

快速排序法 (Quick Sort) – $O(n \log n)$

- 範例: 9, 5, 12, 19, 2, 6, 4, 15, 8, 3, 7
- 目標: 2, 3, 4, 5, 6, 7, 8, 9, 12, 15, 19

快速排序法 (Quick Sort) – $O(n \log n)$

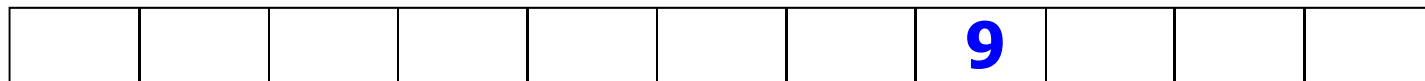
- 範例: 9, 5, 12, 19, 2, 6, 4, 15, 8, 3, 7
- 目標: 2, 3, 4, 5, 6, 7, 8, 9, 12, 15, 19
- 遞迴函式設計:

快速排序法 (Quick Sort) – $O(n \log n)$

- 範例: 9, 5, 12, 19, 2, 6, 4, 15, 8, 3, 7
- 目標: 2, 3, 4, 5, 6, 7, 8, 9, 12, 15, 19
- 遞迴函式設計:
 - 分治法 (Divide and Conquer): 「切割問題, 各個擊破」

快速排序法 (Quick Sort) – $O(n \log n)$

- 範例: 9, 5, 12, 19, 2, 6, 4, 15, 8, 3, 7
 - 目標: 2, 3, 4, 5, 6, 7, 8, 9, 12, 15, 19
 - 遞迴函式設計:
 - 分治法 (Divide and Conquer): 「切割問題, 各個擊破」
 - 每一步驟挑選一個元素**放在正確的位置上**, 將其它數字分為兩群, 大於或是等於這個元素的放右邊, 小於的放左邊
- 例如: 將第一個元素 **9** 放在正確位置上



{5, 2, 6, 4, 8, 3, 7}

{12, 19, 15}

快速排序法 (Quick Sort) – $O(n \log n)$

- 範例: 9, 5, 12, 19, 2, 6, 4, 15, 8, 3, 7
 - 目標: 2, 3, 4, 5, 6, 7, 8, 9, 12, 15, 19
 - 遞迴函式設計:
 - 分治法 (Divide and Conquer): 「切割問題, 各個擊破」
 - 每一步驟挑選一個元素放在正確的位置上, 將其它數字分為兩群, 大於或是等於這個元素的放右邊, 小於的放左邊
- 例如: 將第一個元素 9 放在正確位置上

							9			
--	--	--	--	--	--	--	---	--	--	--

{5, 2, 6, 4, 8, 3, 7}

{12, 19, 15}

- 如此就可以得到兩個元素個數比較少的排序問題

快速排序法 (Quick Sort) – $O(n \log n)$

- 範例: 9, 5, 12, 19, 2, 6, 4, 15, 8, 3, 7
 - 目標: 2, 3, 4, 5, 6, 7, 8, 9, 12, 15, 19
 - 遞迴函式設計:
 - 分治法 (Divide and Conquer): 「切割問題, 各個擊破」
 - 每一步驟挑選一個元素放在正確的位置上, 將其它數字分為兩群, 大於或是等於這個元素的放右邊, 小於的放左邊
- 例如: 將第一個元素 9 放在正確位置上

							9			
--	--	--	--	--	--	--	---	--	--	--

{5, 2, 6, 4, 8, 3, 7}

{12, 19, 15}

- 如此就可以得到兩個元素個數比較少的排序問題
- 問題: ①怎麼把 9 放在正確的位置上? 把 9 和其它數字一個一個比較就可以; ②怎樣才能夠有效率地分成兩群?

快速排序法 (Quick Sort) – $O(n \log n)$

- 範例: 9, 5, 12, 19, 2, 6, 4, 15, 8, 3, 7
- 目標: 2, 3, 4, 5, 6, 7, 8, 9, 12, 15, 19
- 遞迴函式設計:

– 分治法 (Divide and Conquer): 「切割問題, 各個擊破」

– 每一步驟挑選一個數字作為基準 (pivot), 如果可以使用一個額外的陣列 `newdata[]`, 把它數字分為兩群, 大於基準的數字放在右邊, 小於基準的數字放在左邊

例如: 將第一



{5, 2, 6, 4}



{9, 15}

– 如此就可以將問題分解成兩個較小的子問題

– 問題: ① 怎麼選基準? ② 怎麼分割? ③ 怎麼合併? ④ 怎麼知道它數字一個一個比較就可以; ⑤ 怎樣才能夠有**效率**地分成兩群?

快速排序法 (Quick Sort) – $O(n \log n)$

- 範例: 9, 5, 12, 19, 2, 6, 4, 15, 8, 3, 7
- 目標: 2, 3, 4, 5, 6, 7, 8, 9, 12, 15, 19
- 遞迴函式設計:

– 分治法 (Divide and Conquer): 「切割問題, 各個擊破」

– 每一步驟挑選一個數字作為基準 (pivot), 如果可以使用一個額外的陣列 `newdata[]`, 把它數字分為兩群, 大於基準的放左邊, 小於基準的放右邊

例如: 將第一

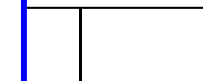


{5, 12, 19, 2, 6, 4, 15, 8, 3, 7}

如果可以使用一個額外的陣列 `newdata[]`, 把 `data[]` 陣列裡資料分成兩群就很容易完成:

```
i1 = start, i2 = end;
for (i=start+1; i<=end; i++) {
    if (data[i]<data[start])
        newdata[i1++] = data[i];
    else
        newdata[i2--] = data[i];
}
newdata[i1++] = data[start];
```

的放左邊



9, 15}

– 如此就可以

– 問題: ① 怎麼

一個一個比較就可以; ② 怎樣才能夠有 **效率** 地分成兩群?

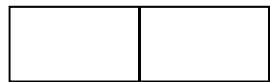
快速排序法 (Quick Sort) – $O(n \log n)$

- 範例: 9, 5, 12, 19, 2, 6, 4, 15, 8, 3, 7
- 目標: 2, 3, 4, 5, 6, 7, 8, 9, 12, 15, 19
- 遞迴函式設計:

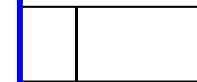
– 分治法 (Divide and Conquer): 「切割問題, 各個擊破」

– 每一步驟挑選一個數字作為基準 (pivot), 將陣列裡資料分成兩群, 大於基準的放右邊, 小於基準的放左邊

例如: 將第一



{5, 12, 19, 2, 6, 4, 15, 8, 3, 7}



{9, 15}

– 如此就可以將陣列分成兩群

– 問題: ① 怎麼挑選基準? ② 怎麼切割? ③ 怎麼合併? ④ 怎麼確保陣列裡資料的相對位置不變? ⑤ 怎麼確保陣列裡資料的相對位置不變? ⑥ 怎麼確保陣列裡資料的相對位置不變?

如果可以使用一個額外的陣列 `newdata[]`, 把 `data[]` 陣列裡資料分成兩群就很容易完成:

```
i1 = start, i2 = end;
for (i=start+1; i<=end; i++) {
    if (data[i]<data[start])
        newdata[i1++] = data[i];
    else
        newdata[i2--] = data[i];
}
newdata[i1++] = data[start];
```

不過接下來的程式我們要完成 **in-place** 的快速排序法, 不需要額外的陣列來暫存資料

④ 怎麼確保陣列裡資料的相對位置不變? ⑤ 怎麼確保陣列裡資料的相對位置不變? ⑥ 怎麼確保陣列裡資料的相對位置不變?

