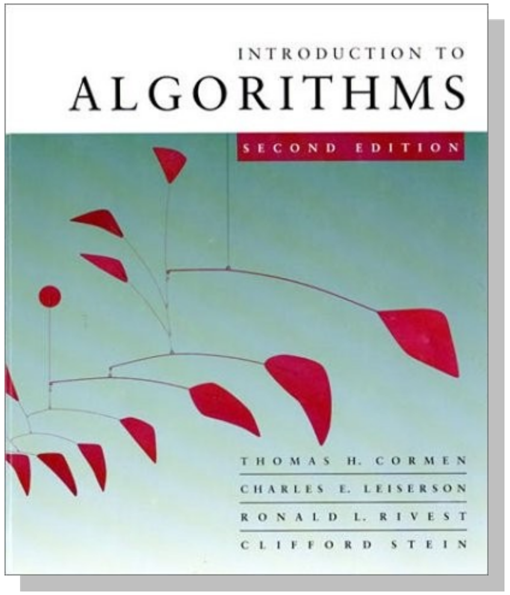
Introduction to Divide and Conquer

Analysis of BS, MS,QS

Introduction to Algorithms

6.046J/18.401J

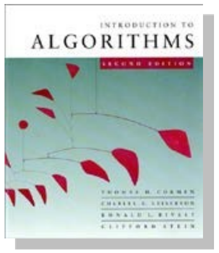


LECTURE 3

Divide and Conquer

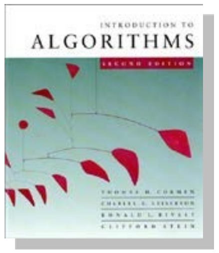
- Binary search
- Powering a number
- Fibonacci numbers
- Matrix multiplication
- Strassen's algorithm
- VLSI tree layout

Prof. Erik D. Demaine



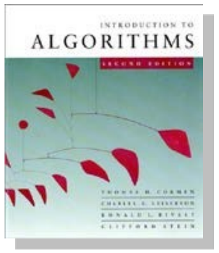
The divide-and-conquer design paradigm

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



Merge sort

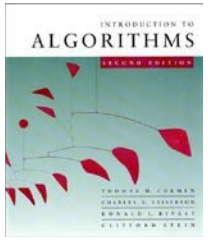
1. ***Divide:*** Trivial.
2. ***Conquer:*** Recursively sort 2 subarrays.
3. ***Combine:*** Linear-time merge.



Merge sort

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

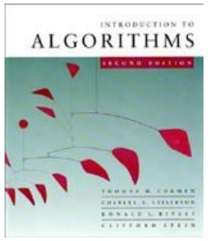
$$T(n) = \underbrace{2}_{\text{\# subproblems}}\, T(\underbrace{n/2}_{\text{subproblem size}}) + \underbrace{\Theta(n)}_{\text{work dividing and combining}}$$



Recurrence for merge sort

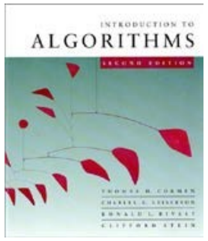
$$T(n) = \begin{cases} \Theta(1) \text{ if } n = 1; \\ 2T(n/2) + \Theta(n) \text{ if } n > 1. \end{cases}$$

- We shall usually omit stating the base case when $T(n) = \Theta(1)$ for sufficiently small n, but only when it has no effect on the asymptotic solution to the recurrence.



Recursion tree

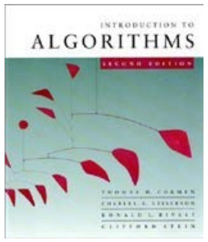
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.



Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

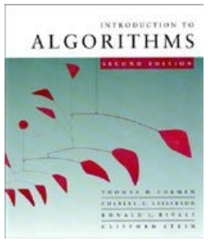
$$T(n)$$



Recursion tree

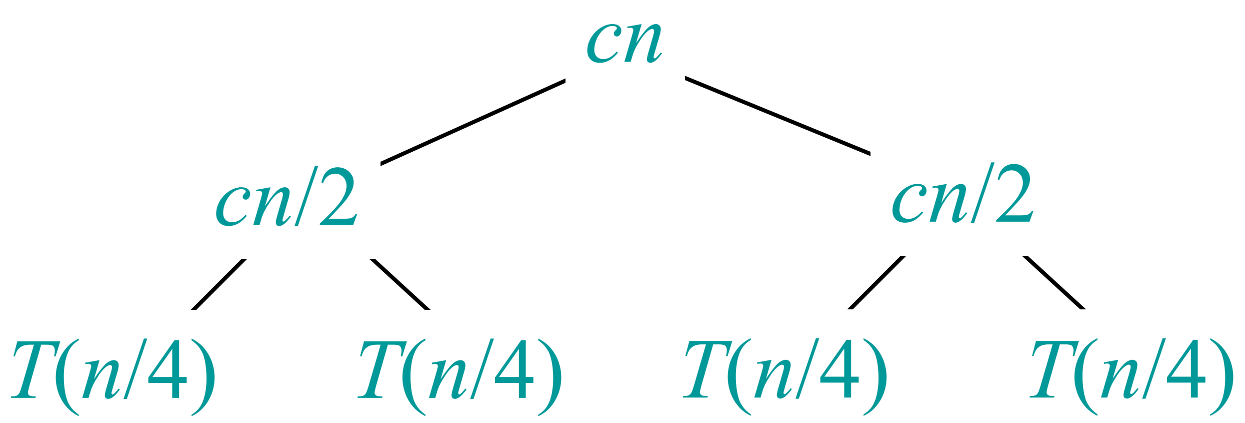
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

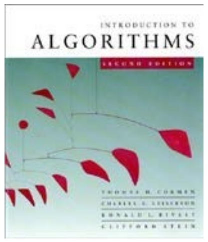
$$\begin{tikzcd}
 & cn \arrow[dl, dash] \arrow[dr, dash] & \\
T(n/2) & & T(n/2)
\end{tikzcd}$$



Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

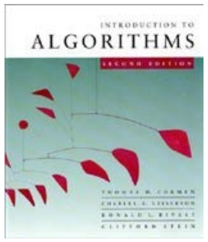




Recursion tree

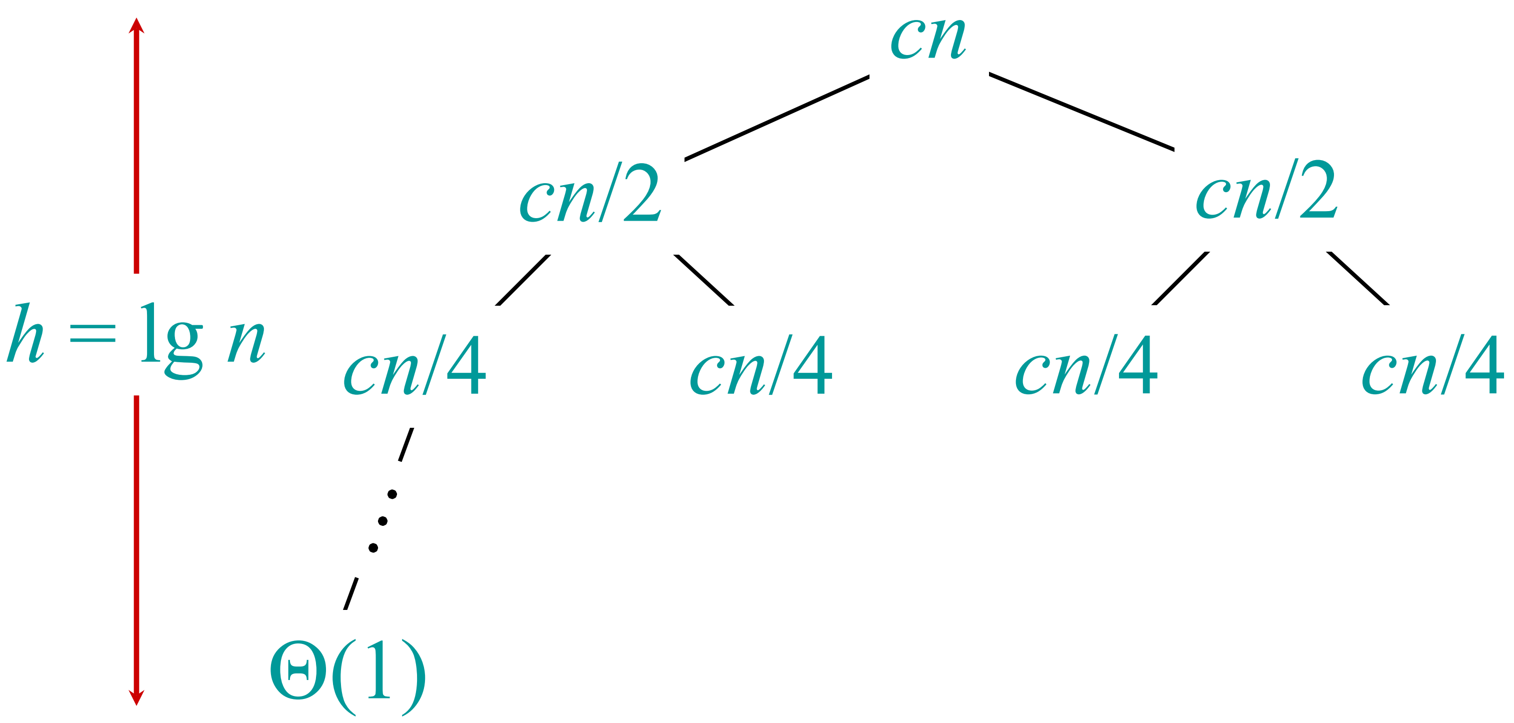
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

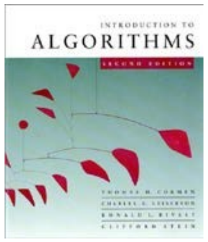
$$\begin{tikzcd}[arrows=-]
 & & & cn \arrow[dll] \arrow[drr] & & & \\
 & cn/2 \arrow[dl] \arrow[dr] & & & & cn/2 \arrow[dl] \arrow[dr] & \\
cn/4 \arrow[d] & & cn/4 & & cn/4 & & cn/4 \\
\vdots \arrow[d] & & & & & & \\
\Theta(1) & & & & & &
\end{tikzcd}$$



