

# Introduction to Algorithm Design and Analysis

[02] Asymptotics

Jingwei Xu

<https://ics.nju.edu.cn/~xjw/>

Institute of Computer Software  
Nanjing University

# In the Last Class...

- **Algorithm - the spirit of computing**
  - Model of computation
- **Algorithm design and analysis**
  - Design
    - Correctness proof by induction
  - Analysis
    - Worst-case / average-case complexity

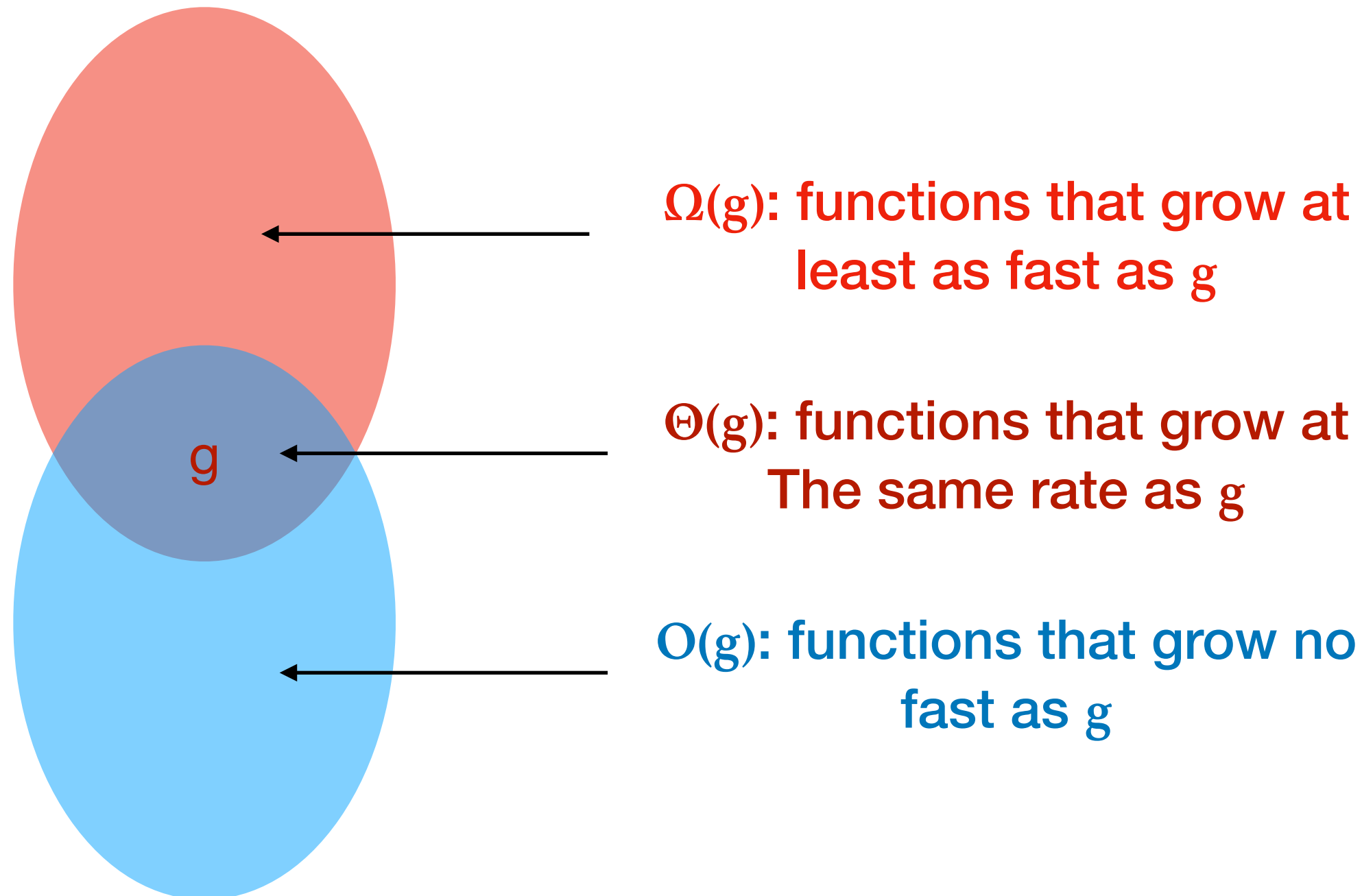
# Asymptotic Behavior

- Asymptotic growth rate of functions
  - Basic idea
- Key notations
  - $O$ ,  $\Omega$ ,  $\Theta$
  - $o$ ,  $\omega$
- 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



# “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 - “ $\epsilon$ -N”**
  - Giving  $g: N \rightarrow R^+$ , then  $O(g)$  is the set of  $f: N \rightarrow R^+$ , such that for some  $c \in R^+$  and some  $n_0 \in N$ ,  $f(n) \leq cg(n)$  for all  $n \geq n_0$
- **Definition - “ $\lim_{n \rightarrow \infty}$ ”**
  - $f \in O(g)$  if  $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = c < \infty$

**The limit may not exist, though it usually does.**

# Example

- Let  $f(n)=n^2$ ,  $g(n)=n\log n$ , then:

L'Hospital's  
rule

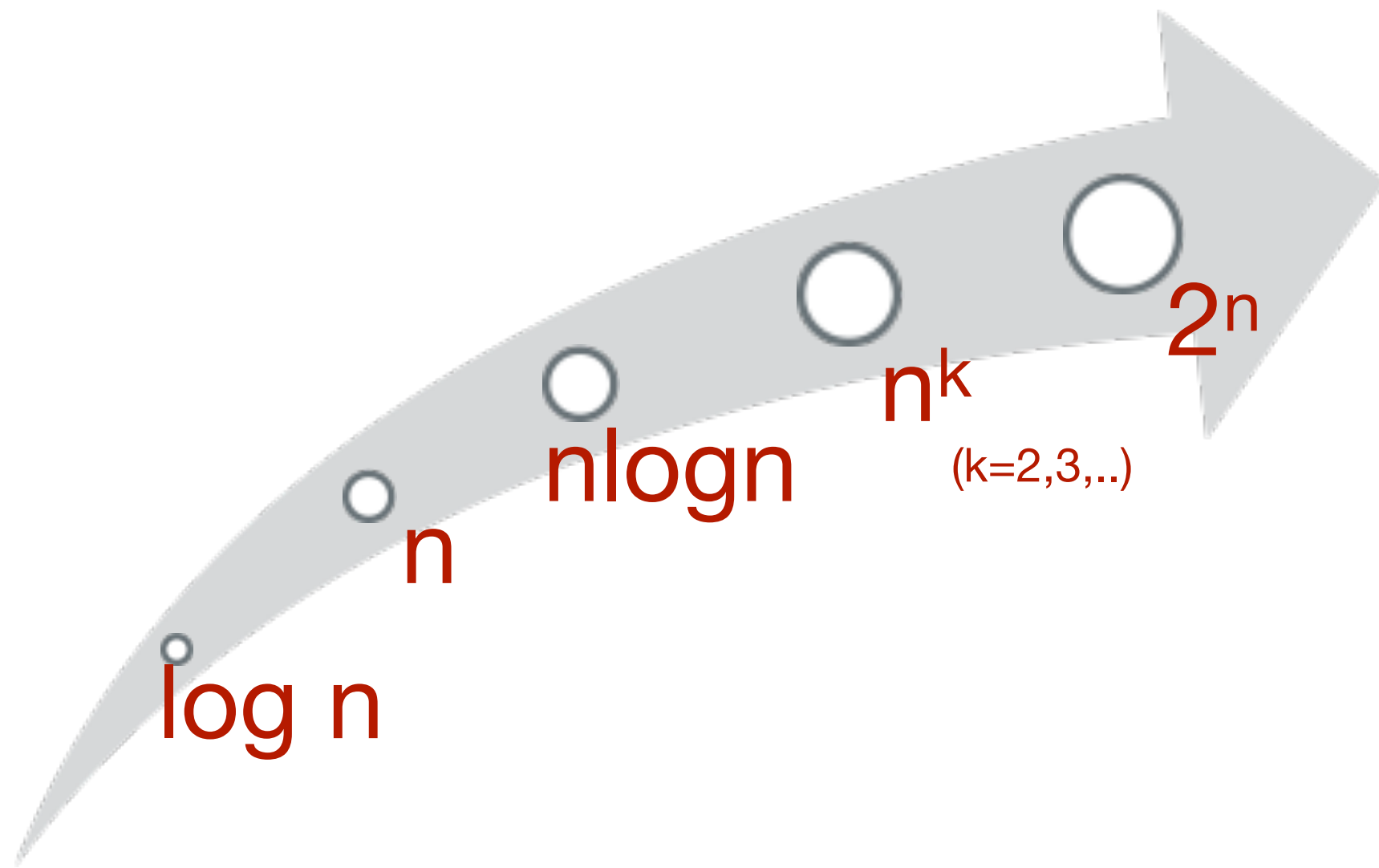
- $f \notin O(g)$ , since

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

- $g \in O(f)$ , since

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

# Asymptotic Growth Rate





# Asymptotic Order

- Logarithm  $\log n$

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

- Power  $n^k$

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

- Factorial  $n!$

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

# “Big $\Omega$ ”

- **Basic idea  $f(n) \in \Omega(g(n))$** 
  - Dual of “O”
- **Definition - “ $\varepsilon$ -N”**
  - Giving  $g: \mathbb{N} \rightarrow \mathbb{R}^+$ , then  $\Omega(g)$  is the set of  $f: \mathbb{N} \rightarrow \mathbb{R}^+$ , such that for some  $c \in \mathbb{R}^+$  and some  $n_0 \in \mathbb{N}$ ,  $f(n) \geq cg(n)$  for all  $n \geq n_0$
- **Definition - “ $\lim_{n \rightarrow \infty}$ ”**
  - $f \in \Omega(g)$  if  $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = c > 0$       the limit may be  $\infty$

# The Set $\Theta$

- **Basic idea  $f(n) \in \Theta(g(n))$**

- Roughly the same
- $\Theta(g) = O(g) \cap \Omega(g)$

- **Definition - “ $\epsilon$ -N”**

- Giving  $g: \mathbb{N} \rightarrow \mathbb{R}^+$ , then  $\Theta(g)$  is the set of  $f: \mathbb{N} \rightarrow \mathbb{R}^+$ , such that for some  $c_1, c_2 \in \mathbb{R}^+$  and some  $n_0 \in \mathbb{N}$ ,  $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ , for all  $n \geq n_0$

