

# **Design and Analysis of Algorithms**

## **Part II: Dynamic Programming**

### **Lecture 11: Maximum Contiguous Subarray Problem II**

---

**童咏昕**

---

**北京航空航天大学  
计算机学院**



- 在算法课程第二部分“动态规划”主题中，我们将主要聚焦于如下经典问题：
  - 0–1 Knapsack (0–1背包问题)
  - Maximum Contiguous Subarray II (最大连续子数组 II)
  - Longest Common Subsequences (最长公共子序列)
  - Longest Common Substrings (最长公共子串)
  - Minimum Edit Distance (最小编辑距离)
  - Rod–Cutting (钢条切割)
  - Chain Matrix Multiplication (矩阵链乘法)

# 最大子数组问题



	1	2	3	4	5	6	7	8	9	10
X	-1	-3	3	5	-4	3	2	-2	3	6

- 子数组  $X[3..7]$ 
  - 求和为:  $3 + 5 - 4 + 3 + 2 = 9$
- 子数组  $X[1..10]$ 
  - 求和为:  $-1 - 3 + 3 + 5 - 4 + 3 + 2 - 2 + 3 + 6 = 12$
- 子数组  $X[3..10]$ 
  - 求和为:  $3 + 5 - 4 + 3 + 2 - 2 + 3 + 6 = 16$

问题: 寻找数组X最大的非空子数组?

答案:  $X[3..10] = 16$



- 形式化定义

## 最大子数组问题

### Max Continuous Subarray, MCS

#### 输入

- 给定一个数组 $X[1..n]$ ，对于任意一对数组下标为 $l, r (l \leq r)$ 的非空子数组，其和记为

$$S(l, r) = \sum_{i=l}^r X[i]$$

#### 输出

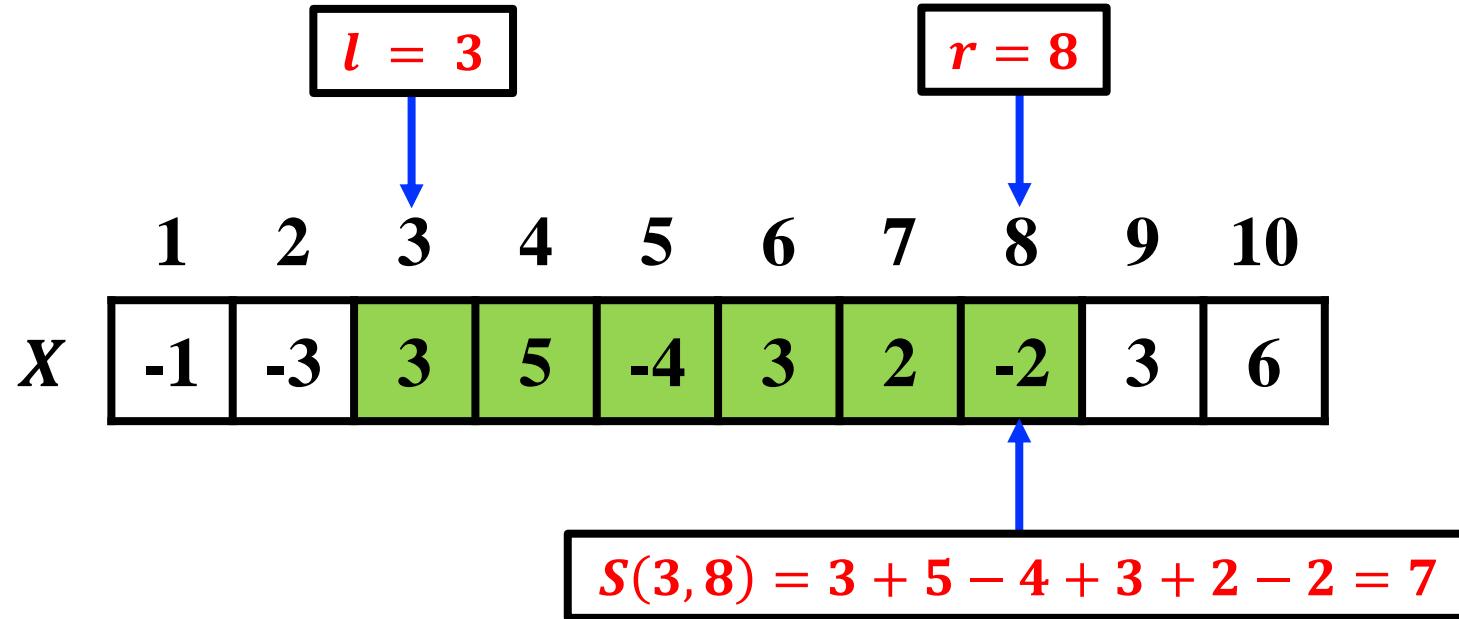
- 求出 $S(l, r)$ 的最大值，记为 $S_{max}$

	1	2	3	4	5	6	7	8	9	10
X	-1	-3	3	5	-4	3	2	-2	3	6

- 数组  $X[1..n]$ , 其所有的下标  $l, r (l \leq r)$  组合分为以下两种情况
  - 当  $l = r$  时, 一共  $C_n^1 = n$  种组合
  - 当  $l < r$  时, 一共  $C_n^2$  种组合
- 枚举  $n + C_n^2$  种下标  $l, r$  组合, 求出最大子数组之和

# 蛮力枚举

- $l = 3, r = 8$
- 计算  $S(3, 8)$ :





# 蛮力枚举

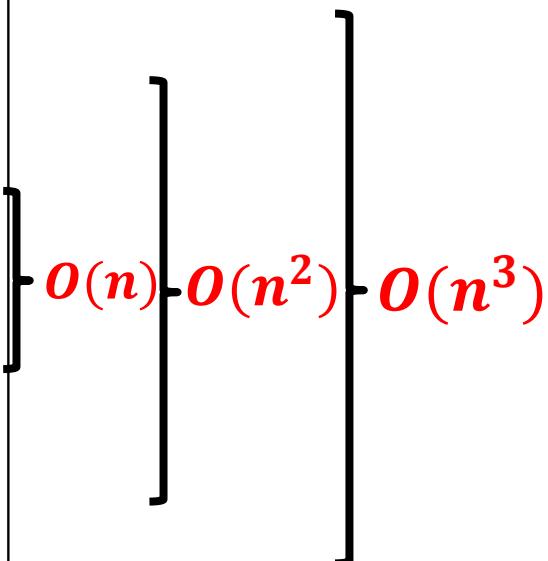
- 枚举 $n + C_n^2$ 种可能的区间 $[l, r](l \leq r)$ , 求解最大子数组之和 $S_{max}$

输入: 数组 $X[1..n]$

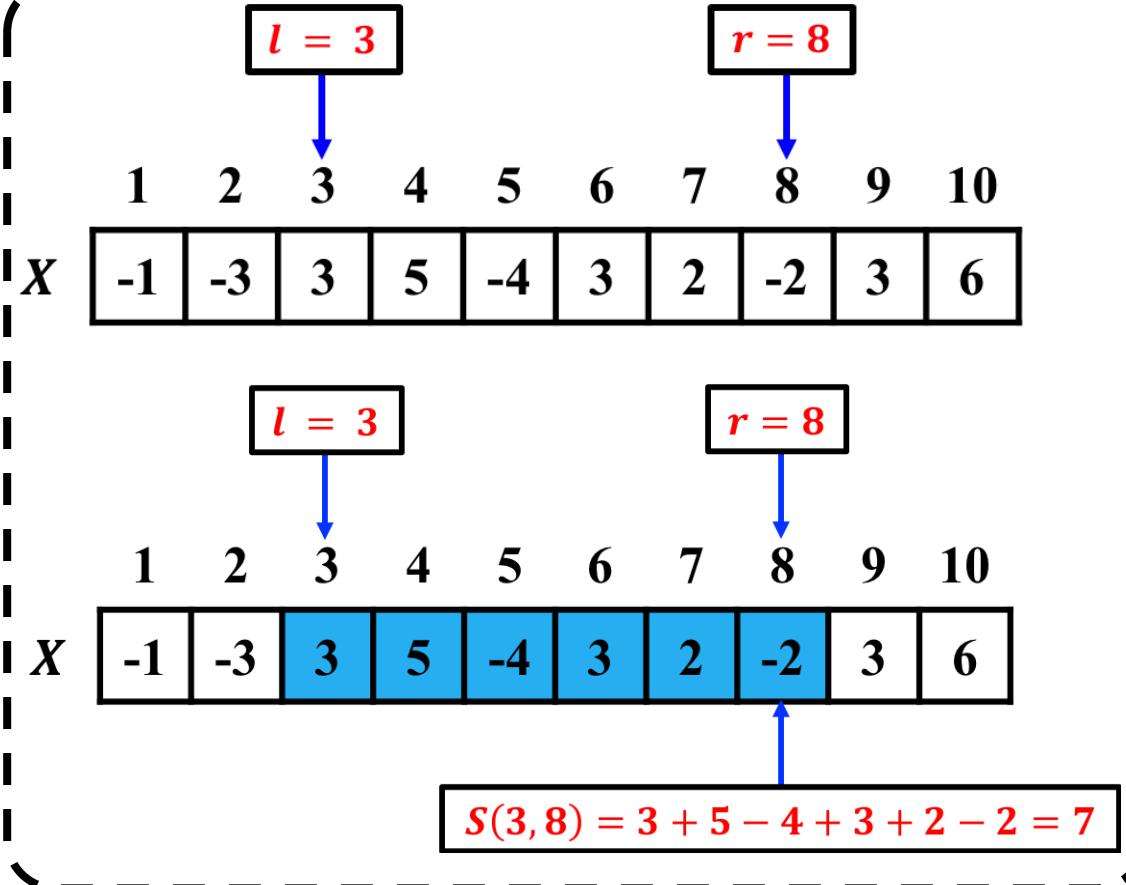
输出: 最大子数组之和 $S_{max}$

```
 $S_{max} \leftarrow -\infty$ 
for  $l \leftarrow 1$  to  $n$  do
    for  $r \leftarrow l$  to  $n$  do
         $S(l, r) \leftarrow 0$ 
        for  $i \leftarrow l$  to  $r$  do
             $S(l, r) \leftarrow S(l, r) + X[i]$ 
        end
         $S_{max} \leftarrow \max\{S_{max}, S(l, r)\}$ 
    end
end
return  $S_{max}$ 
```

