = ce + dg \\ u = cf + dh \end{cases}$$

- Multiply 2×2 matrices with only 7 recursive

$P1 = a \cdot (f - h)$

$P2 = (a + b) \cdot h$

$P3 = (c + d) \cdot e$

$P4 = d \cdot (g - e)$

$P5 = (a + d) \cdot (e + h)$

$P6 = (b - d) \cdot (g + h)$

$P7 = (a - c) \cdot (e + f)$

$r = P5 + P4 - P2 + P6$

= (a+d)(e+h)

+ d(g-e)-(a+b)h

+(b-d)(g+h)

= ae+ah+de+dh

+dg-de-ah-bh

+bg+bh-dg-dh

=ae + bg

Strassen's algorithm

1. Divide: Partition A and B into $(n/2)\times(n/2)$ submatrices. Form terms to be multiplied using $+$ and $-$.

2. Conquer: Perform 7 multiplications of $(n/2)\times(n/2)$ submatrices recursively.

3. Combine: Form C using $+$ and $-$ on $(n/2)\times(n/2)$ submatrices.

$$T(n) = 7\ T(n/2) + \Theta(n^2)$$

Analysis of Strassen

$$T(n) = 7\,T(n/2) + \Theta(n2)$$

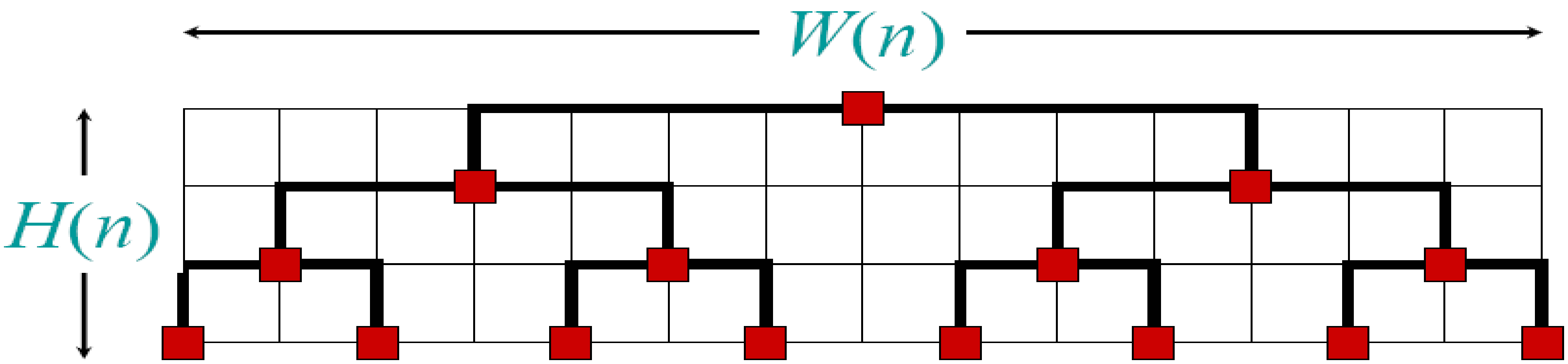
$a=7$, $b=2$, $n^{\log_b a} = n^{\lg 7} \approx n^{2.81} \Rightarrow$ CASE 1 $\Rightarrow T(n) = \Theta(n^{\lg 7})$.

The number 2.81 may not seem much smaller than 3, but because the difference is in the exponent, the impact on running time is significant. In fact, Strassen's algorithm beats the ordinary algorithm on today's machines for $n \geq 30$ or so.

Best to date (of theoretical interest only): $\Theta(n^{2.376\ldots})$.

VLSI layout

Problem: Embed a complete binary tree with n leaves in a grid using minimal area.

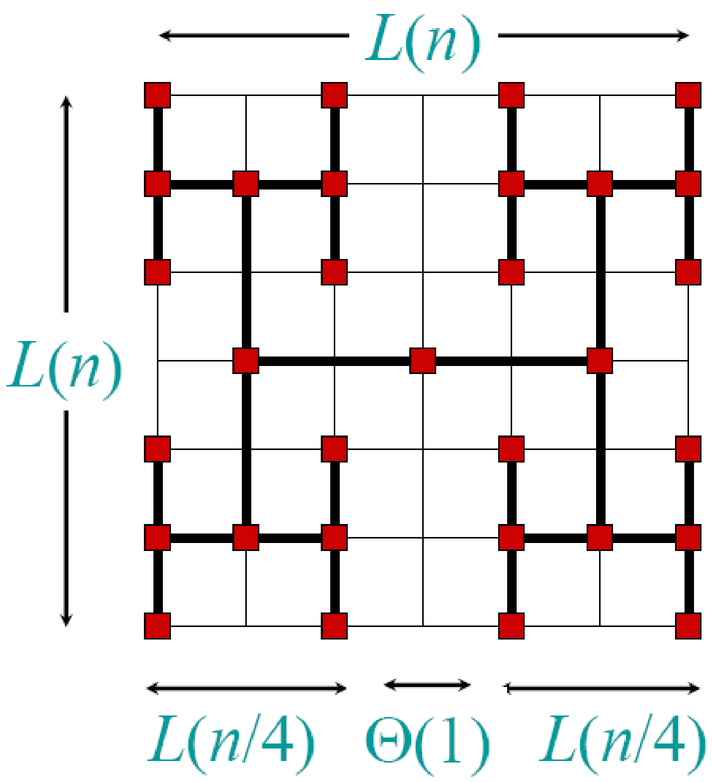


$$H(n) = H(n/2) + \Theta(1) = \Theta(\lg n)$$

$$W(n) = 2W(n/2) + \Theta(1) = \Theta(n)$$

Area $= \Theta(n \lg n)$

H-tree embedding



$$L(n) = 2L(n/4) + \Theta(1)$$
$$= \Theta(\sqrt{n}\,)$$

Area $= \Theta(n)$

Conclusion

- Divide and conquer is just one of several powerful techniques for algorithm design.
- Divide-and-conquer algorithms can be analyzed using recurrences and the master method (so practice this math).
- Can lead to more efficient algorithms