- **Definition - “ $\lim_{n \rightarrow \infty}$ ”**

- $f(n) \in \Theta(g)$  if  $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = c (0 < c < \infty)$

# Some Empirical Data

algorithm		1	2	3	4
Run time in <i>ns</i>		$1.3n^3$	$10n^2$	$47n\log n$	$48n$
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	$1.0 \times 10^6$	$2.1 \times 10^7$
	min	3,600	77,000	$4.9 \times 10^7$	$1.3 \times 10^9$
	hr	14,000	$6.0 \times 10^5$	$2.4 \times 10^9$	$7.6 \times 10^{10}$
	day	41,000	$2.9 \times 10^6$	$5.0 \times 10^{10}$	$1.8 \times 10^{12}$
time for 10 times size		$\times 1000$	$\times 100$	$\times 10+$	$\times 10$

on 400Mhz Pentium II, in C

from: Jon Bentley: *Programming Pearls*

# Properties of $O$ , $\Omega$ and $\Theta$

- **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  $f(n) \in o(g(n))$** 
  - Non-ignorable gap between  $f$  and its upper bound  $g$
- **Definition - “ $\varepsilon$ - $N$ ”**
  - Giving  $g: \mathbb{N} \rightarrow \mathbb{R}^+$ , then  $o(g)$  is the set of  $f: \mathbb{N} \rightarrow \mathbb{R}^+$ , such that for any  $c \in \mathbb{R}^+$ , there exists some  $n_0 \in \mathbb{N}$ ,  $0 < f(n) < cg(n)$ , for all  $n \geq n_0$
- **Definition - “ $\lim_{n \rightarrow \infty}$ ”**
  - $f \in o(g)$  if  $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$

# “Little $\omega$ ”

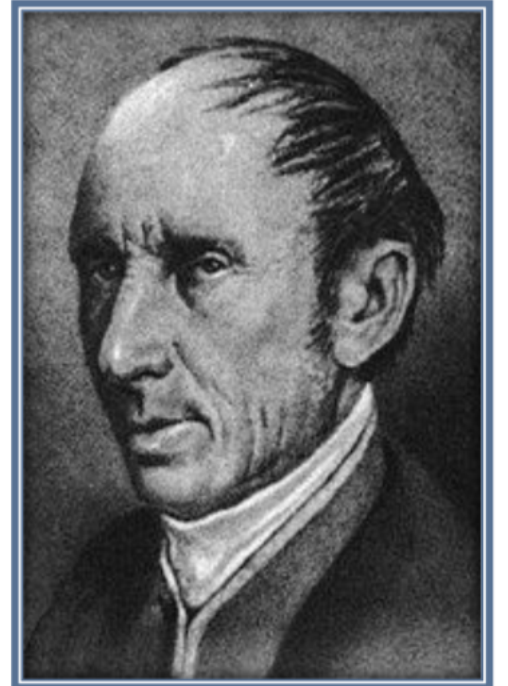
- **Basic idea  $f(n) \in \omega(g(n))$** 
  - Dual of “o”
- **Definition - “ $\varepsilon$ -N”**
  - Giving  $g: \mathbb{N} \rightarrow \mathbb{R}^+$ , then  $\omega(g)$  is the set of  $f: \mathbb{N} \rightarrow \mathbb{R}^+$ , such that for any  $c \in \mathbb{R}^+$ , there exists some  $n_0 \in \mathbb{N}$ ,  
 $0 \leq cg(n) < f(n)$ , for all  $n \geq n_0$
- **Definition - “ $\lim_{n \rightarrow \infty}$ ”**
  - $f \in \omega(g)$  if  $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \infty$

# Do You Know Infinity

- **Mathematical analysis**

- Firm foundation

Cauchy



- **How to talk about infinity?**

- $(\epsilon-N)$ -definition
- $(\epsilon-\delta)$ -definition

Weierstrass





# Brute Force Enumeration by Iteration

- **Swapping array elements**
  - $\langle \text{time, space} \rangle$ 
    - 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

