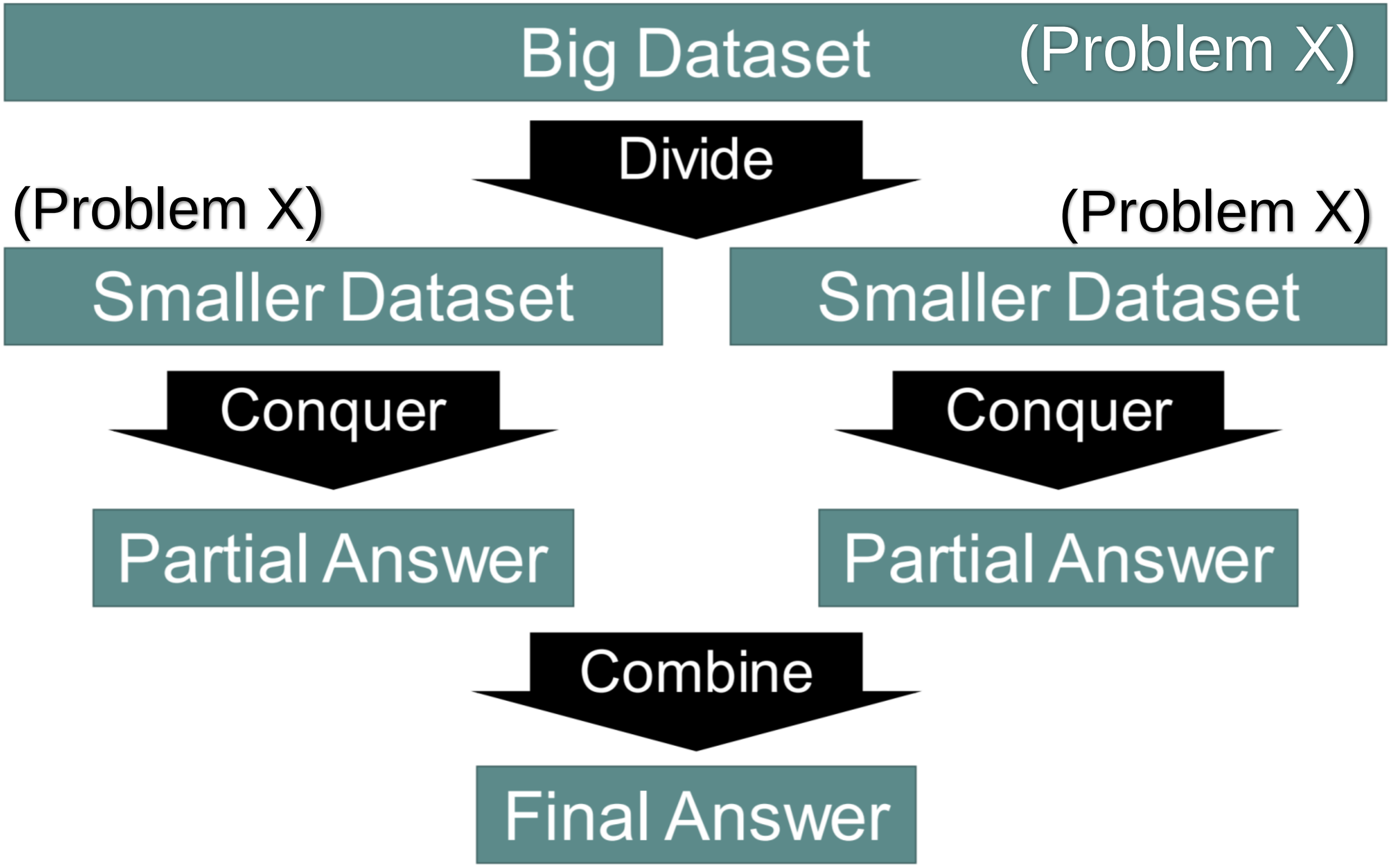
CS141: Intermediate Data Structures and Algorithms

Divide and Conquer: Design and Analysis

Amr Magdy

Divide-and-Conquer (D&C)



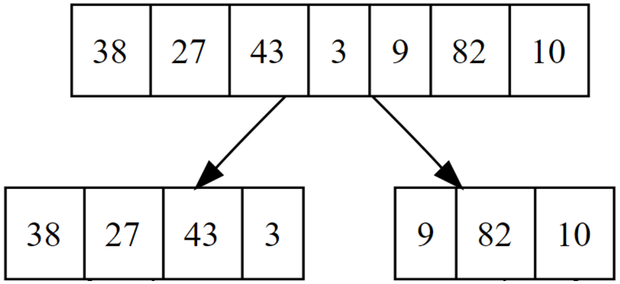
Merge Sort

```
MERGE-SORT(A, p, r)
  if p < r                          // check for base case
      q = ⌊(p + r)/2⌋               // divide
      MERGE-SORT(A, p, q)           // conquer
      MERGE-SORT(A, q + 1, r)       // conquer
      MERGE(A, p, q, r)             // combine
```