Recursion tree

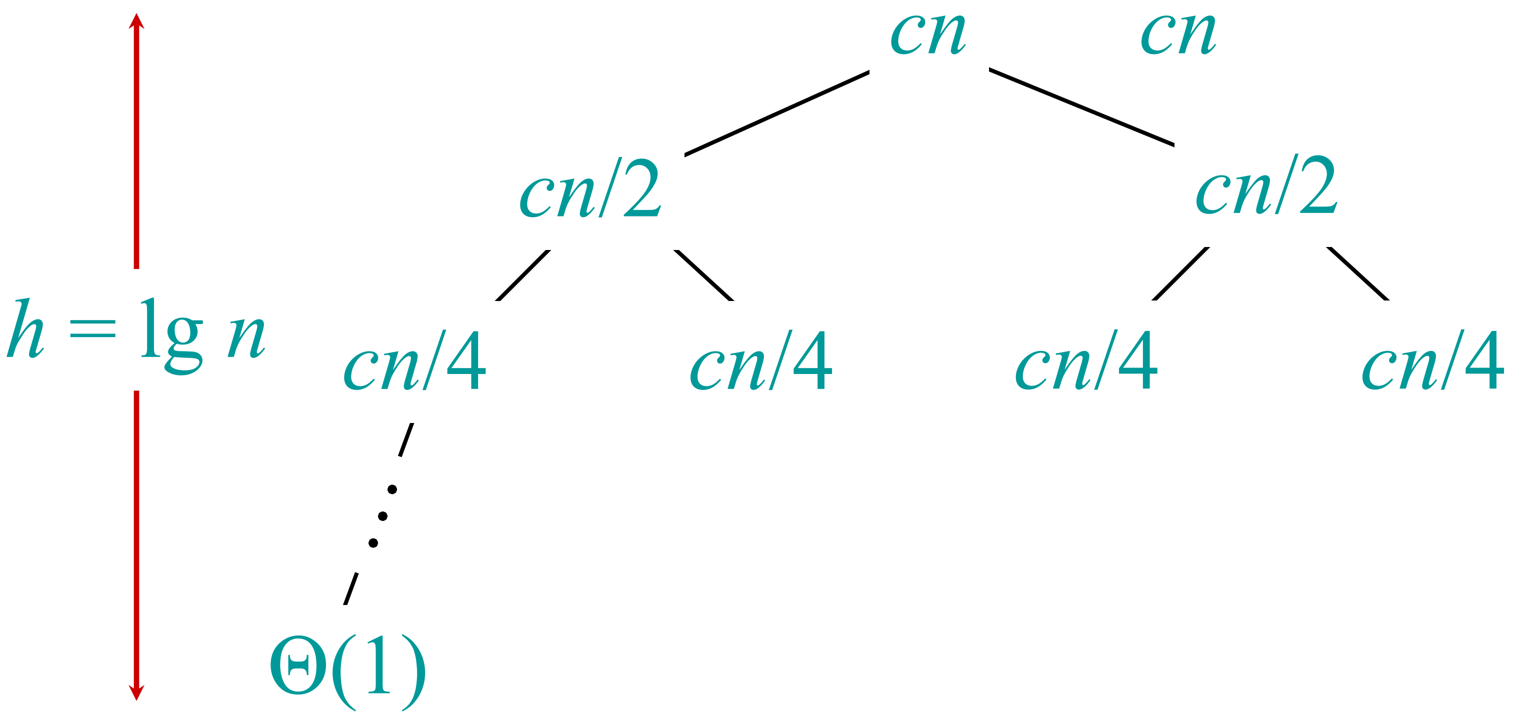
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

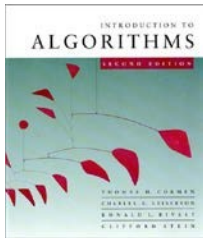




Recursion tree

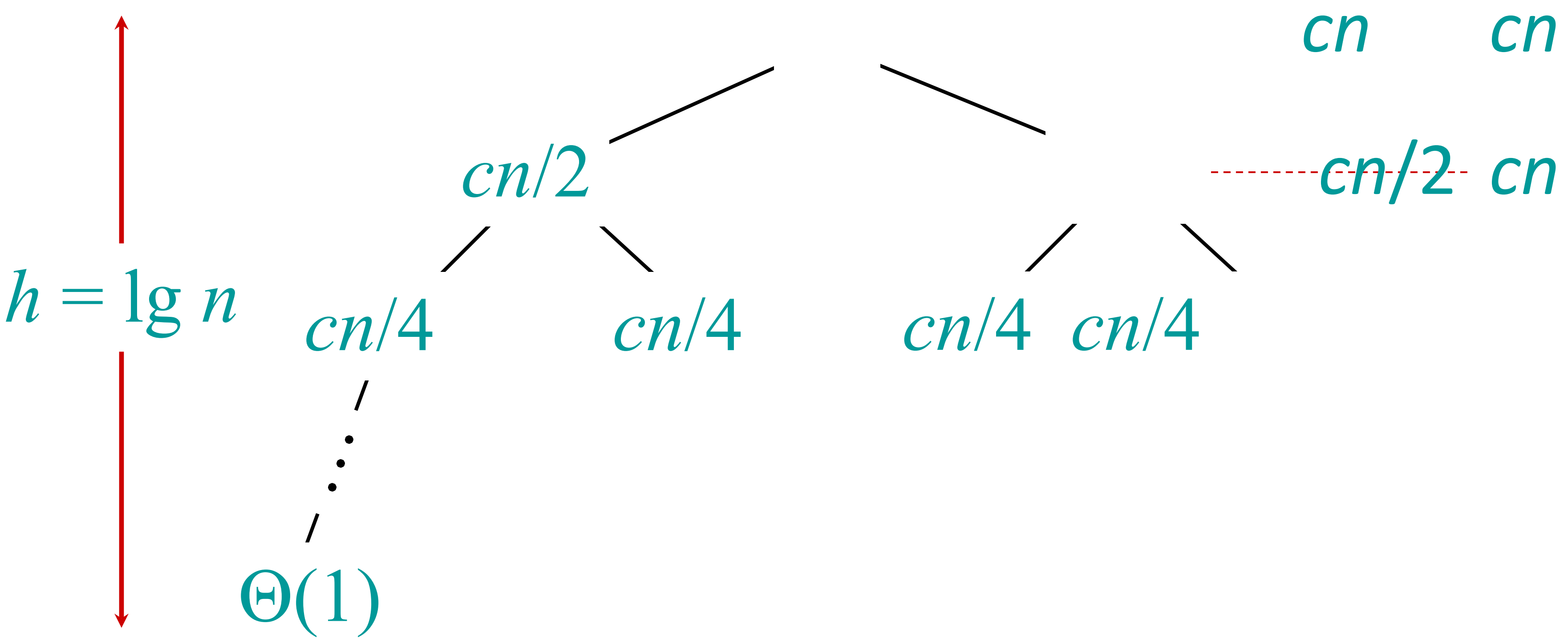
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

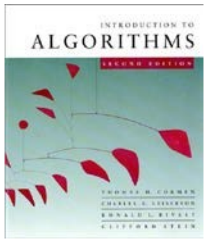




Recursion tree

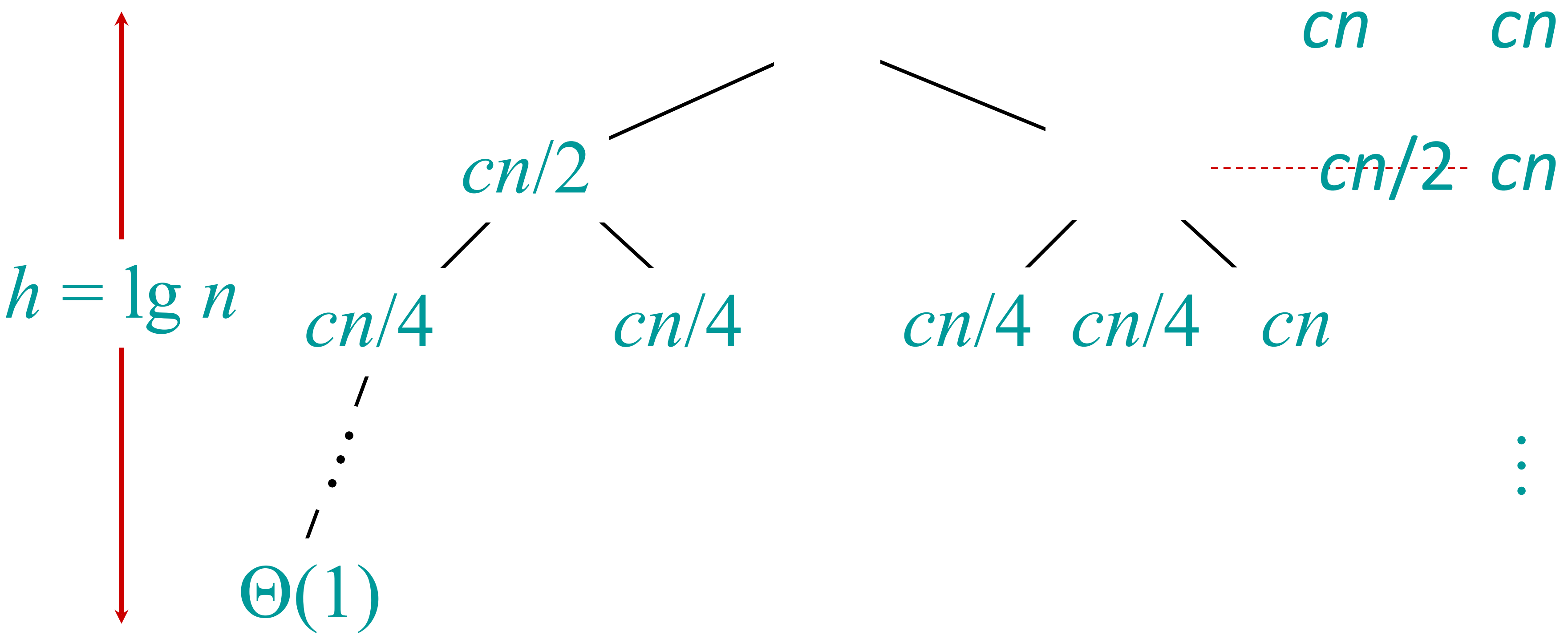
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

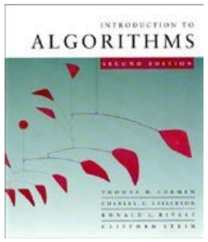




Recursion tree

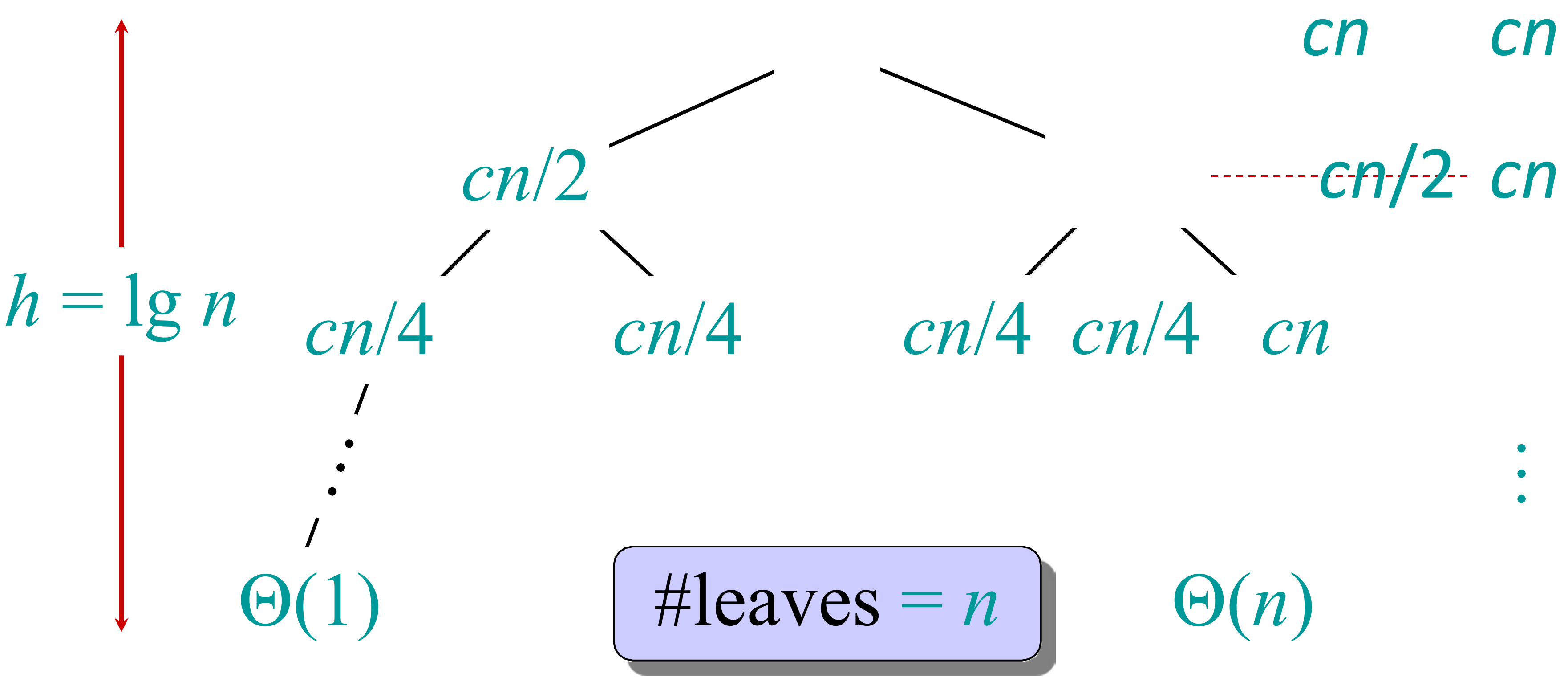
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

