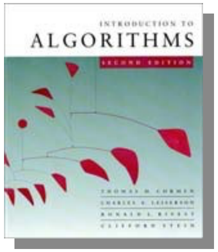
CS60020: Foundations of Algorithm Design and Machine Learning

Sourangshu Bhattacharya

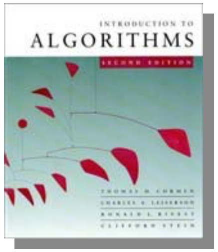
SOME D&C ALGORITHMS



Binary search

- Find an element in a sorted array:

1. ***Divide:*** Check middle element.
2. ***Conquer:*** Recursively search 1 subarray.
3. ***Combine:*** Trivial.



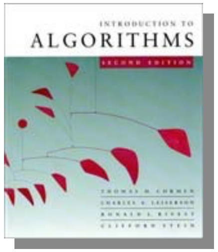
Recurrence for binary search

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

 # subproblems

subproblem size

work dividing and combining



Recurrence for binary search

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

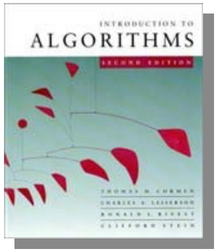
 # subproblems

subproblem size

work dividing and combining

$$n^{\log_b a} = n^{\log_2 1} = n^0 = 1 \Rightarrow \quad \text{CASE } 2 \ (k = 0)$$

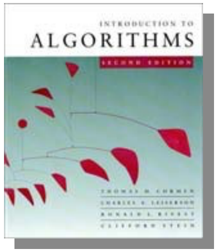
$$\Rightarrow \quad T(n) = \Theta(\lg n)\,.$$



Powering a number

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

Naive algorithm: $\Theta(n)$.



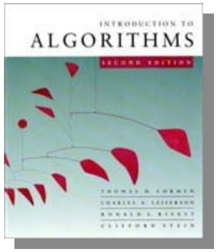
Powering a number

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

Naive algorithm: $\Theta(n)$.

Divide-and-conquer algorithm:

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



Powering a number

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

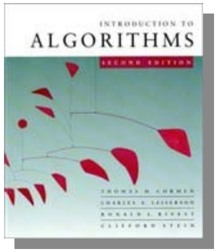
Naive algorithm: $\Theta(n)$.

Divide-and-conquer algorithm:

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

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

STRASSENS MATRIX MULTIPLICATION



Matrix multiplication

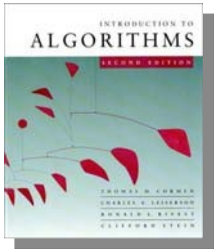
Input: $A = [a_{ij}],\ B = [b_{ij}].$

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

$$\left.\begin{array}{l} \text{Input: } A = [a_{ij}],\ B = [b_{ij}]. \\ \text{Output: } C = [c_{ij}] = A \cdot B. \end{array}\right\} \quad i, j = 1, 2, \ldots, n.$$

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

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



Standard algorithm

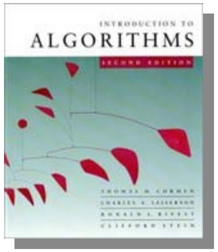
for $i \leftarrow 1$ **to** n

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

        do $c_{ij} \leftarrow 0$

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

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



Standard algorithm

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

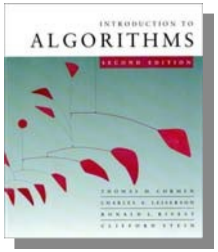
  do for $j \leftarrow 1$ **to** n

    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:

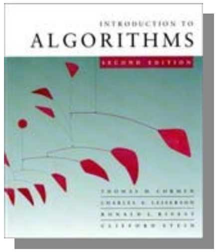
$$\left[\begin{array}{c|c} r & s \\ \hline t & u \end{array}\right] = \left[\begin{array}{c|c} a & b \\ \hline c & d \end{array}\right] \cdot \left[\begin{array}{c|c} e & f \\ \hline g & h \end{array}\right]$$

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

$$\left.\begin{aligned} r &= ae + bg \\ s &= af + bh \\ t &= ce + dg \\ u &= cf + dh \end{aligned}\right\}$$

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

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



Divide-and-conquer algorithm

IDEA:

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

$$\begin{bmatrix} r & s \\ t & u \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot \begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

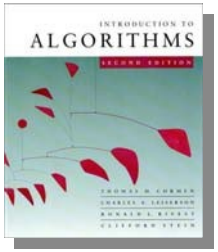
$$C \quad = \quad A \quad \cdot \quad B$$

$$\left.\begin{aligned} r &= ae + bg \\ s &= af + bh \\ t &= ce + dh \\ u &= cf + dg \end{aligned}\right\}$$

recursive

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

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



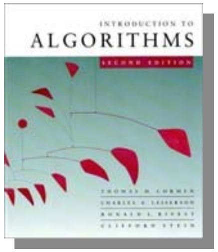
Analysis of D&C algorithm

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

 # submatrices

submatrix size

work adding submatrices



Analysis of D&C algorithm

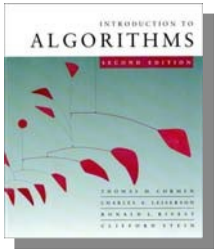
$$T(n) = 8\,T(n/2) + \Theta(n^2)$$

 # submatrices

submatrix size

work adding submatrices

$$n^{\log_b a} = n^{\log_2 8} = n^3 \;\Rightarrow\; \textsc{Case } 1 \;\Rightarrow\; T(n) = \Theta(n^3).$$



Analysis of D&C algorithm

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

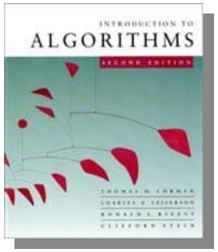
 # submatrices

submatrix size

work adding submatrices

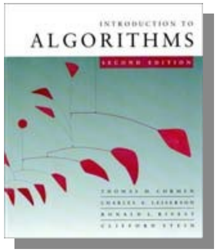
$$n^{\log_b a} = n^{\log_2 8} = n^3 \quad \Rightarrow \quad \text{CASE 1} \quad \Rightarrow \quad T(n) = \Theta(n^3).$$

No better than the ordinary algorithm.



