

# 分而治之篇：归并排序

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》

- 2008年北京奥运会

- 中国举重健儿们刻苦训练、顽强拼搏、勇破纪录
- 北京航空航天大学为举重项目提供了支持与保障



龙清泉 男子举重56公斤级冠军



北航体育馆承办奥运举重项目

- 杠铃增重问题

- 每位参赛运动员向组委会提交**排好序**的三次试举重量

125	130	132
-----	-----	-----



123	127	129
-----	-----	-----



117	121	126
-----	-----	-----

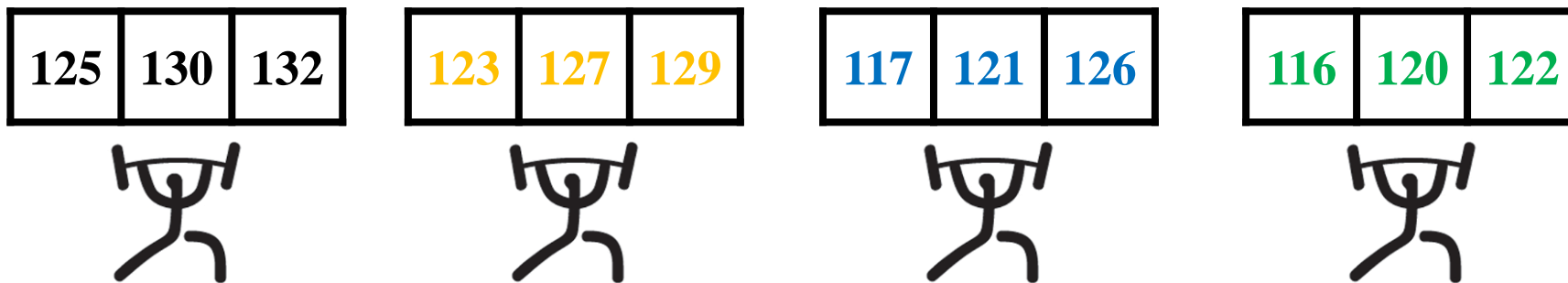


116	120	122
-----	-----	-----

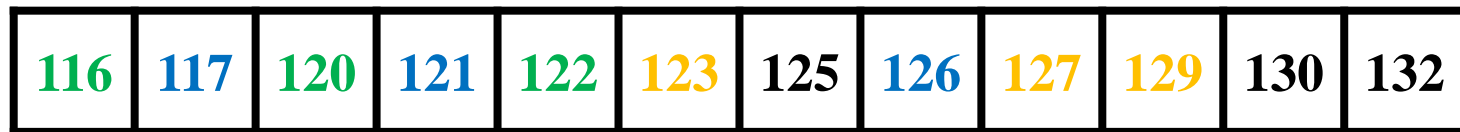


- 杠铃增重问题

- 每位参赛运动员向组委会提交**排好序**的三次试举重量
- 为便于杠铃拆卸，组委会需对**所有**试举重量**递增排序**

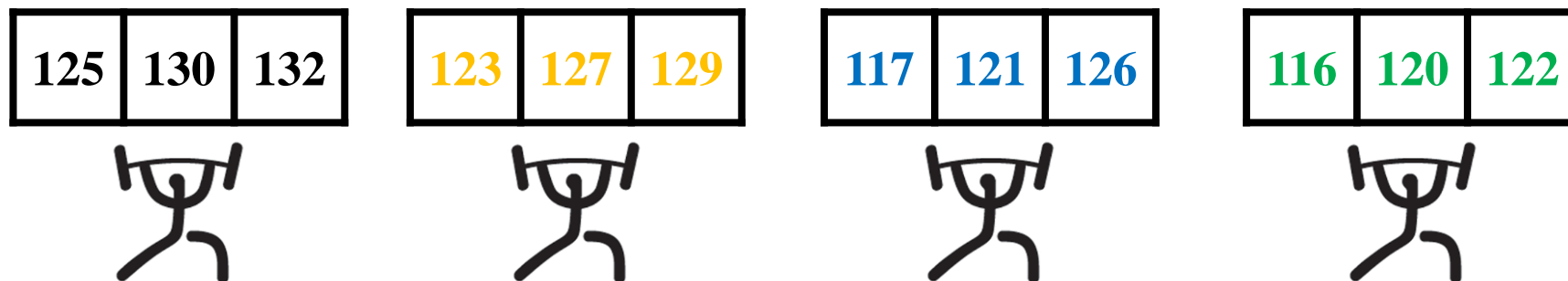


杠铃增重顺序:



- 杠铃增重问题

- 每位参赛运动员向组委会提交**排好序**的三次试举重量
- 为便于杠铃拆卸，组委会需对**所有**试举重量**递增排序**

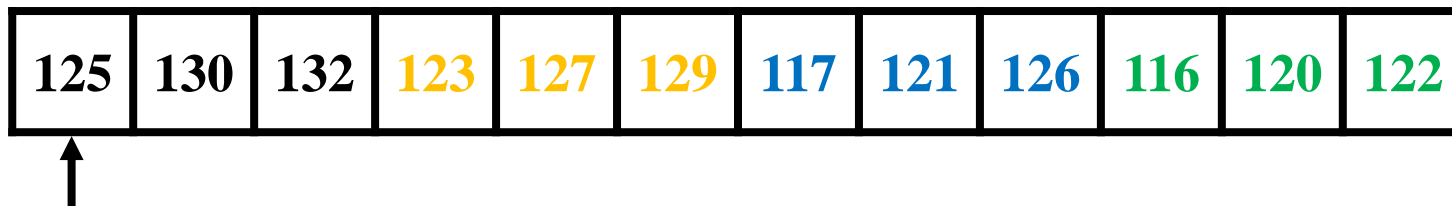


杠铃增重顺序: 

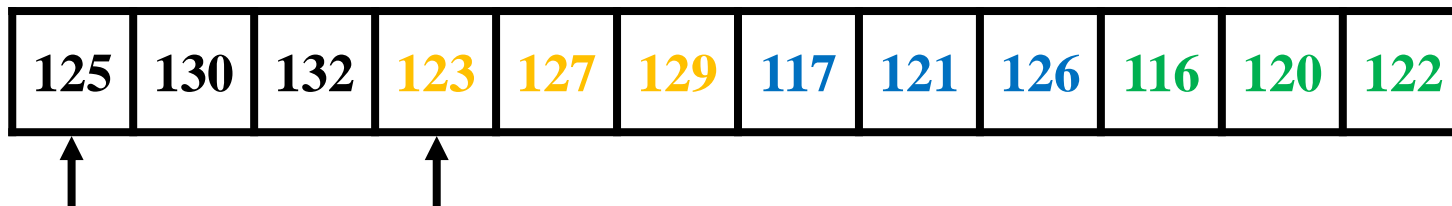
116	117	120	121	122	123	125	126	127	129	130	132
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

问题：组委会如何根据试举重量安排杠铃增重顺序？

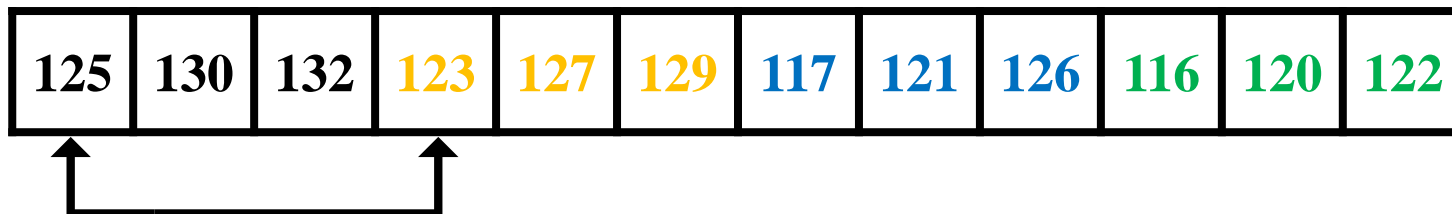
- 杠铃增重问题
  - 选择排序
    - 从待排序元素中迭代选出最小值并排序



- 杠铃增重问题
  - 选择排序
    - 从待排序元素中迭代选出最小值并排序

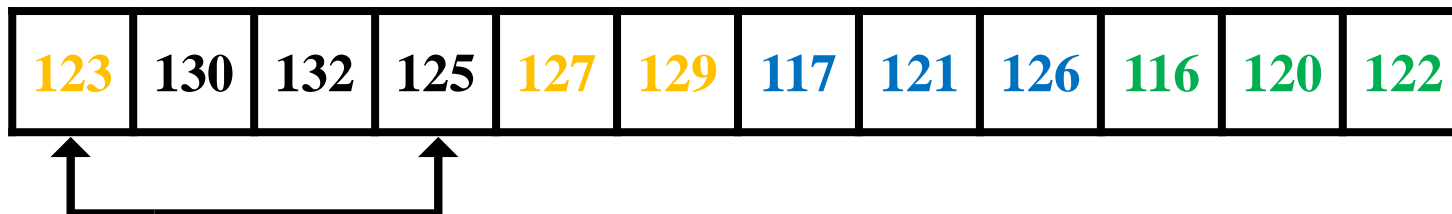


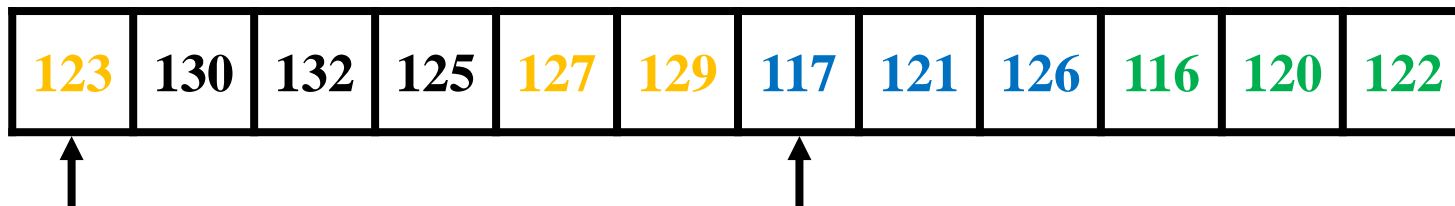
- 杠铃增重问题
  - 选择排序
    - 从待排序元素中迭代选出最小值并排序



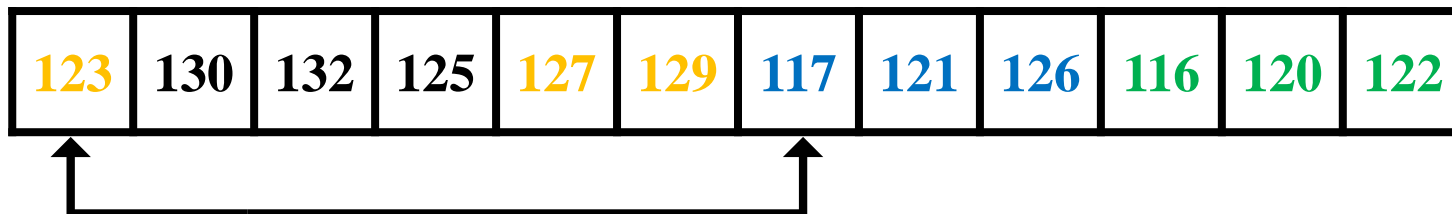


- 杠铃增重问题
  - 选择排序
    - 从待排序元素中迭代选出最小值并排序

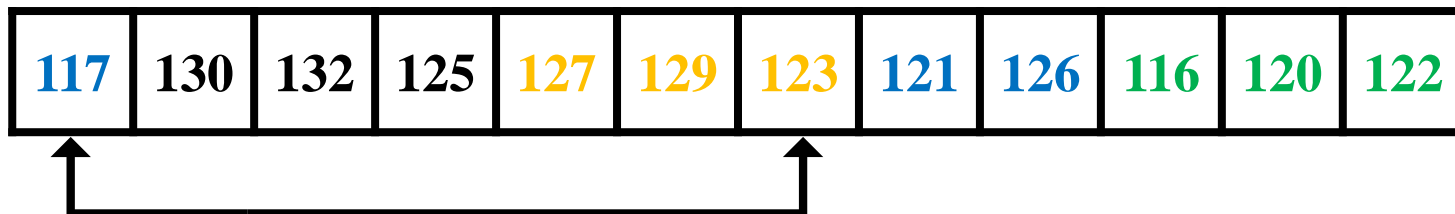




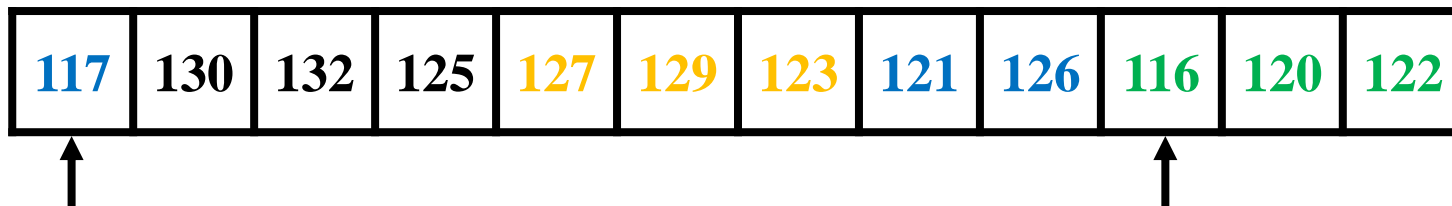
- 杠铃增重问题
  - 选择排序
    - 从待排序元素中迭代选出最小值并排序



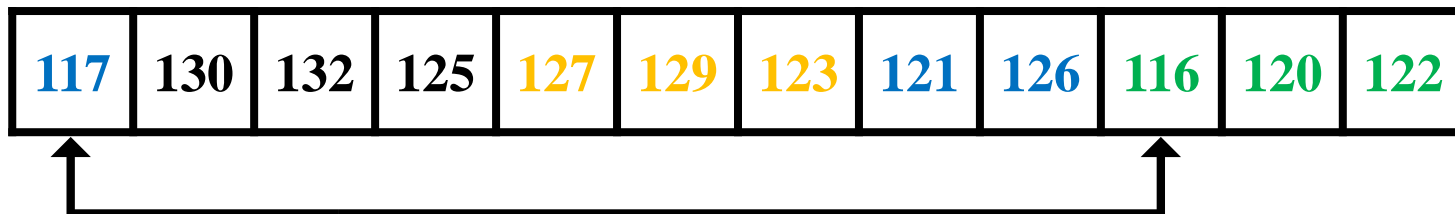
- 杠铃增重问题
  - 选择排序
    - 从待排序元素中迭代选出最小值并排序



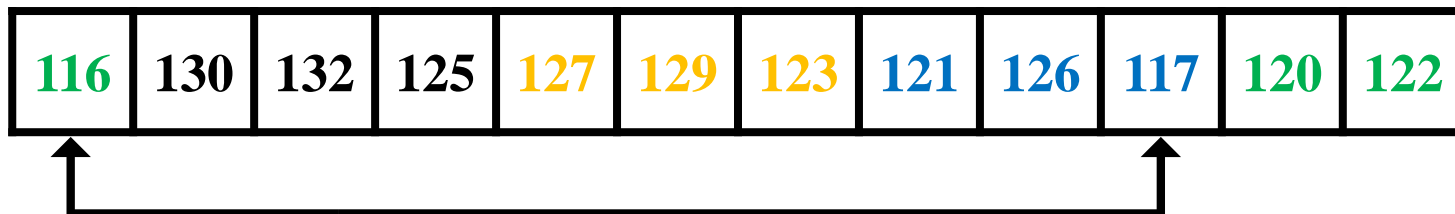
- 杠铃增重问题
  - 选择排序
    - 从待排序元素中迭代选出最小值并排序



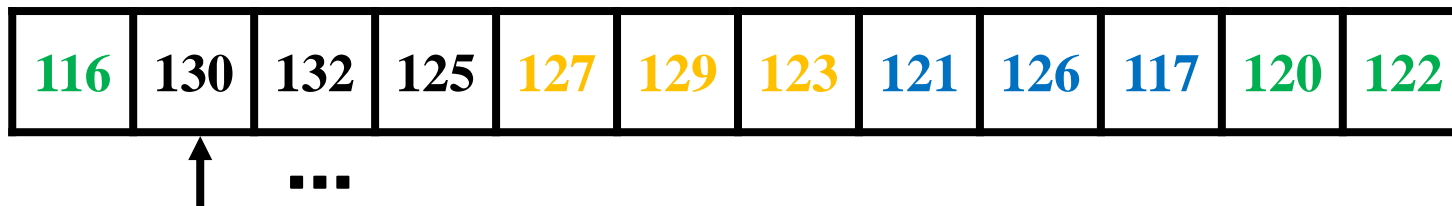
- 杠铃增重问题
  - 选择排序
    - 从待排序元素中迭代选出最小值并排序



- 杠铃增重问题
  - 选择排序
    - 从待排序元素中迭代选出最小值并排序

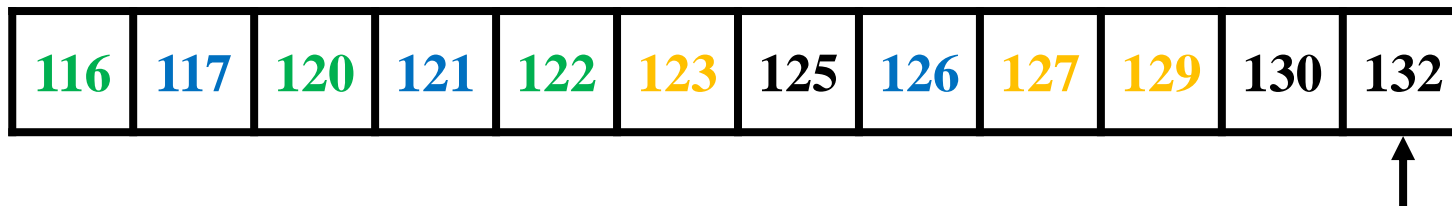


- 杠铃增重问题
  - 选择排序
    - 从待排序元素中迭代选出最小值并排序





- 杠铃增重问题
  - 选择排序
    - 从待排序元素中迭代选出最小值并排序



- 杠铃增重问题

- 选择排序

- 从待排序元素中迭代选出最小值并排序
    - 比较次数：66次

116	117	120	121	122	123	125	126	127	129	130	132
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

- 杠铃增重问题

- 选择排序

- 从待排序元素中迭代选出最小值并排序
    - 比较次数：66次

116	117	120	121	122	123	125	126	127	129	130	132
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

- 插入排序

- 依次将每个元素插入到已排序序列之中

125	130	132	123	127	129	117	121	126	116	120	122
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

↑  
key

- 杠铃增重问题

- 选择排序

- 从待排序元素中迭代选出最小值并排序
    - 比较次数：66次

116	117	120	121	122	123	125	126	127	129	130	132
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

- 插入排序

- 依次将每个元素插入到已排序序列之中

125	130	132	123	127	129	117	121	126	116	120	122
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

↑  
key

- 杠铃增重问题

- 选择排序


- 从待排序元素中迭代选出最小值并排序
    - 比较次数：66次

116	117	120	121	122	123	125	126	127	129	130	132
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

- 插入排序

- 依次将每个元素插入到已排序序列之中

125	130	132	123	127	129	117	121	126	116	120	122
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----



- 杠铃增重问题

- 选择排序


- 从待排序元素中迭代选出最小值并排序
    - 比较次数：66次

116	117	120	121	122	123	125	126	127	129	130	132
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

- 插入排序

- 依次将每个元素插入到已排序序列之中

123	125	130	132	127	129	117	121	126	116	120	122
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----



- 杠铃增重问题

- 选择排序

- 从待排序元素中迭代选出最小值并排序
    - 比较次数：66次

116	117	120	121	122	123	125	126	127	129	130	132
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

- 插入排序

- 依次将每个元素插入到已排序序列之中

123	125	130	132	127	129	117	121	126	116	120	122
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

↑  
key

- 杠铃增重问题

- 选择排序


- 从待排序元素中迭代选出最小值并排序
    - 比较次数：66次

116	117	120	121	122	123	125	126	127	129	130	132
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

- 插入排序

- 依次将每个元素插入到已排序序列之中

123	125	130	132	127	129	117	121	126	116	120	122
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----





- 杠铃增重问题

- 选择排序


- 从待排序元素中迭代选出最小值并排序
    - 比较次数：66次

116	117	120	121	122	123	125	126	127	129	130	132
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

- 插入排序

- 依次将每个元素插入到已排序序列之中

123	127	125	130	132	129	117	121	126	116	120	122
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----



- 杠铃增重问题

- 选择排序

- 从待排序元素中迭代选出最小值并排序
    - 比较次数：66次

116	117	120	121	122	123	125	126	127	129	130	132
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

- 插入排序

- 依次将每个元素插入到已排序序列之中

123	127	125	130	132	129	117	121	126	116	120	122
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

↑  
key

- 杠铃增重问题

- 选择排序

- 从待排序元素中迭代选出最小值并排序
    - 比较次数：66次

116	117	120	121	122	123	125	126	127	129	130	132
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

- 插入排序

- 依次将每个元素插入到已排序序列之中

123	127	125	130	132	129	117	121	126	116	120	122
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

↑  
key

...

- 杠铃增重问题

- 选择排序

- 从待排序元素中迭代选出最小值并排序
    - 比较次数：66次

116	117	120	121	122	123	125	126	127	129	130	132
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

- 插入排序

- 依次将每个元素插入到已排序序列之中

116	117	120	121	122	123	125	126	127	129	130	132
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

↑  
key

- 杠铃增重问题

- 选择排序

- 从待排序元素中迭代选出最小值并排序
- 比较次数：66次

116	117	120	121	122	123	125	126	127	129	130	132
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

- 插入排序

- 依次将每个元素插入到已排序序列之中
- 比较次数：55次

116	117	120	121	122	123	125	126	127	129	130	132
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

- 杠铃增重问题

- 选择排序

- 从待排序元素中迭代选出最小值并排序
    - 比较次数：66次

116	117	120	121	122	123	125	126	127	129	130	132
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

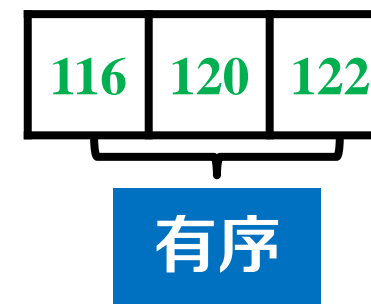
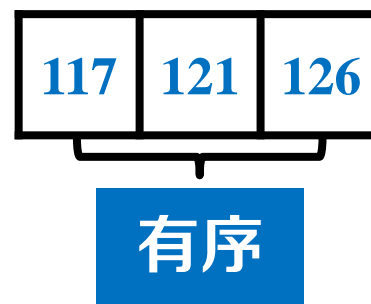
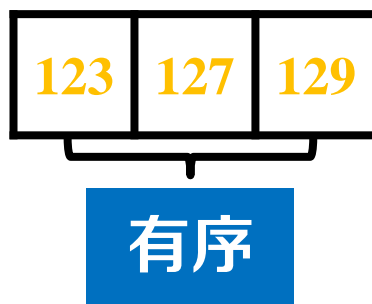
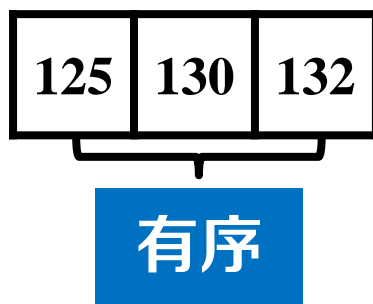
- 插入排序

- 依次将每个元素插入到已排序序列之中
    - 比较次数：55次

116	117	120	121	122	123	125	126	127	129	130	132
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

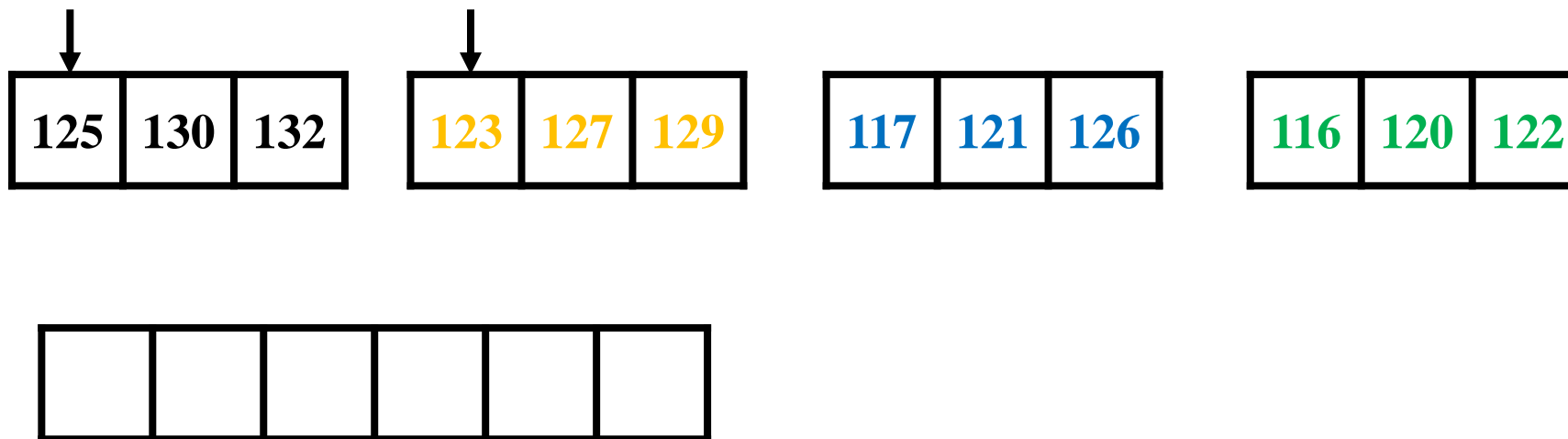
问题：是否还有更高效的算法？

- 杠铃增重问题
  - 问题特点：局部有序



- 杠铃增重问题

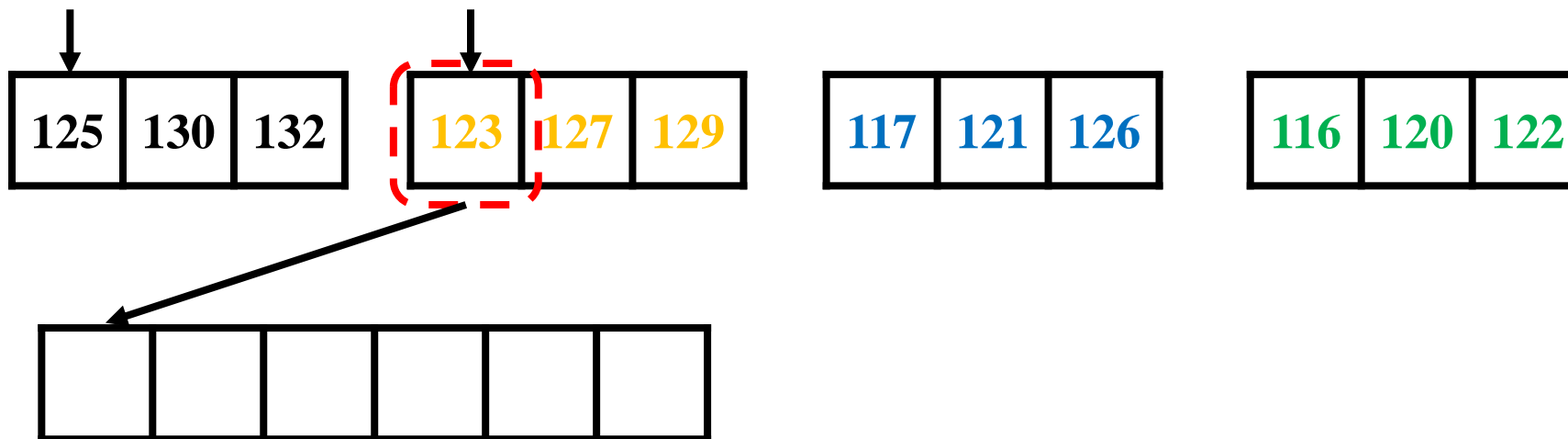
- 问题特点：局部有序
- 快速合并：比较两有序数组当前最小元素，将较小者逐一合入新数组