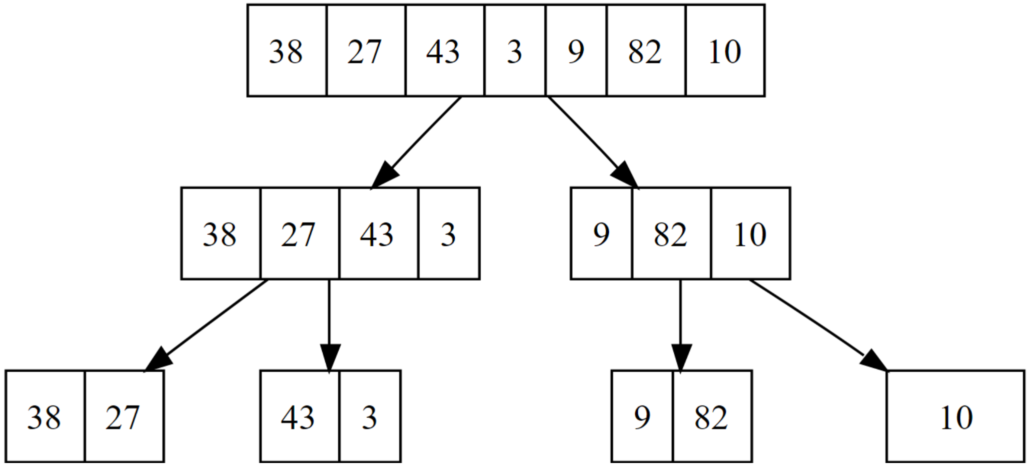
Merge Sort

38	27	43	3	9	82	10

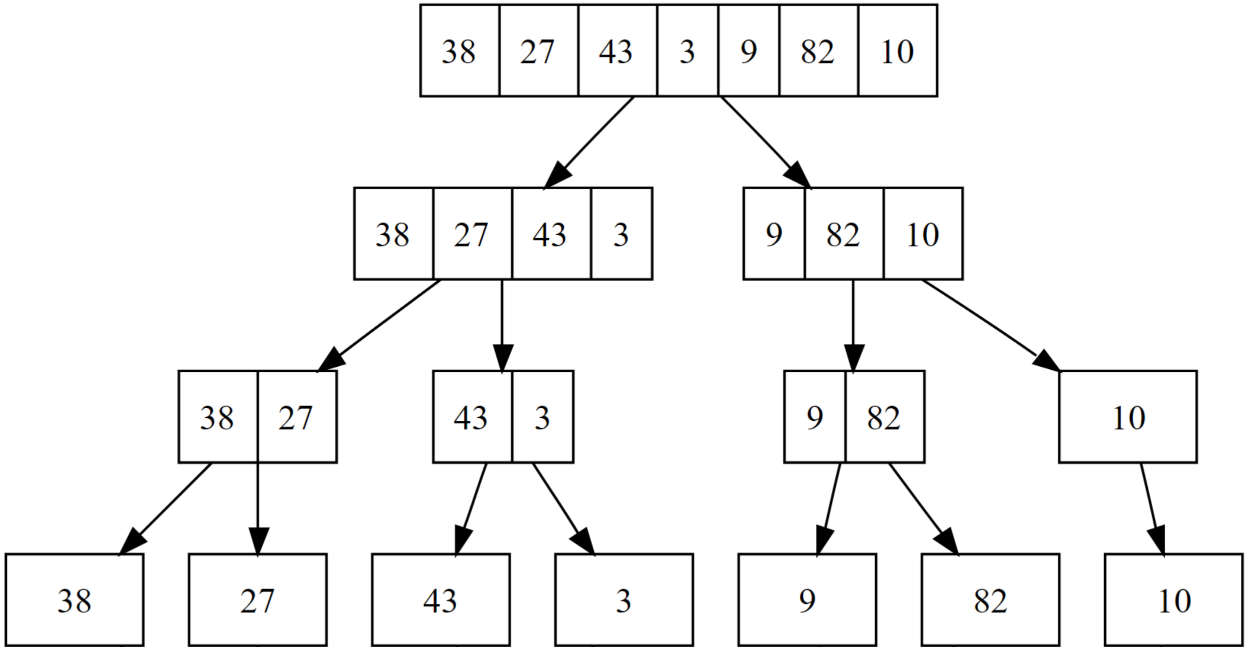
Merge Sort



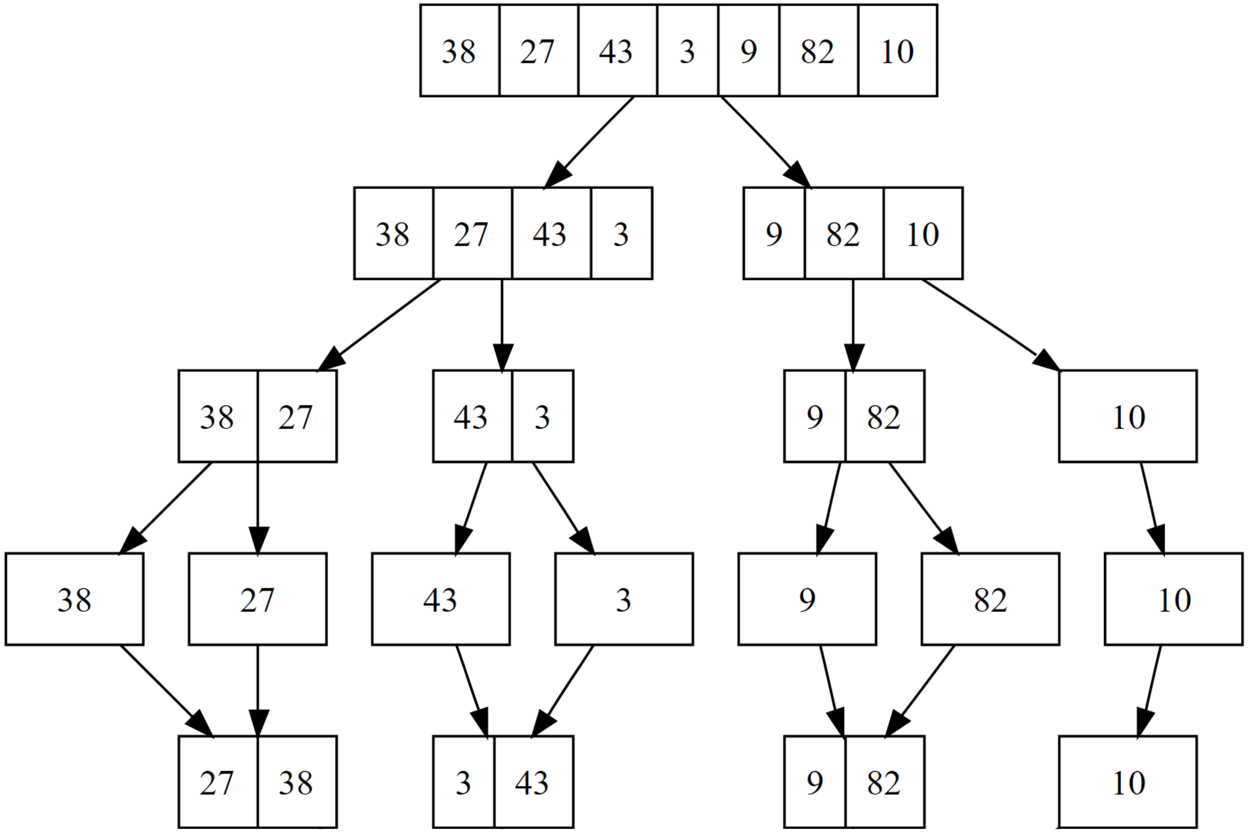
Merge Sort



Merge Sort

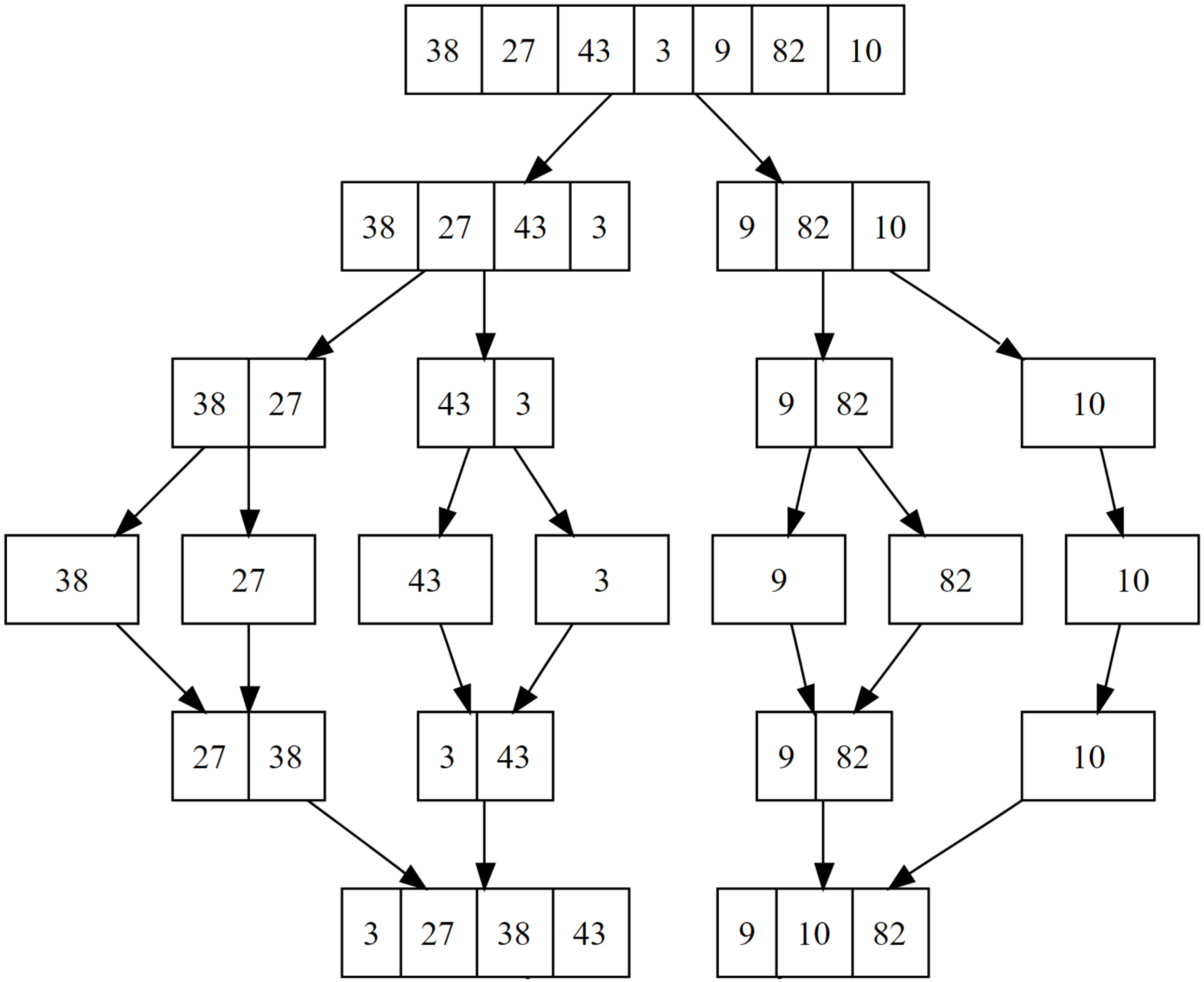


Merge Sort



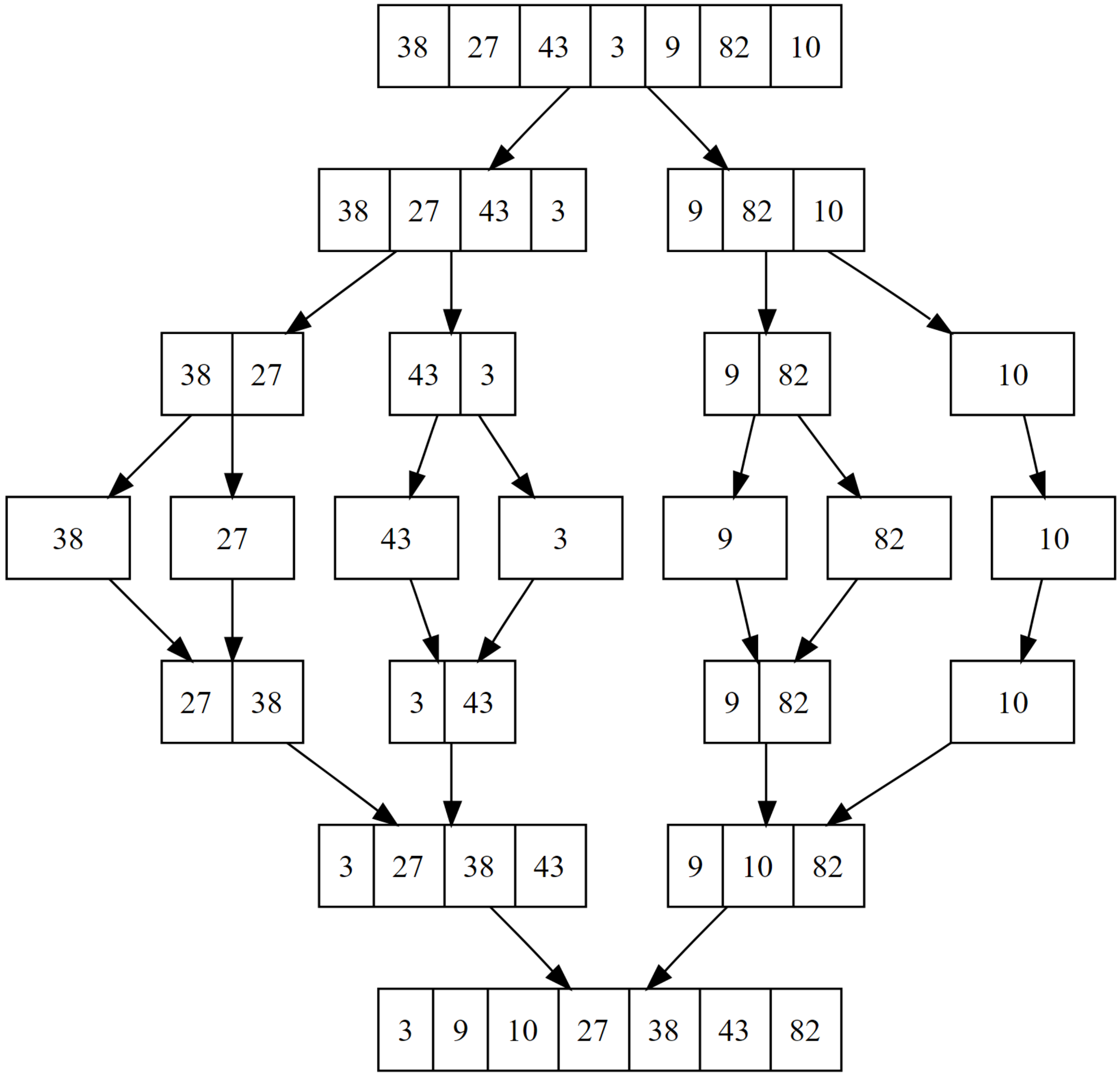
38 27 43 3 9 82 10
38 27 43 3
9 82 10
38 27
43 3
9 82
10
38
27
43
3
9
82
10
27 38
3 43
9 82
10

Merge Sort



38 27 43 3 9 82 10
38 27 43 3
9 82 10
38 27
43 3
9 82
10
38
27
43
3
9
82
10
27 38
3 43
9 82
10
3 27 38 43
9 10 82

Merge Sort


38 27 43 3 9 82 10
38 27 43 3
9 82 10
38 27
43 3
9 82
10
38
27
43
3
9
82
10
27 38
3 43
9 82
10
3 27 38 43
9 10 82
3 9 10 27 38 43 82

Matrix Multiplication

$$\underset{A}{\begin{bmatrix} a & b \\ c & d \end{bmatrix}} \times \underset{B}{\begin{bmatrix} e & f \\ g & h \end{bmatrix}} = \underset{C}{\begin{bmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{bmatrix}}$$

A B C

Matrix Multiplication

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{bmatrix}$$