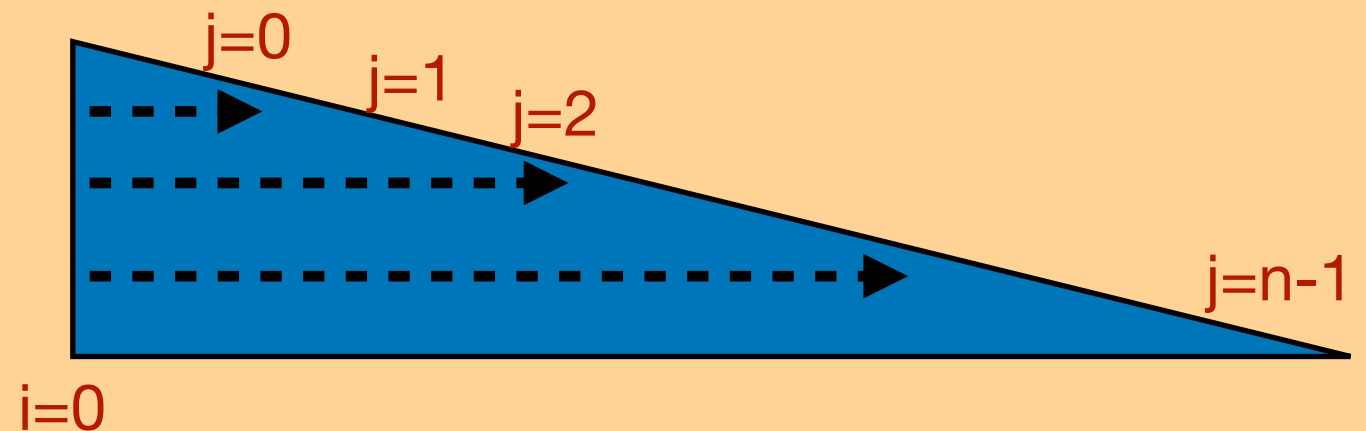
# Max-sum subsequence

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

An example: 2, 11, -4, 13, -5, -2; the result 22: (2, 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;
```



# 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)

An example: 2, 11, -4, 13, -5, -2; the result 22: (2, 11, -4, 13)

A brute-force algorithm:

MaxSum = 0;

for( $i = 0$ ;  $i < N$ ;  $i++$ )

for( $j = i$ ;  $j < N$ ;  $j++$ ){

    ThisSum = 0;

    for( $k = i$ ;  $k \leq j$ ;  $k++$ )

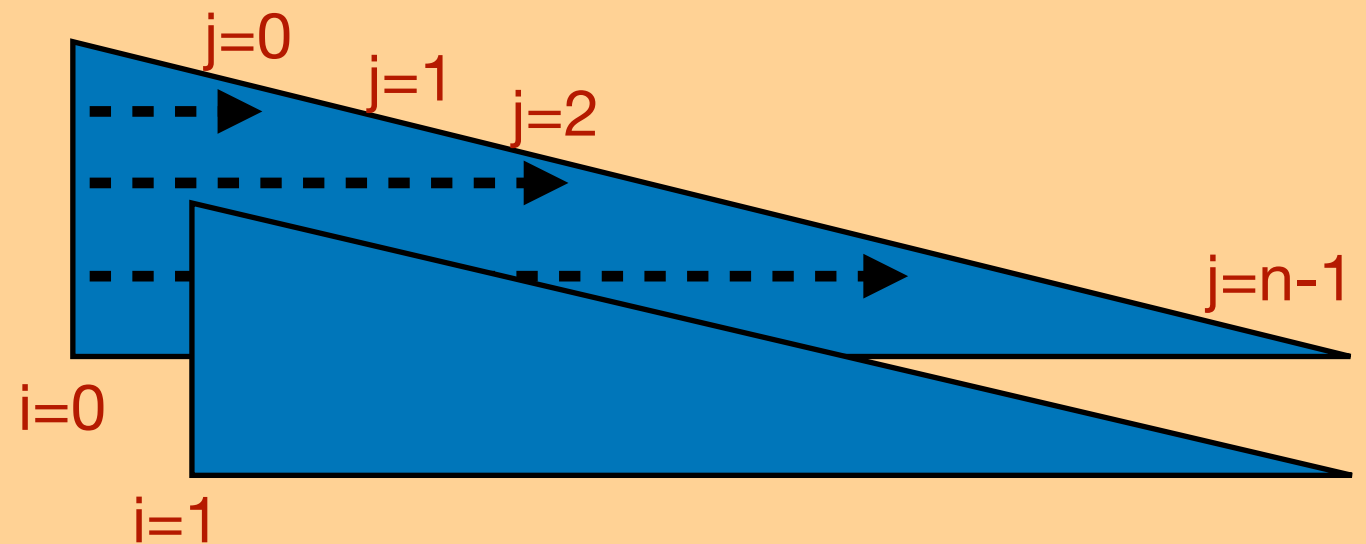
        ThisSum +=  $A[k]$ ;

    if(ThisSum > MaxSum)

        MaxSum = ThisSum;

}

return MaxSum;



# 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)

An example: 2, 11, -4, 13, -5, -2; the result 22: (2, 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++)
```