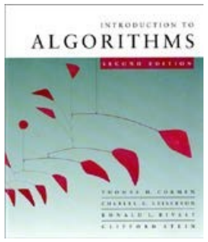




Recursion tree

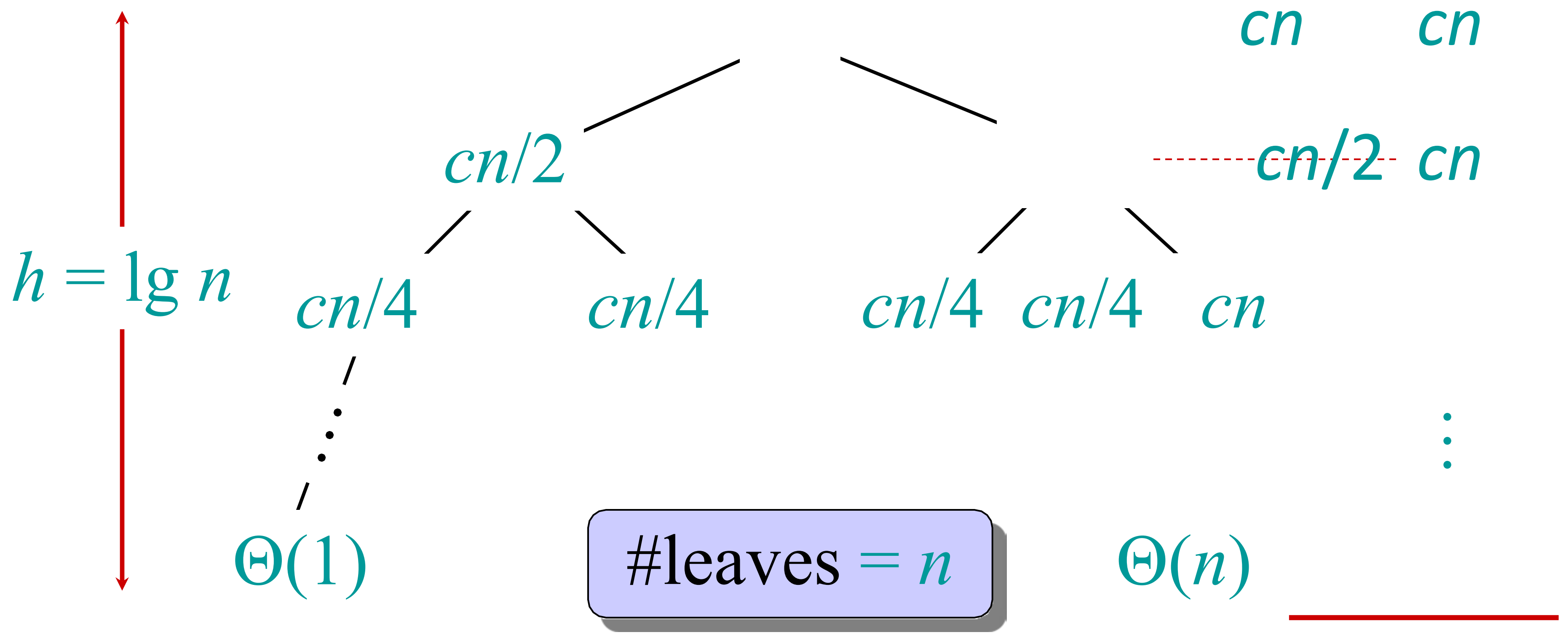
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.



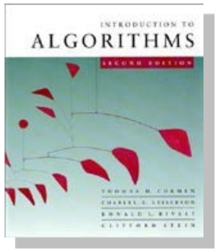


Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.



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



Master theorem (reprise)

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

Case 1: $f(n) = O(n^{\log_b a - \varepsilon})$, constant $\varepsilon > 0$

$\Rightarrow \; T(n) = \Theta(n^{\log_b a})$.

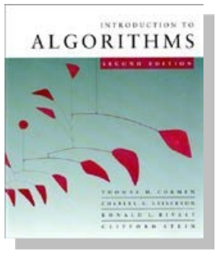
Case 2: $f(n) = \Theta(n^{\log_b a} \lg^k n)$, constant $k \geq 0$

$\Rightarrow \; T(n) = \Theta(n^{\log_b a} \lg^{k+1} n)$.

Case 3: $f(n) = \Omega(n^{\log_b a + \varepsilon})$, constant $\varepsilon > 0$,

and regularity condition

$\Rightarrow \; T(n) = \Theta(f(n))$.



Master theorem (reprise)

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

CASE 1: $f(n) = O(n^{\log_b a - \varepsilon})$, constant $\varepsilon > 0$

$\Rightarrow T(n) = \Theta(n^{\log_b a})$.

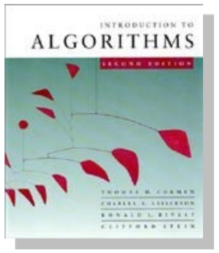
CASE 2: $f(n) = \Theta(n^{\log_b a} \lg^k n)$, constant $k \geq 0$

$\Rightarrow T(n) = \Theta(n^{\log_b a} \lg^{k+1} n)$.

CASE 3: $f(n) = \Omega(n^{\log_b a + \varepsilon})$, constant $\varepsilon > 0$, and regularity condition

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

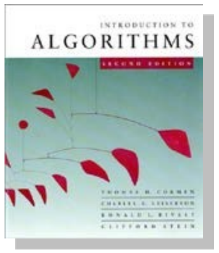
Merge sort: $a = 2,\ b = 2 \Rightarrow n^{\log_b a} = n^{\log_2 2} = n$



Binary search

Find an element in a sorted array:

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



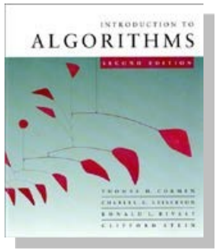
Binary search

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

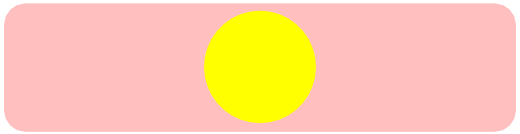


Binary search

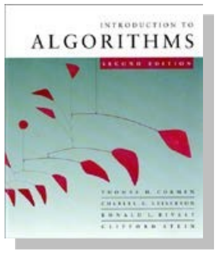
Find an element in a sorted array:

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

Example: Find 9



3 5 7 8 9 12 15



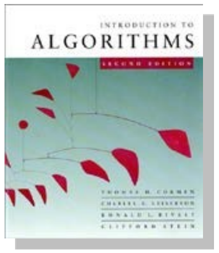
Binary search

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 9 12 15

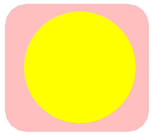


Binary search

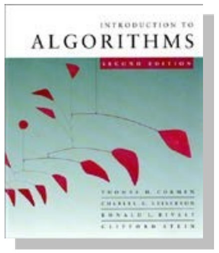
Find an element in a sorted array:

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

Example: Find 9



3 5 7 8 9 12 15



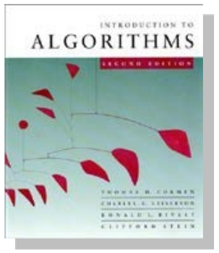
Recurrence for binary search

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

# subproblems

subproblem size

work dividing and combining



Recurrence for binary search

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

# subproblems

subproblem size

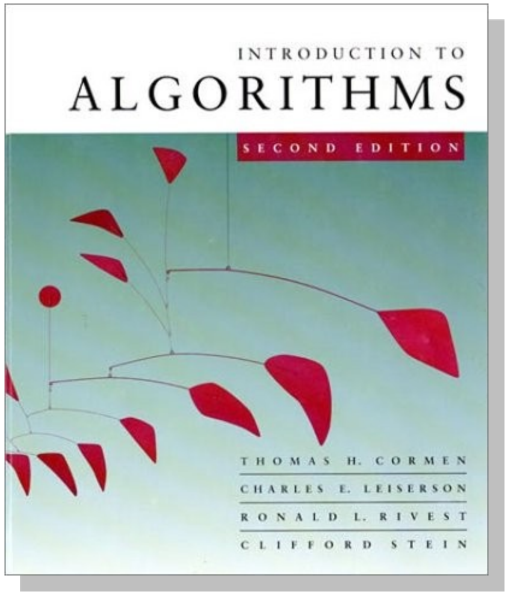
work dividing and combining

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

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

Introduction to Algorithms

6.046J/18.401J

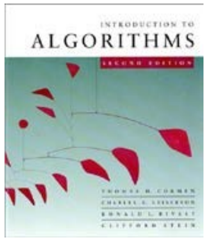


LECTURE 4

Quicksort

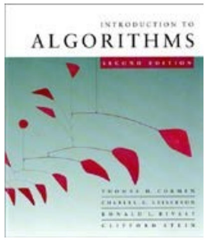
- Divide and conquer
- Partitioning
- Worst-case analysis
- Intuition
- Randomized quicksort
- Analysis

Prof. Charles E. Leiserson



Quicksort

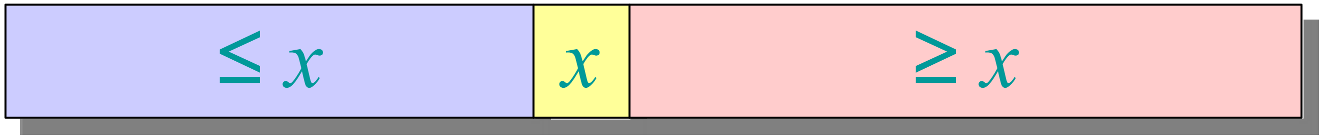
- Proposed by C.A.R. Hoare in 1962.
- Divide-and-conquer algorithm.
- Sorts "in place" (like insertion sort, but not like merge sort).
- Very practical (with tuning).



Divide and conquer

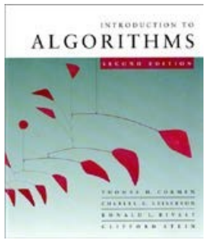
Quicksort an n-element array:

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.*



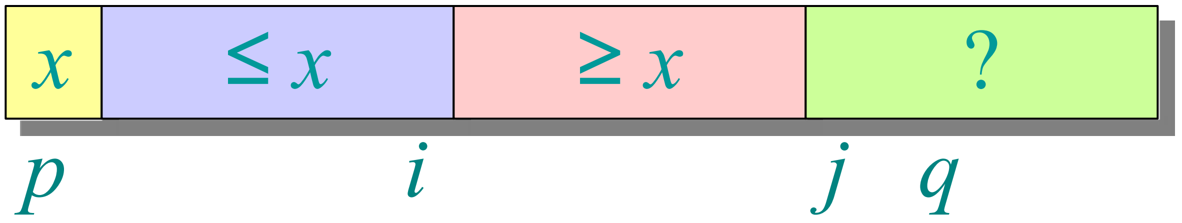
Partitioning subroutine

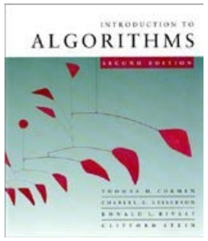
```
PARTITION($A, p, q$)  ▹ $A[p\,.\,.\,q]$
    $x \leftarrow A[p]$      ▹ pivot $= A[p]$
    $i \leftarrow p$
    for $j \leftarrow p+1$ to $q$
        do if $A[j] \le x$
                then  $i \leftarrow i+1$

                      exchange $A[i] \leftrightarrow A[$
    $j]$  exchange $A[p] \leftrightarrow A[i]$
    return $i$
```

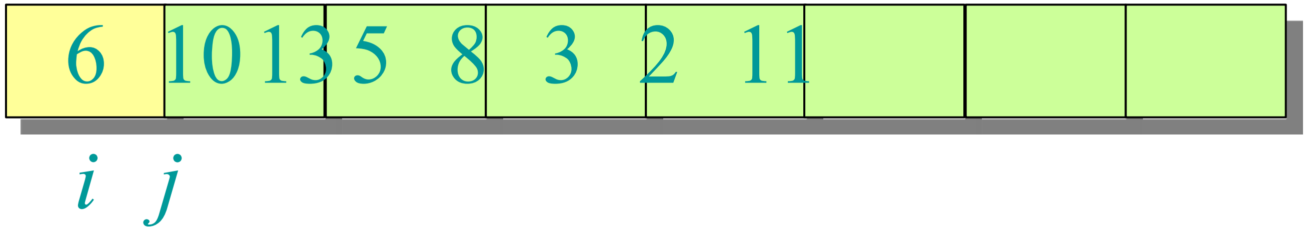
Running time $= O(n)$ for n elements.