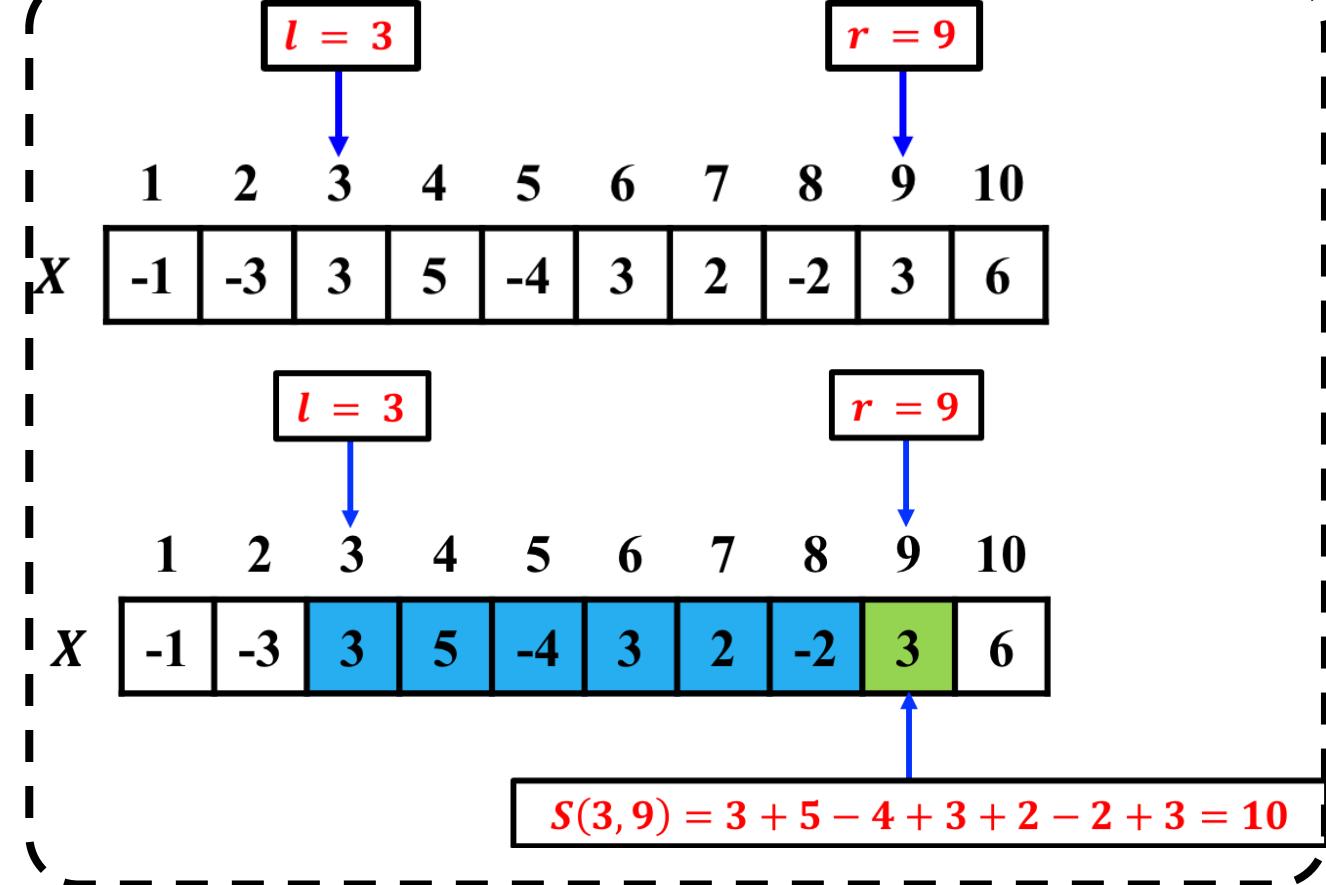
时间复杂度:  $O(n^3)$



# 从蛮力枚举到优化枚举



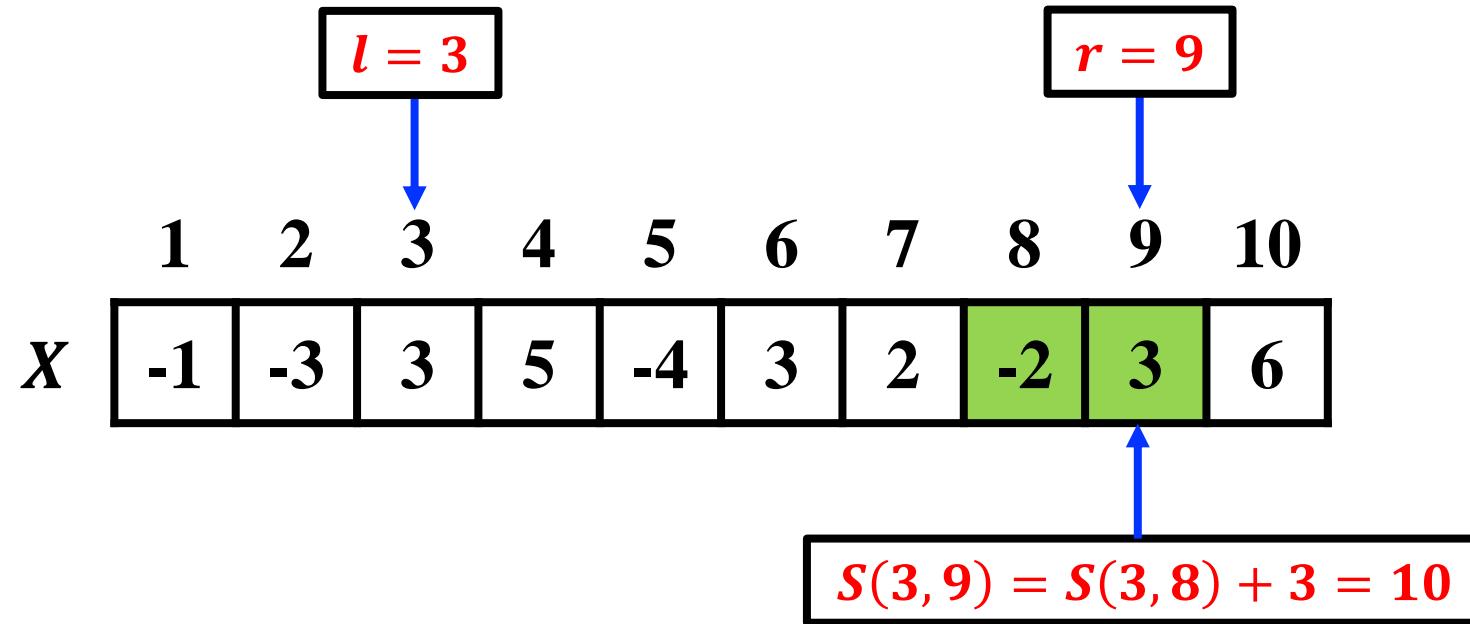
计算  $S(3, 8)$  循环 6 次



计算  $S(3, 9)$  循环 7 次

# 优化枚举

- $l = 3, r = 9, S(3, 9) = ?$





# 优化枚举

- 核心思想:  $S(l, r) = \sum_{i=l}^r X[i] = S(l, r-1) + X[r]$

输入: 数组  $X[1..n]$

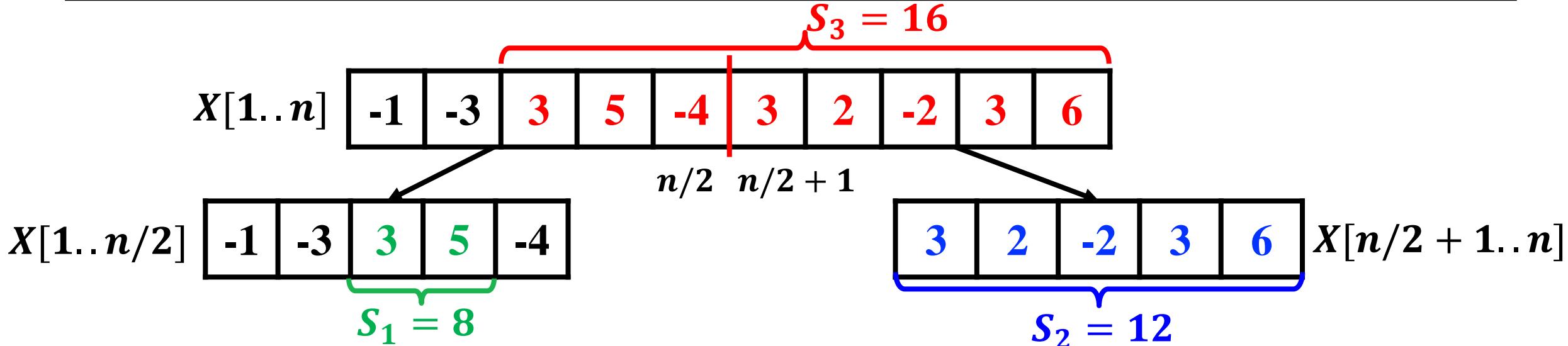
输出: 最大子数组之和  $S_{max}$

```
 $S_{max} \leftarrow -\infty$ 
for  $l \leftarrow 1$  to  $n$  do
     $S \leftarrow 0$ 
    for  $r \leftarrow l$  to  $n$  do
         $S \leftarrow S + X[r]$ 
         $S_{max} \leftarrow \max\{S_{max}, S\}$ 
    end
end
return  $S_{max}$ 
```

$O(n)$   $O(n^2)$

时间复杂度:  $O(n^2)$

# 分而治之



- 将数组  $X[1..n]$  分为  $X[1..n/2]$  和  $X[n/2 + 1..n]$

