Introduction to

Algorithm Design and Analysis

[2] Asymptotics

Yu Huang
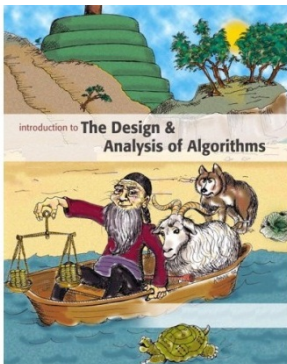
http://cs.nju.edu.cn/yuhuang

Institute of Computer Software

Nanjing University

In the Last Class…

- **Algorithm – the spirit of computing**
  o Model of computation

- **Algorithm design and analysis**
  o Design
    - Correctness proof by induction
  o Analysis
    - Worst-case / average-case complexity

Asymptotic Behavior

- **Asymptotic growth rate of functions**
  - Basic idea

- **Key notations**
  - O, Ω, Θ
  - o, ω

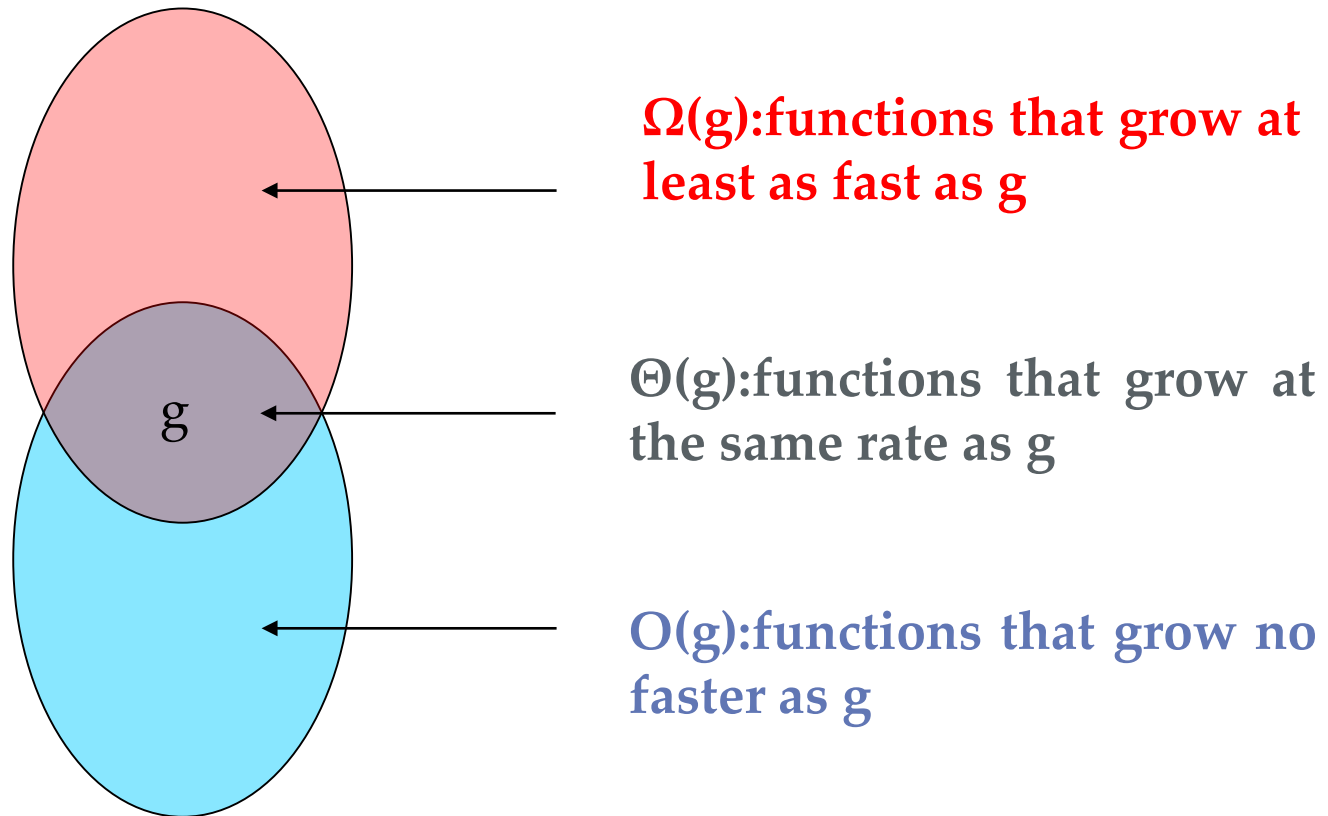- **Brute force enumeration**
  - By iteration
  - By recursion

