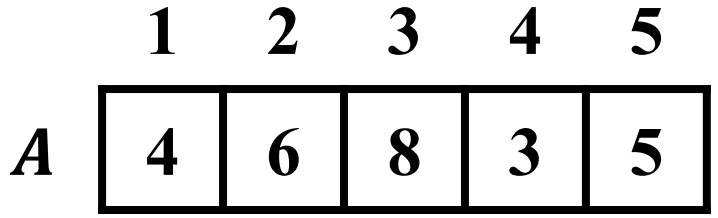
分而治之篇：逆序对计数问题

童咏昕

北京航空航天大学
计算机学院

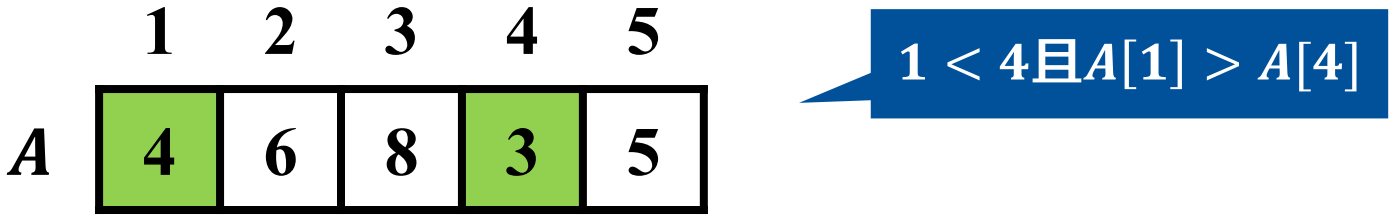
中国大学MOOC北航《算法设计与分析》

问题背景：逆序对



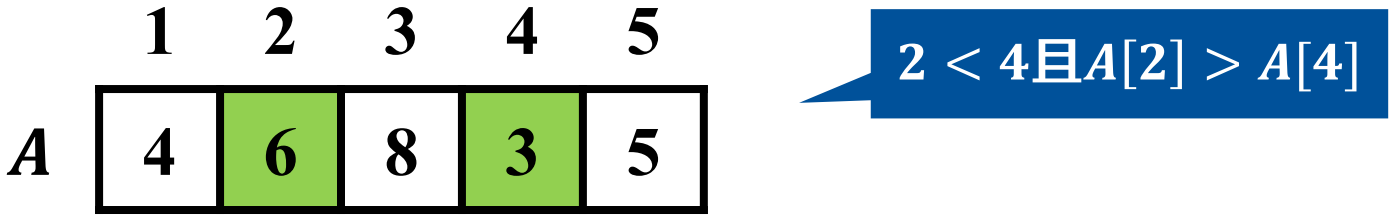
- 逆序对
  - 当$i < j$时$A[i] > A[j]$的二元组$(A[i], A[j])$

问题背景：逆序对



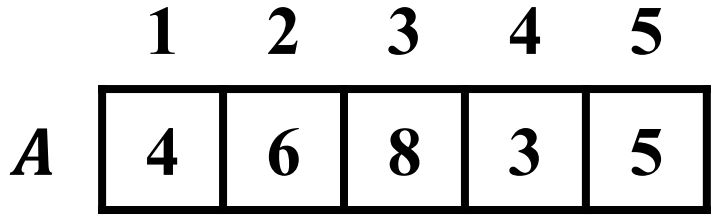
- 逆序对
  - 当$i < j$时$A[i] > A[j]$的二元组$(A[i], A[j])$
- 例如：$(A[1], A[4])$

问题背景：逆序对



- 逆序对
  - 当$i < j$时$A[i] > A[j]$的二元组$(A[i], A[j])$
- 例如：$(A[1], A[4])$，$(A[2], A[4])$

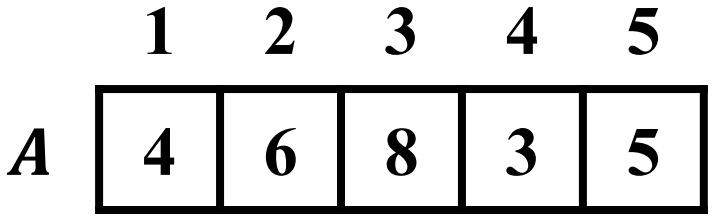
问题背景：逆序对



- 逆序对
  - 当$i < j$时$A[i] > A[j]$的二元组$(A[i], A[j])$
- 例如：$(A[1], A[4])$，$(A[2], A[4])$

问题：数组A中共有多少个逆序对？

问题背景：逆序对



- 逆序对
  - 当$i < j$时$A[i] > A[j]$的二元组$(A[i], A[j])$
- 例如：$(A[1], A[4])$，$(A[2], A[4])$

问题：数组A中共有多少个逆序对？
答案：共5个，$\{(A[1], A[4]), (A[2], A[4]), (A[2], A[5]), (A[3], A[4]), (A[3], A[5])\}$

问题定义

- 形式化定义

逆序对计数问题

Counting Inversions

输入

- 长度为n的数组$A[1..n]$

问题定义

- 形式化定义

逆序对计数问题

Counting Inversions

输入

- 长度为n的数组$A[1..n]$

输出

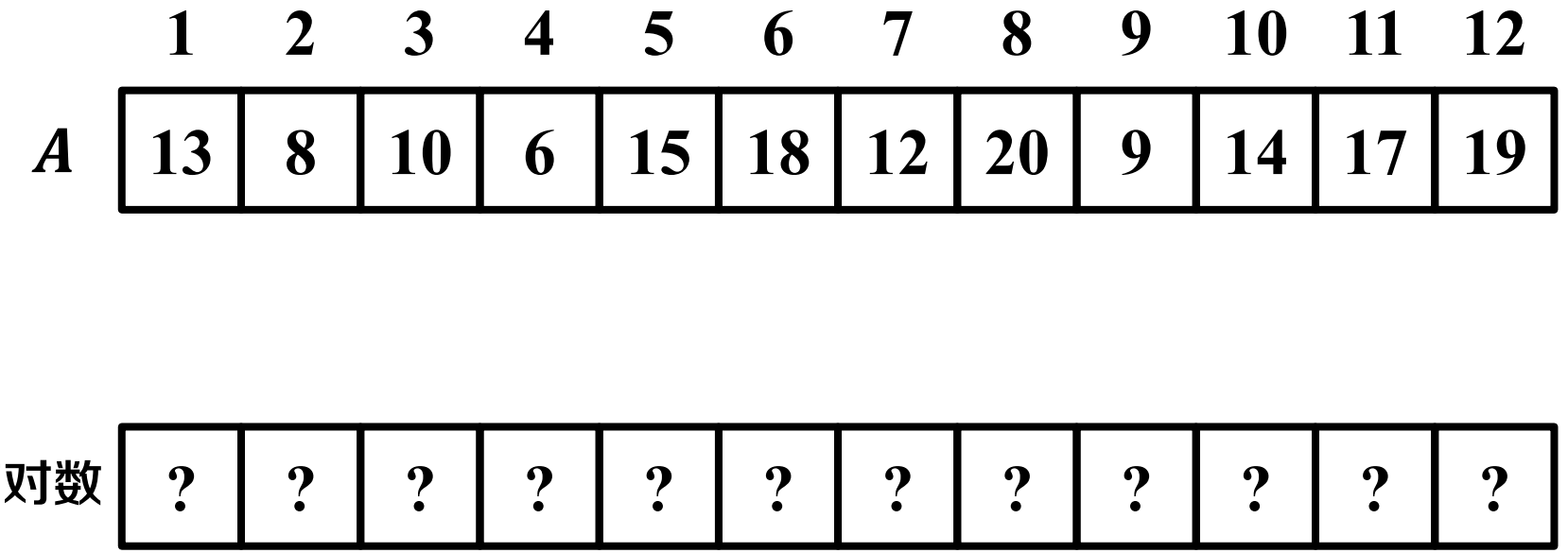
- 数组$A[1..n]$逆序对的总数，即：

$$\sum_{1\le i<j\le n} X_{i,j}$$

$$X_{i,j}=\begin{cases}1, & A[i]>A[j]\\ 0, & A[i]\le A[j]\end{cases}$$

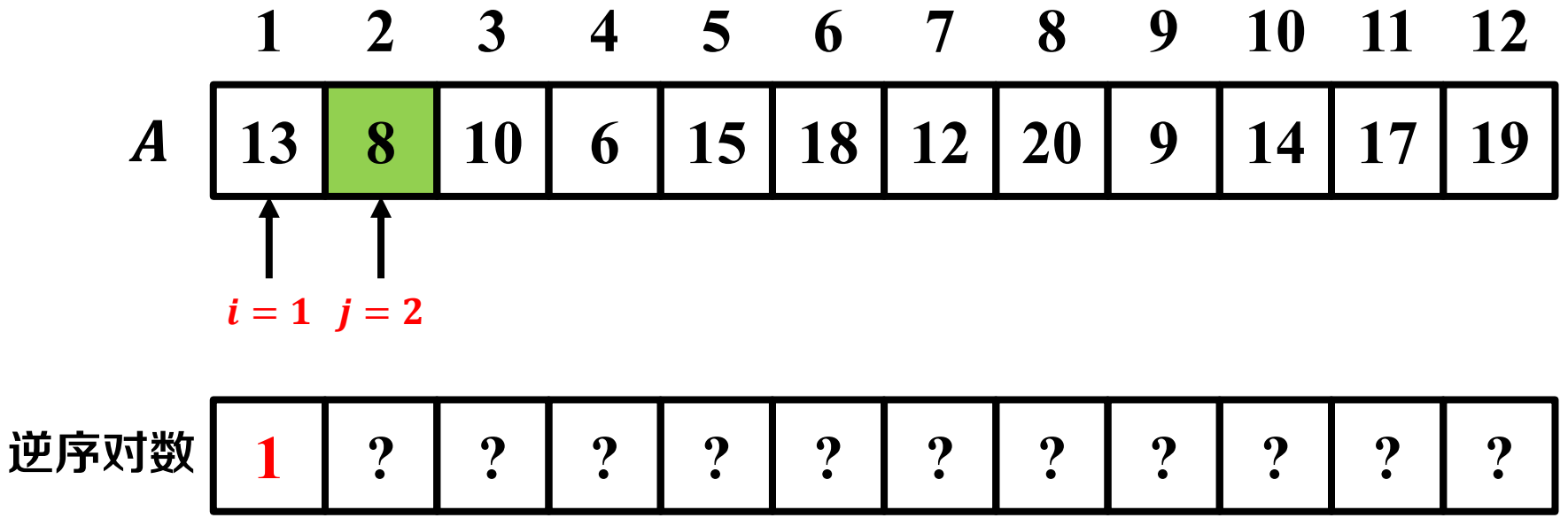
蛮力枚举

- 对于数组的每个元素$A[i]$，枚举$j(j > i)$，并统计逆序对数目



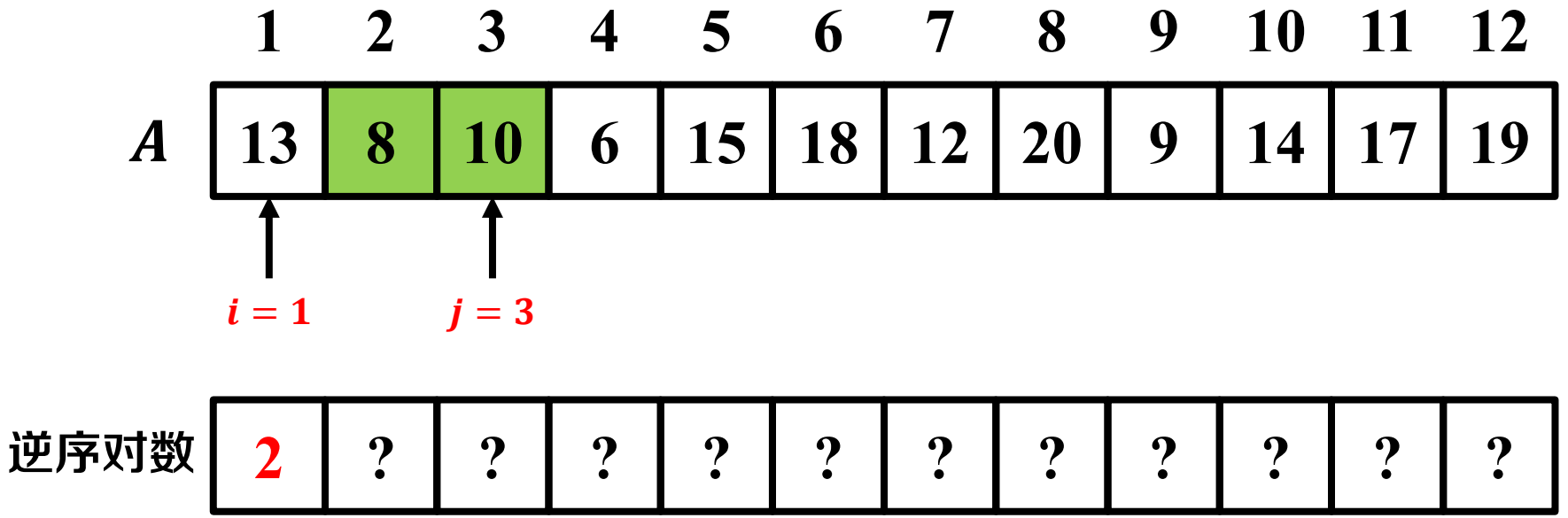
蛮力枚举

- 对于数组的每个元素$A[i]$，枚举$j(j > i)$，并统计逆序对数目



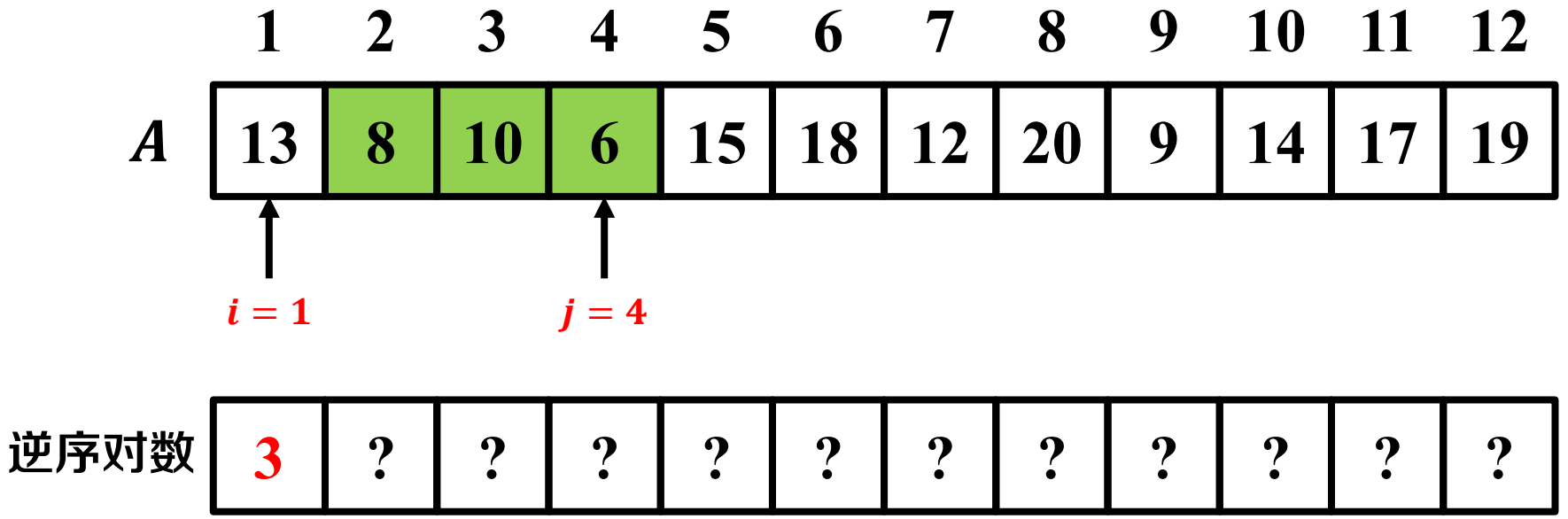
蛮力枚举

- 对于数组的每个元素$A[i]$，枚举$j(j > i)$，并统计逆序对数目



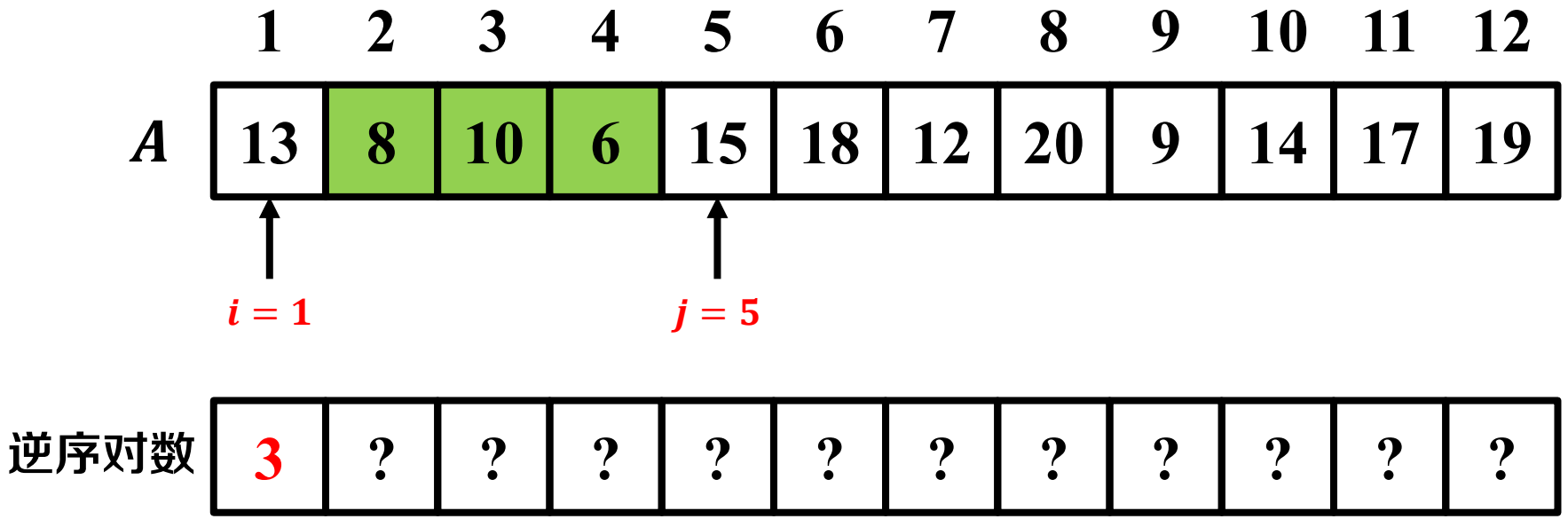
蛮力枚举

- 对于数组的每个元素$A[i]$，枚举$j(j > i)$，并统计逆序对数目



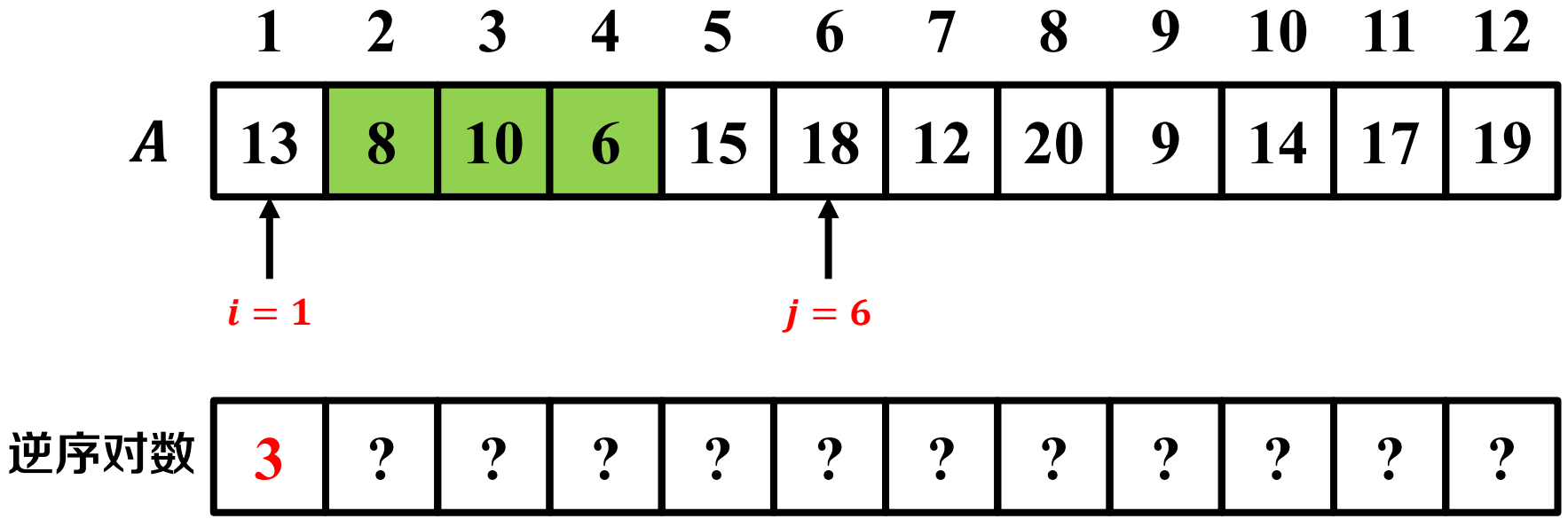
蛮力枚举

- 对于数组的每个元素$A[i]$，枚举$j(j > i)$，并统计逆序对数目



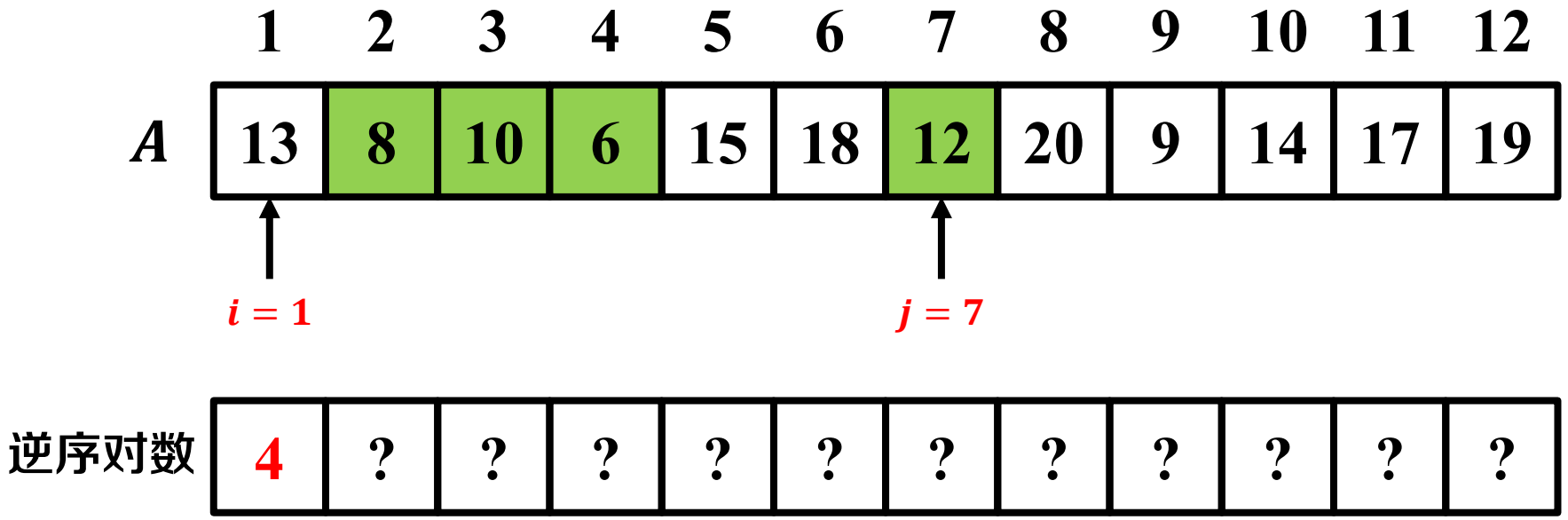
蛮力枚举

- 对于数组的每个元素$A[i]$，枚举$j(j > i)$，并统计逆序对数目



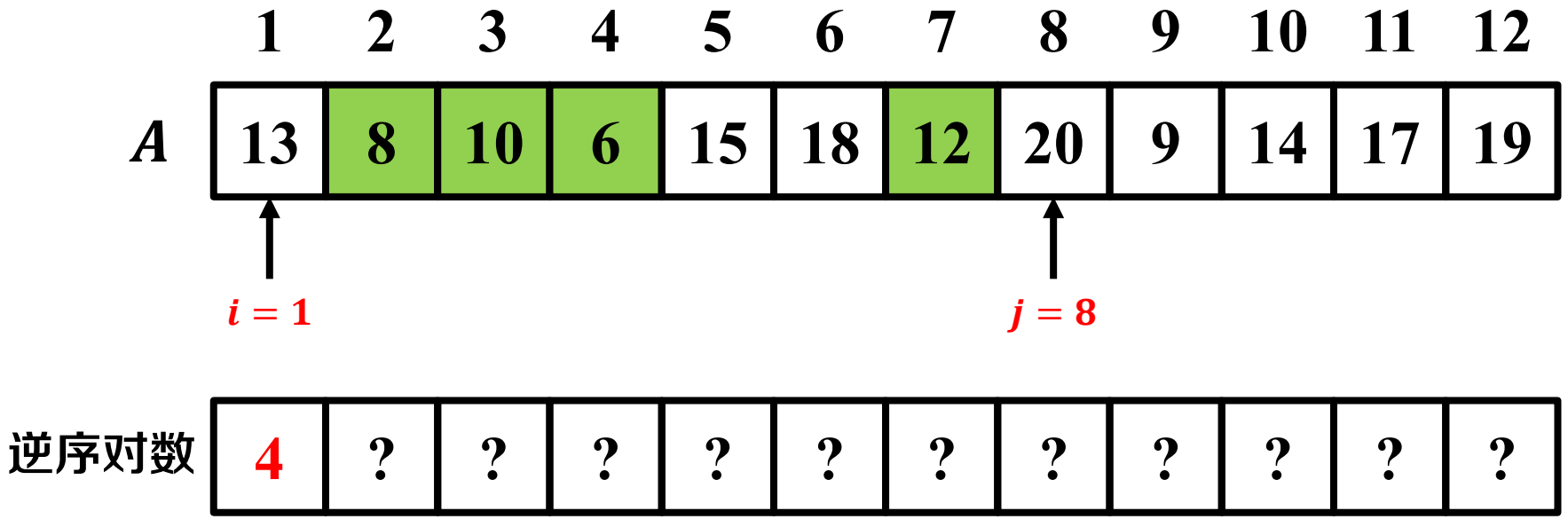
蛮力枚举

- 对于数组的每个元素$A[i]$，枚举$j(j > i)$，并统计逆序对数目



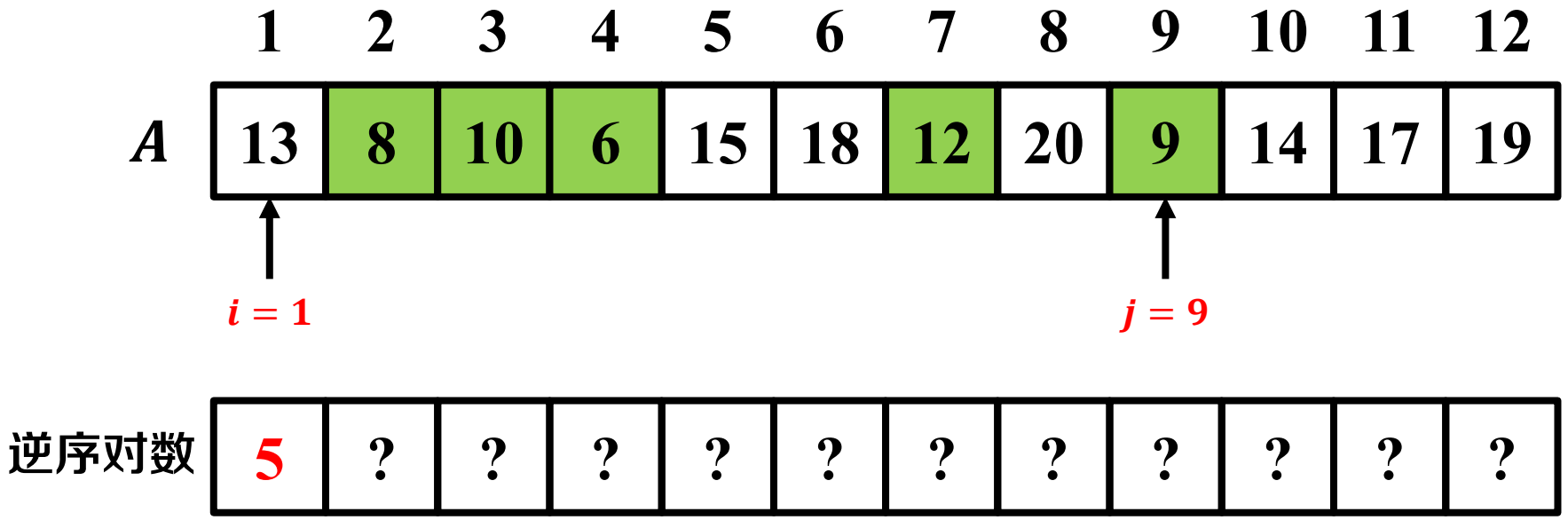
蛮力枚举

- 对于数组的每个元素$A[i]$，枚举$j(j > i)$，并统计逆序对数目



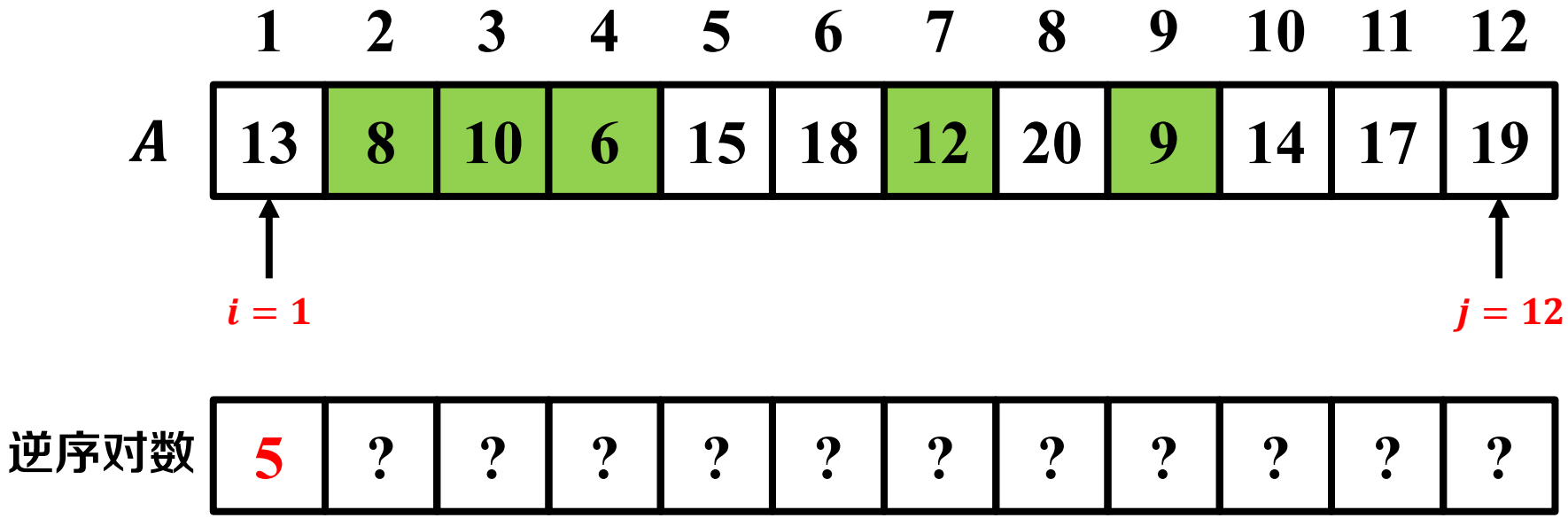
蛮力枚举

- 对于数组的每个元素$A[i]$，枚举$j(j > i)$，并统计逆序对数目



蛮力枚举

- 对于数组的每个元素$A[i]$，枚举$j(j > i)$，并统计逆序对数目



蛮力枚举

- 对于数组的每个元素$A[i]$，枚举$j(j > i)$，并统计逆序对数目

	1	2	3	4	5	6	7	8	9	10	11	12
A	13	8	10	6	15	18	12	20	9	14	17	19

逆序对数	5	1	2	0	3	4	1	4	0	0	0	0

蛮力枚举：伪代码

- 对于数组的每个元素$A[i]$，枚举$j(j > i)$，并统计逆序对数目

```
输入: 一个大小为n的数组A[1..n]
输出: 数组A[1..n]中逆序对数目Sum
Sum ← 0
for i ← 1 to n do
    for j ← i + 1 to n do
        if A[i] > A[j] then
            Sum ← Sum + 1
        end
    end
end
return Sum
```

初始化逆序对数目

蛮力枚举：伪代码

- 对于数组的每个元素$A[i]$，枚举$j(j > i)$，并统计逆序对数目

```
输入: 一个大小为n的数组A[1..n]
输出: 数组A[1..n]中逆序对数目Sum
Sum ← 0
for i ← 1 to n do
    for j ← i + 1 to n do
        if A[i] > A[j] then
            Sum ← Sum + 1
        end
    end
end
return Sum
```

枚举所有下标i

蛮力枚举：伪代码

- 对于数组的每个元素$A[i]$，枚举$j(j > i)$，并统计逆序对数目

输入: 一个大小为n的数组$A[1..n]$
输出: 数组$A[1..n]$中逆序对数目Sum

```
Sum ← 0
for i ← 1 to n do
    for j ← i + 1 to n do
        if A[i] > A[j] then
            Sum ← Sum + 1
        end
    end
end
return Sum
```

枚举右侧元素j
统计逆序对数目

蛮力枚举：伪代码

- 对于数组的每个元素$A[i]$，枚举$j(j > i)$，并统计逆序对数目

```
输入: 一个大小为n的数组A[1..n]
输出: 数组A[1..n]中逆序对数目Sum
Sum ← 0
for i ← 1 to n do
    for j ← i + 1 to n do
        if A[i] > A[j] then
            Sum ← Sum + 1
        end
    end
end
return Sum
```

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

时间复杂度：$O(n^2)$

蛮力枚举：伪代码

- 对于数组的每个元素$A[i]$，枚举$j(j > i)$，并统计逆序对数目

```
输入: 一个大小为n的数组A[1..n]
输出: 数组A[1..n]中逆序对数目Sum
Sum ← 0
for i ← 1 to n do
    for j ← i + 1 to n do
        if A[i] > A[j] then
            Sum ← Sum + 1
        end
    end
end
return Sum
```