$$\quad A \qquad\qquad\qquad B \qquad\qquad\qquad\qquad C$$

A, B and C are square metrices of size N x N

a, b, c and d are submatrices of A, of size N/2 x N/2

e, f, g and h are submatrices of B, of size N/2 x N/2

Simple Matrix Multiplication

```
SQUARE-MAT-MULT(A, B, n)
  let C be a new n × n matrix
  for i = 1 to n
      for j = 1 to n
          c_ij = 0
          for k = 1 to n
              c_ij = c_ij + a_ik · b_kj
  return C
```

Simple Matrix Multiplication

SQUARE-MAT-MULT(A, B, n)

let C be a new $n \times n$ matrix
for $i = 1$ **to** n
  for $j = 1$ **to** n
    $c_{ij} = 0$
    for $k = 1$ **to** n
      $c_{ij} = c_{ij} + a_{ik} \cdot b_{kj}$
return C

n n n

Simple Matrix Multiplication

$$\text{SQUARE-MAT-MULT}(A, B, n)$$

let C be a new $n \times n$ matrix
n { **for** $i = 1$ **to** n
n { **for** $j = 1$ **to** n
$c_{ij} = 0$
n { **for** $k = 1$ **to** n
$c_{ij} = c_{ij} + a_{ik} \cdot b_{kj}$
return C

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

D&C Matrix Multiplication

- $A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix},$
$$C = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$
- $\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} . \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$
- $C_{11} = A_{11}.B_{11} + A_{12}.B_{21}$
- $C_{12} = A_{11}.B_{12} + A_{12}.B_{22}$
- $C_{21} = A_{21}.B_{11} + A_{22}.B_{21}$
- $C_{22} = A_{21}.B_{12} + A_{22}.B_{22}$

D&C Matrix Multiplication

```
REC-MAT-MULT(A, B, n)
  let C be a new n × n matrix
  if n == 1
      c11 = a11 · b11
  else partition A, B, and C into n/2 × n/2 submatrices
      C11 = REC-MAT-MULT(A11, B11, n/2) + REC-MAT-MULT(A12, B21, n/2)
      C12 = REC-MAT-MULT(A11, B12, n/2) + REC-MAT-MULT(A12, B22, n/2)
      C21 = REC-MAT-MULT(A21, B11, n/2) + REC-MAT-MULT(A22, B21, n/2)
      C22 = REC-MAT-MULT(A21, B12, n/2) + REC-MAT-MULT(A22, B22, n/2)
  return C
```

- Implementation details
  - Partitioning matrices: index calculation not copying
- Is this faster than $\Theta(n^3)$? How to analyze a recursive algorithm?