How to Compare Two Algorithms

- **Algorithm analysis, with *simplifications***
  - Measuring the cost by the number of critical operations
  - Large input size only
    - Only the leading term in $f(n)$ is considered
    - Constant coefficients are ignored

- **Capturing the essential part in the cost in a mathematical way**
  - Asymptotic growth rate of $f(n)$

Relative Growth Rate

Ω(g):functions that grow at least as fast as g

Θ(g):functions that grow at the same rate as g

O(g):functions that grow no faster as g

g

"Big Oh"

- **Basic idea** $f(n) \in O(g(n))$
  - For sufficiently large input size, g(n) is an upper bound for f(n)

- **Definition – "ε − N"**
  - Giving g: N→R⁺, then O(g) is the set of f:N→R⁺, such that for some c∈R⁺ and some n_0∈N, f(n)≤cg(n) for all n≥n_0

- **Definition – "$lim_{n \to \infty}$"**
  - $f \in O(g)$ if $\lim_{n \to \infty} \frac{f(n)}{g(n)} = c < \infty$

Example

- **Let f(n)=n², g(n)=nlogn, then:** ~~L'Hospital's rule~~

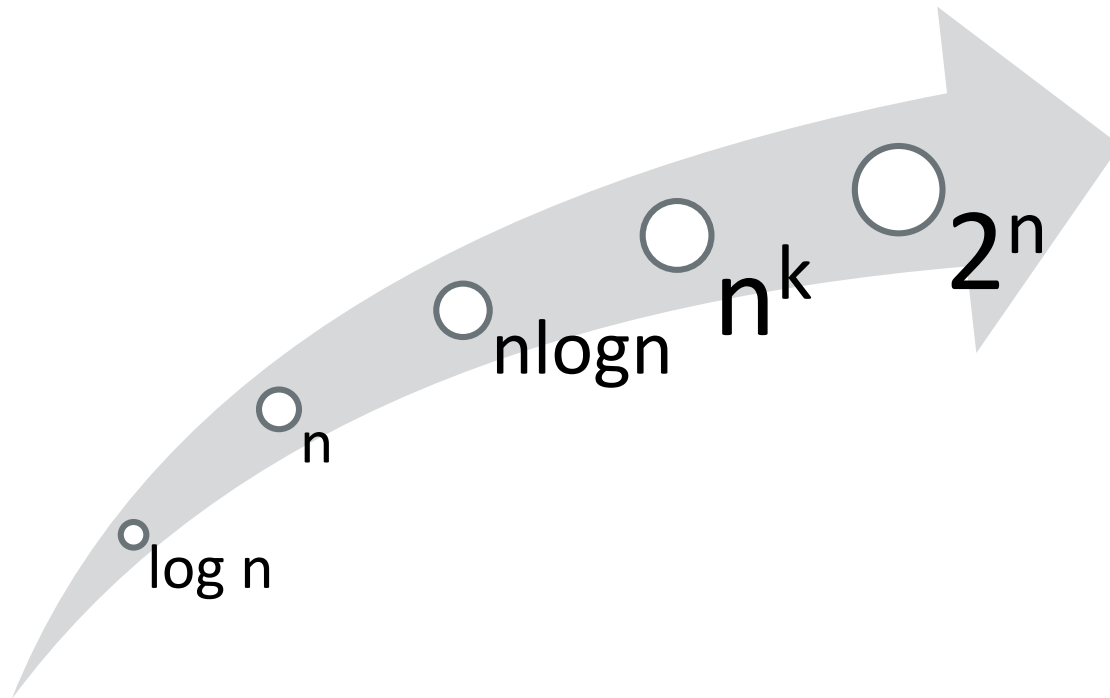 - f∉O(g), since

  $$\lim_{n\to\infty} \frac{n^2}{n\log n} = \lim_{n\to\infty} \frac{n}{\log n} = \lim_{n\to\infty} \frac{1}{\frac{1}{n\ln 2}} = +\infty$$

  - g∈O(f), since

  $$\lim_{n\to\infty} \frac{n\log n}{n^2} = \lim_{n\to\infty} \frac{\log n}{n} = \lim_{n\to\infty} \frac{1}{n\ln 2} = 0$$

Asymptotic Growth Rate

2^n

n^k

nlogn

n

log n

Asymptotic Order

- **Logarithm** *log n*

  $$\log n \in O(n^\alpha) \text{ for } \textbf{\textcolor{red}{any}} \ \alpha > 0$$

- **Power** n^k

  $$n^k \in O(c^n) \text{ for } \textbf{\textcolor{red}{any}} \ c > 1$$

- **Factorial** *n!*

  $$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \ \textbf{(Stirling's formula)}$$

"Big Ω"