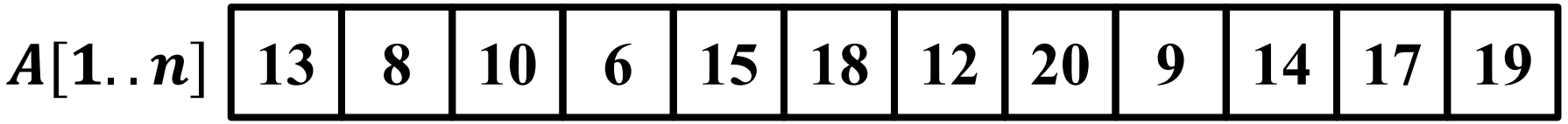
$O(n)$ $O(n^2)$

时间复杂度：$O(n^2)$

问题：如何进一步提高算法效率？能否采用分而治之策略？

分而治之：一般步骤

分而治之：算法框架

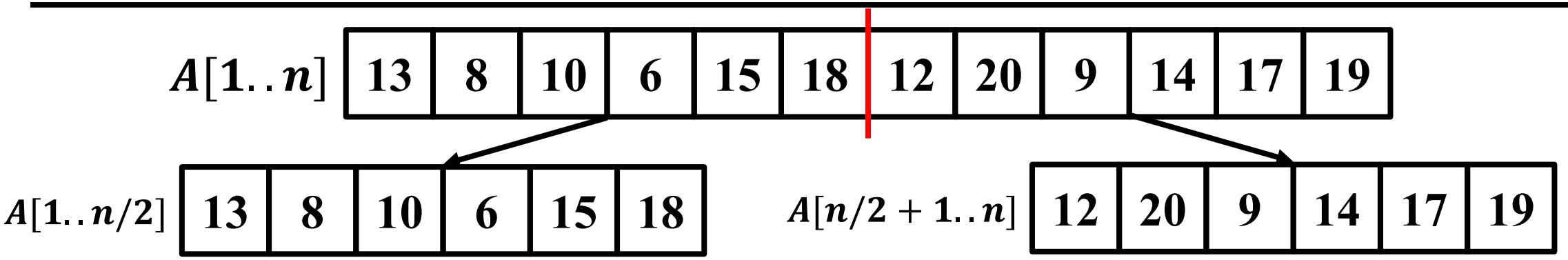


分解原问题

解决子问题

合并问题解

分而治之：算法框架



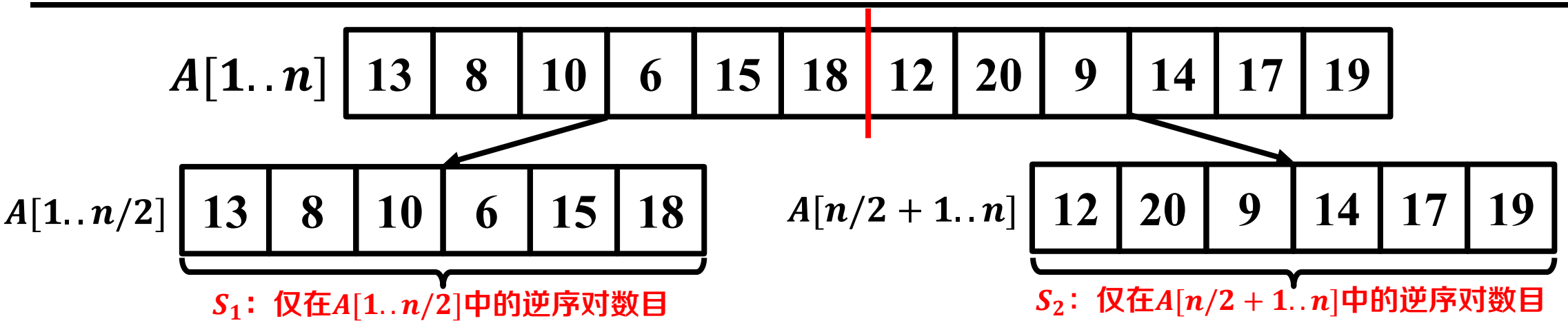
- 把数组A二分为两个子数组$A[1..n/2], A[n/2+1..n]$

分解原问题

解决子问题

合并问题解

分而治之：算法框架



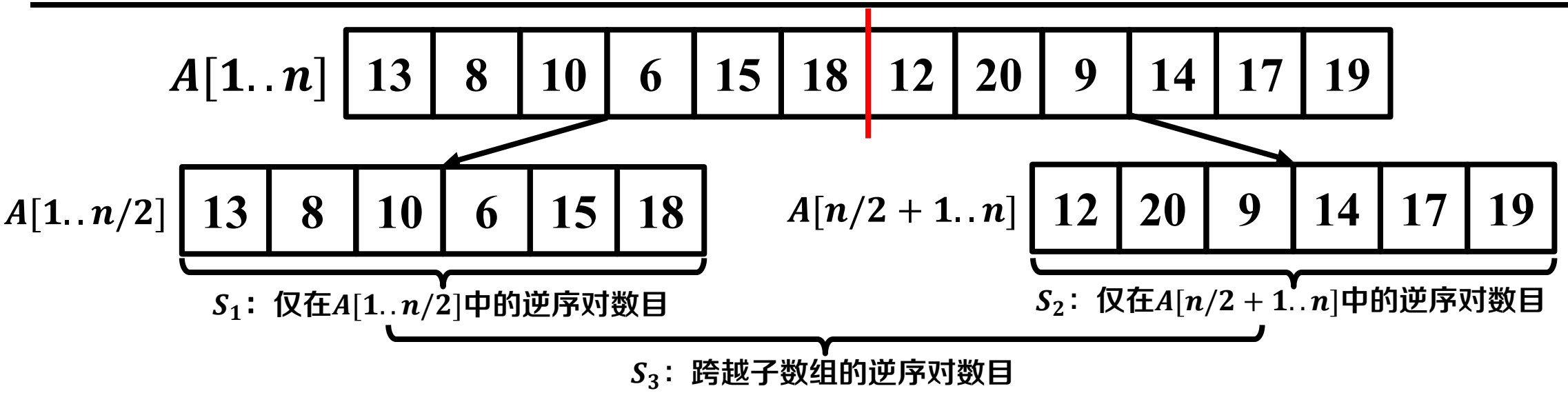
- 把数组A二分为两个子数组$A[1..n/2], A[n/2+1..n]$
- 递归求解子问题
  - 求解S_1：仅在$A[1..n/2]$中的逆序对数目
  - 求解S_2：仅在$A[n/2+1..n]$中的逆序对数目

分解原问题

解决子问题

合并问题解

分而治之：算法框架



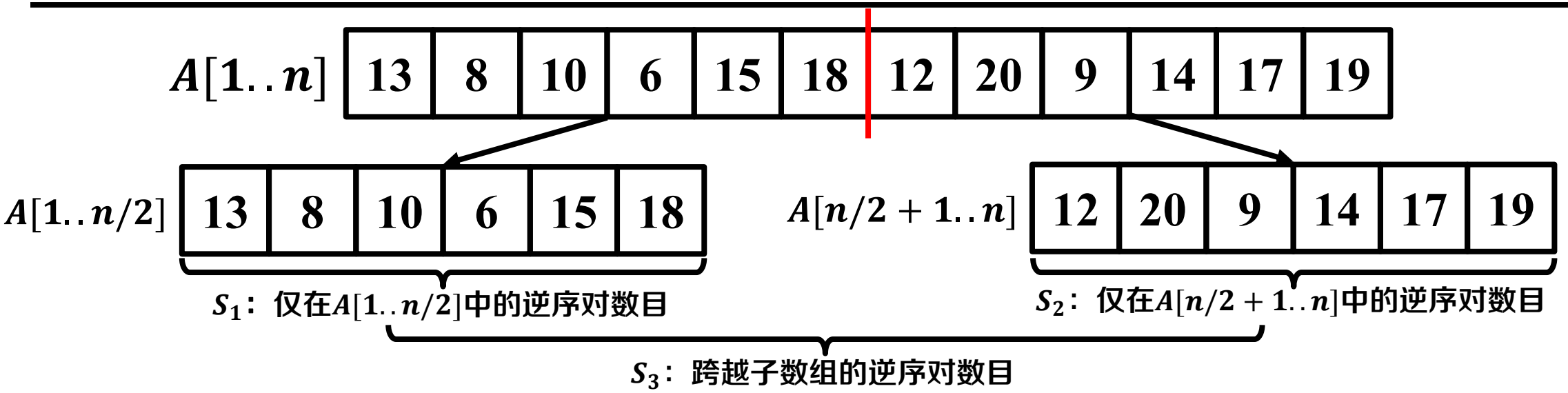
- 把数组A二分为两个子数组$A[1..n/2], A[n/2+1..n]$
- 递归求解子问题
  - 求解S_1：仅在$A[1..n/2]$中的逆序对数目
  - 求解S_2：仅在$A[n/2+1..n]$中的逆序对数目
- 合并$A[1..n/2]$和$A[n/2+1..n]$的解
  - 求解S_3：跨越子数组的逆序对数目

分解原问题

解决子问题

合并问题解

分而治之：算法框架



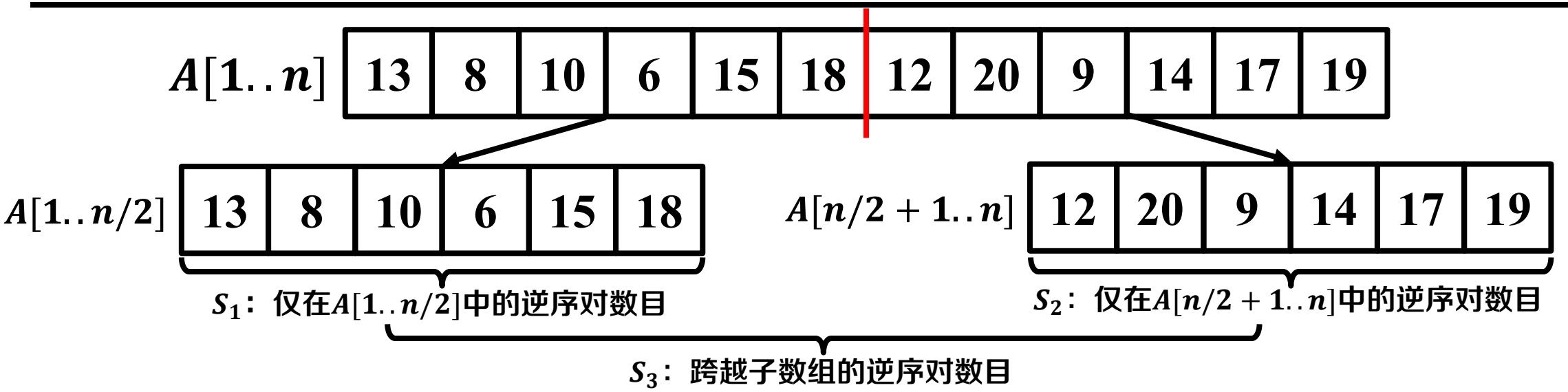
- 把数组A二分为两个子数组$A[1..n/2], A[n/2+1..n]$
- 递归求解子问题
  - 求解S_1：仅在$A[1..n/2]$中的逆序对数目
  - 求解S_2：仅在$A[n/2+1..n]$中的逆序对数目
- 合并$A[1..n/2]$和$A[n/2+1..n]$的解
  - 求解S_3：跨越子数组的逆序对数目
  - $S = S_1 + S_2 + S_3$

分解原问题

解决子问题

合并问题解

分而治之：算法框架



- 把数组A二分为两个子数组$A[1..n/2], A[n/2+1..n]$
- 递归求解子问题
  - 求解S_1：仅在$A[1..n/2]$中的逆序对数目
  - 求解S_2：仅在$A[n/2+1..n]$中的逆序对数目
- 合并$A[1..n/2]$和$A[n/2+1..n]$的解
  - 求解S_3：跨越子数组的逆序对数目
  - $S = S_1 + S_2 + S_3$

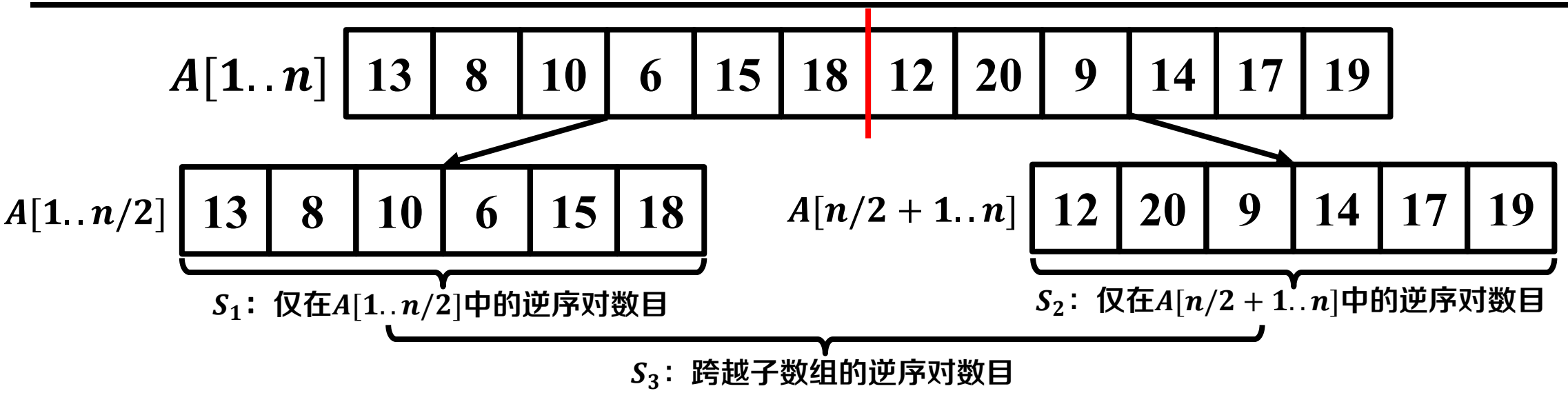
分解原问题

可递归求解

解决子问题

合并问题解

分而治之：算法框架



- 把数组A二分为两个子数组$A[1..n/2], A[n/2+1..n]$ — 分解原问题
- 递归求解子问题 — 解决子问题
  - 求解S_1：仅在$A[1..n/2]$中的逆序对数目
  - 求解S_2：仅在$A[n/2+1..n]$中的逆序对数目

  } 可递归求解
- 合并$A[1..n/2]$和$A[n/2+1..n]$的解 — 合并问题解
  - 求解S_3：跨越子数组的逆序对数目
  - $S = S_1 + S_2 + S_3$

问题：如何求解S_3？

合并问题解：求解S_3

- 策略一：直接求解
  - 对每个$A[j] \in A[m+1..n]$，枚举$A[i] \in A[1..m]$并统计逆序对数目

数组$A[1..m]$

13	8	10	6	15	18

数组$A[m+1..n]$

12	20	9	14	17	19

?	?	?	?	?	?

逆序对数

合并问题解：求解S_3

- 策略一：直接求解
  - 对每个$A[j] \in A[m+1..n]$，枚举$A[i] \in A[1..m]$并统计逆序对数目

数组$A[1..m]$

13	8	10	6	15	18

数组$A[m+1..n]$

12	20	9	14	17	19

$\{(13,12),(15,12),(18,12)\}$

3	?	?	?	?	?

逆序对数

合并问题解：求解S_3

- 策略一：直接求解
  - 对每个$A[j] \in A[m+1..n]$，枚举$A[i] \in A[1..m]$并统计逆序对数目

数组$A[1..m]$

13	8	10	6	15	18

$\emptyset$

数组$A[m+1..n]$

12	20	9	14	17	19

3	0	?	?	?	?

逆序对数

合并问题解：求解S_3

- 策略一：直接求解
  - 对每个$A[j] \in A[m+1..n]$，枚举$A[i] \in A[1..m]$并统计逆序对数目

数组$A[1..m]$

13	8	10	6	15	18

数组$A[m+1..n]$

12	20	9	14	17	19

$\{(13,9),(10,9),(15,9),(18,9)\}$

3	0	4	?	?	?

逆序对数

合并问题解：求解S_3

- 策略一：直接求解
  - 对每个$A[j] \in A[m+1..n]$，枚举$A[i] \in A[1..m]$并统计逆序对数目

数组$A[1..m]$

13	8	10	6	15	18

数组$A[m+1..n]$

12	20	9	14	17	19

$\{(15,14),(18,14)\}$

3	0	4	2	?	?

逆序对数

合并问题解：求解S_3

- 策略一：直接求解
  - 对每个$A[j] \in A[m+1..n]$，枚举$A[i] \in A[1..m]$并统计逆序对数目

数组$A[1..m]$

13	8	10	6	15	18

数组$A[m+1..n]$

12	20	9	14	17	19

$\{(18, 17)\}$

3	0	4	2	1	?

逆序对数

合并问题解：求解S_3

- 策略一：直接求解
  - 对每个$A[j] \in A[m+1..n]$，枚举$A[i] \in A[1..m]$并统计逆序对数目

数组$A[1..m]$

13	8	10	6	15	18

$\emptyset$

数组$A[m+1..n]$

12	20	9	14	17	19

3	0	4	2	1	0

逆序对数

合并问题解：求解S_3

- 策略一：直接求解
  - 对每个$A[j] \in A[m+1..n]$，枚举$A[i] \in A[1..m]$并统计逆序对数目

数组$A[1..m]$

13	8	10	6	15	18

数组$A[m+1..n]$

12	20	9	14	17	19

逆序对总数：

$3+4+2+1=10$

3	0	4	2	1	0

逆序对数

合并问题解：求解S_3

- 策略一：直接求解
  - 对每个$A[j] \in A[m+1..n]$，枚举$A[i] \in A[1..m]$并统计逆序对数目
  - 求解S_3的算法运行时间：$O(n^2)$

数组$A[1..m]$

13	8	10	6	15	18

数组$A[m+1..n]$

12	20	9	14	17	19

逆序对总数：

$3+4+2+1=10$

3	0	4	2	1	0

逆序对数

合并问题解：求解S_3

- 策略一：直接求解
  - 对每个$A[j] \in A[m+1..n]$，枚举$A[i] \in A[1..m]$并统计逆序对数目
  - 求解S_3的算法运行时间：$O(n^2)$
  - 分而治之框架的算法运行时间：$T(n) = 2 \cdot T(n/2) + O(n^2)$ ⟶ $O(n^2)$

数组$A[1..m]$

13	8	10	6	15	18

数组$A[m+1..n]$

12	20	9	14	17	19

逆序对总数：

$3 + 4 + 2 + 1 = 10$

3	0	4	2	1	0

逆序对数

合并问题解：求解S_3

- 策略一：直接求解
  - 对每个$A[j] \in A[m+1..n]$，枚举$A[i] \in A[1..m]$并统计逆序对数目
  - 求解S_3的算法运行时间：$O(n^2)$
  - 分而治之框架的算法运行时间：$T(n) = 2 \cdot T(n/2) + O(n^2)$ ⟶ $O(n^2)$

数组$A[1..m]$

13	8	10	6	15	18

数组$A[m+1..n]$

12	20	9	14	17	19

逆序对总数：

$3 + 4 + 2 + 1 = 10$

3	0	4	2	1	0

逆序对数

直接求解S_3的分而治之较蛮力枚举**并未提高**算法运行时间！

!!!

合并问题解：求解S_3

- 致因分析
  - 运行时间受制于跨越子数组的逆序对计数方法

数组$A[1..m]$

13	8	10	6	15	18

数组$A[m+1..n]$

12	20	9	14	17	19

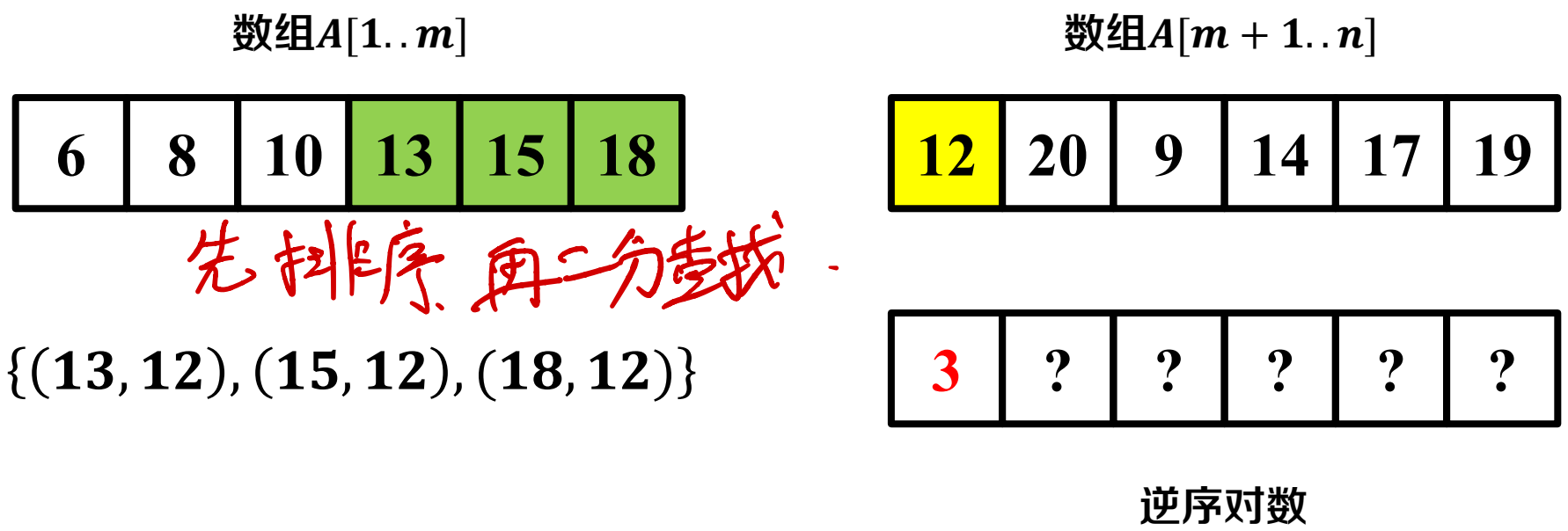
$\{(13,12),(15,12),(18,12)\}$

3	?	?	?	?	?

逆序对数

合并问题解：求解S_3

- 致因分析
  - 运行时间受制于跨越子数组的逆序对计数方法
  - 数组的有序性通常有助于提高算法的运行时间



合并问题解：求解S_3

- 策略二：排序求解

数组$A[1..m]$

13	8	10	6	15	18

数组$A[m+1..n]$

12	20	9	14	17	19

逆序对数：

?	?	?	?	?	?

合并问题解：求解S_3

- 策略二：排序求解
  - 分别对数组$A[1..m]$和$A[m+1..n]$进行**排序**

数组$A[1..m]$

6	8	10	13	15	18

数组$A[m+1..n]$

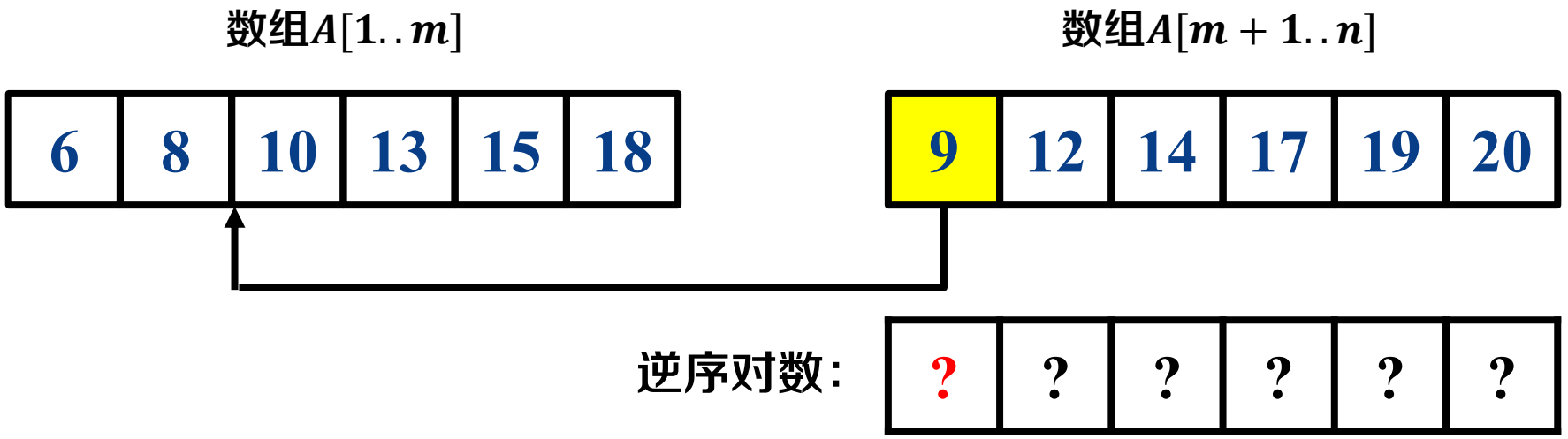
9	12	14	17	19	20

逆序对数：

?	?	?	?	?	?

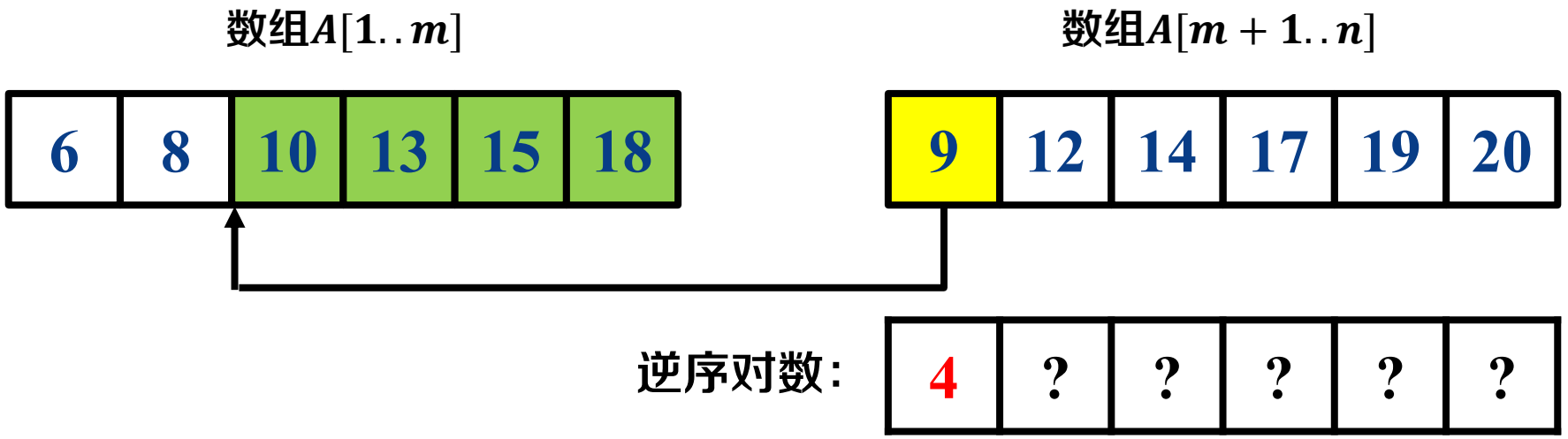
合并问题解：求解S_3

- 策略二：排序求解
  - 分别对数组$A[1..m]$和$A[m+1..n]$进行排序
  - 对于每个$A[j] \in A[m+1..n]$，采用二分查找为其在$A[1..m]$中定位



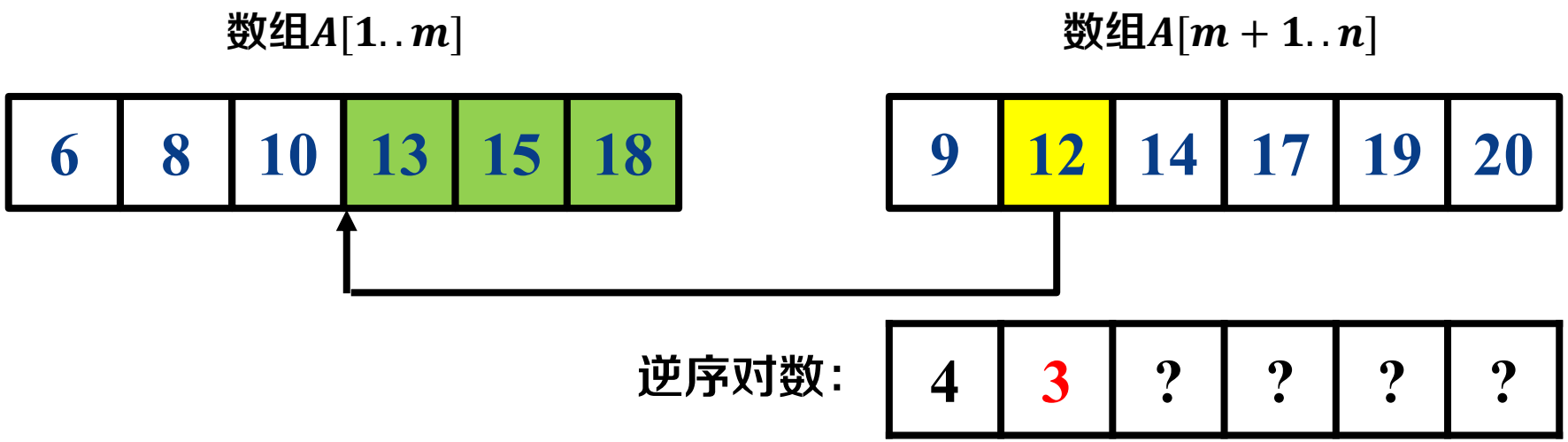
合并问题解：求解S_3

- 策略二：排序求解
  - 分别对数组$A[1..m]$和$A[m+1..n]$进行排序
  - 对于每个$A[j] \in A[m+1..n]$，采用二分查找为其在$A[1..m]$中定位
  - $A[j]$在$A[1..m]$定位点右侧的元素均可与$A[j]$构成逆序对



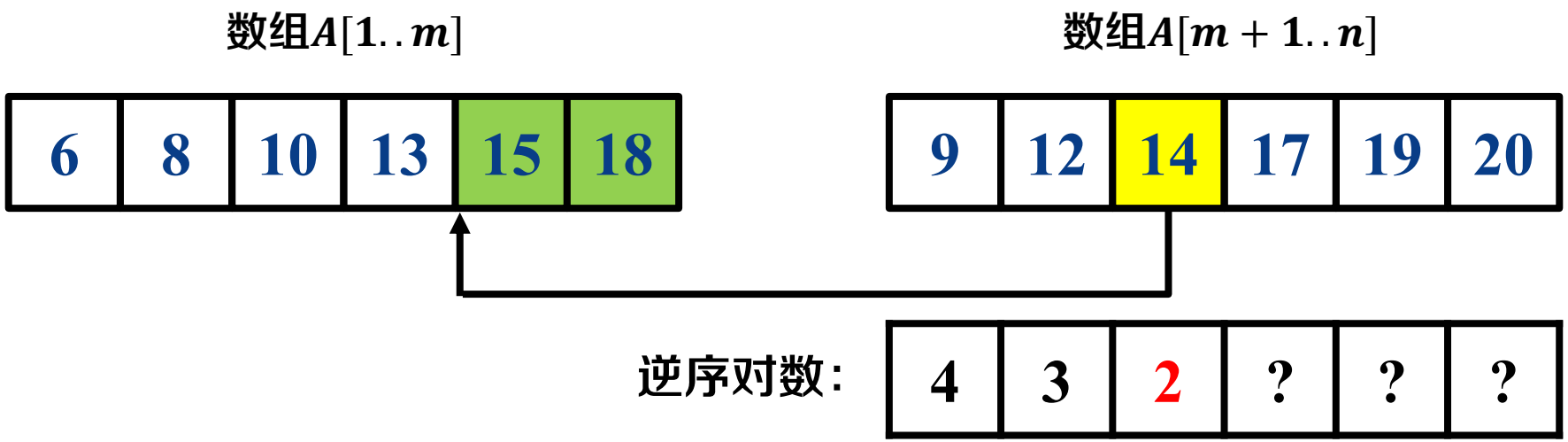
合并问题解：求解S_3

- 策略二：排序求解
  - 分别对数组$A[1..m]$和$A[m+1..n]$进行**排序**
  - 对于每个$A[j] \in A[m+1..n]$，采用**二分查找**为其在$A[1..m]$中定位
  - **$A[j]$在$A[1..m]$定位点右侧的元素**均可与$A[j]$构成逆序对



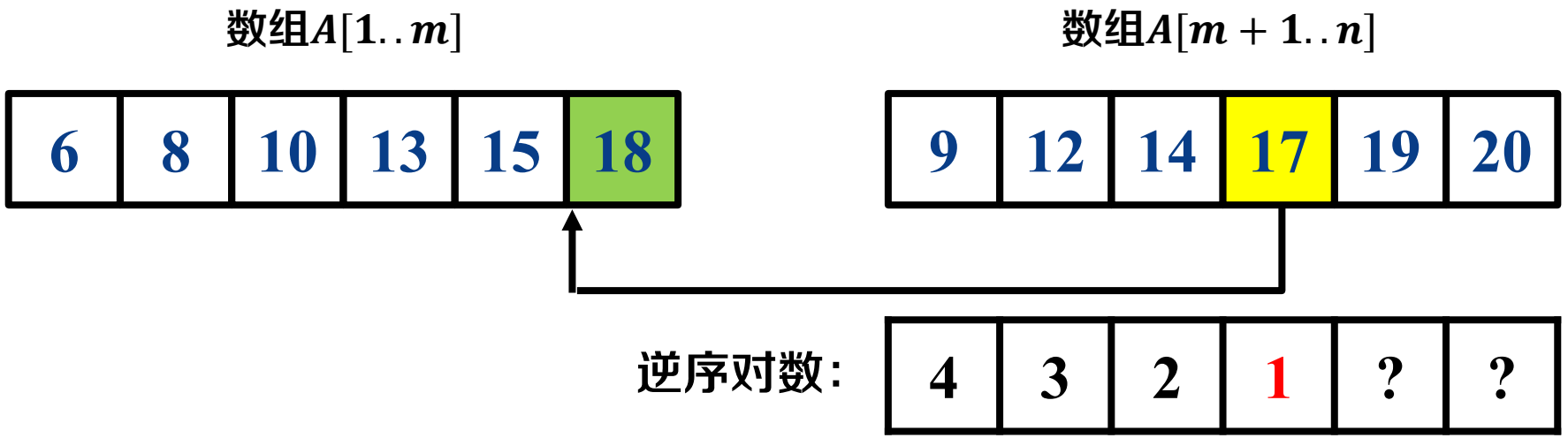
合并问题解：求解S_3

- 策略二：排序求解
  - 分别对数组$A[1..m]$和$A[m+1..n]$进行排序
  - 对于每个$A[j] \in A[m+1..n]$，采用二分查找为其在$A[1..m]$中定位
  - $A[j]$在$A[1..m]$定位点右侧的元素均可与$A[j]$构成逆序对