Analyzing Recursive Algorithms

1. Determine the recursive relation
2. Analyze the complexity of the recursive relation
    - Two methods:
        - Recursive tree expansion (or substitution method)
        - Master theorem

Analyzing Recursive Algorithms: Merge Sort

MERGE-SORT(A, p, r)

if $p < r$
  $q = \lfloor (p + r)/2 \rfloor$
  MERGE-SORT(A, p, q)
  MERGE-SORT$(A, q + 1, r)$
  MERGE(A, p, q, r)

Analyzing Recursive Algorithms: Merge Sort

MERGE-SORT(A, p, r) .... T(n)

  if $p < r$ .... Θ(1)

    $q = \lfloor (p + r)/2 \rfloor$ .... Θ(1)

    MERGE-SORT(A, p, q)............................ T(n/2)

    MERGE-SORT$(A, q + 1, r)$ .... T(n/2)

    MERGE(A, p, q, r) .... Θ(n)

Analyzing Recursive Algorithms: Merge Sort

MERGE-SORT(A, p, r) .... T(n)

if $p < r$ .... Θ(1)

$q = \lfloor (p + r)/2 \rfloor$ .... Θ(1)

MERGE-SORT(A, p, q) .... T(n/2)

MERGE-SORT$(A, q + 1, r)$ .... T(n/2)

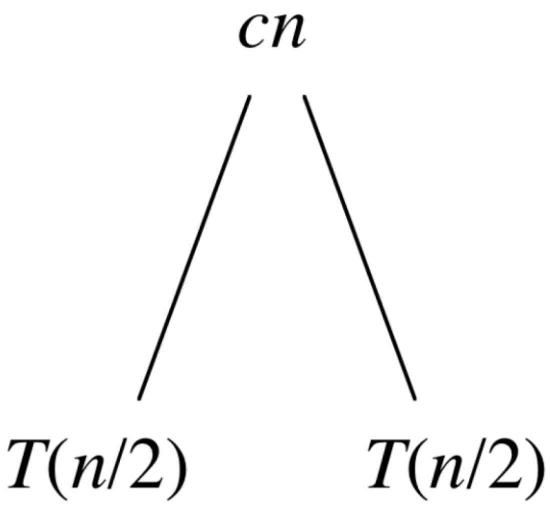
MERGE(A, p, q, r) .... Θ(n)

T(n) = Θ(1) + Θ(1) + T(n/2) + T(n/2) + Θ(n)

T(n) = 2 T(n/2) + Θ(n)

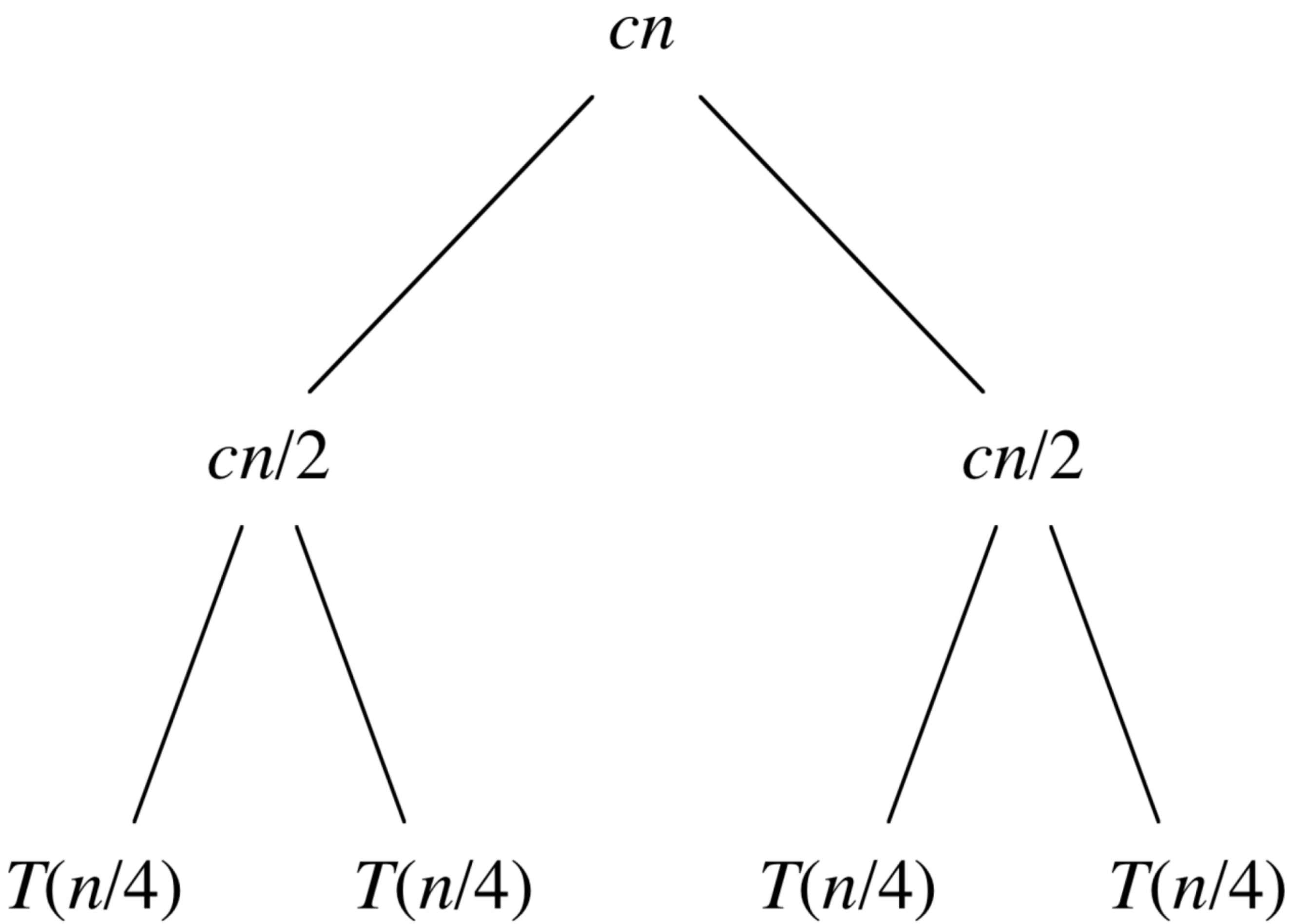
Analyzing Recursive Algorithms: Merge Sort

