24/1/2025

# Linear Search.

## a) Find a given value $v$ in an array $A$

Input : A sequence of $n$ numbers

$A = \langle a_1, a_2, \ldots, a_n \rangle$, and a value $v$

output : An index $i$ s.t. $v = A[i]$, or the special value NIL if $v$ does not appear in $A$

```
Linear-Search (A, v)
i = NIL
for j = 1 to A.length do
    if A[j] = v then
        i = j
        return i
    end if
end for
return i
```

$T(n) = O(n)$

# Linear Search

## b) Find minimum element in the given array A

Input: A seq. of n numbers $A = \langle a_1, a_2, \ldots, a_n \rangle$.

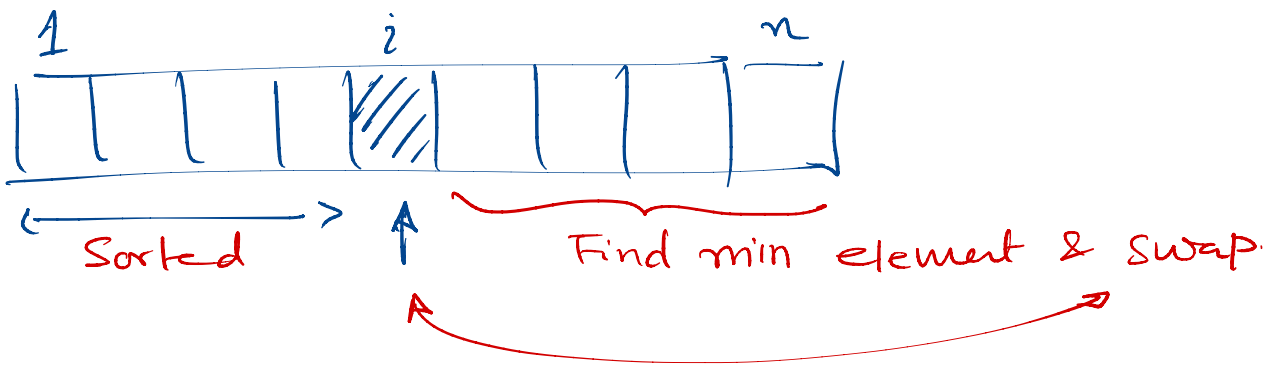
Output: The index i of the minimum elemet of A

```
Find-min (A)
i = 1
for j = 2 to A.length
    if A[j] < A[i]
        i = j
    end if
end for
return i
```

$T(n) = O(n)$

# Selection Sort

Sort n numbers by iteratively selecty the minimum in the subarray and swapping it with the first element of the subarray.



```
Selection_Sort (A)

for i = 1 to n-1 do
    min = i
    for j = i+1 to n do
        // find the index of the ith smallest element
        if A[j] < A[min]
            min = j
        end if
    end for
    swap A[min] with A[i]
end for
```

- Running time $T(n) = \sum_{i=1}^{n-1} (n-i) = n(n-1) - \sum_{i=1}^{n-1} i$

$= n(n-1) - n(n-1)/2$

$= \frac{n(n-1)}{2} = \frac{n^2-n}{2} = \underline{\Theta(n^2)}$

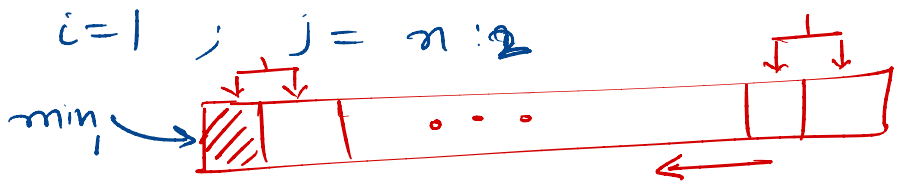
# Bubble Sort

- Popular but inefficient sorting algorithm
- Works by repeatedly swapping adjacent elements that are out of order