Strassen's idea

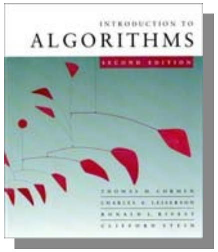
- Multiply $2{\times}2$ matrices with only 7 recursive mults.



Strassen’s idea

- Multiply 2×2 matrices with only 7 recursive mults.

$$P_1 = a \cdot (f - h)$$
$$P_2 = (a + b) \cdot h$$
$$P_3 = (c + d) \cdot e$$
$$P_4 = d \cdot (g - e)$$
$$P_5 = (a + d) \cdot (e + h)$$
$$P_6 = (b - d) \cdot (g + h)$$
$$P_7 = (a - c) \cdot (e + f)$$

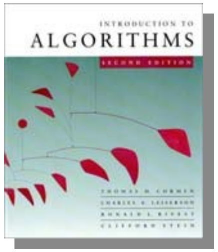


Strassen's idea

- Multiply 2×2 matrices with only 7 recursive mults.

$$
\begin{aligned}
P_1 &= a \cdot (f - h) \\
P_2 &= (a + b) \cdot h \\
P_3 &= (c + d) \cdot e \\
P_4 &= d \cdot (g - e) \\
P_5 &= (a + d) \cdot (e + h) \\
P_6 &= (b - d) \cdot (g + h) \\
P_7 &= (a - c) \cdot (e + f)
\end{aligned}
$$

$$
\begin{aligned}
r &= P_5 + P_4 - P_2 + P_6 \\
s &= P_1 + P_2 \\
t &= P_3 + P_4 \\
u &= P_5 + P_1 - P_3 - P_7
\end{aligned}
$$



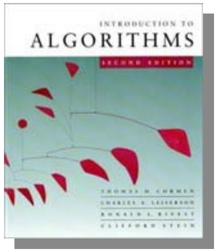
Strassen's idea

- Multiply 2×2 matrices with only 7 recursive mults.

$$
\begin{aligned}
P_1 &= a \cdot (f - h) \\
P_2 &= (a + b) \cdot h \\
P_3 &= (c + d) \cdot e \\
P_4 &= d \cdot (g - e) \\
P_5 &= (a + d) \cdot (e + h) \\
P_6 &= (b - d) \cdot (g + h) \\
P_7 &= (a - c) \cdot (e + f)
\end{aligned}
$$

$$
\begin{aligned}
r &= P_5 + P_4 - P_2 + P_6 \\
s &= P_1 + P_2 \\
t &= P_3 + P_4 \\
u &= P_5 + P_1 - P_3 - P_7
\end{aligned}
$$

7 mults, 18 adds/subs.
Note: No reliance on commutativity of mult!

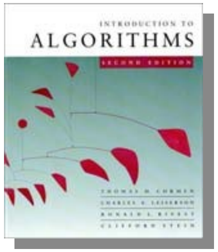


Strassen's idea

- Multiply 2×2 matrices with only 7 recursive mults.

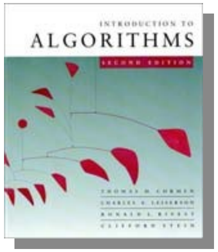
$$
\begin{aligned}
P_1 &= a \cdot (f - h) \\
P_2 &= (a + b) \cdot h \\
P_3 &= (c + d) \cdot e \\
P_4 &= d \cdot (g - e) \\
P_5 &= (a + d) \cdot (e + h) \\
P_6 &= (b - d) \cdot (g + h) \\
P_7 &= (a - c) \cdot (e + f)
\end{aligned}
$$

$$
\begin{aligned}
r &= P_5 + P_4 - P_2 + P_6 \\
&= (a + d)(e + h) \\
&\quad + d(g - e) - (a + b)h \\
&\quad + (b - d)(g + h) \\
&= ae + ah + de + dh \\
&\quad + dg - de - ah - bh \\
&\quad + bg + bh - dg - dh \\
&= ae + bg
\end{aligned}
$$



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.

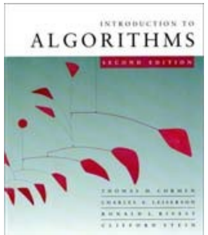


Strassen's algorithm

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.

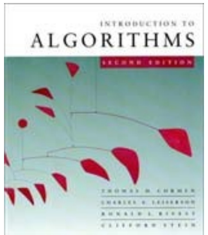
$$T(n) = 7\,T(n/2) + \Theta(n^2)$$

QUICKSORT



Quicksort

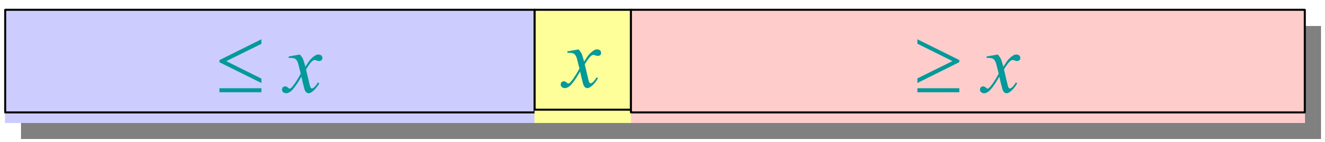
- Proposed by C.A.R. Hoare in 1962.
- Divide-and-conquer algorithm.
- Sorts "in place" (like insertion sort, but not like merge sort).
- Very practical (with tuning).