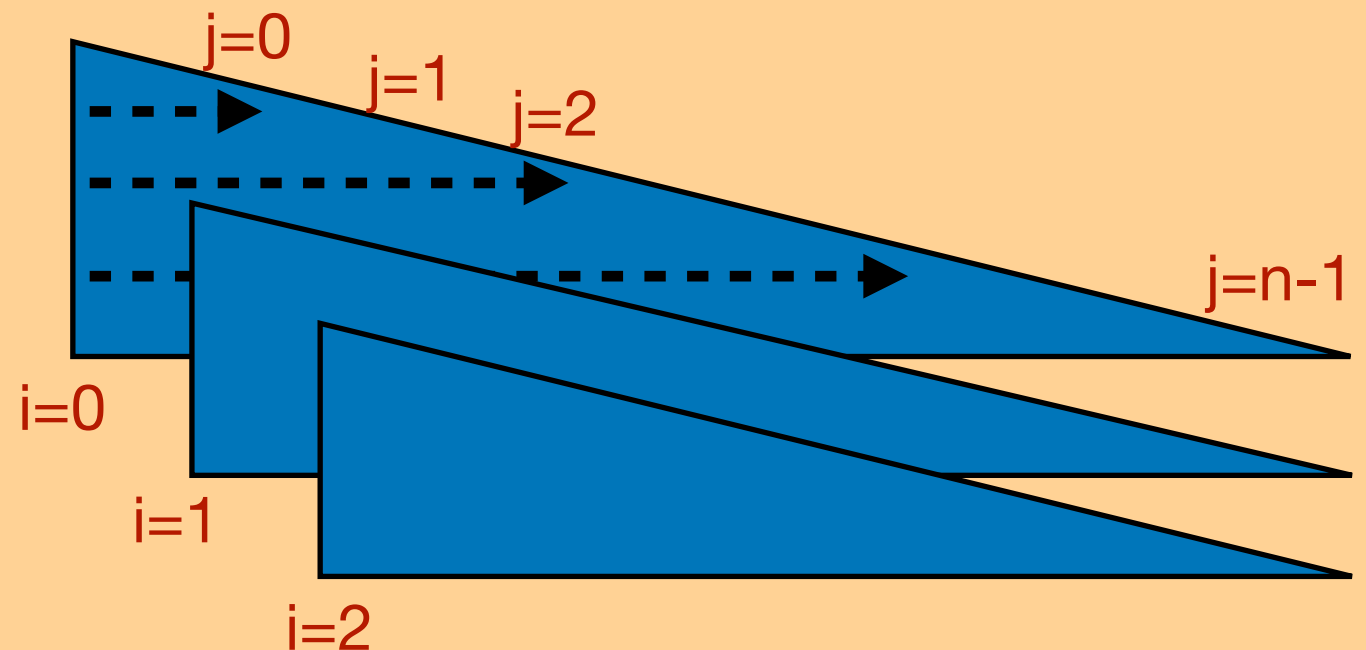
```
      ThisSum += A[k];
```

```
    if(ThisSum > MaxSum)
```

```
      MaxSum = ThisSum;
```

```
  }
```

```
return MaxSum;
```



# 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)

An example: 2, 11, -4, 13, -5, -2; the result 22: (2, 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++)
```

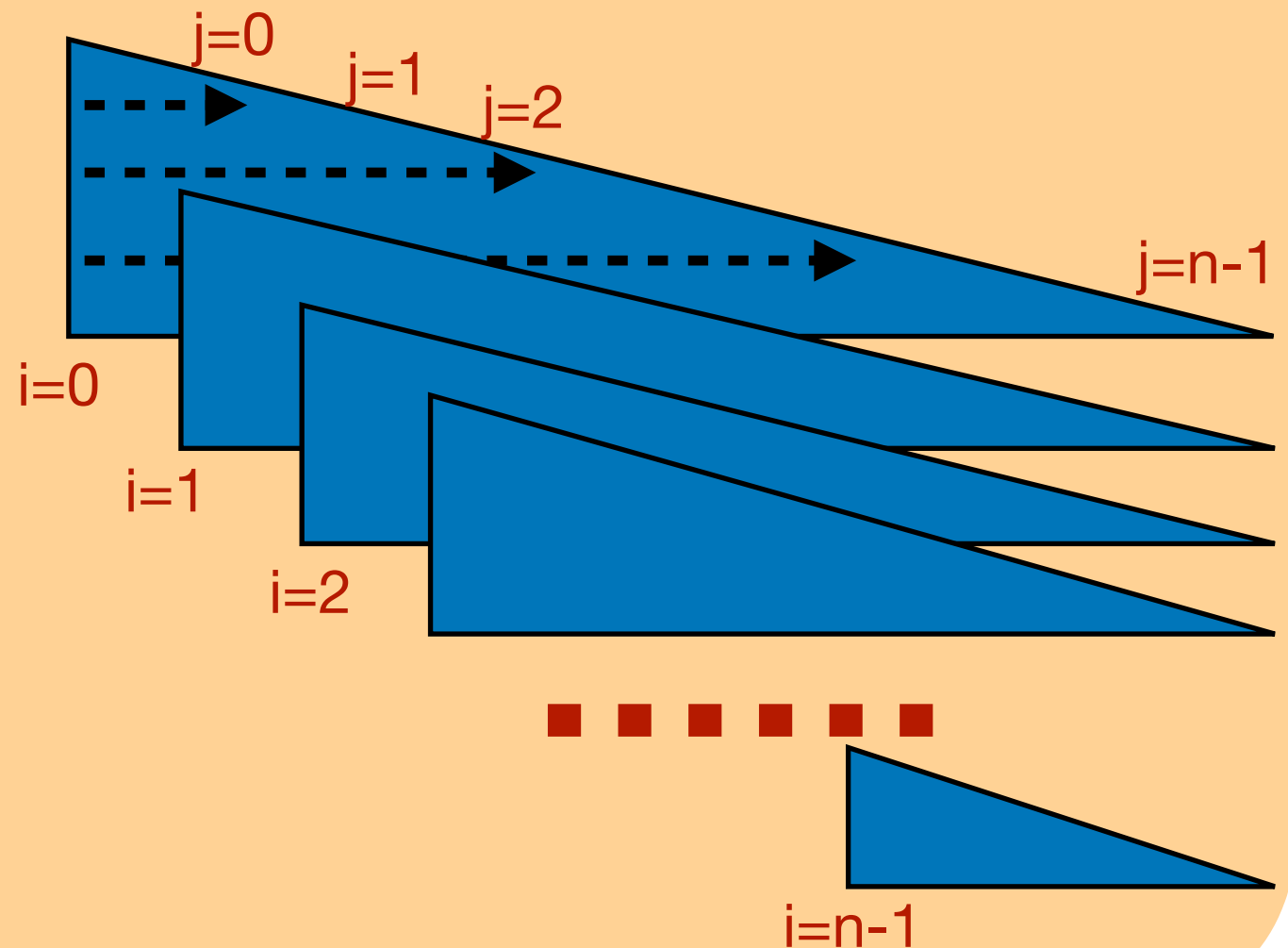
```
      ThisSum += A[k];
```