快速排序法 – $O(n \log n)$ 2/8

9 5 12 19 2 6 4 15 8 3 7



快速排序法 – $O(n \log n)$ 2/8

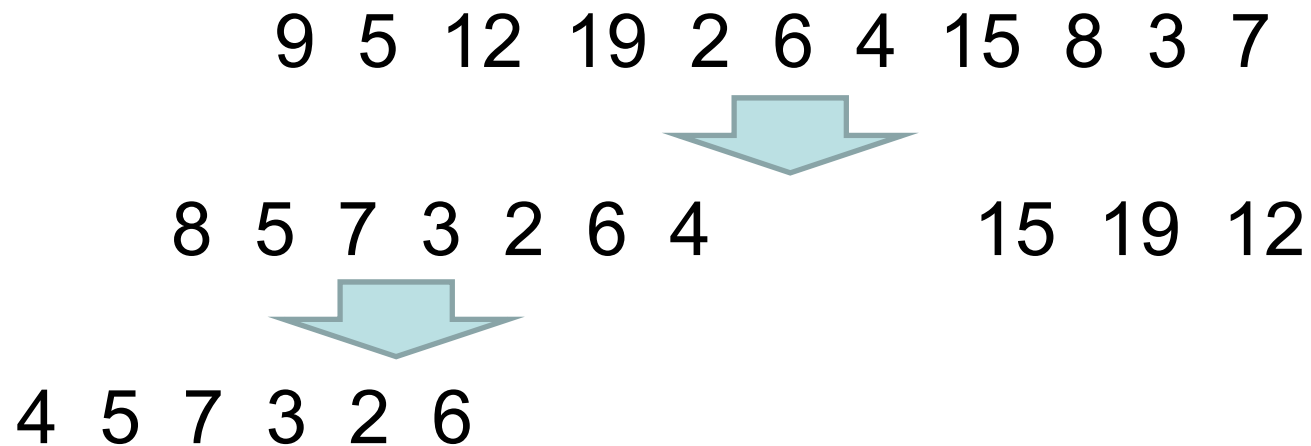
9 5 12 19 2 6 4 15 8 3 7



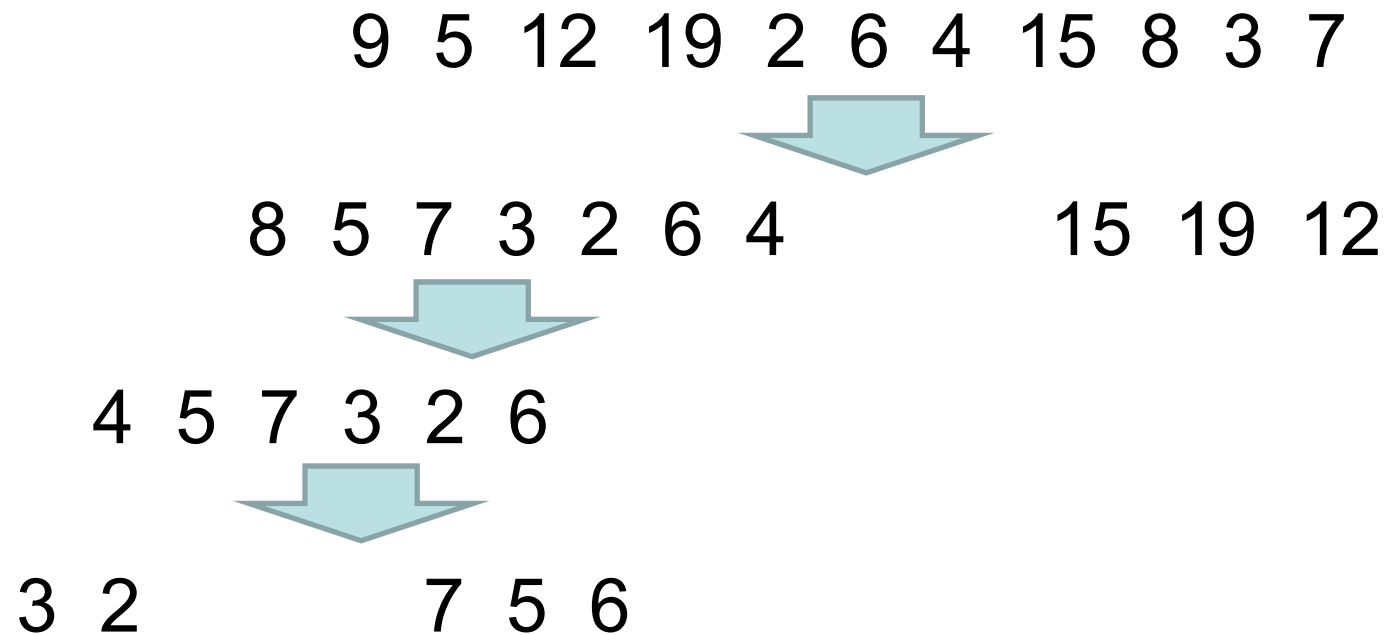
8 5 7 3 2 6 4 15 19 12



快速排序法 – $O(n \log n)$ 2/8



快速排序法 – $O(n \log n)$ 2/8



快速排序法 – $O(n \log n)$ 2/8

9 5 12 19 2 6 4 15 8 3 7



8 5 7 3 2 6 4 15 19 12



4 5 7 3 2 6



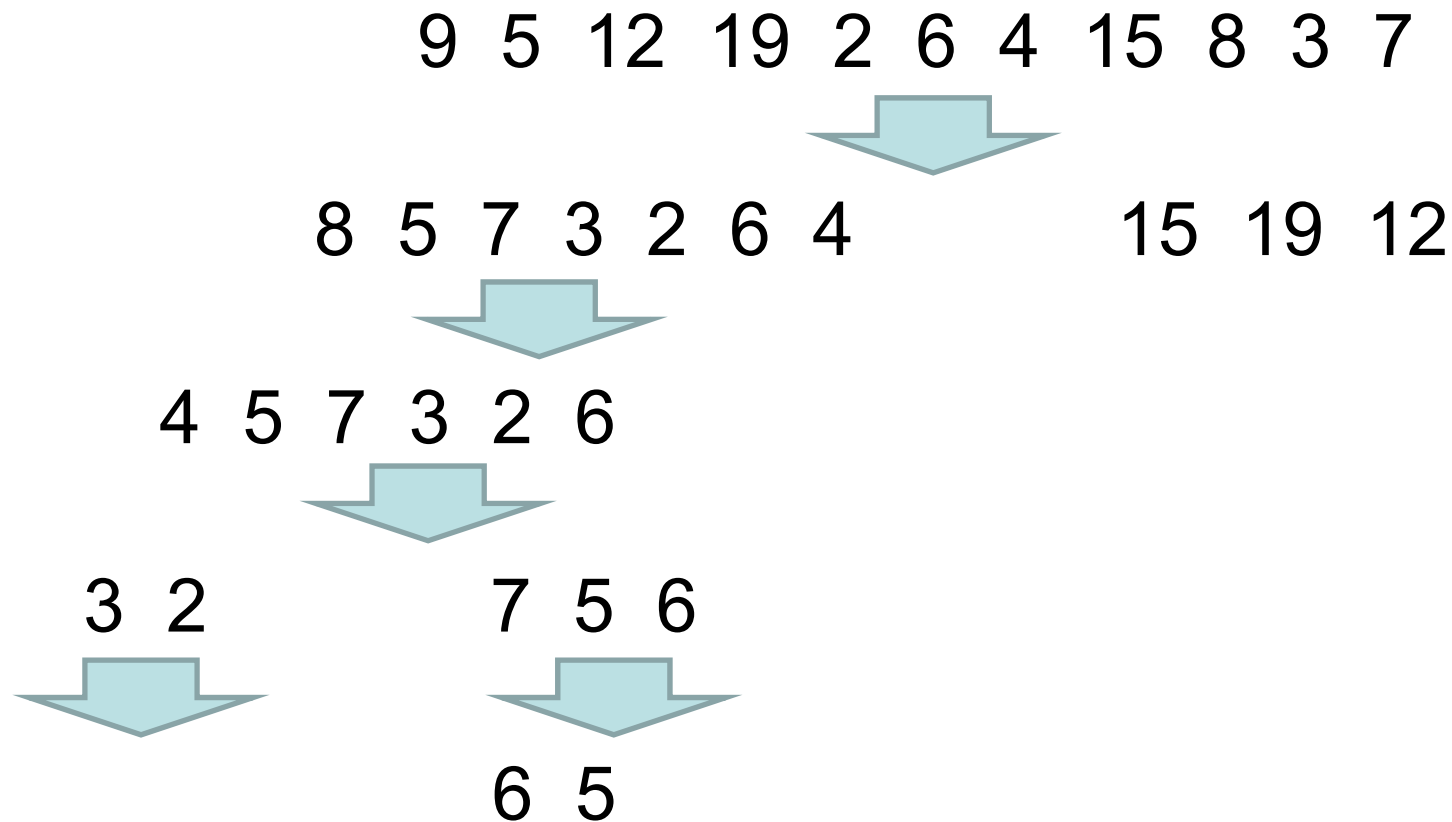
3 2

7 5 6



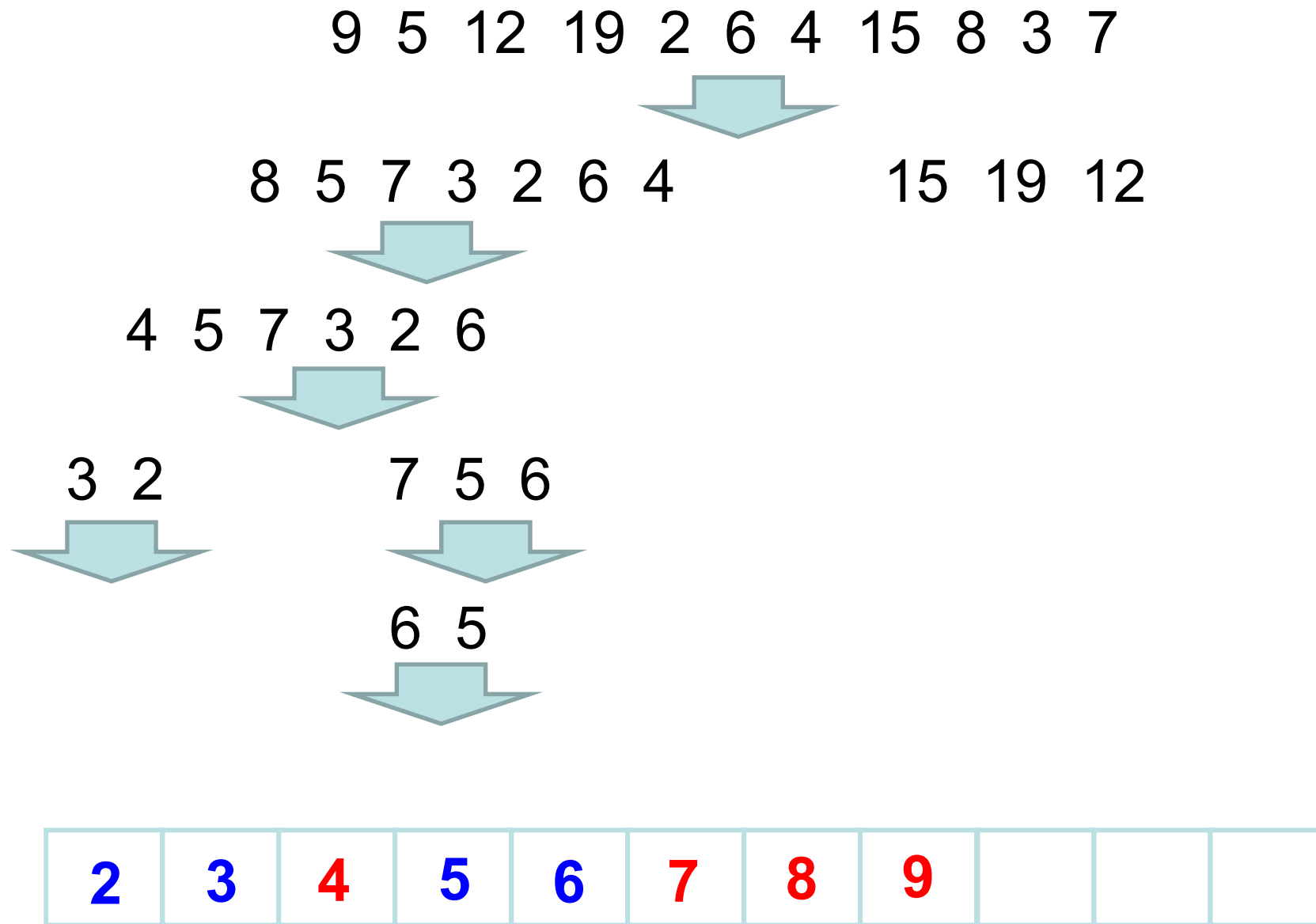
2	3	4				8	9			
---	---	---	--	--	--	---	---	--	--	--