Divide and conquer

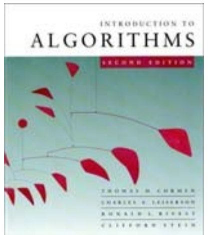
Quicksort an n-element array:

1. ***Divide:*** Partition the array into two subarrays around a ***pivot*** x such that elements in lower subarray $\leq x \leq$ elements in upper subarray.



2. ***Conquer:*** Recursively sort the two subarrays.
3. ***Combine:*** Trivial.

Key: *Linear-time partitioning subroutine.*

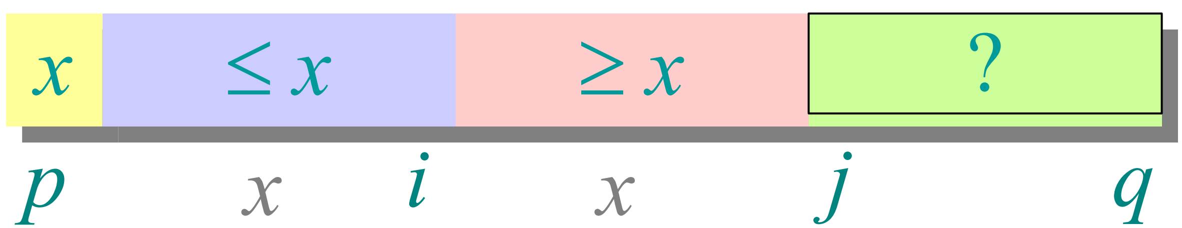


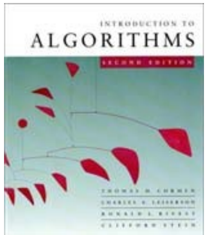
Partitioning subroutine

```
PARTITION(A, p, q)   ◁ A[p . . q]
    x ← A[p]         ◁ pivot = A[p]
    i ← p
    for j ← p + 1 to q
        do if A[j] ≤ x
              then  i ← i + 1
                    exchange A[i] ↔ A[j]
    exchange A[p] ↔ A[i]
    return i
```

Running Time $= O(n)$ for n elements

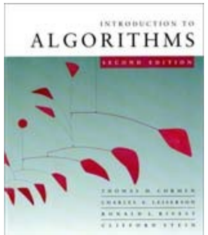
Invariant:



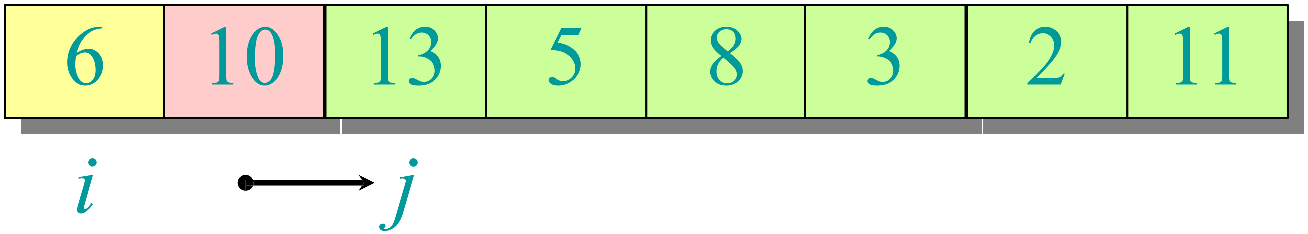


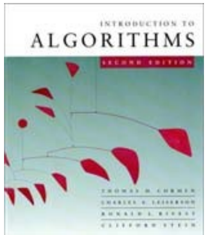
Example of partitioning

6	10	13	5	8	3	2	11
i	*j*						



Example of partitioning

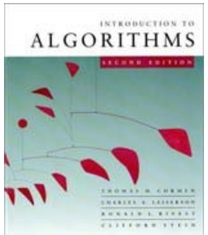




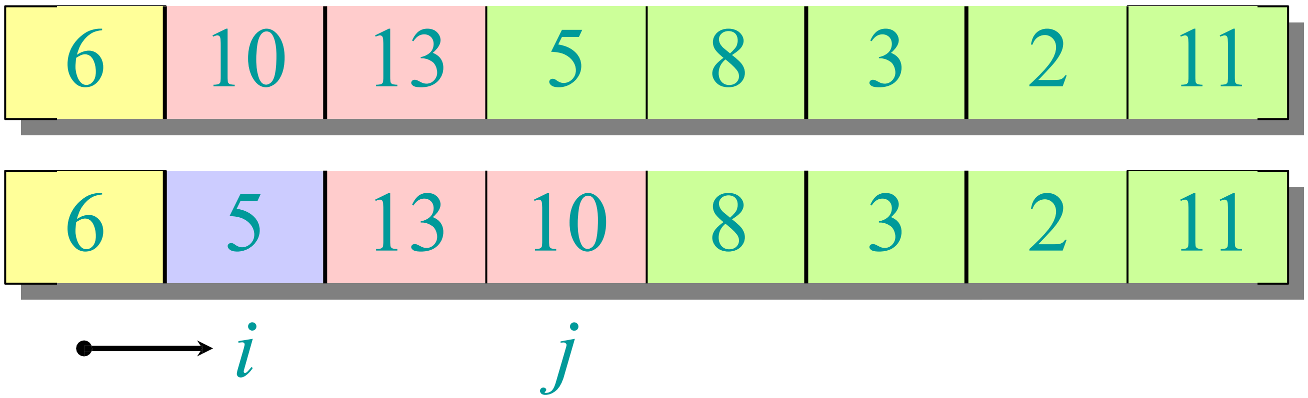
Example of partitioning

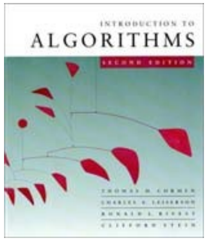
6	10	13	5	8	3	2	11

i $\longrightarrow j$

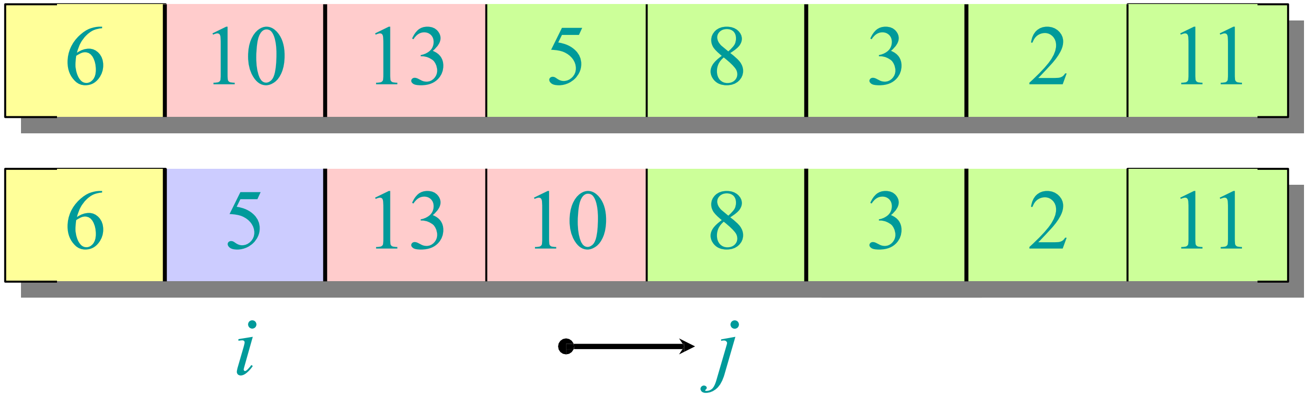


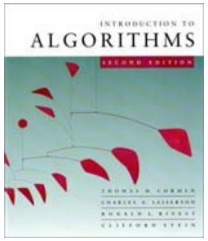
Example of partitioning



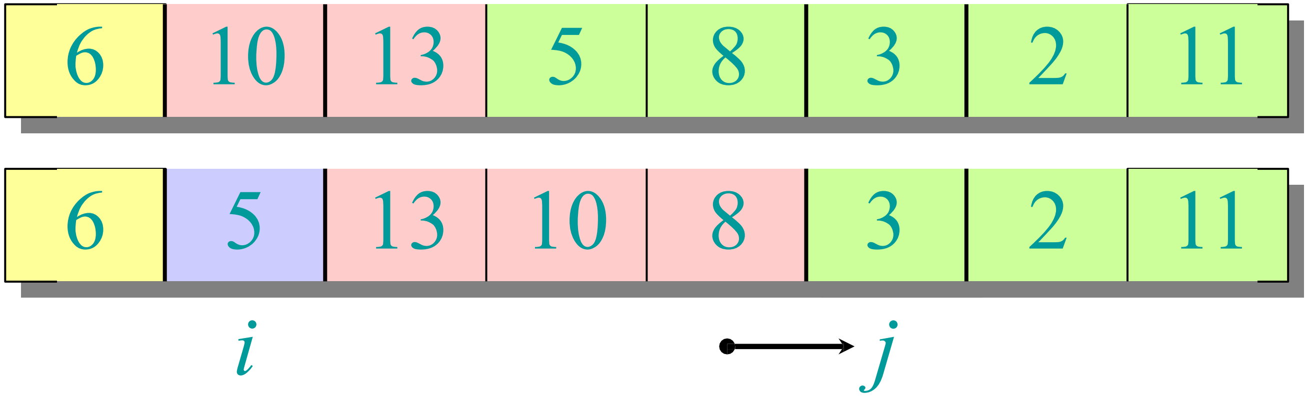


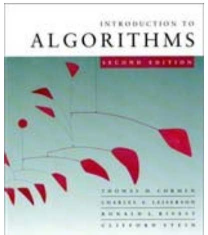
Example of partitioning



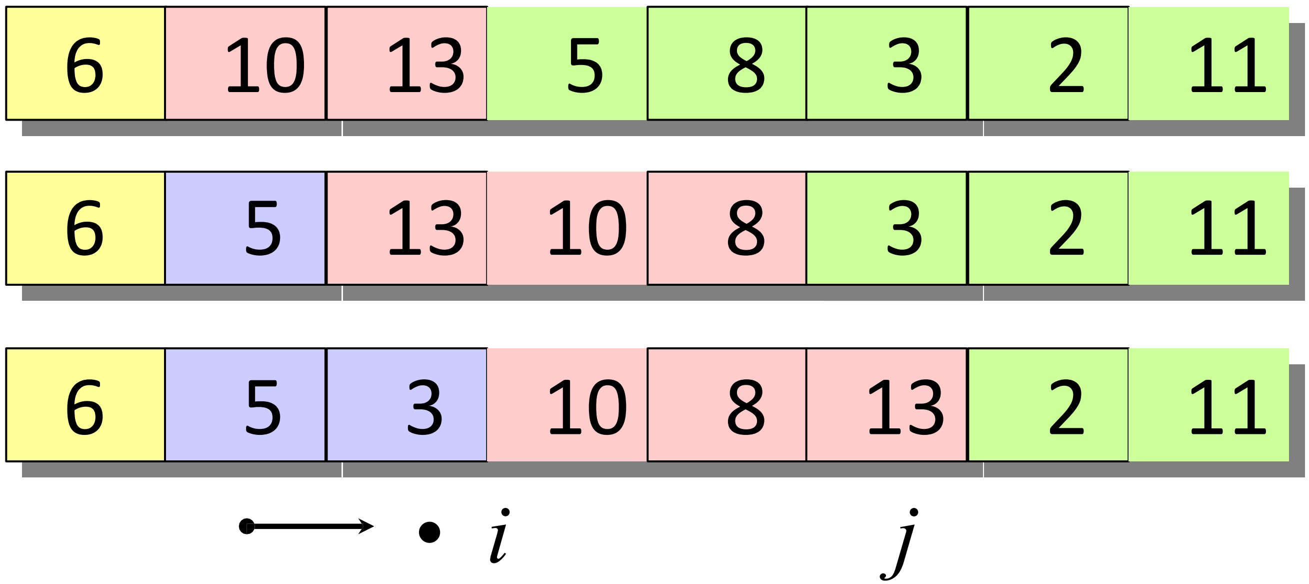


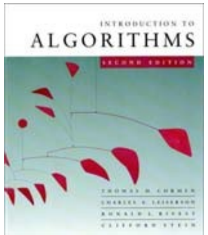
Example of partitioning



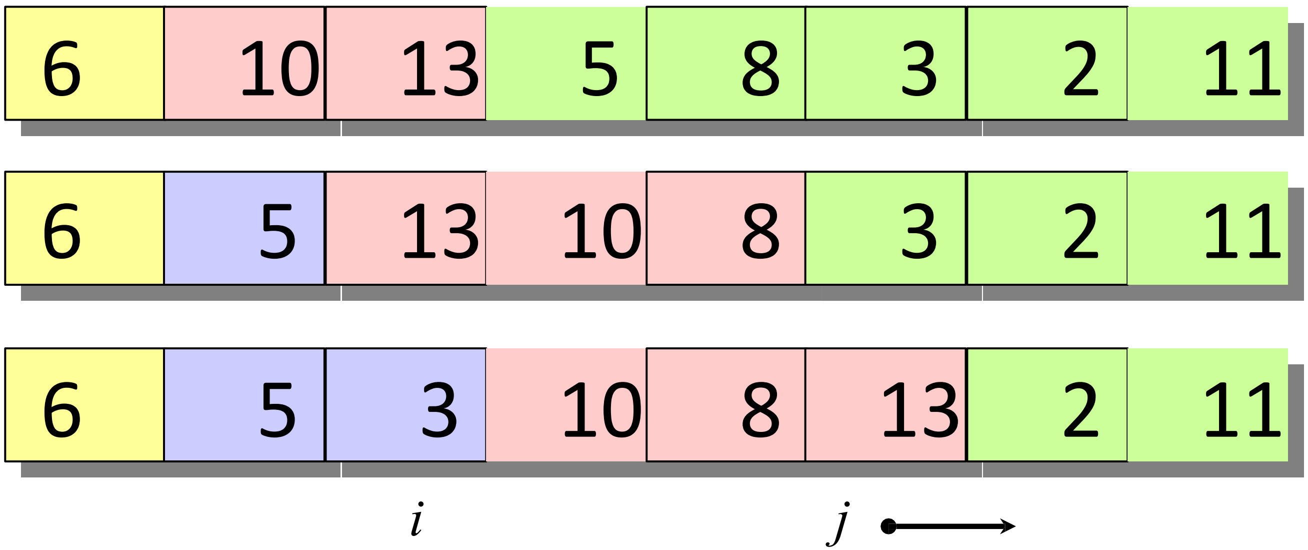


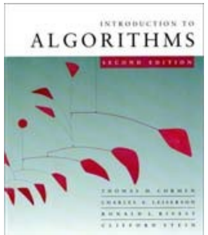