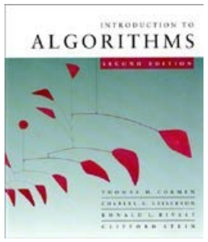
Invariant:



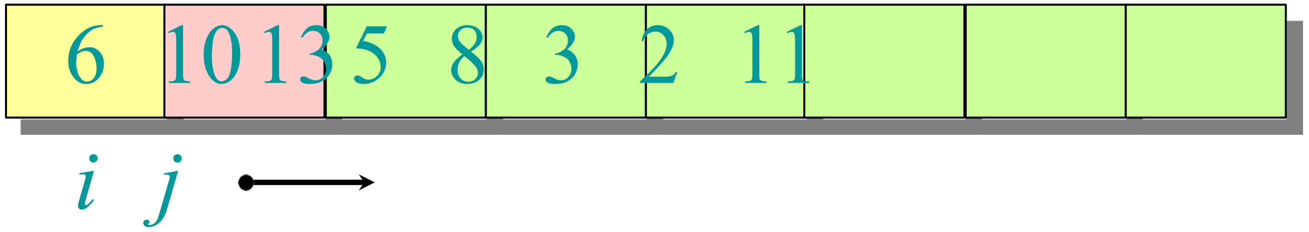


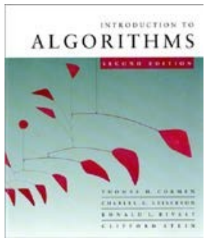
Example of partitioning



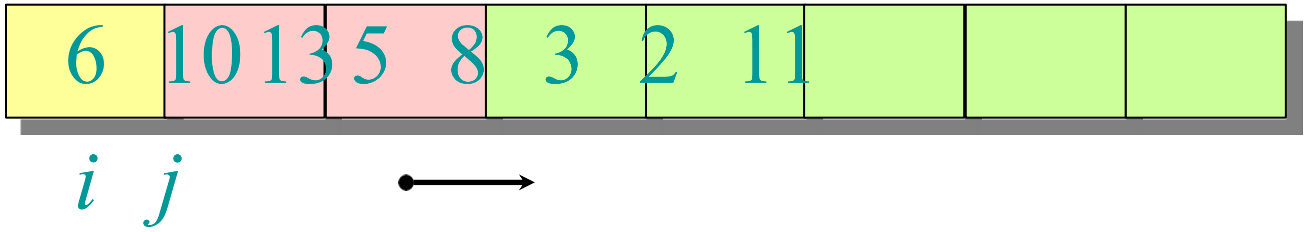


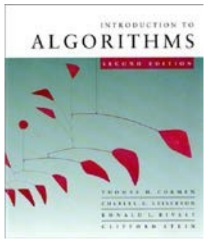
Example of partitioning



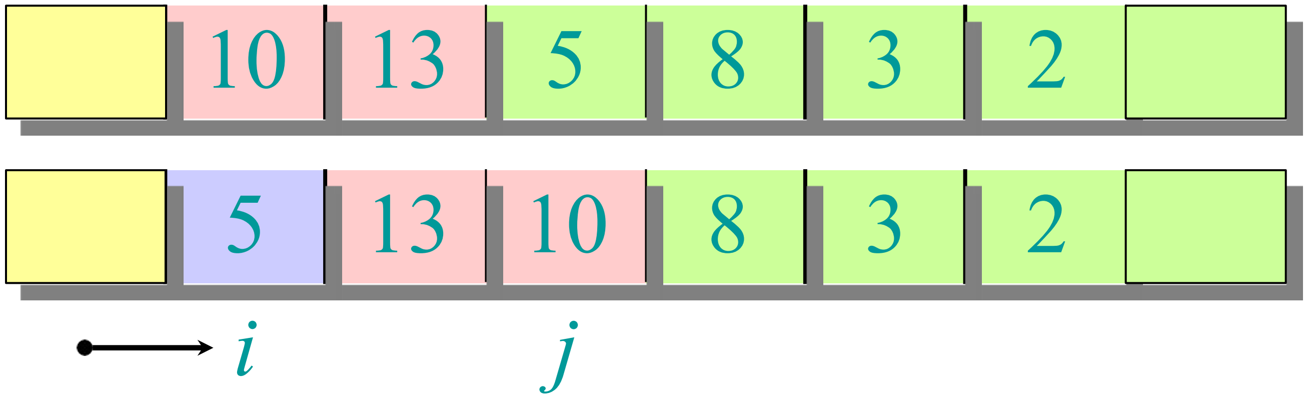


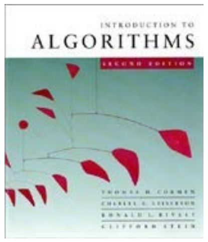
Example of partitioning



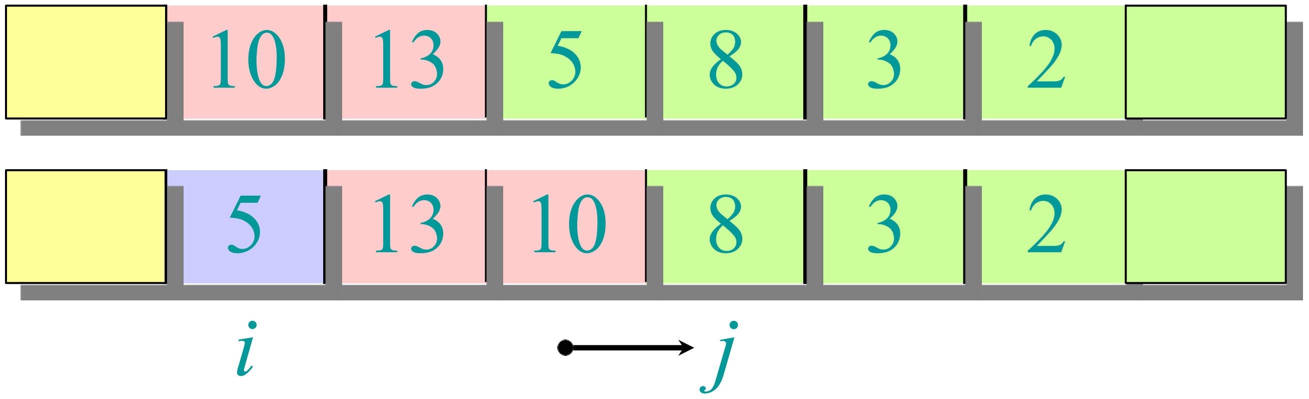


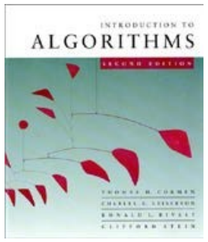
Example of partitioning



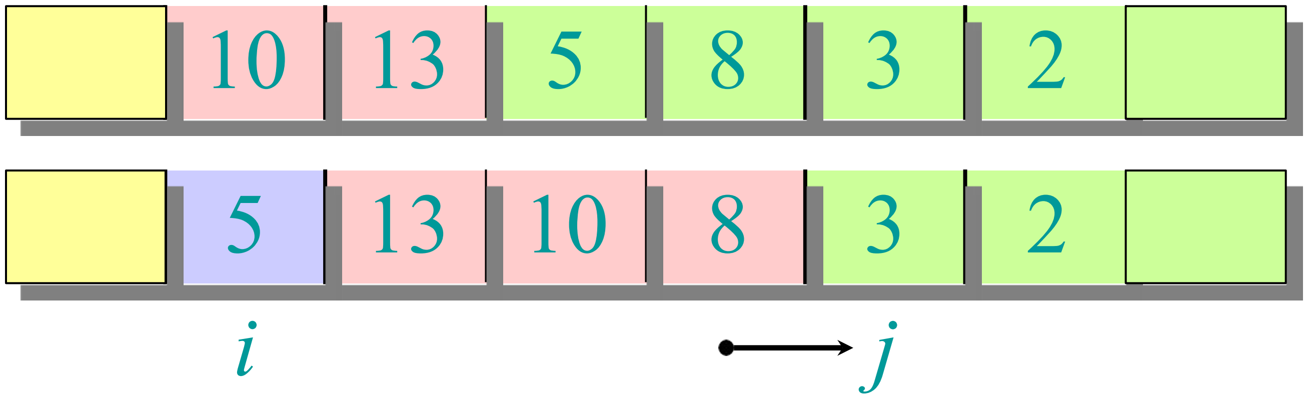


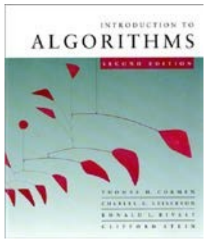
Example of partitioning



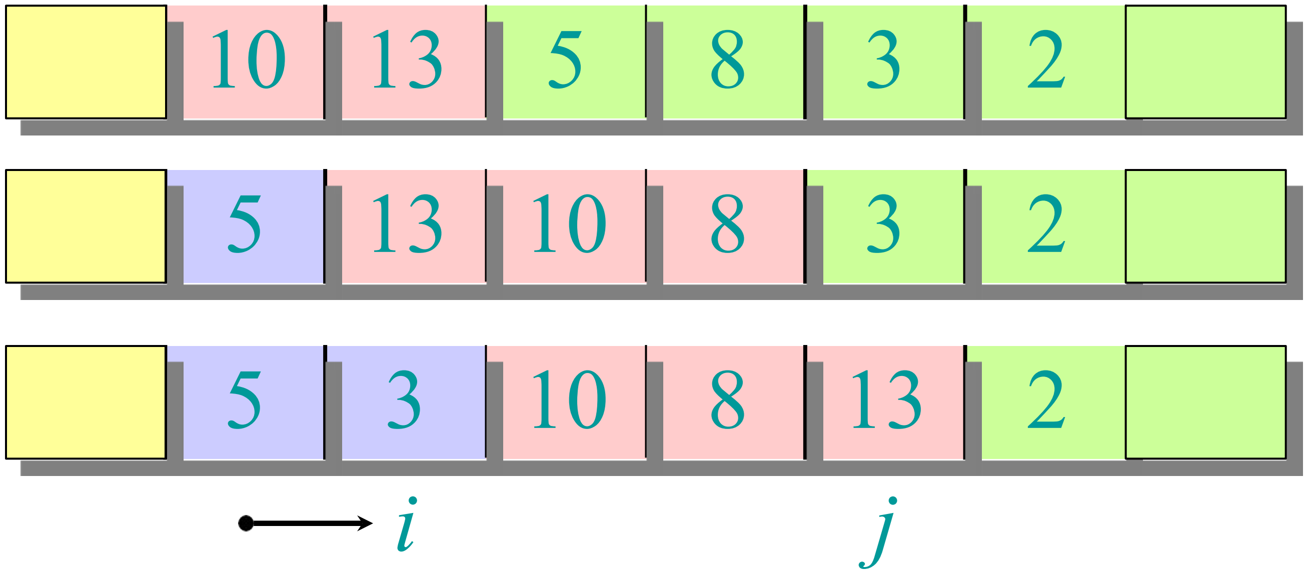


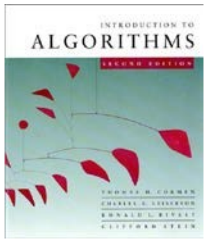
Example of partitioning



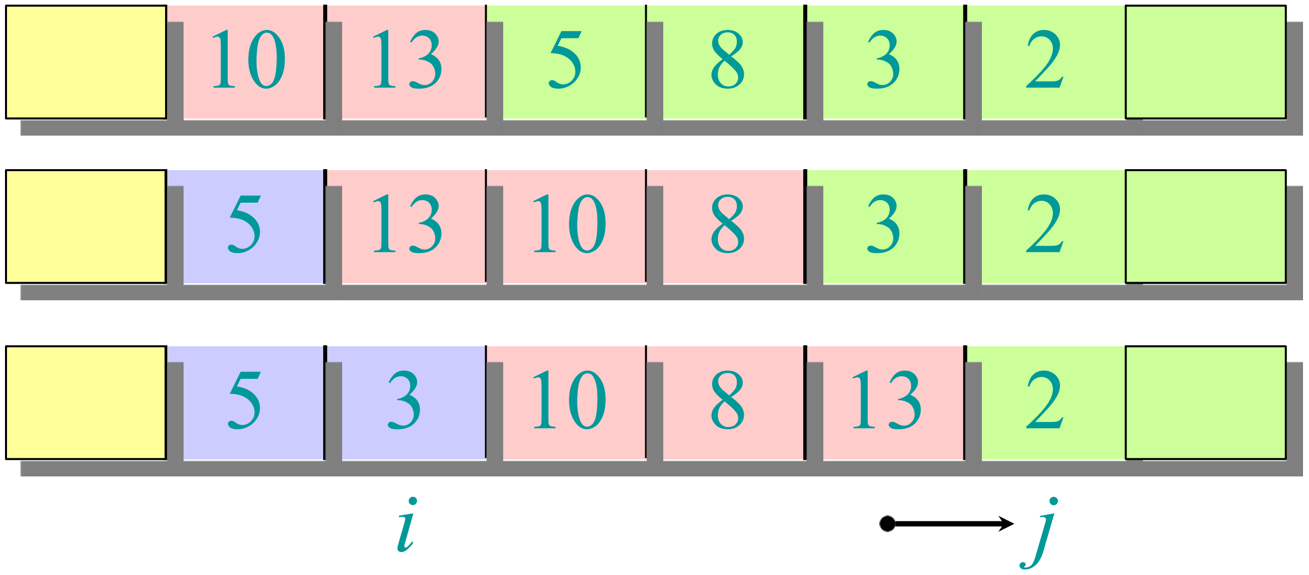


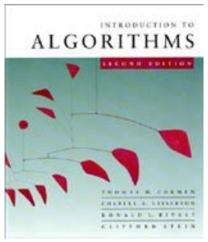
Example of partitioning



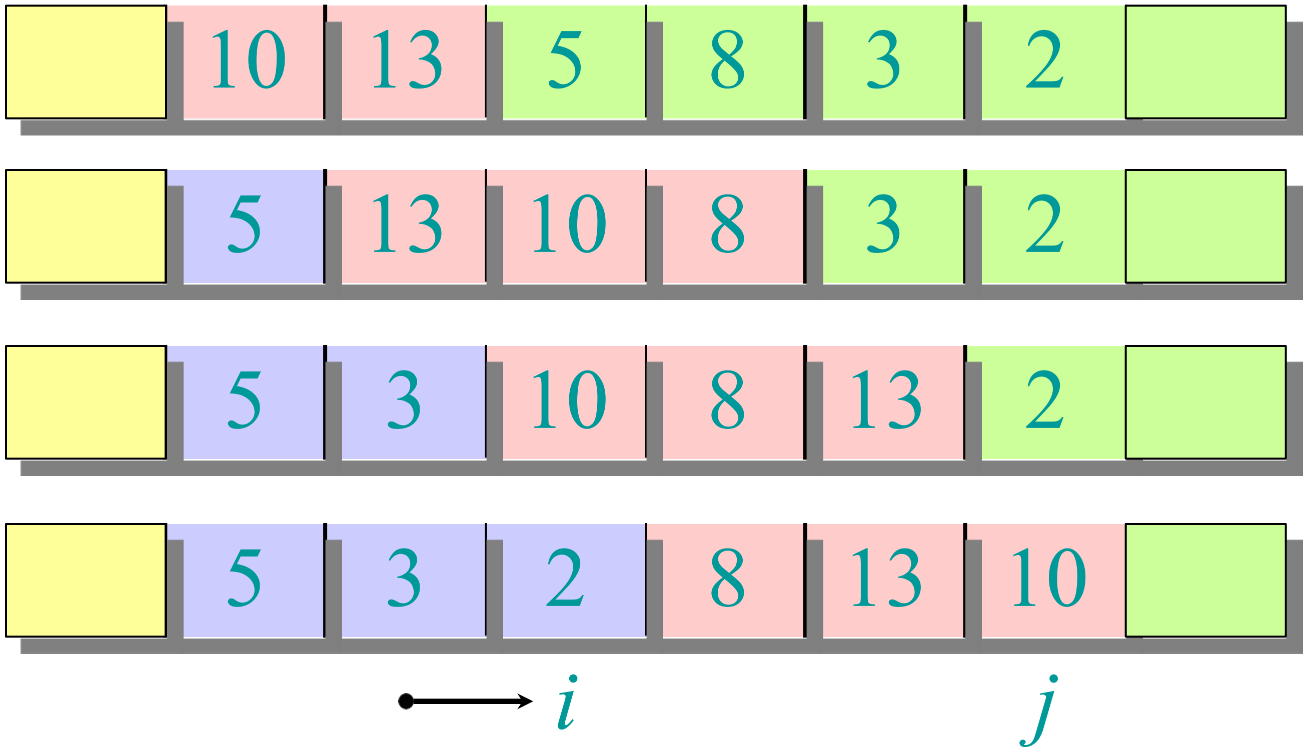


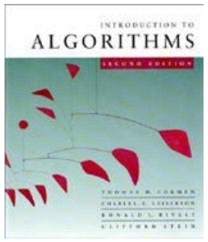
Example of partitioning