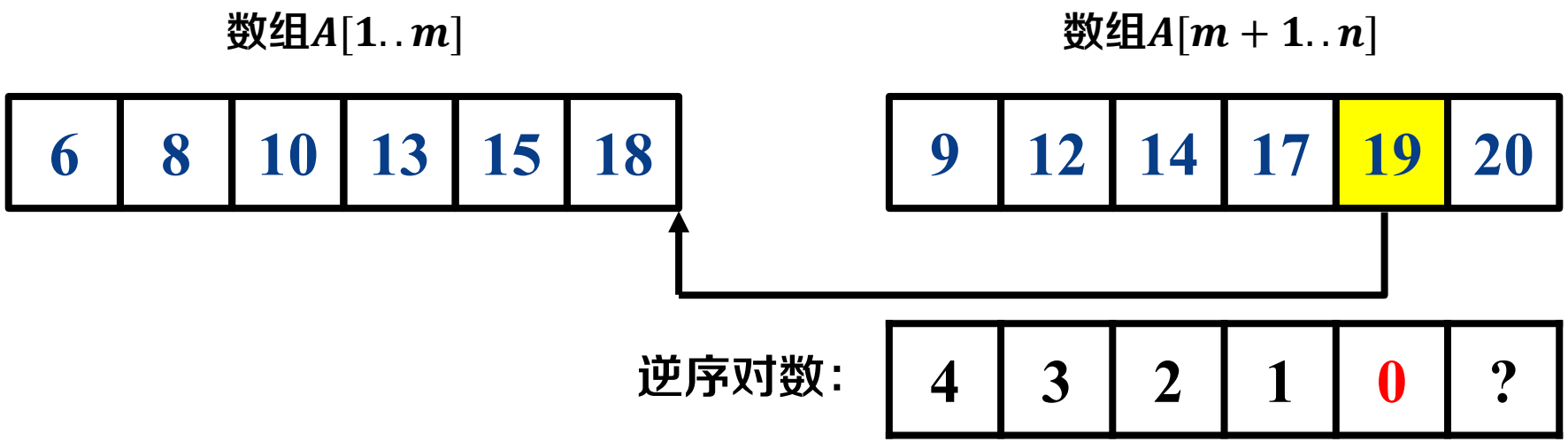
合并问题解：求解S_3

- 策略二：排序求解
  - 分别对数组$A[1..m]$和$A[m+1..n]$进行**排序**
  - 对于每个$A[j] \in A[m+1..n]$，采用**二分查找**为其在$A[1..m]$中定位
  - **$A[j]$在$A[1..m]$定位点右侧的元素**均可与$A[j]$构成逆序对



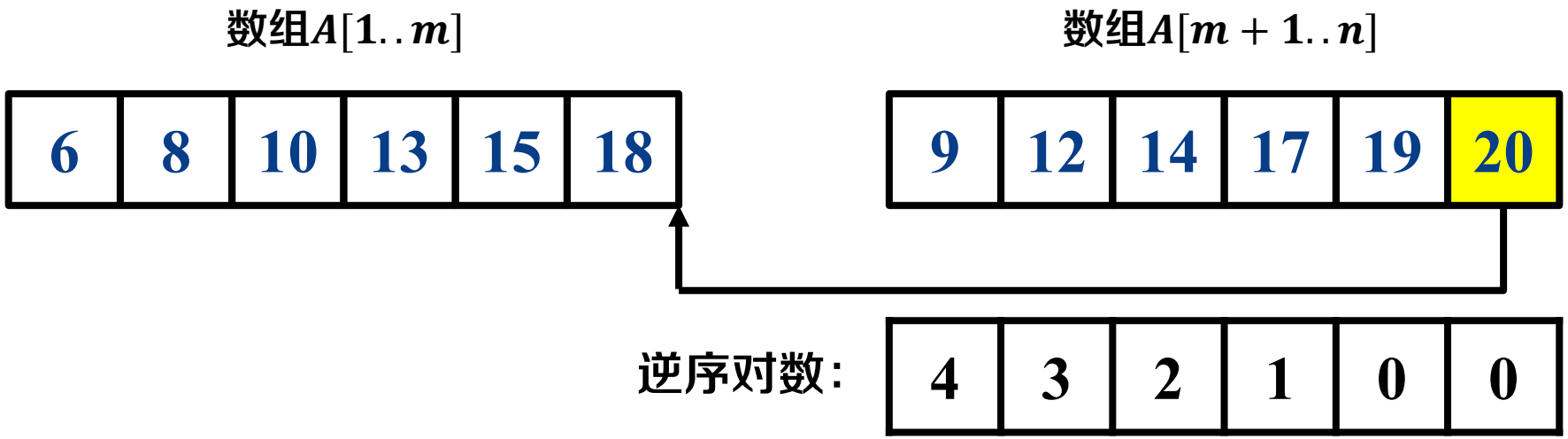
合并问题解：求解S_3

- 策略二：排序求解
  - 分别对数组$A[1..m]$和$A[m+1..n]$进行排序
  - 对于每个$A[j] \in A[m+1..n]$，采用二分查找为其在$A[1..m]$中定位
  - $A[j]$在$A[1..m]$定位点右侧的元素均可与$A[j]$构成逆序对



合并问题解：求解S_3

- 策略二：排序求解
  - 分别对数组$A[1..m]$和$A[m+1..n]$进行排序
  - 对于每个$A[j] \in A[m+1..n]$，采用二分查找为其在$A[1..m]$中定位
  - $A[j]$在$A[1..m]$定位点右侧的元素均可与$A[j]$构成逆序对



合并问题解：求解S_3

- 策略二：排序求解
  - 分别对数组$A[1..m]$和$A[m+1..n]$进行**排序**
  - 对于每个$A[j] \in A[m+1..n]$，采用**二分查找**为其在$A[1..m]$中定位
  - **$A[j]$在$A[1..m]$定位点右侧的元素**均可与$A[j]$构成逆序对
  - 求解S_3的算法运行时间：$O(n \log n)$

} $\frac{n}{2}\log\frac{n}{2}$

合并问题解：求解S_3

- 策略二：排序求解
  - 分别对数组$A[1..m]$和$A[m+1..n]$进行**排序**
  - 对于每个$A[j] \in A[m+1..n]$，采用**二分查找**为其在$A[1..m]$中定位 } $\frac{n}{2}\log\frac{n}{2}$
  - **$A[j]$在$A[1..m]$定位点右侧的元素**均可与$A[j]$构成逆序对
  - 求解S_3的算法运行时间：$O(n\log n)$
  - 分治框架的算法运行时间：$T(n)=2T(n/2)+O(n\log n)$ ⟶ $O(n\log^2 n)$

扩展的主定理求解

比$O(n^2)$优

排序求解S_3的分而治之提高了算法运行时间，是否还有优化可能？

合并问题解：求解S_3

- 算法优化
  - 排序和二分查找均无再优化空间

子数组排序的运行时间：$O(n \log n)$
二分查找的运行时间：$O(n \log n)$

合并问题解：求解S_3

- 算法优化
  - 排序和二分查找均无再优化空间
- 致因分析
  - 未将排序过程融入整个算法框架

子数组排序的运行时间：$O(n\log n)$
二分查找的运行时间：$O(n\log n)$

13	8	10

6	15	18

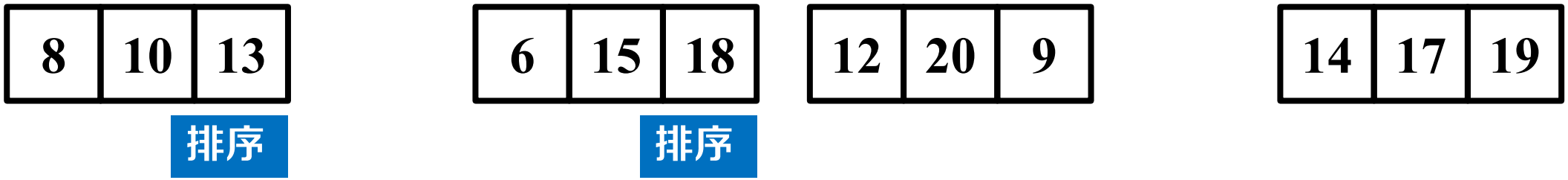
12	20	9

14	17	19

合并问题解：求解S_3

- 算法优化
  - 排序和二分查找均无再优化空间
- 致因分析
  - 未将排序过程融入整个算法框架

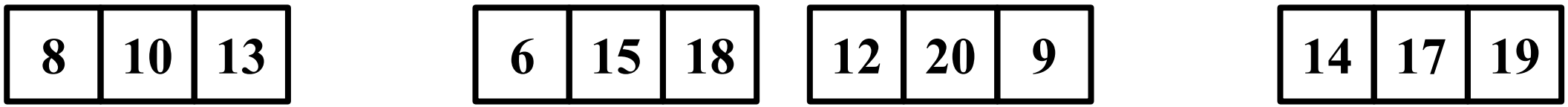
子数组排序的运行时间：$O(n\log n)$
二分查找的运行时间：$O(n\log n)$



合并问题解：求解S_3

- 算法优化
  - 排序和二分查找均无再优化空间
- 致因分析
  - 未将排序过程融入整个算法框架

子数组排序的运行时间：$O(n\log n)$
二分查找的运行时间：$O(n\log n)$

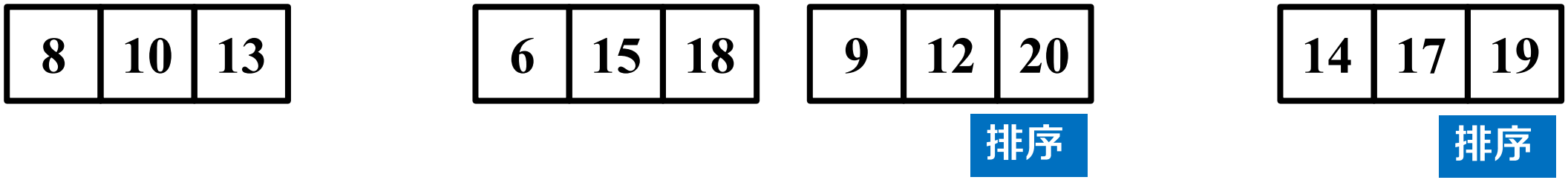


二分查找

合并问题解：求解S_3

- 算法优化
  - 排序和二分查找均无再优化空间
- 致因分析
  - 未将排序过程融入整个算法框架

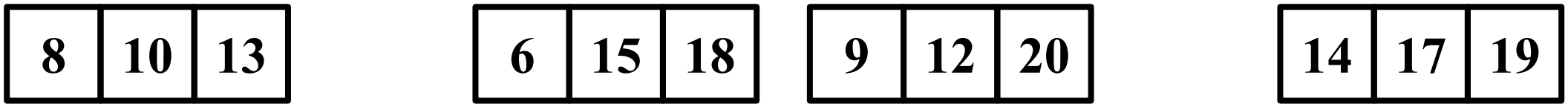
子数组排序的运行时间：$O(n\log n)$
二分查找的运行时间：$O(n\log n)$



合并问题解：求解S_3

- 算法优化
  - 排序和二分查找均无再优化空间
- 致因分析
  - 未将排序过程融入整个算法框架

子数组排序的运行时间：$O(n\log n)$
二分查找的运行时间：$O(n\log n)$

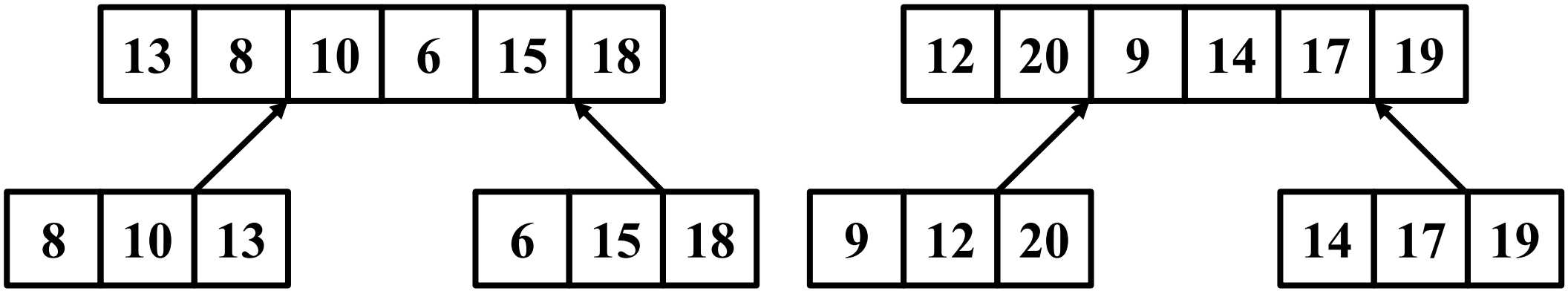


二分查找

合并问题解：求解S_3

- 算法优化
  - 排序和二分查找均无再优化空间
- 致因分析
  - 未将排序过程融入整个算法框架

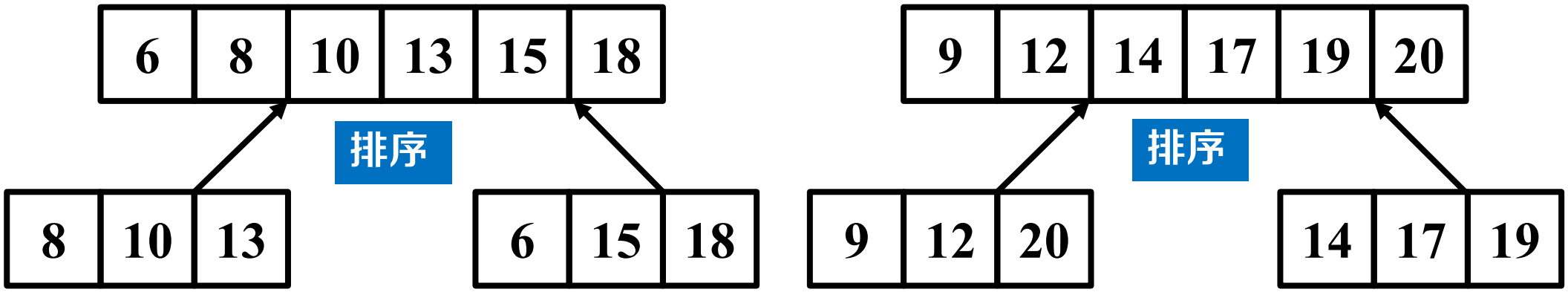
子数组排序的运行时间：$O(n\log n)$
二分查找的运行时间：$O(n\log n)$



合并问题解：求解S_3

- 算法优化
  - 排序和二分查找均无再优化空间
- 致因分析
  - 未将排序过程融入整个算法框架

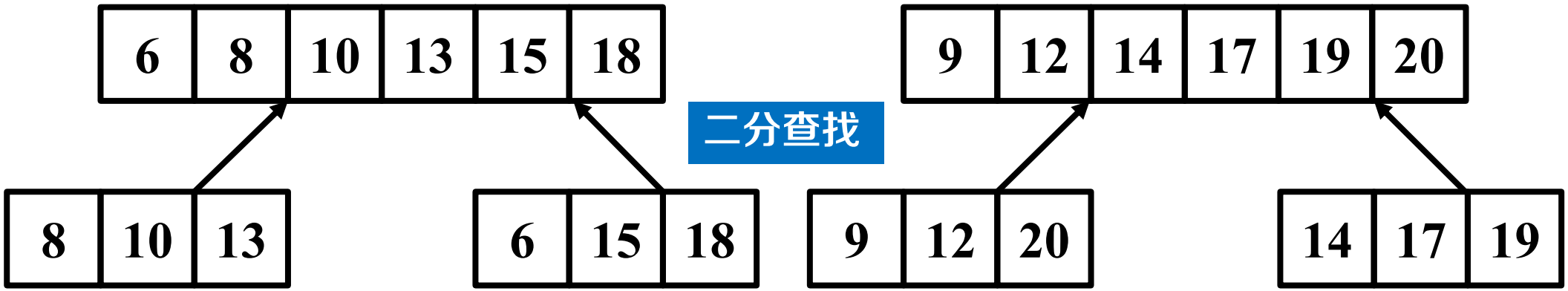
子数组排序的运行时间：$O(n\log n)$
二分查找的运行时间：$O(n\log n)$



合并问题解：求解S_3

- 算法优化
  - 排序和二分查找均无再优化空间
- 致因分析
  - 未将排序过程融入整个算法框架

子数组排序的运行时间：$O(n\log n)$
二分查找的运行时间：$O(n\log n)$

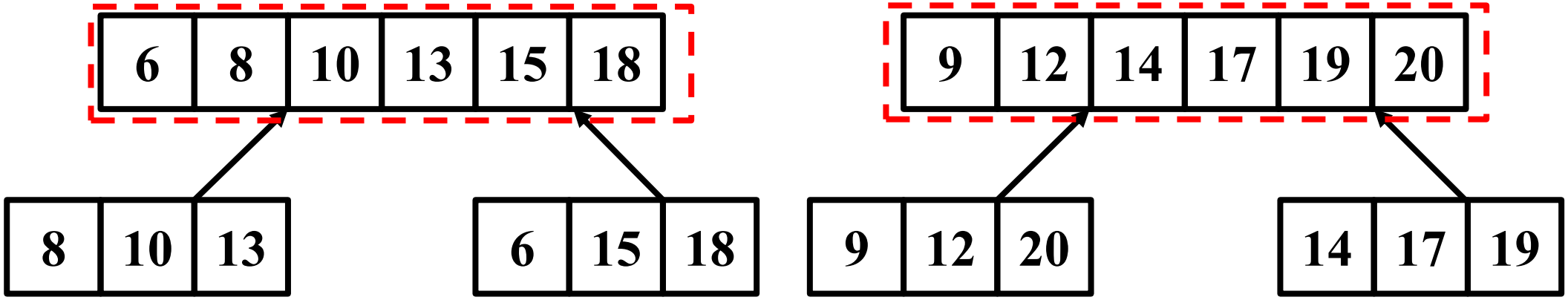


合并问题解：求解S_3

- 算法优化
  - 排序和二分查找均无再优化空间
- 致因分析
  - 未将排序过程融入整个算法框架

子数组排序的运行时间：$O(n\log n)$
二分查找的运行时间：$O(n\log n)$

排序未利用子数组有序性质！



问题：如何将排序过程融入算法框架?

合并问题解：求解S_3

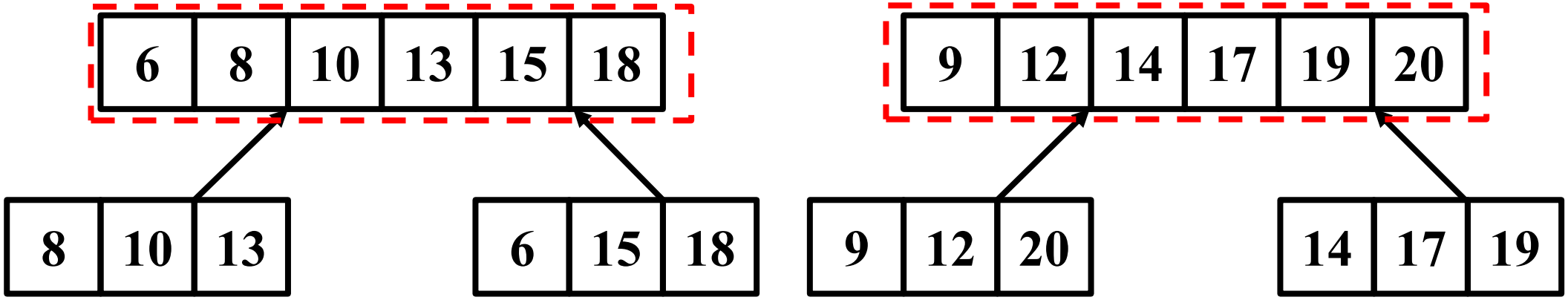
- 算法优化
  - 排序和二分查找均无再优化空间
- 致因分析
  - 未将排序过程融入整个算法框架

子数组排序的运行时间：$O(n\log n)$
二分查找的运行时间：$O(n\log n)$

可用归并排序优化父数组生成策略

可将时间优化到$O(n)$

排序未利用子数组有序性质！



问题：如何将排序过程融入算法框架？归并排序！

合并问题解：求解S_3

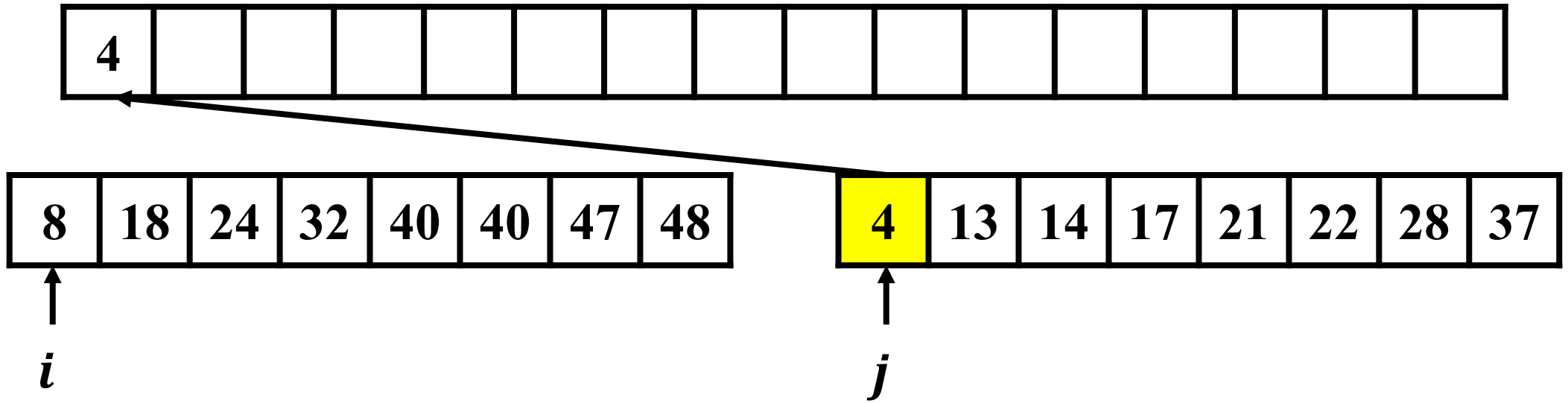
- 算法优化
  - 合并问题解的同时对数组进行排序

子数组排序的运行时间：$O(n)$
二分查找的运行时间：？

合并问题解：求解S_3

- 算法优化
  - 合并问题解的同时对数组进行排序
- 规律发现
  - 归并过程中可同时计算逆序对数目

子数组排序的运行时间：$O(n)$
二分查找的运行时间：？

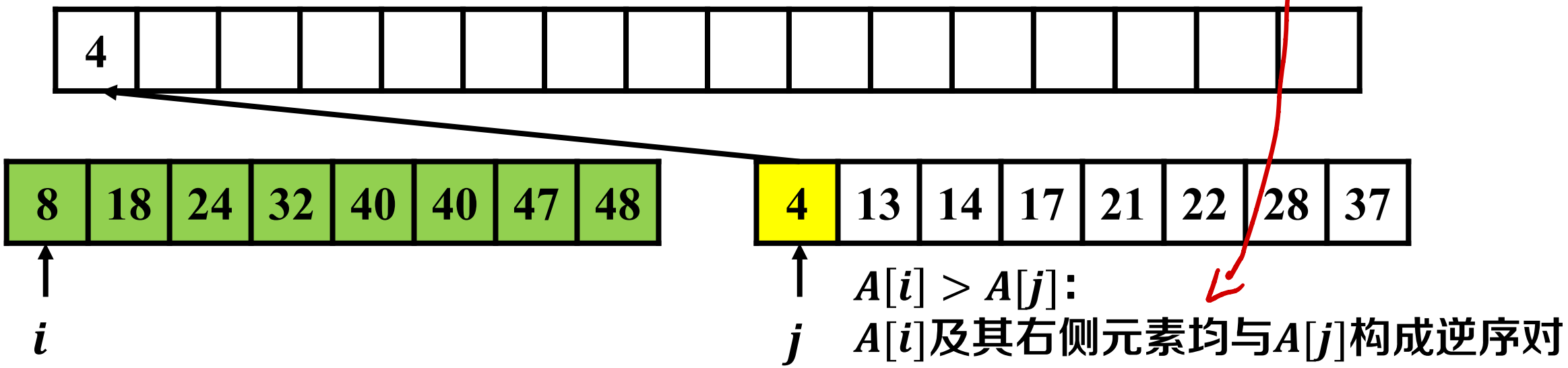


合并问题解：求解S_3

- 算法优化
  - 合并问题解的同时对数组进行排序
- 规律发现
  - 归并过程中可同时计算逆序对数目

子数组排序的运行时间：$O(n)$
二分查找的运行时间：？

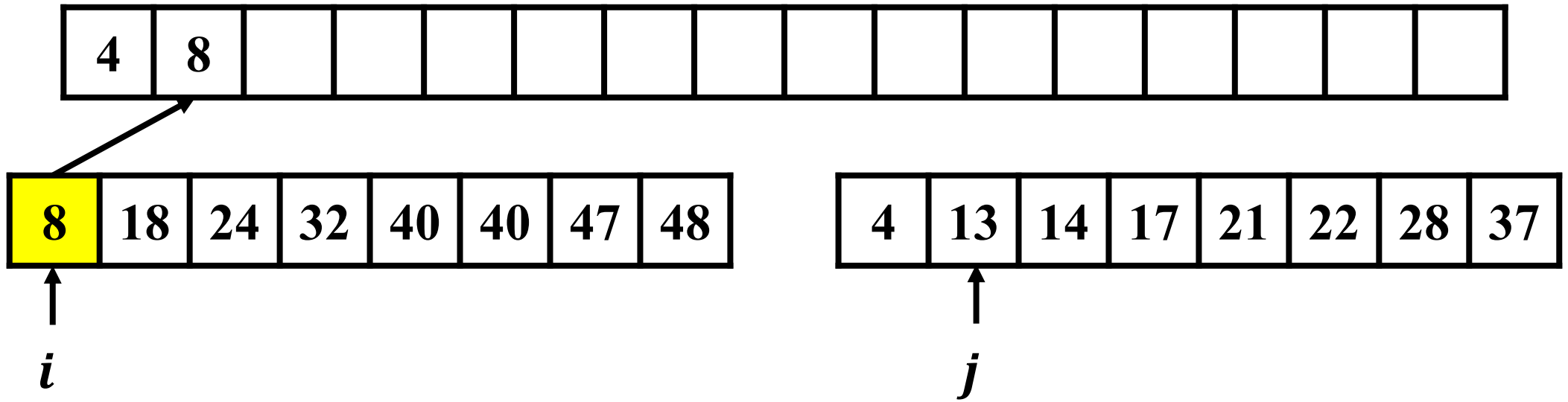
归并过程中无形中解决逆序对计数问题



合并问题解：求解S_3

- 算法优化
  - 合并问题解的同时对数组进行排序
- 规律发现
  - 归并过程中可同时计算逆序对数目

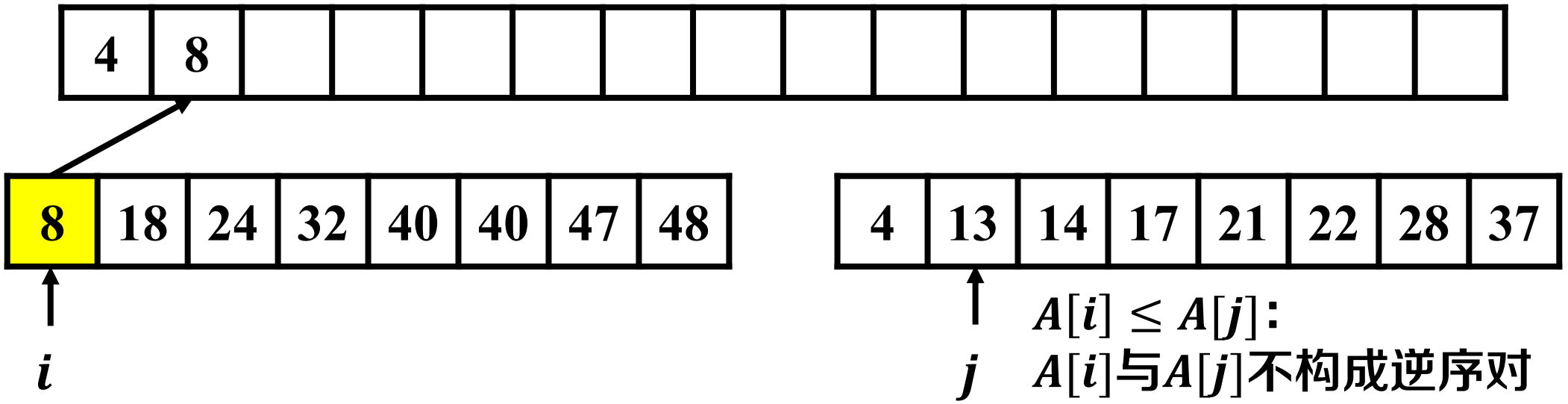
子数组排序的运行时间：$O(n)$
二分查找的运行时间：？



合并问题解：求解S_3

- 算法优化
  - 合并问题解的同时对数组进行排序
- 规律发现
  - 归并过程中可同时计算逆序对数目

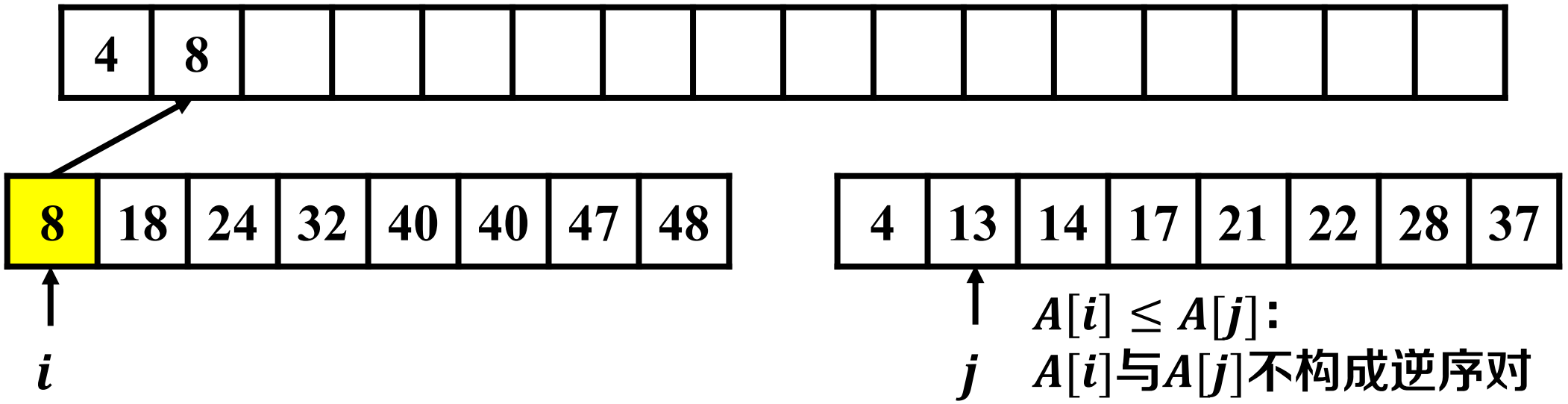
子数组排序的运行时间：$O(n)$
二分查找的运行时间：？



合并问题解：求解S_3

- 算法优化
  - 合并问题解的同时对数组进行排序
- 规律发现
  - 归并过程中可同时计算逆序对数目

子数组排序的运行时间：$O(n)$
二分查找的运行时间：？

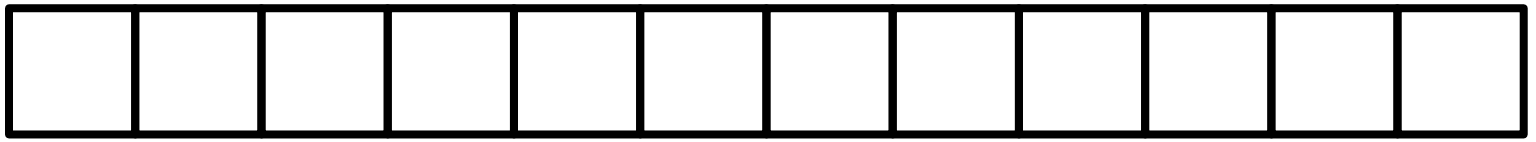


合并问题解：求解S_3

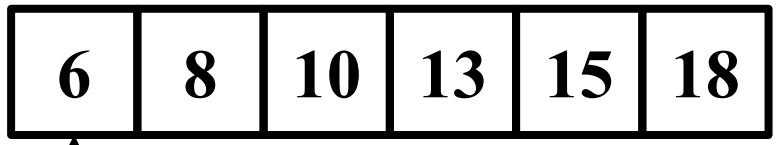
- 策略三：归并求解
  - 从左到右扫描有序子数组：$A[i] \in A[1..m]$，$A[j] \in A[m+1..n]$
    - 如果$A[i] > A[j]$，统计逆序对，j向右移
    - 如果$A[i] \leq A[j]$，i向右移
  - 利用归并排序框架保证合并后数组的有序性

合并问题解：求解S_3

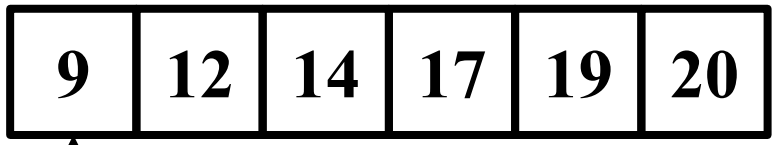
- 策略三：归并求解



子数组:



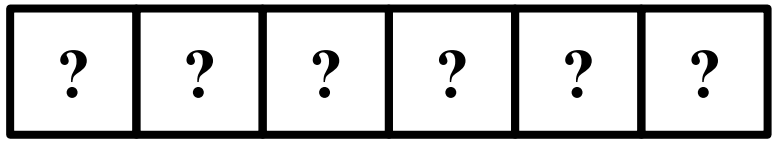
$i = 1$



$j = 7$

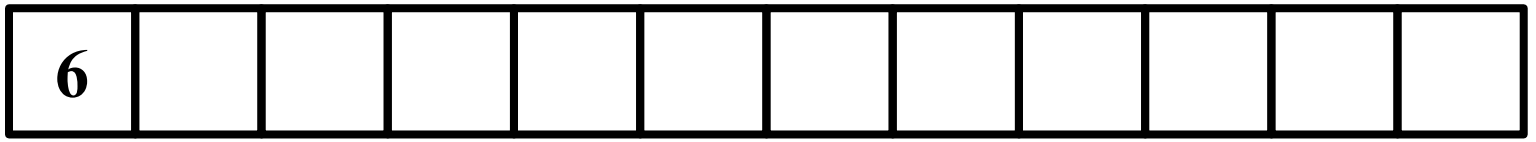
扫描子数组：
若$A[i] > A[j]$，统计逆序对，j向右移
若$A[i] \leq A[j]$，i向右移

逆序对数：

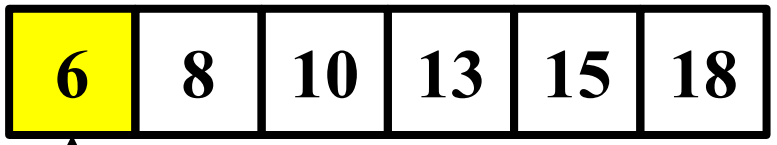


合并问题解：求解S_3

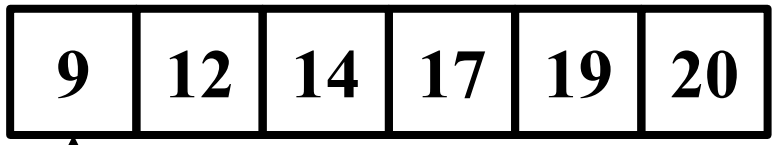
- 策略三：归并求解



子数组:



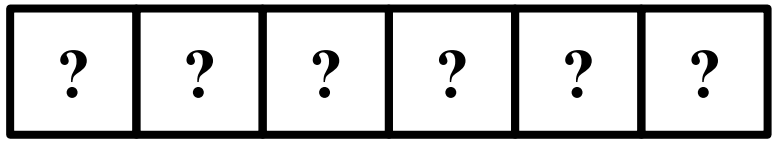
$i = 1$



$j = 7$

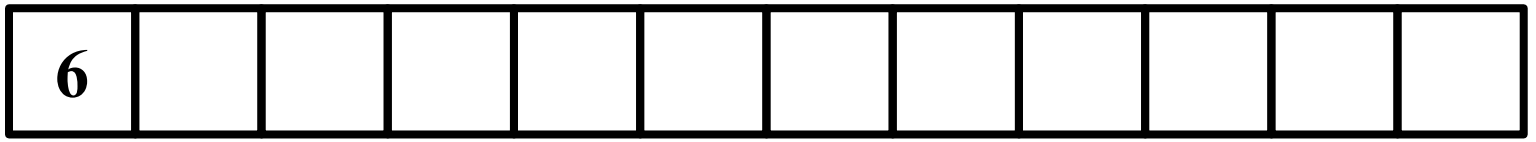
扫描子数组：
若$A[i] > A[j]$，统计逆序对，j向右移
若$A[i] \leq A[j]$，i向右移

逆序对数：

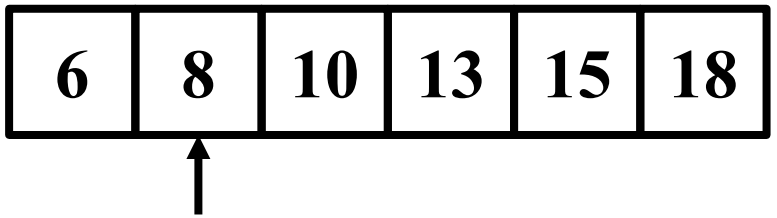


合并问题解：求解S_3

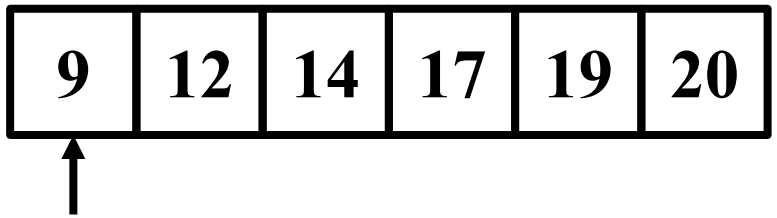
- 策略三：归并求解



子数组:



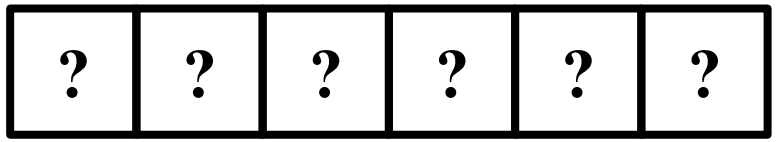
$i = 2$



$j = 7$

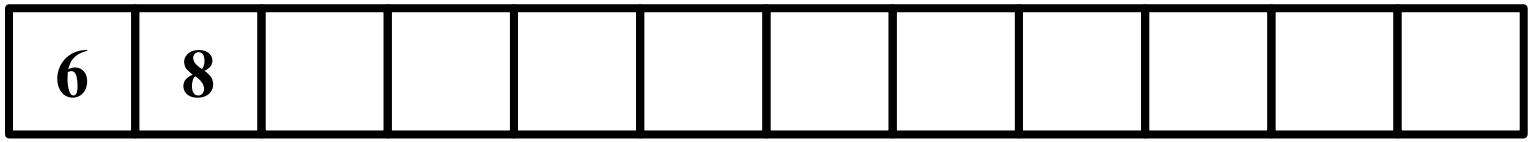
扫描子数组：
若$A[i] > A[j]$，统计逆序对，j向右移
若$A[i] \leq A[j]$，i向右移

逆序对数：

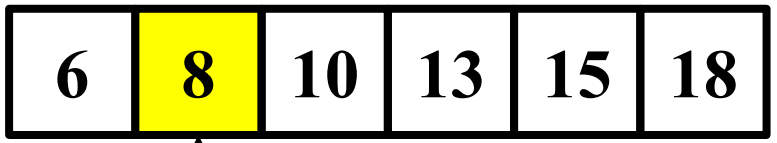


合并问题解：求解S_3

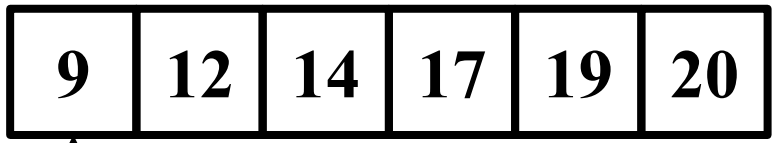
- 策略三：归并求解



子数组:



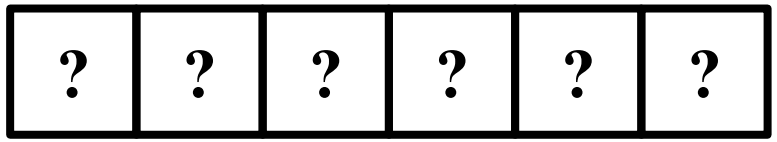
$i = 2$



$j = 7$

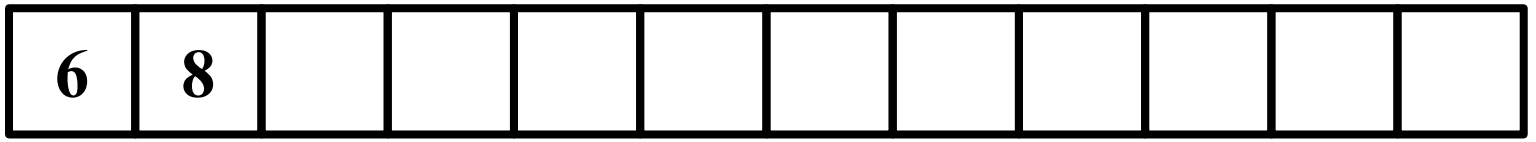
扫描子数组：
若$A[i] > A[j]$，统计逆序对，j向右移
若$A[i] \leq A[j]$，i向右移

逆序对数：

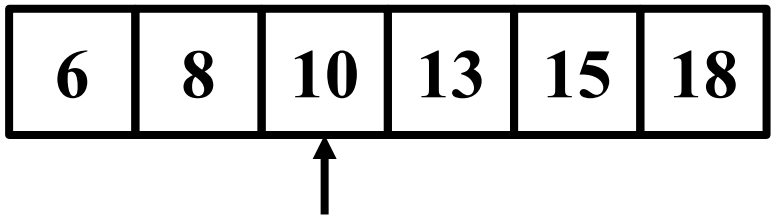


合并问题解：求解S_3

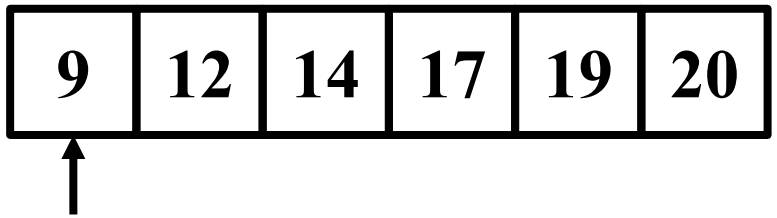
- 策略三：归并求解



子数组:



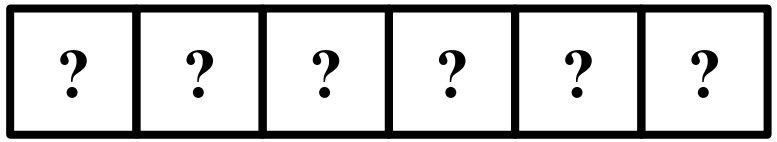
$i = 3$



$j = 7$

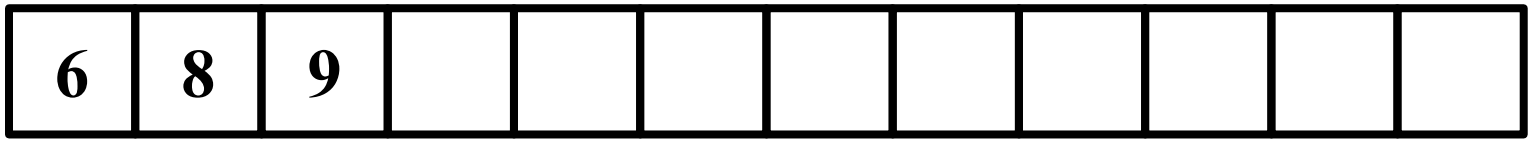
扫描子数组：
若$A[i] > A[j]$，统计逆序对，j向右移
若$A[i] \leq A[j]$，i向右移

逆序对数：

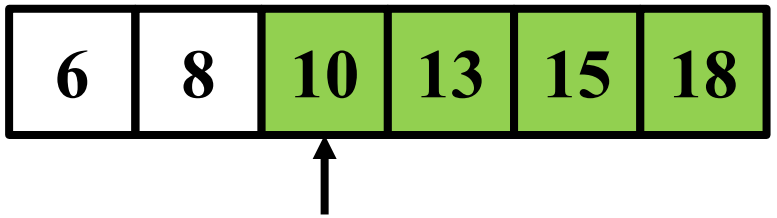


合并问题解：求解S_3

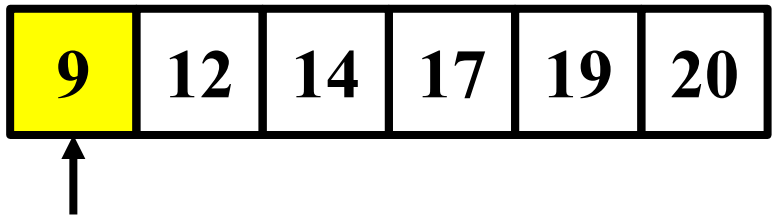
- 策略三：归并求解



子数组:



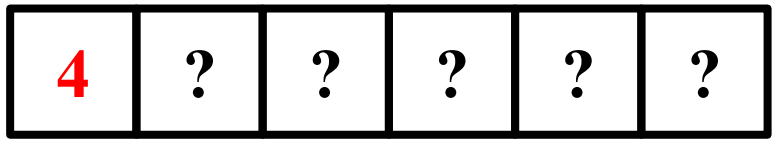
$i = 3$



$j = 7$

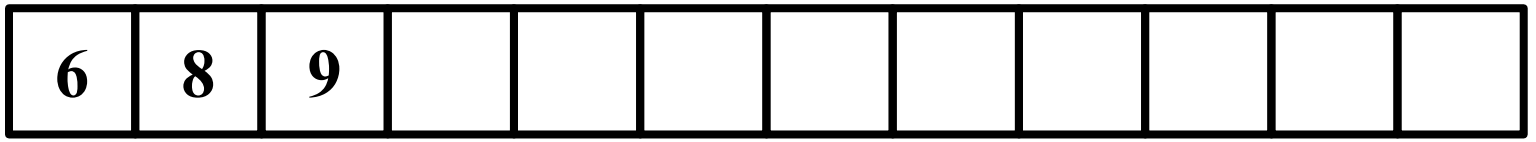
扫描子数组：
若$A[i] > A[j]$，统计逆序对，j向右移
若$A[i] \leq A[j]$，i向右移

逆序对数：

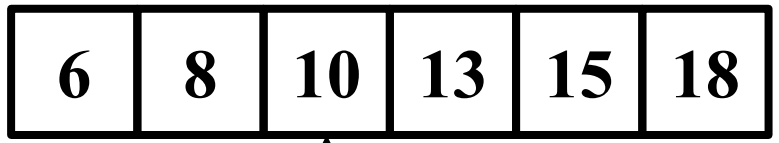


合并问题解：求解S_3

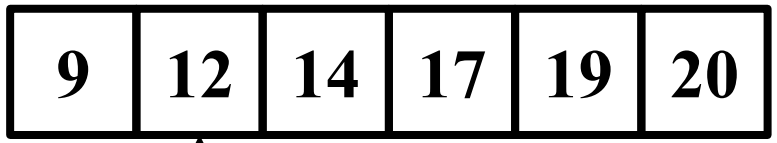
- 策略三：归并求解



子数组:



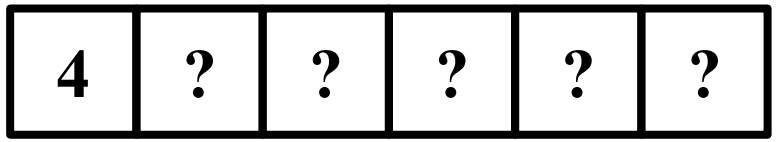
$i = 3$



$j = 8$

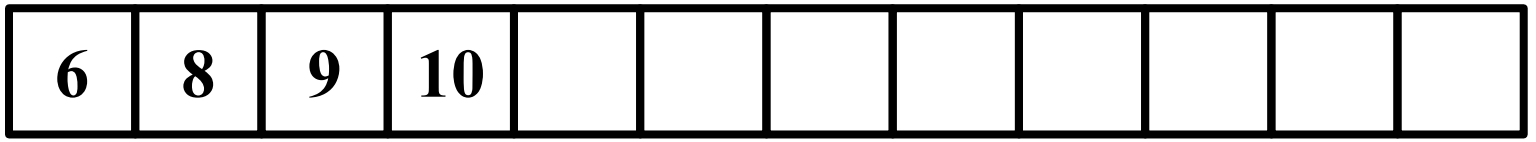
扫描子数组：
若$A[i] > A[j]$，统计逆序对，j向右移
若$A[i] \leq A[j]$，i向右移

逆序对数：

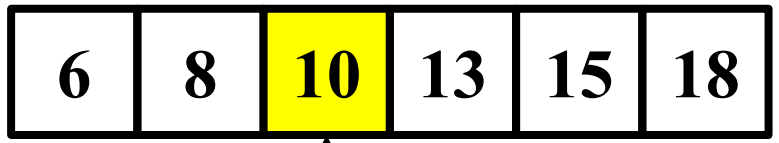


合并问题解：求解S_3

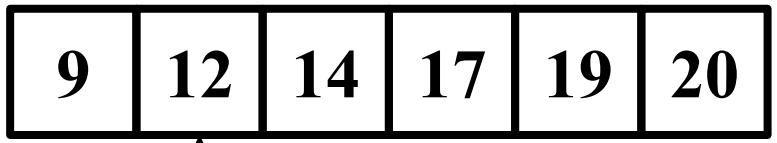
- 策略三：归并求解



子数组:



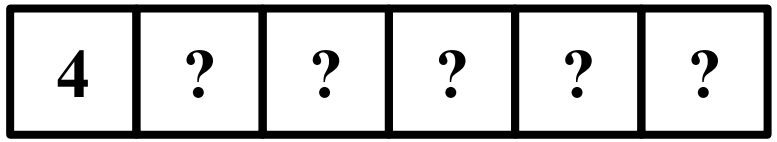
$i = 3$



$j = 8$

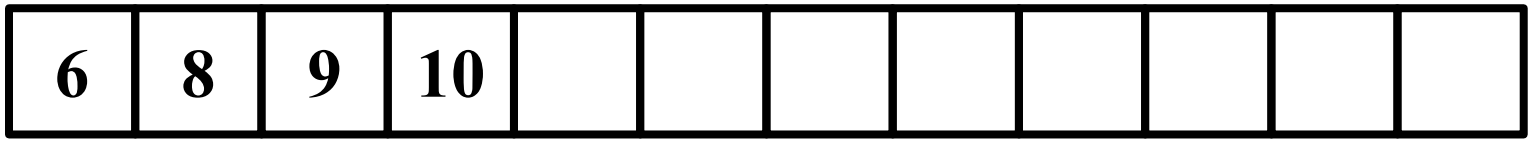
扫描子数组：
若$A[i] > A[j]$，统计逆序对，j向右移
若$A[i] \leq A[j]$，i向右移

逆序对数：

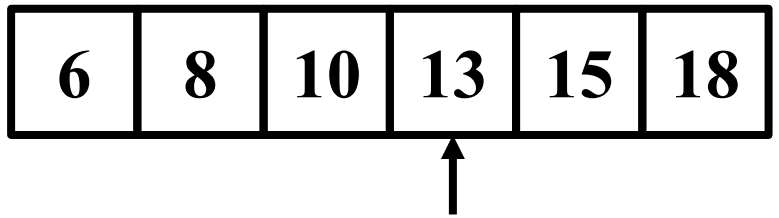


合并问题解：求解S_3

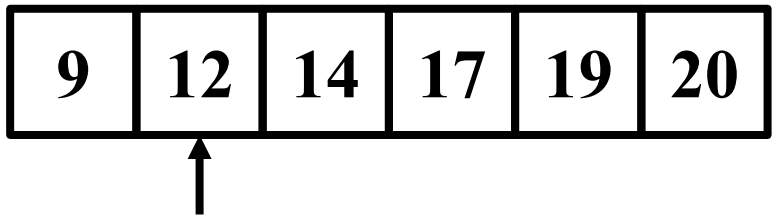
- 策略三：归并求解



子数组:



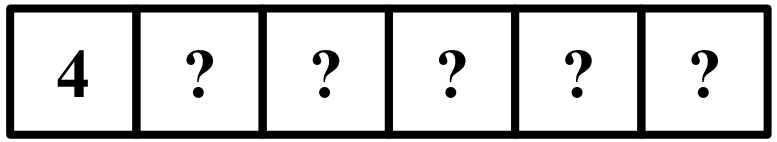
$i = 4$



$j = 8$

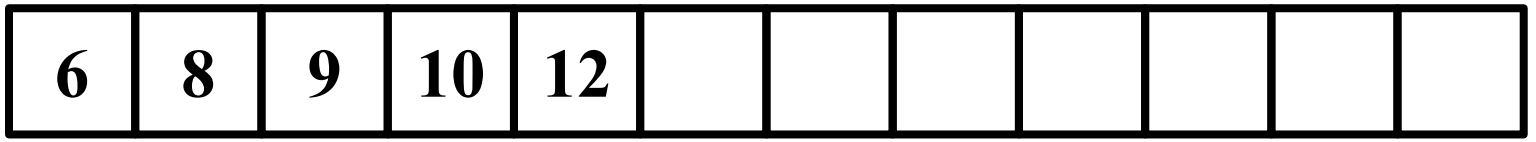
扫描子数组：
若$A[i] > A[j]$，统计逆序对，j向右移
若$A[i] \leq A[j]$，i向右移

逆序对数：

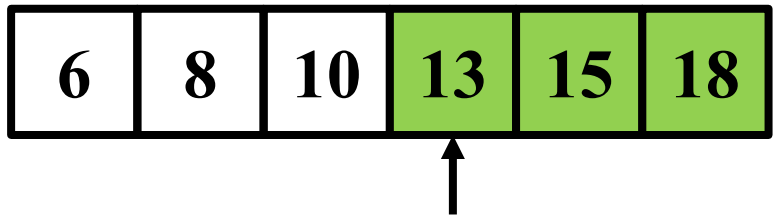


合并问题解：求解S_3

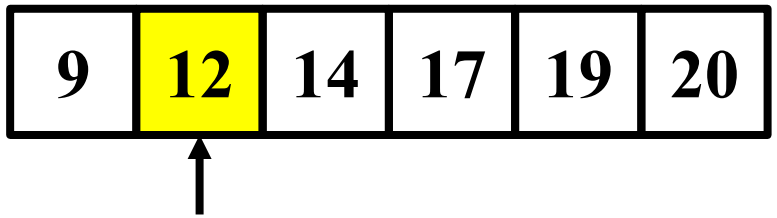
- 策略三：归并求解



子数组:



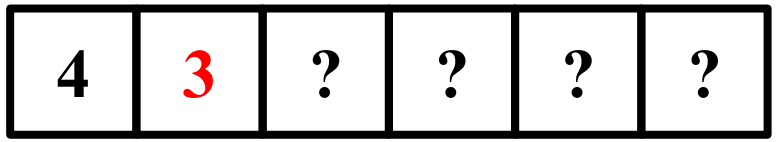
$i = 4$



$j = 8$

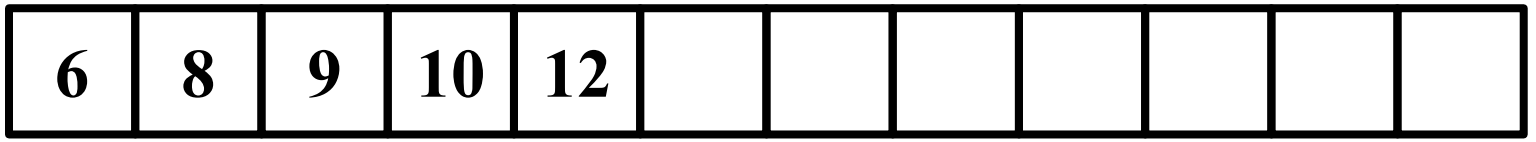
扫描子数组：
若$A[i] > A[j]$，统计逆序对，j向右移
若$A[i] \leq A[j]$，i向右移

逆序对数：

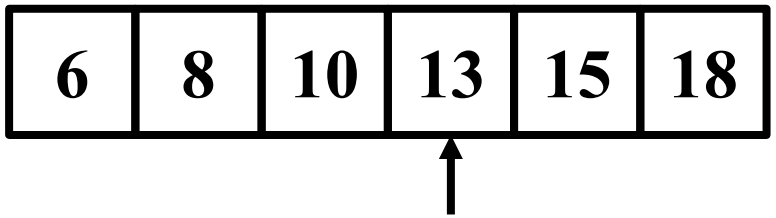


合并问题解：求解S_3

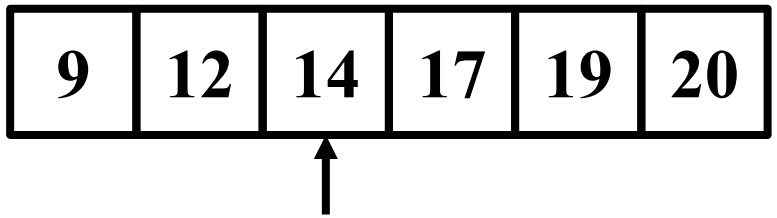
- 策略三：归并求解



子数组:



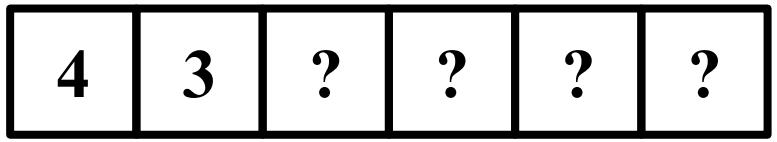
$i = 4$



$j = 9$

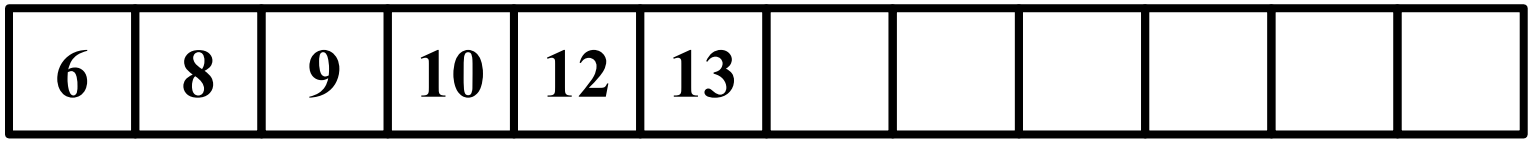
扫描子数组：
若$A[i] > A[j]$，统计逆序对，j向右移
若$A[i] \leq A[j]$，i向右移

逆序对数：

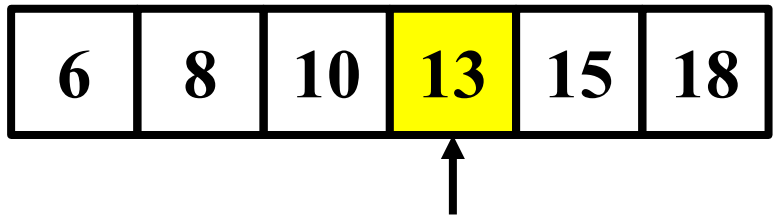


合并问题解：求解S_3

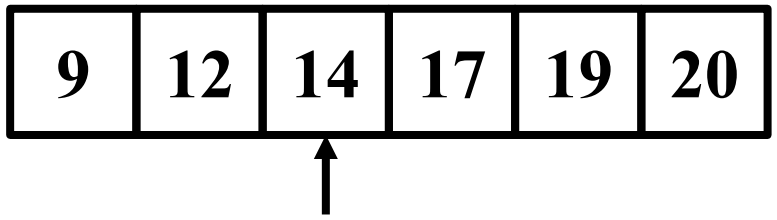
- 策略三：归并求解



子数组:



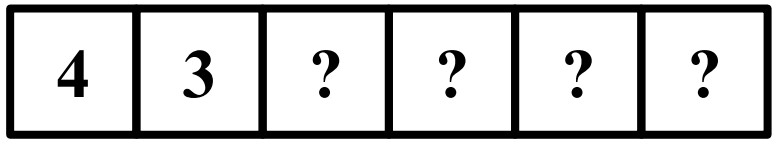
$i = 4$



$j = 9$

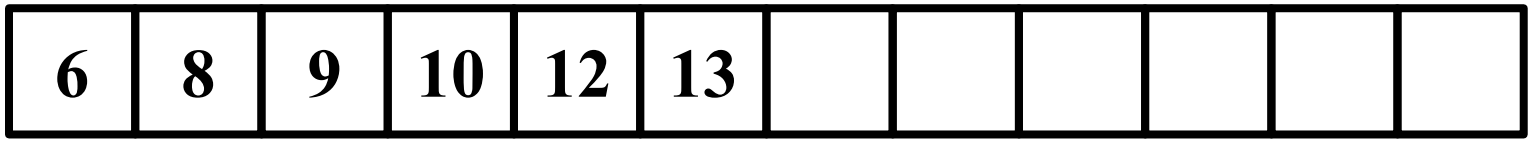
扫描子数组：
若$A[i] > A[j]$，统计逆序对，j向右移
若$A[i] \leq A[j]$，i向右移

逆序对数：

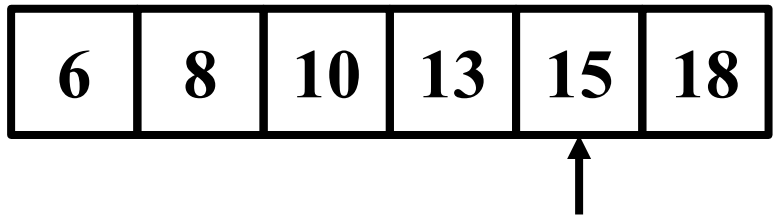


合并问题解：求解S_3

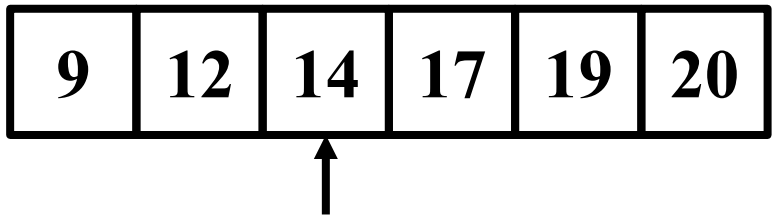
- 策略三：归并求解



子数组:



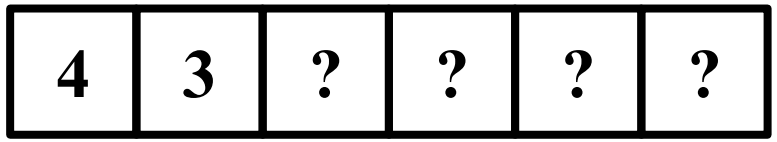
$i = 5$



$j = 9$

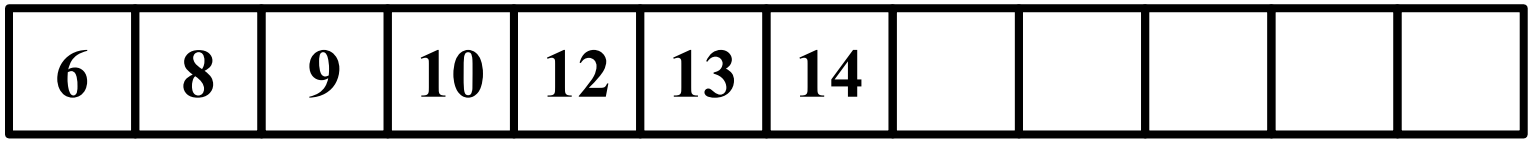
扫描子数组：
若$A[i] > A[j]$，统计逆序对，j向右移
若$A[i] \leq A[j]$，i向右移

逆序对数：

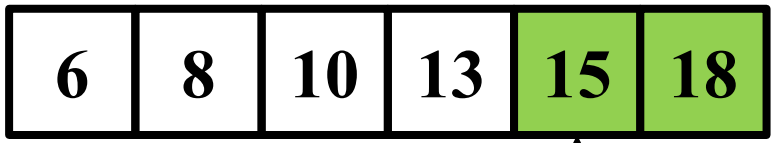


合并问题解：求解S_3

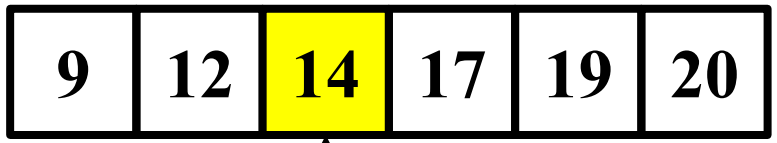
- 策略三：归并求解



子数组:



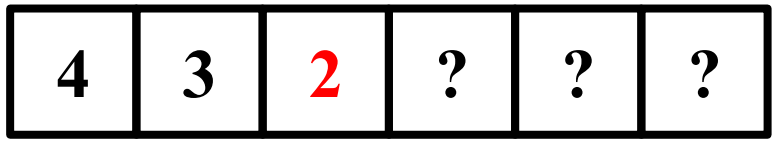
$i = 5$



$j = 9$

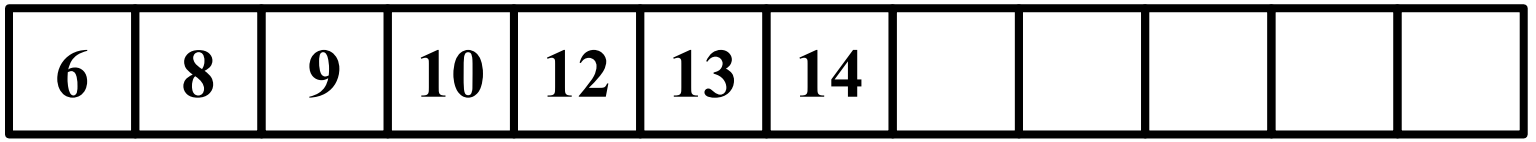
扫描子数组：
若$A[i] > A[j]$，统计逆序对，j向右移
若$A[i] \leq A[j]$，i向右移

逆序对数：

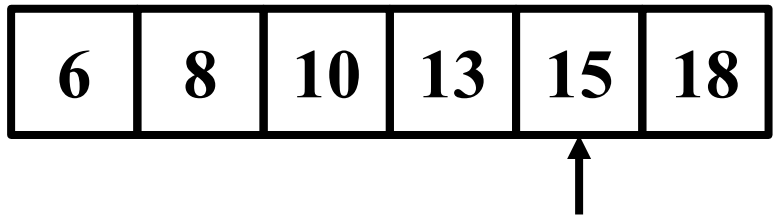


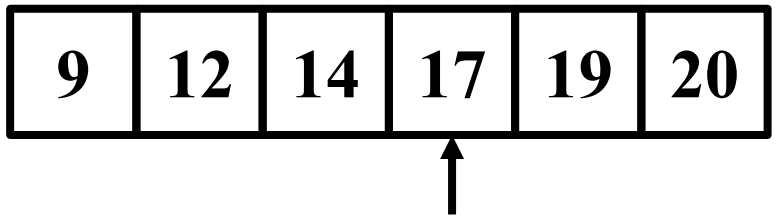
合并问题解：求解S_3

- 策略三：归并求解



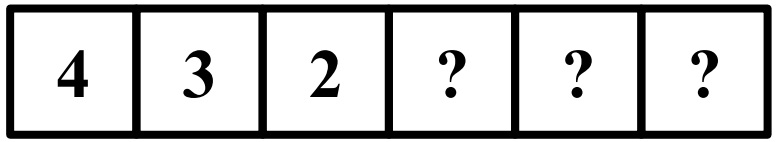
子数组:





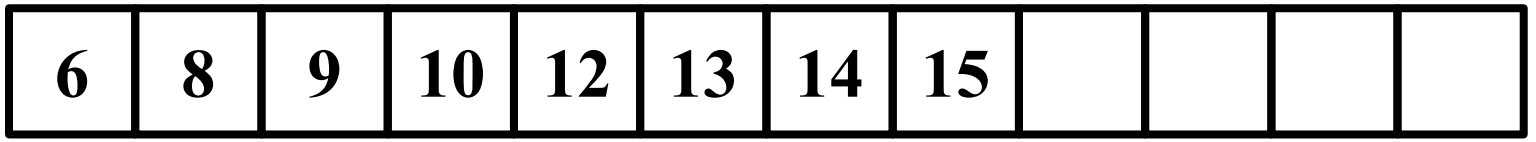
扫描子数组：
若$A[i] > A[j]$，统计逆序对，j向右移
若$A[i] \leq A[j]$，i向右移

逆序对数：

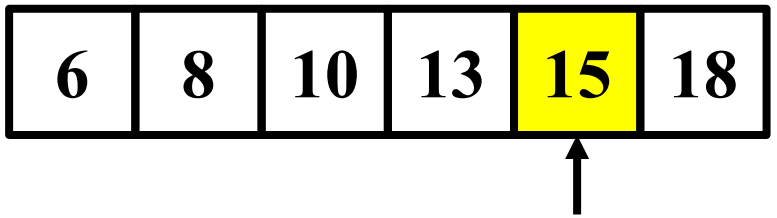


合并问题解：求解S_3

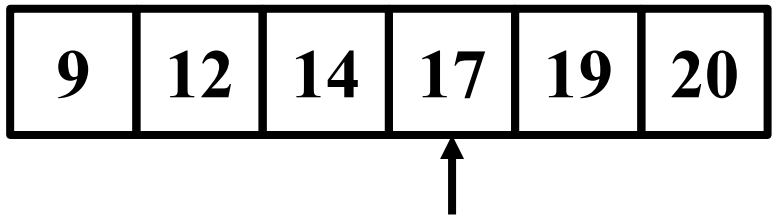
- 策略三：归并求解



子数组:



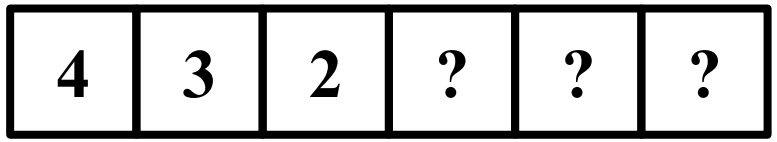
$i = 5$



$j = 10$

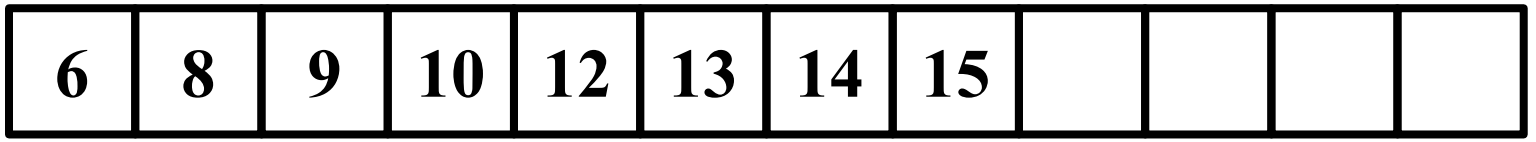
扫描子数组：
若$A[i] > A[j]$，统计逆序对，j向右移
若$A[i] \leq A[j]$，i向右移

逆序对数：

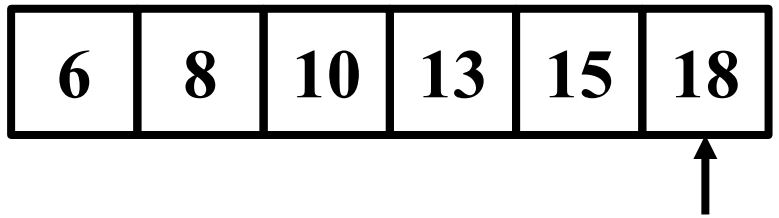


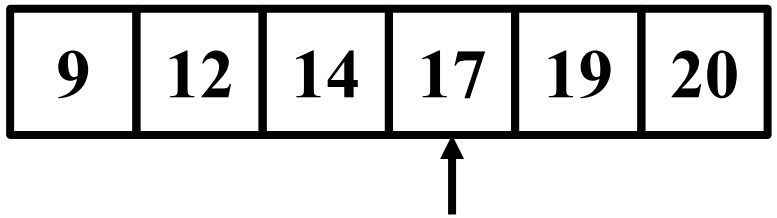
合并问题解：求解S_3

- 策略三：归并求解



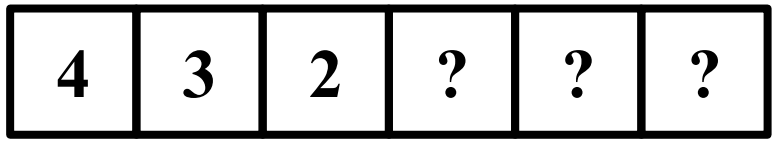
子数组:





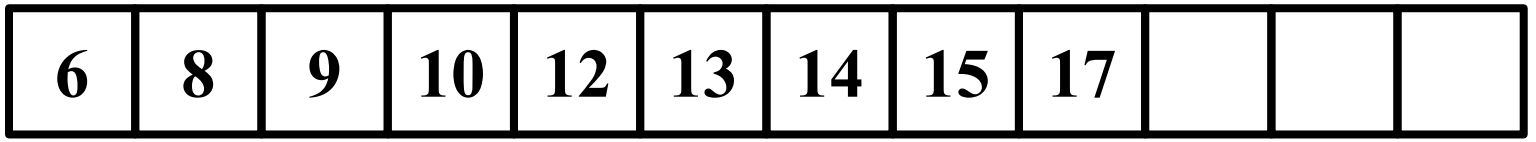
扫描子数组：
若$A[i] > A[j]$，统计逆序对，j向右移
若$A[i] \leq A[j]$，i向右移

逆序对数：

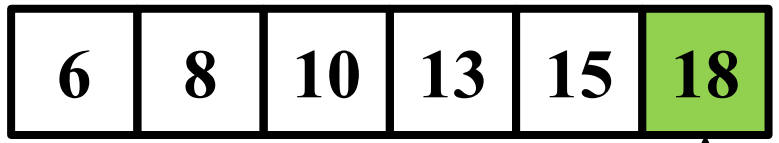


合并问题解：求解S_3

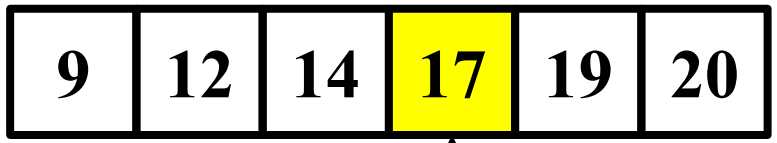
- 策略三：归并求解



子数组:



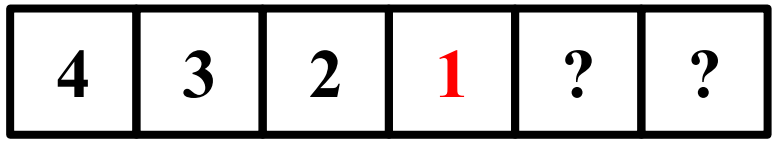
$i = 6$



$j = 10$

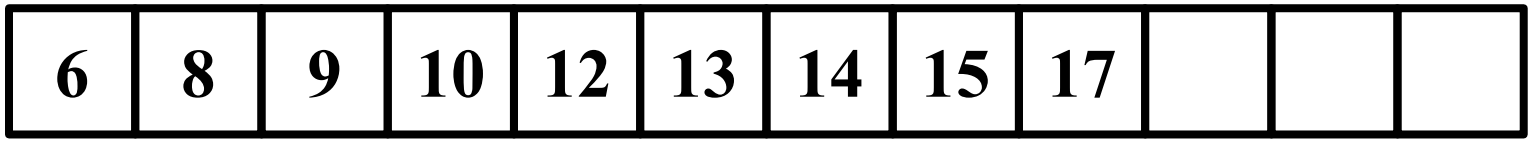
扫描子数组：
若$A[i] > A[j]$，统计逆序对，j向右移
若$A[i] \leq A[j]$，i向右移

逆序对数：

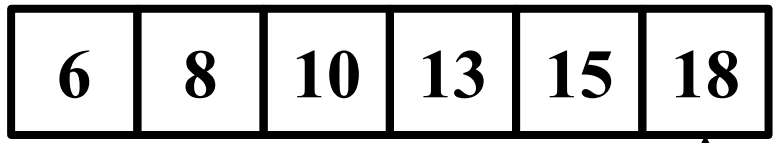


合并问题解：求解S_3

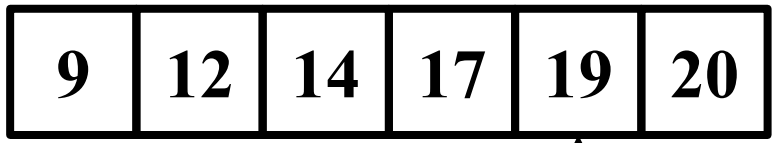
- 策略三：归并求解



子数组:



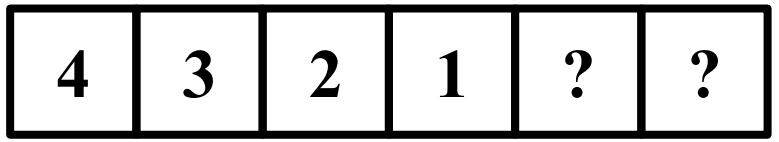
$i = 6$



$j = 11$

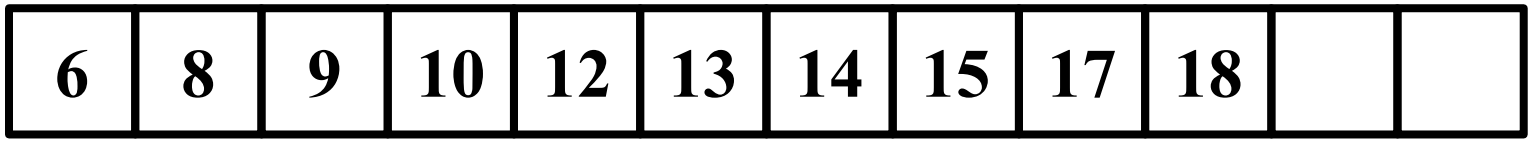
扫描子数组：
若$A[i] > A[j]$，统计逆序对，j向右移
若$A[i] \leq A[j]$，i向右移

逆序对数：

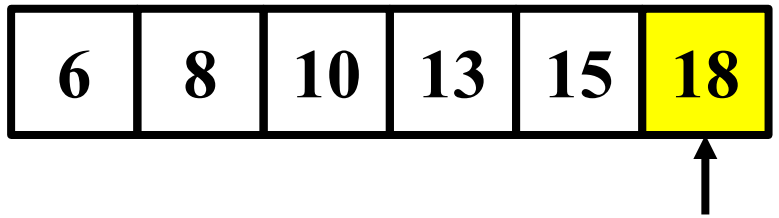


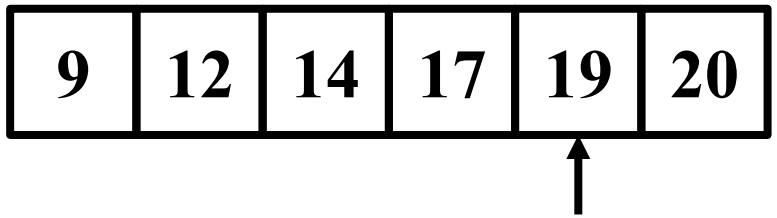
合并问题解：求解S_3

- 策略三：归并求解



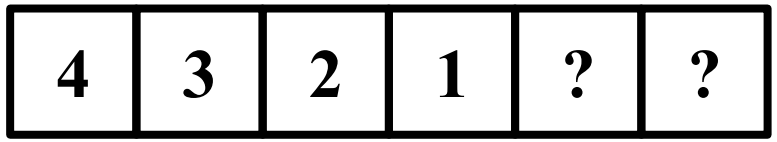
子数组:





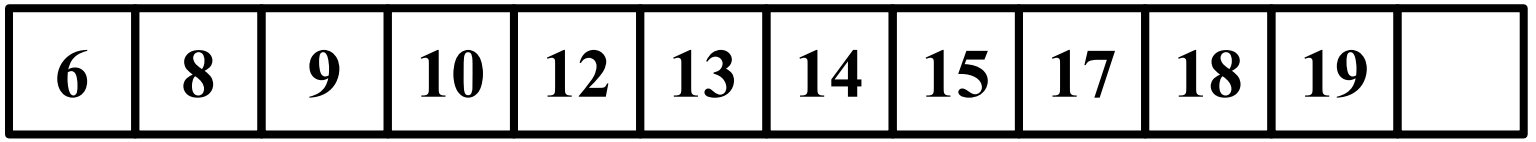
扫描子数组：
若$A[i] > A[j]$，统计逆序对，j向右移
若$A[i] \leq A[j]$，i向右移

逆序对数：

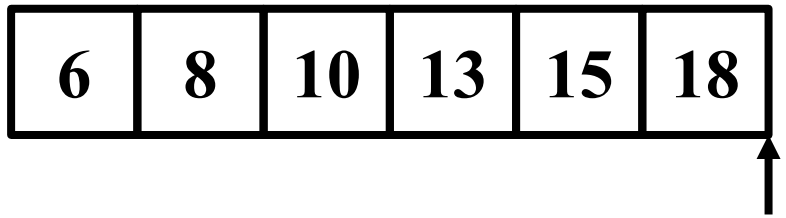


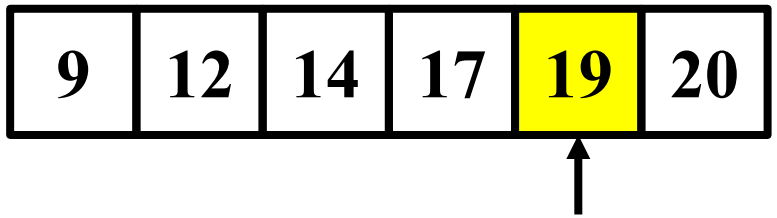
合并问题解：求解S_3

- 策略三：归并求解



子数组:

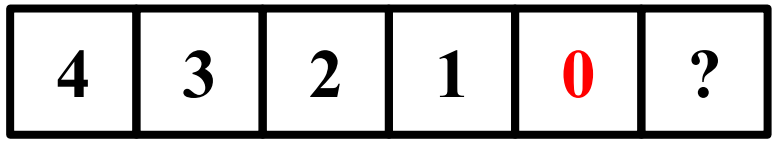




$j = 11$

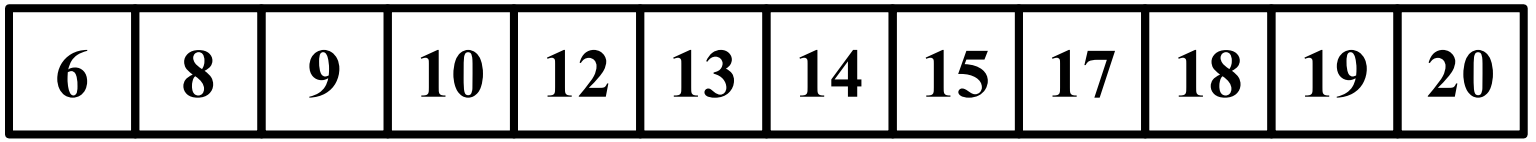
扫描子数组：
若$A[i] > A[j]$，统计逆序对，j向右移
若$A[i] \leq A[j]$，i向右移

逆序对数：

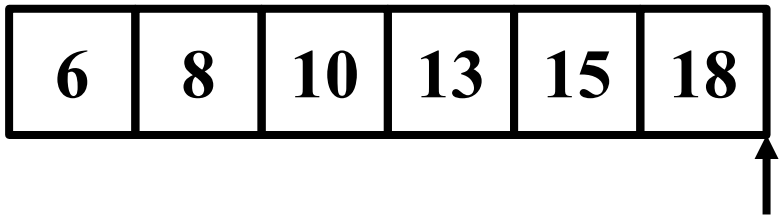


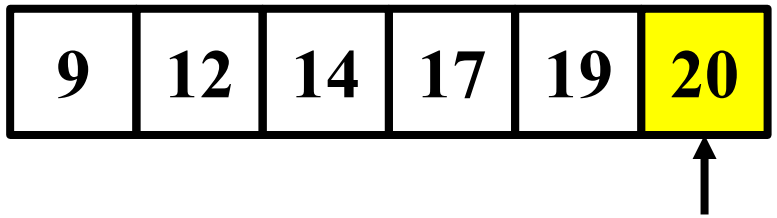
合并问题解：求解S_3

- 策略三：归并求解



子数组:

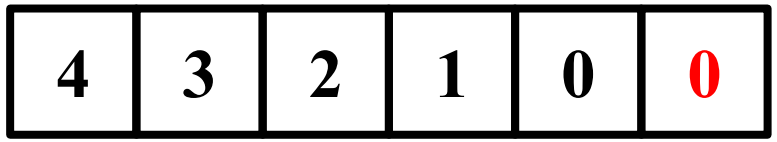




$j = 12$

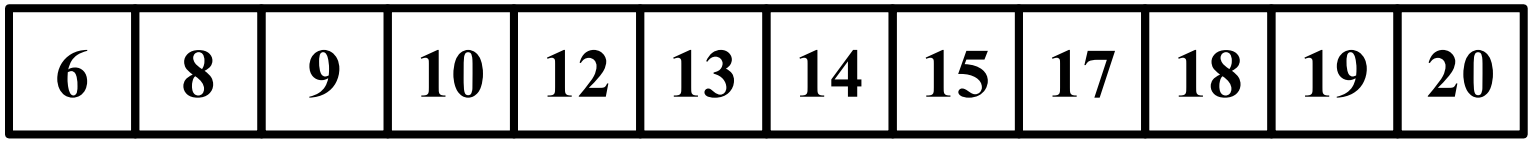
扫描子数组：
若$A[i] > A[j]$，统计逆序对，j向右移
若$A[i] \leq A[j]$，i向右移

逆序对数：

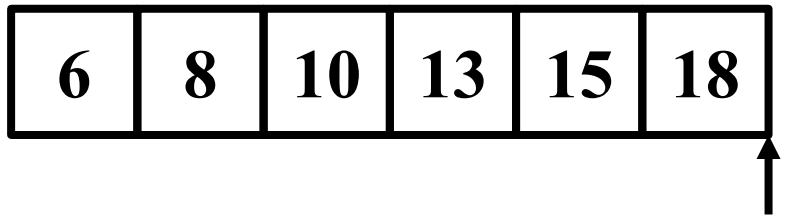


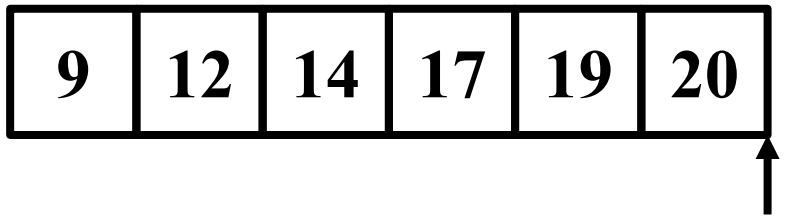
合并问题解：求解S_3

- 策略三：归并求解



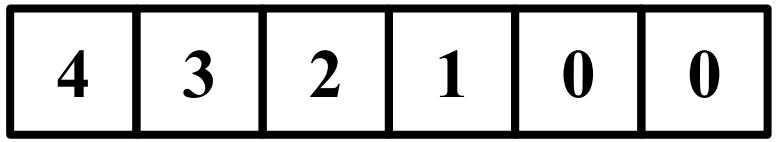
子数组:





$S_3 = 4 + 3 + 2 + 1 = 10$

逆序对数：



求解S_3时间复杂度降至$O(n)$

伪代码

- 归并求解： MergeCount($A, left, mid, right$)

输入: 数组$A[1..n]$,数组下标$left, mid, right$
输出: 跨越数组$A[left..mid]$和$A[mid+1..right]$的逆序对数，$A[left..right]$

```
A'[left..right] ← A[left..right], S3 ← 0
i ← left, j ← mid + 1, k ← 0
while i ≤ mid and j ≤ right do
    if A'[i] ≤ A'[j] then
        A[left + k] ← A'[i]
        k ← k + 1, i ← i + 1
    end
    else
        A[left + k] ← A'[j]
        S3 ← S3 + (mid − i + 1)
        k ← k + 1, j ← j + 1
    end
end
if i ≤ mid then
    A[k..right] ← A'[i..mid]
end
else
    A[k..right] ← A'[j..right]
end
return S3, A[left..right]
```

遍历子数组时计算S_3

去掉这行，就是归并排序

伪代码

- 归并求解： MergeCount($A, left, mid, right$)

输入: 数组$A[1..n]$,数组下标$left, mid, right$
输出: 跨越数组$A[left..mid]$和$A[mid+1..right]$的逆序对数，$A[left..right]$

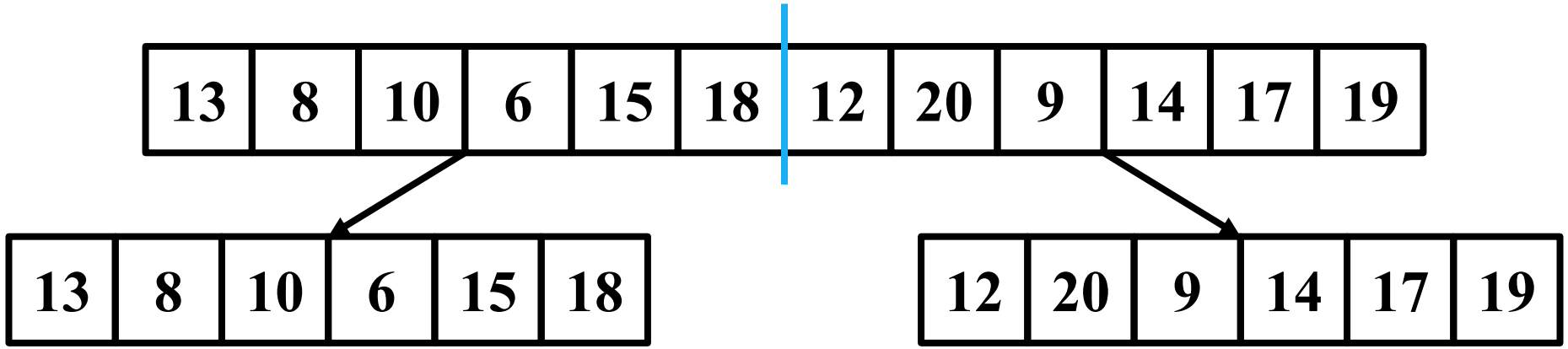
```
A'[left..right] ← A[left..right], S3 ← 0
i ← left, j ← mid + 1, k ← 0
while i ≤ mid and j ≤ right do
    if A'[i] ≤ A'[j] then
        A[left + k] ← A'[i]
        k ← k + 1, i ← i + 1
    end
    else
        A[left + k] ← A'[j]
        S3 ← S3 + (mid − i + 1)
        k ← k + 1, j ← j + 1
    end
end
if i ≤ mid then
    A[k..right] ← A'[i..mid]
end
else
    A[k..right] ← A'[j..right]
end
return S3, A[left..right]
```

添加剩余元素保证有序

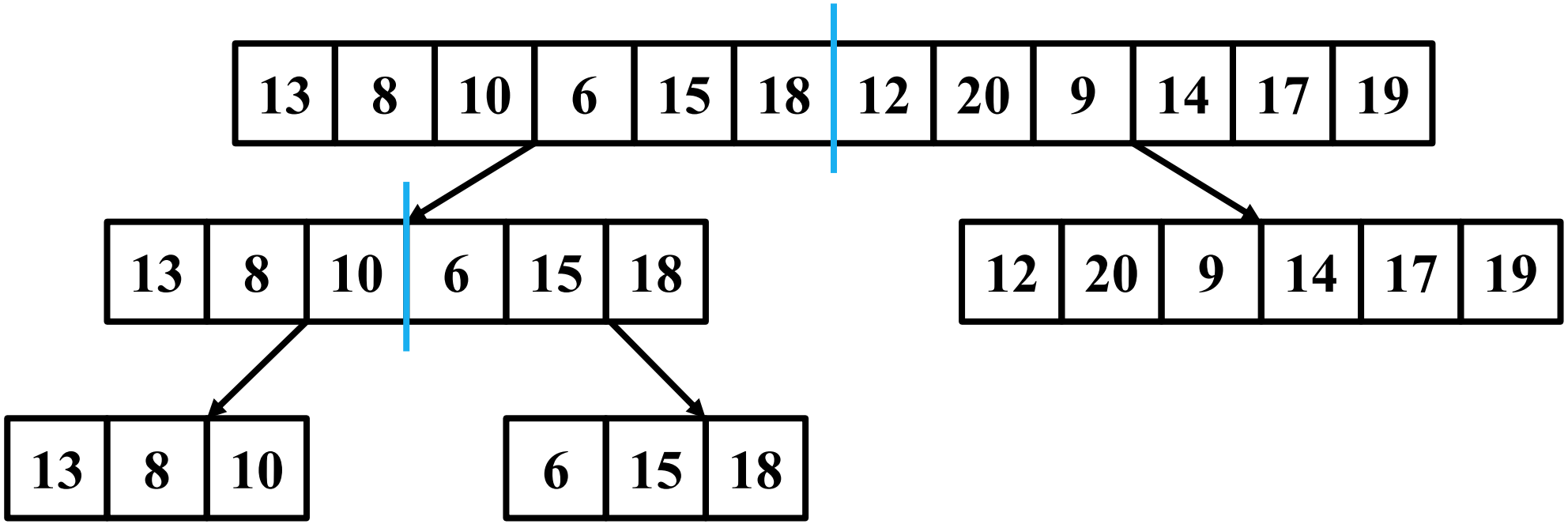
算法实例

13	8	10	6	15	18	12	20	9	14	17	19

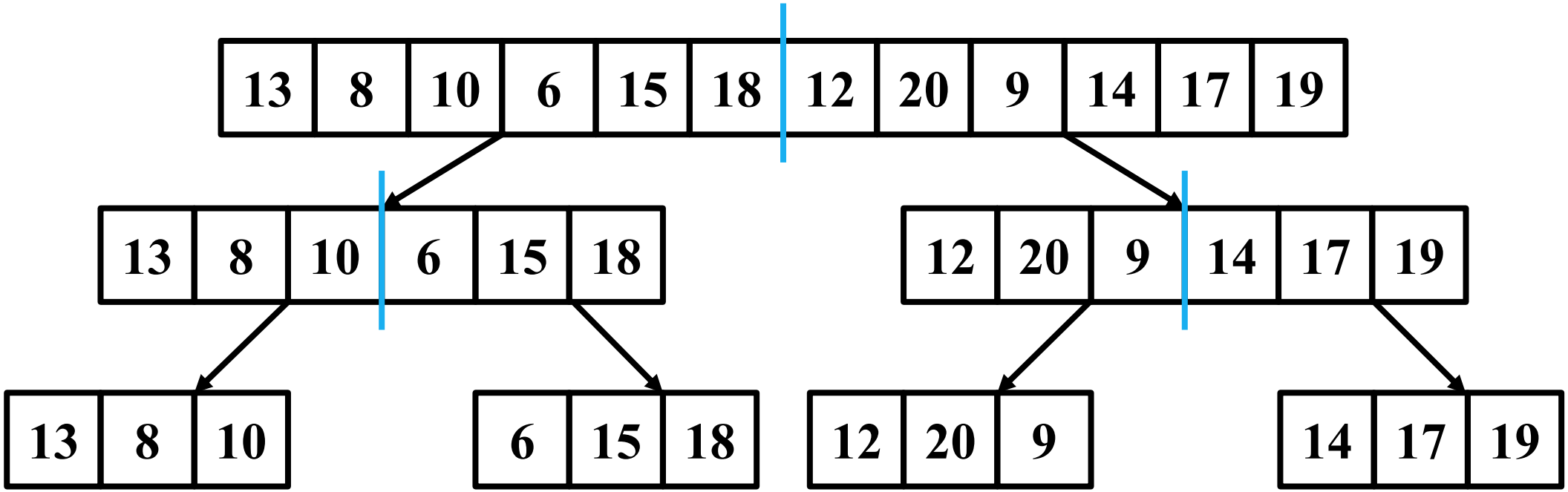
算法实例



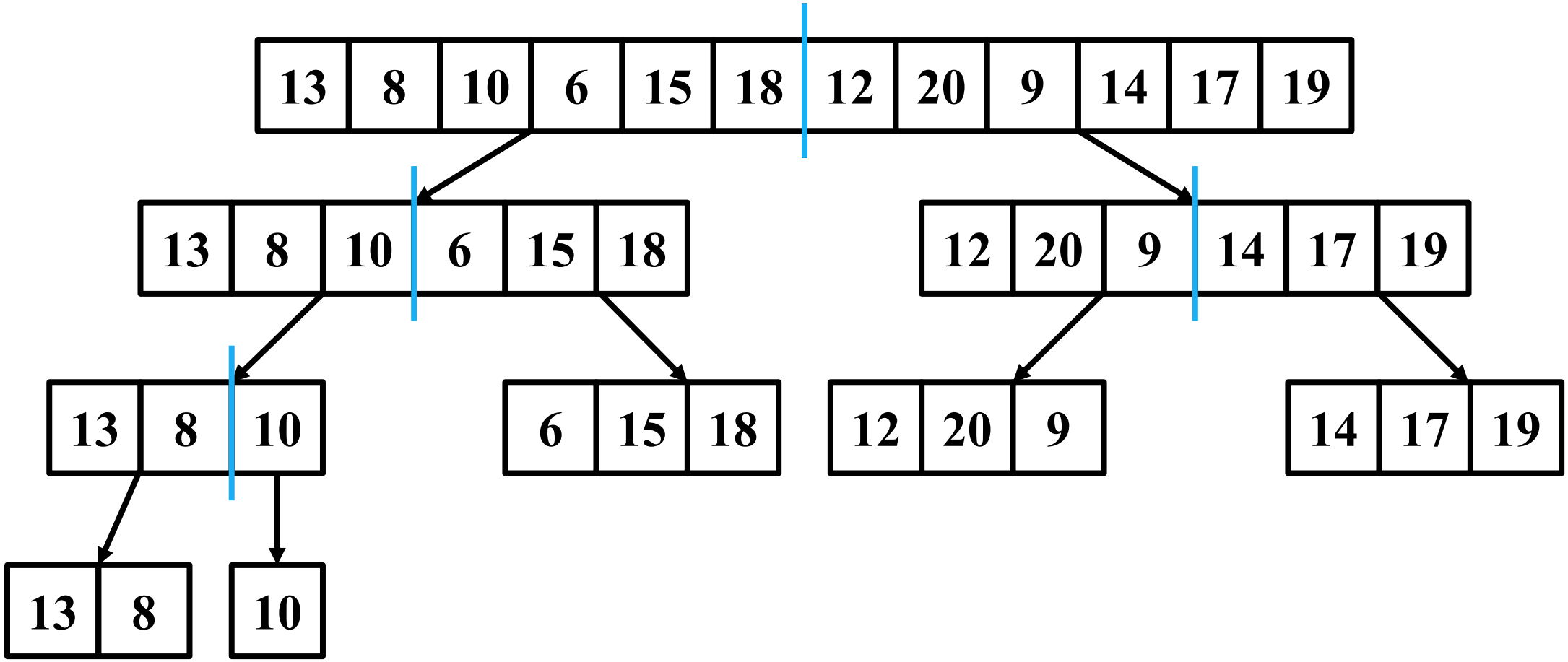
算法实例


13 8 10 6 15 18 12 20 9 14 17 19
13 8 10 6 15 18
12 20 9 14 17 19
13 8 10
6 15 18

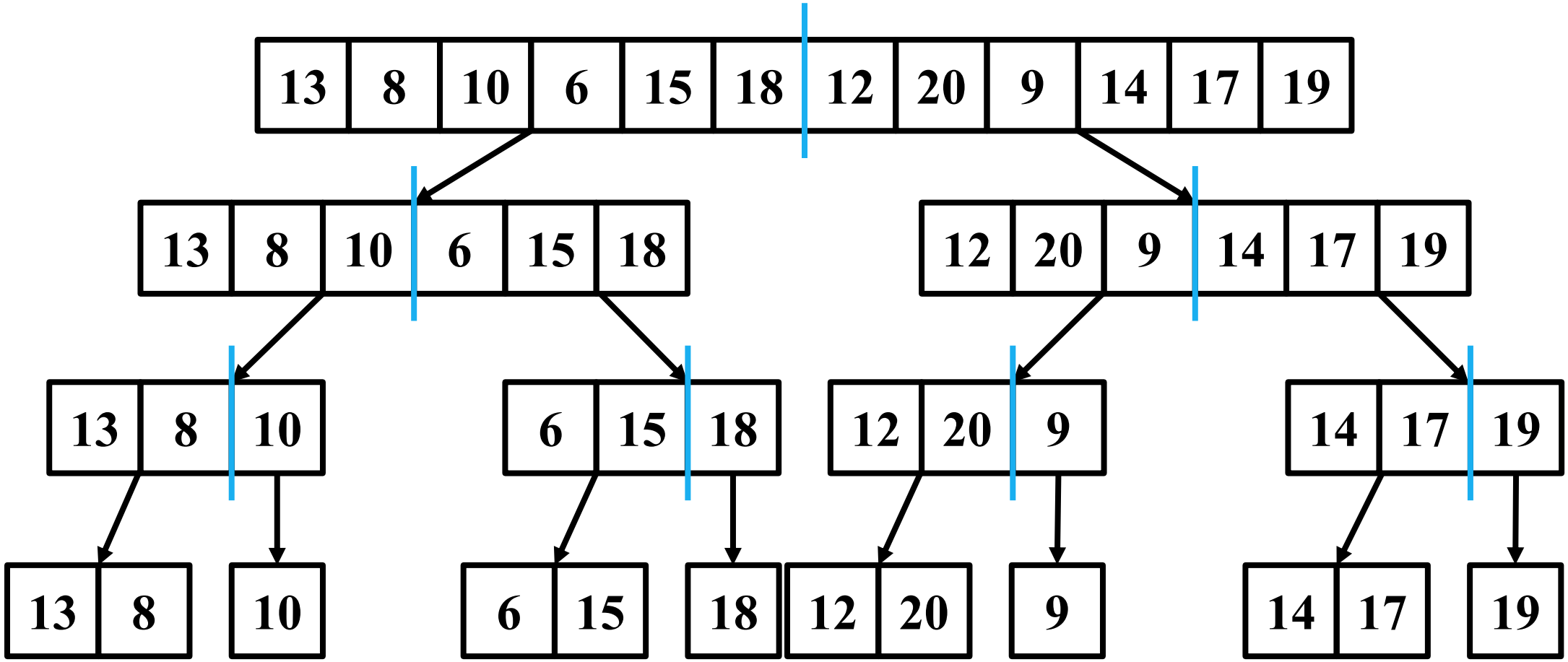
算法实例


13 8 10 6 15 18 12 20 9 14 17 19
13 8 10 6 15 18
12 20 9 14 17 19
13 8 10
6 15 18
12 20 9
14 17 19