分解原问题

- 递归求解子问题

解决子问题

- $S_1$ : 数组  $X[1..n/2]$  的最大子数组
- $S_2$ : 数组  $X[n/2 + 1..n]$  的最大子数组

- 合并子问题，得到  $S_{max}$

合并问题解

- $S_3$ : 跨中点的最大子数组
- 数组  $X$  的最大子数组之和  $S_{max} = \max\{S_1, S_2, S_3\}$



# 分而治之

## ● 时间复杂度分析

输入: 数组  $X$ , 数组下标  $low, high$

输出: 最大子数组之和  $S_{max}$

**if**  $low = high$  **then**

**return**  $X[low]$

**end**

**else**

$mid \leftarrow \lfloor \frac{low+high}{2} \rfloor$

$S_1 \leftarrow \text{MaxSubArray}(X, low, mid)$

$S_2 \leftarrow \text{MaxSubArray}(X, mid+1, high)$

$S_3 \leftarrow \text{CrossingSubArray}(X, low, mid, high)$

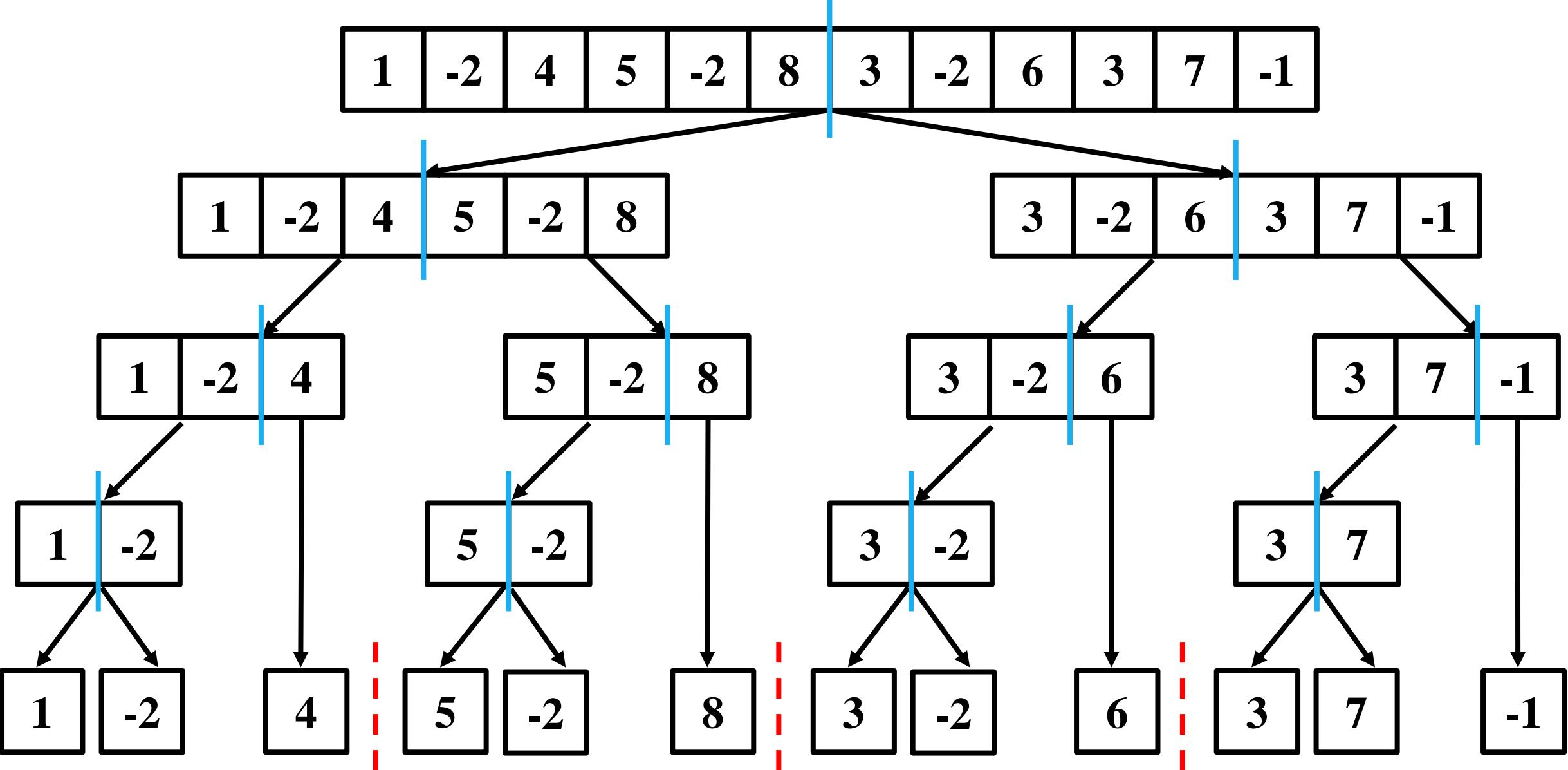
$S_{max} \leftarrow \max\{S_1, S_2, S_3\}$