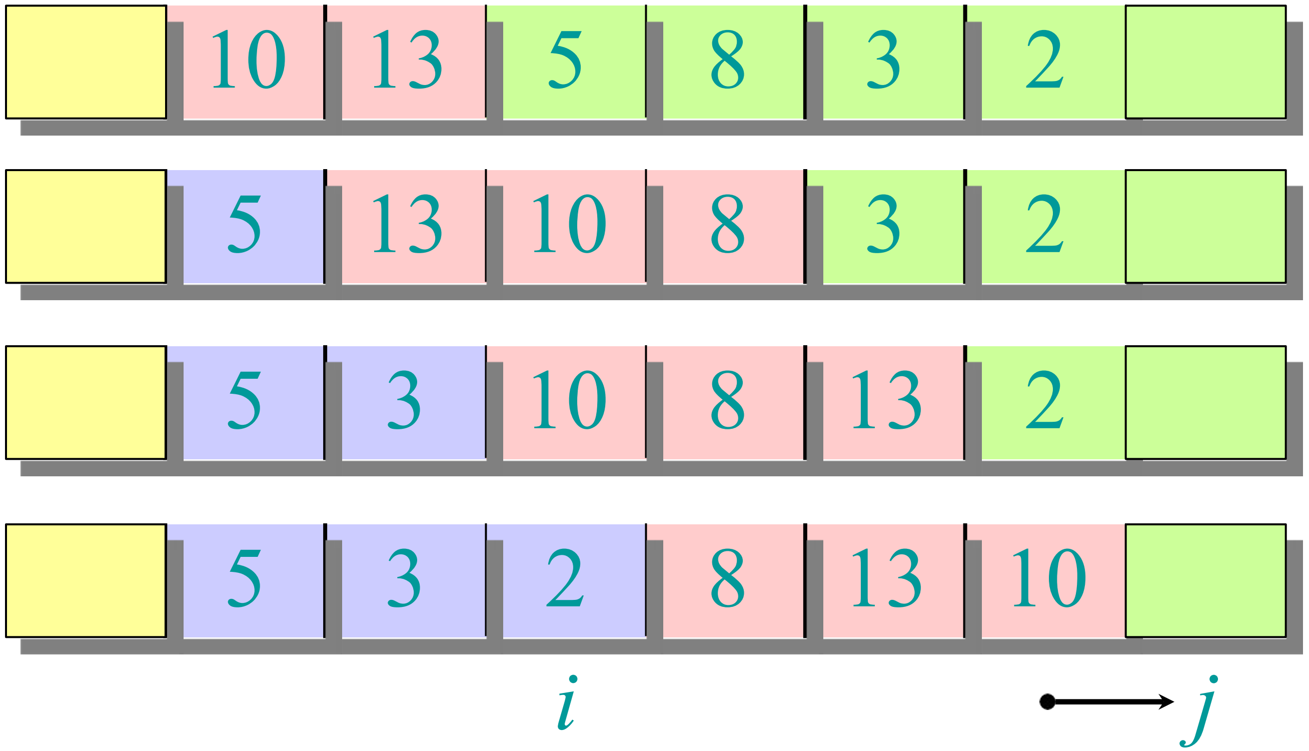


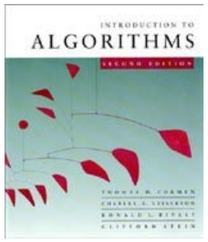
Example of partitioning



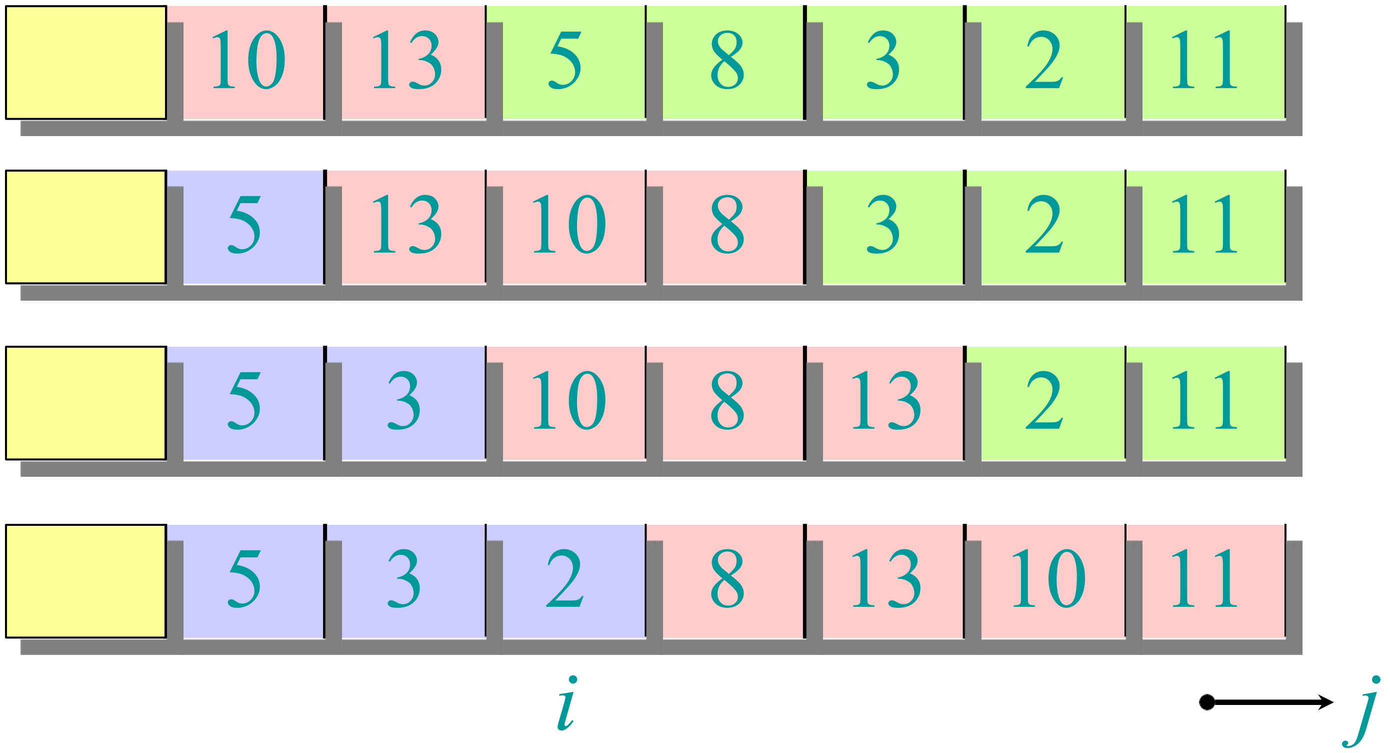


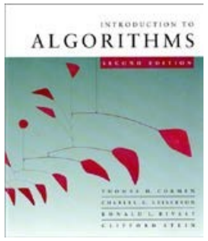
Example of partitioning



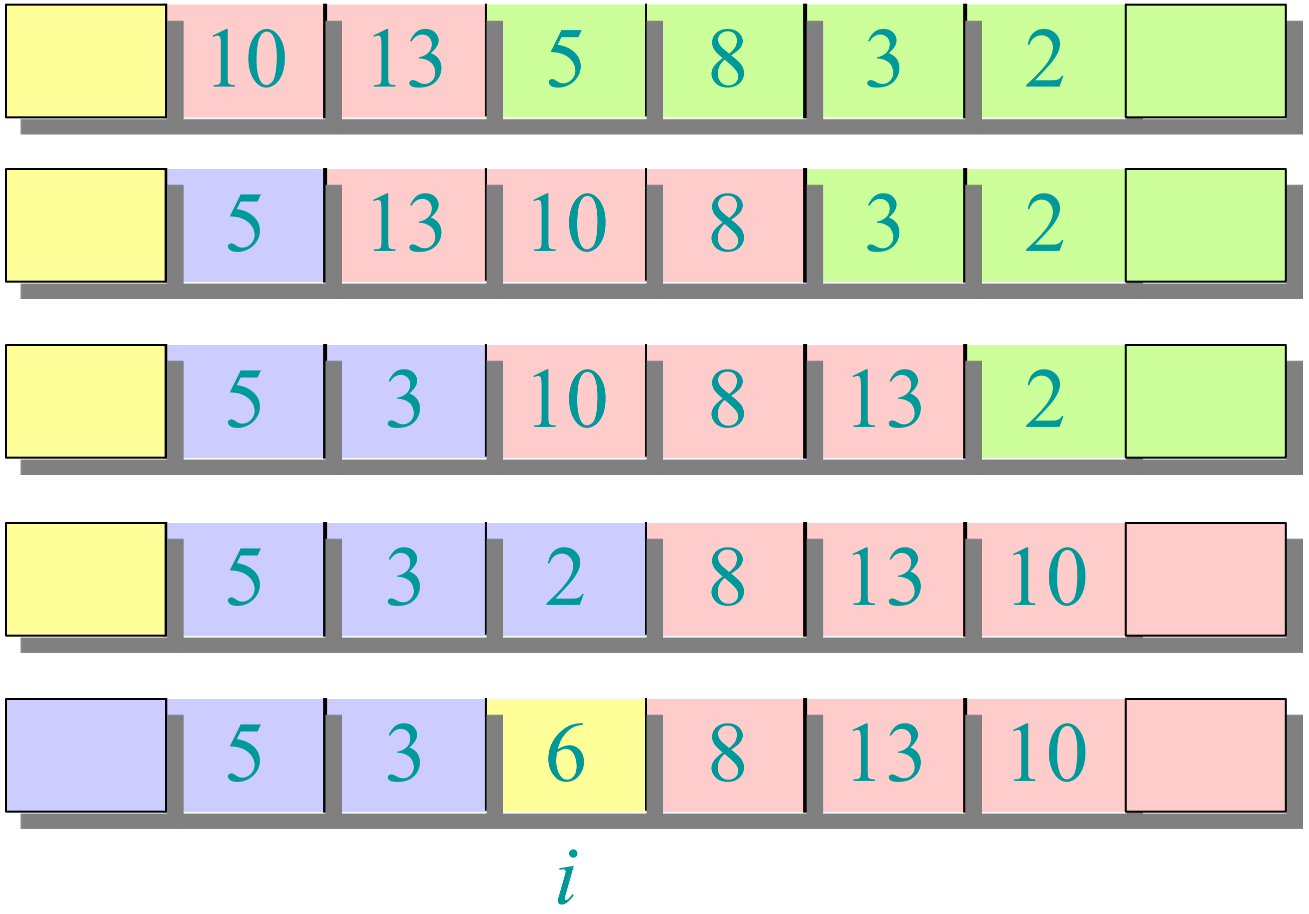


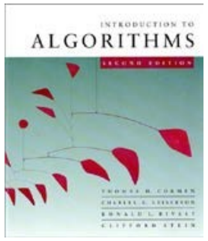
Example of partitioning





Example of partitioning

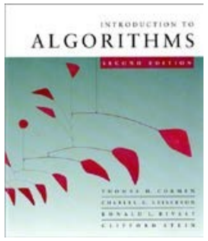




Pseudocode for quicksort

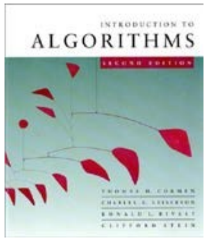
```
QUICKSORT$(A, p, r)$
    if $p < r$
        then $q \leftarrow$ PARTITION$(A, p, r)$
             QUICKSORT$(A, p, q-1)$
             QUICKSORT$(A, q+1, r)$
```

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



Analysis of quicksort

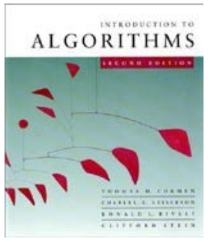
- Assume all input elements are distinct.
- In practice, there are better partitioning algorithms for when duplicate input elements may exist.
- Let $T(n)$ = worst-case running time on an array of n elements.