```
    if(ThisSum > MaxSum)
```

```
      MaxSum = ThisSum;
```

```
  }
```

```
return MaxSum;
```



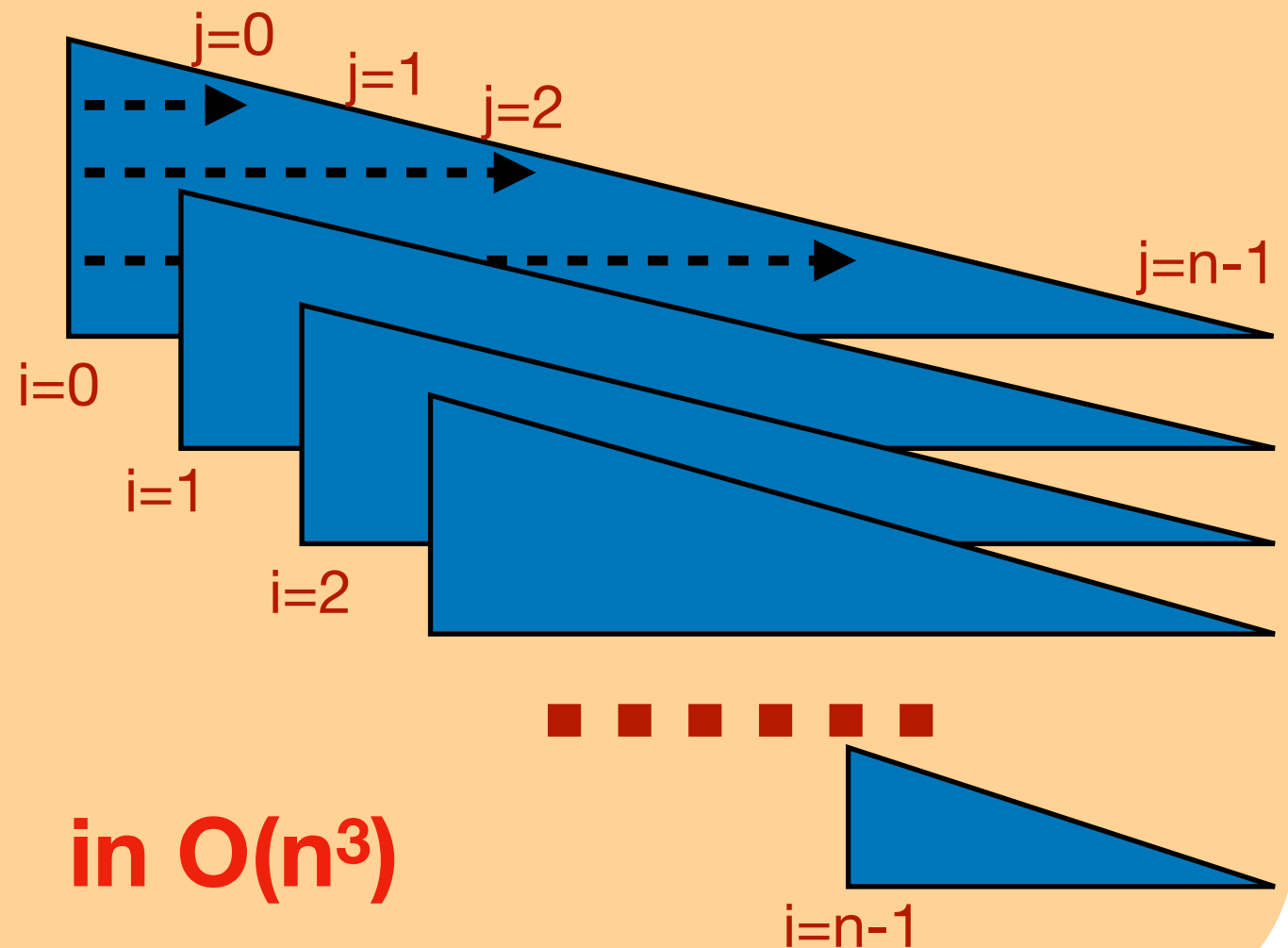
# Max-sum subsequence

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

An example: 2, 11, -4, 13, -5, -2; the result 22: (2, 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;
```



# More Precise Complexity

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



# More Precise Complexity

● The total cost is:

$$\sum_{i=0}^{n-1} \sum_{j=i}^{n-1} \sum_{k=i}^j 1$$
$$\sum_{k=i}^j 1 = j - i + 1$$