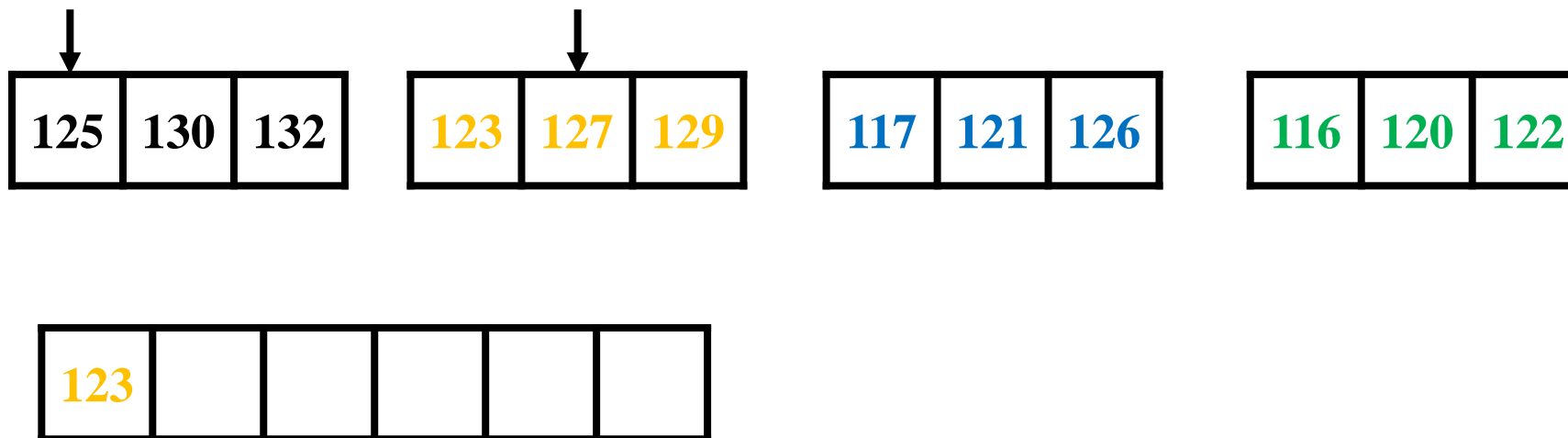
- 杠铃增重问题

- 问题特点：局部有序
- 快速合并：比较两有序数组当前最小元素，将较小者逐一合入新数组



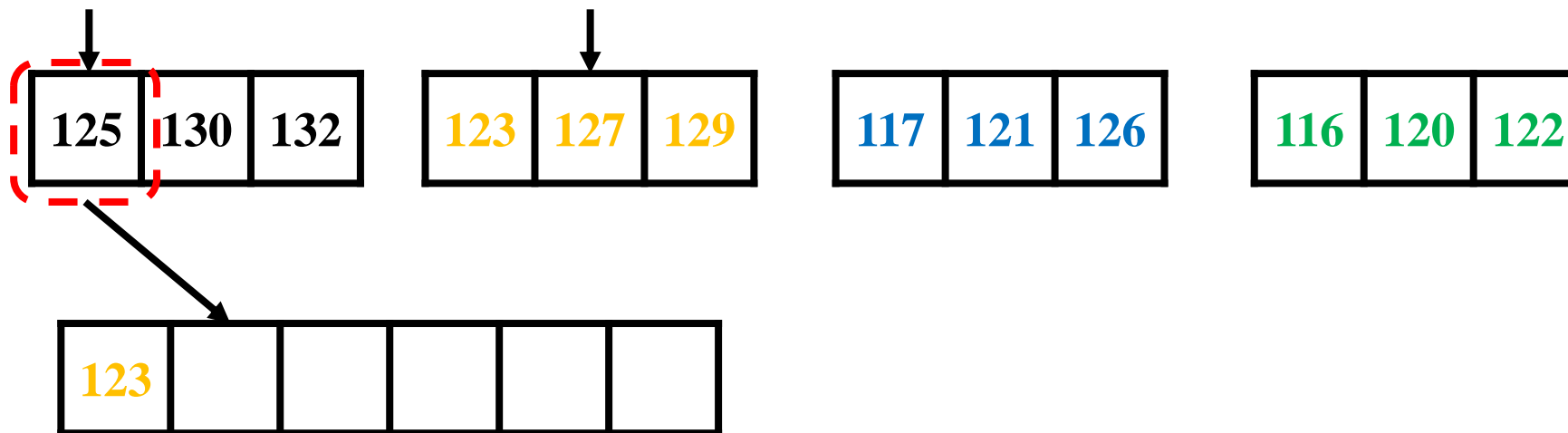
- 杠铃增重问题

- 问题特点：局部有序
- 快速合并：比较两有序数组当前最小元素，将较小者逐一合入新数组



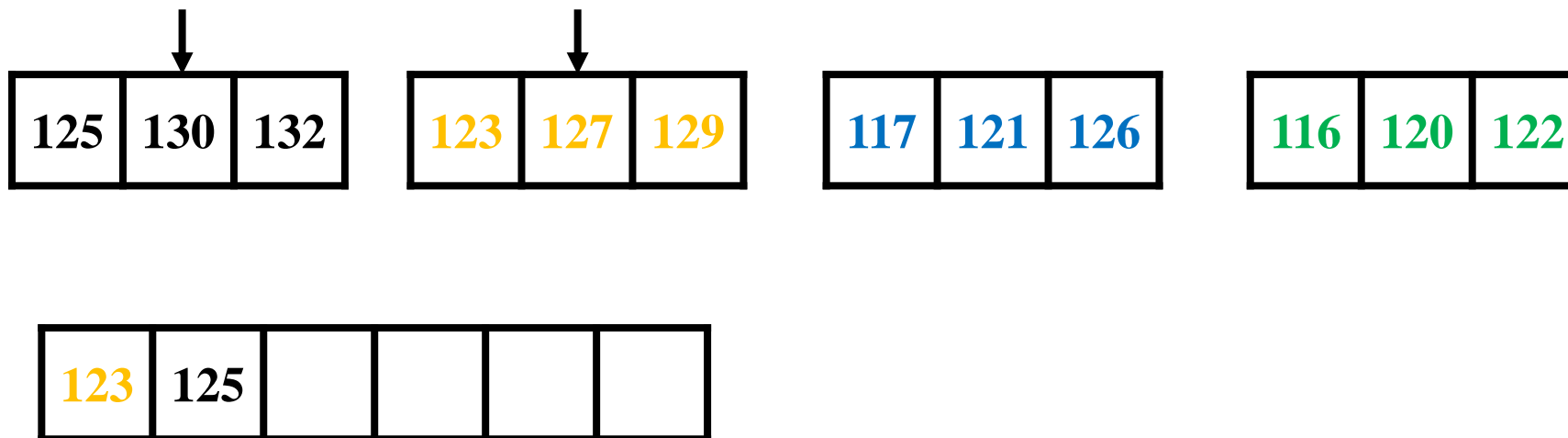
- 杠铃增重问题

- 问题特点：局部有序
- 快速合并：比较两有序数组当前最小元素，将较小者逐一合入新数组



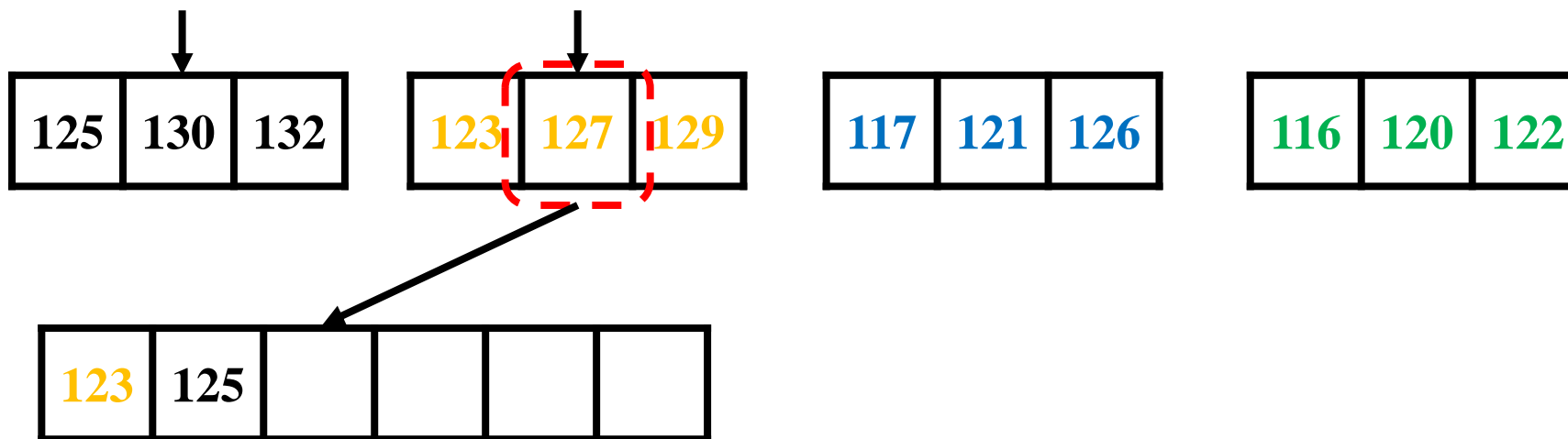
- 杠铃增重问题

- 问题特点：局部有序
- 快速合并：比较两有序数组当前最小元素，将较小者逐一合入新数组



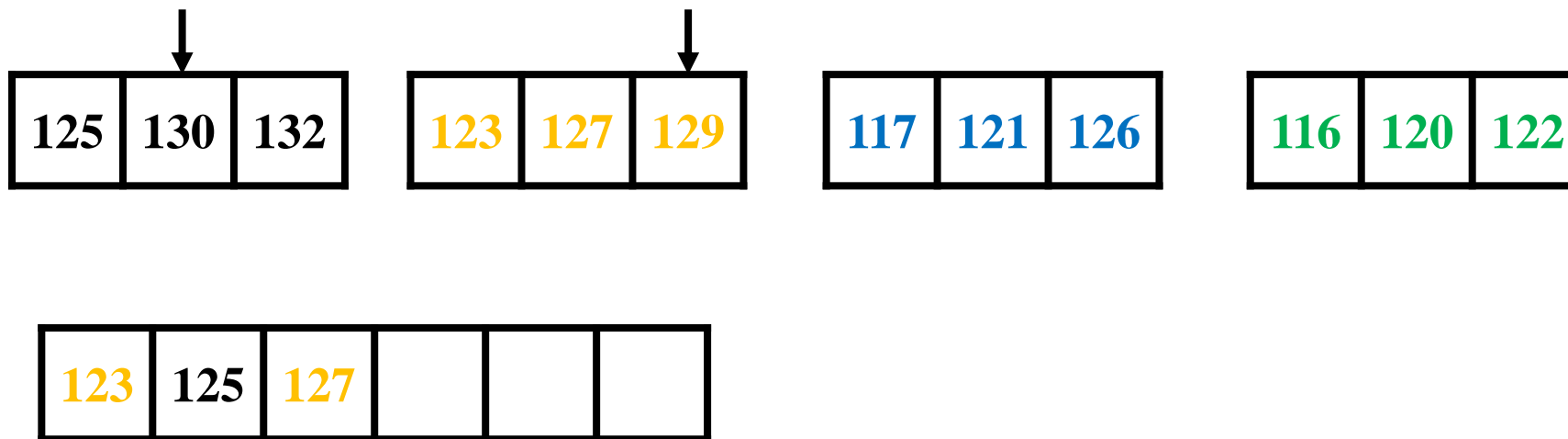
- 杠铃增重问题

- 问题特点：局部有序
- 快速合并：比较两有序数组当前最小元素，将较小者逐一合入新数组



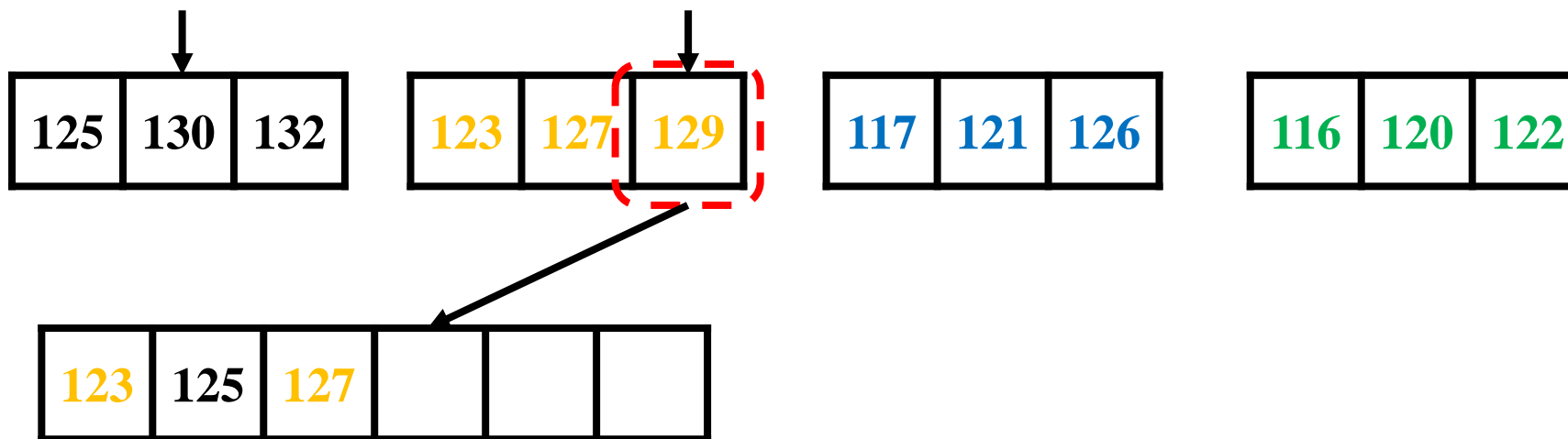
- 杠铃增重问题

- 问题特点：局部有序
- 快速合并：比较两有序数组当前最小元素，将较小者逐一合入新数组



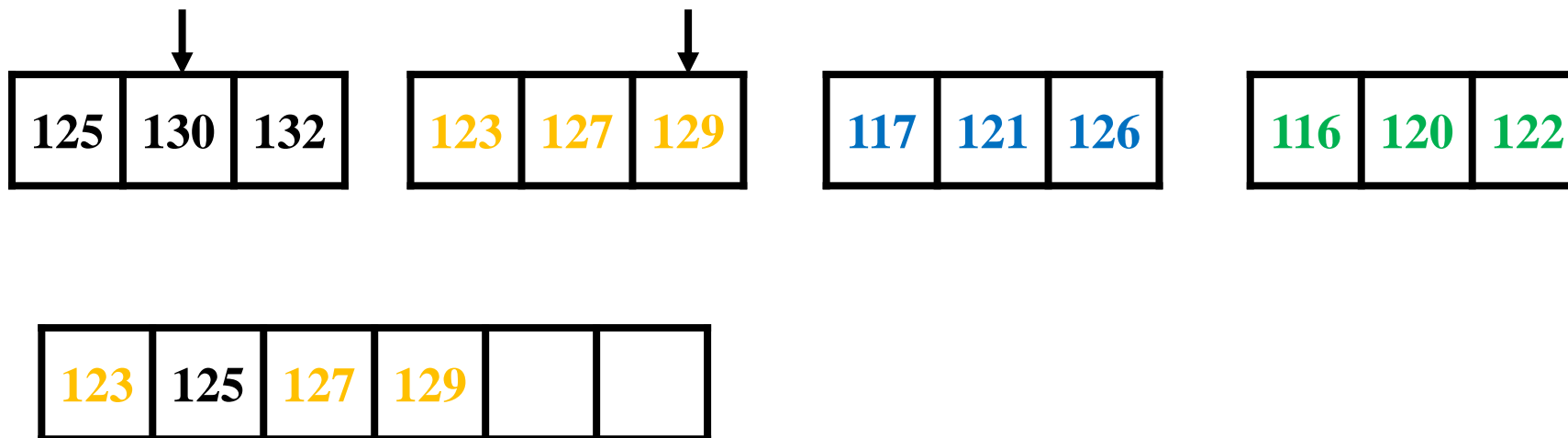
- 杠铃增重问题

- 问题特点：局部有序
- 快速合并：比较两有序数组当前最小元素，将较小者逐一合入新数组



- 杠铃增重问题

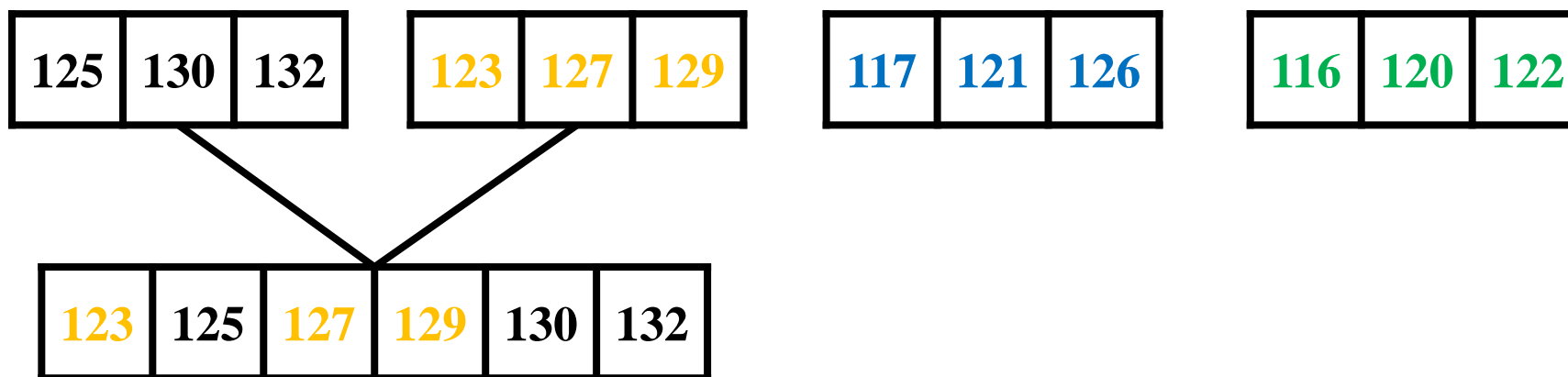
- 问题特点：局部有序
- 快速合并：比较两有序数组当前最小元素，将较小者逐一合入新数组