- Recursion tree expansion for T(n) = 2 T(n/2) + Θ(n)



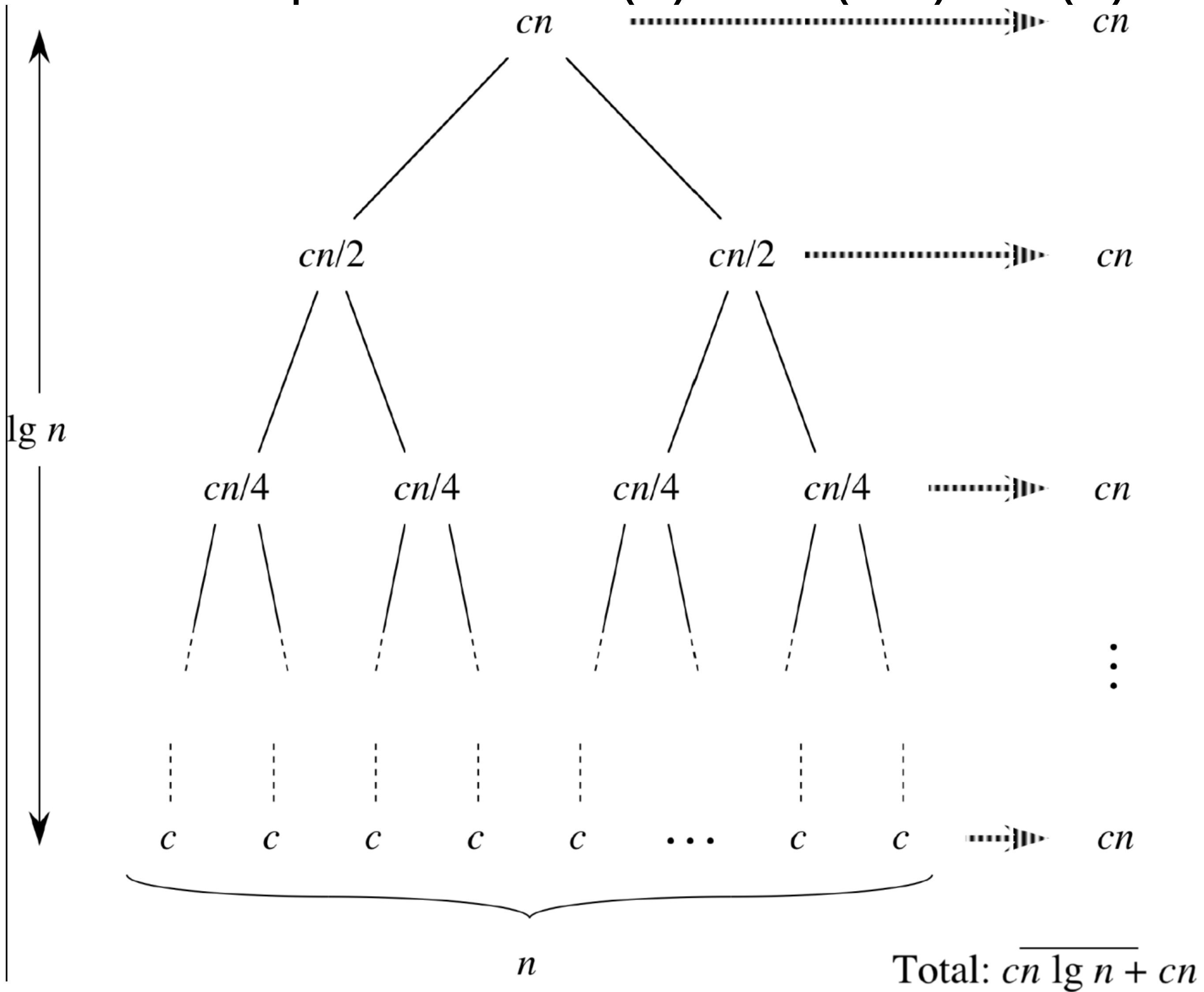
Analyzing Recursive Algorithms: Merge Sort

- Recursion tree expansion for T(n) = 2 T(n/2) + Θ(n)