- **Basic idea of** $f(n) \in \Omega(g(n))$
  - ○ Dual of "O"

- **Definition – "ε − N"**
  - ○ Giving g: N→R⁺, then $\Omega(g)$ is the set of f:N→R⁺, such that for some c∈R⁺ and some n_0∈N, f(n)≥cg(n) for all n≥n_0

- **Definition – "$lim_{n\to\infty}$"**
  - ○ $f \in \Omega(g)$ if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c > 0$ (the limit may be ∞)

The Set Θ

- **Basic idea of $f(n) \in \Theta(g(n))$**
  - Roughly the same
  - $\Theta(g) = O(g) \cap \Omega(g)$

- **Definition – "$\varepsilon - N$"**
  - Giving g:N→R⁺, then $\Theta(g)$ is the set of f:N→R⁺, such that for some $c_1, c_2 \in$R⁺ and some $n_0 \in$N,
  $$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n), \text{ for all } n \geq n_0$$

- **Definition – "$lim_{n \to \infty}$"**
  - $f \in \Theta(g)$ if $\lim_{n \to \infty} \frac{f(n)}{g(n)} = c \ (0 < c < \infty)$

Some Empirical Data

algorithm		1	2	3	4
Run time in ns		$1.3n^3$	$10n^2$	$47n\log n$	$48n$
time for size	10^3	$1.3s$	$10ms$	$0.4ms$	$0.05ms$
	10^4	$22m$	$1s$	$6ms$	$0.5ms$
	10^5	$15d$	$1.7m$	$78ms$	$5ms$
	10^6	$41yrs$	$2.8hrs$	$0.94s$	$48ms$
	10^7	$41mill$	$1.7wks$	$11s$	$0.48s$
max Size in time	sec	920	10,000	1.0×10^6	2.1×10^7
	min	3,600	77,000	4.9×10^7	1.3×10^9
	hr	14,000	6.0×10^5	2.4×10^9	7.6×10^{10}
	day	41,000	2.9×10^6	5.0×10^{10}	1.8×10^{12}
time for 10 times size		$\times1000$	$\times100$	$\times10+$	$\times10$

on 400Mhz Pentium II, in C
from: Jon Bentley: *Programming Pearls*

Properties of O, Ω and Θ

- **Transitive property:**
  - If $f \in O(g)$ and $g \in O(h)$, then $f \in O(h)$
- **Symmetric properties**
  - $f \in O(g)$ if and only if $g \in \Omega(f)$
  - $f \in \Theta(g)$ if and only if $g \in \Theta(f)$
- **Order of sum function**
  - $O(f+g) = O(\max(f,g))$