Example of partitioning





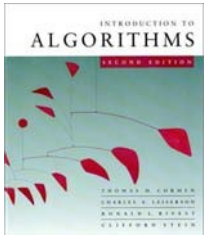
Example of partitioning



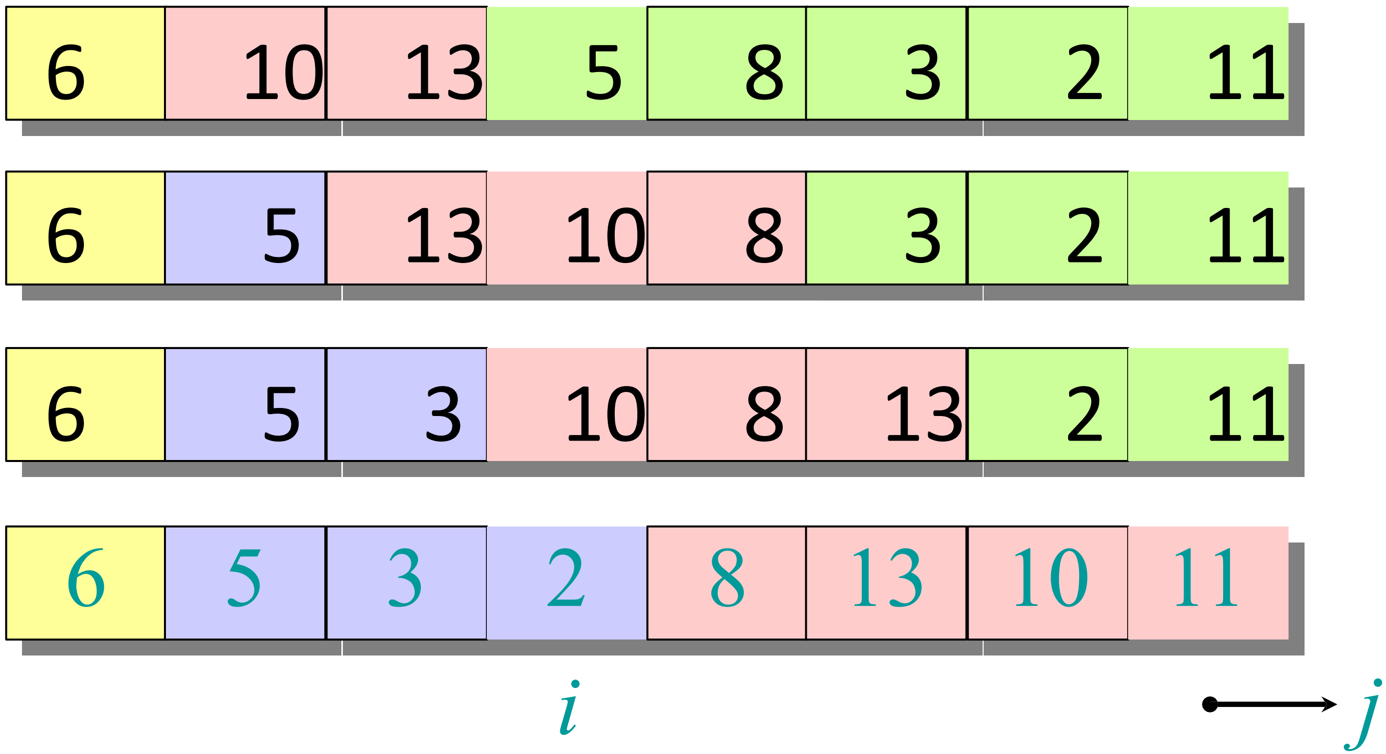


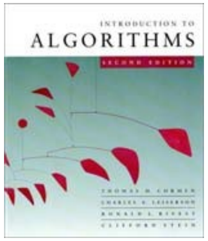
Example of partitioning

6	10	13	5	8	3	2	11
6	5	13	10	8	3	2	11
6	5	3	10	8	13	2	11
6	5	3	2	8	13	10	11
			i			j	



Example of partitioning

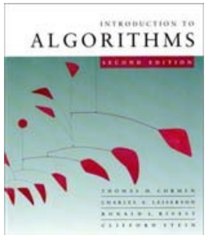




Example of partitioning

6	10	13	5	8	3	2	11
6	5	13	10	8	3	2	11
6	5	3	10	8	13	2	11
6	5	3	2	8	13	10	11
2	5	3	6	8	13	10	11

i

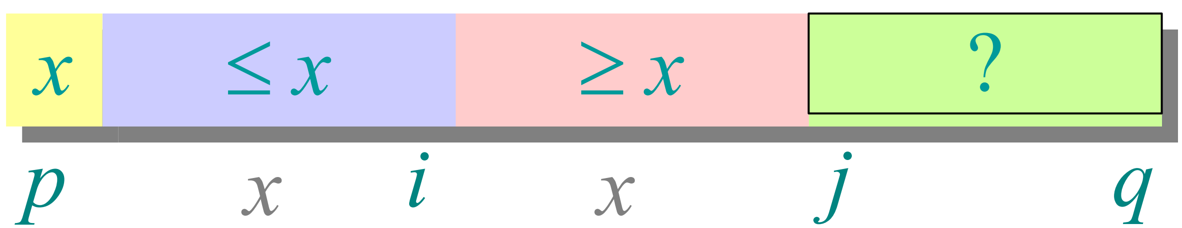


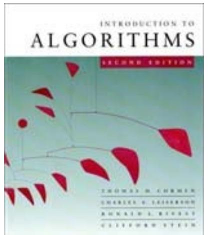
Partitioning subroutine

PARTITION(A, p, q) $\triangleleft A[p \,..\, q]$
    $x \leftarrow A[p]$ $\triangleleft$ pivot $= A[p]$
    $i \leftarrow p$
    for $j \leftarrow p + 1$ **to** q
        do if $A[j] \le x$