快速排序法 – $O(n \log n)$ 2/8

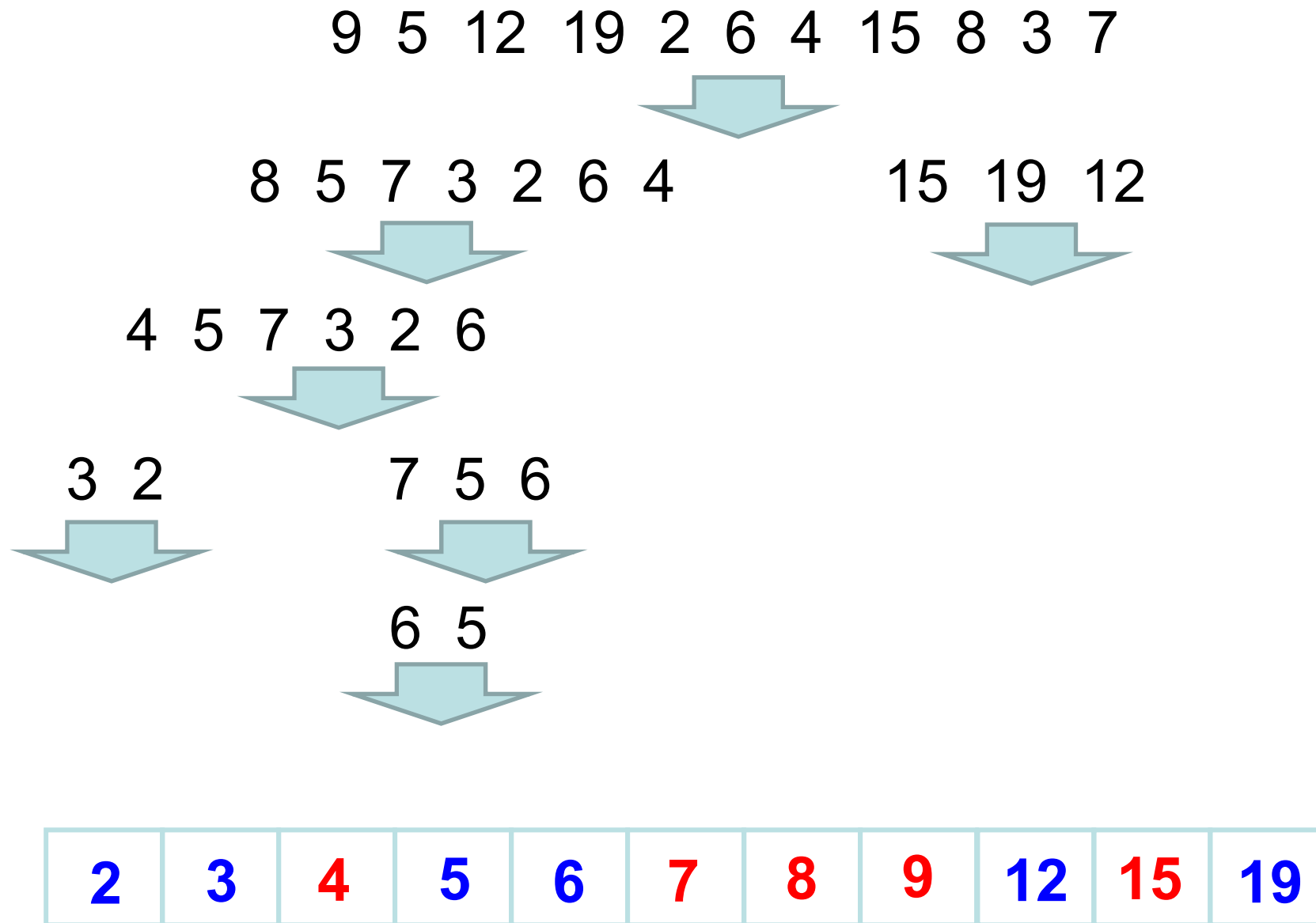


2	3	4			7	8	9			
---	---	---	--	--	---	---	---	--	--	--

快速排序法 – $O(n \log n)$ 2/8



快速排序法 – $O(n \log n)$ 2/8



快速排序法 – $O(n \log n)$ 2/8

9 5 12 19 2 6 4 15 8 3 7



8 5 7 3 2 6 4

15 19 12



4 5 7 3 2 6



3 2



7 5 6



6 5



```
void quicksort(int data[],
                int start, int end) {
    // terminating conditions
    int pivot = partition(data, start, end);
    quicksort(data, start, pivot-1);
    quicksort(data, pivot+1, end);
}
```

2	3	4	5	6	7	8	9	12	15	19
---	---	---	---	---	---	---	---	----	----	----

快速排序法 – $O(n \log n)$ 3/8

start → 9, 5, 12, 19, 2, 6, 4, 15, 8, 3, 7
front ↑ end
rear

快速排序法 – $O(n \log n)$ 3/8

start → 9, 5, 12, 19, 2, 6, 4, 15, 8, 3, 7
front ↑ end
rear

步驟 1. while front ≤ rear &&
data[rear] ≥ data[start]
rear--

9, 5, 12, 19, 2, 6, 4, 15, 8, 3, 7
front ↑ rear

快速排序法 – $O(n \log n)$ 3/8

start → 9, 5, 12, 19, 2, 6, 4, 15, 8, 3, 7
front ↑ end
rear

步驟 1. while front ≤ rear &&
data[rear] ≥ data[start]
rear--

9, 5, 12, 19, 2, 6, 4, 15, 8, 3, 7
front ↑ rear

步驟 2. while front ≤ rear &&
data[front] < data[start]
front++

9, 5, 12, 19, 2, 6, 4, 15, 8, 3, 7
front ↑ rear

快速排序法 – $O(n \log n)$ 3/8

start → 9, 5, 12, 19, 2, 6, 4, 15, 8, 3, 7
front ↑ end
rear

步驟 1. while front ≤ rear &&
data[rear] ≥ data[start]
rear--

9, 5, 12, 19, 2, 6, 4, 15, 8, 3, 7
front ↑ rear

步驟 2. while front ≤ rear &&
data[front] < data[start]
front++

9, 5, 12, 19, 2, 6, 4, 15, 8, 3, 7
front ↑ rear

步驟 3. if front < rear
交換 data[front++] 及 data[rear--]

9, 5, 7, 19, 2, 6, 4, 15, 8, 3, 12
front ↑ rear

快速排序法 – $O(n \log n)$ 3/8

start → 9, 5, 12, 19, 2, 6, 4, 15, 8, 3, 7 ← end
front rear

步驟 1. while front ≤ rear &&
data[rear] ≥ data[start]
rear--

9, 5, 12, 19, 2, 6, 4, 15, 8, 3, 7
front rear

步驟 2. while front ≤ rear &&
data[front] < data[start]
front++

9, 5, 12, 19, 2, 6, 4, 15, 8, 3, 7
front rear

步驟 3. if front < rear
交換 data[front++] 及 data[rear--]

9, 5, 7, 19, 2, 6, 4, 15, 8, 3, 12
front rear

重複步驟 1 至步驟 3 直到 front > rear
(i.e. while (front ≤ rear) { ... })

9, 5, 7, 3, 2, 6, 4, 15, 8, 19, 12

9, 5, 7, 3, 2, 6, 4, 8, 15, 19, 12

9, 5, 7, 3, 2, 6, 4, 8, 15, 19, 12

快速排序法 – $O(n \log n)$ 3/8

start → 9, 5, 12, 19, 2, 6, 4, 15, 8, 3, 7 ← end
front rear

步驟 1. while front ≤ rear &&
data[rear] ≥ data[start]
rear--

9, 5, 12, 19, 2, 6, 4, 15, 8, 3, 7
front rear

步驟 2. while front ≤ rear &&
data[front] < data[start]
front++

9, 5, 12, 19, 2, 6, 4, 15, 8, 3, 7
front rear

步驟 3. if front < rear
交換 data[front++] 及 data[rear--]

9, 5, 7, 19, 2, 6, 4, 15, 8, 3, 12
front rear

重複步驟 1 至步驟 3 直到 front > rear
(i.e. while (front ≤ rear) { ... })