"Little Oh"

- **Basic idea of $f(n) \in o(g(n))$**
  - Non-ignorable gap between f and its upper bound g
- **Definition –"$\varepsilon - N$"**
  - Giving g:N→R⁺, then o(g) is the set of f:N→R⁺, such that for any c∈R⁺ , there exists some n_0∈N,

  $$0 \leq f(n) < cg(n), \text{ for all } n \geq n_0$$

- **Definition – "$lim_{n \to \infty}$"**
 - $f \in o(g)$ if $\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$

"Little ω"

- **Basic idea of** $f(n) \in \omega(g(n))$
  - Dual of "o"

- **Definition – "ε − N"**
  - Giving $g{:}N \rightarrow R^+$, then $\omega(g)$ is the set of $f{:}N \rightarrow R^+$, such that for any $c \in R^+$, there exists some $n_0 \in N$,

  $$0 \leq cg(n) < f(n), \text{ for all } n \geq n_0$$

- **Definition – "$lim_{n \rightarrow \infty}$"**
  - $f \in \omega(g)$ if $\lim\limits_{n \rightarrow \infty} \dfrac{f(n)}{g(n)} = \infty$

Do you know Infinity

- **Mathematical analysis**

  (Differentiation / integration)
  - Early years
  - Firm foundation
    of mathematical analysis

  Cauchy

- **How to talk about *infinity*?**
  - $(\varepsilon - N)$-definition
  - $(\varepsilon - \delta)$-definition
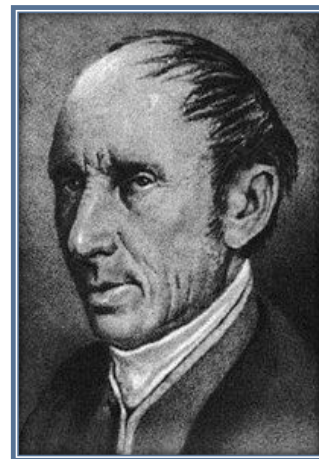
  Weierstrass

Brute Force Enumeration by Iteration

- **Swapping array elements**
  - \<time, space>
    - From $\langle O(n^2), O(1) \rangle$
    - To $\langle O(n), O(n) \rangle$
    - To $\langle O(n), O(1) \rangle$

- **Maximum subsequence sum**
  - Time
    - From $O(n^3)$
    - To $O(n^2)$
    - To $O(n \log n)$
    - To $O(n)$

Swapping Array Elements

- **E.g., 1,2,3,4 | 5,6,7 => 5,6,7,1,2,3,4**
- **Brute force**

	Time	Space
BF1	$O(n^2)$	$O(1)$
BF2	$O(n)$	$O(n)$
Your Task	$O(n)$	$O(1)$

Space sensitive

Time sensitive

- **Your task**
  - Both time and space efficient

Max-sum Subsequence

- **The problem: Given a sequence S of integer, find the largest sum of a consecutive subsequence of S.** (0, if all negative items)

  o An example: -2, 11, -4, 13, -5, -2; the result 20: (11, -4, 13)