            then $i \leftarrow i + 1$
                exchange $A[i] \leftrightarrow A[j]$
    exchange $A[p] \leftrightarrow A[i]$
    return i

Running Time $= O(n)$ for n elements

Invariant:

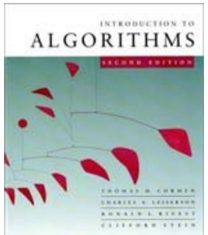




Pseudocode for quicksort

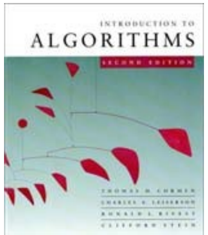
QUICKSORT(A, p, r)
    if $p < r$
        then $q \leftarrow$ PARTITION(A, p, r)
            QUICKSORT$(A, p, q\text{–}1)$
            QUICKSORT$(A, q+1, r)$

Initial call: QUICKSORT$(A, 1, n)$



Analysis of quicksort

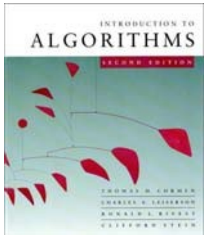
- Assume all input elements are distinct.
- In practice, there are better partitioning algorithms for when duplicate input elements may exist.
- Let $T(n)$ = worst-case running time on an array of n elements.