9, 5, 7, 3, 2, 6, 4, 15, 8, 19, 12

if rear > start
交換 data[start] 及 data[rear]

9, 5, 7, 3, 2, 6, 4, 8, 15, 19, 12

9, 5, 7, 3, 2, 6, 4, 8, 15, 19, 12

以 data[start] 為分割點

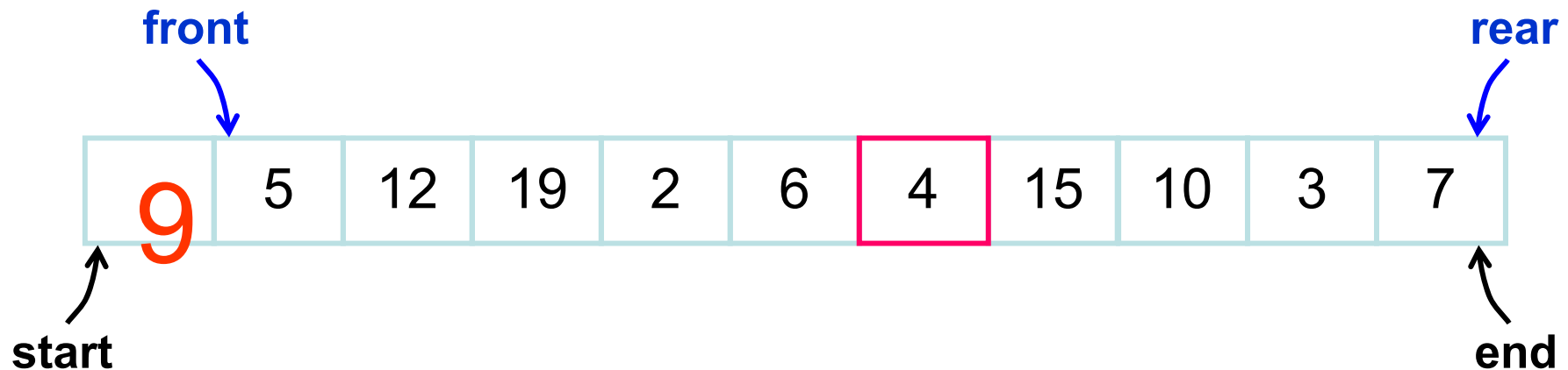
得到兩個長度比較短的排序問題

8, 5, 7, 3, 2, 6, 4, 9, 15, 19, 12

快速排序法 – $O(n \log n)$ 4/8

未排序資料 9, 5, 12, 19, 2, 6, 4, 15, 10, 3, 7

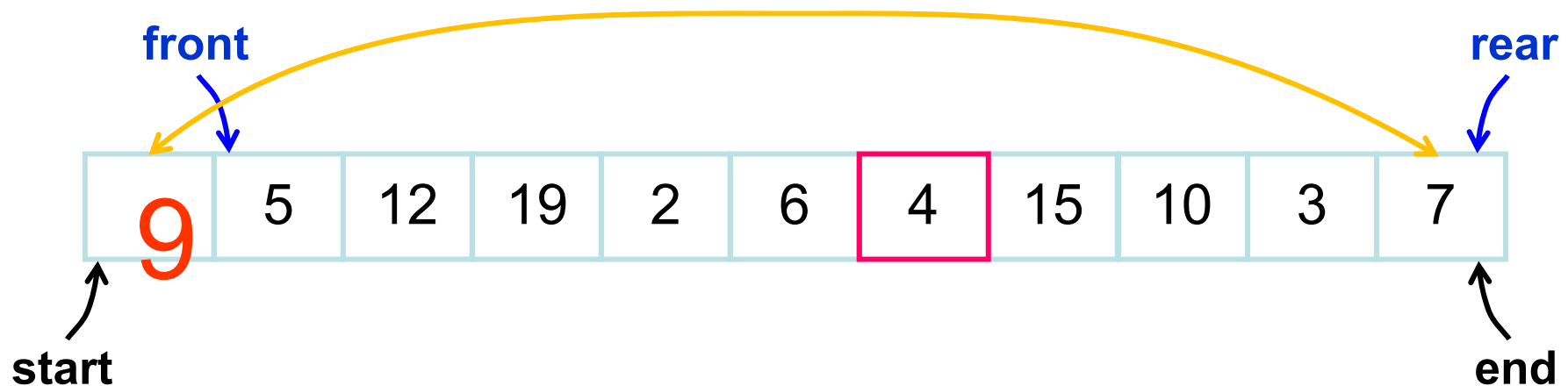
由小到大排序完成之資料 2, 3, 4, 5, 6, 7, 9, 10, 12, 15, 19



快速排序法 – $O(n \log n)$ 4/8

未排序資料 9, 5, 12, 19, 2, 6, 4, 15, 10, 3, 7

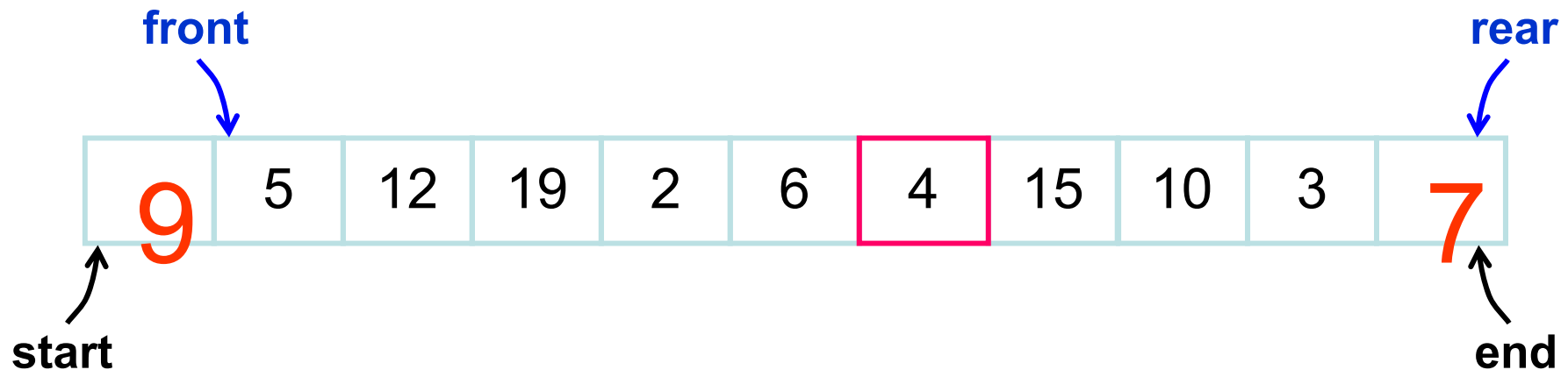
由小到大排序完成之資料 2, 3, 4, 5, 6, 7, 9, 10, 12, 15, 19



快速排序法 – $O(n \log n)$ 4/8

未排序資料 9, 5, 12, 19, 2, 6, 4, 15, 10, 3, 7

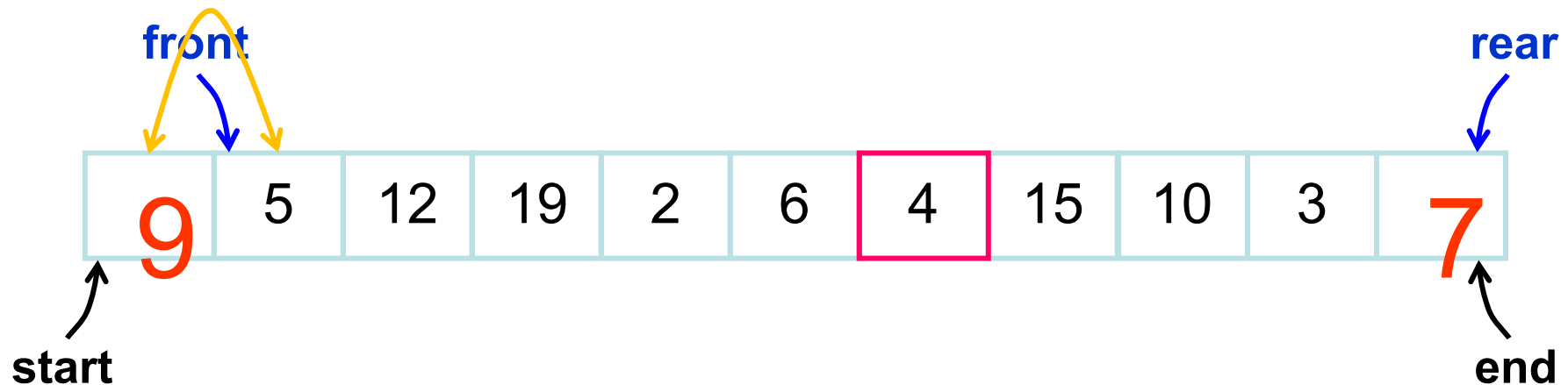
由小到大排序完成之資料 2, 3, 4, 5, 6, 7, 9, 10, 12, 15, 19