**return**  $S_{max}$

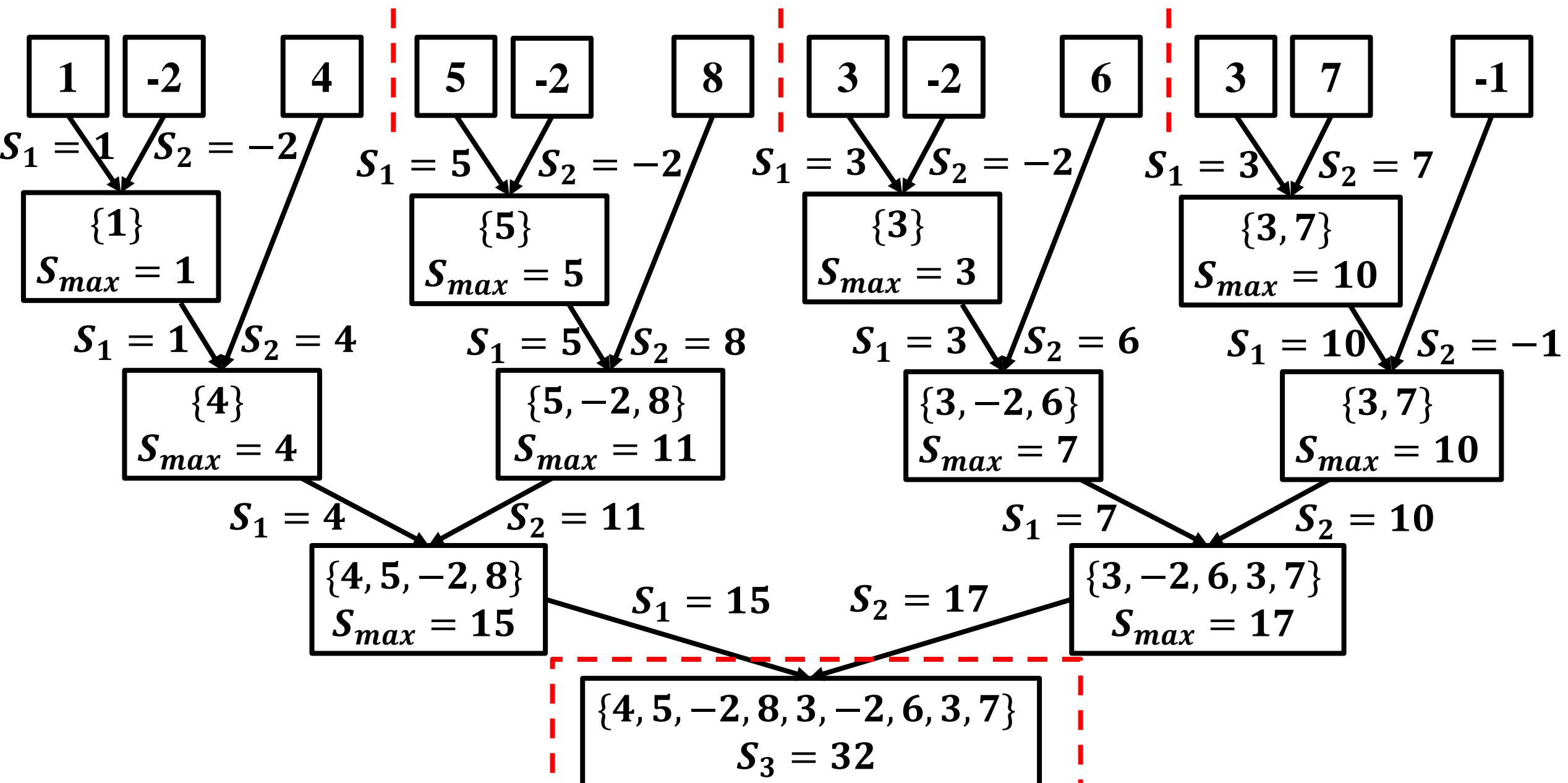
**end**

时间复杂度:  $O(n \log n)$

# 分而治之



# 算法实例





算法名称	时间复杂度
蛮力枚举	$O(n^3)$
优化枚举	$O(n^2)$
分而治之	$O(n \log n)$
?	$O(n)$

问题：是否可以设计一个时间复杂度为 $O(n)$ 的算法？

# 算法比较

基于枚举 [

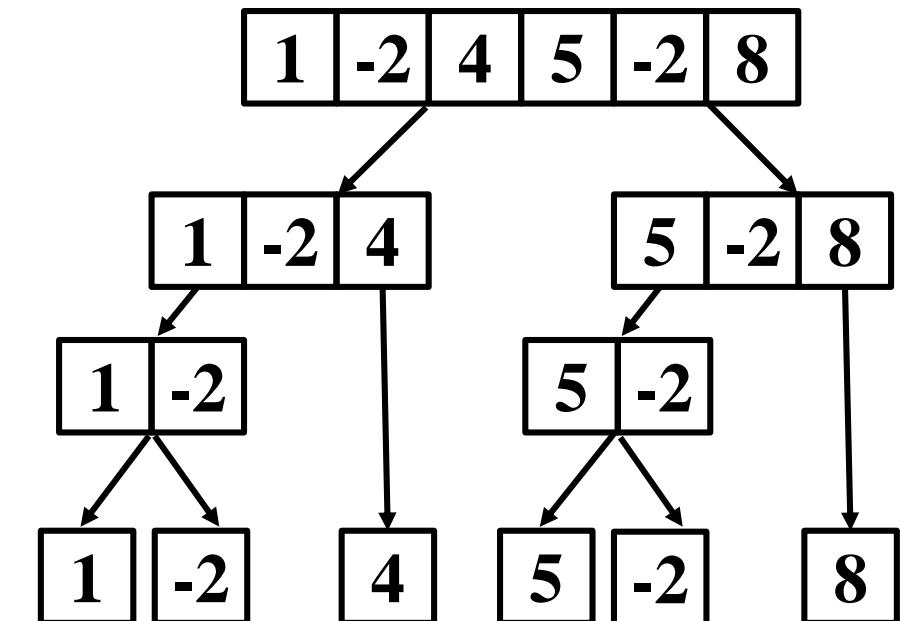
	算法名称	时间复杂度
	蛮力枚举	$O(n^3)$
	优化枚举	$O(n^2)$
	分而治之	$O(n \log n)$
	动态规划	$O(n)$



- 枚举区间



存在重叠子问题



子问题相互独立



## • 两层枚举

- 第1层：枚举位置*i*作为区间开头
- 第2层：枚举位置*j*作为区间结尾



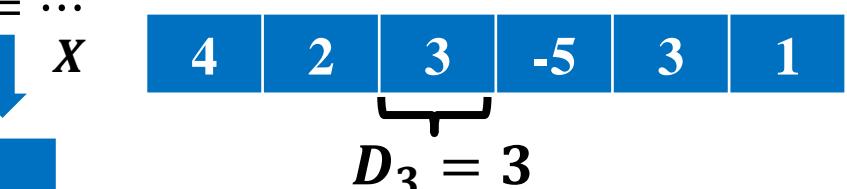
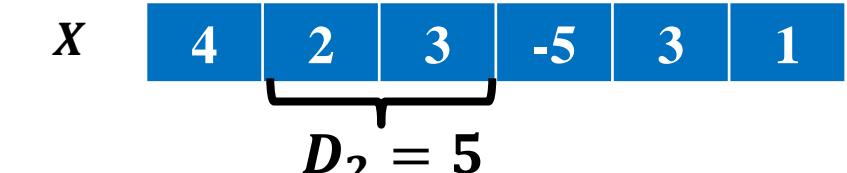
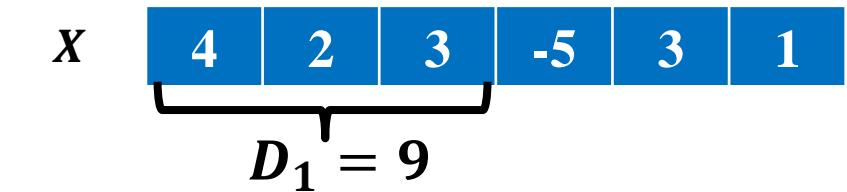
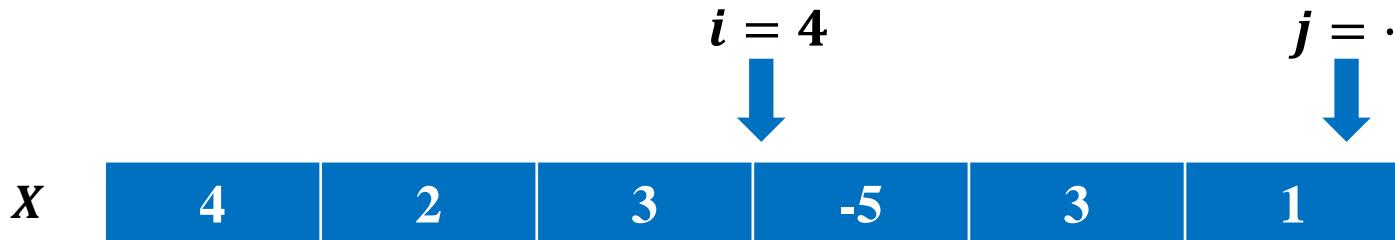
当前最大值 = 9