Worst-case of quicksort

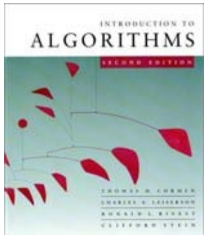
- Input sorted or reverse sorted.
- Partition around min or max element.
- One side of partition always has no elements.

$$
\begin{aligned}
T(n) &= T(0) + T(n-1) + \Theta(n) \\
&= \Theta(1) + T(n-1) + \Theta(n) \\
&= T(n-1) + \Theta(n) \\
&= \Theta(n^2) \qquad \textbf{\textit{(arithmetic series)}}
\end{aligned}
$$



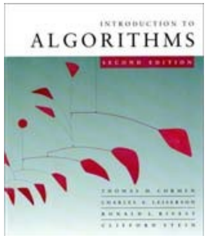
Worst-case recursion tree

$$T(n) = T(0) + T(n-1) + cn$$



Worst-case recursion tree

$$T(n) = T(0) + T(n-1) + cn \quad T(n)$$

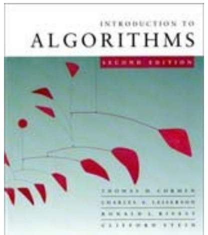


Worst-case recursion tree

$$T(n) = T(0) + T(n-1) + cn$$

$$cn$$