快速排序法 – $O(n \log n)$ 4/8

未排序資料 9, 5, 12, 19, 2, 6, 4, 15, 10, 3, 7

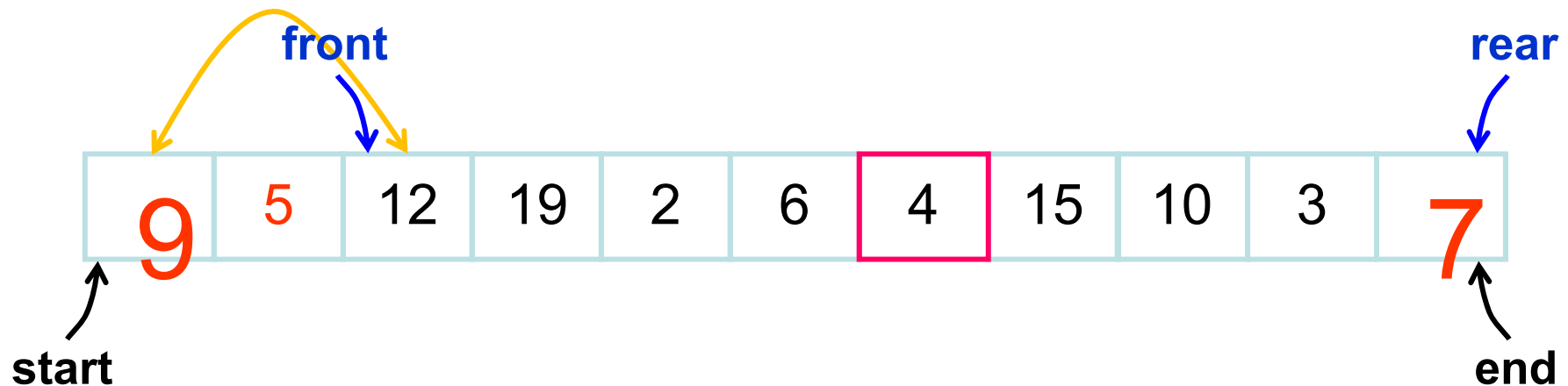
由小到大排序完成之資料 2, 3, 4, 5, 6, 7, 9, 10, 12, 15, 19



快速排序法 – $O(n \log n)$ 4/8

未排序資料 9, 5, 12, 19, 2, 6, 4, 15, 10, 3, 7

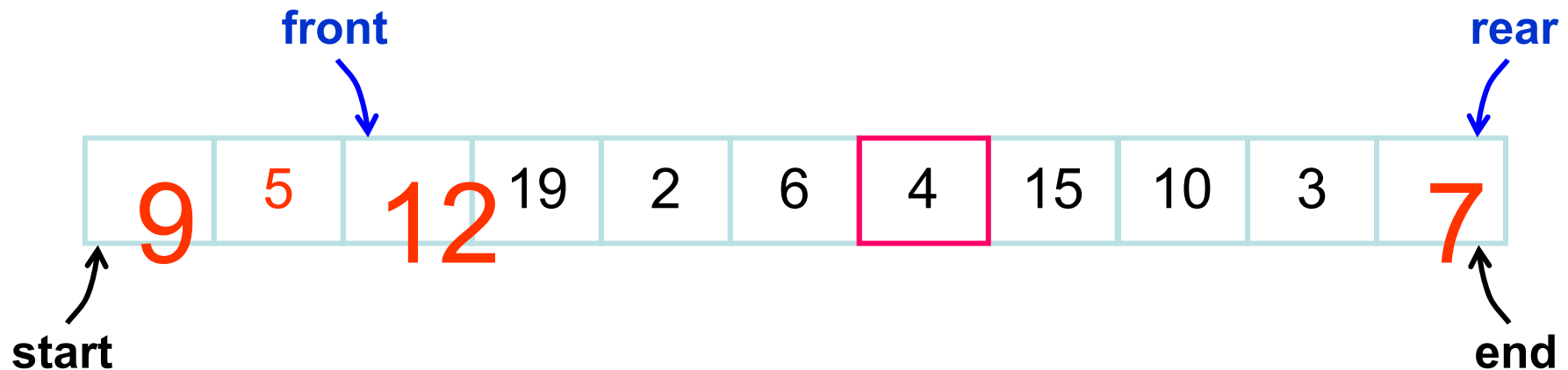
由小到大排序完成之資料 2, 3, 4, 5, 6, 7, 9, 10, 12, 15, 19



快速排序法 – $O(n \log n)$ 4/8

未排序資料 9, 5, 12, 19, 2, 6, 4, 15, 10, 3, 7

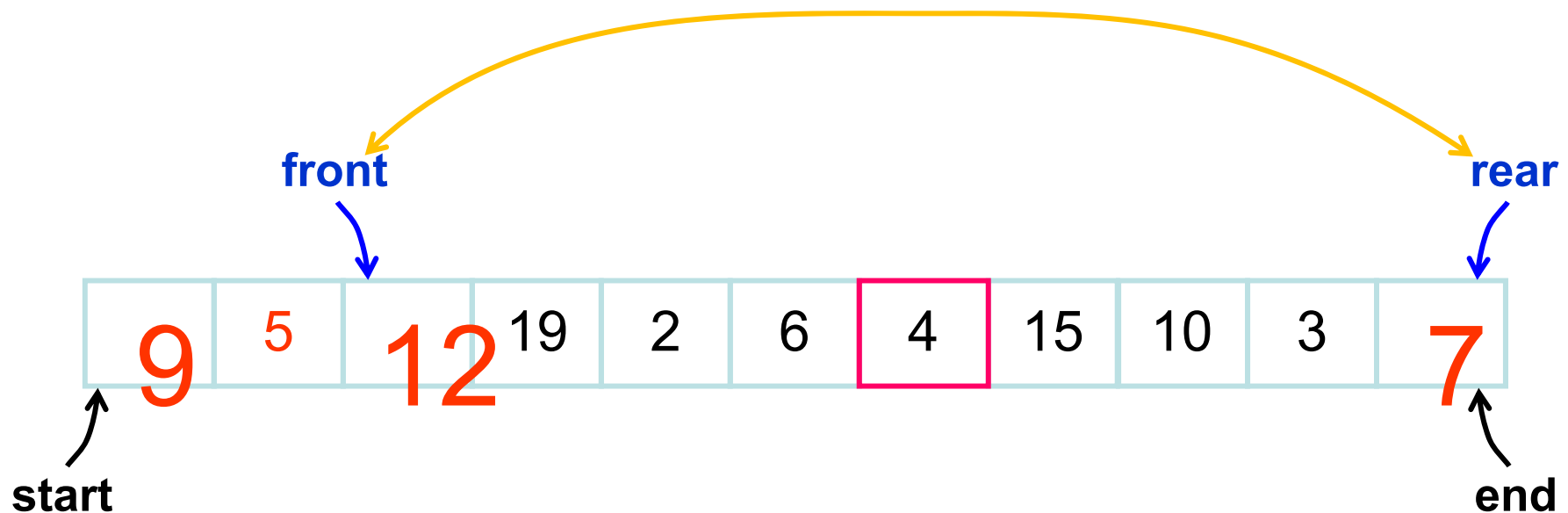
由小到大排序完成之資料 2, 3, 4, 5, 6, 7, 9, 10, 12, 15, 19



快速排序法 – $O(n \log n)$ 4/8

未排序資料 9, 5, 12, 19, 2, 6, 4, 15, 10, 3, 7

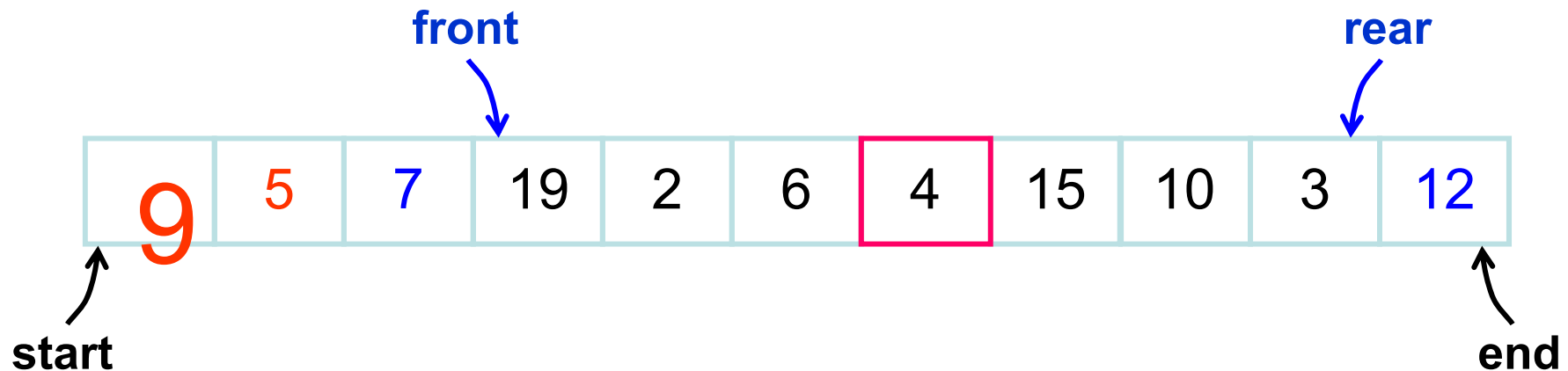
由小到大排序完成之資料 2, 3, 4, 5, 6, 7, 9, 10, 12, 15, 19