$D_i$ : 以 $X[i]$ 开头的最大子数组

# 枚举过程分析

- 两层枚举

- 第1层：枚举位置*i*作为区间开头
- 第2层：枚举位置*j*作为区间结尾

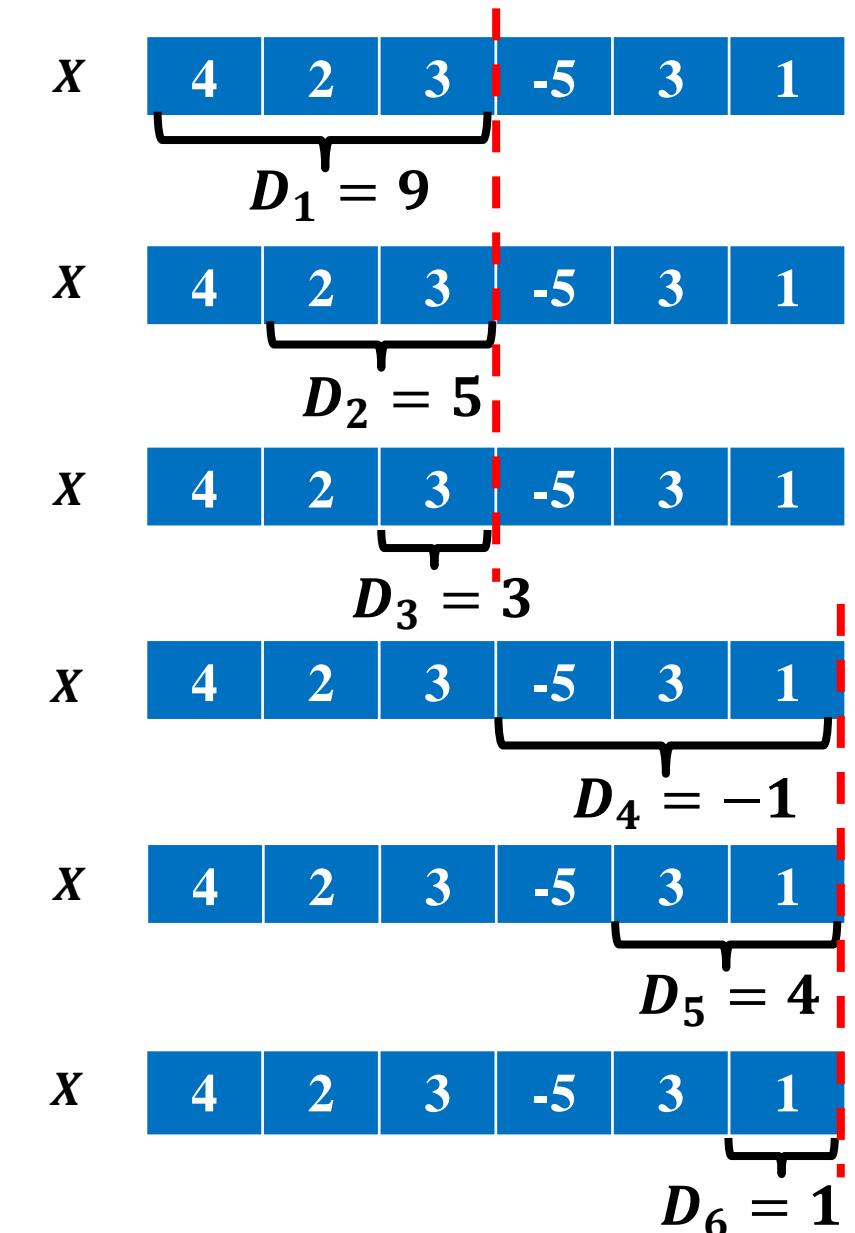


# 枚举过程分析



- **两层枚举**

- 第1层：枚举位置*i*作为区间开头
- 第2层：枚举位置*j*作为区间结尾



# 规律观察

- 结尾位置相同

  - $D_1, D_2, D_3$

    - $D_2 = X[2] + D_3$
    - $D_1 = X[1] + D_2$



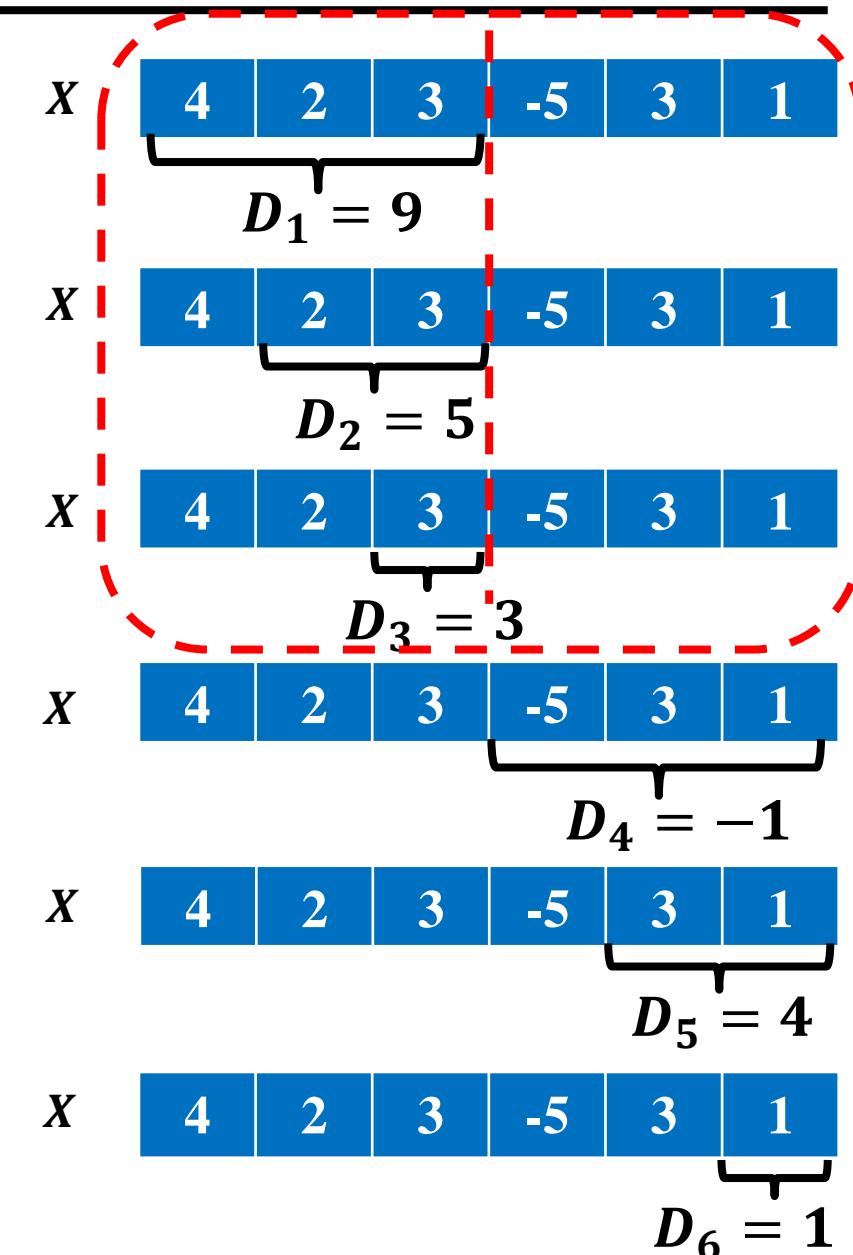
$$D_1 = 9$$



$$D_2 = 5$$



$$D_3 = 3$$



# 规律观察

- 结尾位置不同

  - $D_3, D_4$

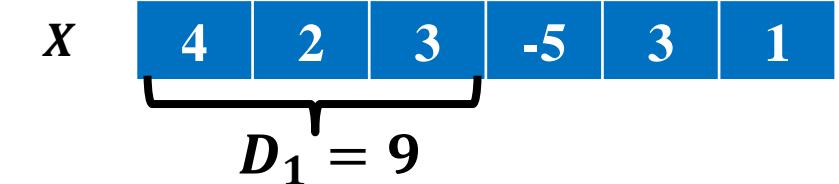
  - $D_3 = X[3]$
  - $D_4 < 0$



$$D_3 = 3$$



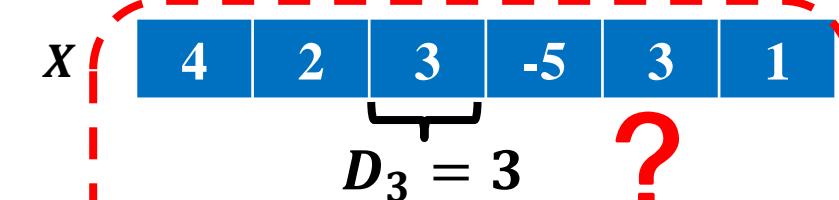
$$D_4 = -1$$



$$D_1 = 9$$



$$D_2 = 5$$

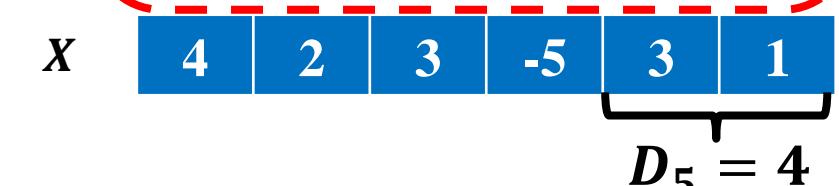


$$D_3 = 3$$

?



$$D_4 = -1$$



$$D_5 = 4$$



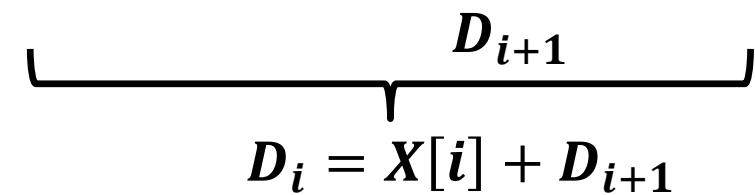
$$D_6 = 1$$



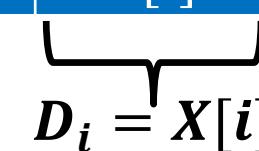
# 规律描述

- $D_i$ : 以 $X[i]$ 开头的最大子数组和

- 情况1:  $D_{i+1} > 0$ 时  $X[1] \mid \dots \mid X[i] \mid X[i+1] \mid \dots \mid \dots \mid X[n]$