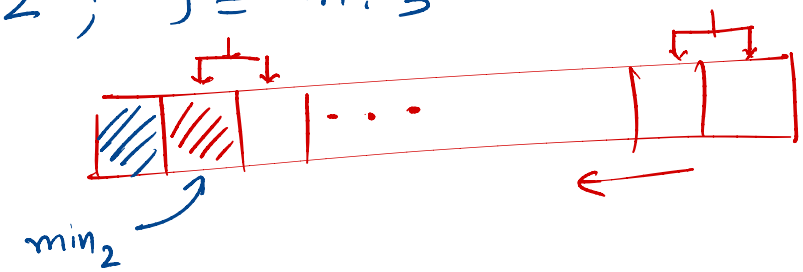
**BubbleSort (A)**

```
for i = 1 to A.length - 1
    for j = A.length down to (i+1)
        if A[j] < A[j-1]
            exchange A[j] with A[j-1]
        end if
    end for
end for
```

ex. Run: $i = 1$ ; $j = n : 2$



$i = 2$ ; $j = n : 3$

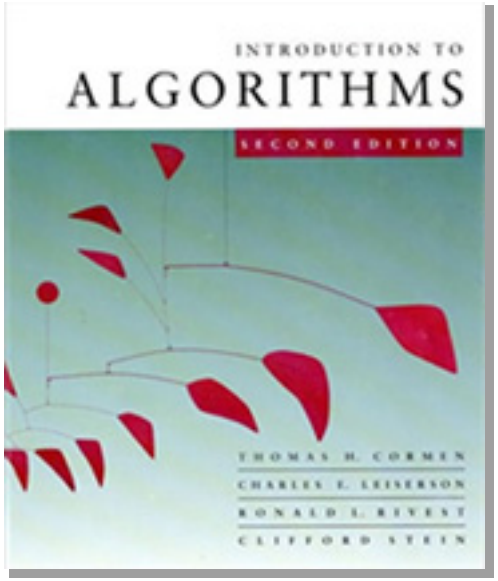



**Loop invariant**: At the start of each iteration, the subarray $A[1 : i-1]$ contains the $(i-1)$ smallest elements in sorted order.

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

# *Introduction to Algorithms*

## 6.046J/18.401J

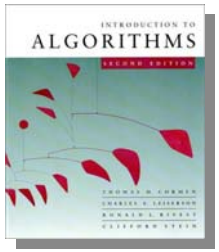



## LECTURE 3

**Divide and Conquer**

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

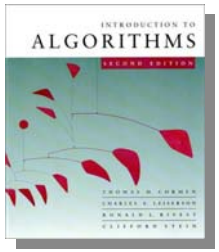
**Prof. Erik D. Demaine**




# 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.

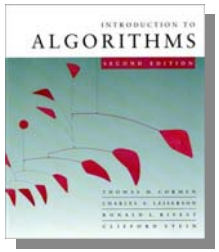
  




# Merge sort

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




# Merge sort

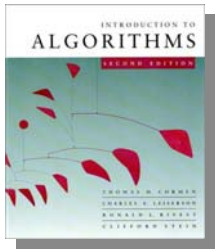
1. ***Divide:*** Trivial.
2. ***Conquer:*** Recursively sort 2 subarrays.
3. ***Combine:*** Linear-time merge.

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

*# subproblems*

*subproblem size*

*work dividing and combining*



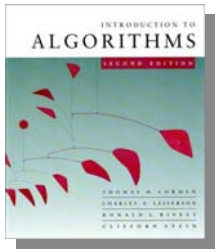

# Master theorem (reprise)

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

**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 $\varepsilon > 0$
$\Rightarrow T(n) = \Theta(n^{\log_b a})$ .

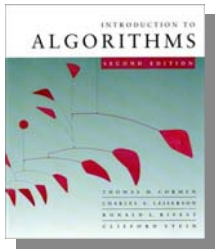
**CASE 2**: $f(n) = \Theta(n^{\log_b a} \lg^k n)$, constant $k \geq 0$
$\Rightarrow T(n) = \Theta(n^{\log_b a} \lg^{k+1} n)$ .