快速排序法 – $O(n \log n)$ 4/8

未排序資料 9, 5, 12, 19, 2, 6, 4, 15, 10, 3, 7

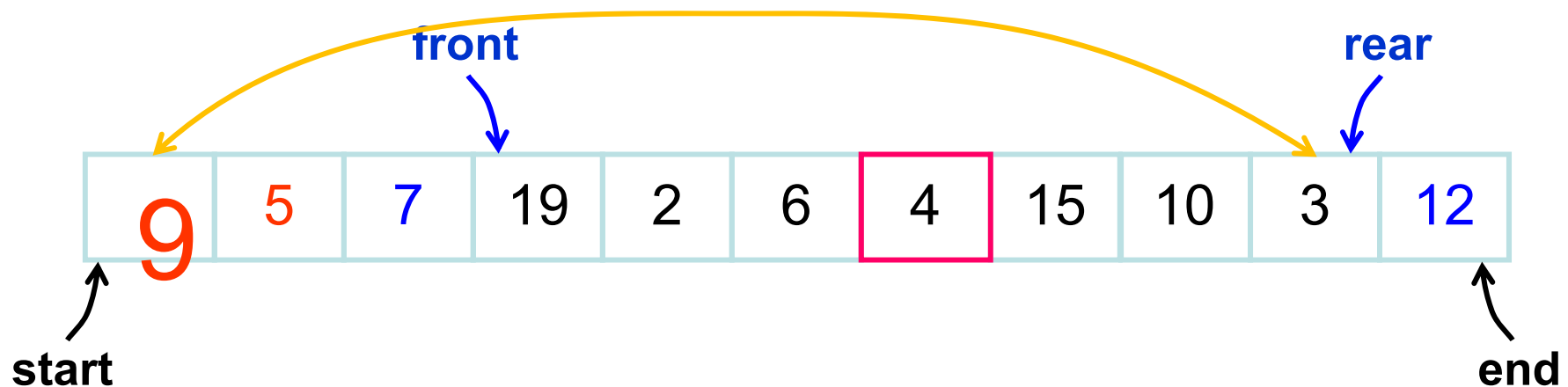
由小到大排序完成之資料 2, 3, 4, 5, 6, 7, 9, 10, 12, 15, 19



快速排序法 – $O(n \log n)$ 4/8

未排序資料 9, 5, 12, 19, 2, 6, 4, 15, 10, 3, 7

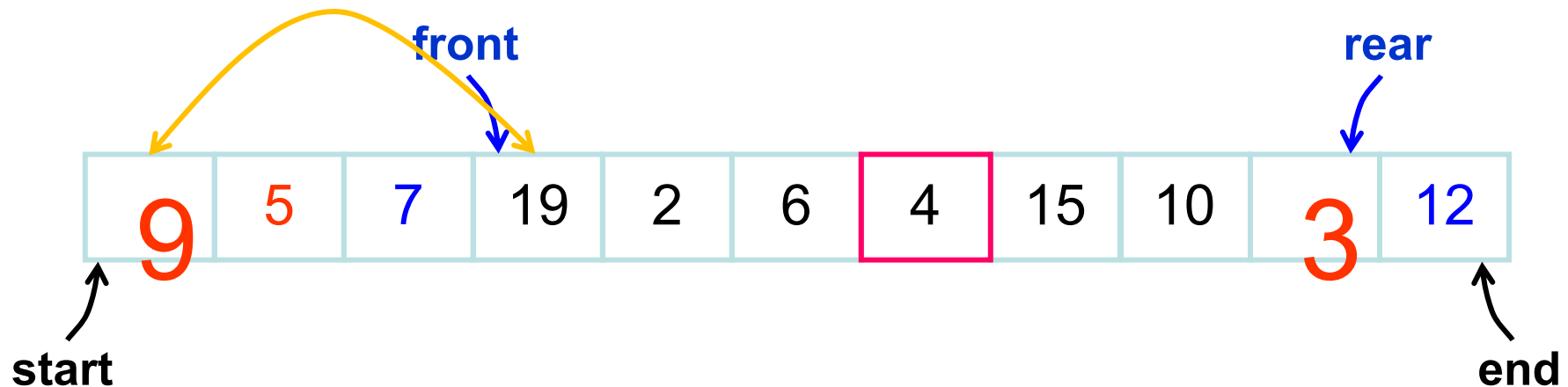
由小到大排序完成之資料 2, 3, 4, 5, 6, 7, 9, 10, 12, 15, 19



快速排序法 – $O(n \log n)$ 4/8

未排序資料 9, 5, 12, 19, 2, 6, 4, 15, 10, 3, 7

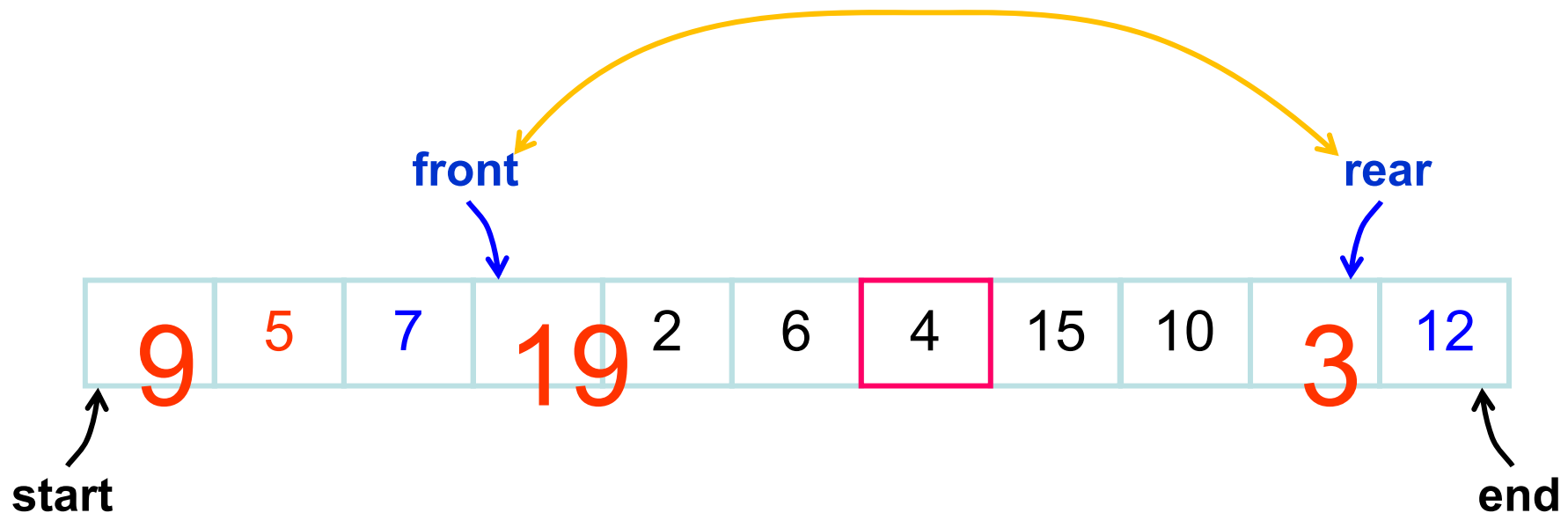
由小到大排序完成之資料 2, 3, 4, 5, 6, 7, 9, 10, 12, 15, 19



快速排序法 – $O(n \log n)$ 4/8

未排序資料 9, 5, 12, 19, 2, 6, 4, 15, 10, 3, 7

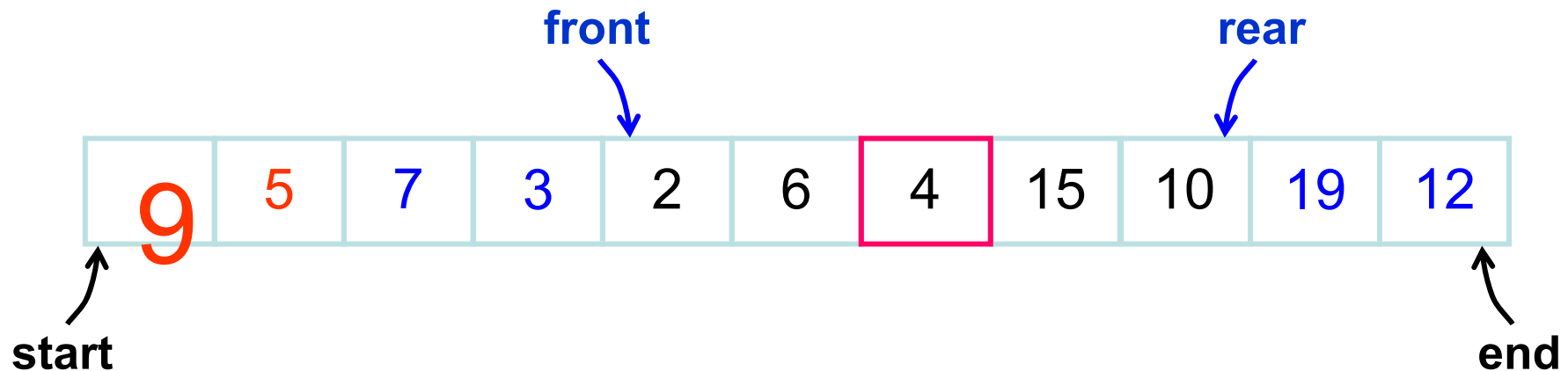
由小到大排序完成之資料 2, 3, 4, 5, 6, 7, 9, 10, 12, 15, 19