- 杠铃增重问题

- 问题特点：局部有序
- 快速合并：比较两有序数组当前最小元素，将较小者逐一合入新数组



- 杠铃增重问题
  - 问题特点：局部有序
  - 快速合并：比较两有序数组当前最小元素，将较小者逐一合入新数组
  - 后续策略：
    - 逐一合并

- 杠铃增重问题

- 问题特点：局部有序
- 快速合并：比较两有序数组当前最小元素，将较小者逐一合入新数组
- 后续策略：
  - 逐一合并

125	130	132
-----	-----	-----

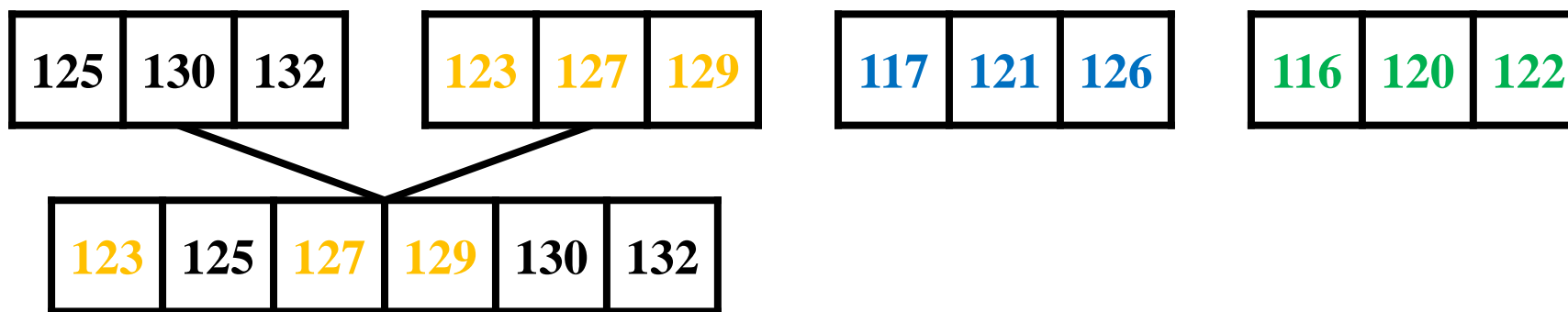
123	127	129
-----	-----	-----

117	121	126
-----	-----	-----

116	120	122
-----	-----	-----

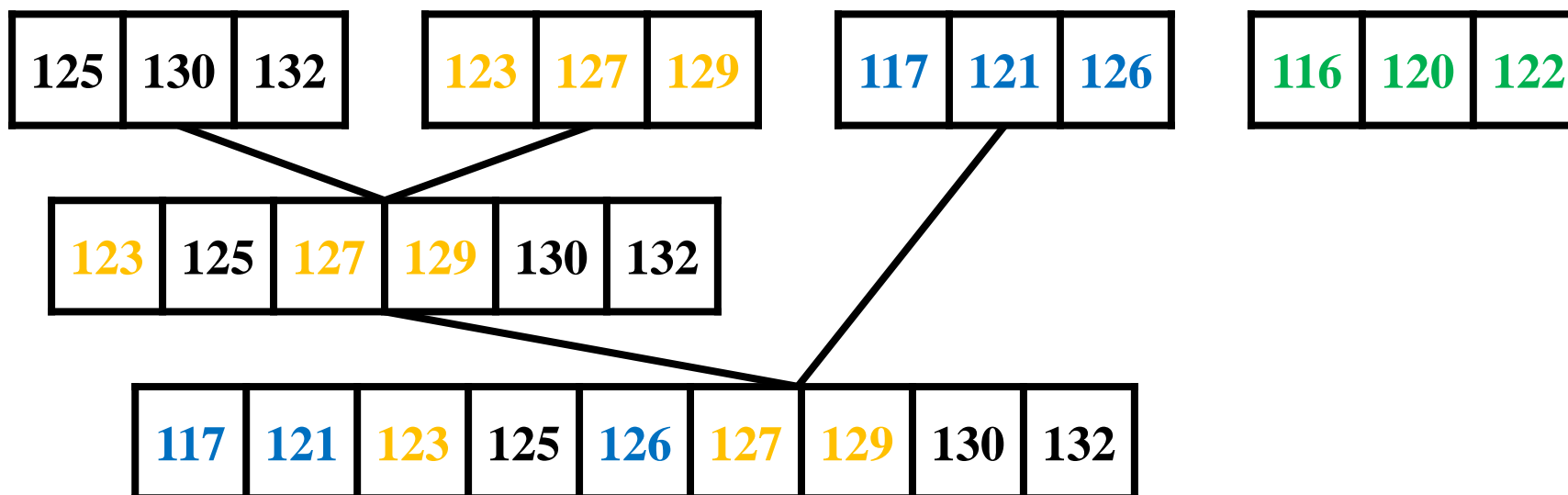
- 杠铃增重问题

- 问题特点：局部有序
- 快速合并：比较两有序数组当前最小元素，将较小者逐一合入新数组
- 后续策略：
  - 逐一合并



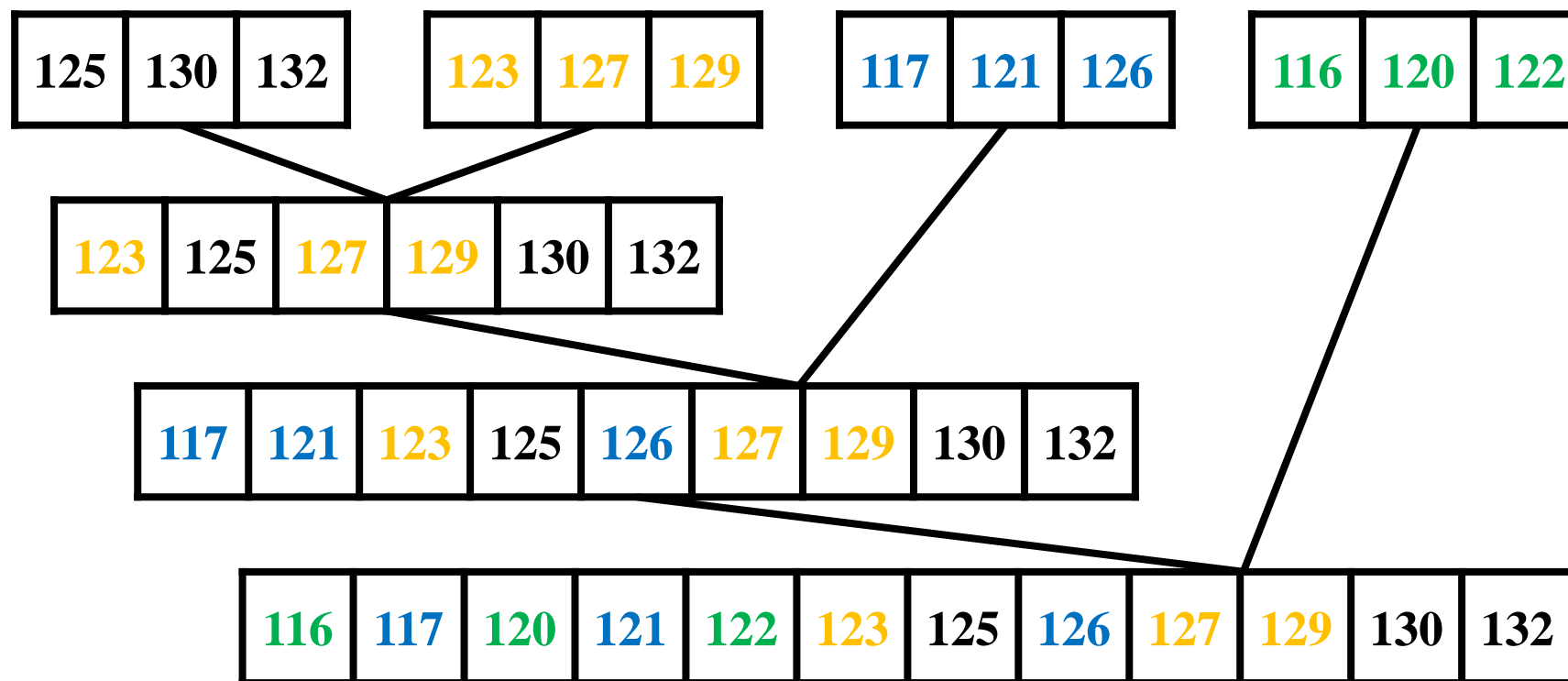
- 杠铃增重问题

- 问题特点：局部有序
- 快速合并：比较两有序数组当前最小元素，将较小者逐一合入新数组
- 后续策略：
  - 逐一合并



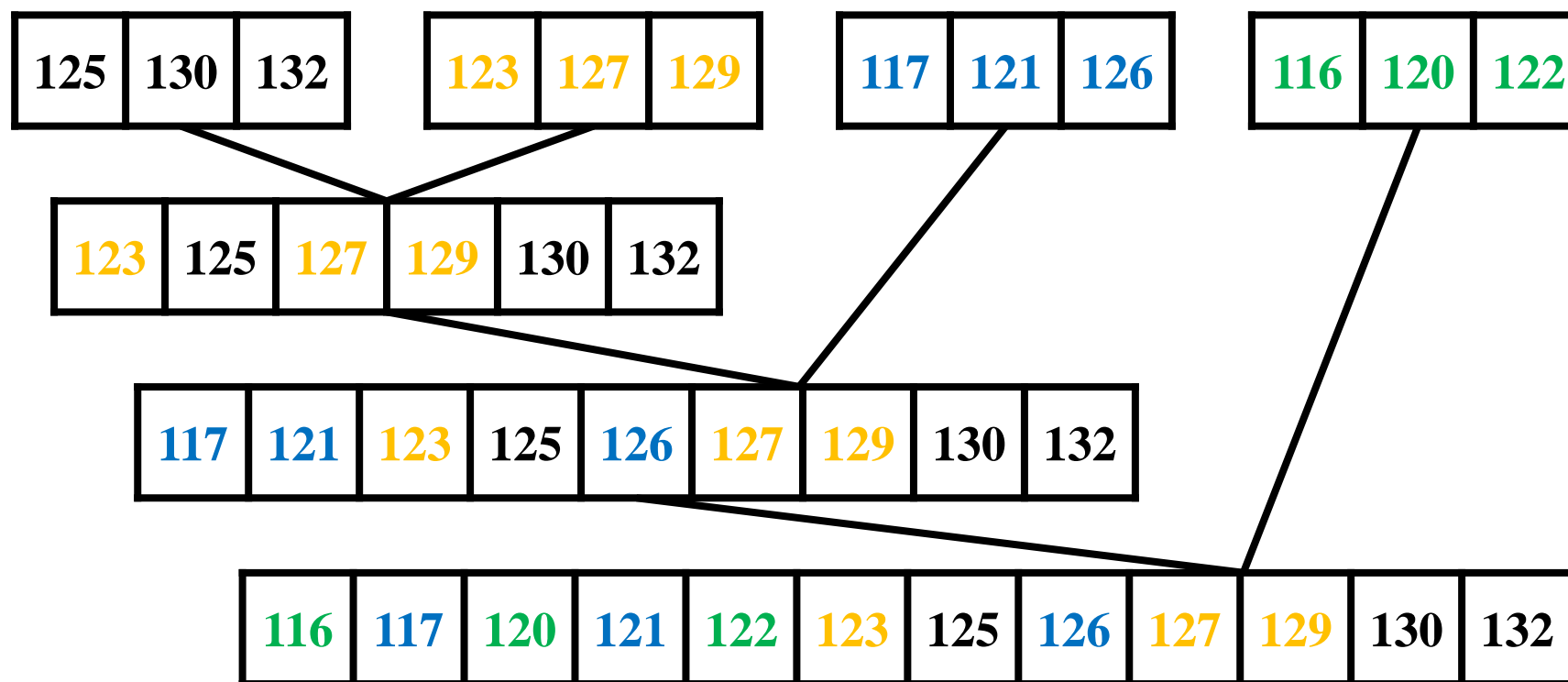
- 杠铃增重问题

- 问题特点：局部有序
- 快速合并：比较两有序数组当前最小元素，将较小者逐一合入新数组
- 后续策略：
  - 逐一合并



- 杠铃增重问题

- 问题特点：局部有序
- 快速合并：比较两有序数组当前最小元素，将较小者逐一合入新数组
- 后续策略：
  - 逐一合并，比较27次



- 杠铃增重问题

- 问题特点：局部有序
- 快速合并：比较两有序数组当前最小元素，将较小者逐一合入新数组
- 后续策略：
  - 逐一合并，比较27次
  - 两两合并

125	130	132
-----	-----	-----

123	127	129
-----	-----	-----

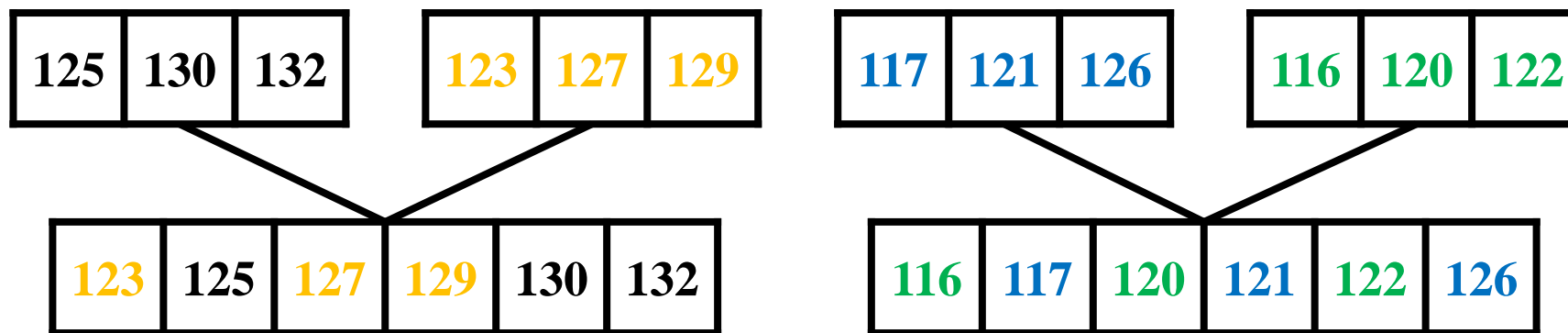
117	121	126
-----	-----	-----

116	120	122
-----	-----	-----



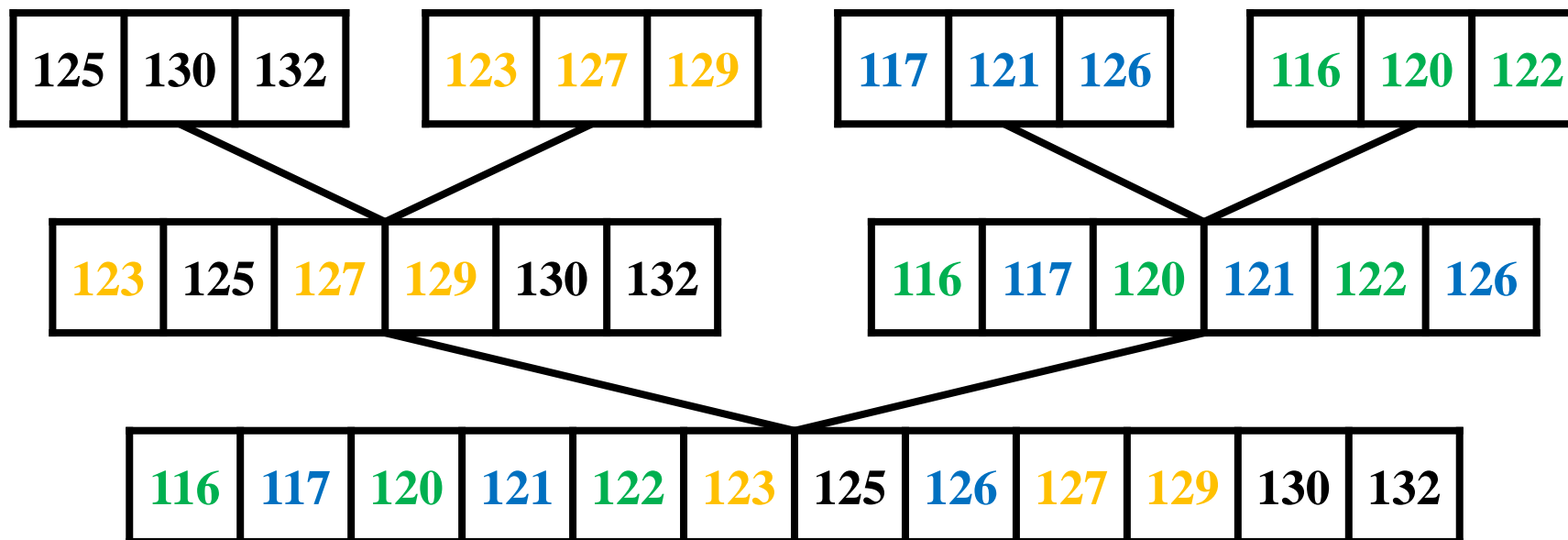
- 杠铃增重问题

- 问题特点：局部有序
- 快速合并：比较两有序数组当前最小元素，将较小者逐一合入新数组
- 后续策略：
  - 逐一合并，比较27次
  - 两两合并



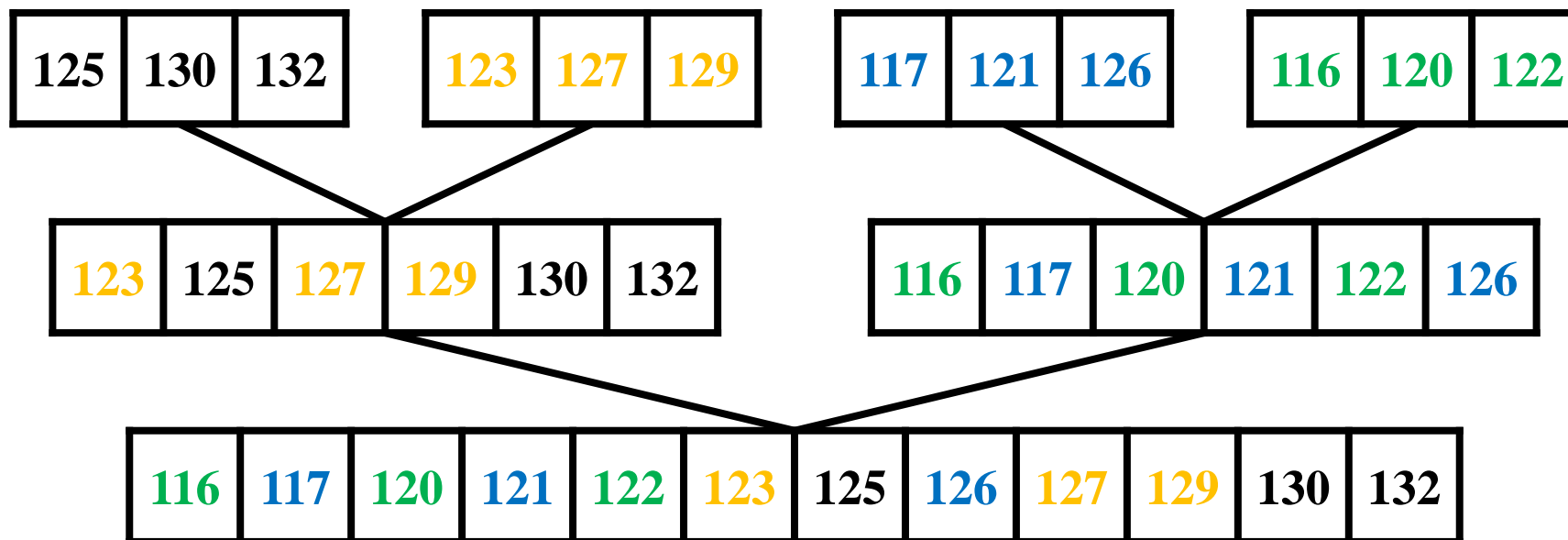
- 杠铃增重问题

- 问题特点：局部有序
- 快速合并：比较两有序数组当前最小元素，将较小者逐一合入新数组
- 后续策略：
  - 逐一合并，比较27次
  - 两两合并



- 杠铃增重问题

- 问题特点：局部有序
- 快速合并：比较两有序数组当前最小元素，将较小者逐一合入新数组
- 后续策略：
  - 逐一合并，比较27次
  - 两两合并，比较24次



- 杠铃增重问题

- 问题特点：局部有序
- 快速合并：比较两有序数组当前最小元素，将较小者逐一合入新数组
- 后续策略：
  - 逐一合并，比较27次
  - 两两合并，比较24次

策略名称	4位选手	8位选手	16位选手
逐一合并	27次	105次	405次
两两合并	24次	72次	192次

- 杠铃增重问题

- 问题特点：局部有序
- 快速合并：比较两有序数组当前最小元素，将较小者逐一合入新数组
- 后续策略：
  - 逐一合并，比较27次
  - 两两合并，比较24次

策略名称	4位选手	8位选手	16位选手
逐一合并	27次	105次	405次
两两合并	24次	72次	192次

求解杠铃增重问题的两两合并策略对排序问题有何启发？

# 从杠铃增重问题到排序问题



## • 排序问题回顾

### 排序问题

#### Sorting Problem

输入

- 包含 $n$ 个数字的序列  $\langle a_1, \dots, a_n \rangle$

输出

- 输入序列的升序

$$\langle a'_1, a'_2, \dots, a'_n \rangle$$
$$\text{满足 } a'_1 \leq a'_2 \leq \dots \leq a'_n$$

示例

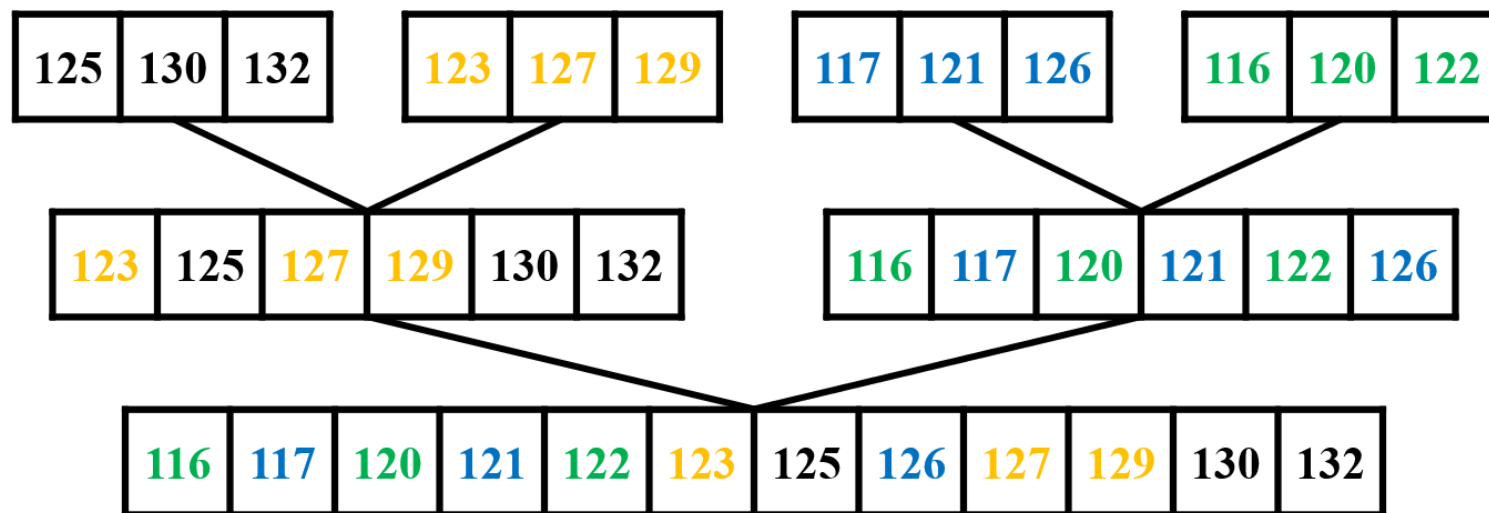
24	17	40	28	13	14	22	32	40	21	48	4	47	8	37	18
----	----	----	----	----	----	----	----	----	----	----	---	----	---	----	----

# 从杠铃增重问题到排序问题



- 问题输入变化

杠铃增重问题



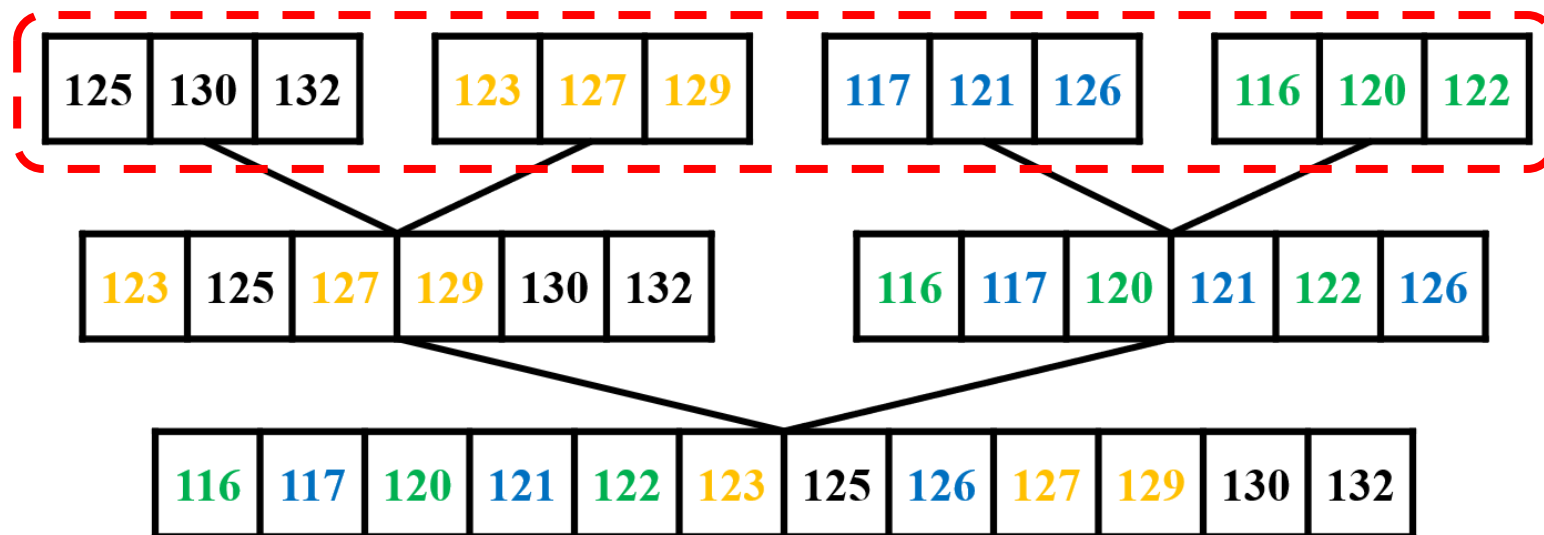
排序问题

24	17	40	28	13	14	22	32	40	21	48	4	47	8	37	18
----	----	----	----	----	----	----	----	----	----	----	---	----	---	----	----

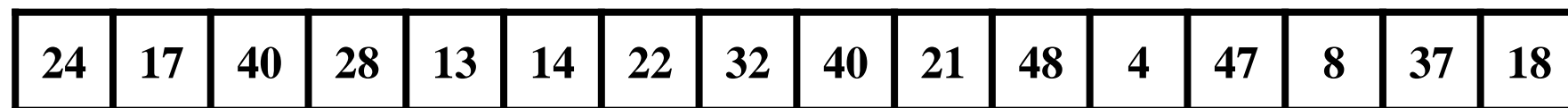
# 从杠铃增重问题到排序问题

- 问题输入变化
  - 完整数组输入

## 杠铃增重问题



## 排序问题





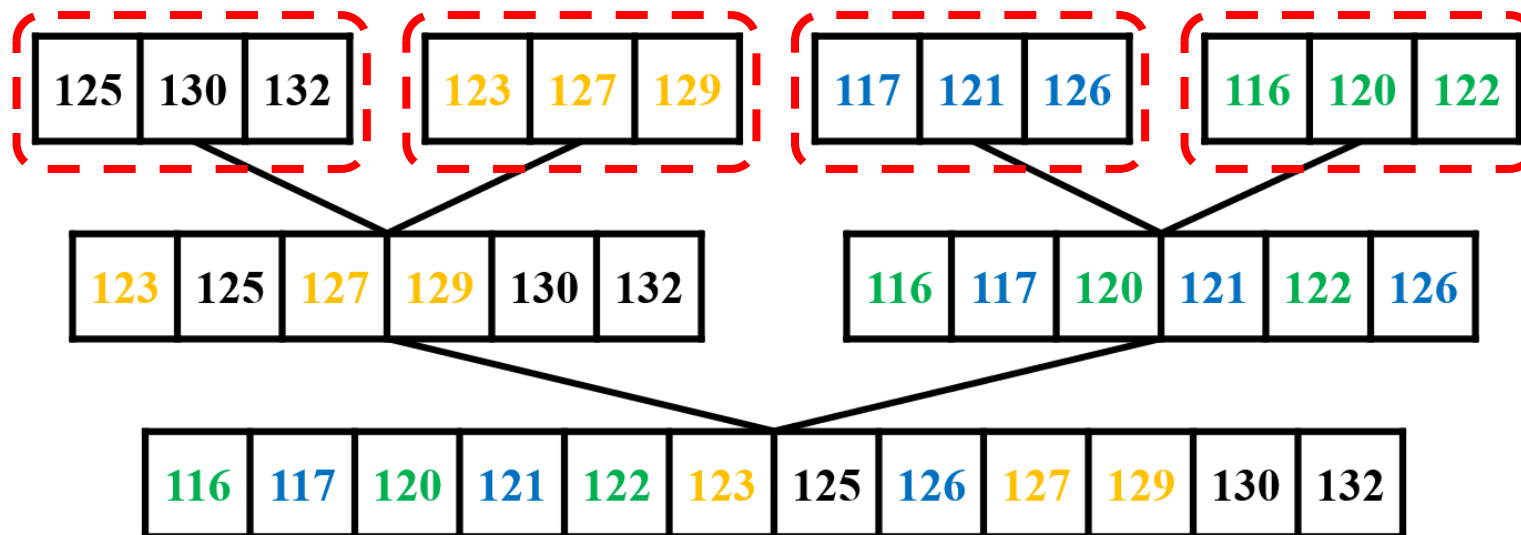
# 从杠铃增重问题到排序问题



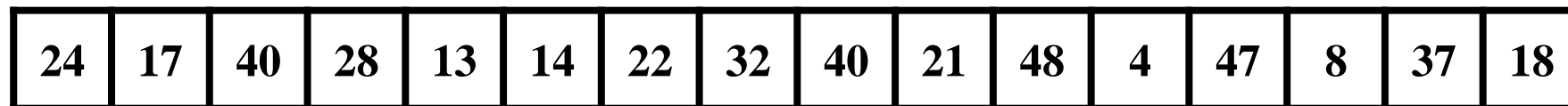
- 问题输入变化

- 完整数组输入
- 局部有序缺失

杠铃增重问题



排序问题



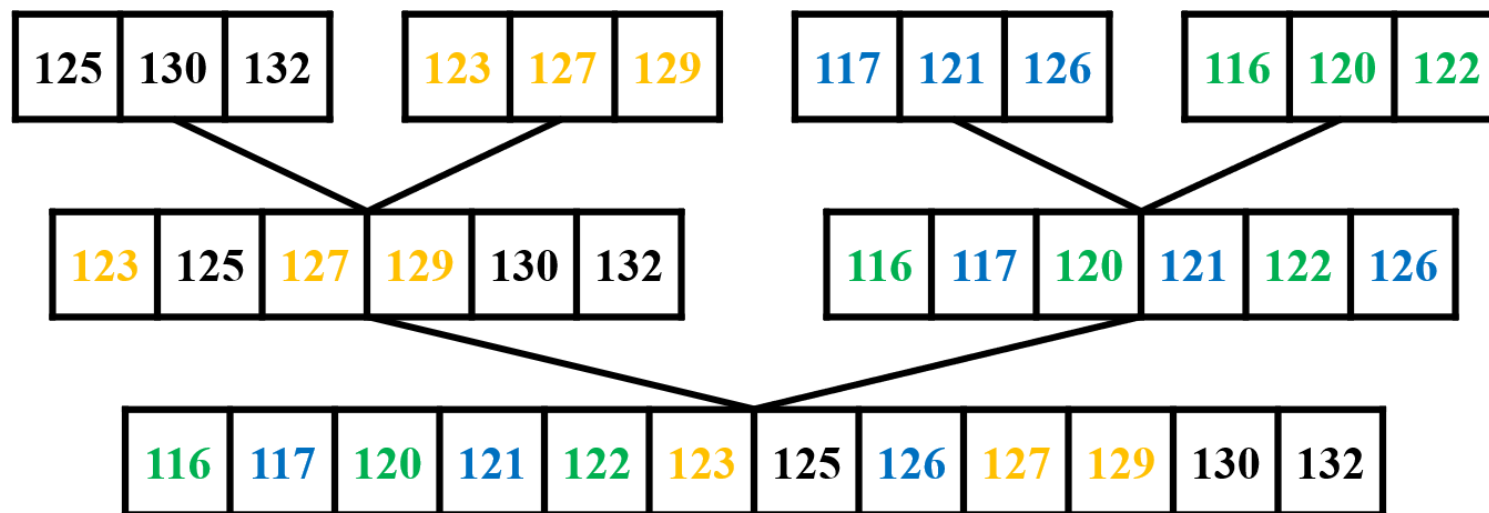
# 从杠铃增重问题到排序问题

## ● 问题输入变化

- 完整数组输入
- 局部有序缺失

两两合并策略如何适应问题输入的变化？

杠铃增重问题



排序问题

24	17	40	28	13	14	22	32	40	21	48	4	47	8	37	18
----	----	----	----	----	----	----	----	----	----	----	---	----	---	----	----

# 从杠铃增重问题到排序问题



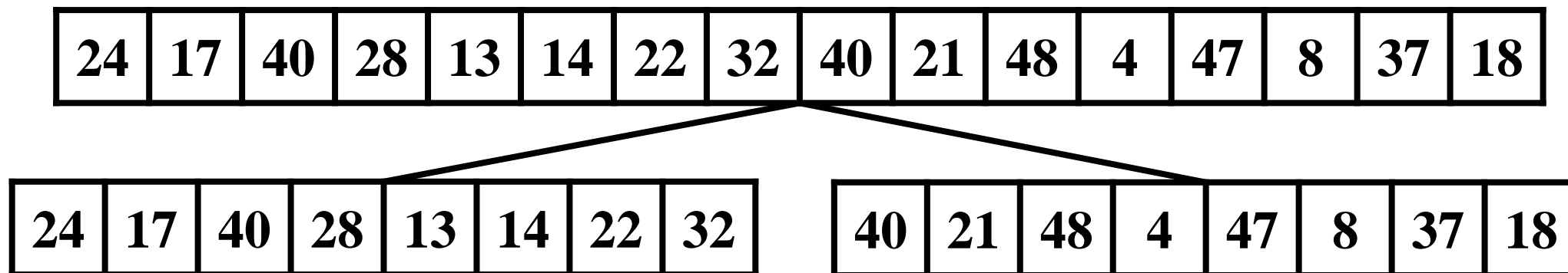
- 解决输入变化：分解输入

24	17	40	28	13	14	22	32	40	21	48	4	47	8	37	18
----	----	----	----	----	----	----	----	----	----	----	---	----	---	----	----