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

$f(n)$

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

***Merge sort:*** $a = 2$, $b = 2$ $\Rightarrow$ $n^{\log_b a} = n^{\log_2 2} = n$
$\Rightarrow$ CASE 2 ($k = 0$) $\Rightarrow$ $T(n) = \Theta(n \lg n)$ .

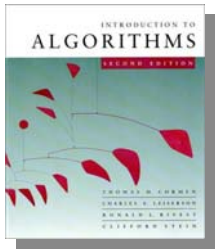



# Binary search

Find an element in a sorted array:

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




# Binary search

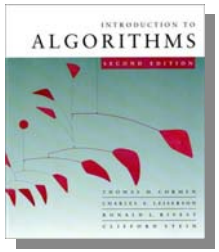
Find an element in a sorted array:

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

***Example:*** Find 9

| 3 | 5 | 7 | 8 | 9 | 12 | 15 |
|---|---|---|---|---|---|---|



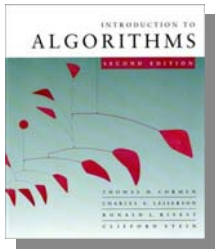

# Binary search

Find an element in a sorted array:

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

***Example:*** Find 9




# Binary search

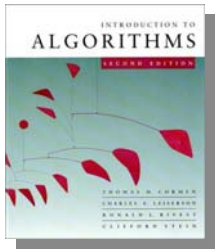
Find an element in a sorted array:

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

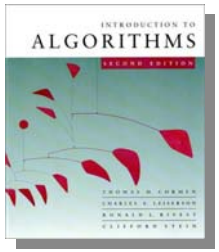
Find an element in a sorted array:

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

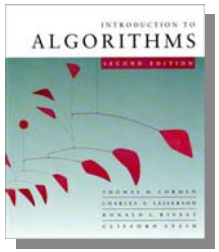
Find an element in a sorted array:

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

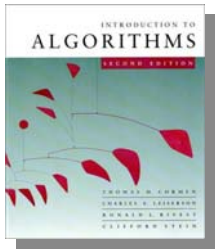
Find an element in a sorted array:

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*

$T(n) = aT(n/b) + f(n)$ : $f(n) = \Theta(1)$

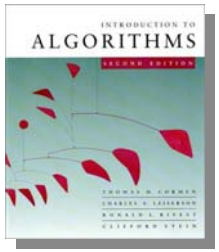
$n^{\log_b a}$ $a=1$ $b=2$

$n^{\log_2 1} = n^0 = 1$

2nd Case of master th.

$T(n) = \Theta(1 \cdot \lg^1 n)$

$= \Theta(\lg n)$ $k=0$




# Recurrence for binary search

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

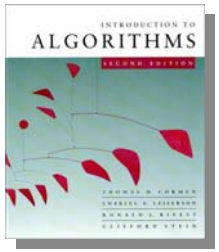
*# subproblems*

*subproblem size*

*work dividing and combining*

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

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

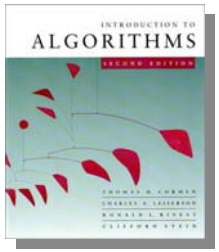



# Powering a number

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

**Naive algorithm:** $\Theta(n)$.




# Powering a number

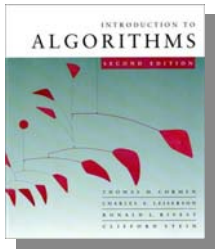
**Problem:** Compute $a^n$, where $n \in \mathbb{N}$.

**Naive algorithm:** $\Theta(n)$.

**Divide-and-conquer algorithm:**

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




# Powering a number

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

**Naive algorithm:** $\Theta(n)$.

**Divide-and-conquer algorithm:**

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

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