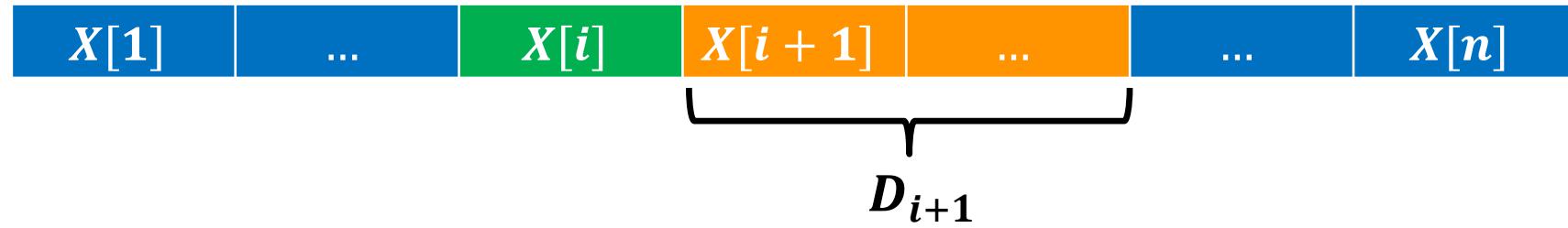


- 情况2:  $D_{i+1} \leq 0$ 时  $X[1] \mid \dots \mid X[i] \mid X[i+1] \mid \dots \mid \dots \mid X[n]$

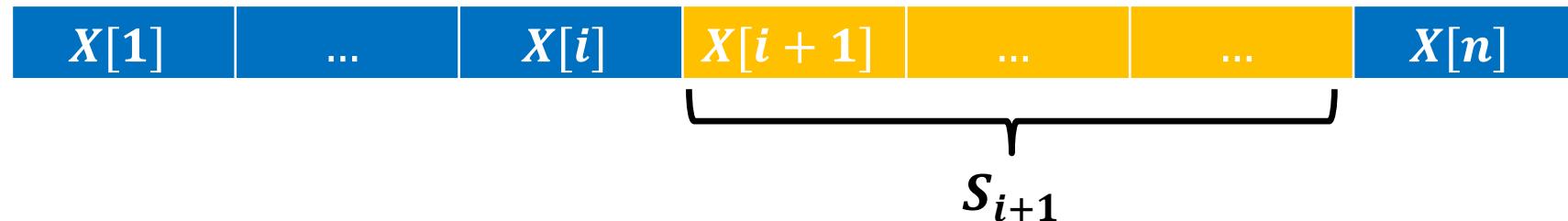


# 规律证明：情况1

- $D_{i+1}$ : 以 $X[i + 1]$ 开头的最大子数组和 ( $D_{i+1} > 0$ )



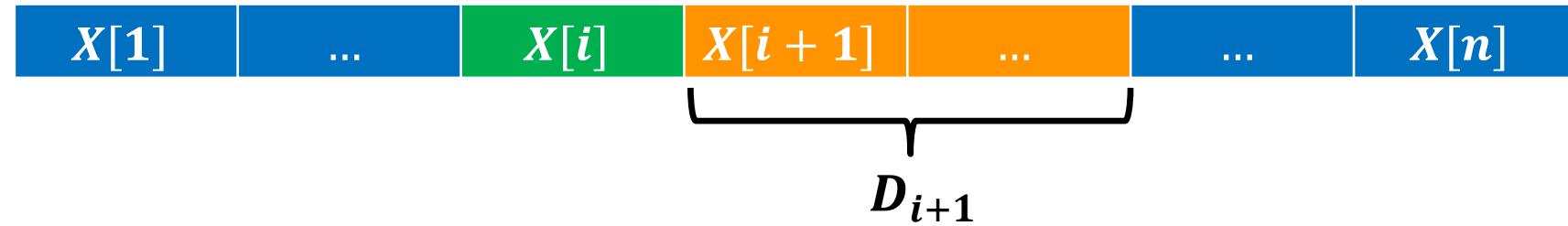
- $S_{i+1}$ : 以 $X[i + 1]$ 开头的任一子数组和 ( $S_{i+1} \leq D_{i+1}$ )



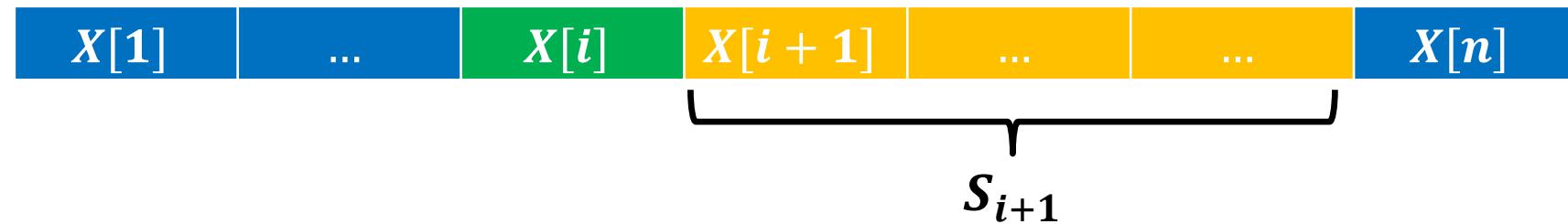
- $X[i] + D_{i+1} \geq X[i] + S_{i+1}$
- $D_i = X[i] + D_{i+1}$

# 规律证明：情况2

- $D_{i+1}$ : 以 $X[i + 1]$ 开头的最大子数组和 ( $D_{i+1} \leq 0$ )



- $S_{i+1}$ : 以 $X[i + 1]$ 开头的任一子数组和 ( $S_{i+1} \leq D_{i+1}$ )



- $X[i] + S_{i+1} \leq X[i] + D_{i+1} \leq X[i]$
- $D_i = X[i]$

# 问题结构分析



- 给出问题表示

- $D[i]$ : 以 $x[i]$ 开头的最大子数组和

- 明确原始问题

- $S_{max} = \max_{1 \leq i \leq n} \{D[i]\}$

问题结构分析

递推关系建立

自底向上计算

最优方案追踪

# 递推关系建立：分析最优（子）结构

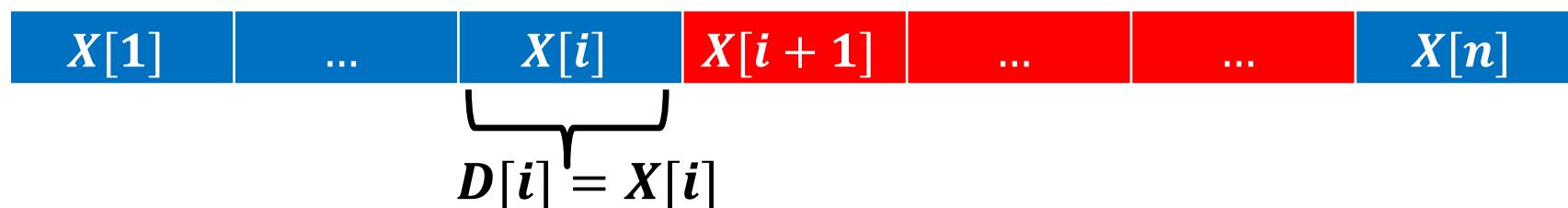
- $D[i]$ : 以 $X[i]$ 开头的**最大子数组和**
- 情况1:  $D[i + 1] > 0$



$$D[i] = X[i] + D[i+1]$$

- 情况2:  $D[i + 1] \leq 0$

**最优子结构**



问题结构分析

递推关系建立

自底向上计算

最优方案追踪

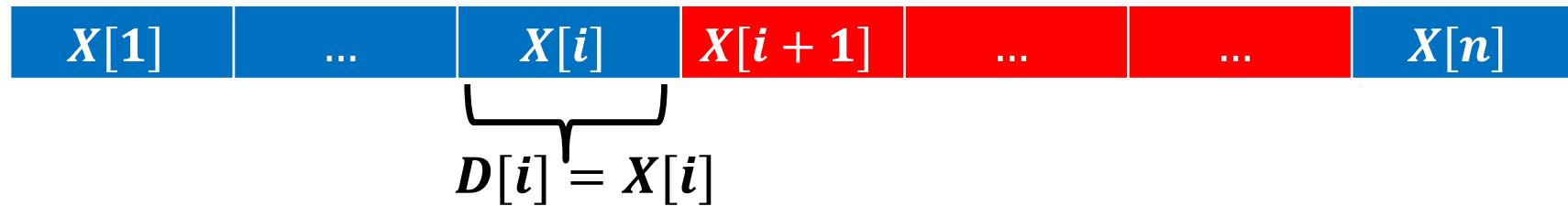


# 递推关系建立：构造递推公式



问题结构分析

$$D[i] = X[i] + D[i+1]$$



递推关系建立

$$\bullet D[i] = \begin{cases} X[i] + D[i+1], & \text{if } D[i+1] > 0 \\ X[i], & \text{if } D[i+1] \leq 0 \end{cases}$$