# 从杠铃增重问题到排序问题



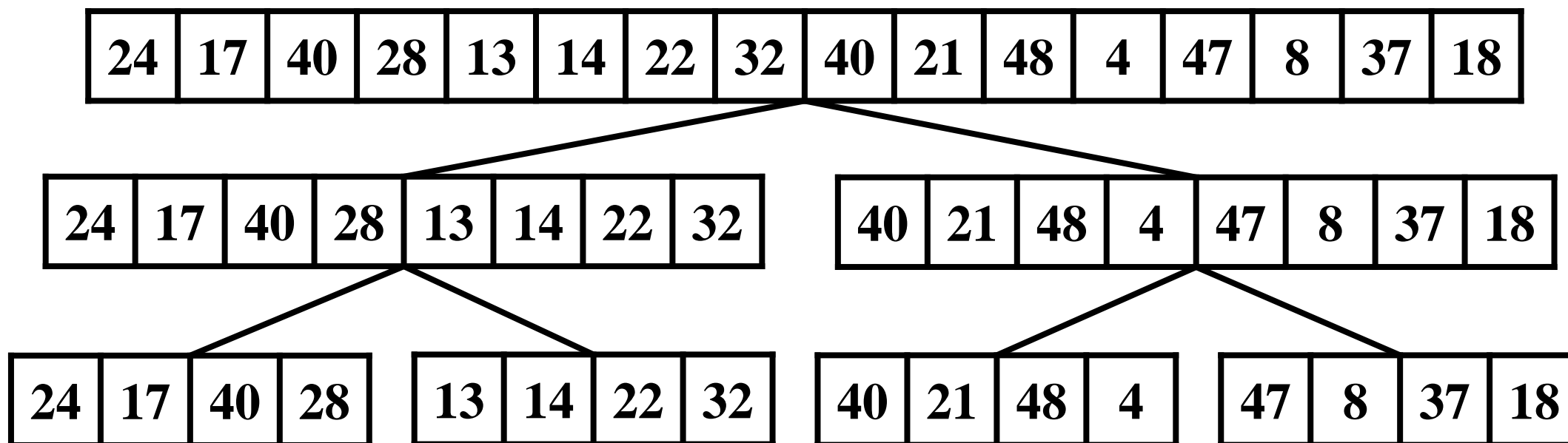
- 解决输入变化：分解输入



# 从杠铃增重问题到排序问题



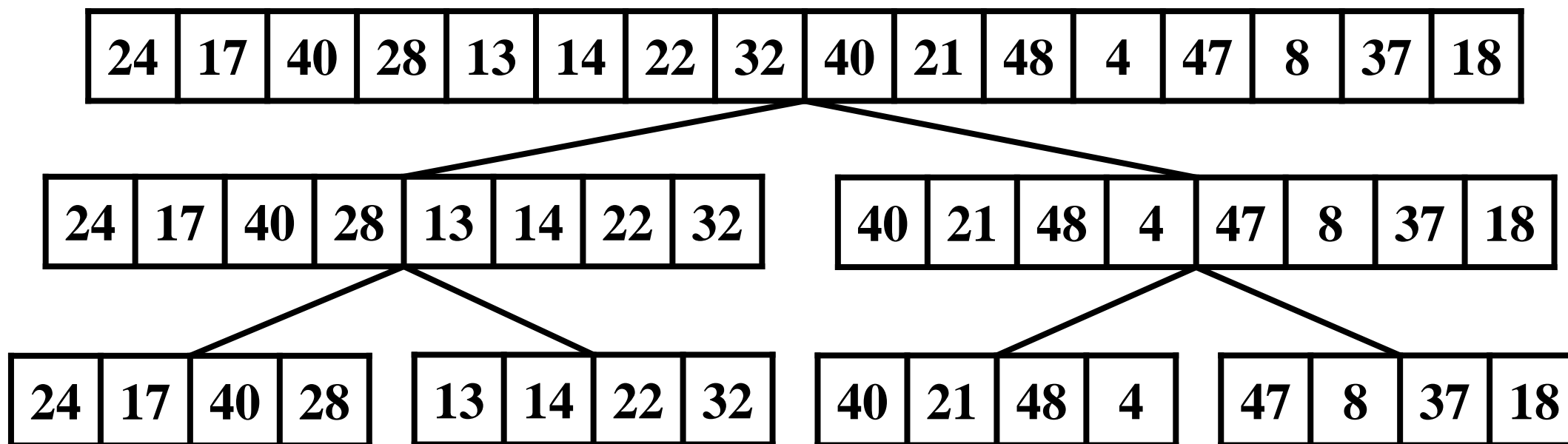
- 解决输入变化：分解输入



# 从杠铃增重问题到排序问题



- 解决输入变化：分解输入

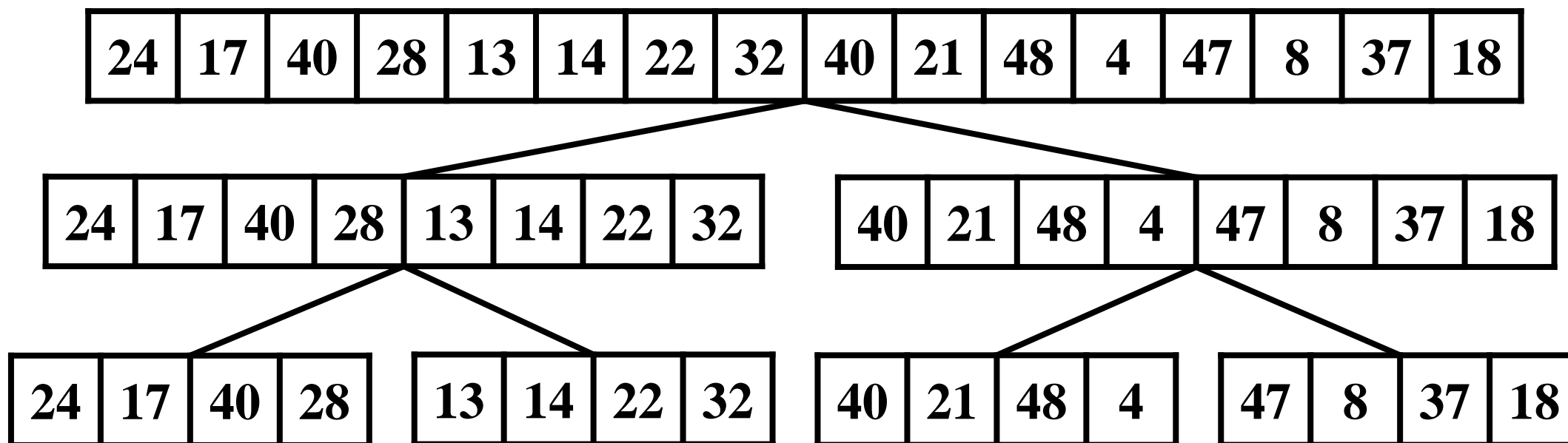


通过分解处理完整数组输入的变化

# 从杠铃增重问题到排序问题



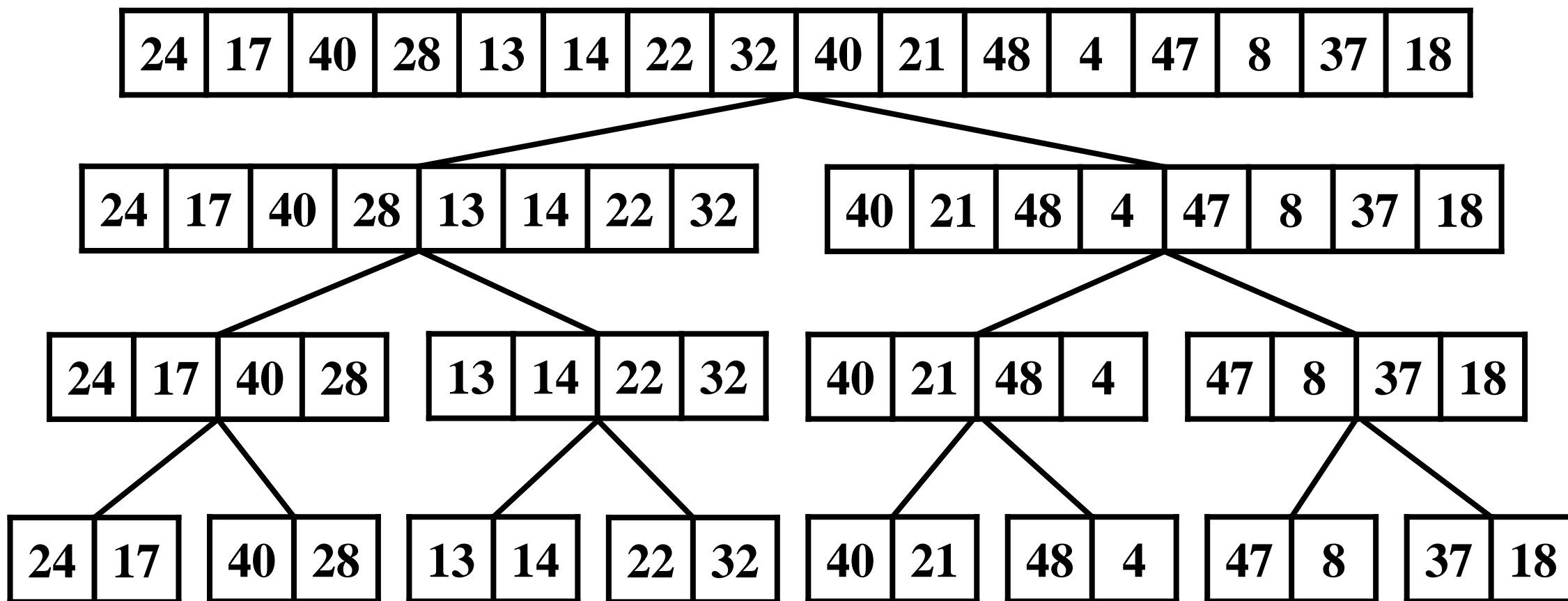
- 解决输入变化：分解输入



# 从杠铃增重问题到排序问题



- 解决输入变化：分解输入

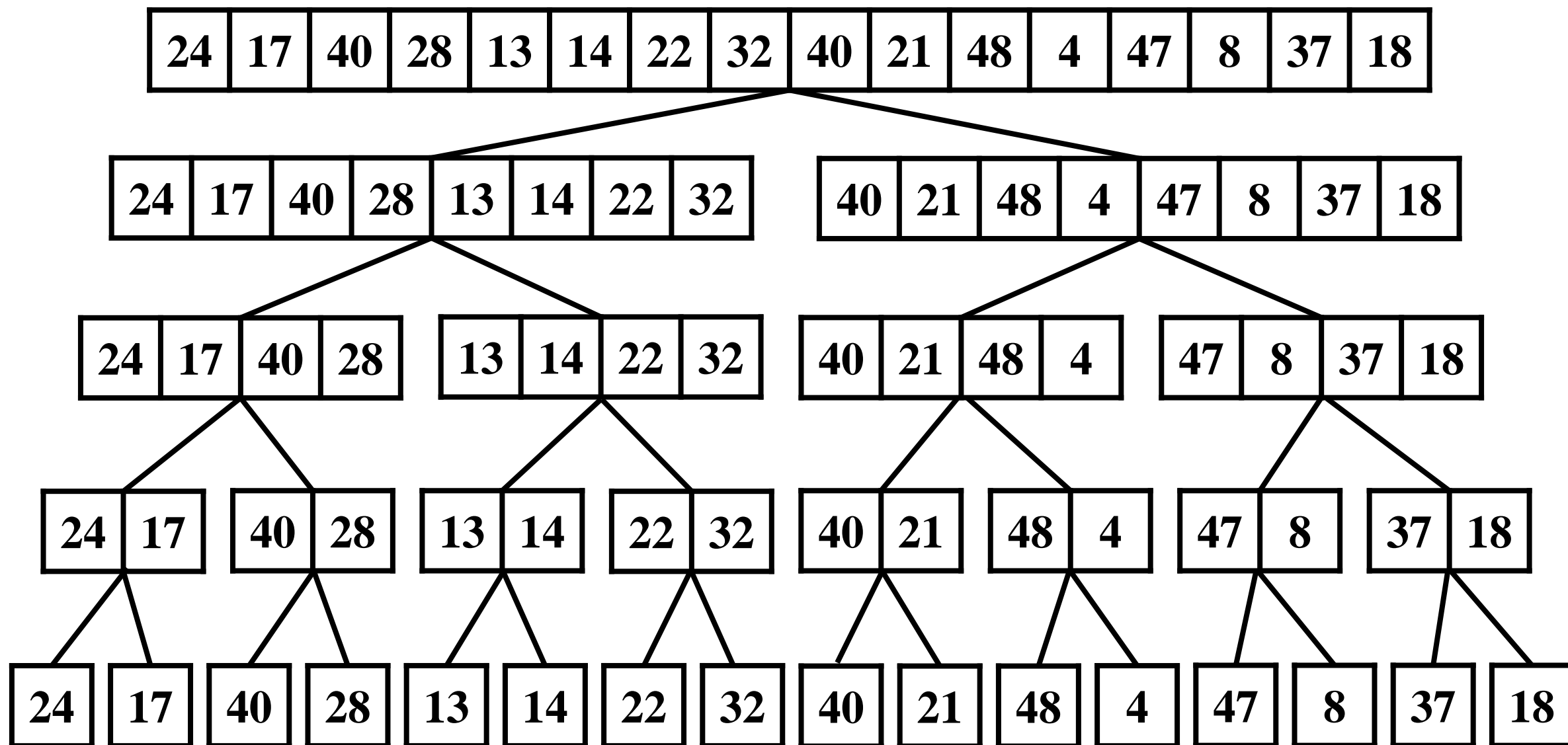




# 从杠铃增重问题到排序问题



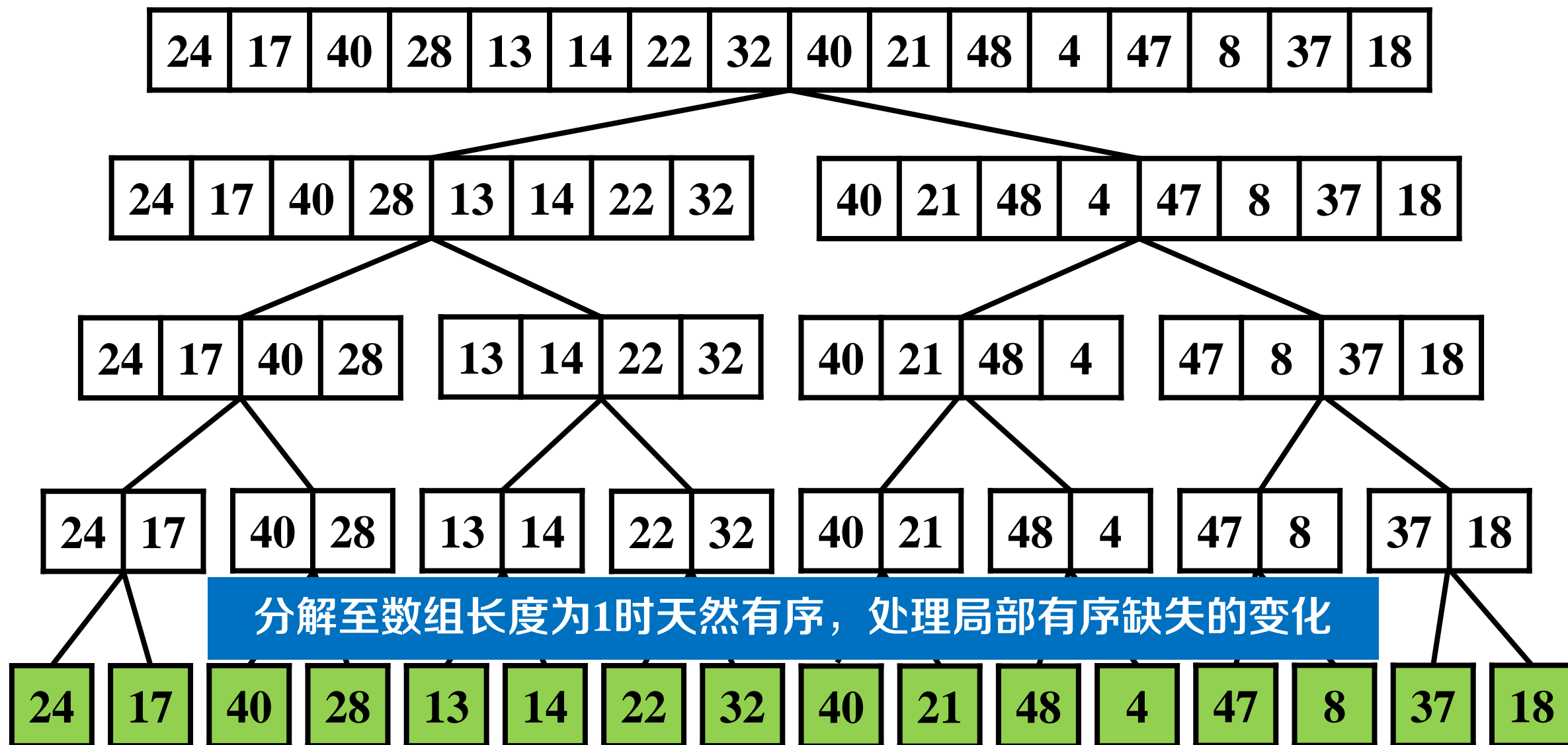
- 解决输入变化：分解输入



# 从杠铃增重问题到排序问题



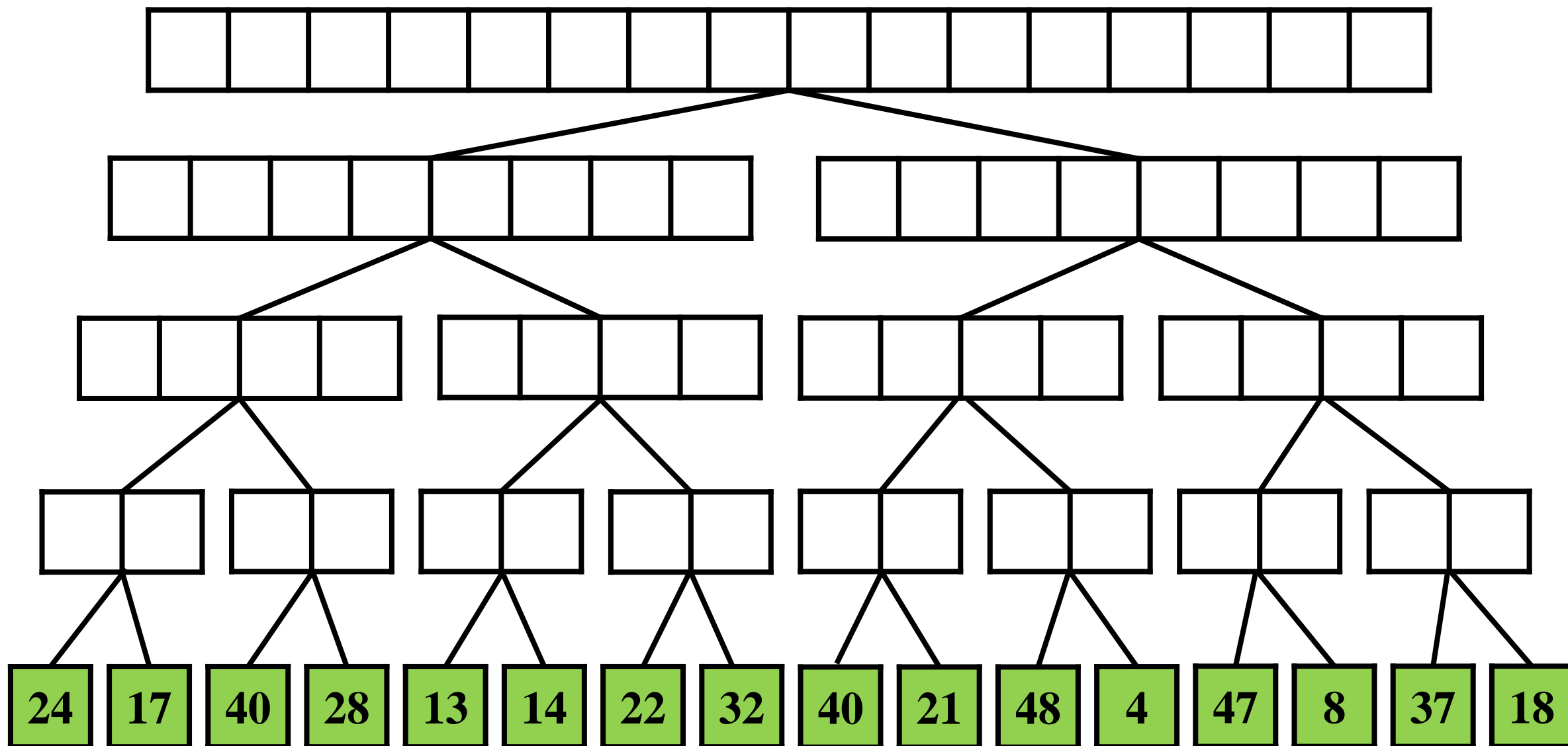
- 解决输入变化：分解输入



# 从杠铃增重问题到排序问题



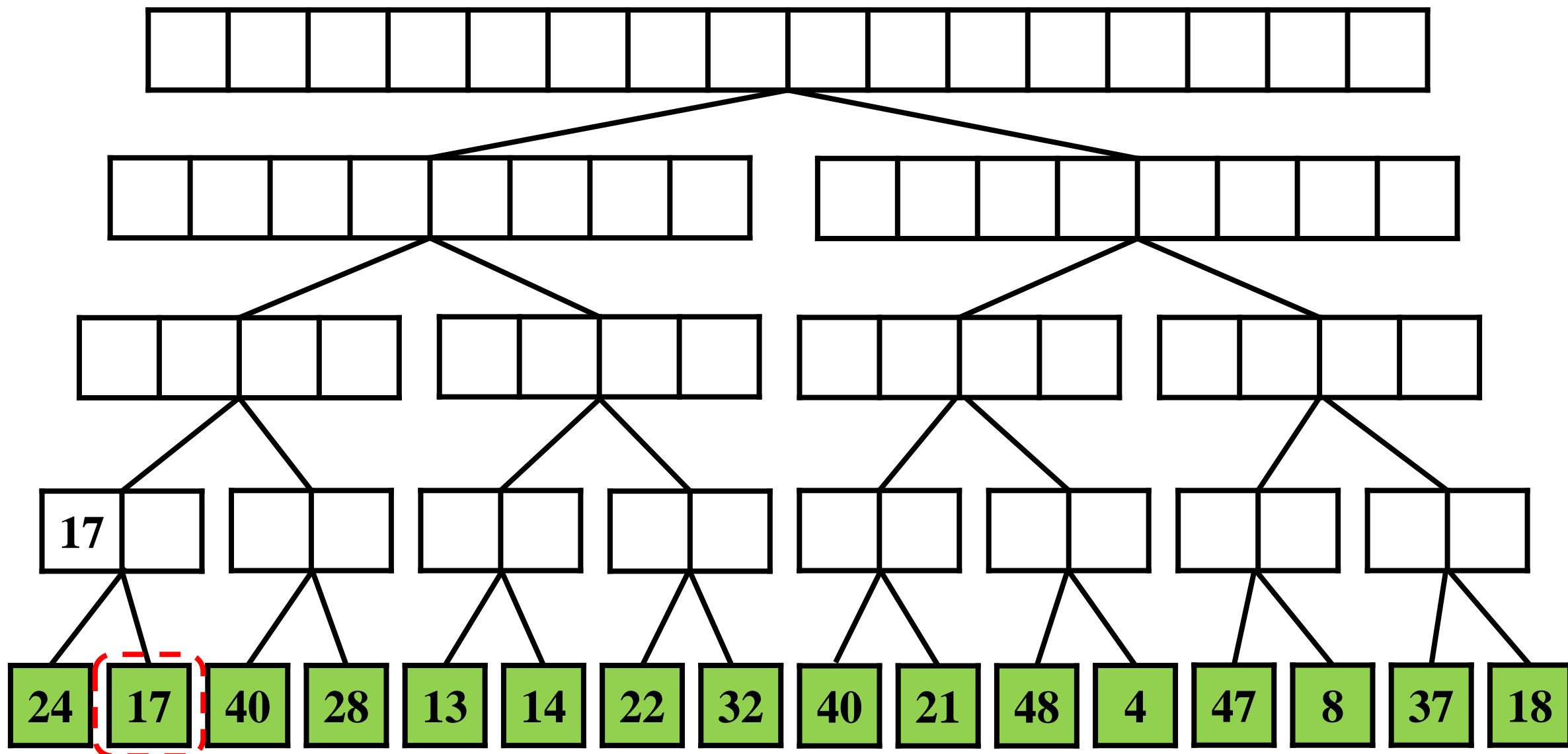
- 构建有序数组：两两合并



# 从杠铃增重问题到排序问题



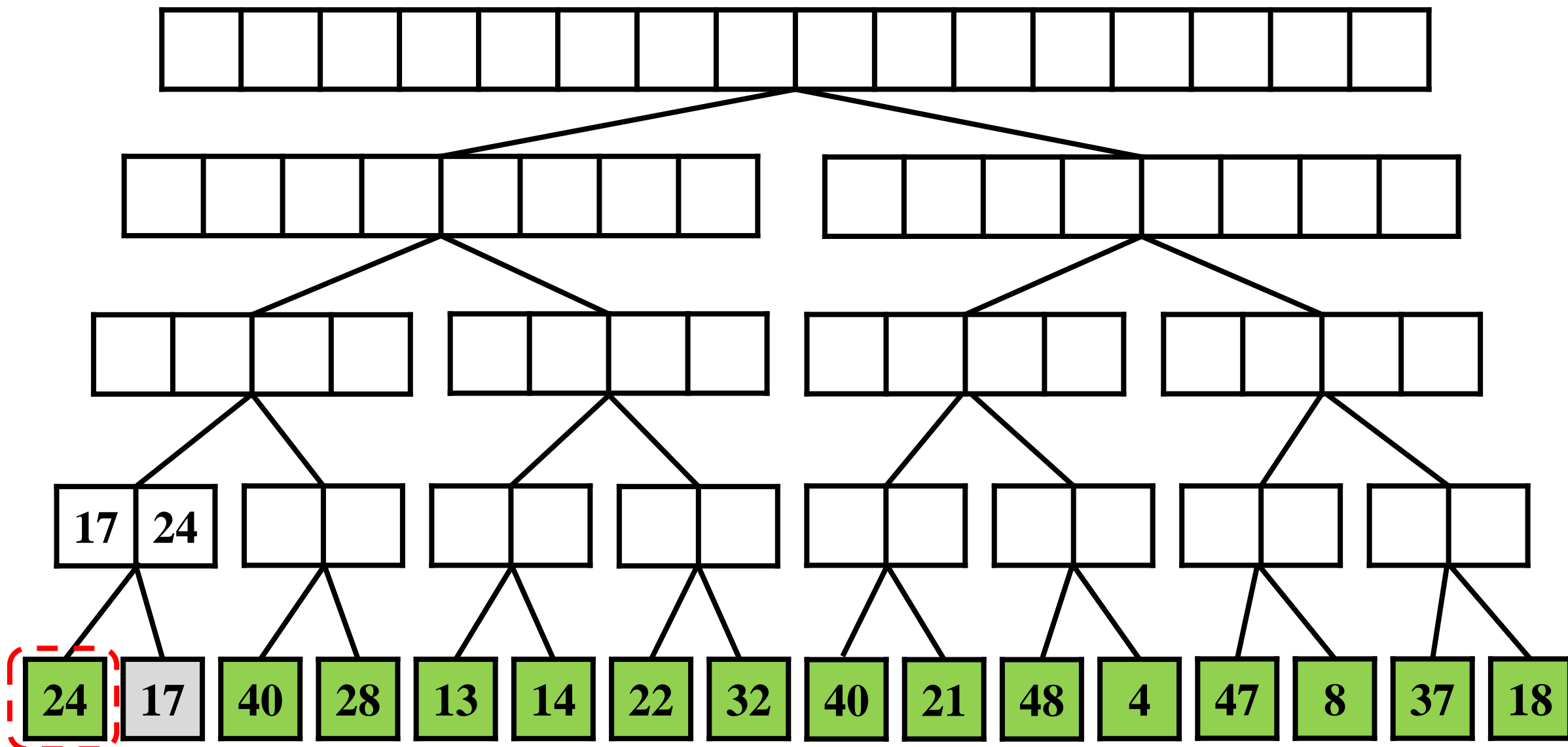
- 构建有序数组：两两合并