快速排序法 – $O(n \log n)$ 4/8

未排序資料 9, 5, 12, 19, 2, 6, 4, 15, 10, 3, 7

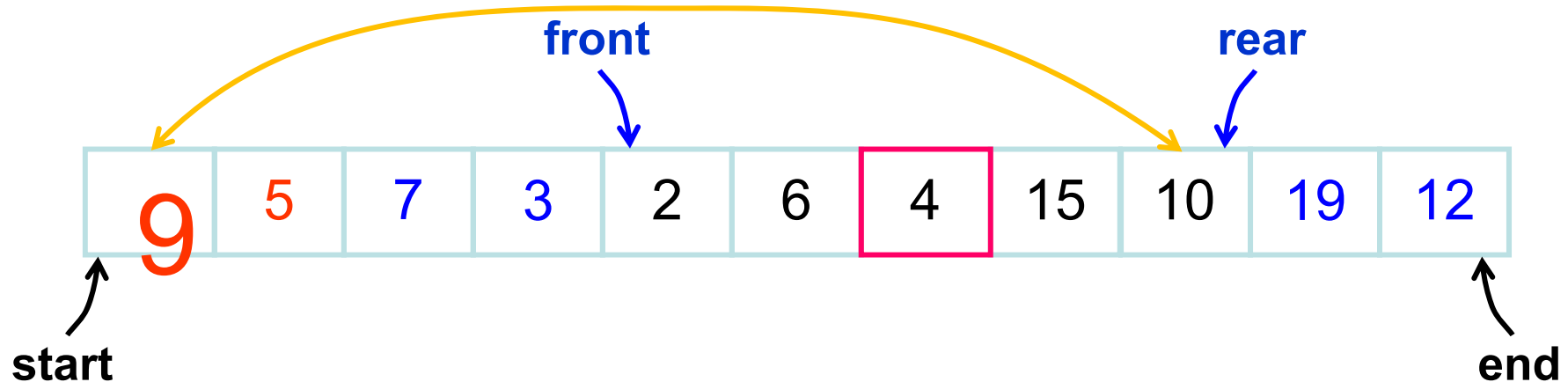
由小到大排序完成之資料 2, 3, 4, 5, 6, 7, 9, 10, 12, 15, 19



快速排序法 – $O(n \log n)$ 4/8

未排序資料 9, 5, 12, 19, 2, 6, 4, 15, 10, 3, 7

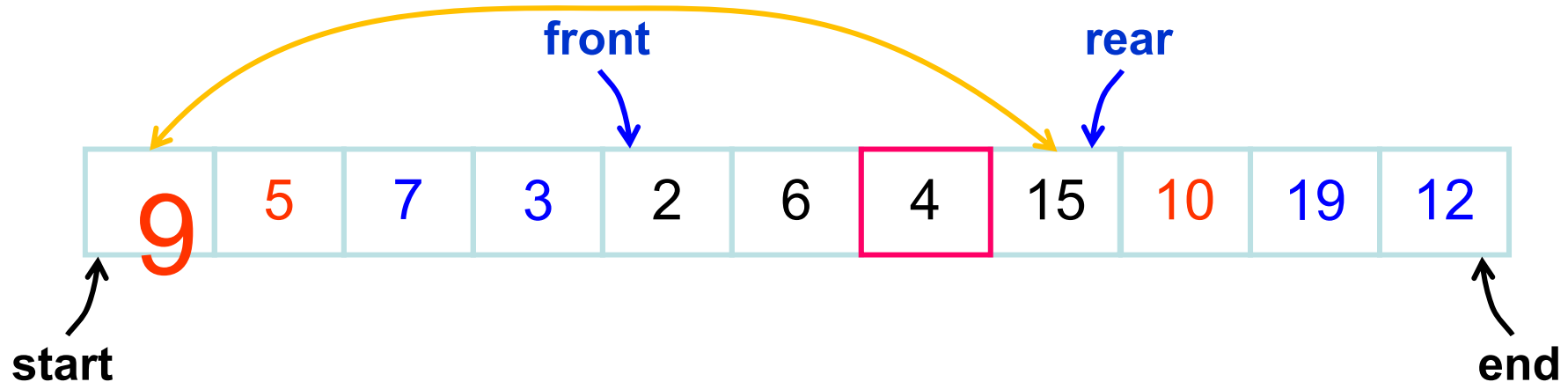
由小到大排序完成之資料 2, 3, 4, 5, 6, 7, 9, 10, 12, 15, 19



快速排序法 – $O(n \log n)$ 4/8

未排序資料 9, 5, 12, 19, 2, 6, 4, 15, 10, 3, 7

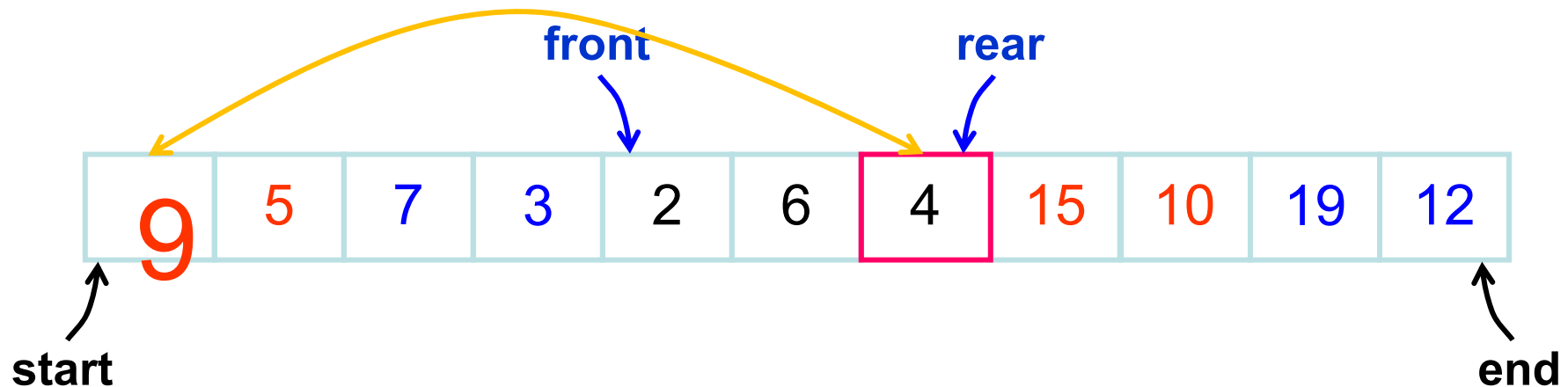
由小到大排序完成之資料 2, 3, 4, 5, 6, 7, 9, 10, 12, 15, 19



快速排序法 – $O(n \log n)$ 4/8

未排序資料 9, 5, 12, 19, 2, 6, 4, 15, 10, 3, 7

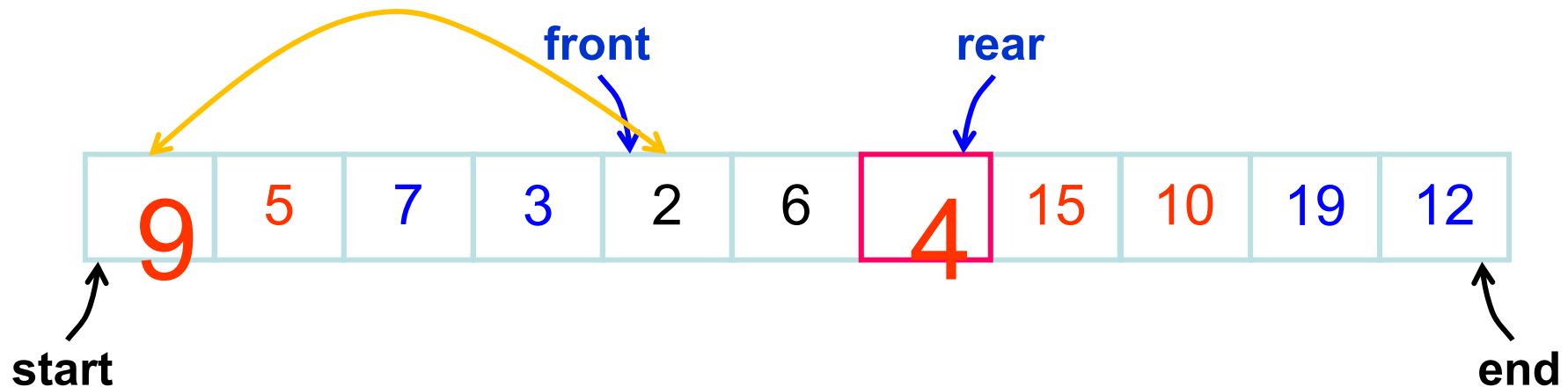
由小到大排序完成之資料 2, 3, 4, 5, 6, 7, 9, 10, 12, 15, 19



快速排序法 – $O(n \log n)$ 4/8

未排序資料 9, 5, 12, 19, 2, 6, 4, 15, 10, 3, 7

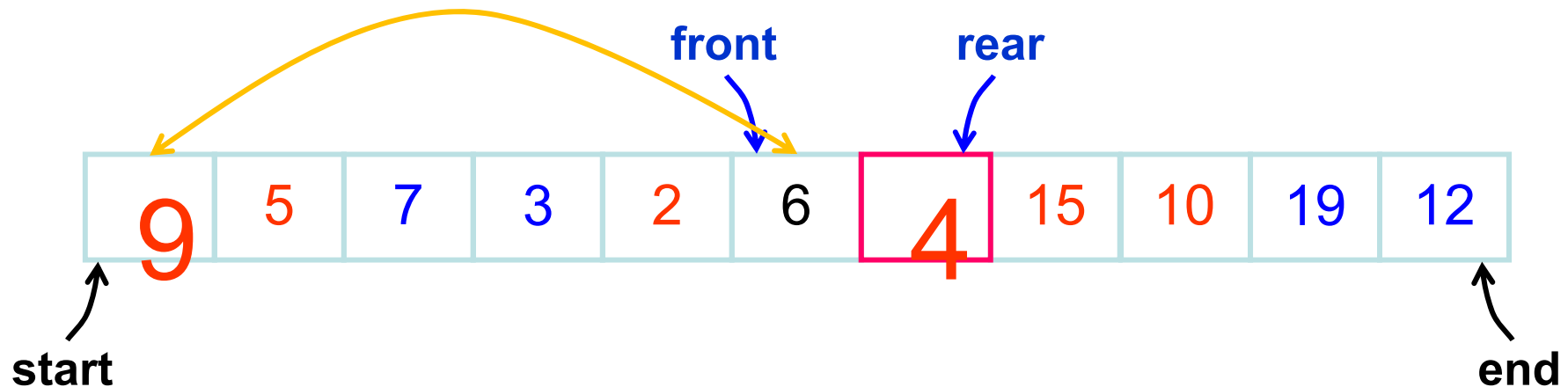
由小到大排序完成之資料 2, 3, 4, 5, 6, 7, 9, 10, 12, 15, 19