$$T(0) \qquad T(n-1)$$



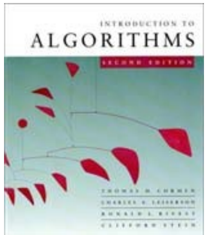
Worst-case recursion tree

$$T(n) = T(0) + T(n-1) + cn$$

$$cn$$

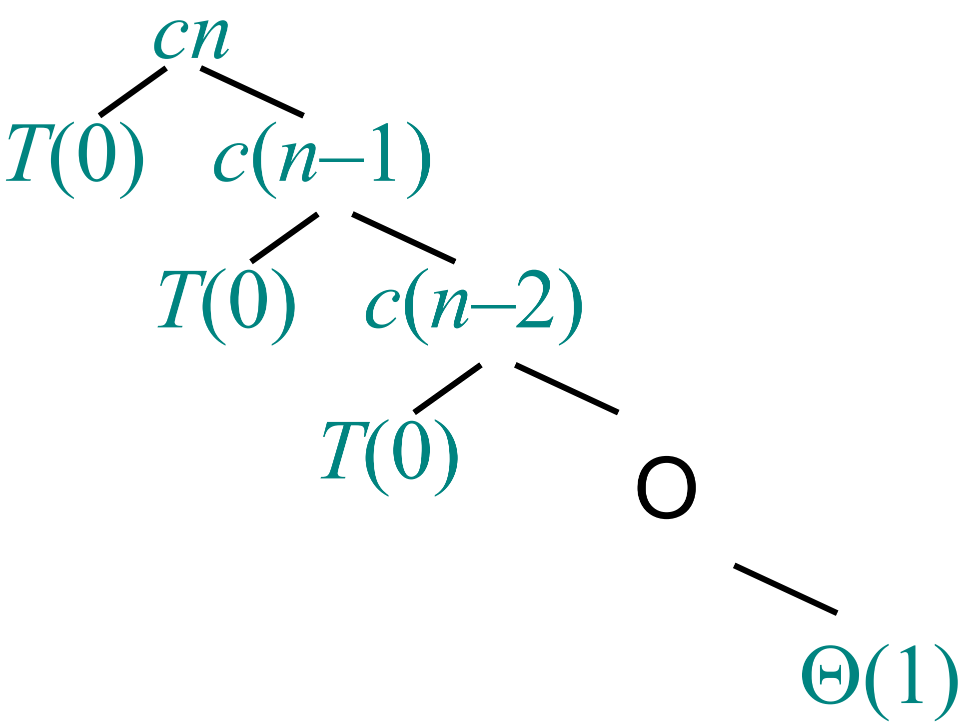
$$T(0) \qquad c(n-1)$$

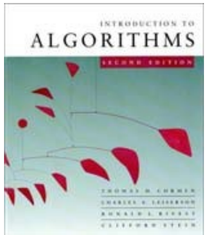
$$T(0) \qquad T(n-2)$$



Worst-case recursion tree

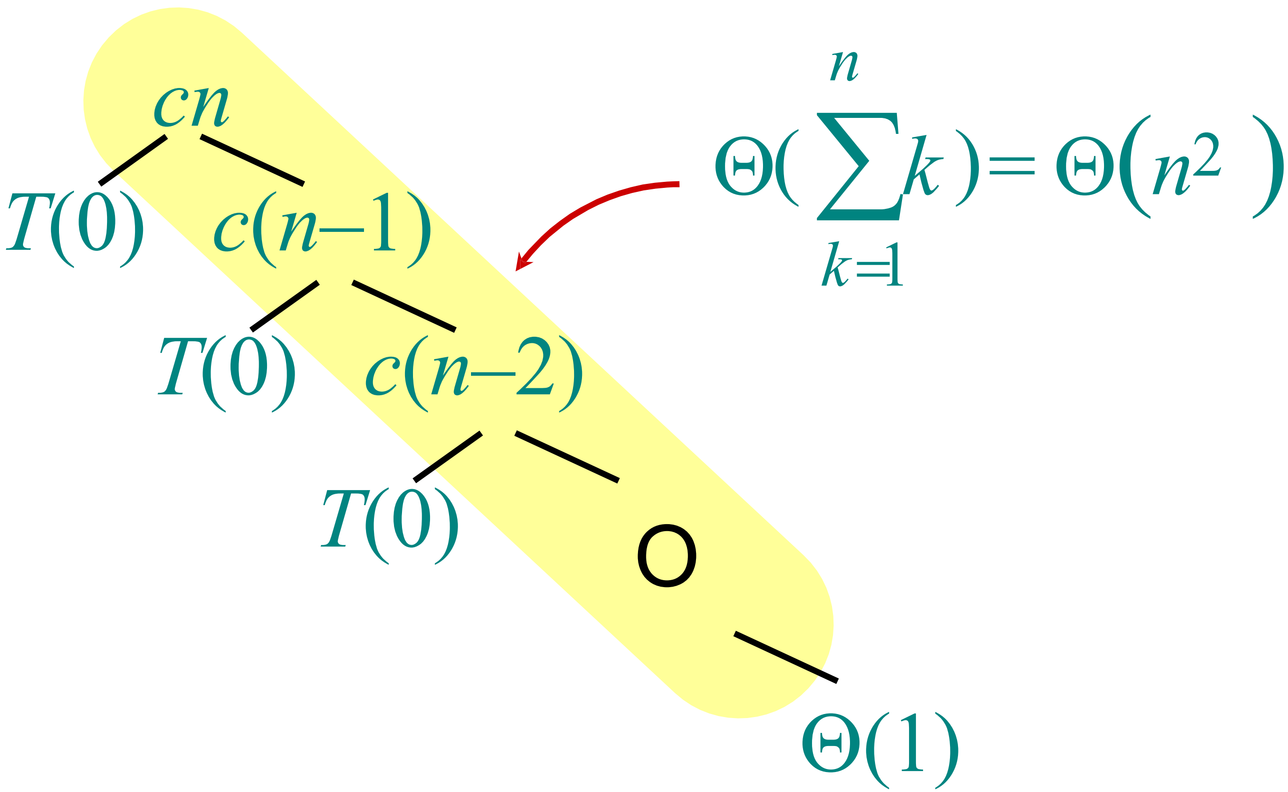
$$T(n) = T(0) + T(n-1) + cn$$

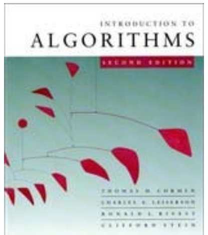




Worst-case recursion tree

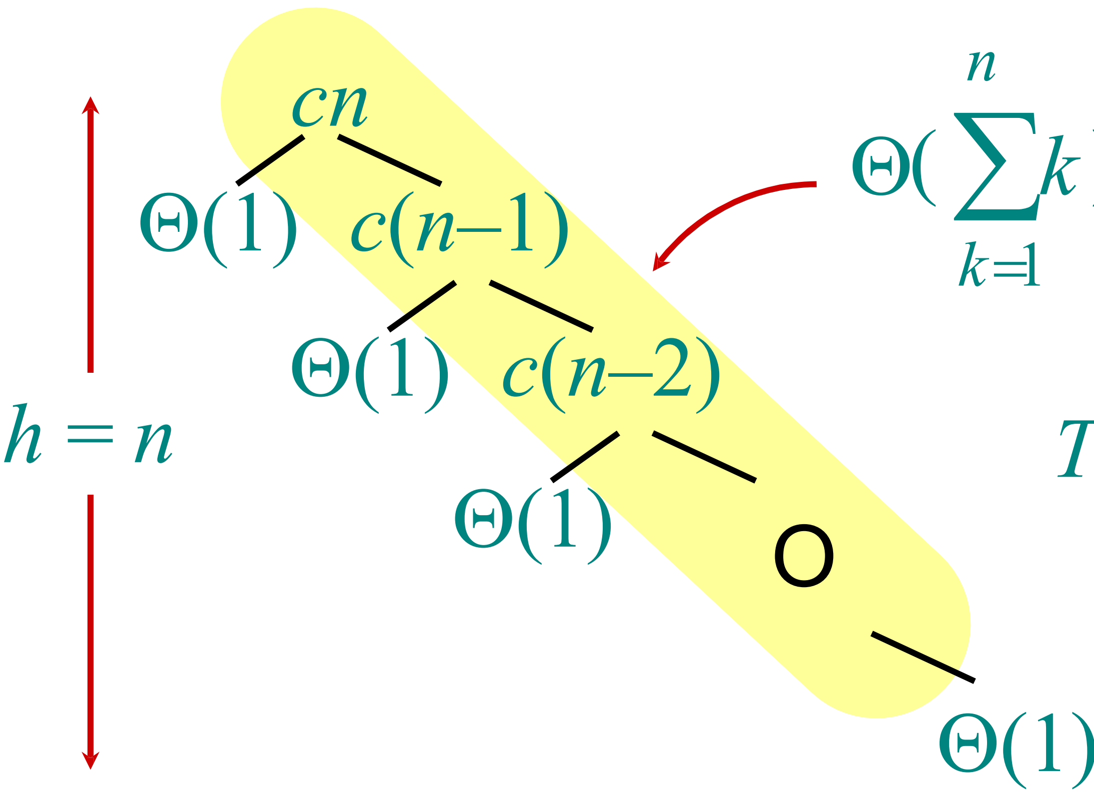
$$T(n) = T(0) + T(n-1) + cn$$





Worst-case recursion tree

$$T(n) = T(0) + T(n-1) + cn$$



$$\Theta\left(\sum_{k=1}^{n} k\right) = \Theta\left(n^2\right)$$

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