# 从杠铃增重问题到排序问题



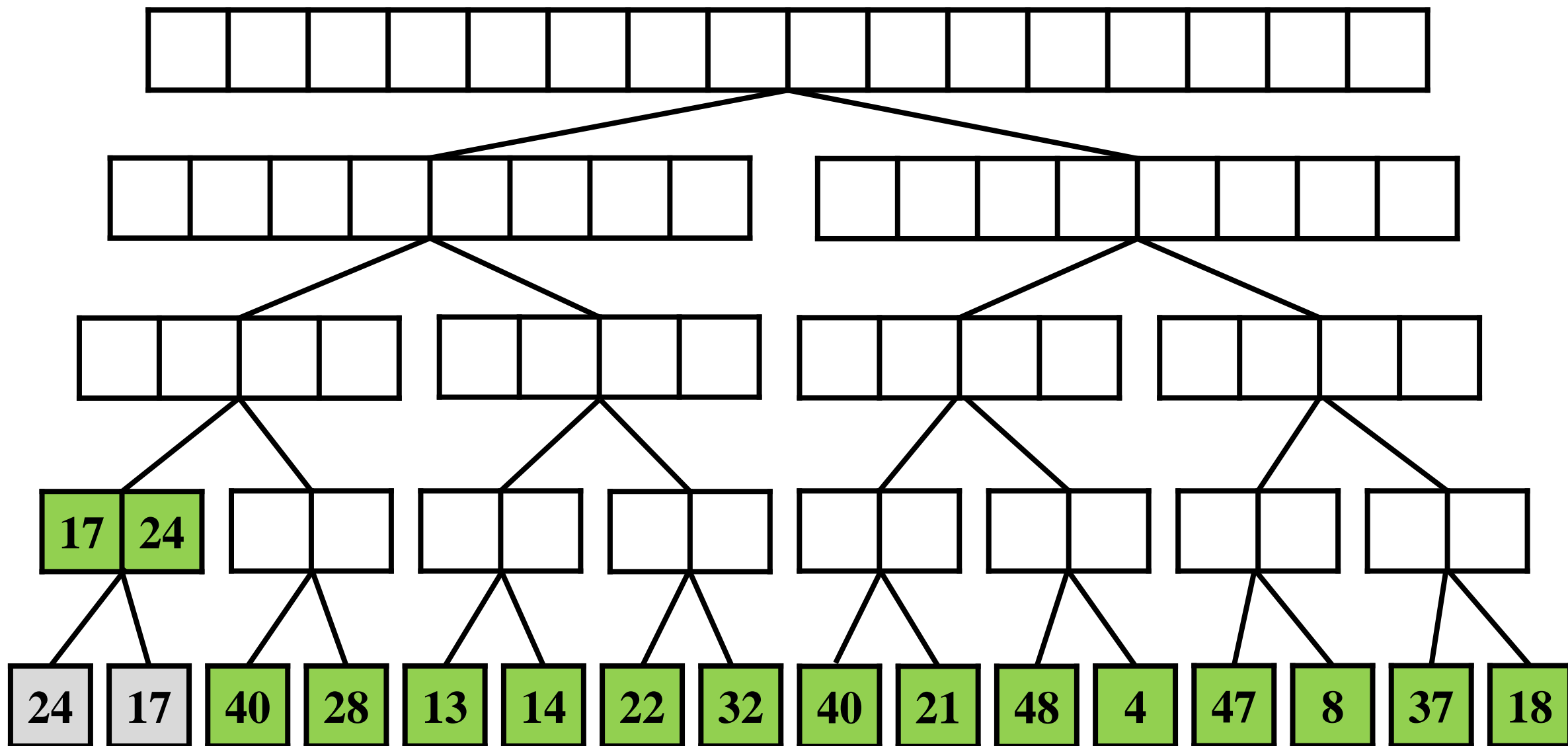
- 构建有序数组：两两合并



# 从杠铃增重问题到排序问题



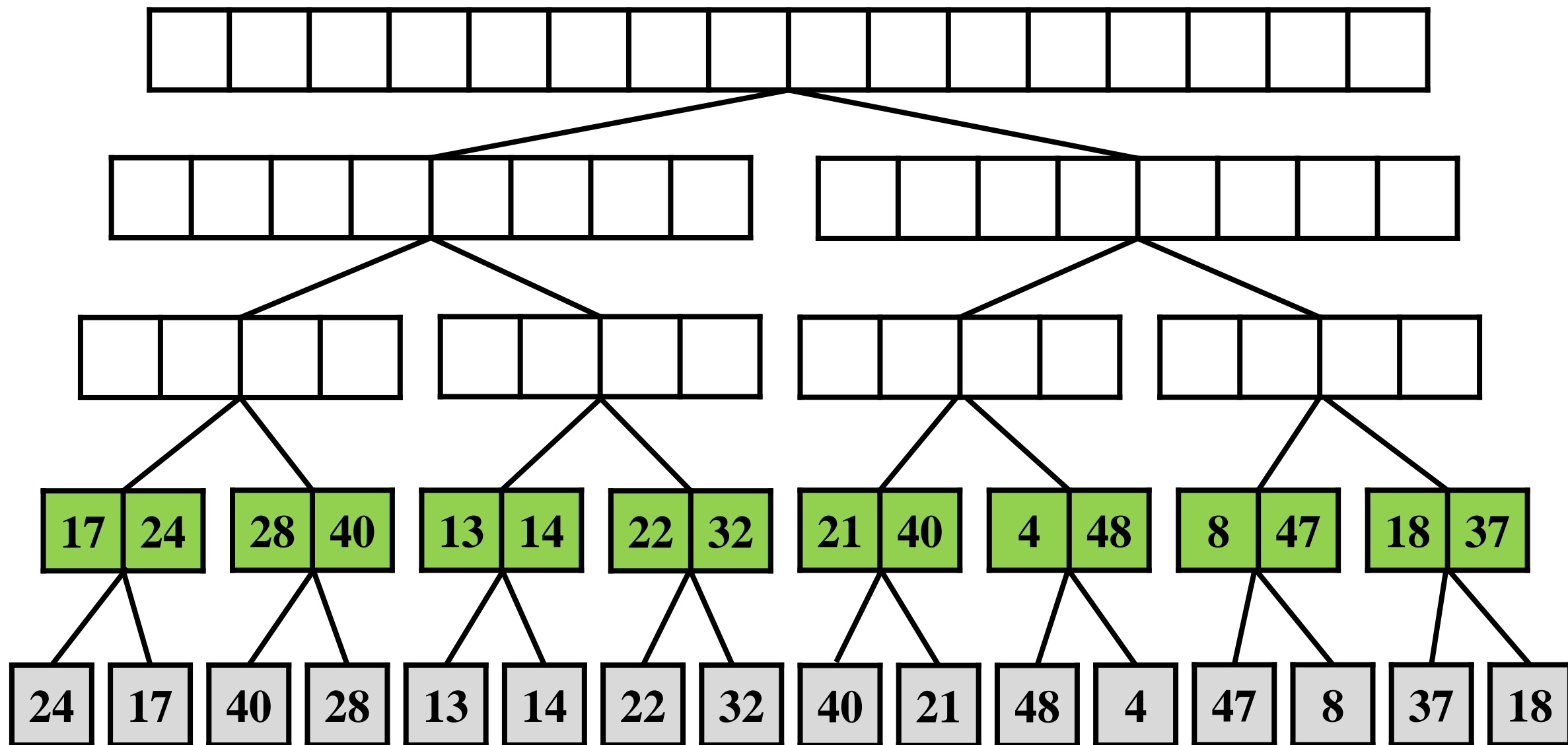
- 构建有序数组：两两合并



# 从杠铃增重问题到排序问题



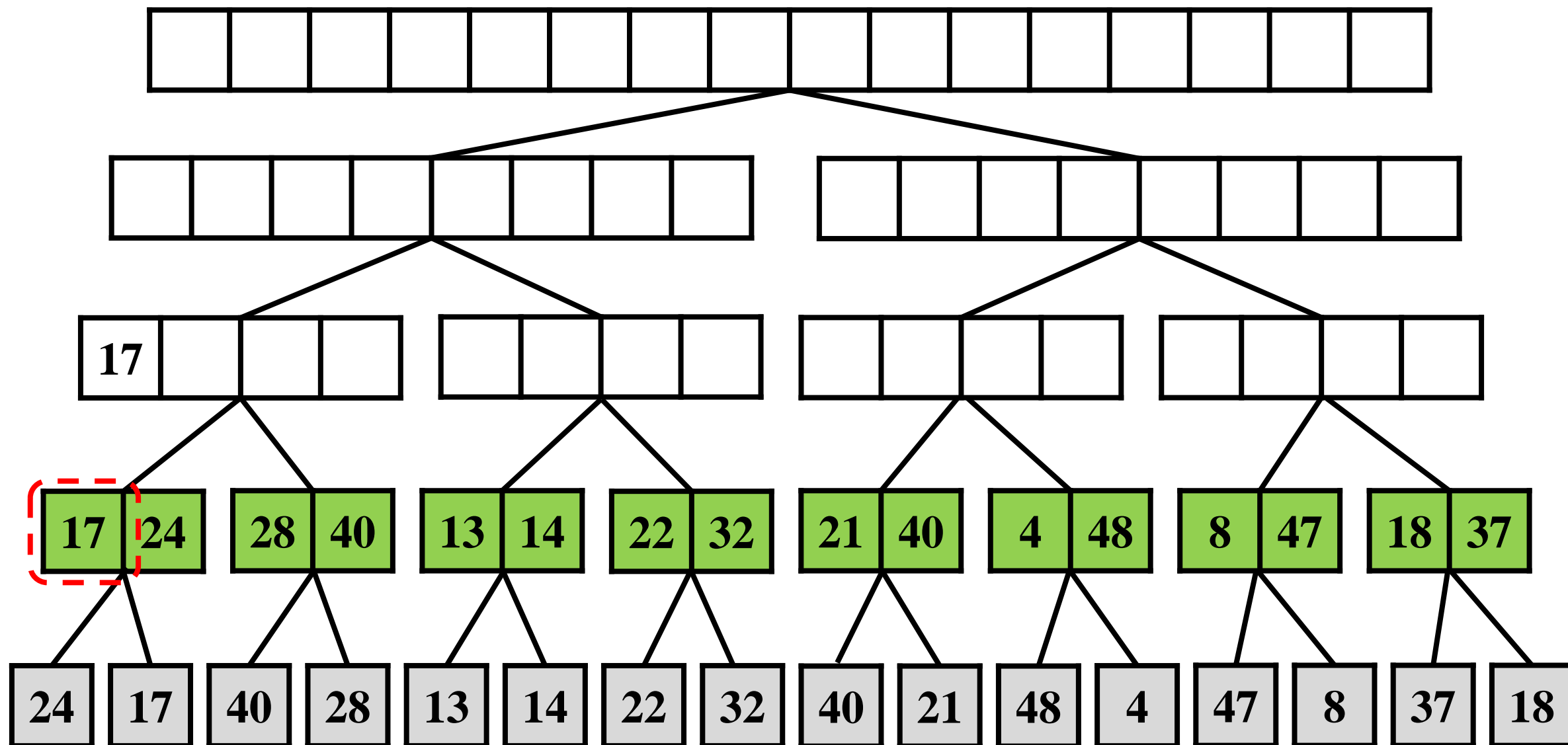
- 构建有序数组：两两合并



# 从杠铃增重问题到排序问题



- 构建有序数组：两两合并

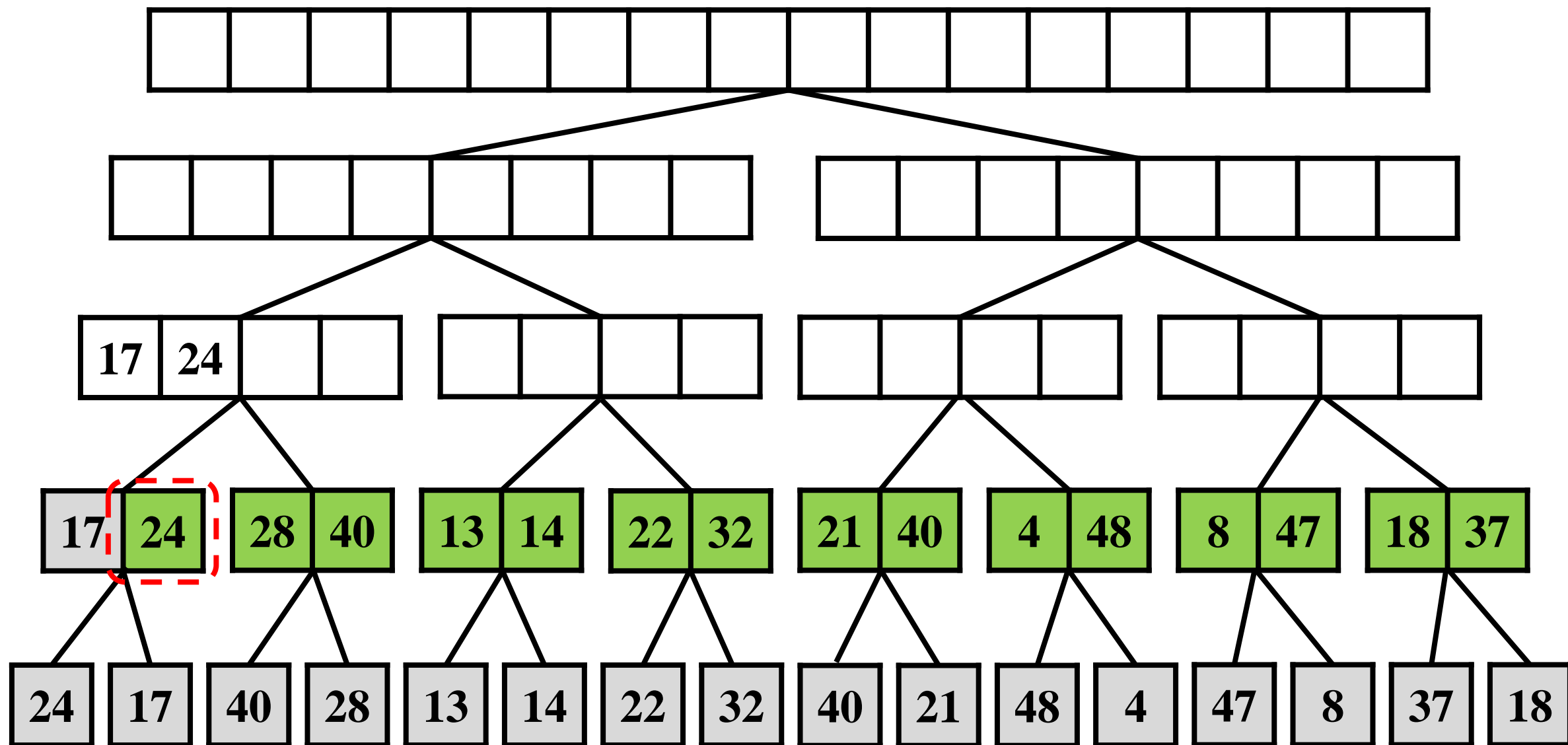




# 从杠铃增重问题到排序问题



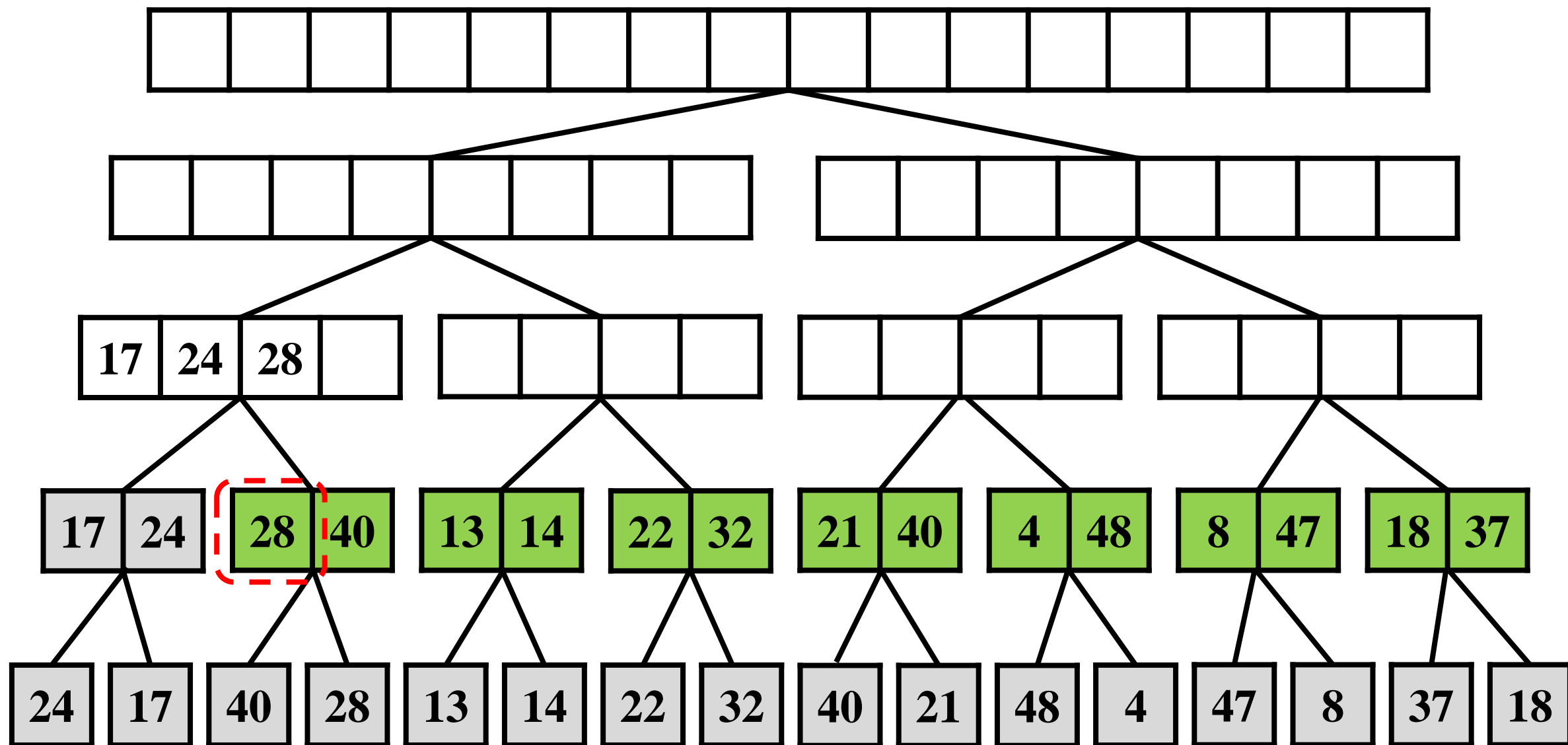
- 构建有序数组：两两合并



# 从杠铃增重问题到排序问题



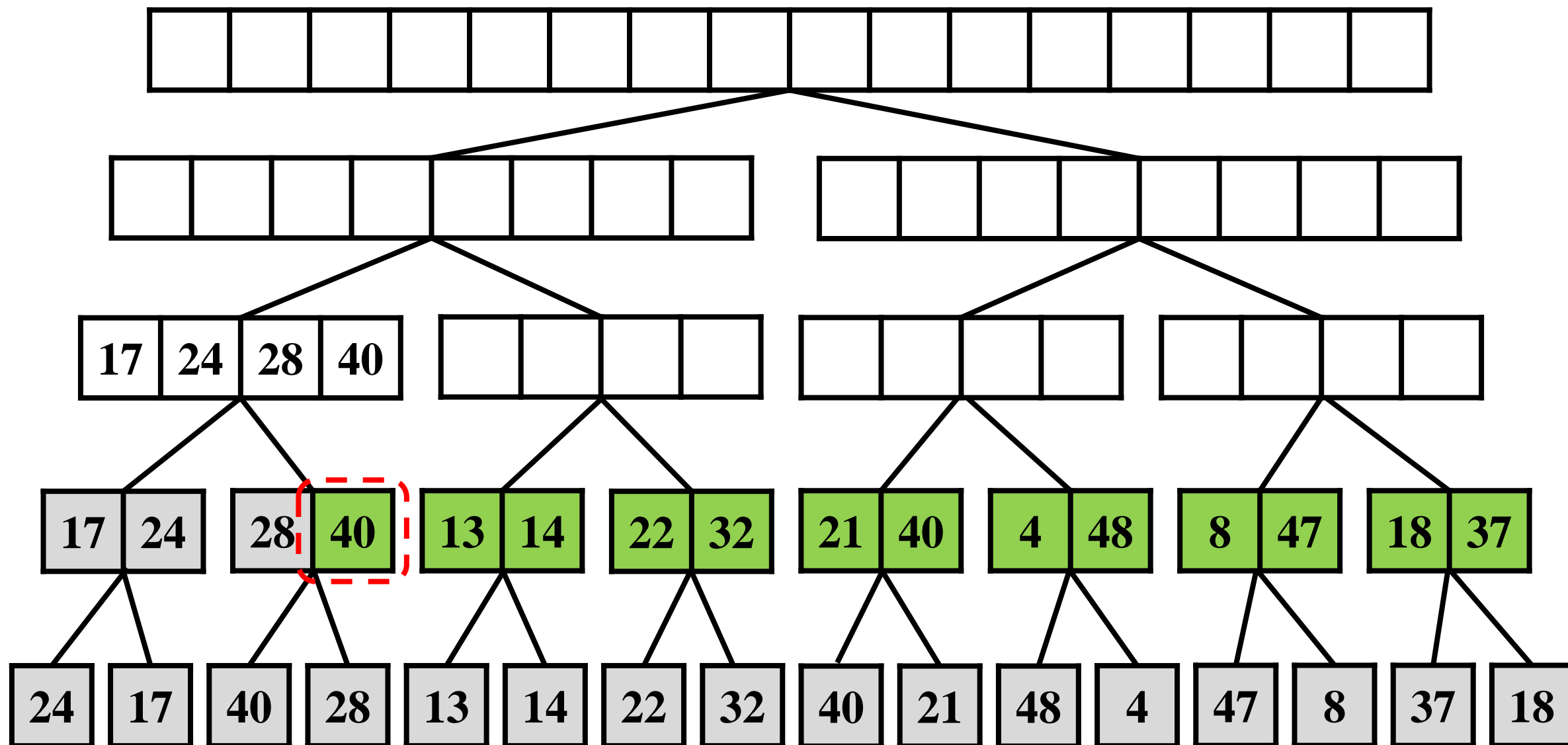
- 构建有序数组：两两合并



# 从杠铃增重问题到排序问题



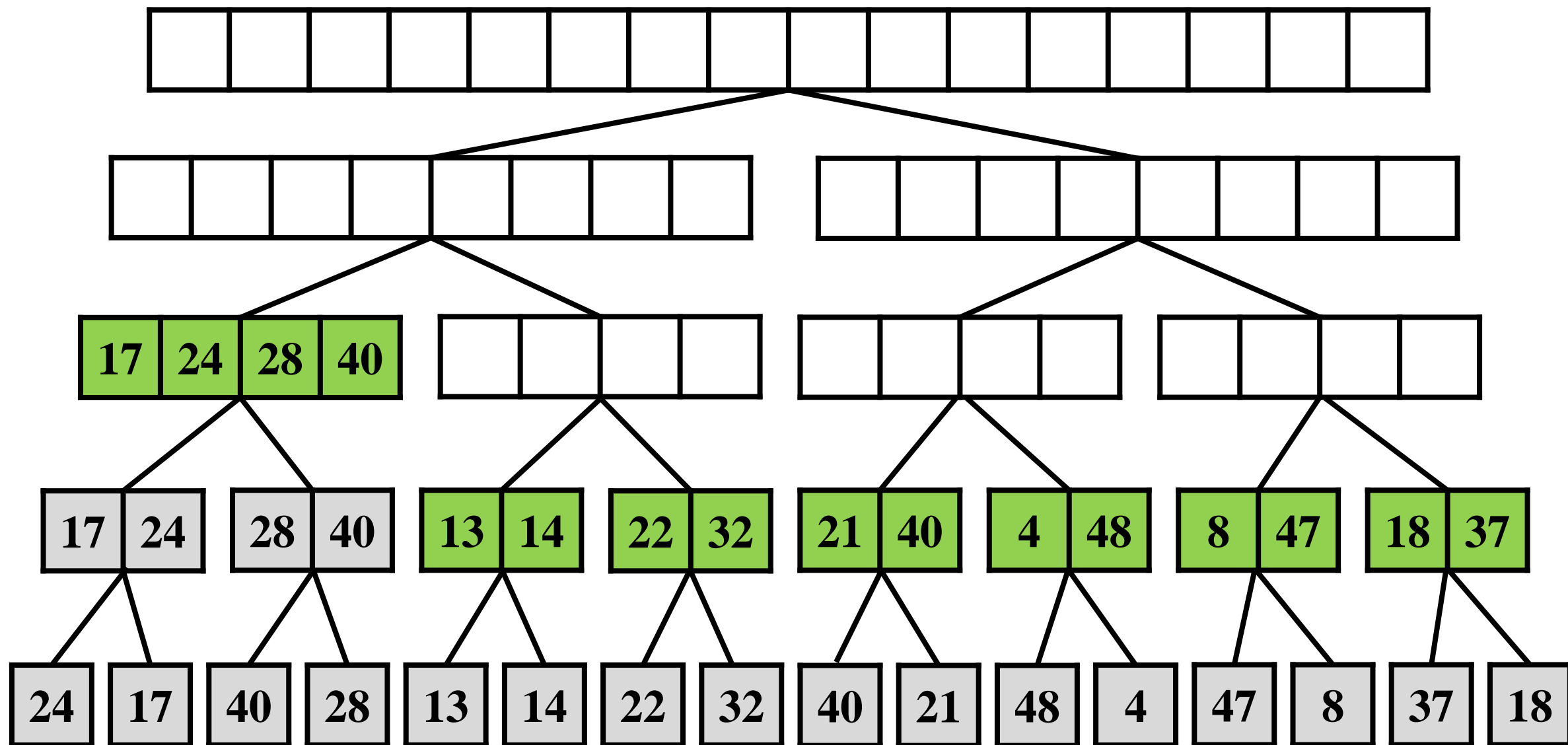
- 构建有序数组：两两合并



# 从杠铃增重问题到排序问题



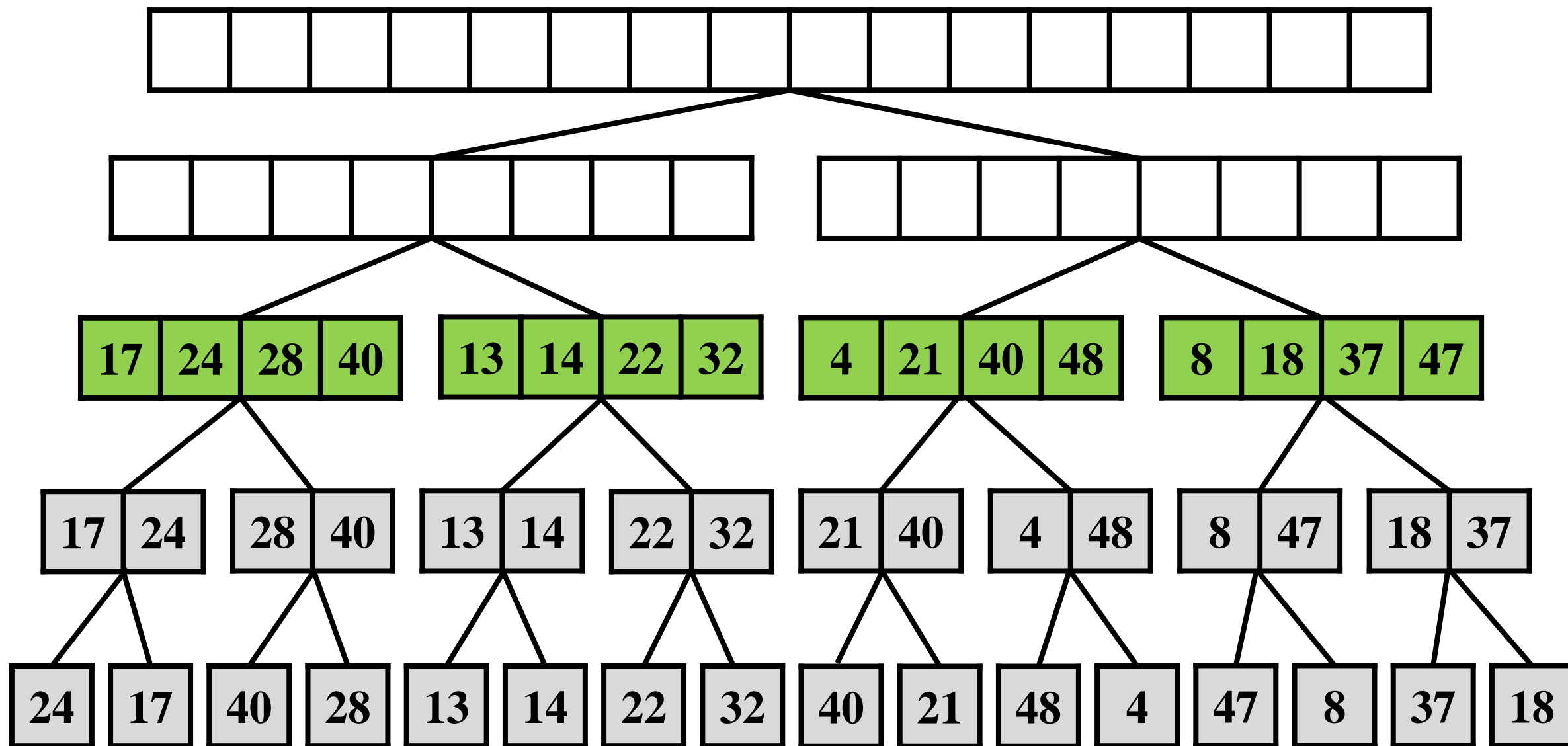
- 构建有序数组：两两合并



# 从杠铃增重问题到排序问题



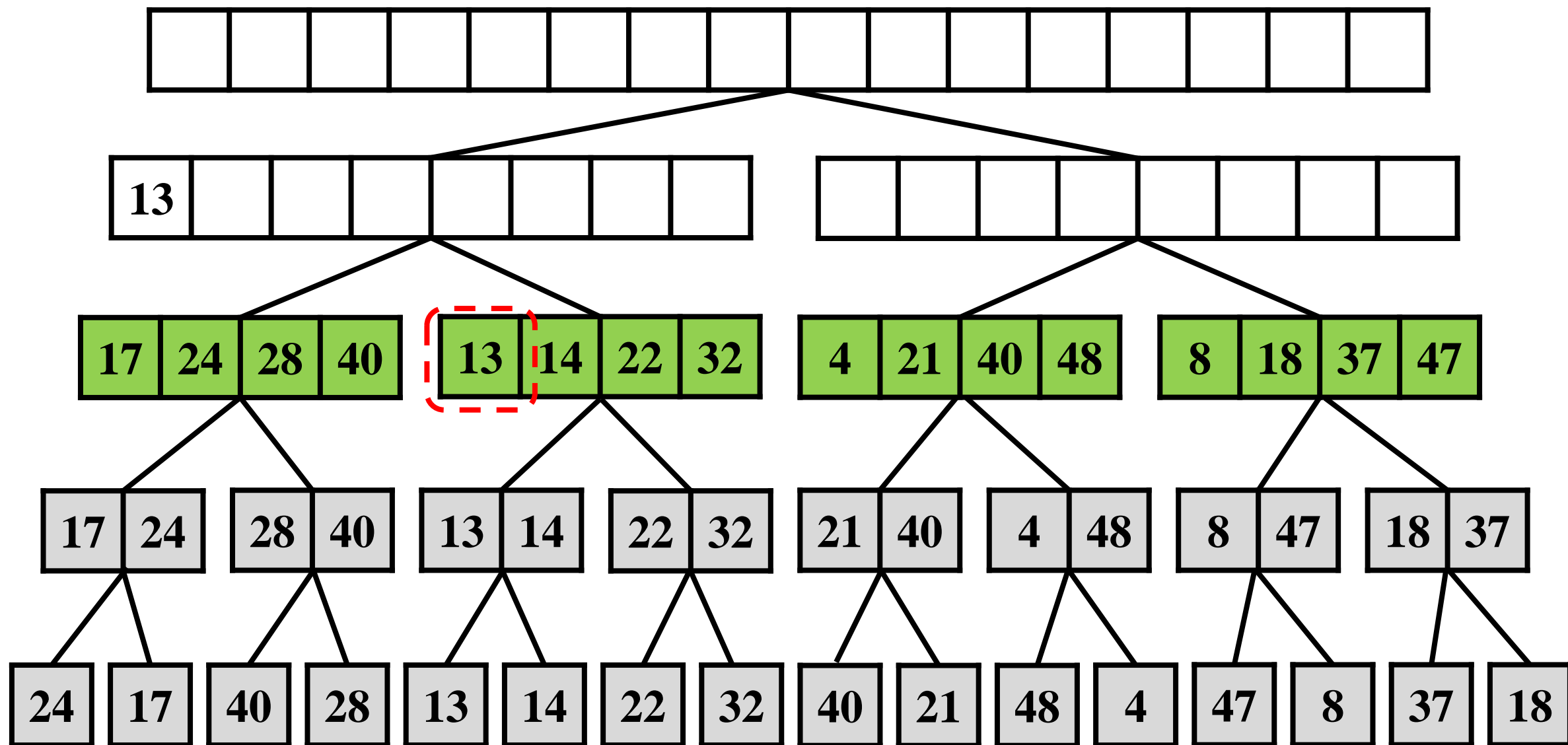
- 构建有序数组：两两合并



# 从杠铃增重问题到排序问题



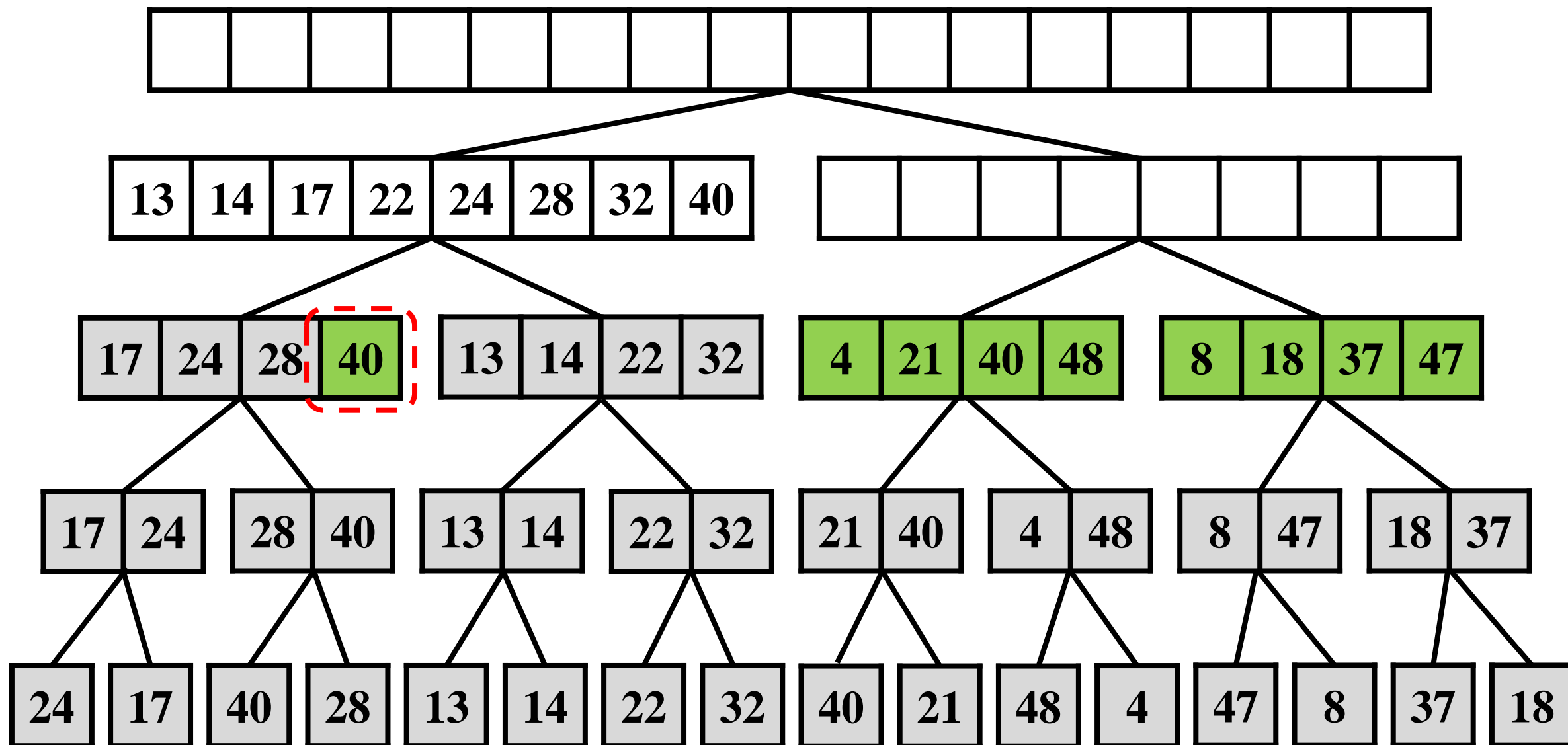
- 构建有序数组：两两合并