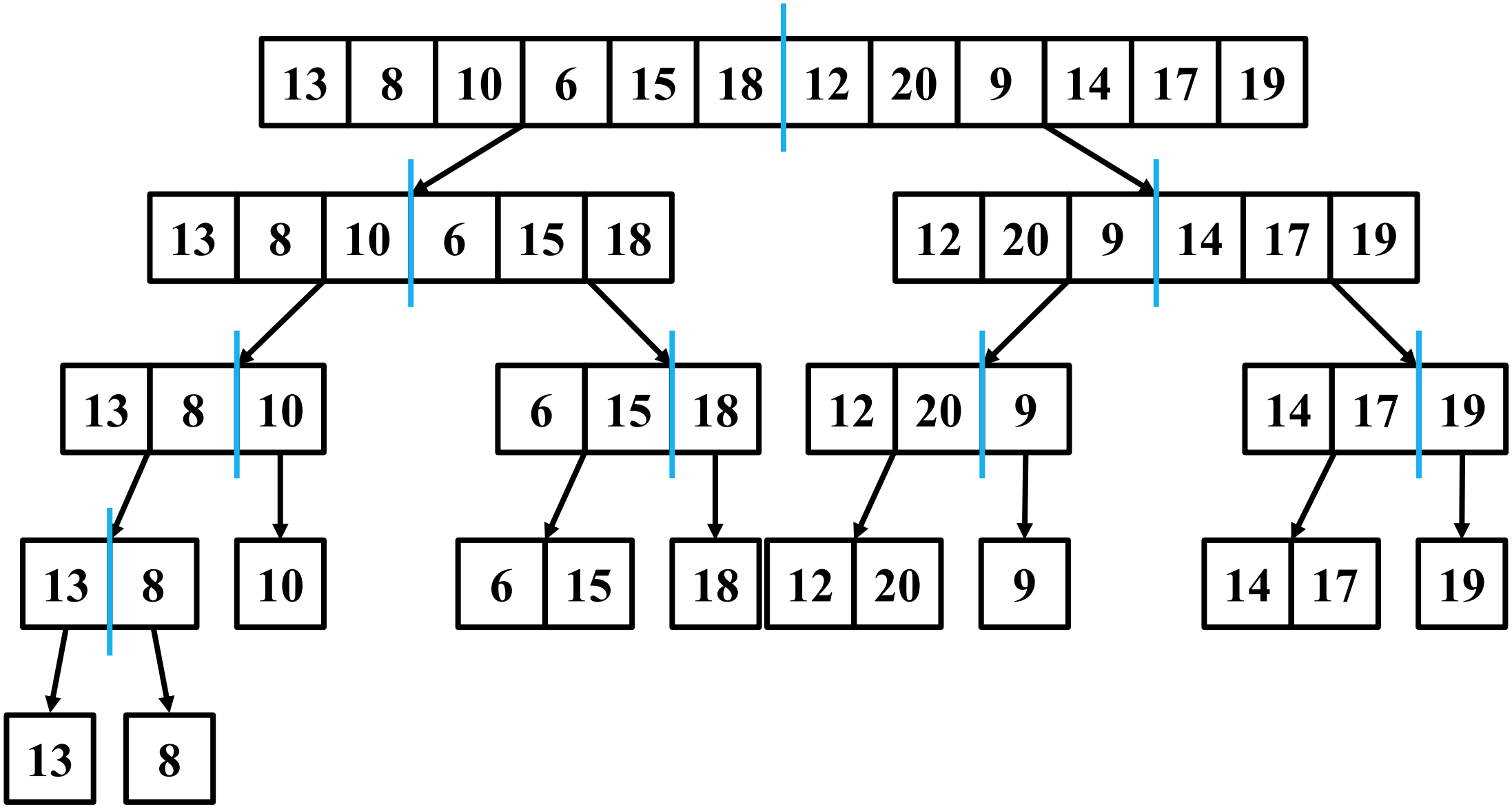
算法实例


13 8 10 6 15 18 12 20 9 14 17 19
13 8 10 6 15 18
12 20 9 14 17 19
13 8 10
6 15 18
12 20 9
14 17 19
13 8
10

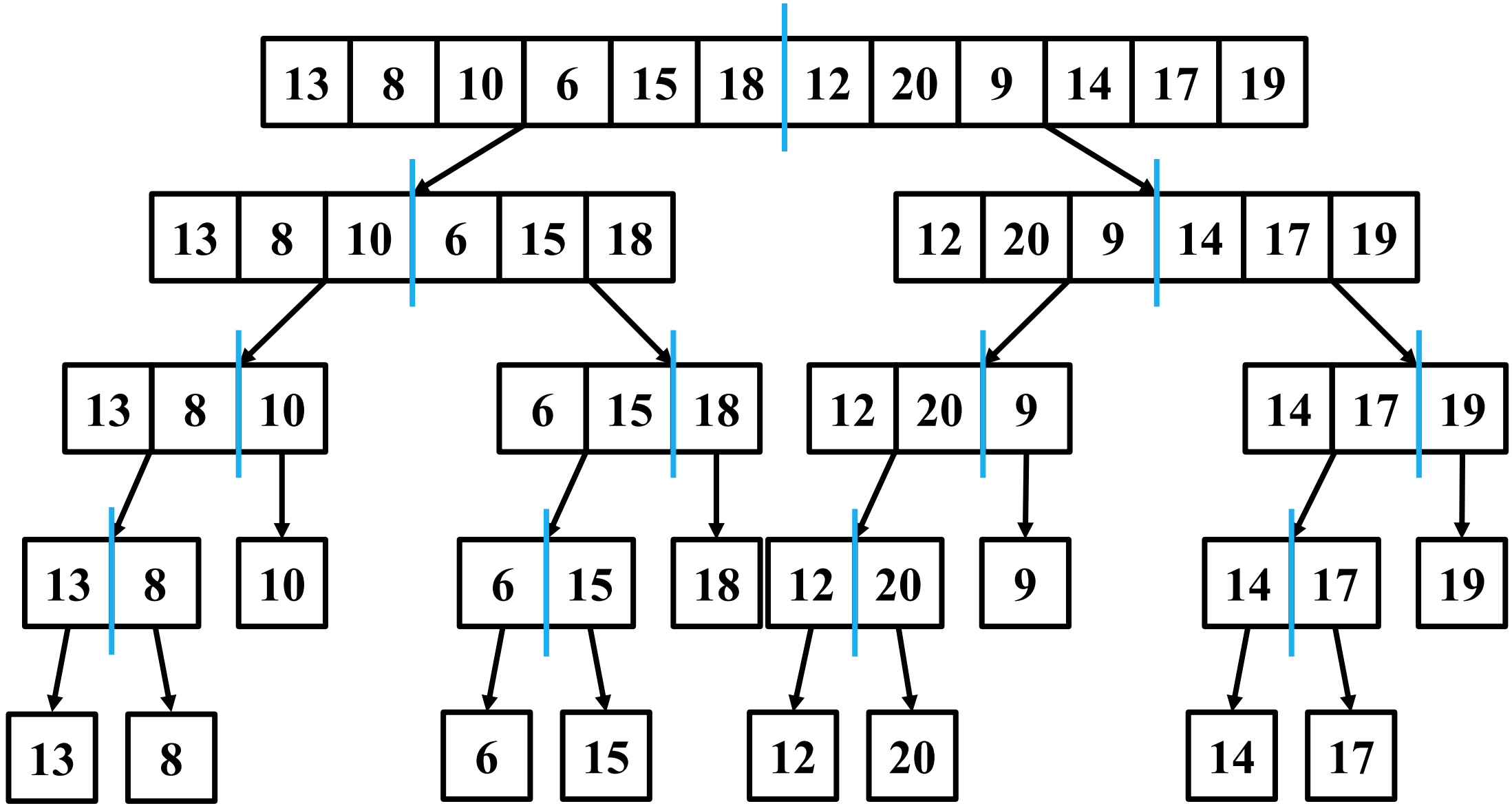
算法实例


13 8 10 6 15 18 12 20 9 14 17 19
13 8 10 6 15 18
12 20 9 14 17 19
13 8 10
6 15 18
12 20 9
14 17 19
13 8
10
6 15
18
12 20
9
14 17
19

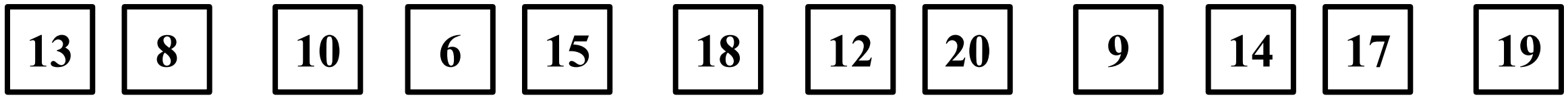
算法实例


13 8 10 6 15 18 12 20 9 14 17 19
13 8 10 6 15 18
12 20 9 14 17 19
13 8 10
6 15 18
12 20 9
14 17 19
13 8
10
6 15
18
12 20
9
14 17
19
13
8

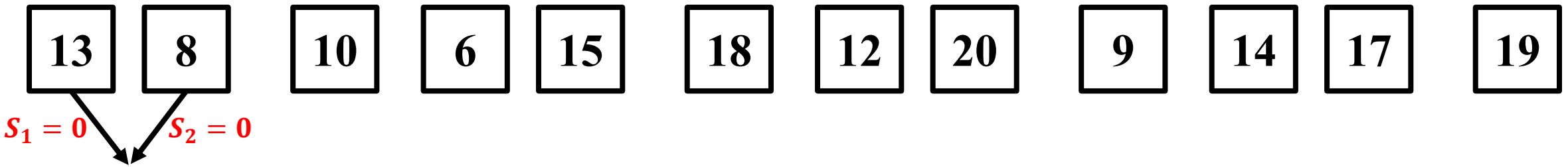
算法实例


13 8 10 6 15 18 12 20 9 14 17 19
13 8 10 6 15 18
12 20 9 14 17 19
13 8 10
6 15 18
12 20 9
14 17 19
13 8
10
6 15
18
12 20
9
14 17
19
13
8
6
15
12
20
14
17

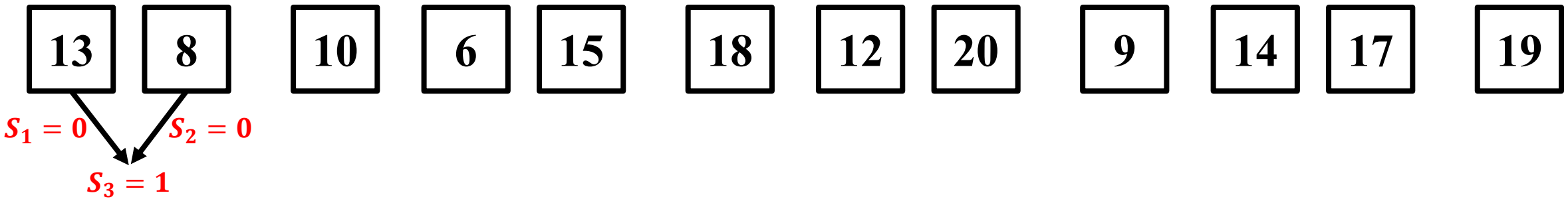
算法实例



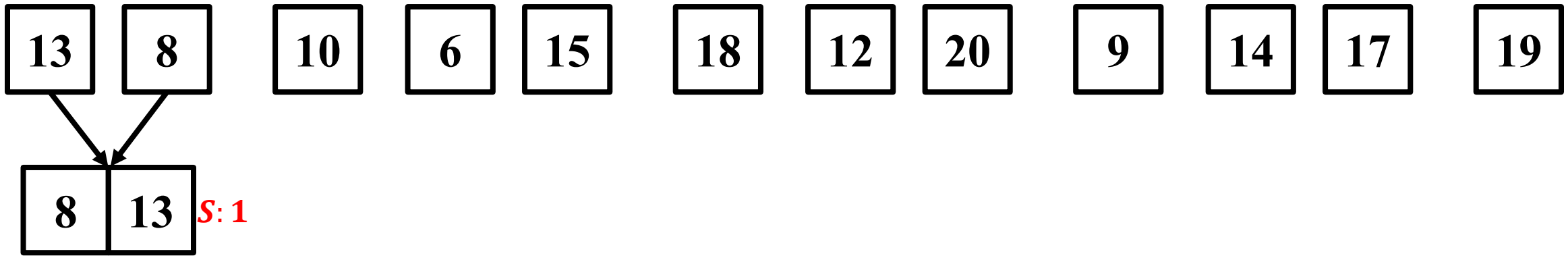
算法实例



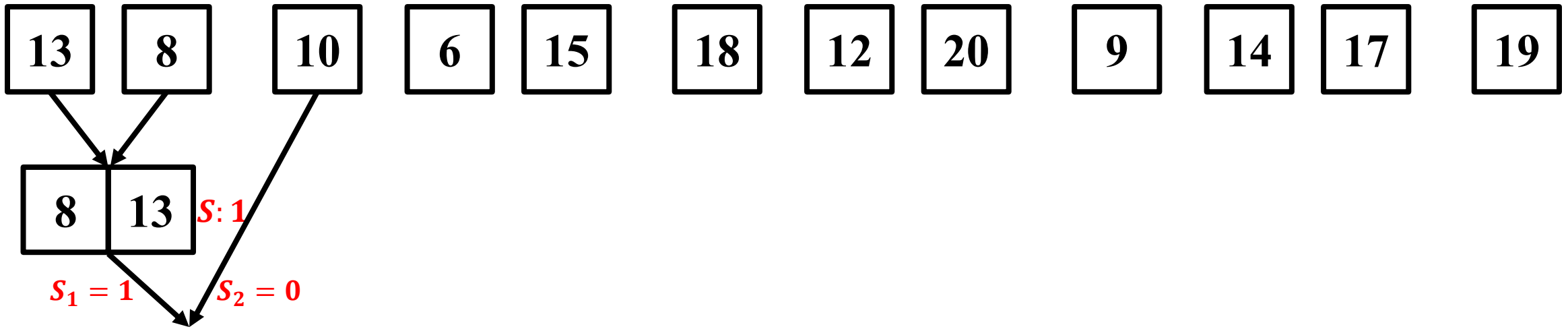
算法实例



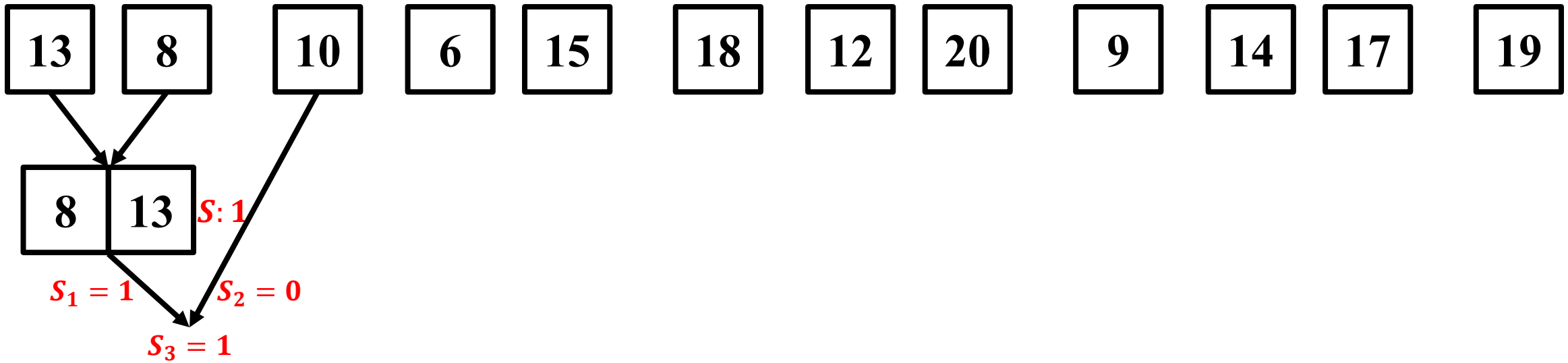
算法实例



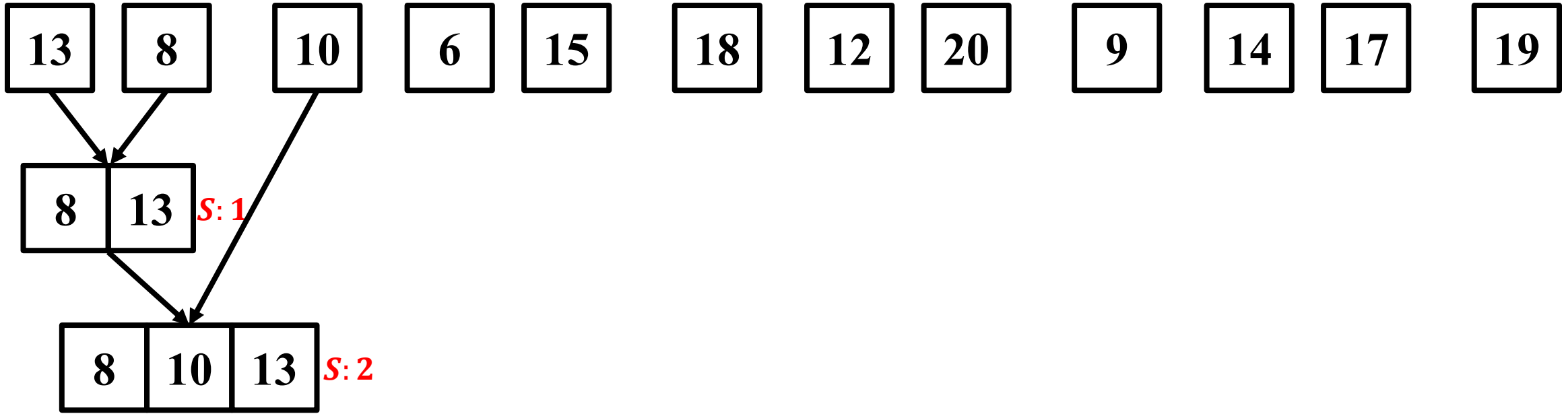
算法实例



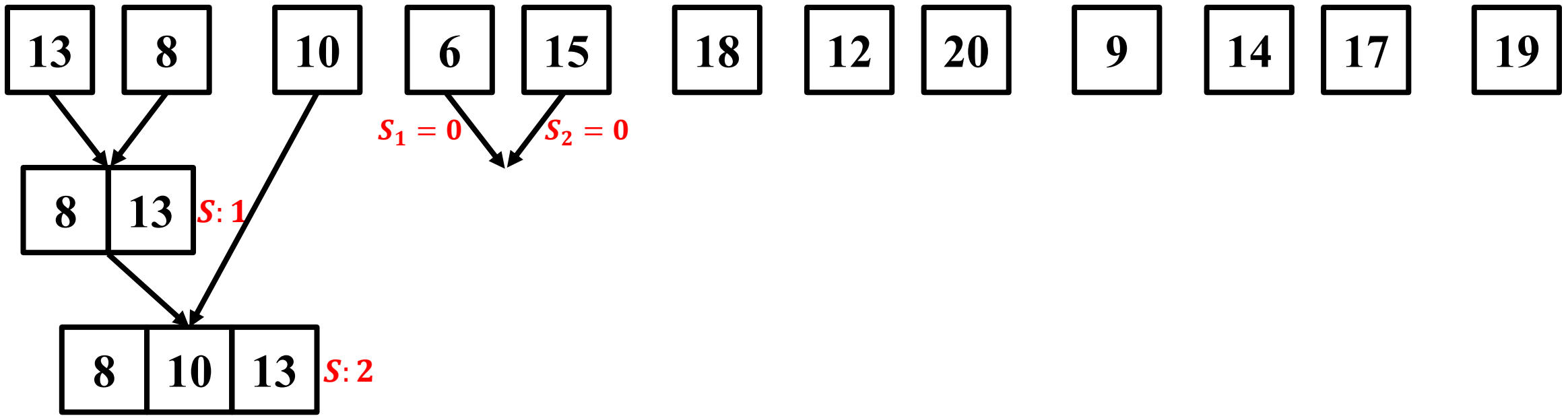
算法实例



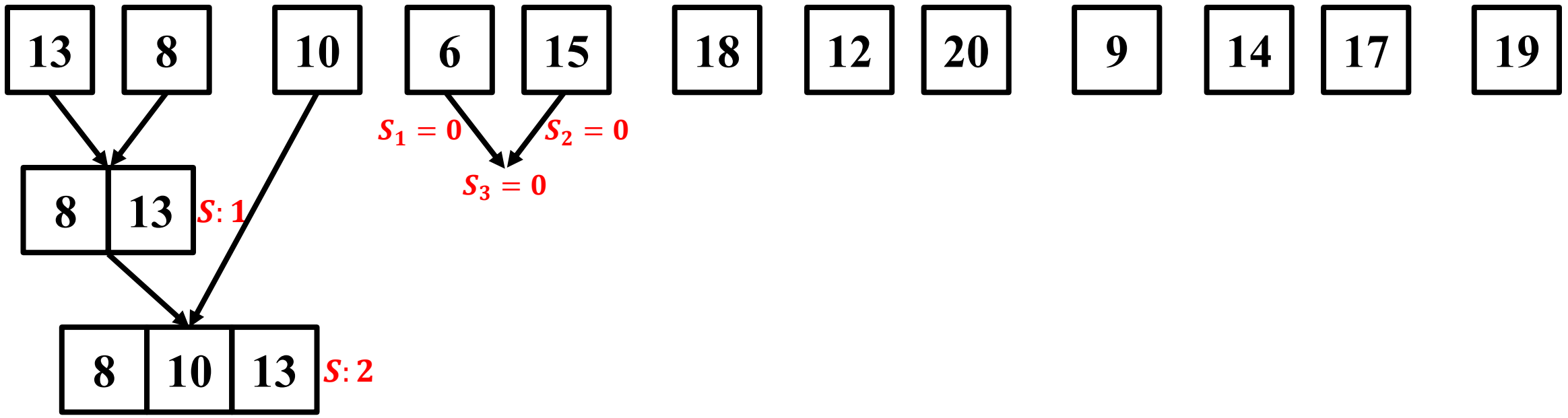
算法实例



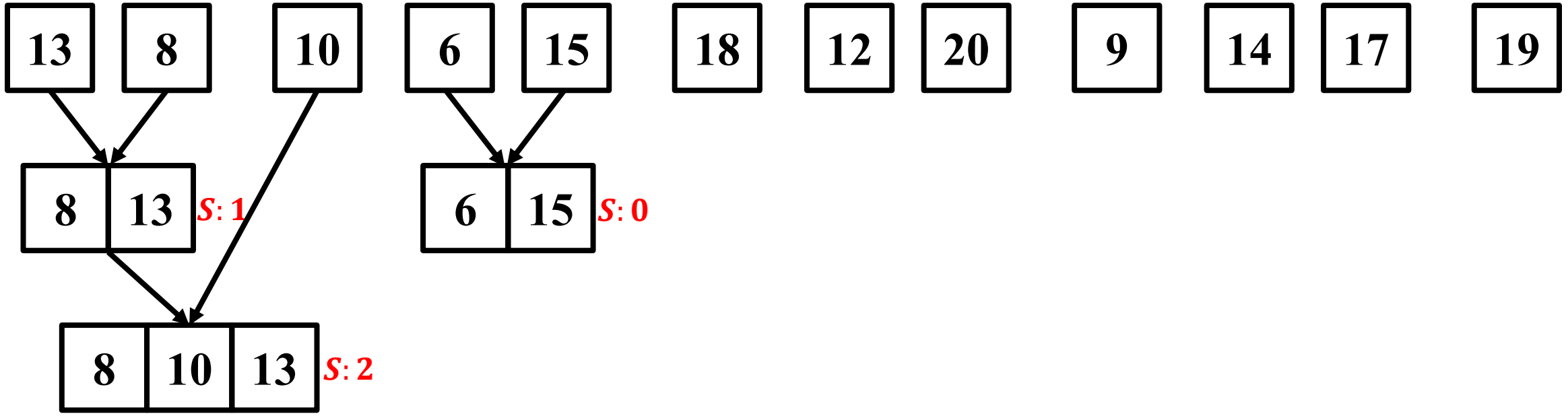
算法实例



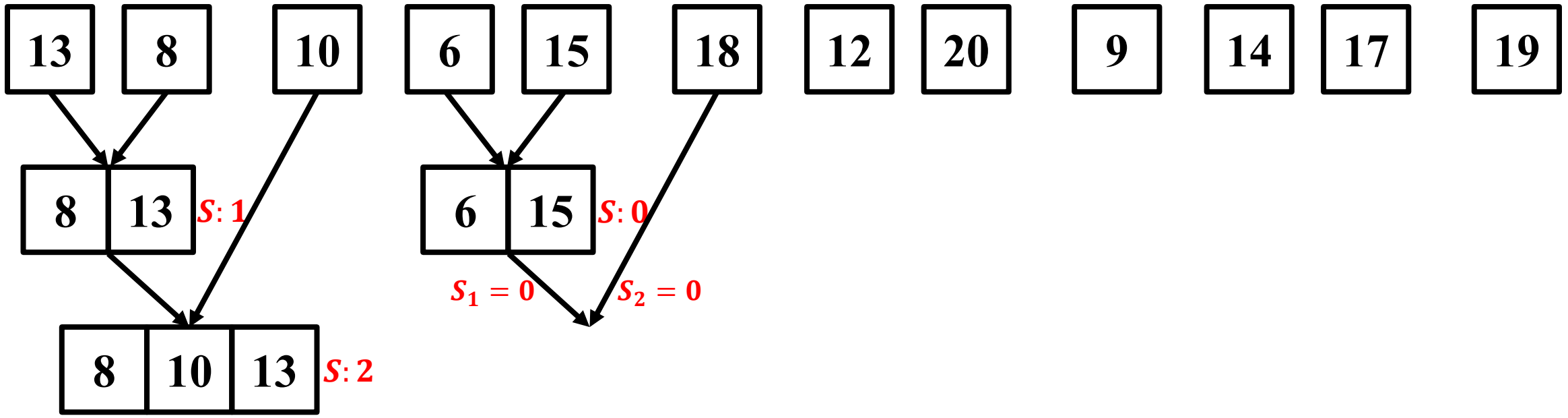
算法实例



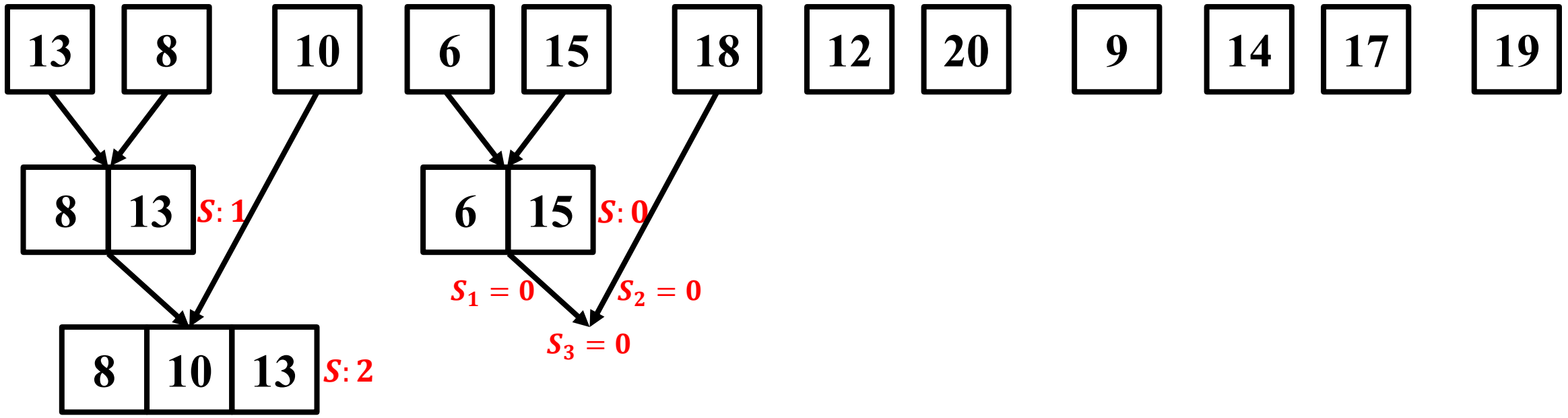
算法实例



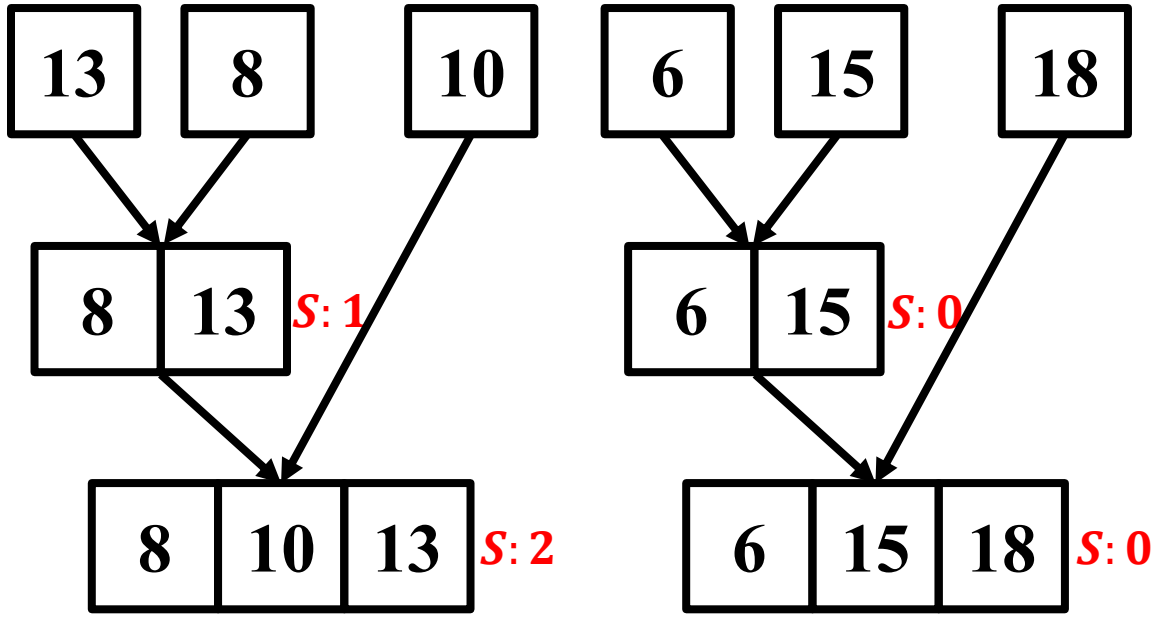
算法实例



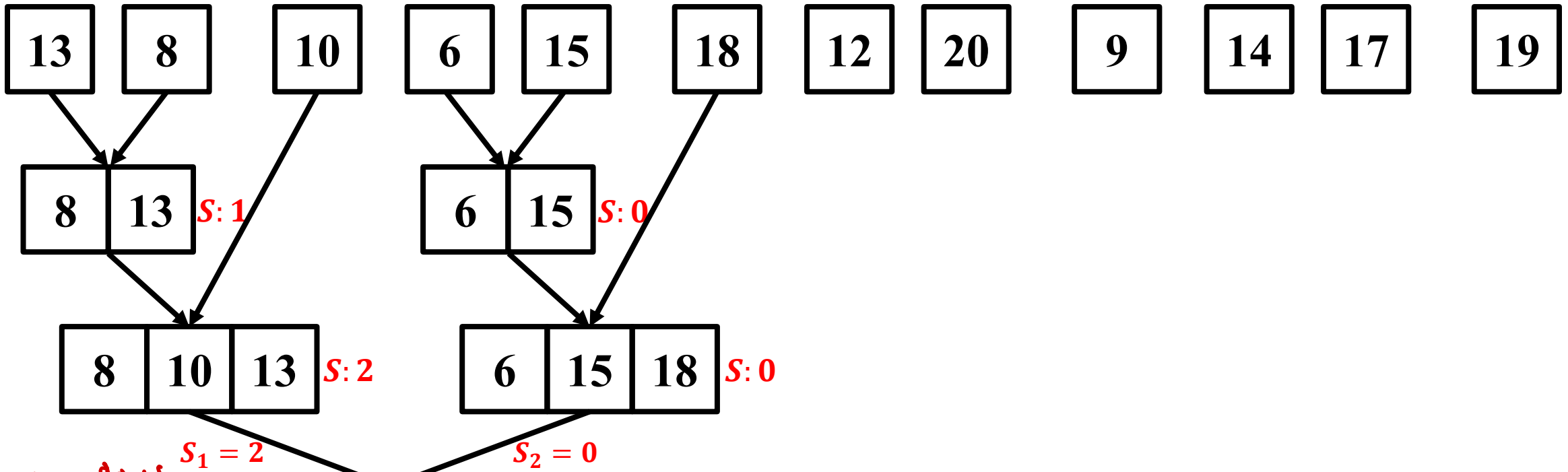
算法实例



算法实例



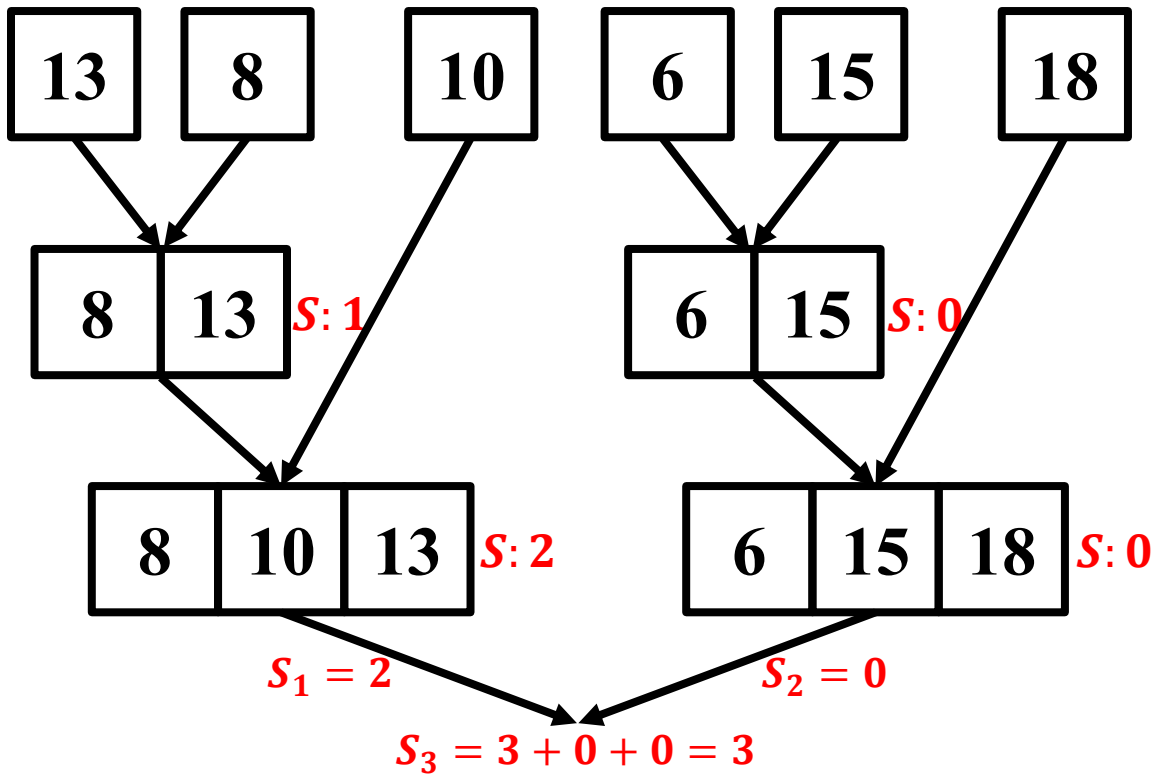
算法实例



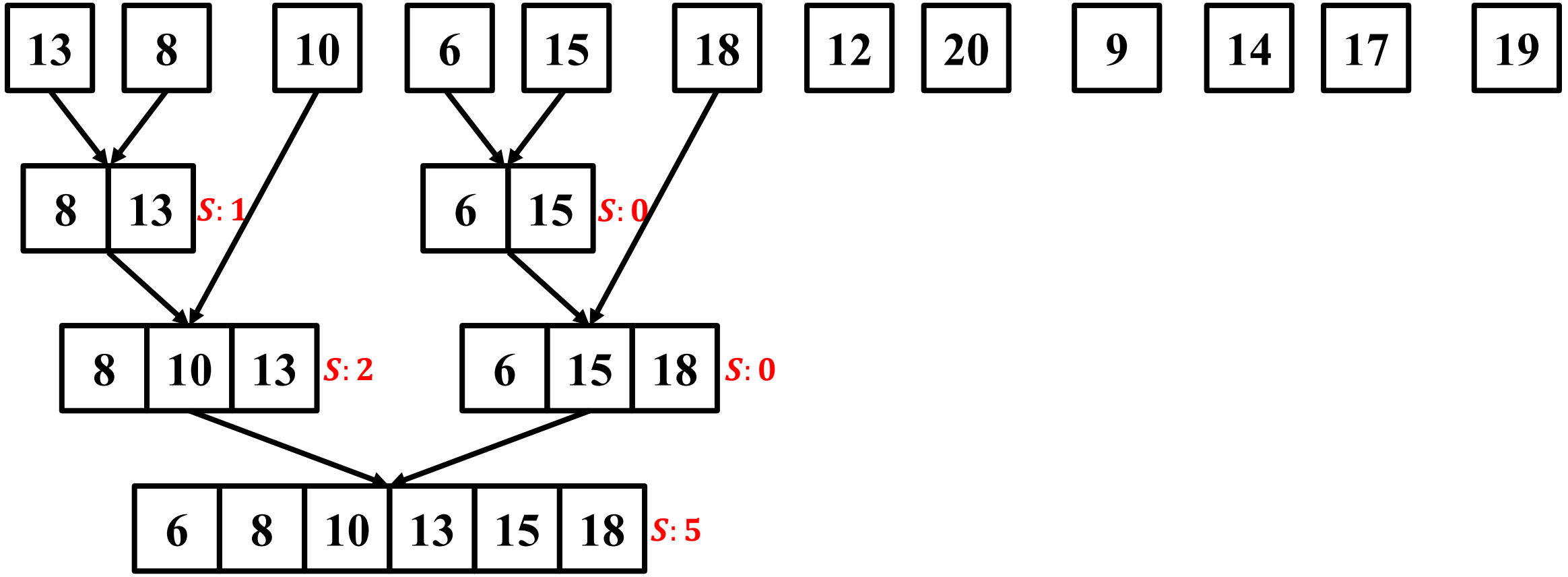
上一步求解的S=2
作为这一步的输入(直接作S1的结果)

左子串重排也不影响S1的计算.