# More Precise Complexity

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

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

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

# More Precise Complexity

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

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

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

---

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

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

# More Precise Complexity

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

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

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

---

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

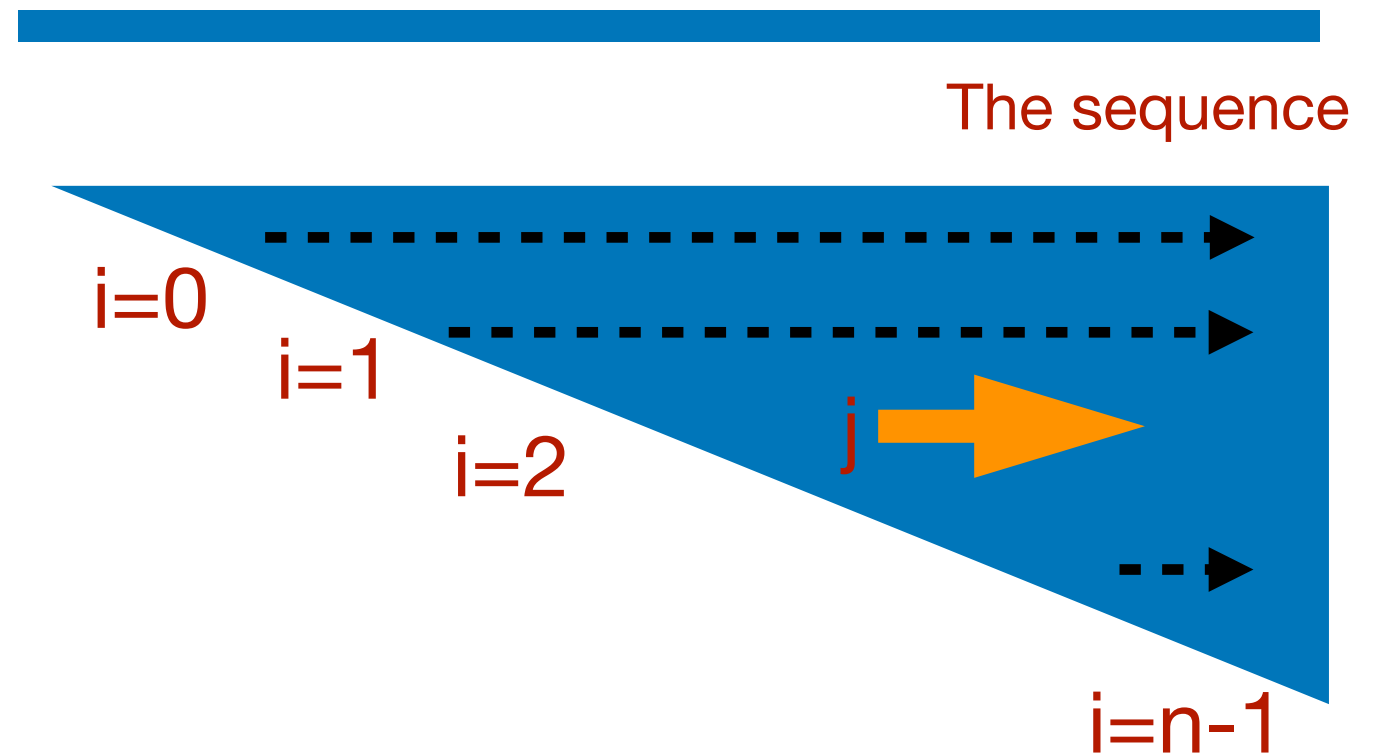
$$= \frac{1}{2} \sum_{i=1}^n i^2 - \left(n + \frac{3}{2}\right) \sum_{i=1}^n i + \frac{1}{2}(n^2 + 3n + 2) \sum_{i=1}^n 1$$

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

# Decreasing the Number of Loops

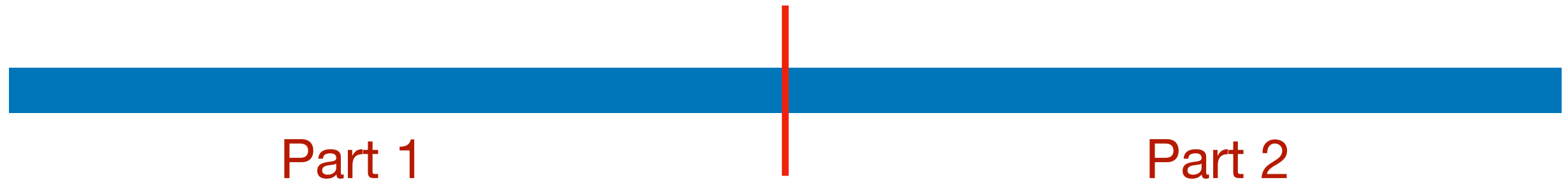
A improved algorithm:

```
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;
```