# 从杠铃增重问题到排序问题



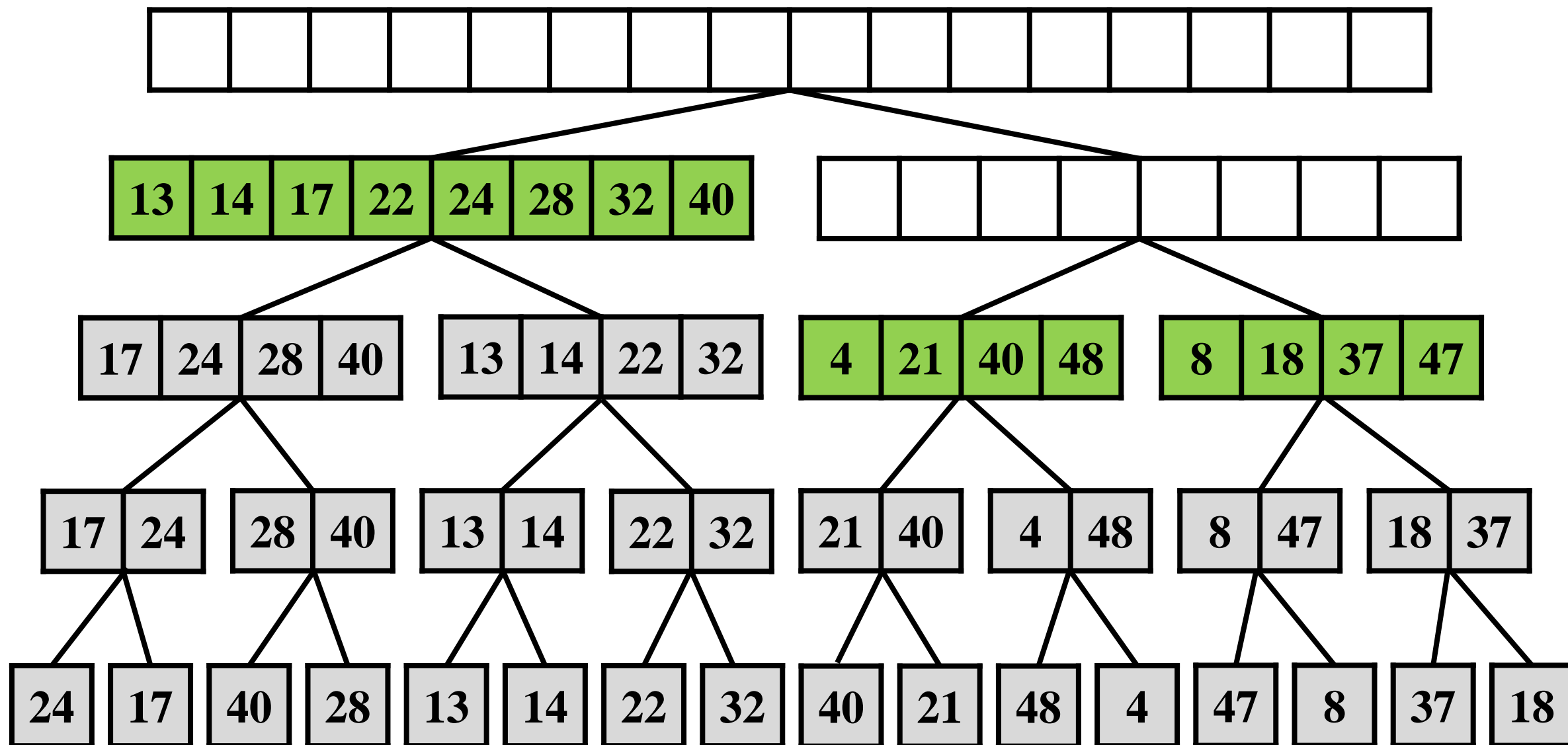
- 构建有序数组：两两合并



# 从杠铃增重问题到排序问题



- 构建有序数组：两两合并

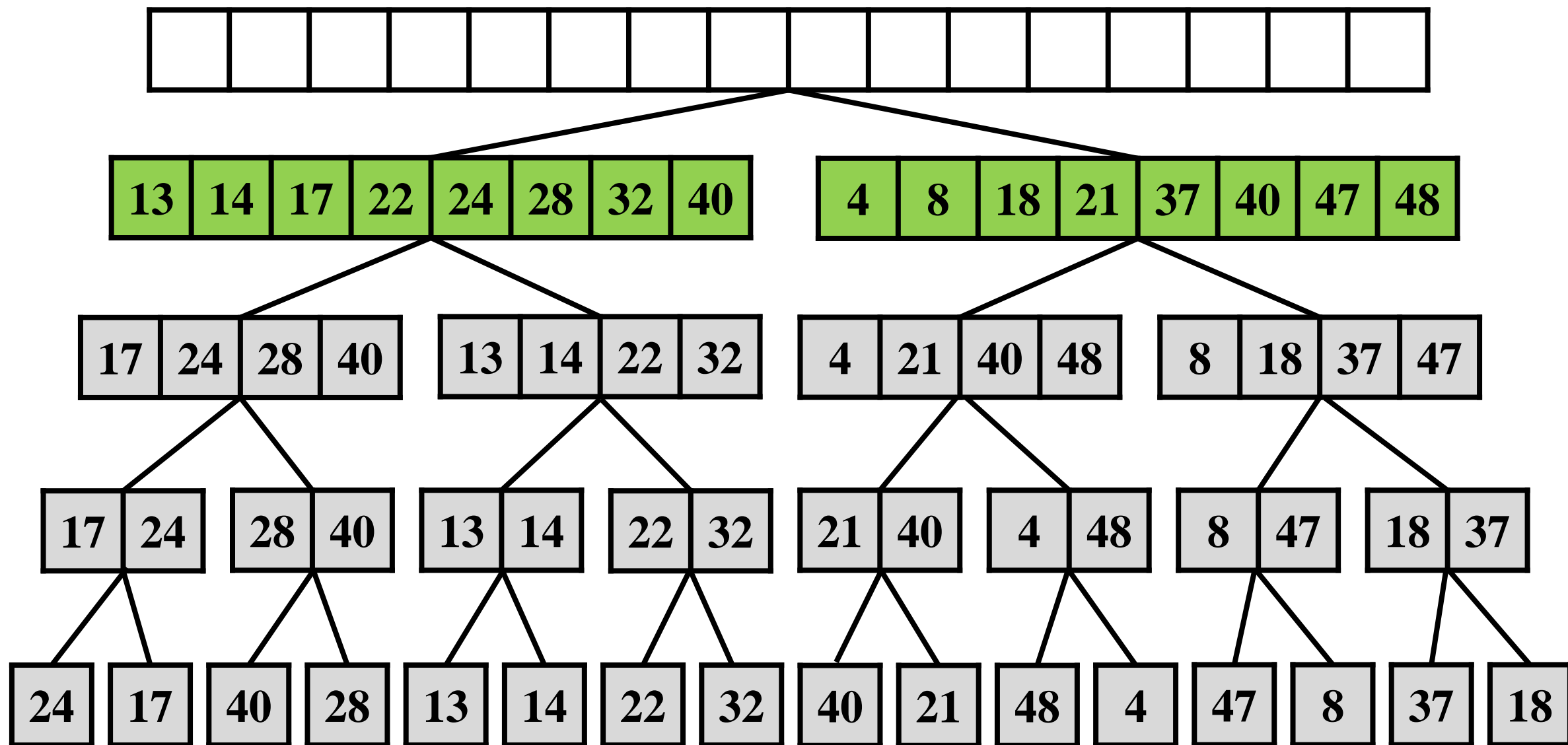




# 从杠铃增重问题到排序问题



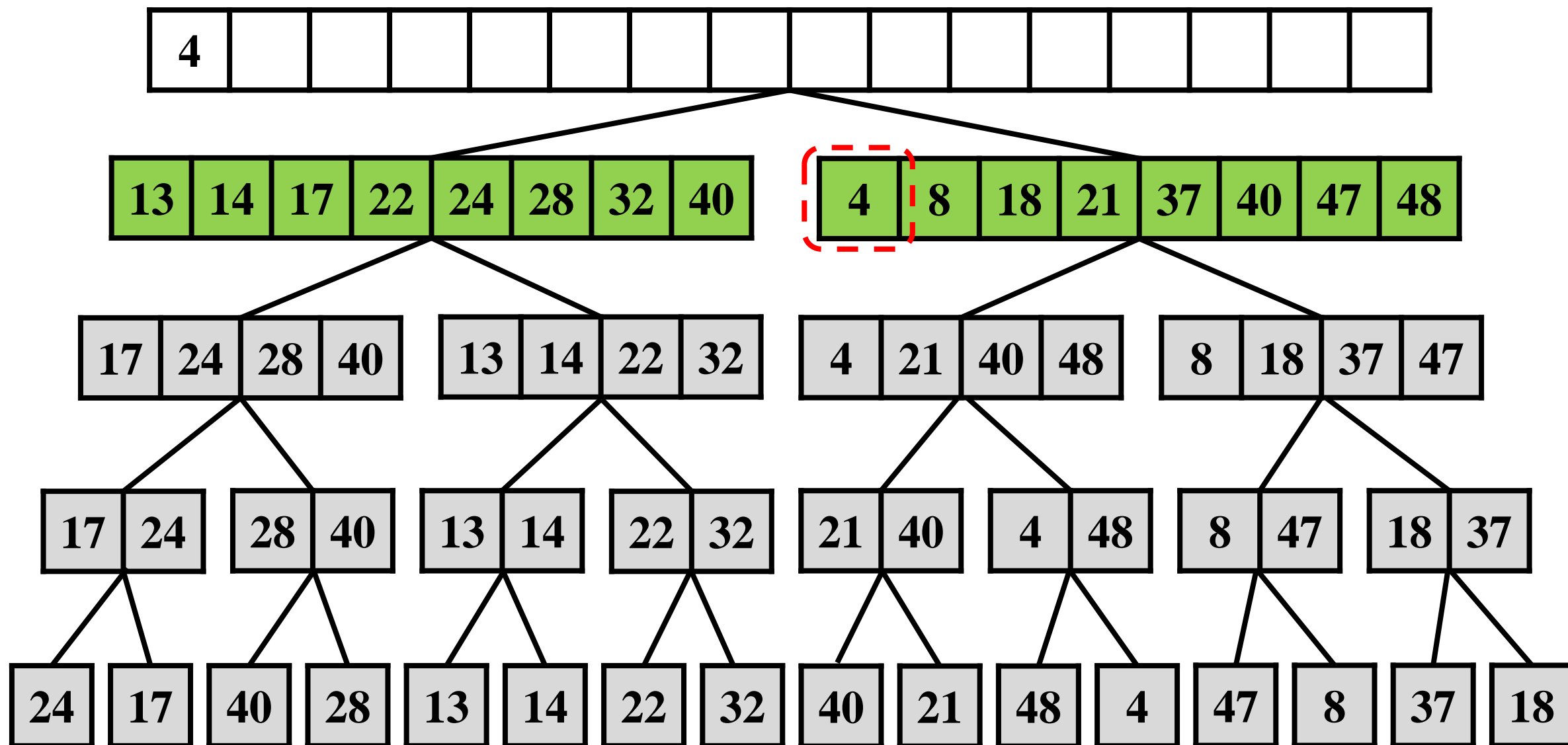
- 构建有序数组：两两合并



# 从杠铃增重问题到排序问题



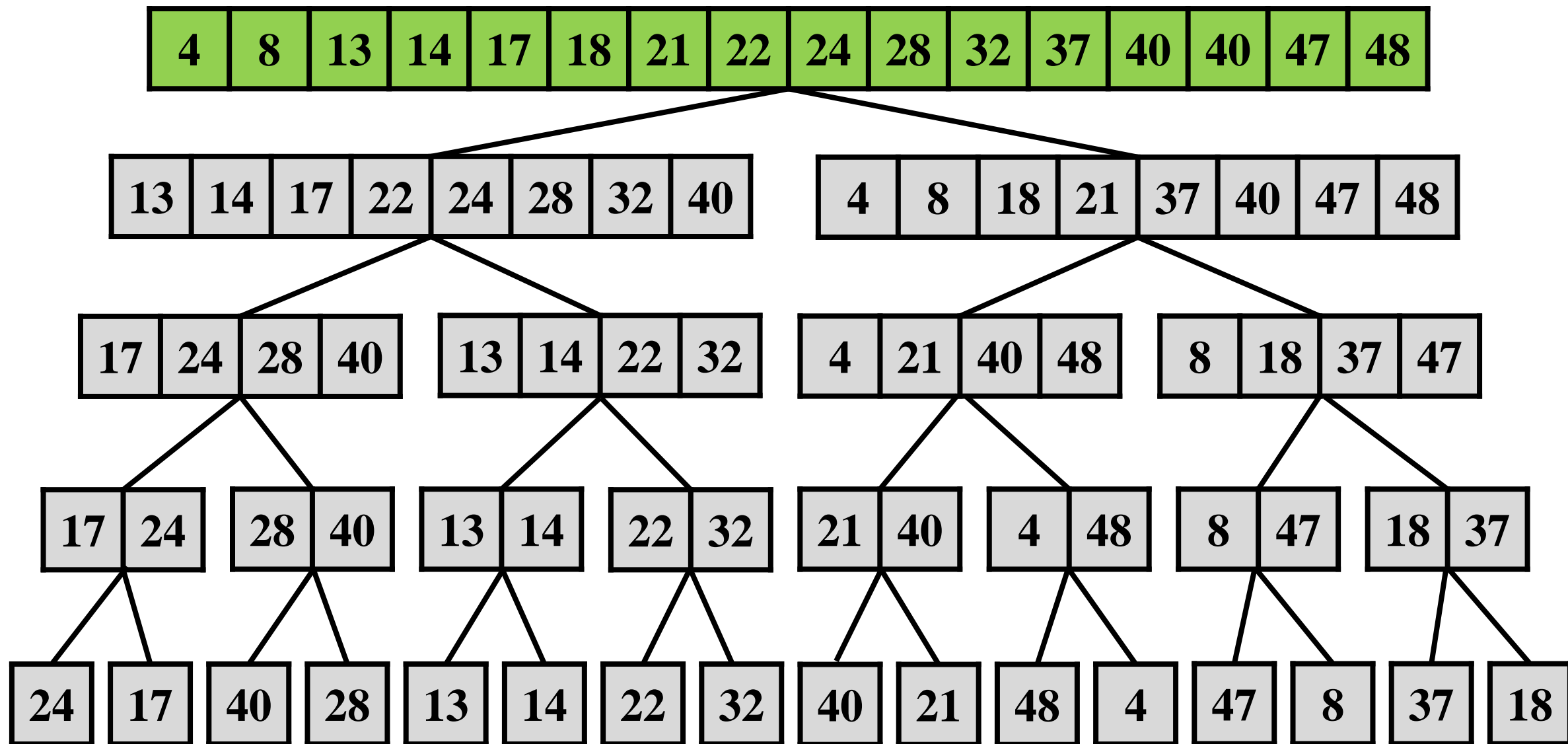
- 构建有序数组：两两合并



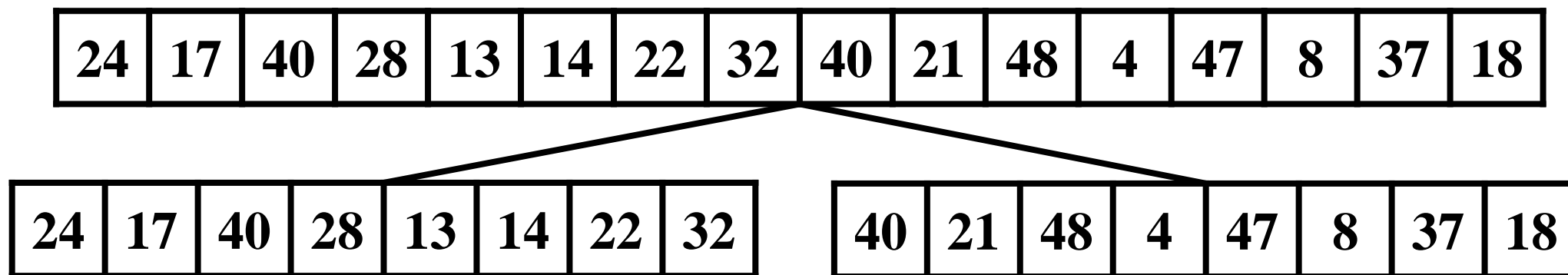
# 从杠铃增重问题到排序问题



- 构建有序数组：两两合并



# 归并排序

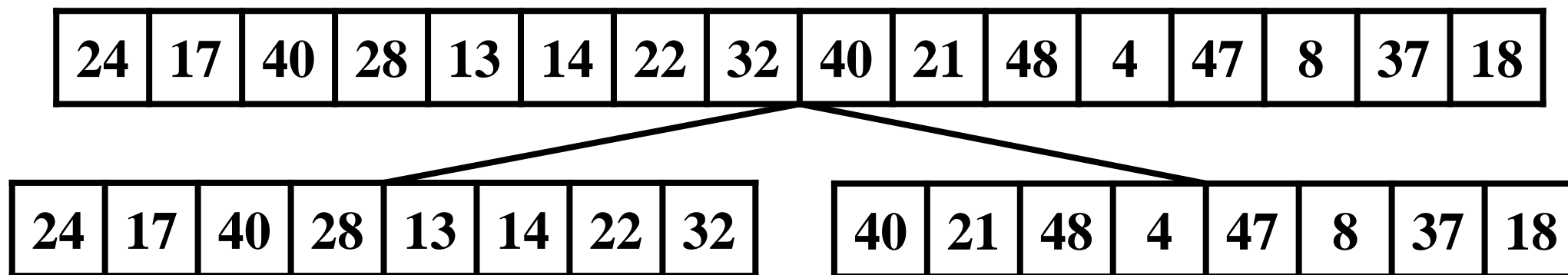


- 1945年，冯·诺伊曼提出归并排序

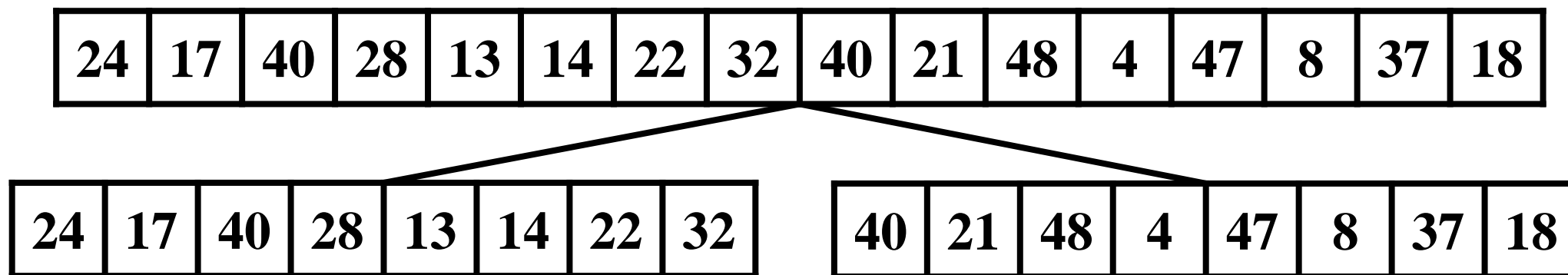


现代计算机之父  
约翰·冯·诺伊曼  
John von Neumann

# 归并排序



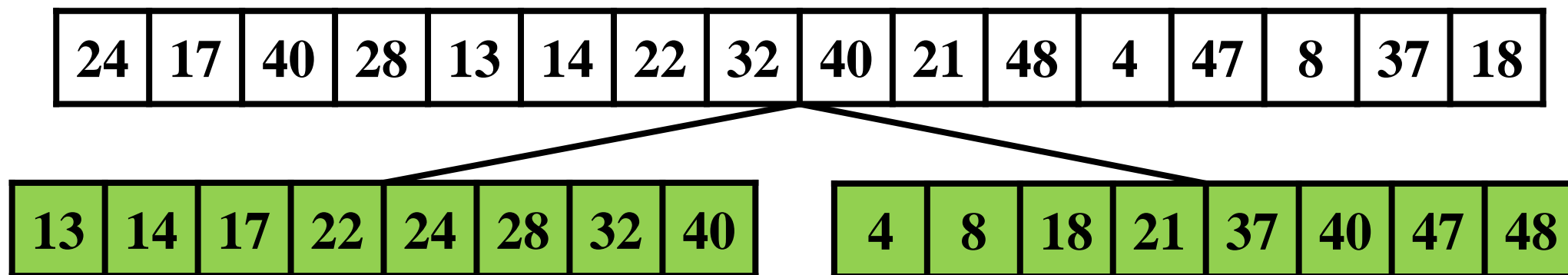
- 算法流程



## ● 算法流程

- 将数组 $A[1, n]$ 排序问题**分解**为 $A[1, \lfloor \frac{n}{2} \rfloor]$ 和 $A[\lfloor \frac{n}{2} \rfloor + 1, n]$ 排序问题