Worst-case of quicksort

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

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



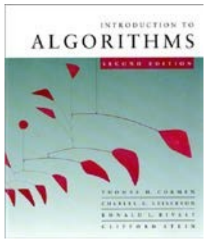
Worst-case recursion tree

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

Worst-case recursion tree

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

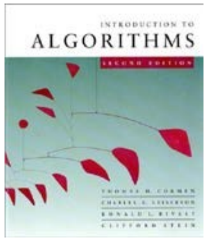
$$T(n)$$



Worst-case recursion tree

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

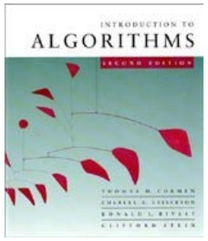
$$\begin{tikzcd}[column sep=small] & cn \arrow[dl, no head] \arrow[dr, no head] & \\ T(0) & & T(n-1) \end{tikzcd}$$



Worst-case recursion tree

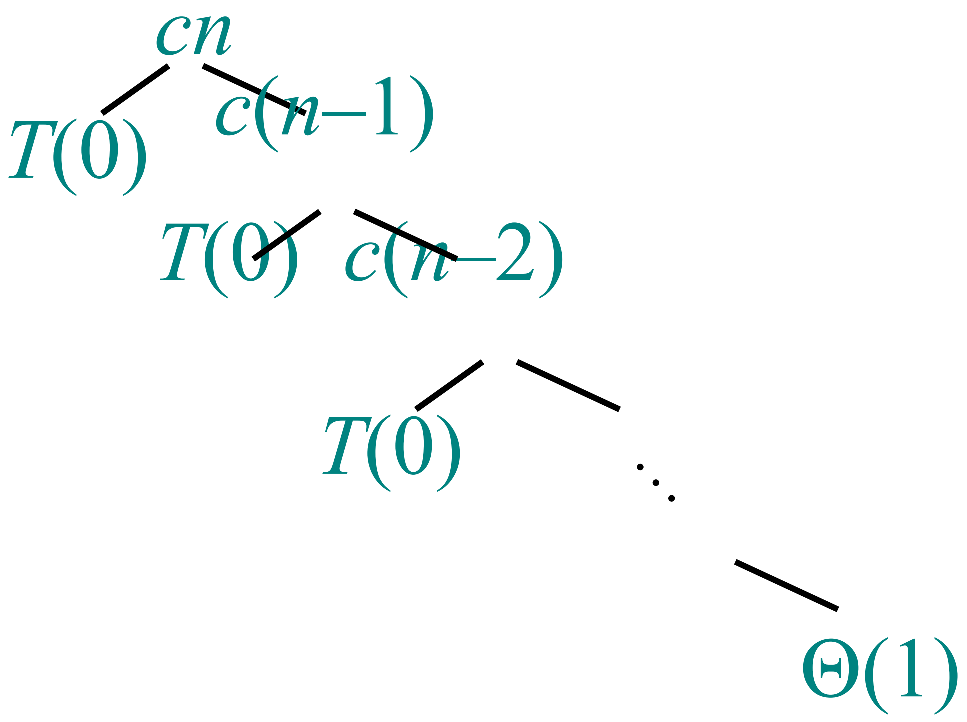
$$T(n) = T(0) + T(n–1) + cn$$

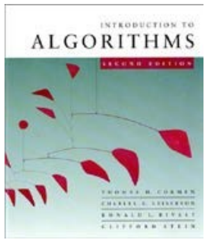
$$\begin{tikzcd}[arrows=dash]
 & cn \arrow[dl] \arrow[dr] & & \\
T(0) & & c(n–1) \arrow[dl] \arrow[dr] & \\
 & T(0) & & T(n–2)
\end{tikzcd}$$