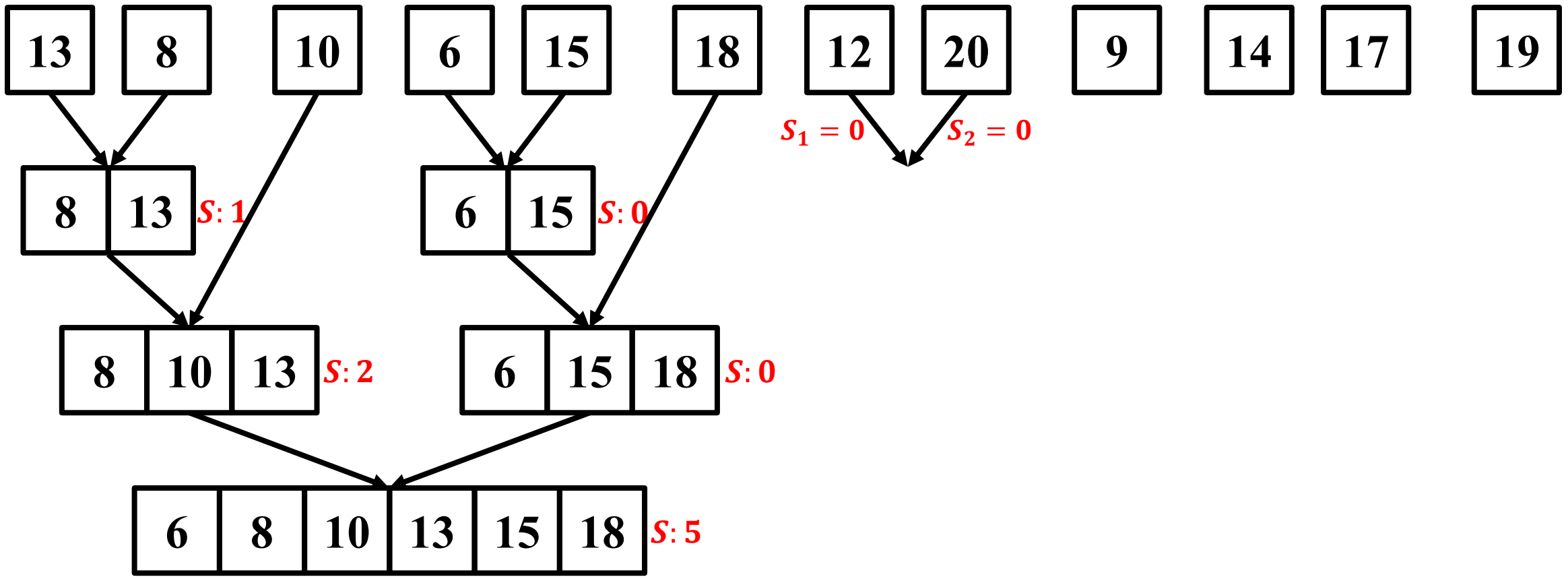
算法实例


13
8
10
6
15
18
12
20
9
14
17
19
8
13
S: 1
6
15
S: 0
8
10
13
S: 2
6
15
18
S: 0
$S_1 = 2$
$S_2 = 0$
$S_3 = 3 + 0 + 0 = 3$

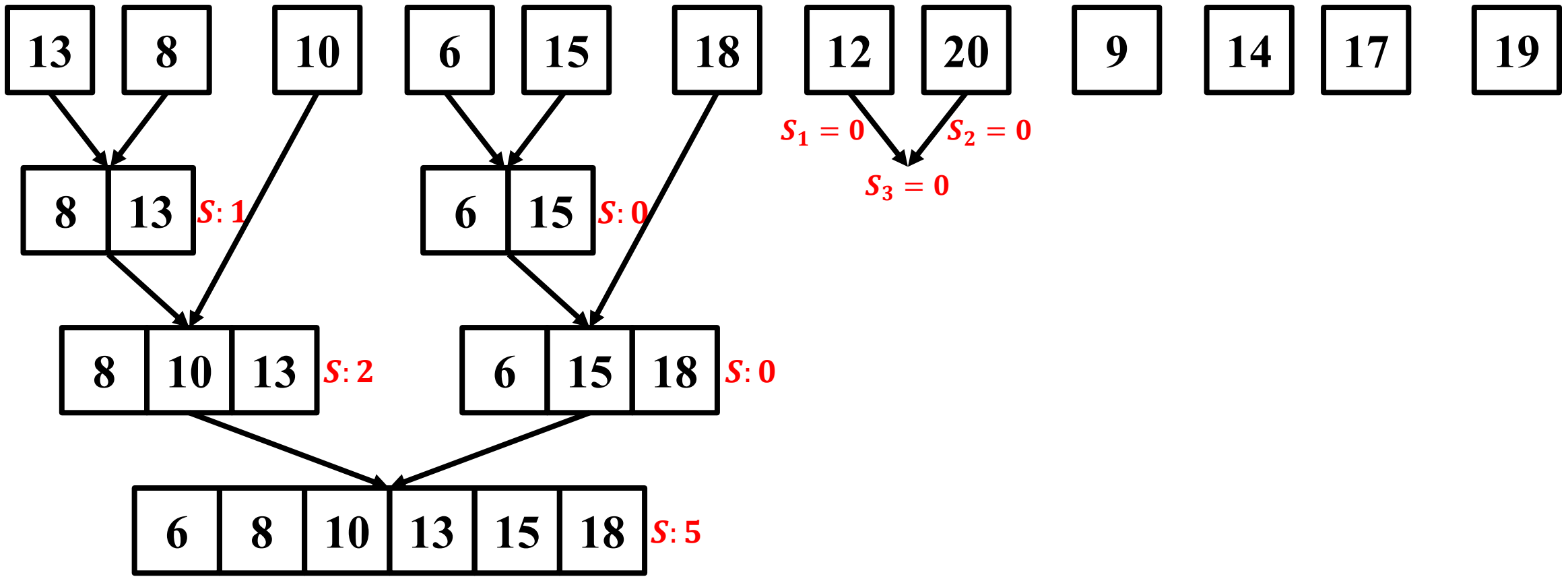
算法实例


13
8
10
6
15
18
12
20
9
14
17
19
8
13
S: 1
6
15
S: 0
8
10
13
S: 2
6
15
18
S: 0
6
8
10
13
15
18
S: 5

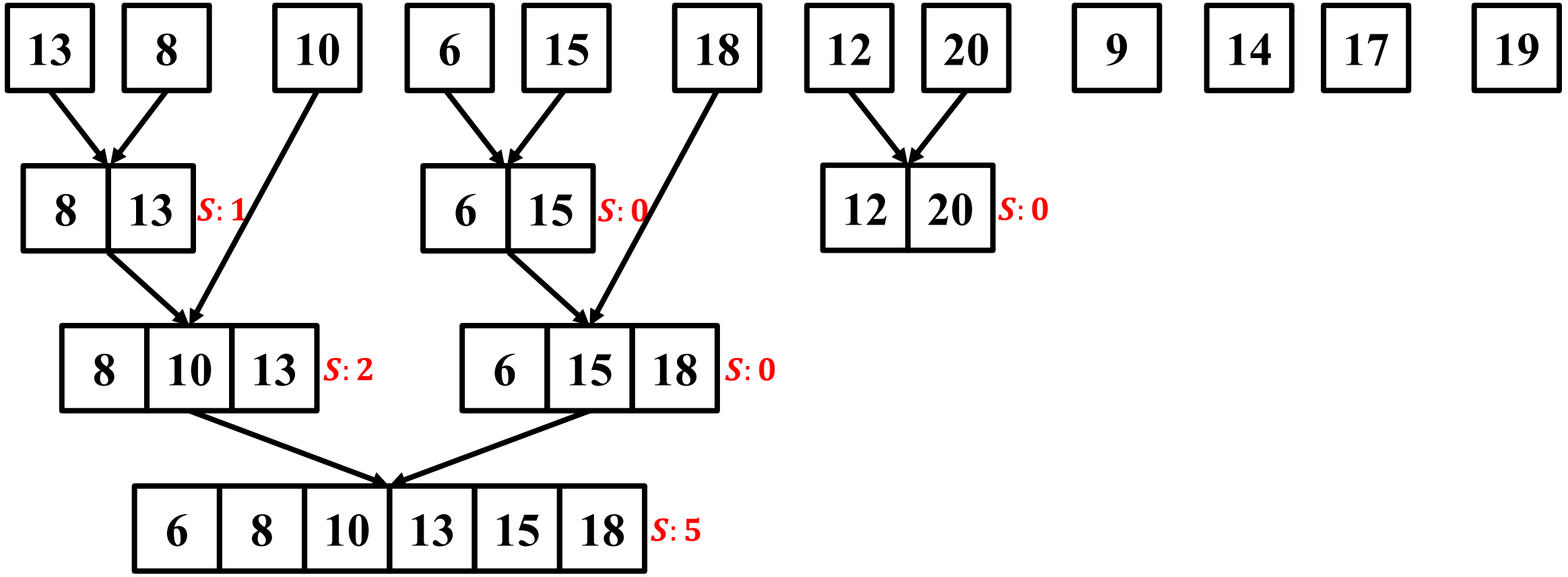
算法实例


13
8
10
6
15
18
12
20
9
14
17
19
$S_1 = 0$
$S_2 = 0$
8
13
S: 1
6
15
S: 0
8
10
13
S: 2
6
15
18
S: 0
6
8
10
13
15
18
S: 5

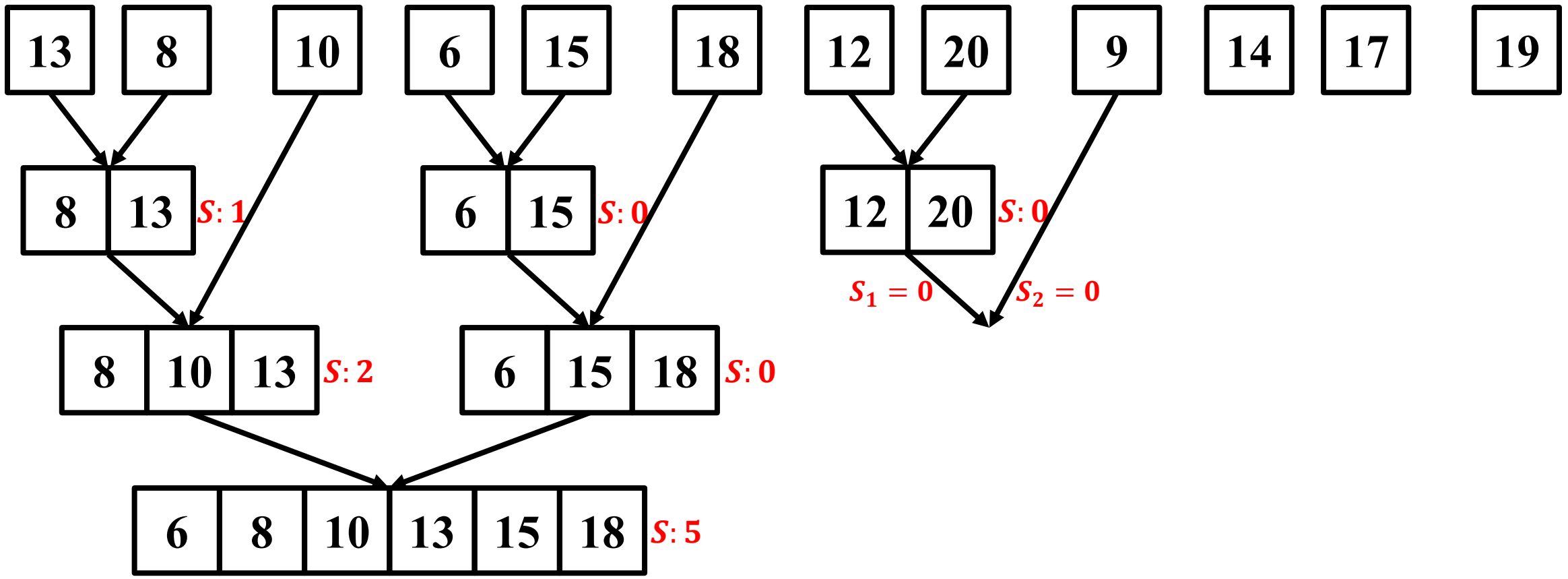
算法实例


13
8
10
6
15
18
12
20
9
14
17
19
$S_1 = 0$
$S_2 = 0$
$S_3 = 0$
8
13
S: 1
6
15
S: 0
8
10
13
S: 2
6
15
18
S: 0
6
8
10
13
15
18
S: 5

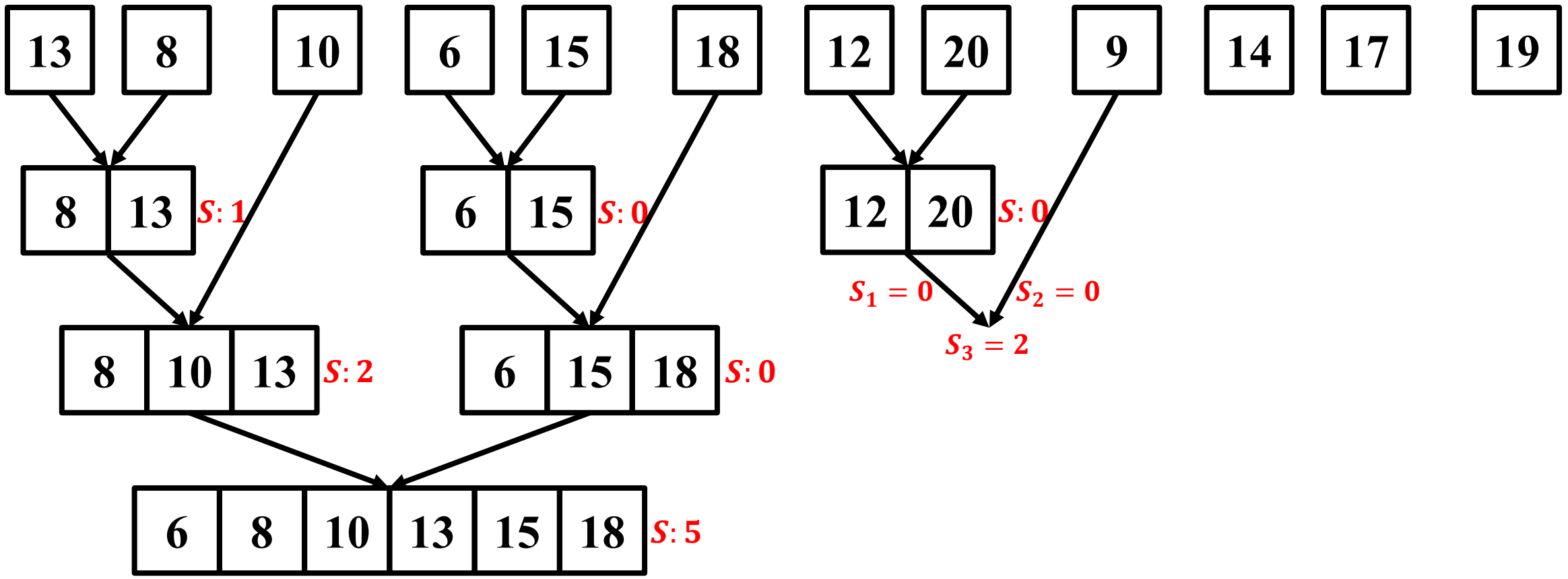
算法实例


13
8
10
6
15
18
12
20
9
14
17
19
8
13
S: 1
6
15
S: 0
12
20
S: 0
8
10
13
S: 2
6
15
18
S: 0
6
8
10
13
15
18
S: 5

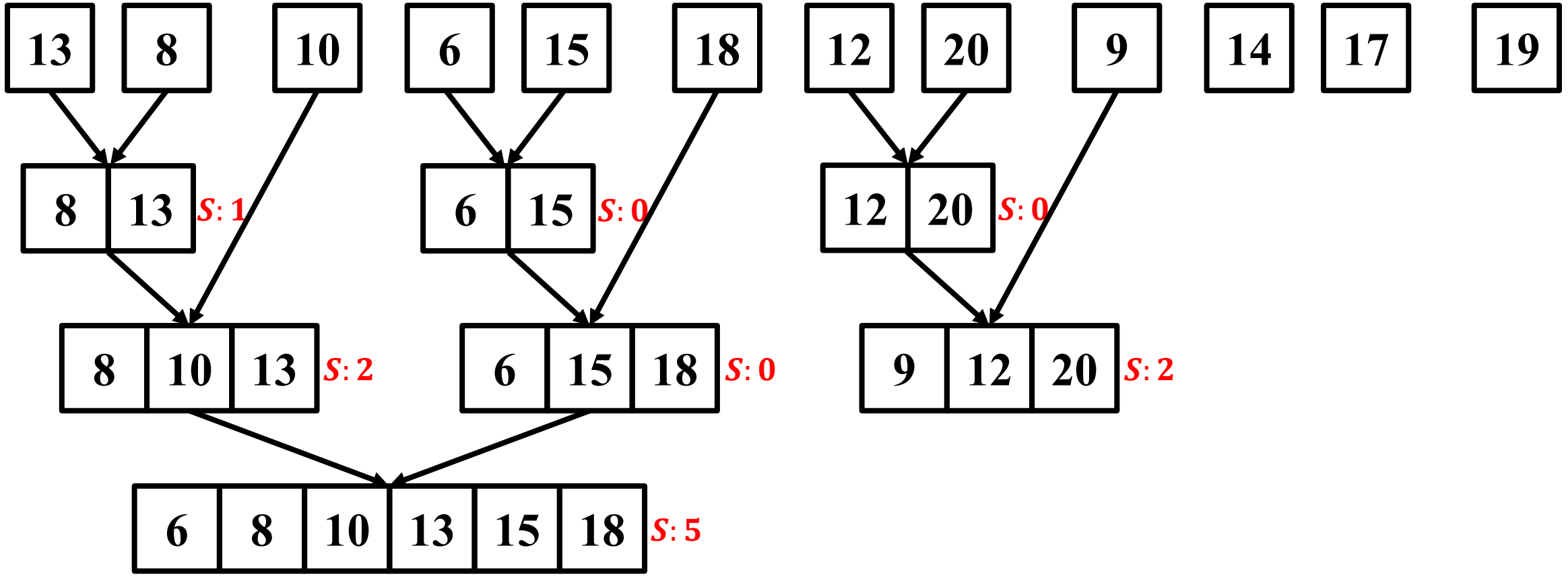
算法实例


13
8
10
6
15
18
12
20
9
14
17
19
8
13
S: 1
6
15
S: 0
12
20
S: 0
S1 = 0
S2 = 0
8
10
13
S: 2
6
15
18
S: 0
6
8
10
13
15
18
S: 5

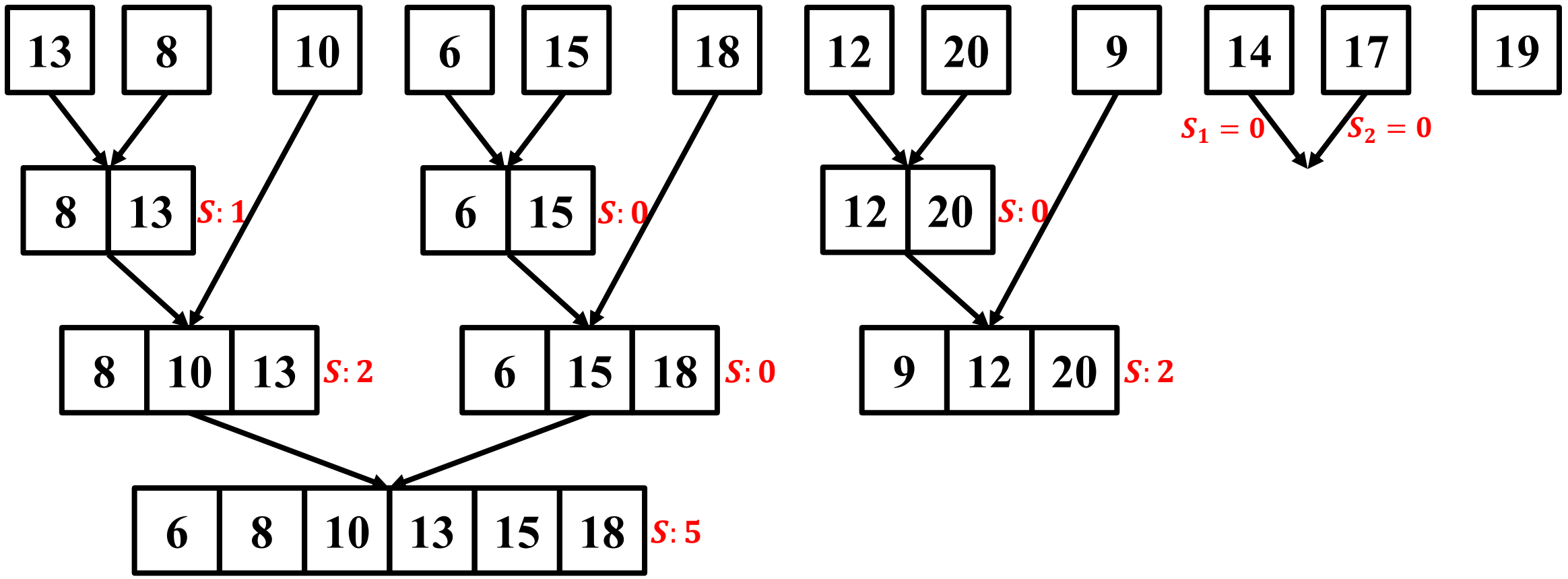
算法实例


13
8
10
6
15
18
12
20
9
14
17
19
8
13
S: 1
6
15
S: 0
12
20
S: 0
S1 = 0
S2 = 0
S3 = 2
8
10
13
S: 2
6
15
18
S: 0
6
8
10
13
15
18
S: 5

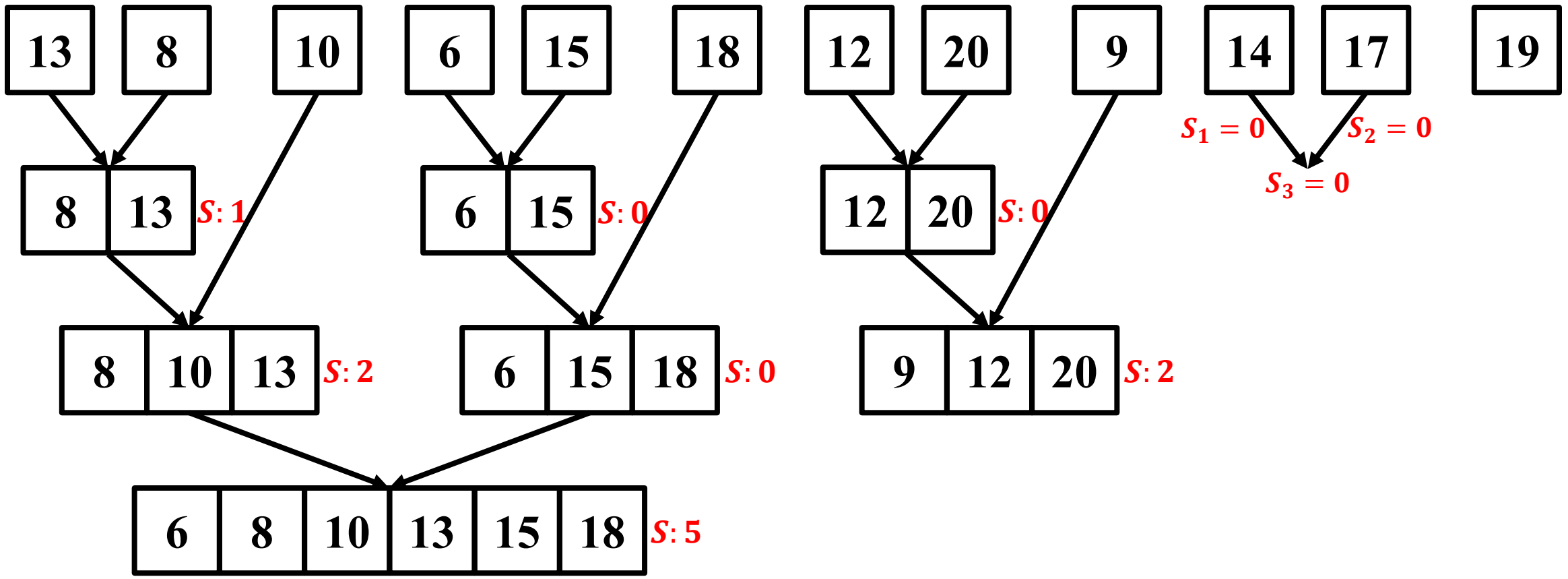
算法实例


13
8
10
6
15
18
12
20
9
14
17
19
8
13
S: 1
6
15
S: 0
12
20
S: 0
8
10
13
S: 2
6
15
18
S: 0
9
12
20
S: 2
6
8
10
13
15
18
S: 5

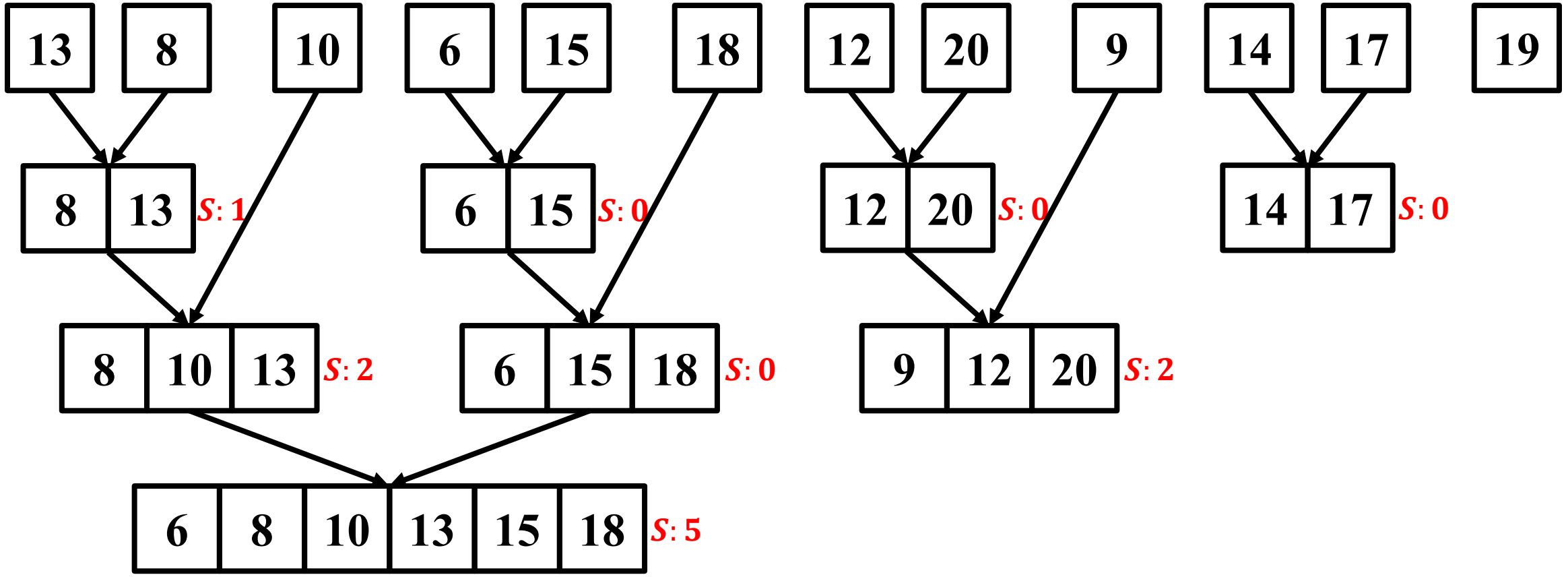
算法实例


13
8
10
6
15
18
12
20
9
14
17
19
$S_1 = 0$
$S_2 = 0$
8
13
S: 1
6
15
S: 0
12
20
S: 0
8
10
13
S: 2
6
15
18
S: 0
9
12
20
S: 2
6
8
10
13
15
18
S: 5

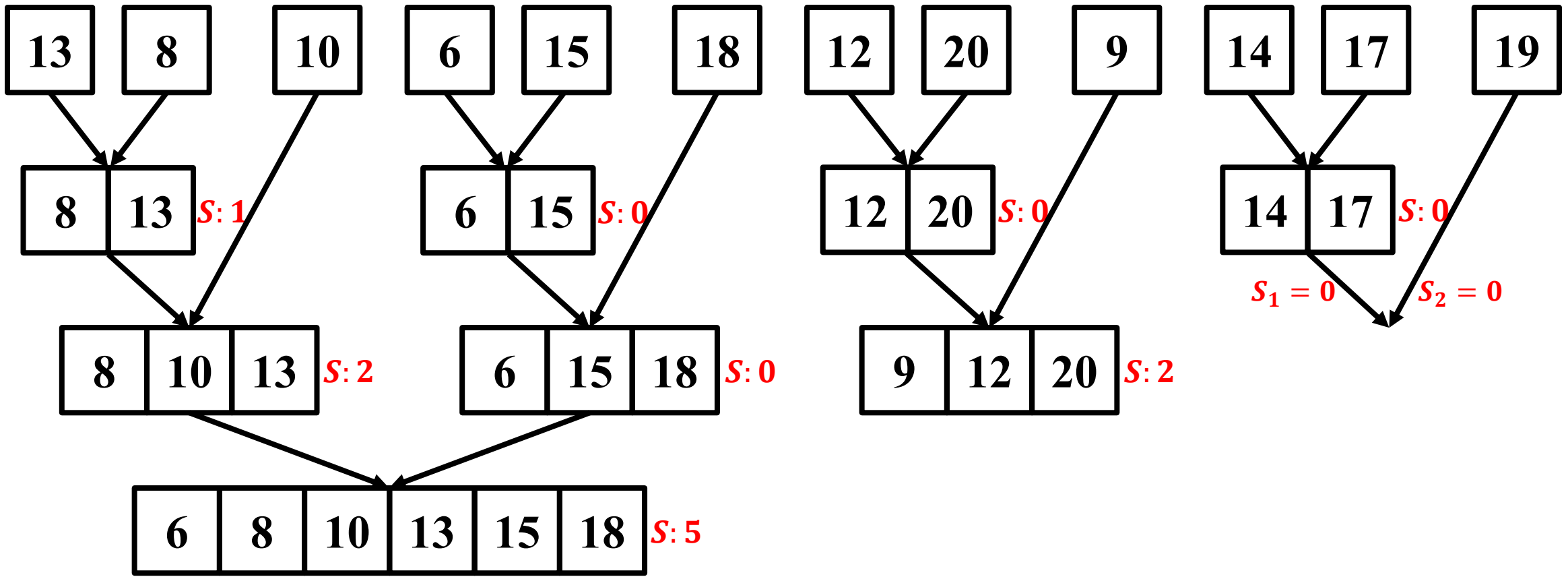
算法实例


13
8
10
6
15
18
12
20
9
14
17
19
$S_1 = 0$
$S_2 = 0$
$S_3 = 0$
8
13
S: 1
6
15
S: 0
12
20
S: 0
8
10
13
S: 2
6
15
18
S: 0
9
12
20
S: 2
6
8
10
13
15
18
S: 5