分解原问题

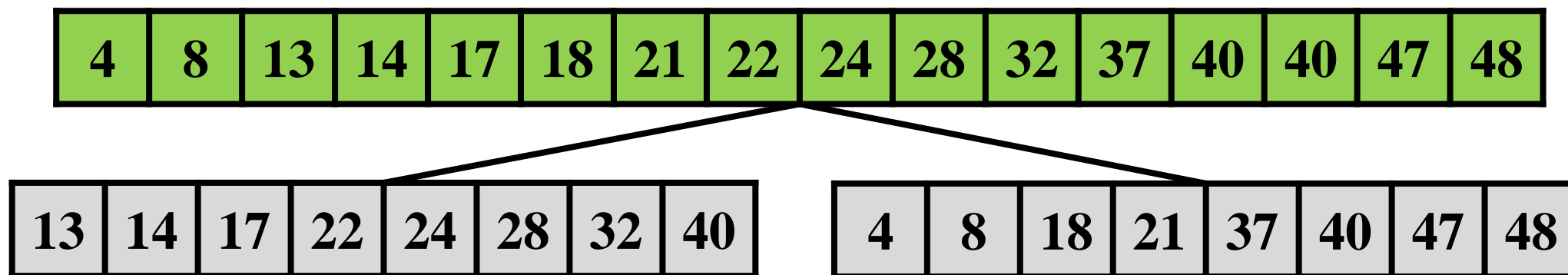


## ● 算法流程

- 将数组 $A[1, n]$ 排序问题**分解**为 $A[1, \lfloor \frac{n}{2} \rfloor]$ 和 $A[\lfloor \frac{n}{2} \rfloor + 1, n]$ 排序问题
- **递归解决**子问题得到两个有序的子数组

分解原问题

解决子问题



## ● 算法流程

- 将数组 $A[1, n]$ 排序问题**分解**为 $A[1, \lfloor \frac{n}{2} \rfloor]$ 和 $A[\lfloor \frac{n}{2} \rfloor + 1, n]$ 排序问题
- **递归解决**子问题得到两个有序的子数组
- 将两个有序子数组**合并**为一个有序数组

分解原问题

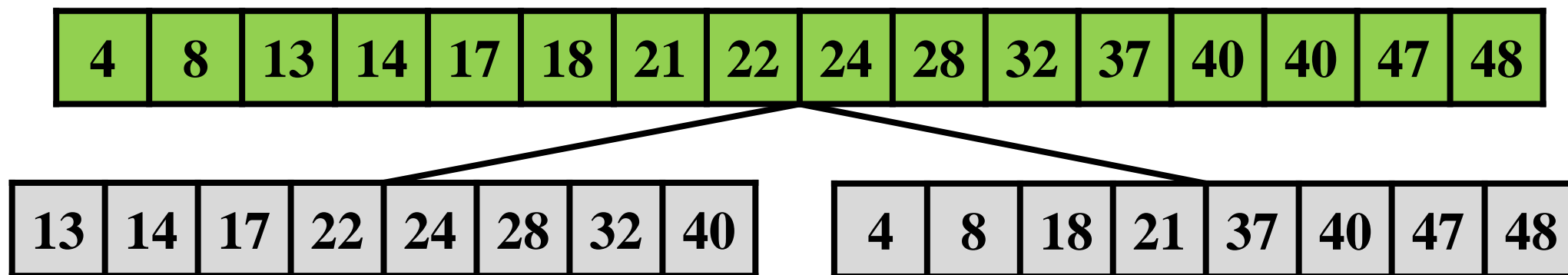
解决子问题

合并问题解

归并排序：分解数组，递归求解，合并排序



# 归并排序



## ● 算法流程

- 将数组 $A[1, n]$ 排序问题**分解**为 $A[1, \lfloor \frac{n}{2} \rfloor]$ 和 $A[\lfloor \frac{n}{2} \rfloor + 1, n]$ 排序问题
- **递归解决**子问题得到两个有序的子数组
- 将两个有序子数组**合并**为一个有序数组

分解原问题

解决子问题

合并问题解

归并排序：分解数组，递归求解，合并排序



分而治之

# 分而治之：一般步骤



**分解**原问题



**解决**子问题



**合并**问题解

原问题分解成多个子问题

递归地求解各个子问题

将结果合并为原问题解

# 归并排序：伪代码

- MergeSort( $A, left, right$ )

初始调用：MergeSort( $A, 1, n$ )

输入：数组  $A[1..n]$ , 数组下标  $left, right$

输出：递增数组  $A[left..right]$

if  $left \geq right$  then

    | return  $A[left..right]$

end

$mid \leftarrow \lfloor \frac{left+right}{2} \rfloor$

MergeSort( $A, left, mid$ )

MergeSort( $A, mid + 1, right$ )

Merge( $A, left, mid, right$ )

return  $A[left..right]$

# 归并排序：伪代码

- MergeSort( $A, left, right$ )

初始调用：MergeSort( $A, 1, n$ )

输入：数组  $A[1..n]$ , 数组下标  $left, right$

输出：递增数组  $A[left..right]$

if  $left \geq right$  then

| return  $A[left..right]$

end

$mid \leftarrow \lfloor \frac{left+right}{2} \rfloor$

MergeSort( $A, left, mid$ )

MergeSort( $A, mid + 1, right$ )

Merge( $A, left, mid, right$ )

return  $A[left..right]$

递归终止：仅有一个元素

# 归并排序：伪代码

- MergeSort( $A, left, right$ )

初始调用：MergeSort( $A, 1, n$ )

输入：数组  $A[1..n]$ , 数组下标  $left, right$

输出：递增数组  $A[left..right]$

if  $left \geq right$  then

    | return  $A[left..right]$

end

$mid \leftarrow \lfloor \frac{left+right}{2} \rfloor$

MergeSort( $A, left, mid$ )

MergeSort( $A, mid + 1, right$ )

Merge( $A, left, mid, right$ )

return  $A[left..right]$

计算子问题规模

# 归并排序：伪代码

- MergeSort( $A, left, right$ )

初始调用：MergeSort( $A, 1, n$ )

输入：数组  $A[1..n]$ , 数组下标  $left, right$

输出：递增数组  $A[left..right]$

if  $left \geq right$  then

    | return  $A[left..right]$

end

$mid \leftarrow \lfloor \frac{left+right}{2} \rfloor$

MergeSort( $A, left, mid$ )

MergeSort( $A, mid + 1, right$ )

Merge( $A, left, mid, right$ )

return  $A[left..right]$

递归求解子问题

# 归并排序：伪代码

- MergeSort( $A, left, right$ )

初始调用：MergeSort( $A, 1, n$ )

输入：数组  $A[1..n]$ , 数组下标  $left, right$

输出：递增数组  $A[left..right]$

if  $left \geq right$  then  
| return  $A[left..right]$

end

$mid \leftarrow \lfloor \frac{left+right}{2} \rfloor$

MergeSort( $A, left, mid$ )

MergeSort( $A, mid + 1, right$ )

Merge( $A, left, mid, right$ )

return  $A[left..right]$

合并子问题解

# 归并排序：伪代码



- Merge( $A, left, mid, right$ )

输入: 数组  $A[1..n]$ , 数组下标  $left, mid, right$

输出: 递增数组  $A[left..right]$

$A'[left..right] \leftarrow A[left..right]$

$i \leftarrow left, j \leftarrow mid + 1, k \leftarrow 0$

**while**  $i \leq mid$  **and**  $j \leq right$  **do**

**if**  $A'[i] \leq A'[j]$  **then**

$A[left + k] \leftarrow A'[i]$

$k \leftarrow k + 1, i \leftarrow i + 1$

**end**

**else**

$A[left + k] \leftarrow A'[j]$

$k \leftarrow k + 1, j \leftarrow j + 1$

**end**

**end**

**if**  $i \leq mid$  **then**

$A[left + k..right] \leftarrow A'[i..mid]$

**end**

**else**

$A[left + k..right] \leftarrow A'[j..right]$

**end**

**return**  $A[left..right]$

初始化



# 归并排序：伪代码



- Merge( $A, left, mid, right$ )

输入: 数组  $A[1..n]$ , 数组下标  $left, mid, right$

输出: 递增数组  $A[left..right]$

$A'[left..right] \leftarrow A[left..right]$

$i \leftarrow left, j \leftarrow mid + 1, k \leftarrow 0$

**while**  $i \leq mid$  **and**  $j \leq right$  **do**

**if**  $A'[i] \leq A'[j]$  **then**

$A[left + k] \leftarrow A'[i]$

$k \leftarrow k + 1, i \leftarrow i + 1$

**end**

**else**

$A[left + k] \leftarrow A'[j]$

$k \leftarrow k + 1, j \leftarrow j + 1$

**end**

**end**

**if**  $i \leq mid$  **then**

$A[left + k..right] \leftarrow A'[i..mid]$

**end**

**else**

$A[left + k..right] \leftarrow A'[j..right]$

**end**

**return**  $A[left..right]$

遍历子数组，进行合并

# 归并排序：伪代码



- Merge( $A, left, mid, right$ )

输入: 数组  $A[1..n]$ , 数组下标  $left, mid, right$

输出: 递增数组  $A[left..right]$

$A'[left..right] \leftarrow A[left..right]$

$i \leftarrow left, j \leftarrow mid + 1, k \leftarrow 0$

**while**  $i \leq mid$  **and**  $j \leq right$  **do**

**if**  $A'[i] \leq A'[j]$  **then**

$A[left + k] \leftarrow A'[i]$

$k \leftarrow k + 1, i \leftarrow i + 1$

**end**

**else**

$A[left + k] \leftarrow A'[j]$

$k \leftarrow k + 1, j \leftarrow j + 1$

**end**

**end**

**if**  $i \leq mid$  **then**

$A[left + k..right] \leftarrow A'[i..mid]$

**end**

**else**

$A[left + k..right] \leftarrow A'[j..right]$

**end**

**return**  $A[left..right]$

添加剩余元素保证有序

# 归并排序：伪代码



- Merge( $A, left, mid, right$ )

输入: 数组  $A[1..n]$ , 数组下标  $left, mid, right$

输出: 递增数组  $A[left..right]$

$A'[left..right] \leftarrow A[left..right]$

$i \leftarrow left, j \leftarrow mid + 1, k \leftarrow 0$

**while**  $i \leq mid$  **and**  $j \leq right$  **do**

**if**  $A'[i] \leq A'[j]$  **then**

$A[left + k] \leftarrow A'[i]$

$k \leftarrow k + 1, i \leftarrow i + 1$

**end**

**else**

$A[left + k] \leftarrow A'[j]$

$k \leftarrow k + 1, j \leftarrow j + 1$

**end**

**end**

**if**  $i \leq mid$  **then**

$A[left + k..right] \leftarrow A'[i..mid]$

**end**

**else**

$A[left + k..right] \leftarrow A'[j..right]$

**end**

**return**  $A[left..right]$

时间复杂度:  $O(n)$

# 归并排序：复杂度分析

---



- $T(n)$  : 完成MergeSort( $A, 1, n$ ) 的运行时间
  - 为便于分析, 假设 $n$ 是2的幂

# 归并排序：复杂度分析

- $T(n)$  : 完成MergeSort( $A, 1, n$ ) 的运行时间
  - 为便于分析, 假设 $n$ 是2的幂
- MergeSort( $A, left, right$ )

输入: 数组 $A[1..n]$ , 数组下标 $left, right$

输出: 递增数组 $A[left..right]$

if  $left \geq right$  then

    | return  $A[left..right]$

end

$mid \leftarrow \lfloor \frac{left+right}{2} \rfloor$

MergeSort( $A, left, mid$ )

MergeSort( $A, mid + 1, right$ )

Merge( $A, left, mid, right$ )

return  $A[left..right]$

# 归并排序：复杂度分析

- $T(n)$  : 完成MergeSort( $A, 1, n$ ) 的运行时间
  - 为便于分析, 假设 $n$ 是2的幂
- MergeSort( $A, left, right$ )

输入: 数组 $A[1..n]$ , 数组下标 $left, right$

输出: 递增数组 $A[left..right]$

if  $left \geq right$  then

    | return  $A[left..right]$

end

$mid \leftarrow \lfloor \frac{left+right}{2} \rfloor$

MergeSort( $A, left, mid$ )

MergeSort( $A, mid + 1, right$ )

Merge( $A, left, mid, right$ )

return  $A[left..right]$

}  $O(1)$

→  $T(n/2)$

→  $T(n/2)$

→  $O(n)$

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

# 归并排序：复杂度分析

- $T(n)$  : 完成MergeSort( $A, 1, n$ ) 的运行时间
  - 为便于分析, 假设 $n$ 是2的幂
- MergeSort( $A, left, right$ )

输入: 数组 $A[1..n]$ , 数组下标 $left, right$

输出: 递增数组 $A[left..right]$

if  $left \geq right$  then

    | return  $A[left..right]$

end

$mid \leftarrow \lfloor \frac{left+right}{2} \rfloor$

MergeSort( $A, left, mid$ )

MergeSort( $A, mid + 1, right$ )

Merge( $A, left, mid, right$ )

return  $A[left..right]$

}  $O(1)$

→  $T(n/2)$

→  $T(n/2)$

→  $O(n)$

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

# 归并排序：复杂度分析



- 递归树法：用树的形式表示抽象递归

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



# 归并排序：复杂度分析



- 递归树法：用树的形式表示抽象递归

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

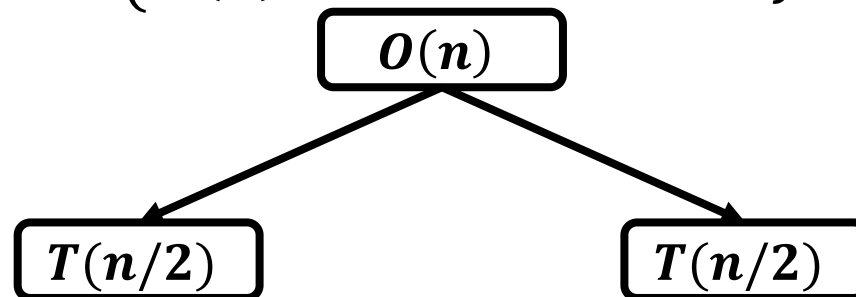
$T(n)$

# 归并排序：复杂度分析



- 递归树法：用树的形式表示抽象递归

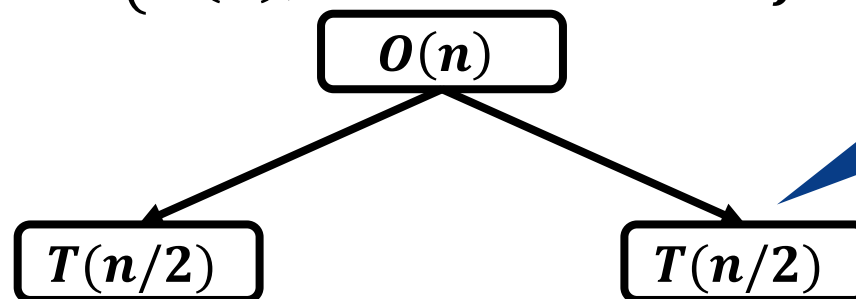
$$T(n) = \begin{cases} 2T(n/2) + O(n), & \text{if } n > 1 \\ O(1), & \text{if } n = 1 \end{cases}$$



# 归并排序：复杂度分析

- 递归树法：用树的形式表示抽象递归

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



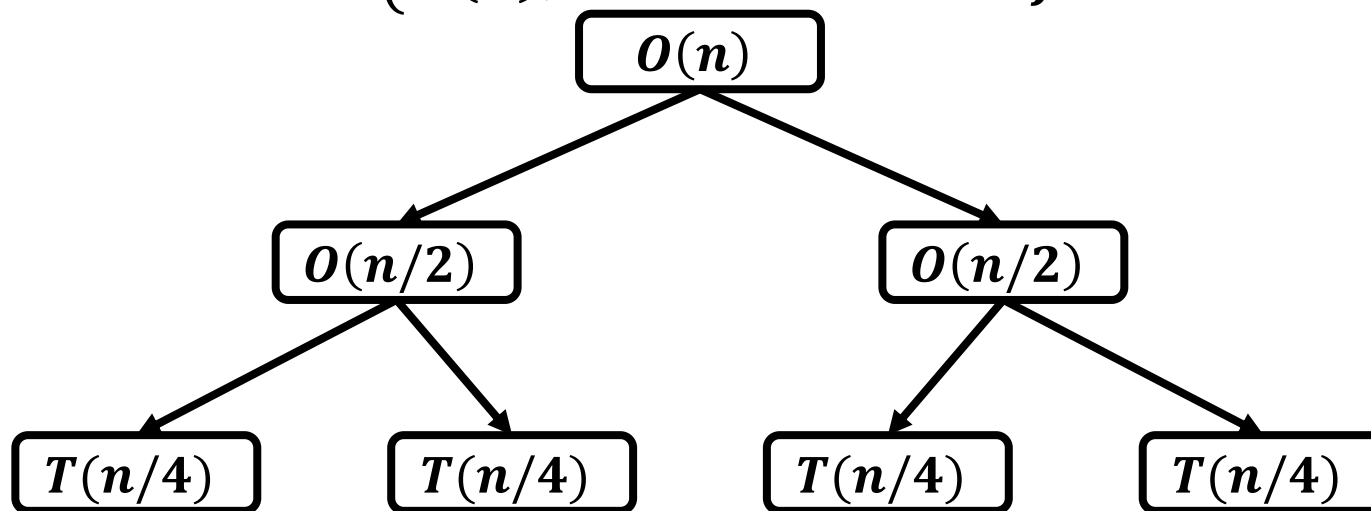
每个结点代表解决一个子问题的代价

# 归并排序：复杂度分析



- 递归树法：用树的形式表示抽象递归

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

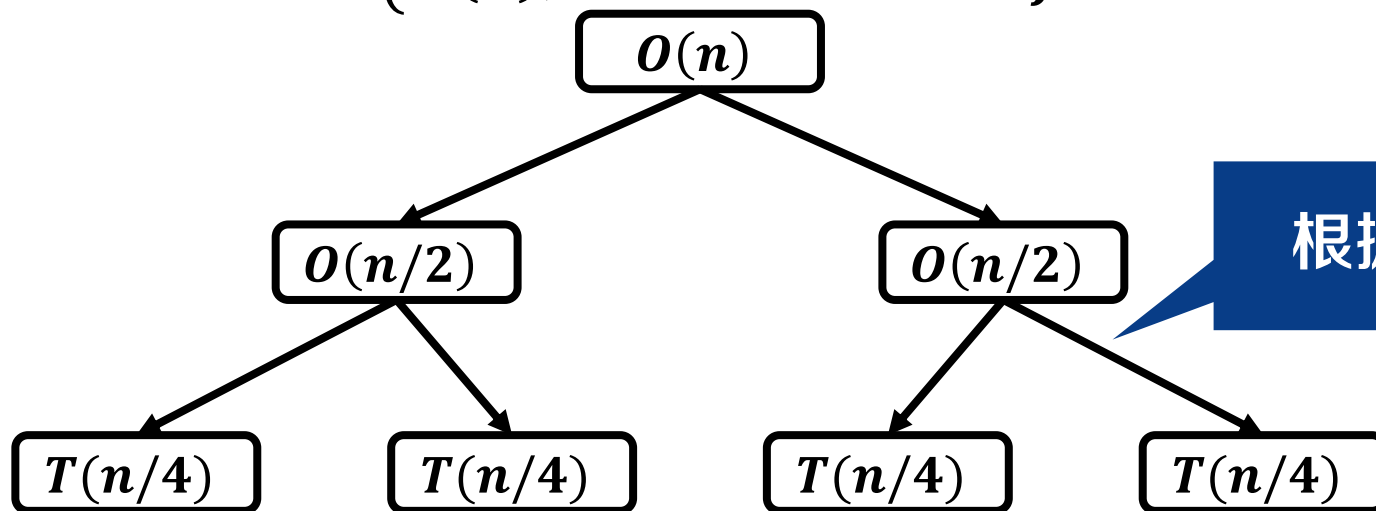


# 归并排序：复杂度分析



- 递归树法：用树的形式表示抽象递归

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



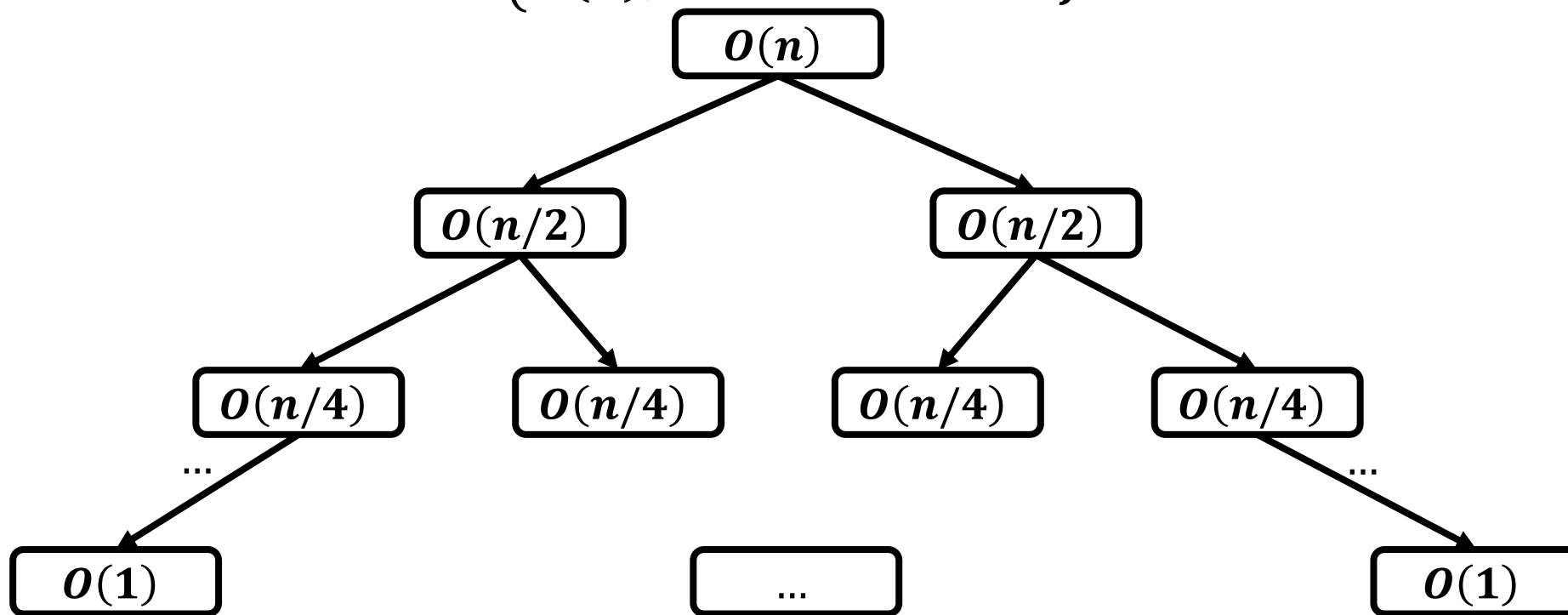
根据递归式向下分解

# 归并排序：复杂度分析



- 递归树法：用树的形式表示抽象递归

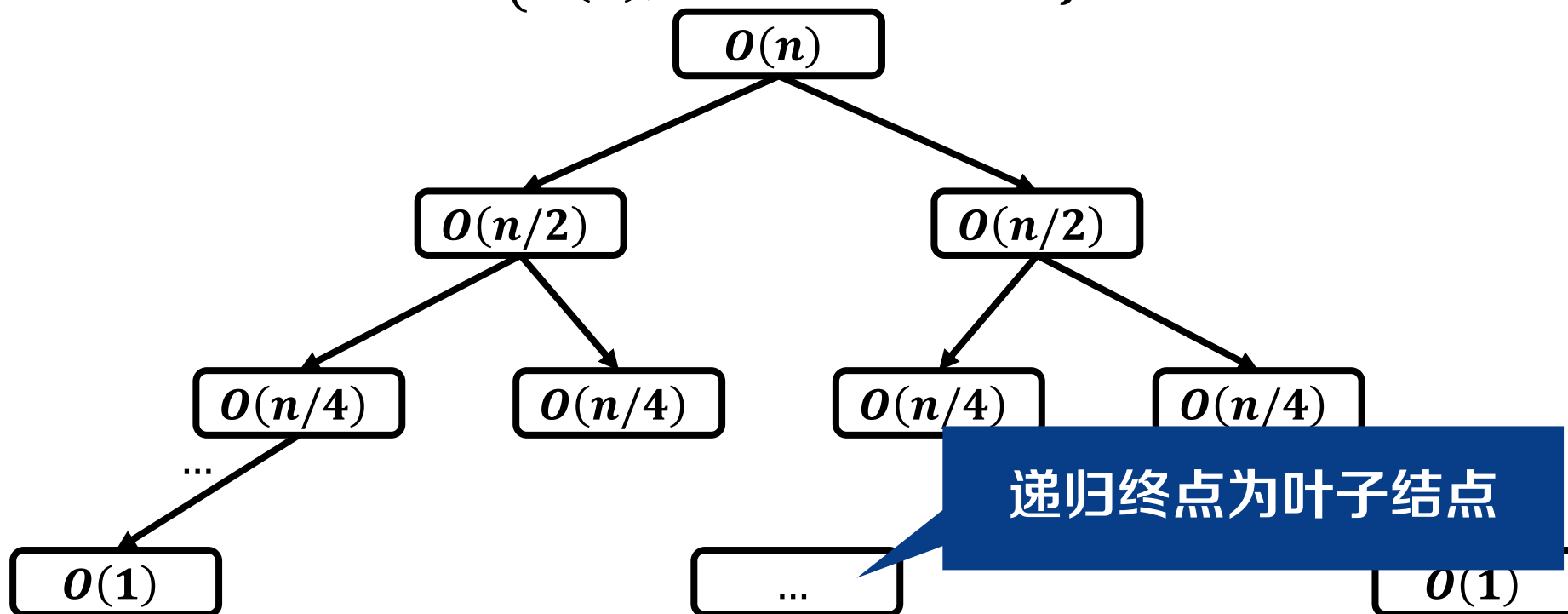
$$T(n) = \begin{cases} 2T(n/2) + O(n), & \text{if } n > 1 \\ O(1), & \text{if } n = 1 \end{cases}$$