Worst-case recursion tree

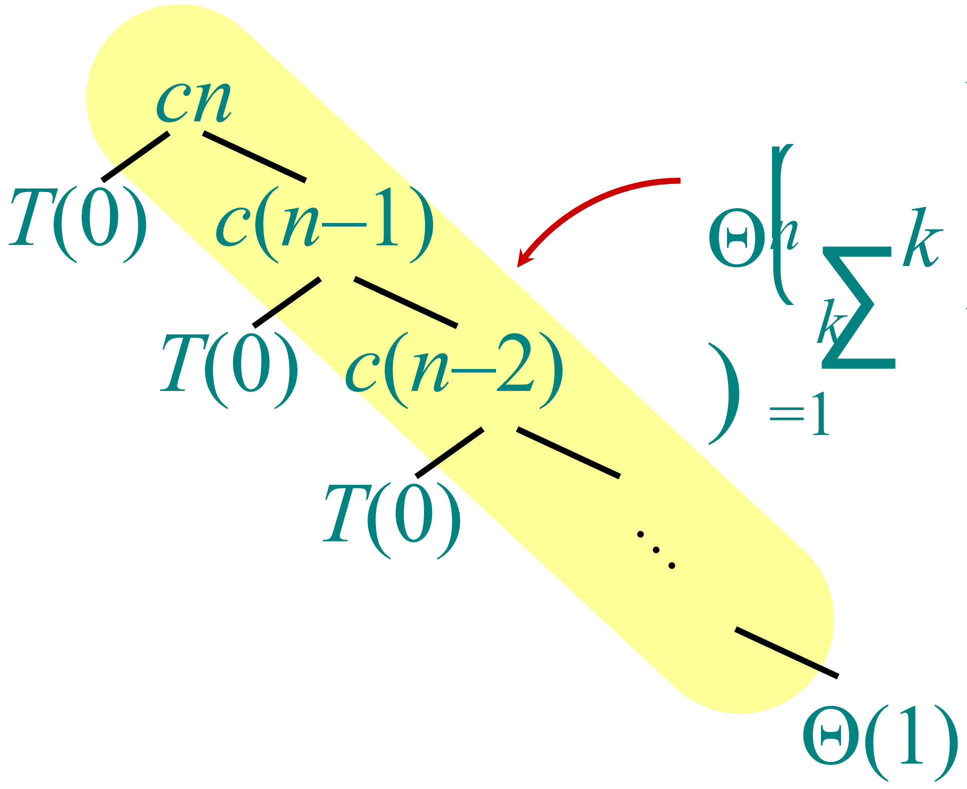
$$T(n) = T(0) + T(n-1) + cn$$



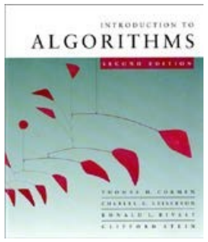


Worst-case recursion tree

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

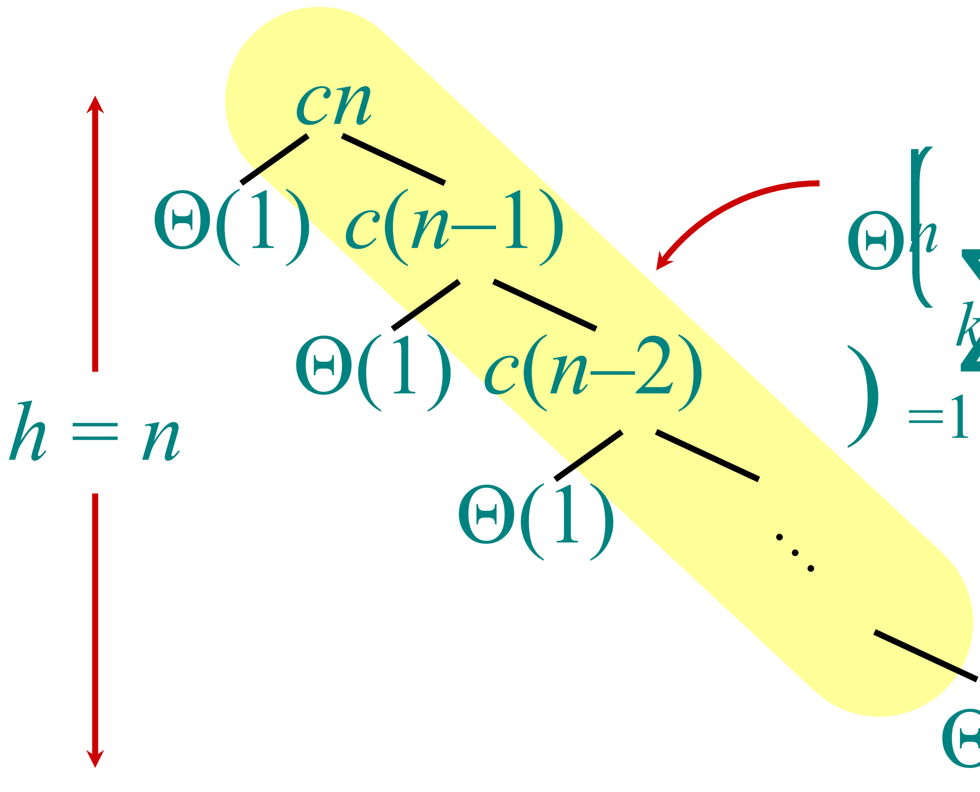


$$\Theta\left(\sum_{k=1}^{n} k\right) = \Theta(n^2)$$



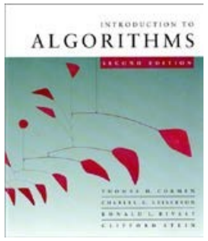
Worst-case recursion tree

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



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

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



Best-case analysis

(For intuition only!)

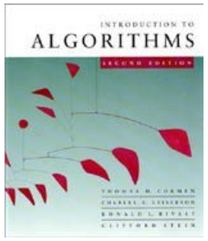
If we’re lucky, PARTITION splits the array evenly:

$T(n) = 2T(n/2) + \Theta(n)$
$= \Theta(n \lg n)$ (same as merge sort)

What if the split is always $\frac{1}{10} : \frac{9}{10}$?

$$T(n) = T\left(\tfrac{1}{10}n\right) + T\left(\tfrac{9}{10}n\right) + \Theta(n)$$

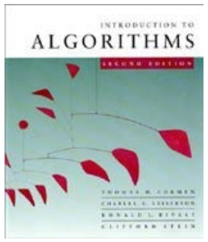
What is the solution to this recurrence?



Analysis of “almost-best” case

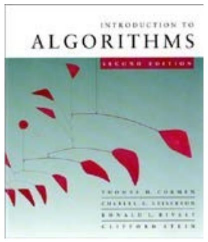
T

(n)



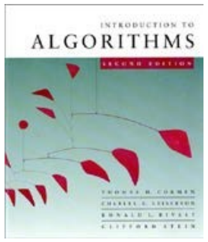
Analysis of “almost-best” case

$$\begin{tikzcd}
 & cn \arrow[dl, dash] \arrow[dr, dash] & \\
T\left(\frac{1}{10}n\right) & & T\left(\frac{9}{10}n\right)
\end{tikzcd}$$

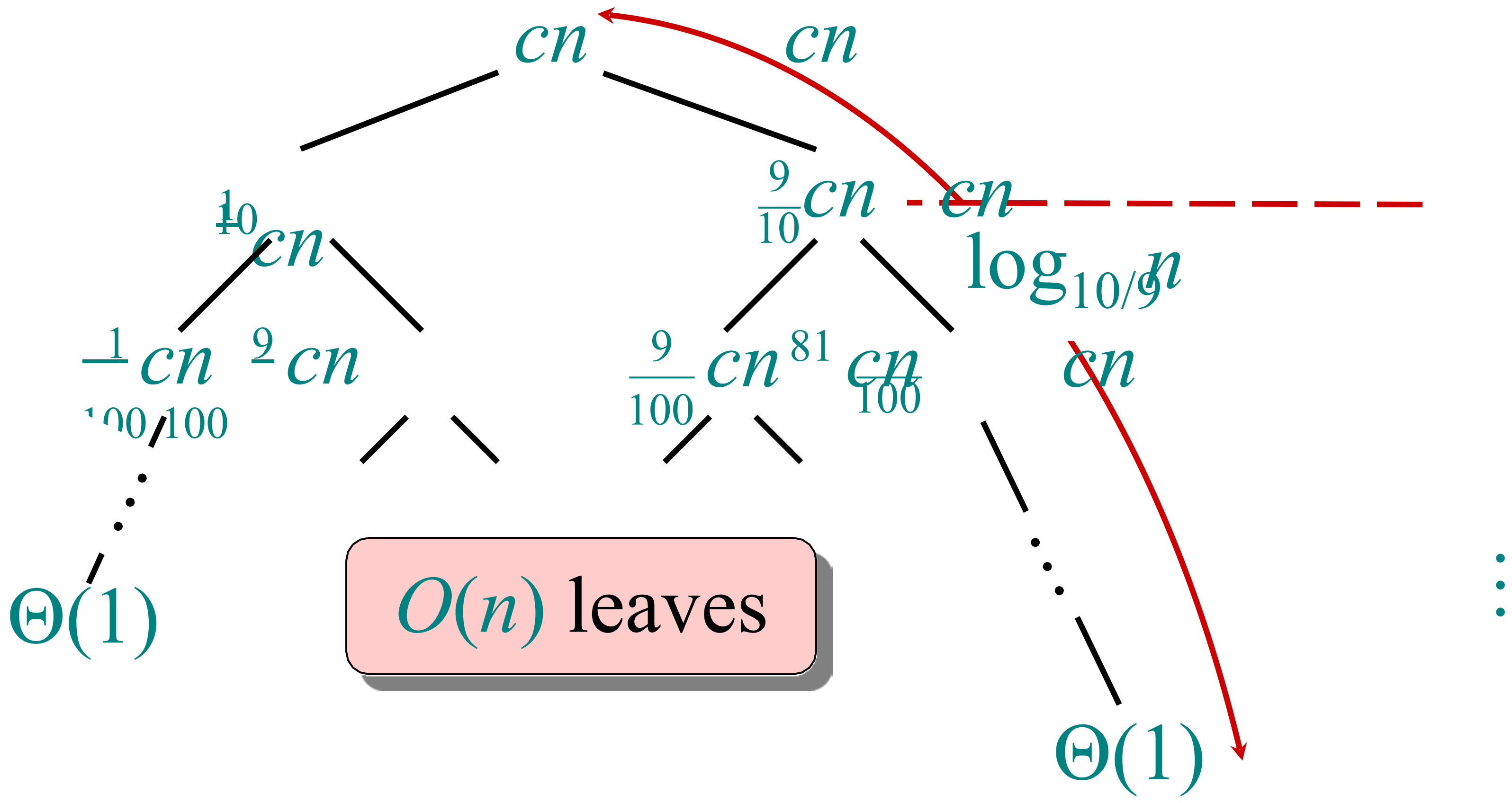


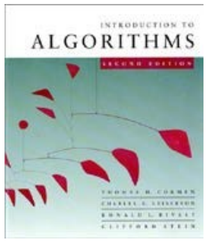
Analysis of “almost-best” case

$$\begin{tikzcd}
 & & & cn \arrow[dll, dash] \arrow[drr, dash] & & & \\
 & \frac{1}{10}cn \arrow[dl, dash] \arrow[dr, dash] & & & & \frac{9}{10}cn \arrow[dl, dash] \arrow[dr, dash] & \\
T\left(\frac{1}{100}n\right) & & T\left(\frac{9}{100}n\right) & & T\left(\frac{9}{100}n\right) & & T\left(\frac{81}{100}n\right)
\end{tikzcd}$$