自底向上计算

最优方案追踪

# 自底向上计算：确定计算顺序

## ● 初始化

- $D[n] = X[n]$

已知

	1	2	...	$i$	$i+1$	...	$n-1$	$n$
$X$	$X[1]$	$X[2]$		$X[i]$				$X[n]$
	1	2	...	$i$	$i+1$	...	$n-1$	$n$
$D$								$X[n]$

问题结构分析

递推关系建立

自底向上计算

最优方案追踪



# 自底向上计算：依次求解问题

- 初始化

- $D[n] = X[n]$

- 递推公式

- $D[i] = \begin{cases} X[i] + D[i+1], & \text{if } D[i+1] > 0 \\ X[i], & \text{if } D[i+1] \leq 0 \end{cases}$

已知

	1	2	...	$i$	$i+1$	...	$n-1$	$n$
$X$	$X[1]$	$X[2]$		$X[i]$				$X[n]$

	1	2	...	$i$	$i+1$	...	$n-1$	$n$
$D$								$X[n]$

自底向上计算

问题结构分析

递推关系建立

自底向上计算

最优方案追踪

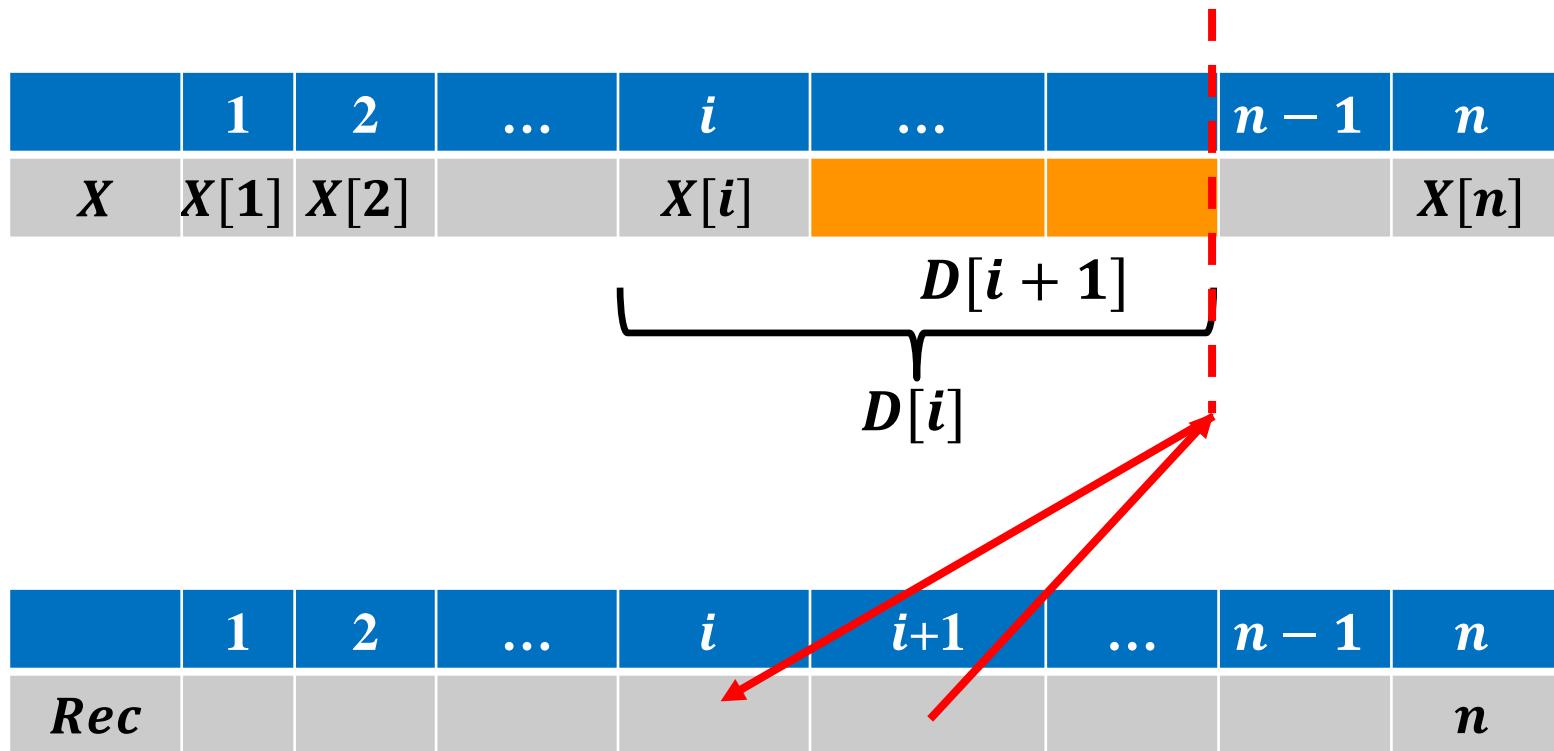
# 最优方案追踪：记录决策过程



- 构造追踪数组  $Rec[1..n]$

- 情况1：结尾相同

- $Rec[i] = Rec[i + 1]$



问题结构分析

递推关系建立

自底向上计算

最优方案追踪

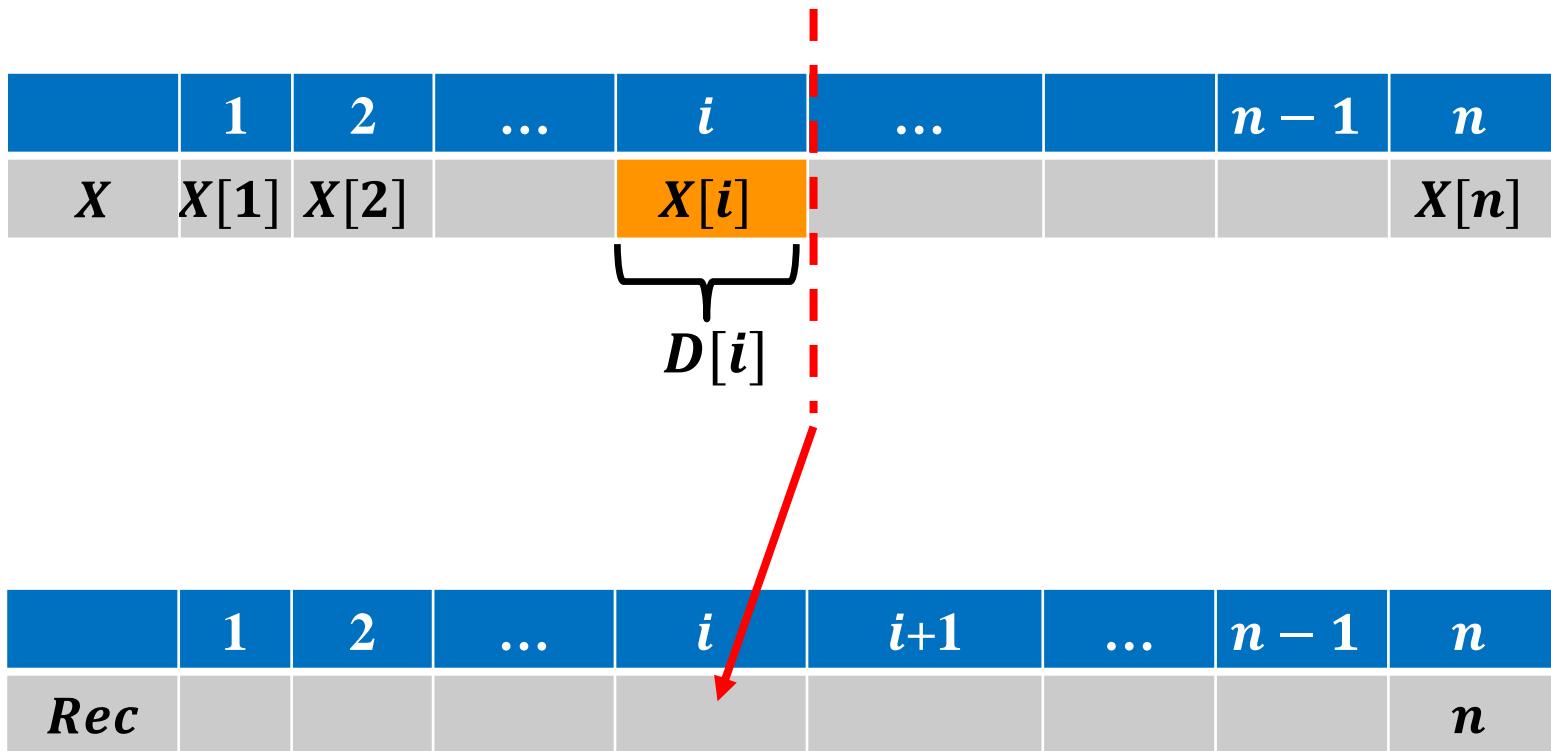
# 最优方案追踪：记录决策过程



- 构造追踪数组  $Rec[1..n]$

- 情况2：结尾不同

- $Rec[i] = i$



问题结构分析

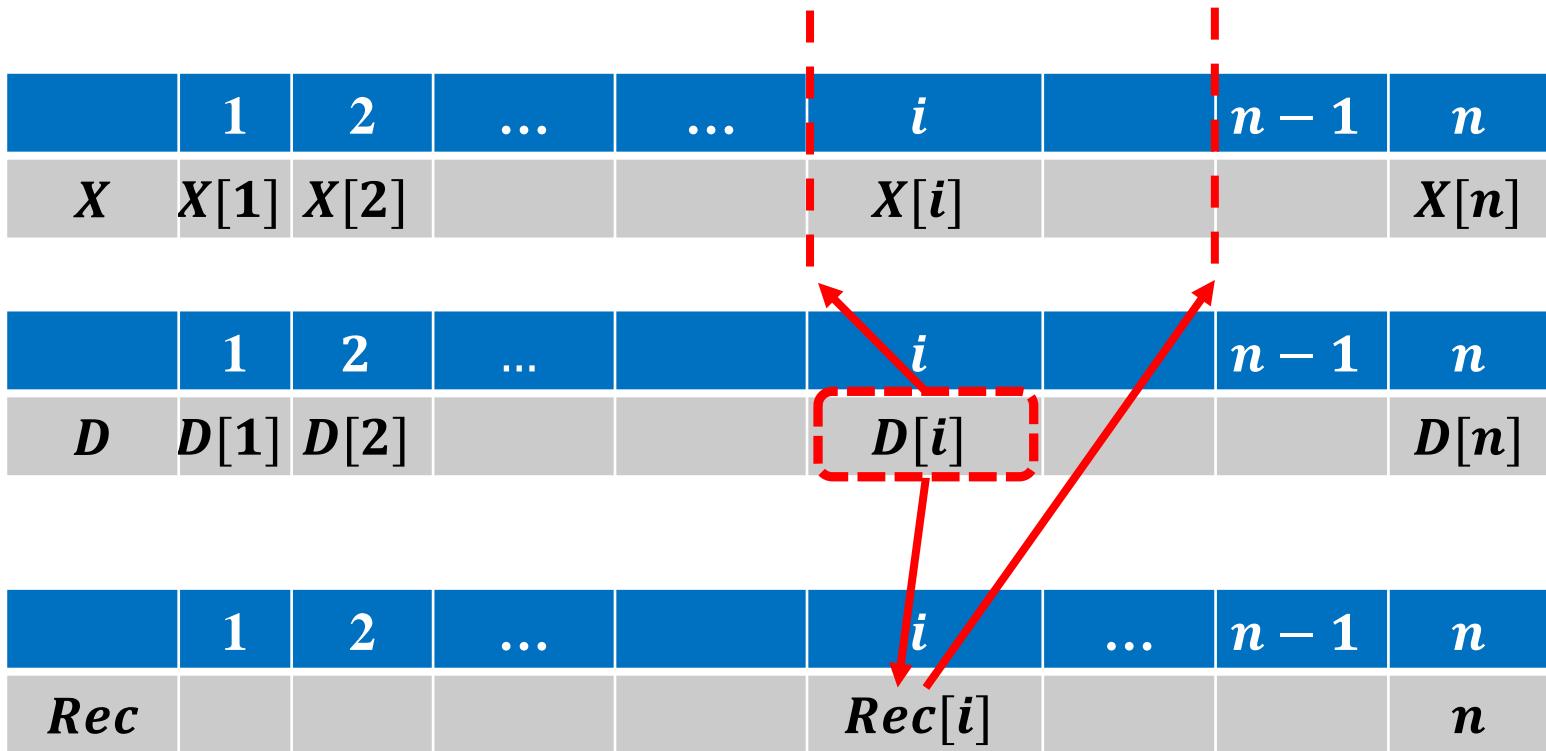
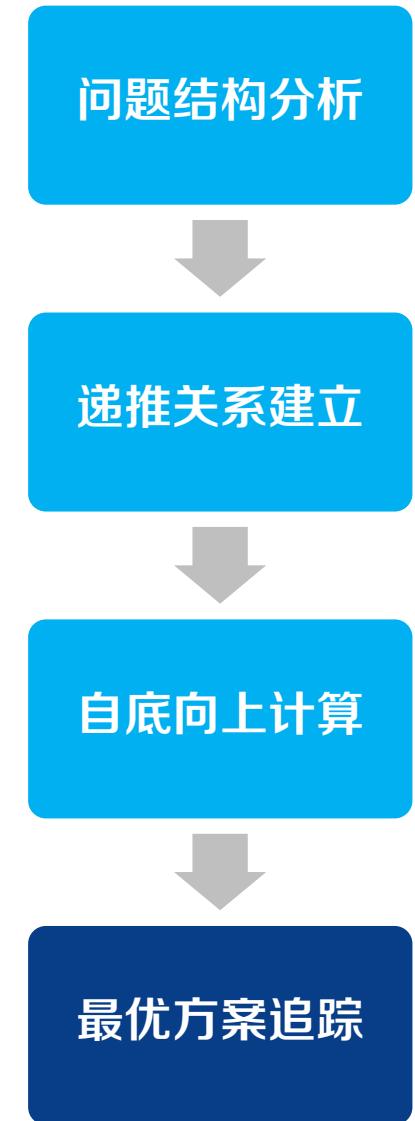
递推关系建立

自底向上计算

最优方案追踪

# 最优方案追踪：输出最优方案

- 从子问题中查找最优解
- 最大子数组开头位置： $i$
- 最大子数组结尾位置： $Rec[i]$



# 算法实例



$i$	1	2	3	4	5	6	7	8	9	10	11	12
$X$	1	-2	4	5	-2	8	3	-2	6	3	7	-1

# 算法实例



$i$	1	2	3	4	5	6	7	8	9	10	11	12
$X$	1	-2	4	5	-2	8	3	-2	6	3	7	-1

初始化

# 算法实例



$i$	1	2	3	4	5	6	7	8	9	10	11	12
$X$	1	-2	4	5	-2	8	3	-2	6	3	7	-1

$$D[i] = \begin{cases} X[i] + D[i+1], & \text{if } D[i+1] > 0 \\ X[i], & \text{if } D[i+1] \leq 0 \end{cases}$$



# 算法实例

$i$	1	2	3	4	5	6	7	8	9	10	11	12	