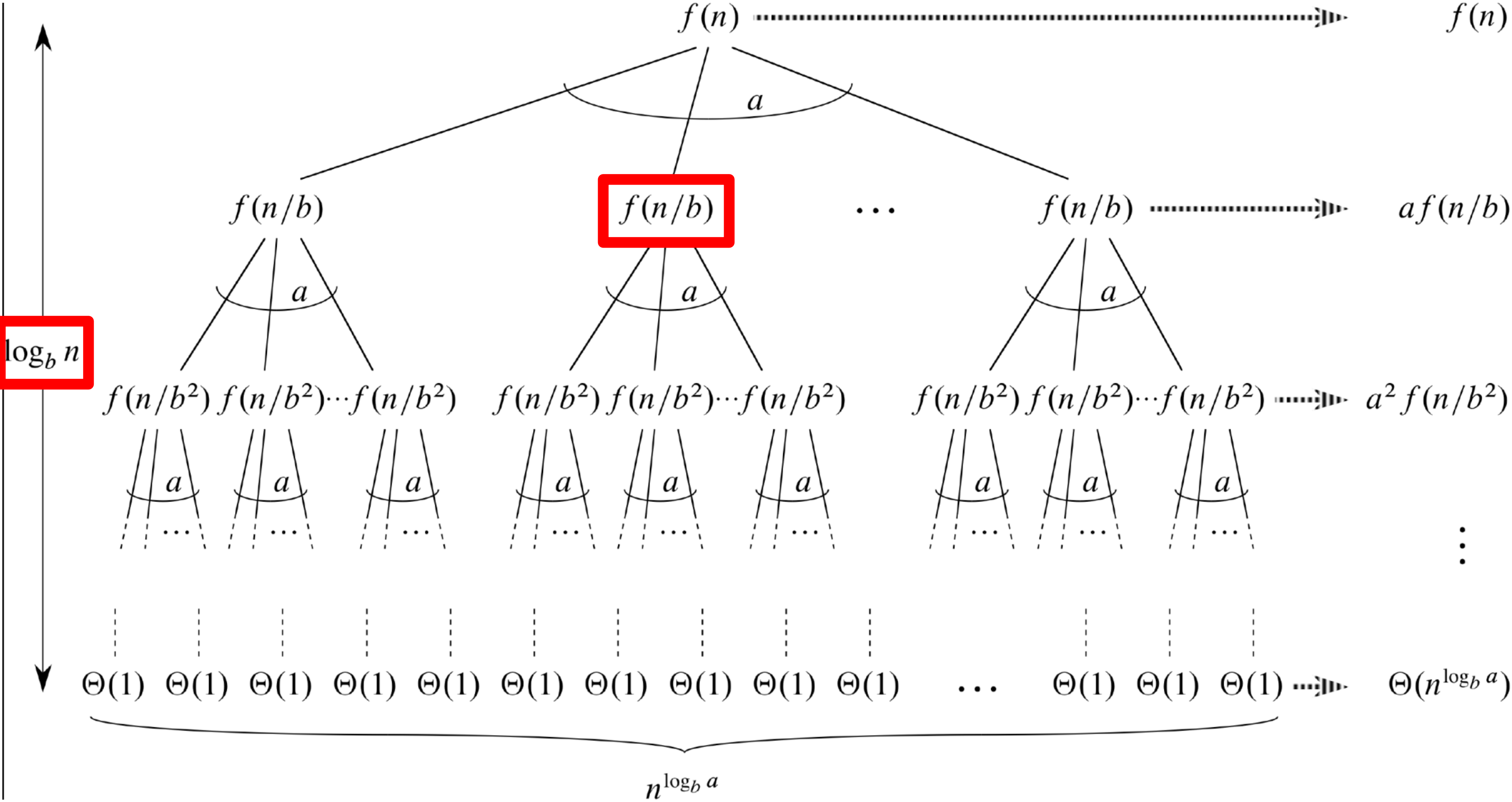
Analyzing Recursive Algorithms: Merge Sort

- Recursion tree expansion for T(n) = 2 T(n/2) + Θ(n)



Analyzing Recursive Algorithms

- General recursion tree



Analyzing Recursive Algorithms

- Recursive Fibonacci algorithm?

Analyzing Recursive Algorithms: Master Theorem

- Master Theorem:
  Used to solve recurrences in the form
  T(n) = aT(n/b) + f(n),   a >= 1, b > 1,
  f(n) is a function, T(n) defined on nonnegative integers
- T(n) bound depends on **polynomial comparison** between f(n) and $n^{\log_b a}$

if f(n) polynomial less $n^{\log_b a}$ → T(n) = Θ($n^{\log_b a}$)

if f(n) polynomial equal $n^{\log_b a}$ → T(n) = Θ($n^{\log_b a} \log n$)

if f(n) polynomial greater $n^{\log_b a}$ → T(n) = Θ($f(n)$)

Analyzing Recursive Algorithms: Master Theorem

- Master Theorem:
  Used to solve recurrences in the form
  T(n) = aT(n/b) + f(n), a >= 1, b > 1,
  f(n) is a function, T(n) defined on nonnegative integers
- T(n) bound depends on **polynomial comparison** between f(n) and $n^{\log_b a}$
- Formally:

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$. ■

Analyzing Recursive Algorithms: Merge Sort

- T(n) = 2 T(n/2) + Θ(n)
- In the general form of Master theorem, a=2, b=2, f(n)=cn
- $n^{\log_b a} = n^{\log_2 2} = n$
  then f(n) = Θ($n^{\log_b a}$), case 2
- T(n) = Θ($n^{\log_b a} \log n$) = Θ($n \log n$)

Analyzing Recursive Algorithms

- Recursive Fibonacci algorithm?

Analyzing Recursive Algorithms: Matrix Multiplication

REC-MAT-MULT(A, B, n) .... T(n)

let C be a new $n \times n$ matrix

if $n == 1$ .... Θ(1)