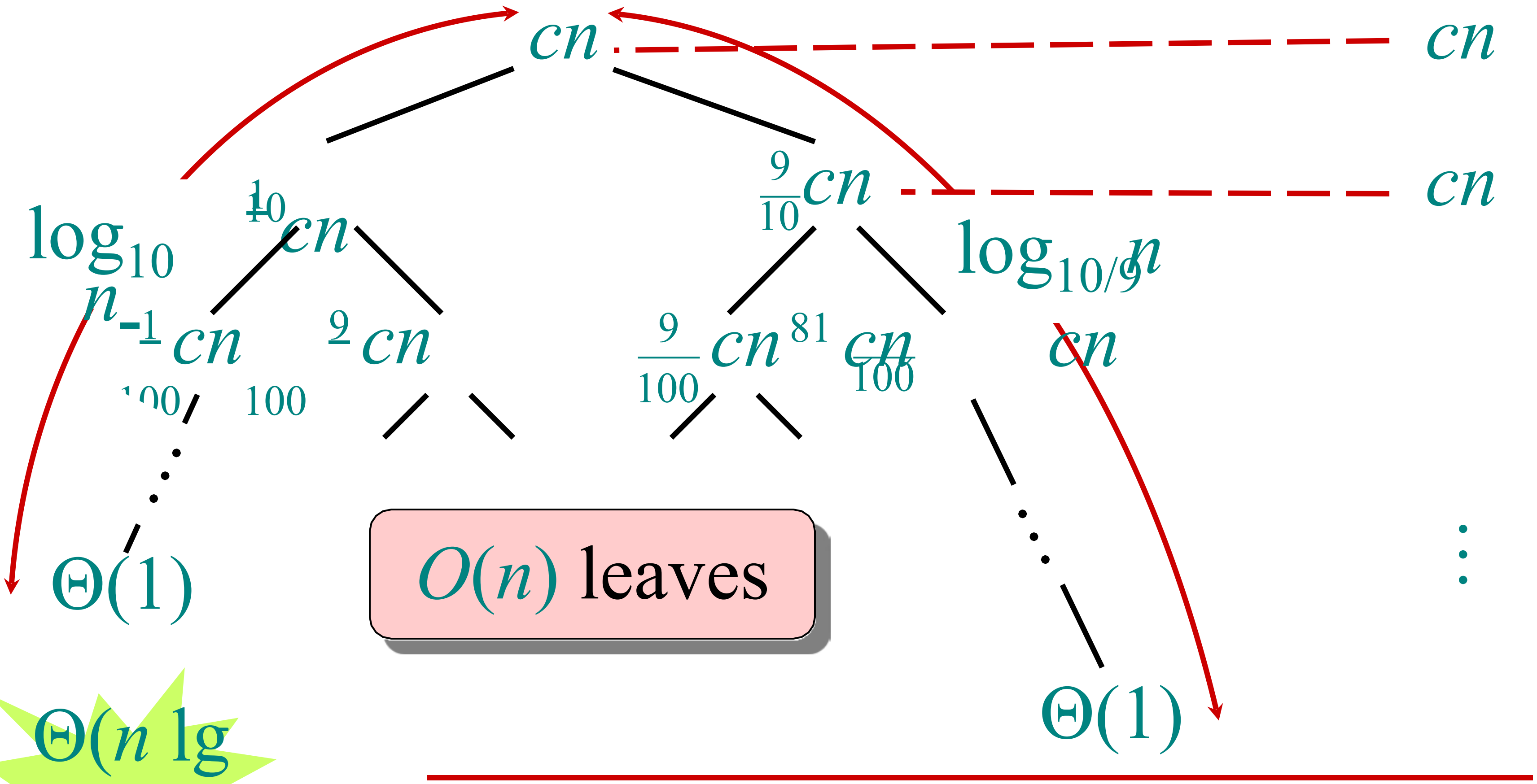


Analysis of "almost-best" case





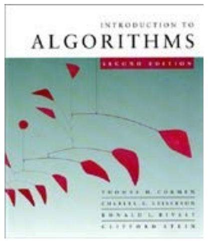
Analysis of “almost-best” case



$\Theta(n \lg n)$

Lucky!

$cn \log_{10} n \leq T(n) \leq cn \log_{10/9} n + O(n)$



Quicksort in practice

- Quicksort is a great general-purpose sorting algorithm.
- Quicksort is typically over twice as fast as merge sort.
- Quicksort can benefit substantially from ***code tuning***.
- Quicksort behaves well even with caching and virtual memory.

Quicksort / partition-exchange sort

Quick sort is a divide and conquer algorithm.

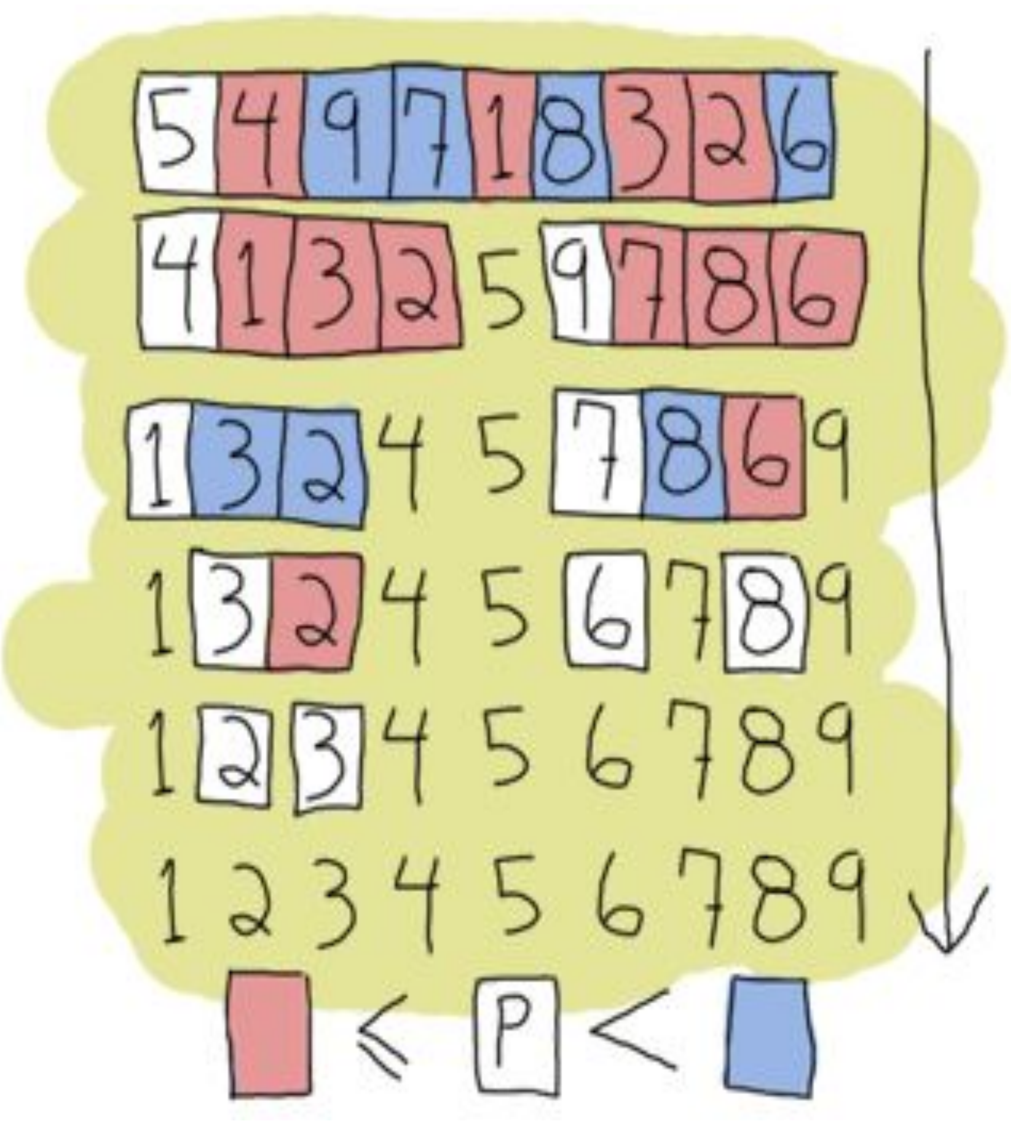
Quick sort first divides a large list into two smaller sub-lists: the low elements and the high elements. Quick sort can then recursively sort the sub-lists.

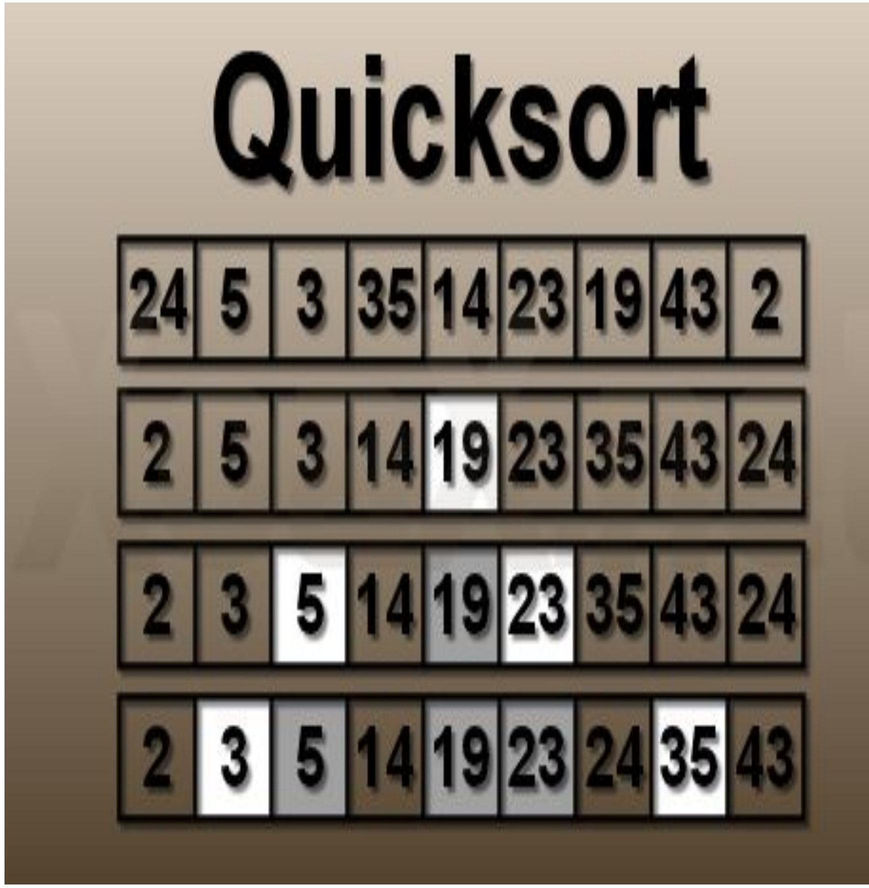
The steps are:

1. Pick an element, called a pivot, from the list.
2. Reorder the list so that all elements with values less than the pivot come before the pivot, while all elements with values greater than the pivot come after it (equal values can go either way). After this partitioning, the pivot is in its final position. This is called the partition operation.
3. Recursively apply the above steps to the sub-list of elements with smaller values and separately the sub-list of elements with greater values.

The base case of the recursion are lists of size zero or one, which never need to be sorted.

Quick Sort


5 4 9 7 1 8 3 2 6
4 1 3 2 5 9 7 8 6
1 3 2 4 5 7 8 6 9
1 3 2 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9
≤ P <


Quicksort
24 5 3 35 14 23 19 43 2
2 5 3 14 19 23 35 43 24
2 3 5 14 19 23 35 43 24
2 3 5 14 19 23 24 35 43

```
44 33 11 55 77 90 40 60 99 22 88
```

Let **44** be the **Pivot** element and scanning done from right to left

Comparing **44** to the right-side elements, and if right-side elements are **smaller** than **44**, then swap it. As **22** is smaller than **44** so swap them.

```
  22    33    11    55    77    90    40    60    99    44    88
```

Now comparing **44** to the left side element and the element must be **greater** than 44 then swap them. As **55** are greater than **44** so swap them.

```
22    33    11    44    77    90    40    60    99    55    88
```

Recursively, repeating steps 1 & steps 2 until we get two lists one left from pivot element **44** & one right from pivot element.

```
22    33    11    40    77    90    44    60    99    55    88
```

Swap with 77:

```
22    33    11    40    44    90    77    60    99    55    88
```

Quick Sort

```
void quickSort(int a[], int first, int last);
int pivot(int a[], int first, int last);
void swap(int& a, int& b);
Void main()
{
  int test[] = { 7, -13, 1, 3, 10, 5, 2, 4 };
  int N = sizeof(test)/sizeof(int);
  quickSort(test, 0, N-1);
}
void swap(int& a, int& b) {
  int temp = a;
  a = b;
  b = temp;
}
```

```
void quickSort( int a[], int first, int last) {
  int pivotElement;
  if(first < last)  {
    pivotElement = pivot(a, first, last);
    quickSort(a, first, pivotElement-1);
    quickSort(a, pivotElement+1, last); }
}
int pivot(int a[], int first, int last){
  int p = first;
  int pivotElement = a[first];
  for(int i = first+1 ; i <= last ; i++){
    if(a[i] <= pivotElement){
      p++;
      swap(a[i], a[p]);
    }
  }
  swap(a[p], a[first]);
  return p;
}
```

```
int pivot(int a[], int first, int last){
  int p = first;
  int pivotElement = a[first];
  for(int i = first+1 ; i <= last ; i++){
    if(a[i] <= pivotElement){
      p++;
      swap(a[i], a[p]);
    }
  }
  swap(a[p], a[first]);
  return p;
}
```