A brute-force algorithm:

```
MaxSum = 0;
 for (i = 0; i < N; i++)
   for (j = i; j < N; j++)
   {
     ThisSum = 0;
     for (k = i; k <= j; k++)
     ThisSum += A[k];
     if (ThisSum > MaxSum)
       MaxSum = ThisSum;
   }
 return MaxSum;
```

the sequence

$j=0$ $j=1$ $j=2$ $j=n$-1

$i=0$

$i=1$

k

$i=2$

$i=n$-1
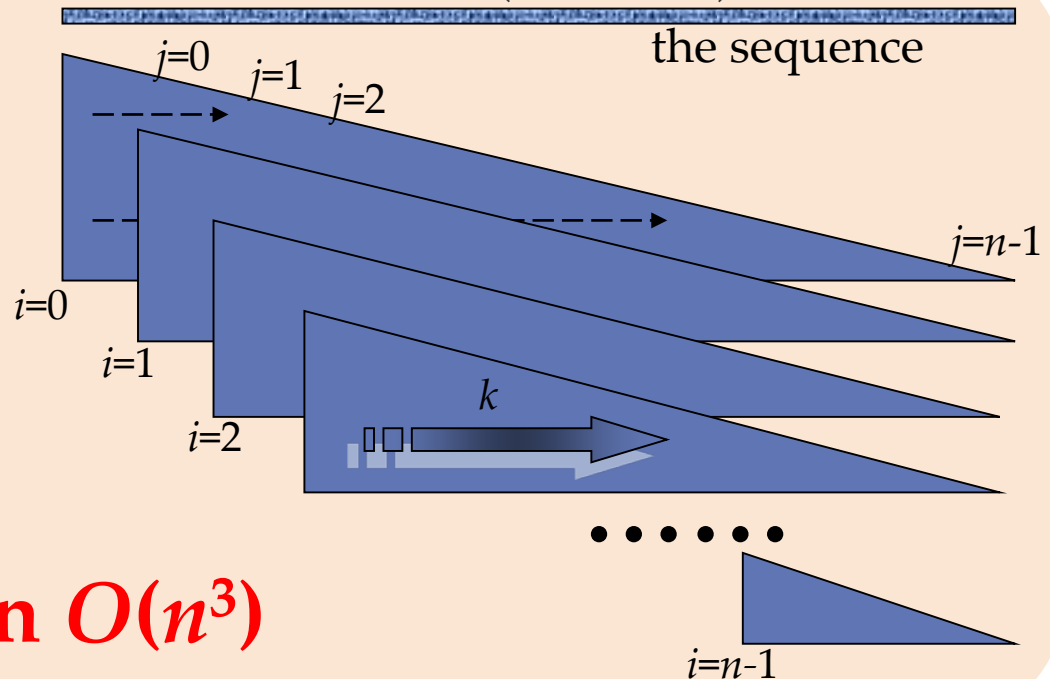
in $O(n^3)$

More Precise Complexity

The total cost is : $\displaystyle\sum_{i=0}^{n-1}\sum_{j=i}^{n-1}\sum_{k=i}^{j}1$

$$\sum_{k=i}^{j}1 = j - i + 1$$

$$\sum_{j=i}^{n-1}(j - i + 1) = 1 + 2 + \dots + (n - i) = \frac{(n-i+1)(n-i)}{2}$$

$$\sum_{i=0}^{n-1}\frac{(n-i+1)(n-i)}{2} = \sum_{i=1}^{n}\frac{(n-i+2)(n-i+1)}{2}$$

$$= \frac{1}{2}\sum_{i=1}^{n}i^2 - (n + \frac{3}{2})\sum_{i=1}^{n}i + \frac{1}{2}(n^2 + 3n + 2)\sum_{i=1}^{n}1$$

$$= \frac{n^3 + 3n^2 + 2n}{6}$$

Decreasing the number of loops

An improved algorithm
```
MaxSum = 0;
  for (i = 0; i < N; i++)
  {
    ThisSum = 0;
    for (j = i; j < N; j++)
    {
      ThisSum += A[j];
      if (ThisSum > MaxSum)
        MaxSum = ThisSum;
    }
  }
  return MaxSum;
```