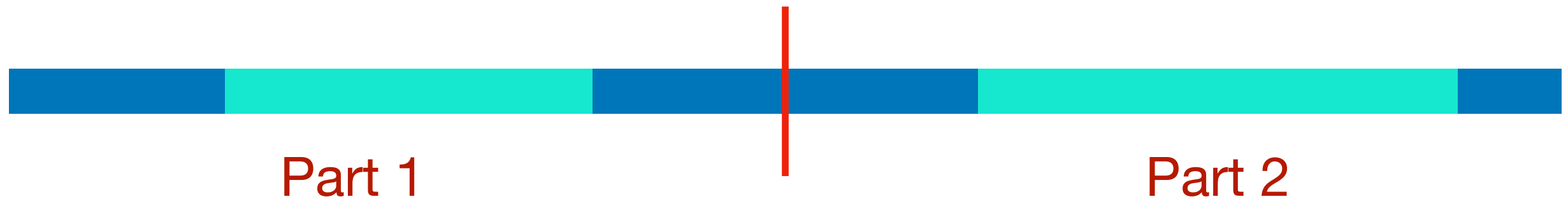


**in  $O(n^2)$**

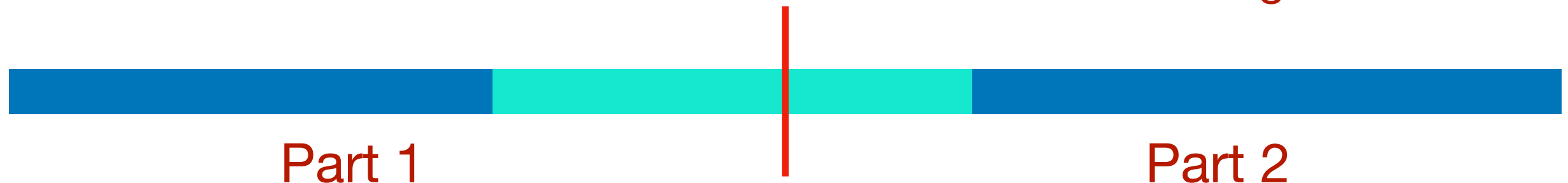
# Power of Divide and Conquer



The sub with largest sum may be in:



or:  The largest is the result



# 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.**

**in  $O(n \log n)$**

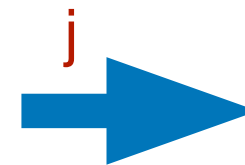
# A Linear Algorithm

```
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



This is an example  
of “online algorithm”



Negative item or subsequence  
cannot be a prefix of the  
subsequence we want.

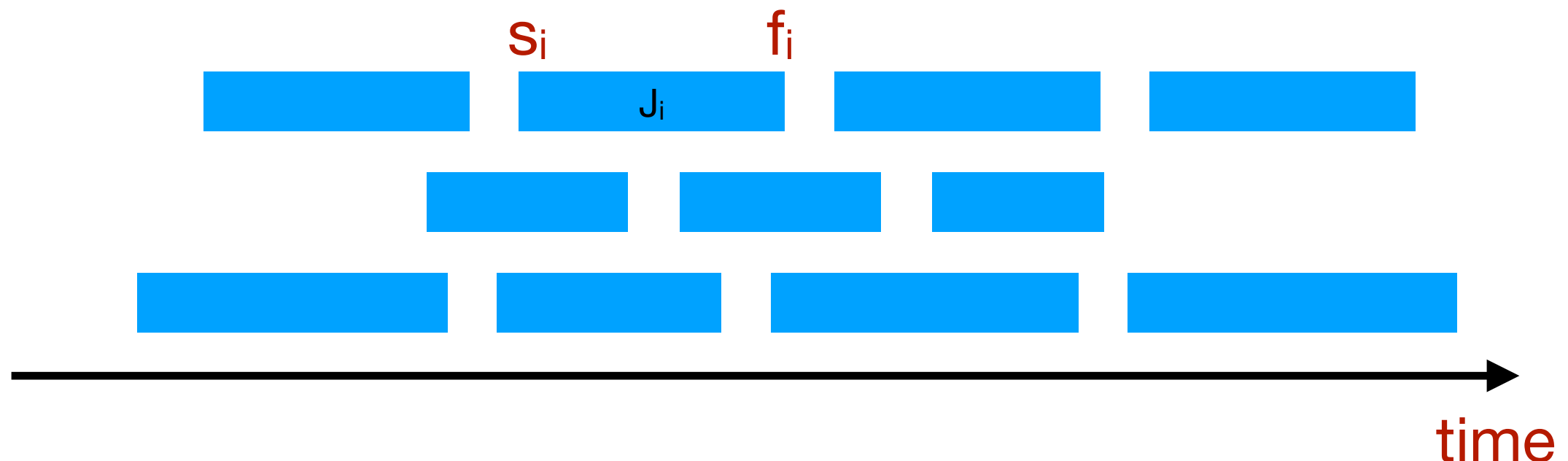


# Brute Force Enumeration By Recursion

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

# Job Scheduling

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



# Matrix Chain Multiplication

- The task:

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

An example:  $A_1 \times A_2 \times A_3 \times A_4$

$30 \times 1 \quad 1 \times 40 \quad 40 \times 10 \quad 10 \times 25$

$((A_1 \times A_2) \times A_3) \times A_4$ : 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

# Solutions

- **Brute force recursion (L16)**
  - BF1
  - BF2
- **Dynamic programming (L16)**
  - Based on brute force recursion 2

Thank you!

Q & A