$X$		1	-2	4	5	-2	8	3	-2	6	3	7	-1

$i$	1	2	3	4	5	6	7	8	9	10	11	12	
$D$		31	30	32	28	23	25	17	14	16	10	7	-1

最优解

$i$	1	2	3	4	5	6	7	8	9	10	11	12
$Rec$		11	11	11	11	11	11	11	11	11	11	12

终止位置

$$S = \{4, 5, -2, 8, 3, -2, 6, 3, 7\}$$



# 动态规划：伪代码

- Max-Continuous-Subarray-DP( $X, n$ )

输入: 数组 $X$ , 数组长度 $n$

输出: 最大子数组和 $S_{max}$ , 子数组起止位置 $l, r$

新建一维数组 $D[1..n]$ 和 $Rec[1..n]$

//初始化

$D[n] \leftarrow X[n]$

$Rec[n] \leftarrow n$

//动态规划

for  $i \leftarrow n - 1$  to 1 do

    if  $D[i + 1] > 0$  then

$D[i] \leftarrow X[i] + D[i + 1]$

$Rec[i] \leftarrow Rec[i + 1]$

    end

    else

$D[i] \leftarrow X[i]$

$Rec[i] \leftarrow i$

    end

end



# 动态规划：伪代码

- Max-Continuous-Subarray-DP( $X, n$ )

```
//查找解
 $S_{max} \leftarrow D[1]$ 
for  $i \leftarrow 2$  to  $n$  do
    if  $S_{max} < D[i]$  then
         $S_{max} \leftarrow D[i]$ 
         $l \leftarrow i$ 
         $r \leftarrow Rec[i]$ 
    end
end
return  $S_{max}, l, r$ 
```

# 时间复杂度分析



输入: 数组 $X$ , 数组长度 $n$

输出: 最大子数组和 $S_{max}$ , 子数组起止位置 $l, r$

新建一维数组 $D[1..n]$ 和 $Rec[1..n]$

//初始化

$D[n] \leftarrow X[n]$

$Rec[n] \leftarrow n$

//动态规划

```
for i ← n - 1 to 1 do
    if D[i + 1] > 0 then
        | D[i] ← X[i] + D[i + 1]
        | Rec[i] ← Rec[i + 1]
    end
    else
        | D[i] ← X[i]
        | Rec[i] ← i
    end
end
```

$O(n)$

时间复杂度:  $O(n)$



---

謝謝