the sequence

$i=0$
$i=1$
$i=2$
j

in $O(n^2)$

$i=n-1$

Power of Divide and Conquer

Part 1

Part 2
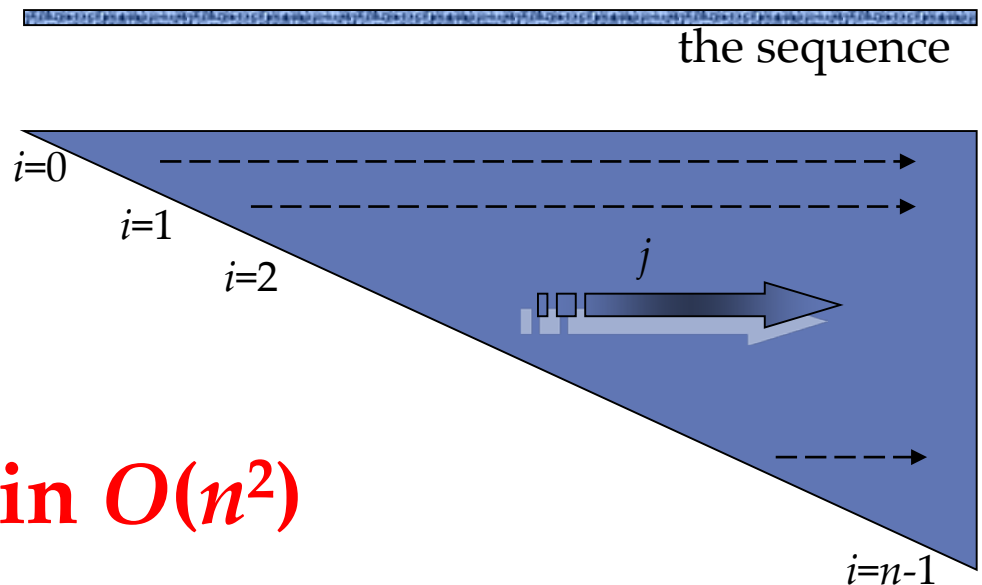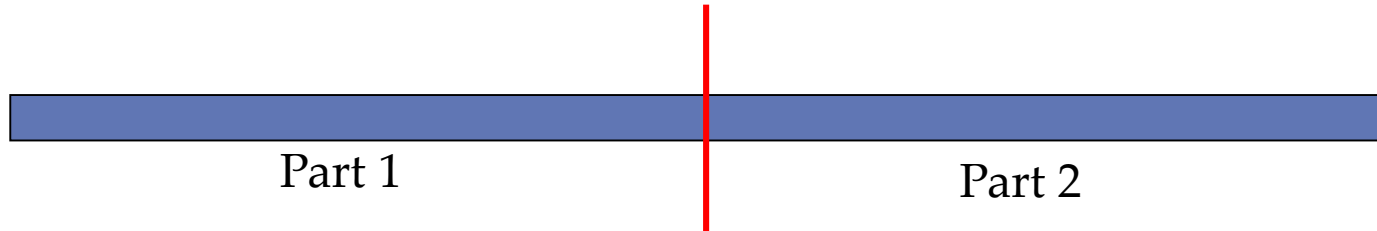
the sub with largest sum may be in:

recursion

Part 1

Part 2

or:

The largest is the result

Part 1

Part 2
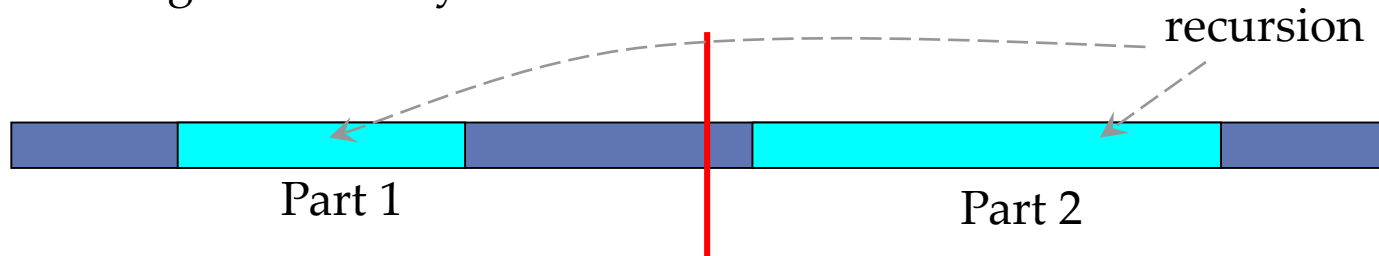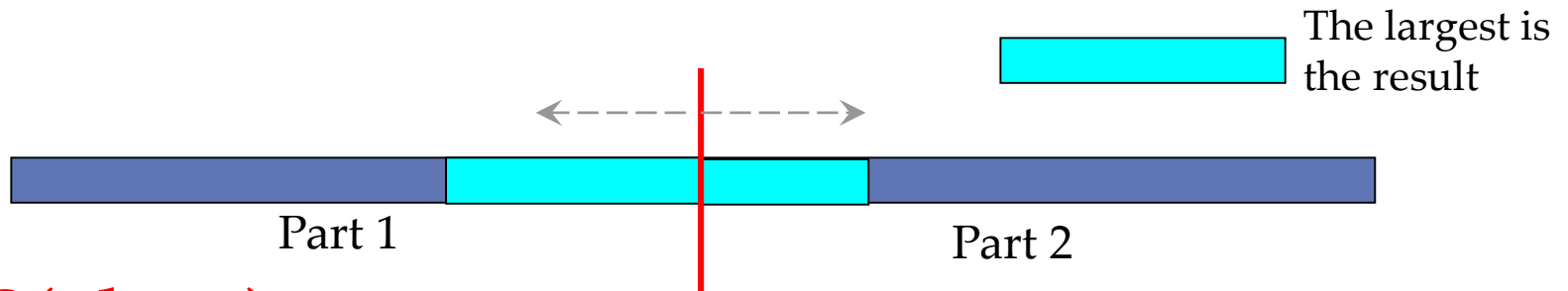
in $O(n\log n)$

Power of
Divide and Conquer

```
Center = (Left + Right) / 2;
 MaxLeftSum = MaxSubSum(A, Left, Center);  MaxRightSum = MaxSubSum(A, Center + 1,
Right);


 MaxLeftBorderSum = 0; LeftBorderSum = 0;
 for (i = Center; i >= Left; i--)
 {
  LeftBorderSum += A[i];
  if (LeftBorderSum > MaxLeftBorderSum)    MaxLeftBorderSum = LeftBorderSum;
 }


 MaxRightBorderSum = 0; RightBorderSum = 0;
 for (i = Center + 1; i <= Right; i++)
 {
  RightBorderSum += A[i];
  if (RightBorderSum > MaxRightBorderSum)  MaxRightBorderSum = RightBorderSum;
 }
 return Max3(MaxLeftSum, MaxRightSum,
     MaxLeftBorderSum + MaxRightBorderSum);
```

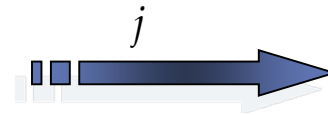Note: this is the core part of the procedure, with base case and wrap omitted.

A Linear Algorithm

First scan the array to eliminate the case of "all negative integers"
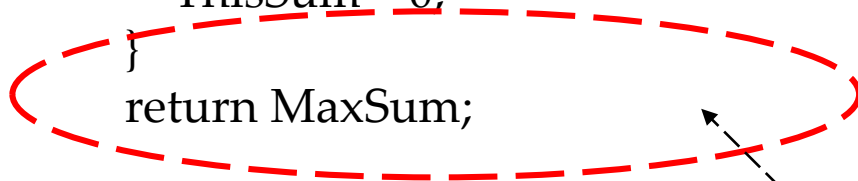
```
ThisSum = MaxSum = 0;
 for (j = 0; j < N; j++)
 {
   ThisSum += A[j];
   if (ThisSum > MaxSum)
     MaxSum = ThisSum;
   else if (ThisSum < 0)
     ThisSum = 0;
 }
 return MaxSum;
```

the sequence

j

This is an example of "online algorithm"