# 归并排序：复杂度分析

- 递归树法：用树的形式表示抽象递归

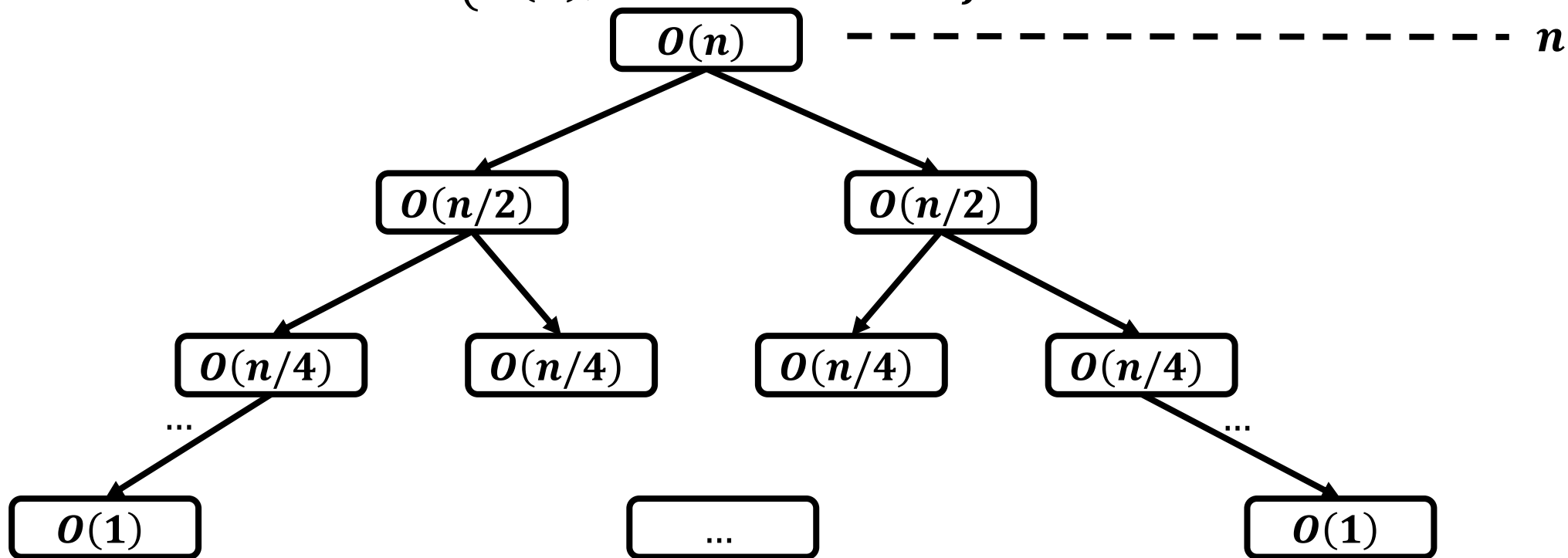
$$T(n) = \begin{cases} 2T(n/2) + O(n), & \text{if } n > 1 \\ O(1), & \text{if } n = 1 \end{cases}$$



# 归并排序：复杂度分析

- 递归树法：用树的形式表示抽象递归

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

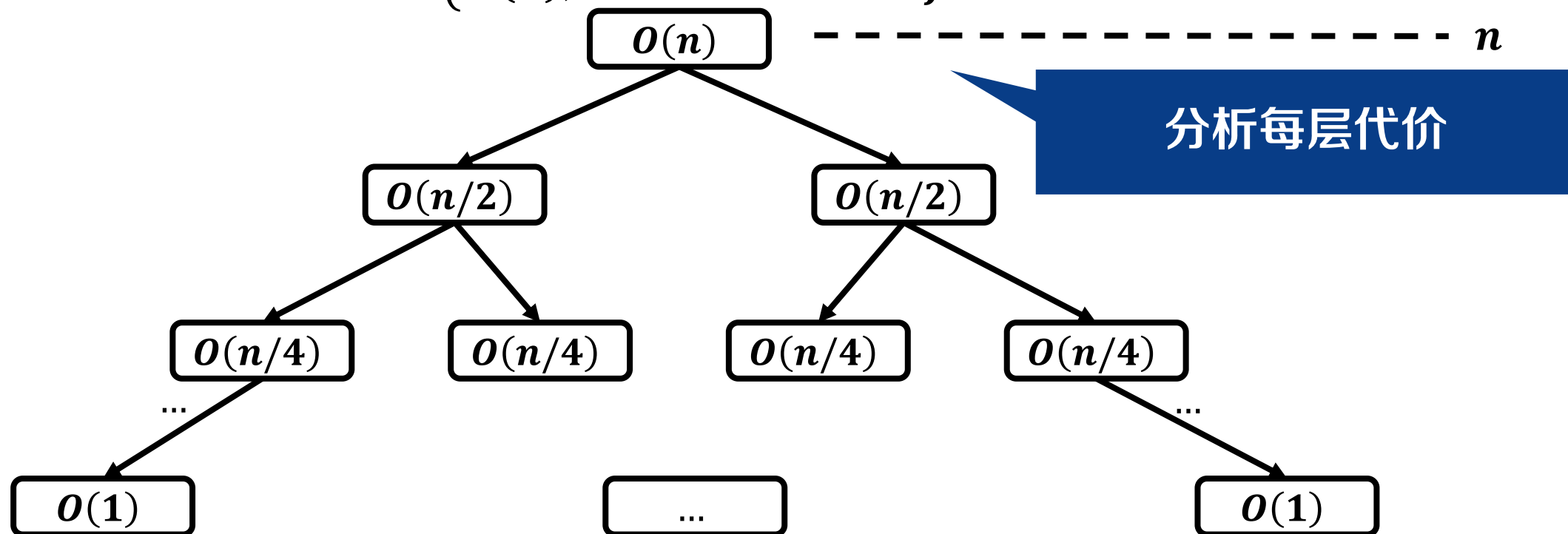




# 归并排序：复杂度分析

- 递归树法：用树的形式表示抽象递归

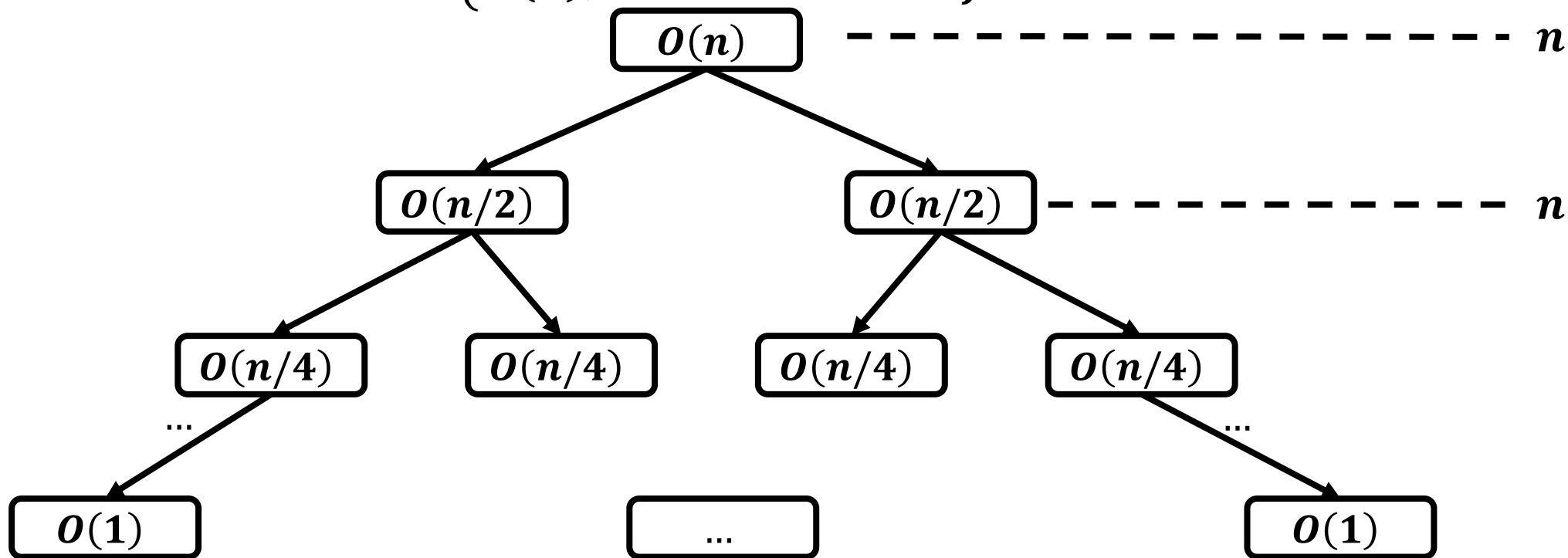
$$T(n) = \begin{cases} 2T(n/2) + O(n), & \text{if } n > 1 \\ O(1), & \text{if } n = 1 \end{cases}$$



# 归并排序：复杂度分析

- 递归树法：用树的形式表示抽象递归

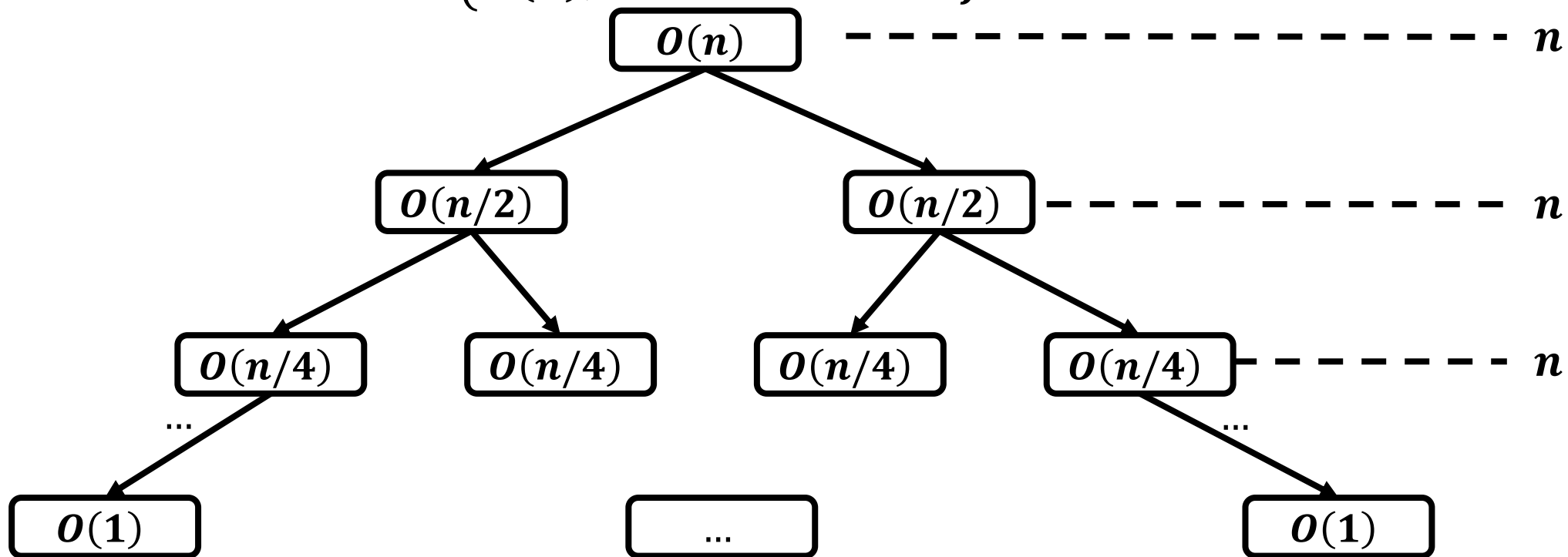
$$T(n) = \begin{cases} 2T(n/2) + O(n), & \text{if } n > 1 \\ O(1), & \text{if } n = 1 \end{cases}$$



# 归并排序：复杂度分析

- 递归树法：用树的形式表示抽象递归

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

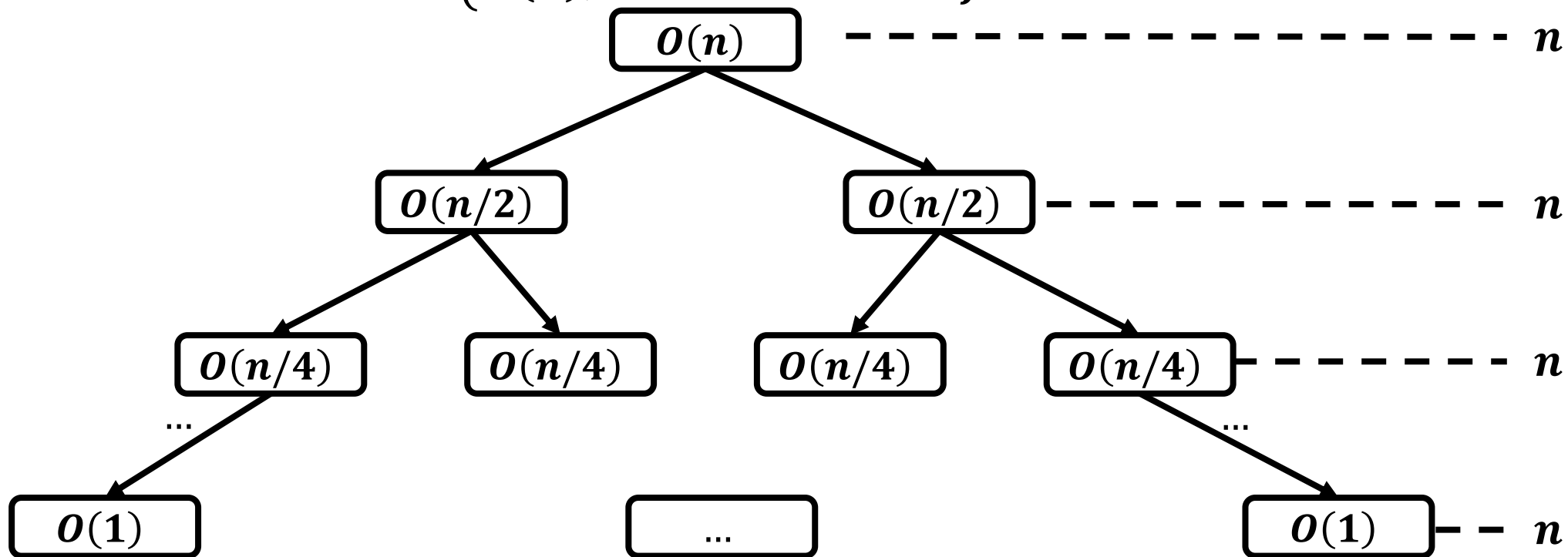


# 归并排序：复杂度分析



- 递归树法：用树的形式表示抽象递归

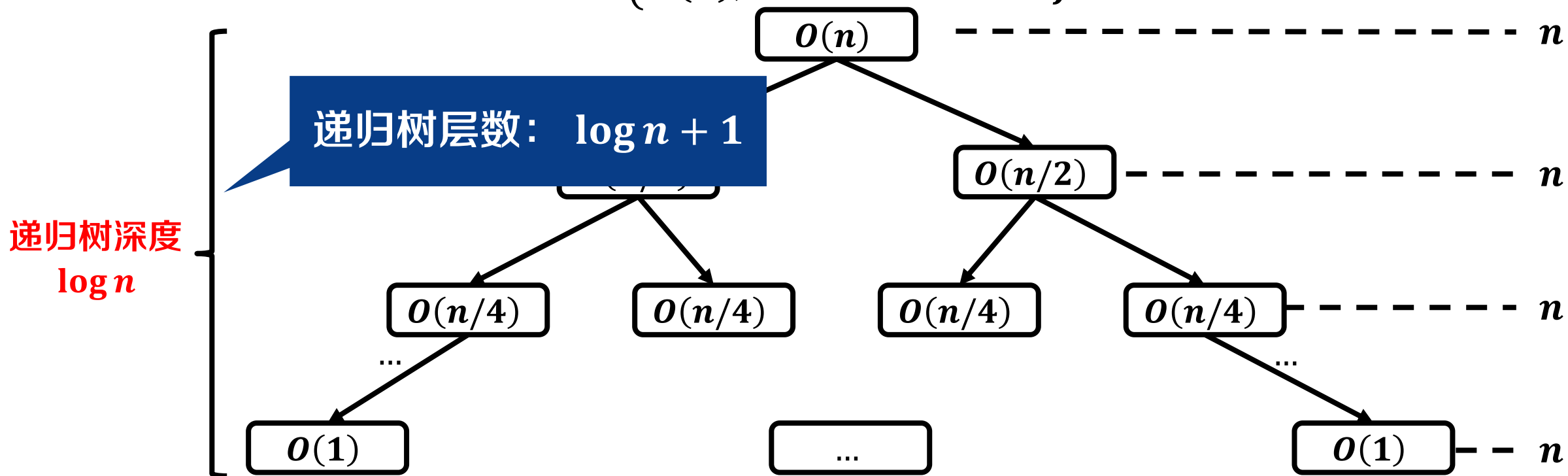
$$T(n) = \begin{cases} 2T(n/2) + O(n), & \text{if } n > 1 \\ O(1), & \text{if } n = 1 \end{cases}$$



# 归并排序：复杂度分析

- 递归树法：用树的形式表示抽象递归

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

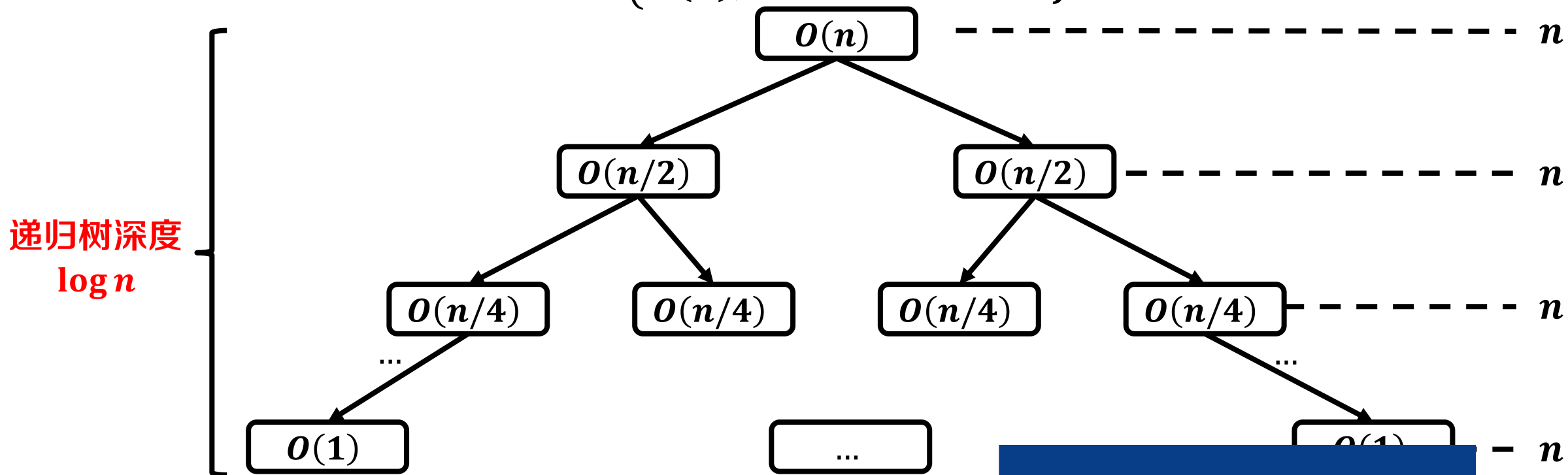


由于树的深度通常由0开始计数，故  
层数=深度+1，后续统一用“深度”

# 归并排序：复杂度分析

- 递归树法：用树的形式表示抽象递归

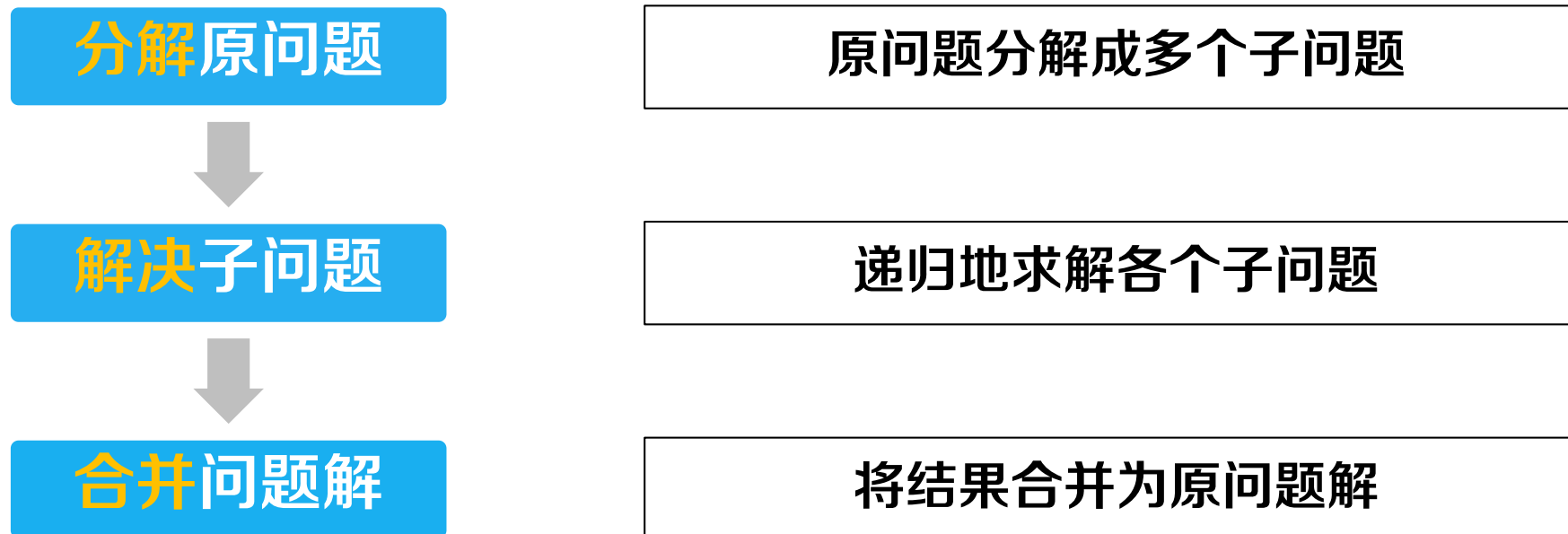
$$T(n) = \begin{cases} 2T(n/2) + O(n), & \text{if } n > 1 \\ O(1), & \text{if } n = 1 \end{cases}$$



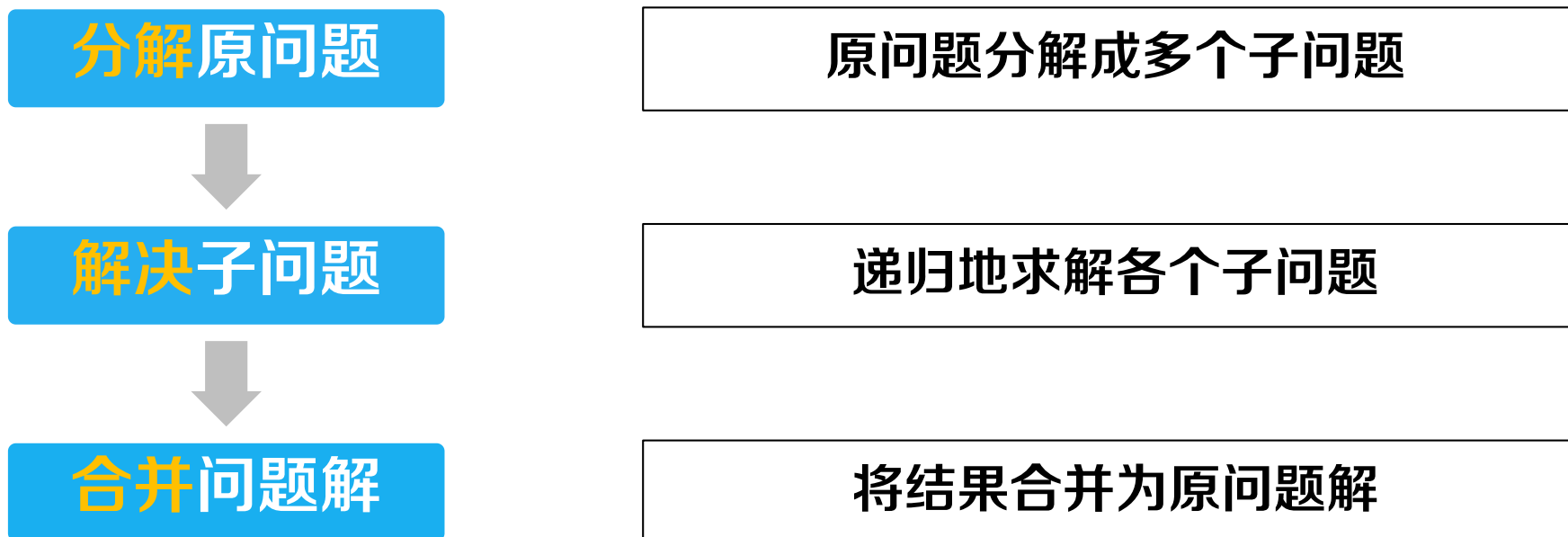
二者相乘为总代价

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

- 分而治之框架



- 分而治之框架



- 归并排序算法分析

- 含递归式的时间复杂度

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