$c_{11} = a_{11} \cdot b_{11}$ .... Θ(1)

else partition A, B, and C into $n/2 \times n/2$ submatrices

$C_{11} =$ REC-MAT-MULT$(A_{11}, B_{11}, n/2) +$ REC-MAT-MULT$(A_{12}, B_{21}, n/2)$

$C_{12} =$ REC-MAT-MULT$(A_{11}, B_{12}, n/2) +$ REC-MAT-MULT$(A_{12}, B_{22}, n/2)$

$C_{21} =$ REC-MAT-MULT$(A_{21}, B_{11}, n/2) +$ REC-MAT-MULT$(A_{22}, B_{21}, n/2)$

$C_{22} =$ REC-MAT-MULT$(A_{21}, B_{12}, n/2) +$ REC-MAT-MULT$(A_{22}, B_{22}, n/2)$

return C

→ $(2T(n/2) + \frac{n}{2} * \frac{n}{2})*4$

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

Analyzing Recursive Algorithms: Matrix Multiplication

- $T(n) = 8\,T(n/2) + \Theta(n^2)$
- In the general form of Master theorem, a=8, b=2, $f(n)= \Theta(n^2)$
- $n^{\log_b a} = n^{\log_2 8} = n^3$
  then $f(n) = O(n^{\log_b a-\varepsilon}) = O(n^{3-1})$, $\epsilon = 1$, case 1
- $T(n) = \Theta(n^{\log_b a}) = \Theta(n^3)$
- D&C matrix multiplication is as fast as the simple matrix multiplication

Strassen's Matrix Multiplication

p1 = a(f - h)

p2 = (a + b)h

p3 = (c + d)e

p4 = d(g - e)

p5 = (a + d)(e + h)

p6 = (b - d)(g + h)

p7 = (a - c)(e + f)

The A x B can be calculated using above seven multiplications.
Following are values of four sub-matrices of result C

$$\underset{A}{\begin{bmatrix} a & b \\ c & d \end{bmatrix}} \times \underset{B}{\begin{bmatrix} e & f \\ g & h \end{bmatrix}} = \underset{C}{\begin{bmatrix} p5 + p4 - p2 + p6 & p1 + p2 \\ p3 + p4 & p1 + p5 - p3 - p7 \end{bmatrix}}$$

A, B and C are square metrices of size N x N
a, b, c and d are submatrices of A, of size N/2 x N/2
e, f, g and h are submatrices of B, of size N/2 x N/2
p1, p2, p3, p4, p5, p6 and p7 are submatrices of size N/2 x N/2

Strassen's Matrix Multiplication

p1 = a(f - h)
p2 = (a + b)h
p3 = (c + d)e
p4 = d(g - e)
p5 = (a + d)(e + h)
p6 = (b - d)(g + h)
p7 = (a - c)(e + f)

The A x B can be calculated using above seven multiplications.
Following are values of four sub-matrices of result C

$$\underbrace{\begin{bmatrix} a & b \\ c & d \end{bmatrix}}_{A} \times \underbrace{\begin{bmatrix} e & f \\ g & h \end{bmatrix}}_{B} = \underbrace{\begin{bmatrix} p5 + p4 - p2 + p6 & p1 + p2 \\ p3 + p4 & p1 + p5 - p3 - p7 \end{bmatrix}}_{C}$$

A, B and C are square metrices of size N x N
a, b, c and d are submatrices of A, of size N/2 x N/2
e, f, g and h are submatrices of B, of size N/2 x N/2
p1, p2, p3, p4, p5, p6 and p7 are submatrices of size N/2 x N/2

- Each P has T(n/2) and $\Theta(n^2)$ (adding two $\frac{n}{2} x \frac{n}{2}$ matrices)
- $T(n) = 7T(n/2) + \Theta(n^2)$

Analyzing Recursive Algorithms: Strassen's Matrix Multiplication

- T(n) = 7 T(n/2) + Θ(n^2)
- In the general form of Master theorem, a=7, b=2, f(n)= Θ(n^2)
- $n^{\log_b a} = n^{\log_2 7} = n^{2.807}$
  then f(n) = O($n^{\log_b a-\varepsilon}$) = O($n^{2.807-0.8}$), ϵ = 0.8, case 1

- T(n) = Θ($n^{\log_b a}$) = Θ($n^{2.807}$)

Analyzing Recursive Algorithms: Strassen's Matrix Multiplication

- $T(n) = 7\,T(n/2) + \Theta(n^2)$
- In the general form of Master theorem, a=7, b=2, $f(n)= \Theta(n^2)$
- $n^{\log_b a} = n^{\log_2 7} = n^{2.807}$
  then $f(n) = O(n^{\log_b a-\varepsilon}) = O(n^{2.807-0.8})$, $\epsilon = 0.8$, case 1

- $T(n) = \Theta(n^{\log_b a}) = \Theta(n^{2.807})$

n	$n^{2.807}$	n^3
10	641.2096	1000
100	411149.7	1000000
1000	2.64E+08	1E+09
10000	1.69E+11	1E+12
100000	1.08E+14	1E+15
1000000	6.95E+16	1E+18
10000000	4.46E+19	1E+21

When Master Theorem Fails?

- When $n^{\log_b a}$ and f(n) are not polynomially comparable
- Example 1: T(n) = 3 T(n/4) + n log n

a=3, b=4, f(n) = n log n

$n^{\log_b a} = n^{\log_4 3} = n^{0.79}$

$$f(n)/n^{\log_b a} = n^{0.21} \log n$$

f(n) is polynomially larger than $n^{\log_b a}$ and case 3 applies

What values of constant c satisfies the condition
a f(n/b) <= cf(n)?

When Master Theorem Fails?

- When $n^{\log_b a}$ and f(n) are not polynomially comparable
- Example 2: T(n) = 2 T(n/2) + n log n

a=2, b=2, f(n) = n log n

$n^{\log_b a} = n^{\log_2 2} = n$

$$f(n)/n^{\log_b a} = \log n$$

f(n) is not polynomially larger than $n^{\log_b a}$ (i.e., there is not polynomial factor n^x in the ratio $f(n)/n^{\log_b a}, no\ \varepsilon > 0$ exists) and Master theorem does not apply

Credits & Book Readings

- Book Readings
  - 2.3, Ch. 4 Intro, 4.2, 4.3, 4.4, 4.5
- Credits
  - Prof. Ahmed Eldawy notes
  - Online Sources
    - https://upload.wikimedia.org/wikipedia/commons/e/e6/Merge_sort_algorithm_diagram.svg
    - http://www.geeksforgeeks.org/wp-content/uploads/stressen_formula_new_new.png