Negative item or subsequence cannot be a prefix of the subsequence we want.
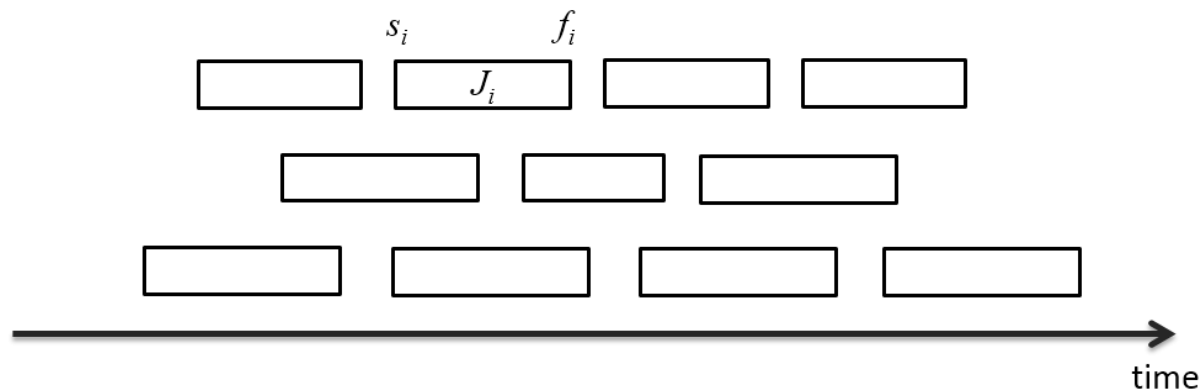
in $O(n)$

Brute Force Enumeration by Recursion

- **Job scheduling**
  - Problem definition
  - Brute force recursion
  - Further improvements

- **Matrix chain multiplication**
  - Problem definition
  - Brute force recursion(s)
  - Further improvements

Job Scheduling

- **Jobs:** $J_i = [s_i, j_i)$
- **Max number of compatible jobs**
- **Further improvements**
  - Dynamic programming (L16)
  - Greedy algorithms (L14)

Matrix Chain Multiplication

- **The task:**

  Find the product: $A_1 \times A_2 \times \ldots \times A_{n-1} \times A_n$

  A_i is 2-dimensional array of different legal size

- **The Challenge:**
  - Matrix multiplication is associative
  - Different computing order results in great difference in the number of operations

- **The problem:**
  - Which is the best computing order

Cost of Matrix Multiplication

Let $C = A_{p \times q} \times$

An example: $A_1 \times A_2 \times A_3 \times A_4$
$\quad\quad\quad\quad 30 \times 1 \quad 1 \times 40 \quad 40 \times 10 \quad 10 \times 25$

$((A_1 \times A_2) \times A_3) \times A_4$: 20700 multiplications
$A_1 \times (A_2 \times (A_3 \times A_4))$: 11750
$(A_1 \times A_2) \times (A_3 \times A_4)$: 41200
$A_1 \times ((A_2 \times A_3) \times A_4)$: 1400

$$c_{i,j} = \sum_{k=1}^{q} a_{ik} b_{kj}$$

There are q multiplication

C has $p \times r$ elements as $c_{i,j}$

So, pqr multiplications altogether

Solutions

- **Brute force recursion (L16)**
  - BF1
  - BF2

- **Dynamic programming (L16)**
  - Based on brute force recursion 2

Thank you!

Q & A

Yu Huang

yuhuang@nju.edu.cn

http://cs.nju.edu.cn/yuhuang