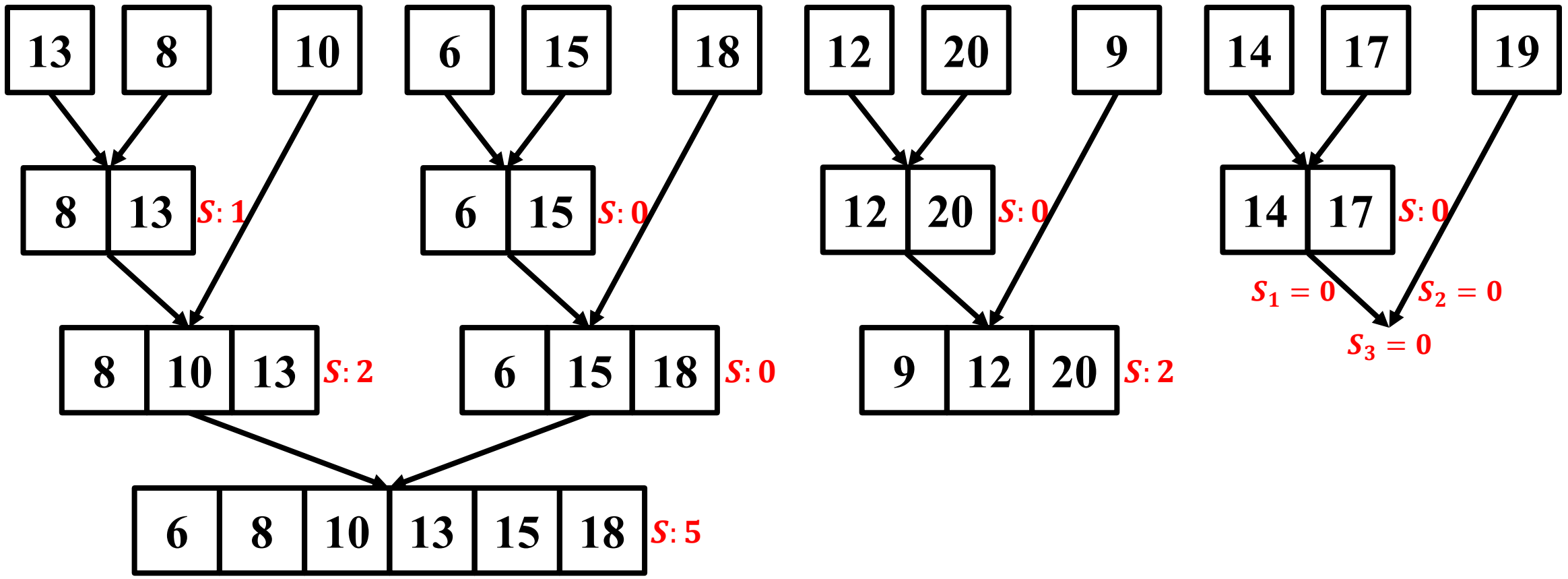
算法实例


13
8
10
6
15
18
12
20
9
14
17
19
8
13
S: 1
6
15
S: 0
12
20
S: 0
14
17
S: 0
8
10
13
S: 2
6
15
18
S: 0
9
12
20
S: 2
6
8
10
13
15
18
S: 5

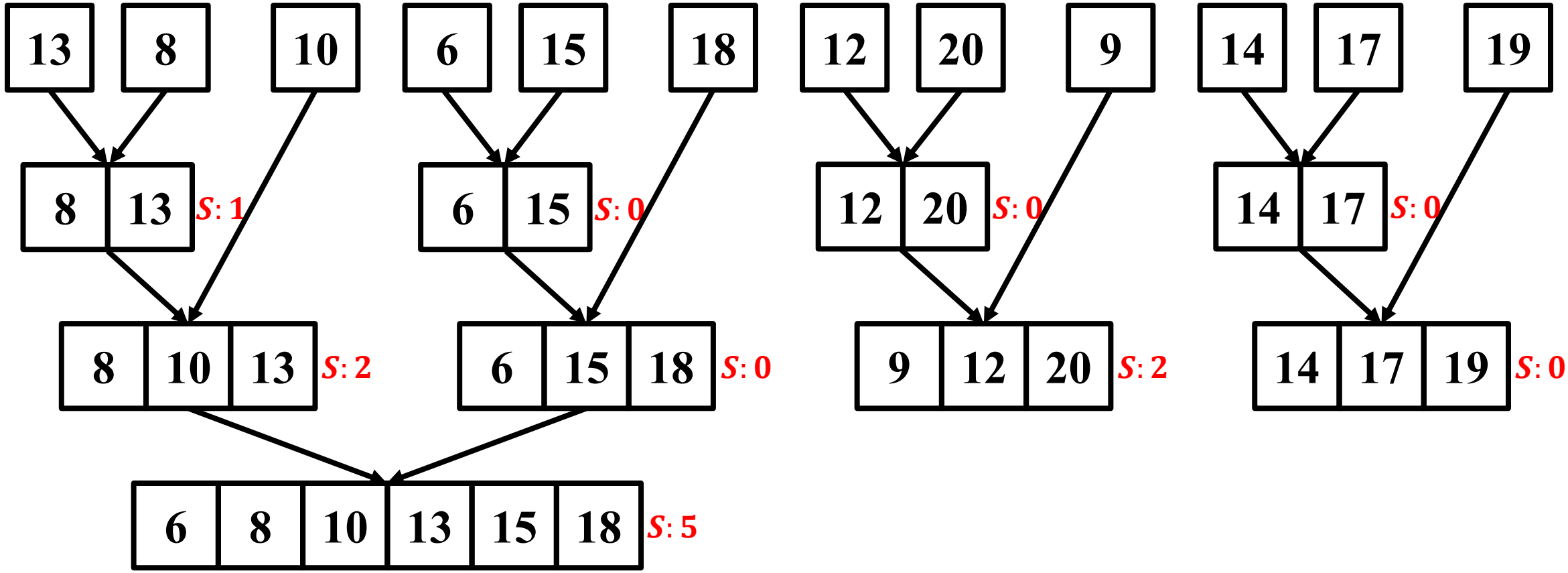
算法实例


13
8
10
6
15
18
12
20
9
14
17
19
8
13
S: 1
6
15
S: 0
12
20
S: 0
14
17
S: 0
$S_1 = 0$
$S_2 = 0$
8
10
13
S: 2
6
15
18
S: 0
9
12
20
S: 2
6
8
10
13
15
18
S: 5

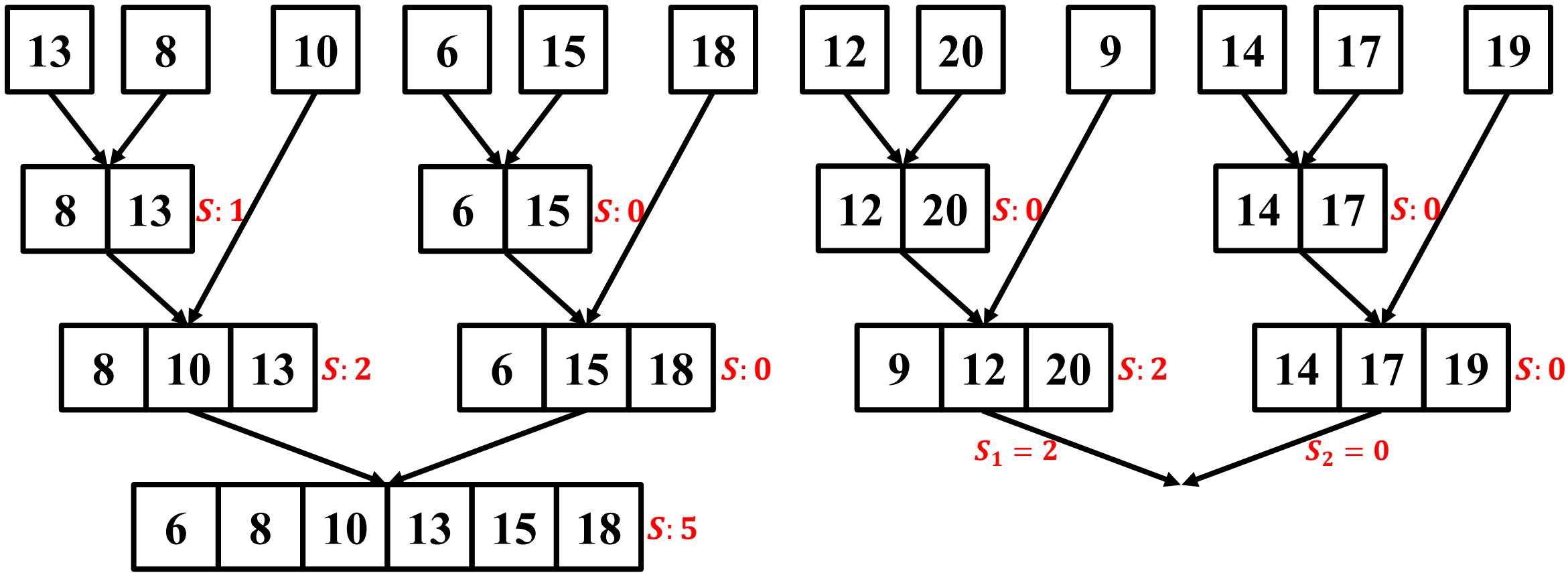
算法实例


13
8
10
6
15
18
12
20
9
14
17
19
8
13
S: 1
6
15
S: 0
12
20
S: 0
14
17
S: 0
S1 = 0
S2 = 0
S3 = 0
8
10
13
S: 2
6
15
18
S: 0
9
12
20
S: 2
6
8
10
13
15
18
S: 5

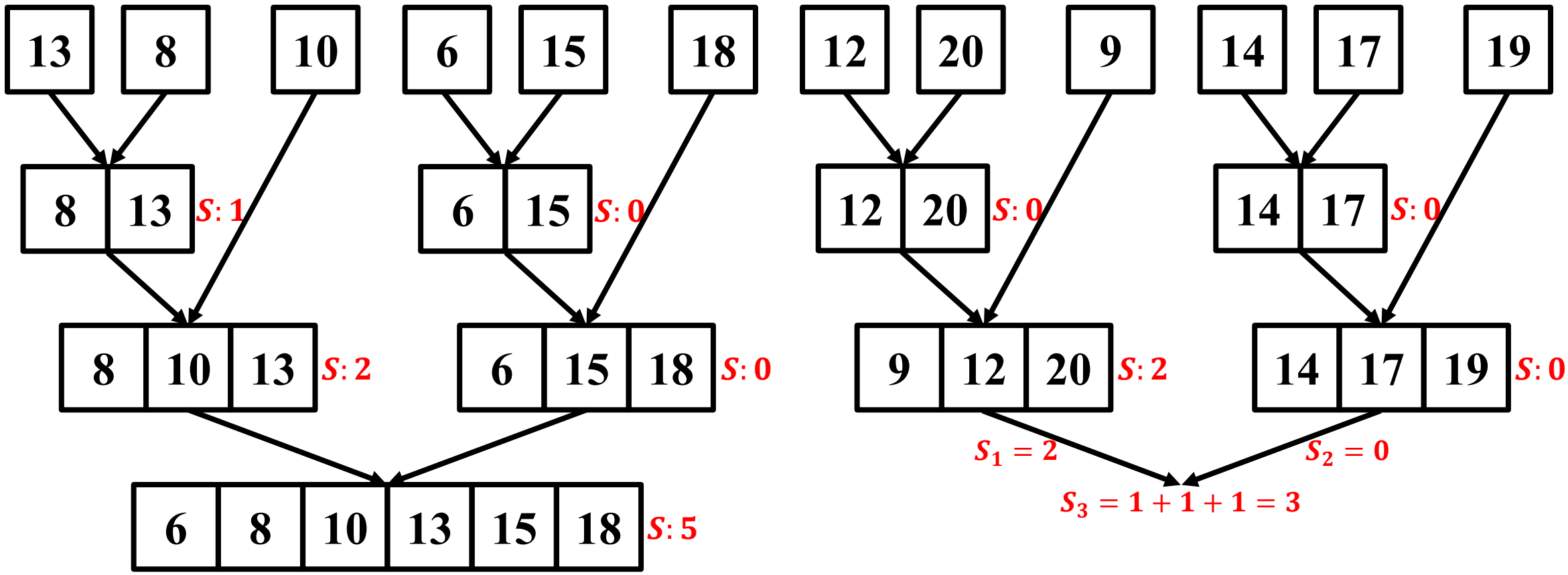
算法实例


13 8 10
8 13 S: 1
8 10 13 S: 2
6 15 18
6 15 S: 0
6 15 18 S: 0
12 20 9
12 20 S: 0
9 12 20 S: 2
14 17 19
14 17 S: 0
14 17 19 S: 0
6 8 10 13 15 18 S: 5

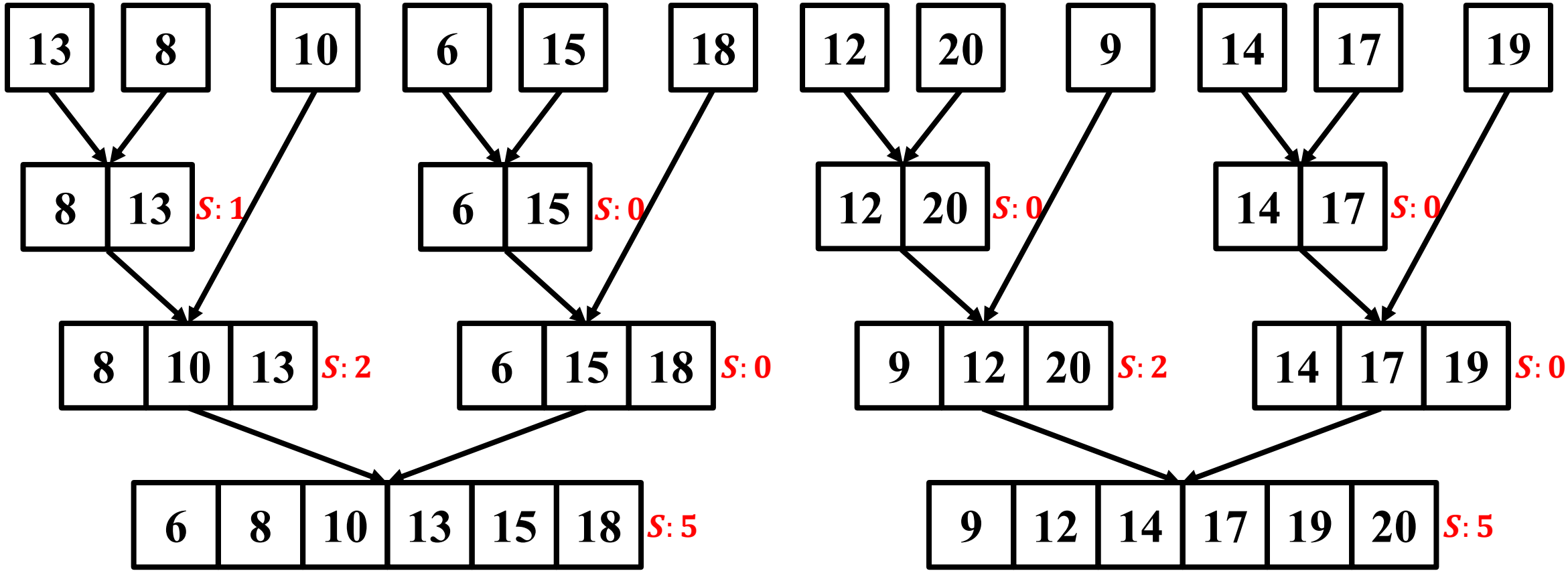
算法实例


13
8
10
6
15
18
12
20
9
14
17
19
8
13
S: 1
6
15
S: 0
12
20
S: 0
14
17
S: 0
8
10
13
S: 2
6
15
18
S: 0
9
12
20
S: 2
14
17
19
S: 0
S1 = 2
S2 = 0
6
8
10
13
15
18
S: 5

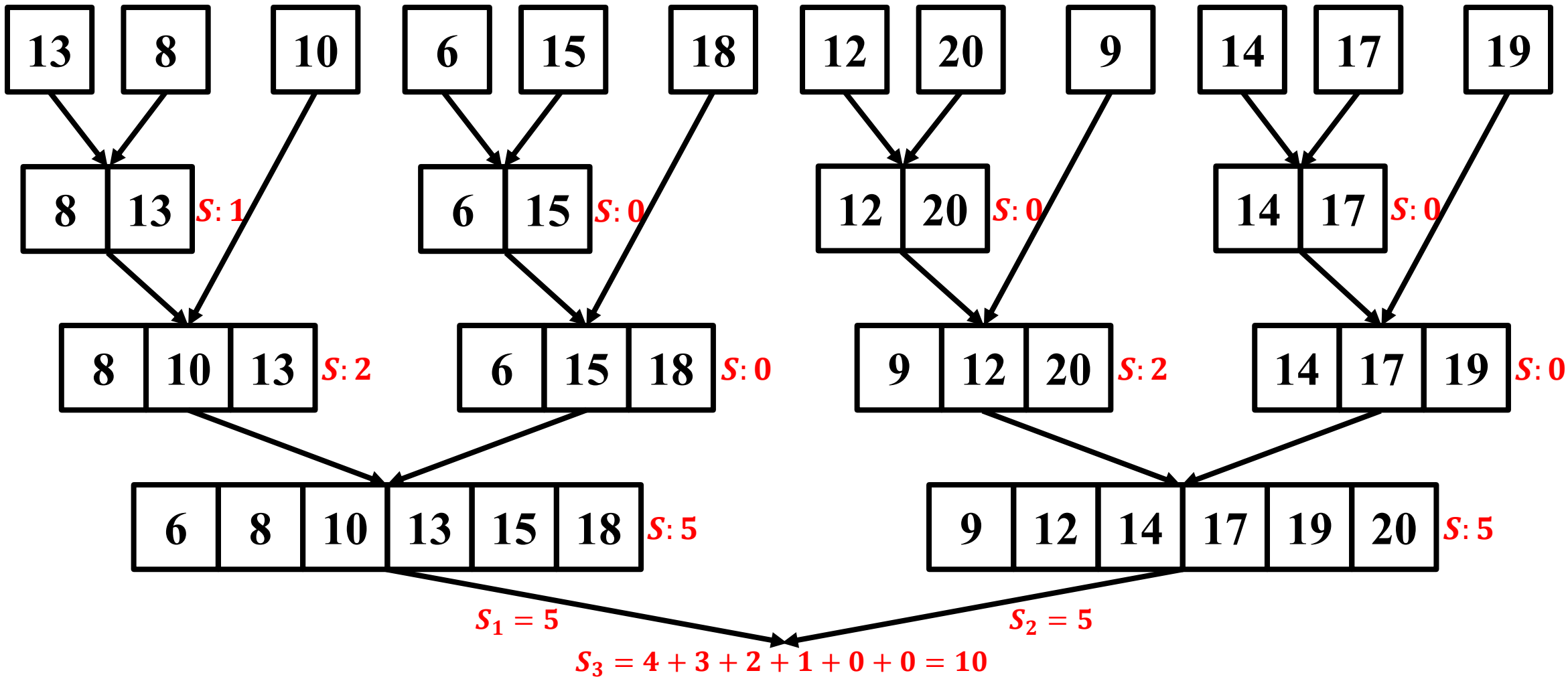
算法实例


13 8 10 6 15 18 12 20 9 14 17 19
8 13 S: 1
6 15 S: 0
12 20 S: 0
14 17 S: 0
8 10 13 S: 2
6 15 18 S: 0
9 12 20 S: 2
14 17 19 S: 0
S1 = 2
S2 = 0
6 8 10 13 15 18 S: 5
S3 = 1 + 1 + 1 = 3

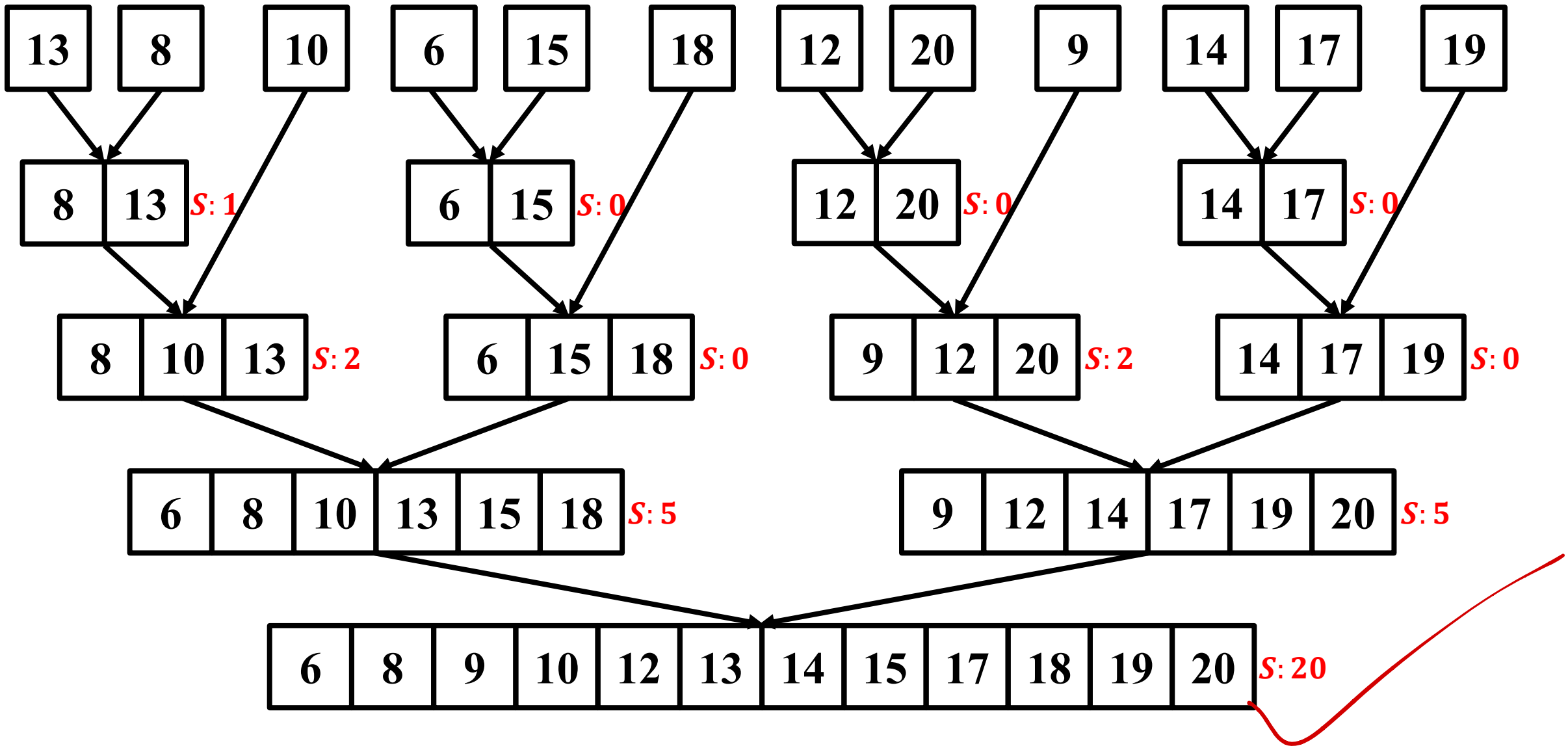
算法实例


13 8 10 6 15 18 12 20 9 14 17 19
8 13 S: 1
6 15 S: 0
12 20 S: 0
14 17 S: 0
8 10 13 S: 2
6 15 18 S: 0
9 12 20 S: 2
14 17 19 S: 0
6 8 10 13 15 18 S: 5
9 12 14 17 19 20 S: 5

算法实例


13 8 10
6 15 18
12 20 9
14 17 19
8 13 S: 1
6 15 S: 0
12 20 S: 0
14 17 S: 0
8 10 13 S: 2
6 15 18 S: 0
9 12 20 S: 2
14 17 19 S: 0
6 8 10 13 15 18 S: 5
9 12 14 17 19 20 S: 5
$S_1 = 5$
$S_2 = 5$
$S_3 = 4 + 3 + 2 + 1 + 0 + 0 = 10$

算法实例


13 8 10 6 15 18 12 20 9 14 17 19
8 13 S: 1
6 15 S: 0
12 20 S: 0
14 17 S: 0
8 10 13 S: 2
6 15 18 S: 0
9 12 20 S: 2
14 17 19 S: 0
6 8 10 13 15 18 S: 5
9 12 14 17 19 20 S: 5
6 8 9 10 12 13 14 15 17 18 19 20 S: 20

伪代码

- 归并求解： CountInver$(A, left, right)$

输入: 数组$A[1..n]$,数组下标$left, right$
输出: 数组$A[left..right]$的逆序对数，递增数组$A[left..right]$

```
if left ≥ right then
    return A[left..right]
end
mid ← ⌊(left+right)/2⌋
S1 ←CountInver(A, left, mid)
S2 ←CountInver(A, mid + 1, right)
S3 ←MergeCount(A, left, mid, right)
S ← S1 + S2 + S3
return S, A[left..right]
```

分解原问题

伪代码

- 归并求解： CountInver$(A, left, right)$

输入: 数组$A[1..n]$,数组下标$left, right$
输出: 数组$A[left..right]$的逆序对数，递增数组$A[left..right]$

```
if left ≥ right then
    return A[left..right]
end
mid ← ⌊(left+right)/2⌋
S1 ←CountInver(A, left, mid)
S2 ←CountInver(A, mid + 1, right)
S3 ←MergeCount(A, left, mid, right)
S ← S1 + S2 + S3
return S, A[left..right]
```

解决子问题

伪代码

- 归并求解： CountInver$(A, left, right)$

输入: 数组$A[1..n]$,数组下标$left, right$
输出: 数组$A[left..right]$的逆序对数，递增数组$A[left..right]$
if $left \geq right$ **then**
  return $A[left..right]$
end
$mid \leftarrow \lfloor \frac{left+right}{2} \rfloor$
$S_1 \leftarrow$CountInver$(A, left, mid)$
$S_2 \leftarrow$CountInver$(A, mid+1, right)$
$S_3 \leftarrow$MergeCount$(A, left, mid, right)$
$S \leftarrow S_1 + S_2 + S_3$
return $S, A[left..right]$

合并问题解

伪代码

- 归并求解： CountInver$(A, left, right)$

初始调用： CountInver $(A, 1, n)$

输入: 数组$A[1..n]$,数组下标$left, right$
输出: 数组$A[left..right]$的逆序对数，递增数组$A[left..right]$
if $left \geq right$ **then**
  return $A[left..right]$
end
$mid \leftarrow \lfloor \frac{left+right}{2} \rfloor$
$S_1 \leftarrow$CountInver$(A, left, mid)$
$S_2 \leftarrow$CountInver$(A, mid+1, right)$
$S_3 \leftarrow$MergeCount$(A, left, mid, right)$
$S \leftarrow S_1 + S_2 + S_3$
return $S, A[left..right]$

时间复杂度分析

- 输入规模为：n

$$T(n)=\begin{cases}1, & n=1\\ 2\cdot T(n/2)+O(n), & n>1\end{cases}$$

输入: 数组$A[1..n]$,数组下标$left, right$
输出: 数组$A[left..right]$的逆序对数，递增数组$A[left..right]$

```
if left ≥ right then
    return A[left..right]
end
mid ← ⌊(left+right)/2⌋
S1 ←CountInver(A, left, mid)
S2 ←CountInver(A, mid + 1, right)
S3 ←MergeCount(A, left, mid, right)
S ← S1 + S2 + S3
return S, A[left..right]
```

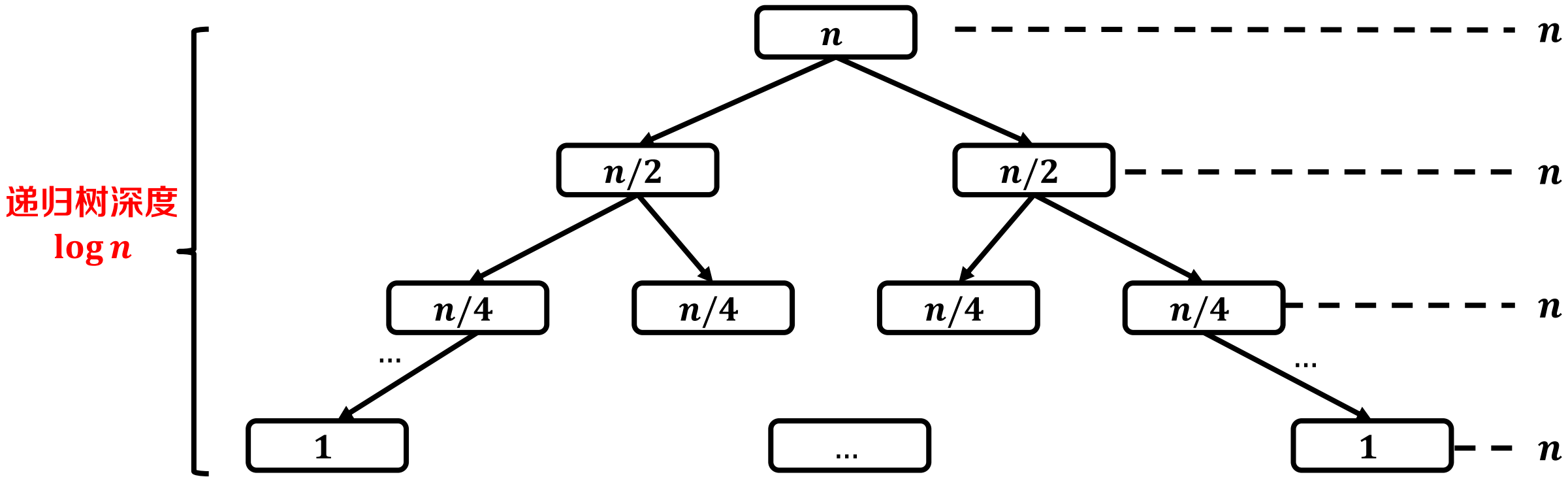
S_1 ←CountInver$(A, left, mid)$
S_2 ←CountInver$(A, mid+1, right)$
S_3 ←MergeCount$(A, left, mid, right)$

$2\cdot T(n/2)$

$O(n)$

时间复杂度分析

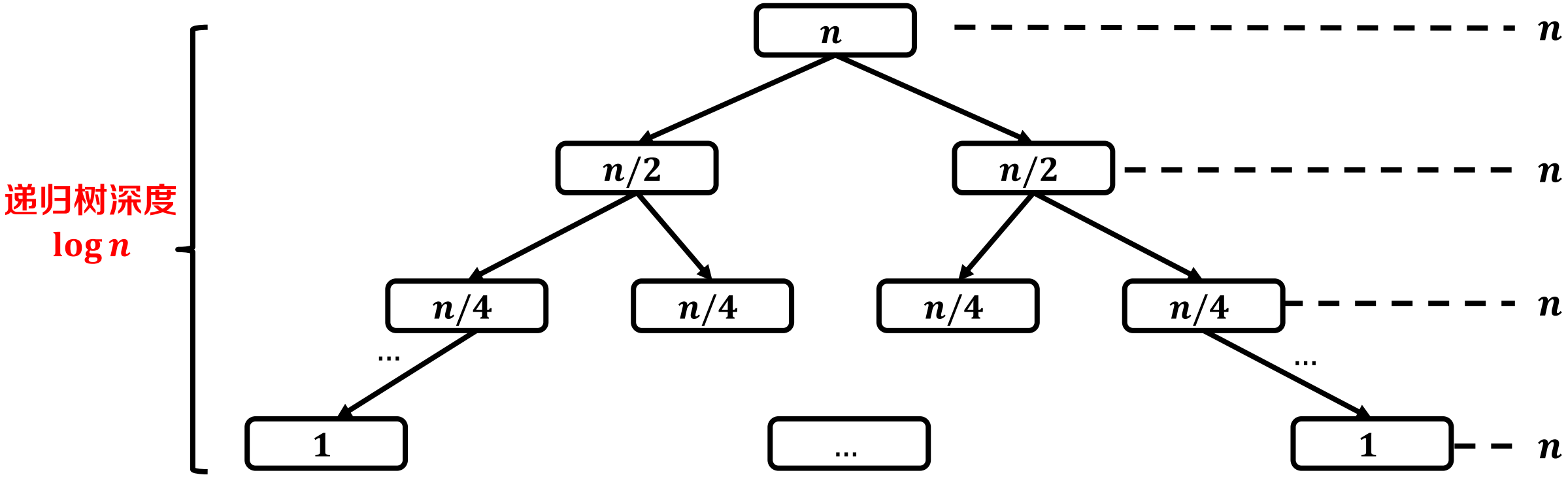
- 输入规模为：n
  - $T(n) = \begin{cases} 1, & n = 1 \\ 2 \cdot T(n/2) + O(n), & n > 1 \end{cases}$



时间复杂度分析

- 输入规模为：n
  - $$T(n)=\begin{cases}1, & n=1\\ 2\cdot T(n/2)+O(n), & n>1\end{cases}$$

$$T(n)=O(n\log n)$$



小结

- 在本问题中，我们设计了四个算法：

算法名称	合并求解复杂度	时间复杂度
蛮力枚举	–	$O(n^2)$
分而治之+直接计算	$O(n^2)$	$O(n^2)$
分而治之+排序求解	$O(n\ log\ n)$	$O(n\ log^2 n)$
分而治之+归并求解	$O(n)$	$O(n\ log\ n)$

小结

- 在本问题中，我们设计了四个算法：

算法名称	合并求解复杂度	时间复杂度
蛮力枚举	–	$O(n^2)$
分而治之+直接计算	$O(n^2)$	$O(n^2)$
分而治之+排序求解	$O(n \log n)$	$O(n \log^2 n)$
分而治之+归并求解	$O(n)$	$O(n \log n)$

问题：分治策略关键是什么？
答案：合理设计合并求解算法