快速排序法 – $O(n \log n)$ 4/8

未排序資料 9, 5, 12, 19, 2, 6, 4, 15, 10, 3, 7

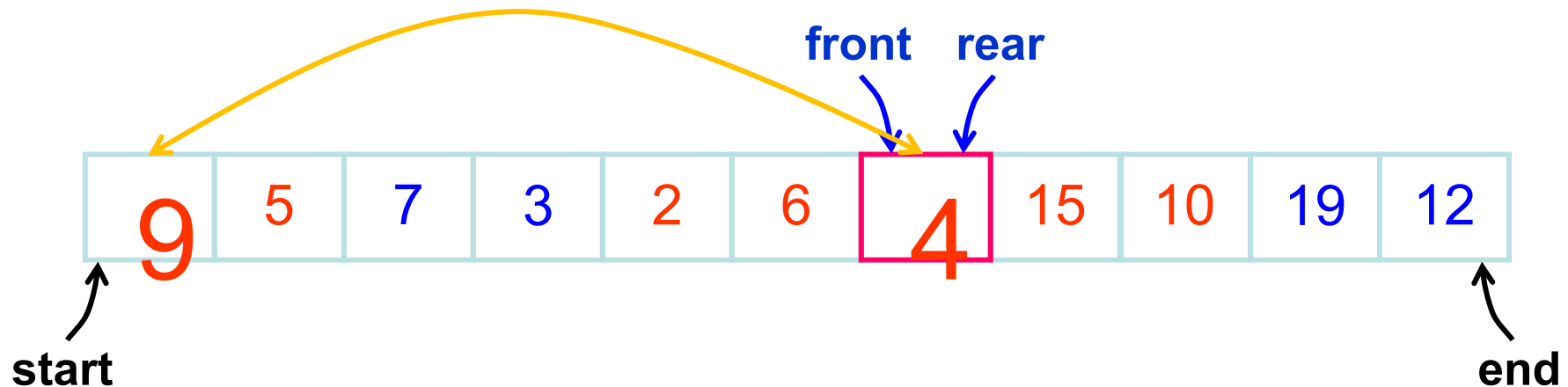
由小到大排序完成之資料 2, 3, 4, 5, 6, 7, 9, 10, 12, 15, 19



快速排序法 – $O(n \log n)$ 4/8

未排序資料 9, 5, 12, 19, 2, 6, 4, 15, 10, 3, 7

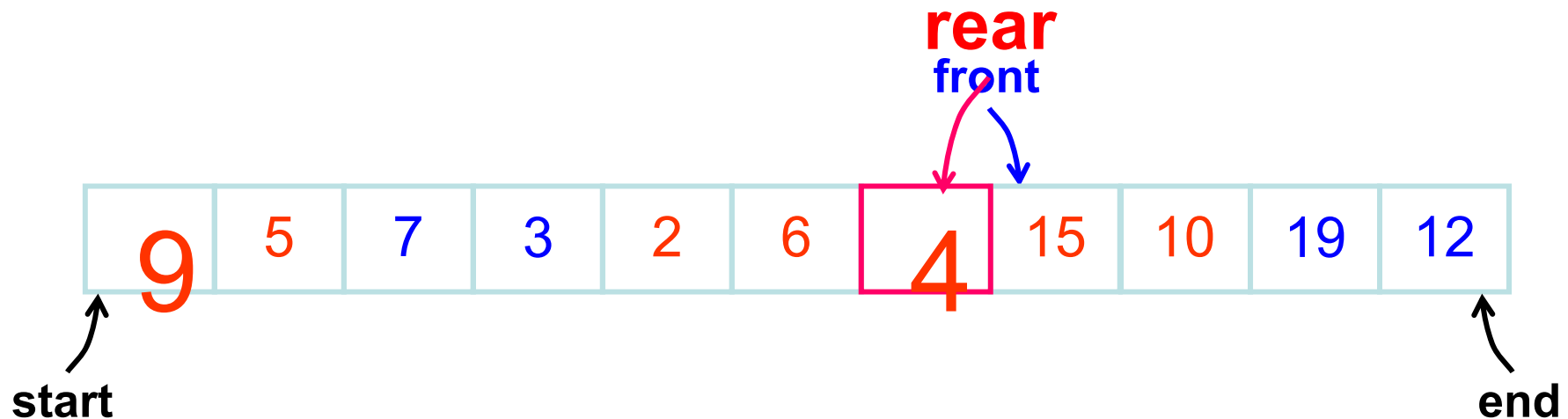
由小到大排序完成之資料 2, 3, 4, 5, 6, 7, 9, 10, 12, 15, 19



快速排序法 – $O(n \log n)$ 4/8

未排序資料 9, 5, 12, 19, 2, 6, 4, 15, 10, 3, 7

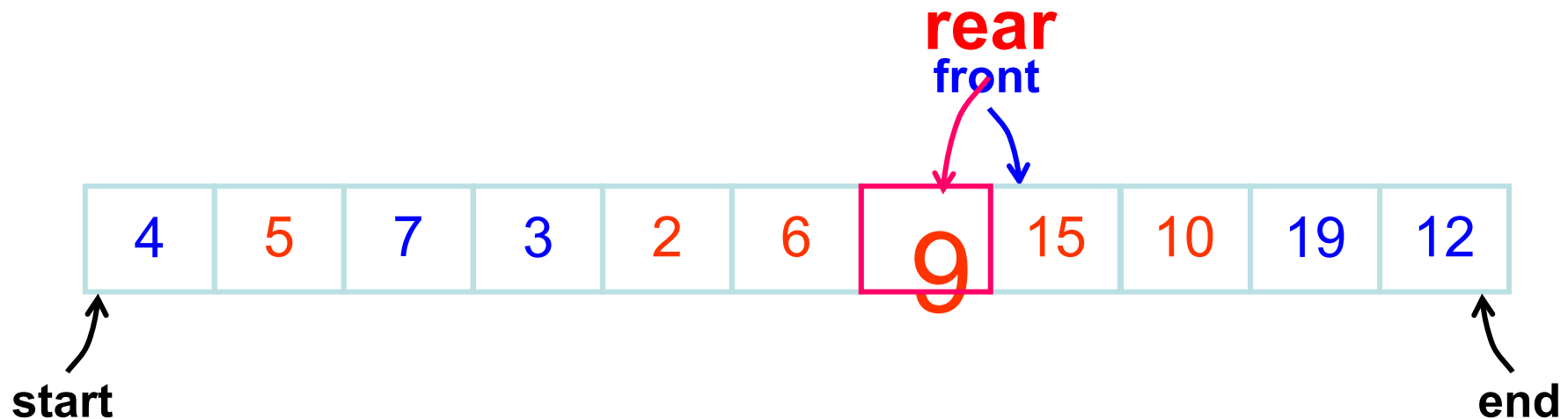
由小到大排序完成之資料 2, 3, 4, 5, 6, 7, 9, 10, 12, 15, 19



快速排序法 – $O(n \log n)$ 4/8

未排序資料 9, 5, 12, 19, 2, 6, 4, 15, 10, 3, 7

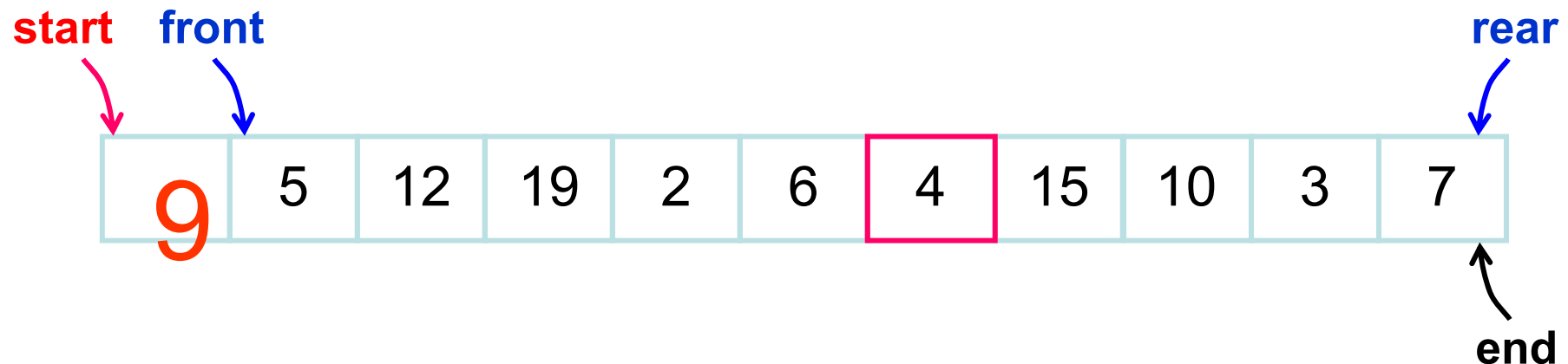
由小到大排序完成之資料 2, 3, 4, 5, 6, 7, 9, 10, 12, 15, 19



快速排序法 – $O(n \log n)$ 5/8

未排序資料 9, 5, 12, 19, 2, 6, 4, 15, 10, 3, 7

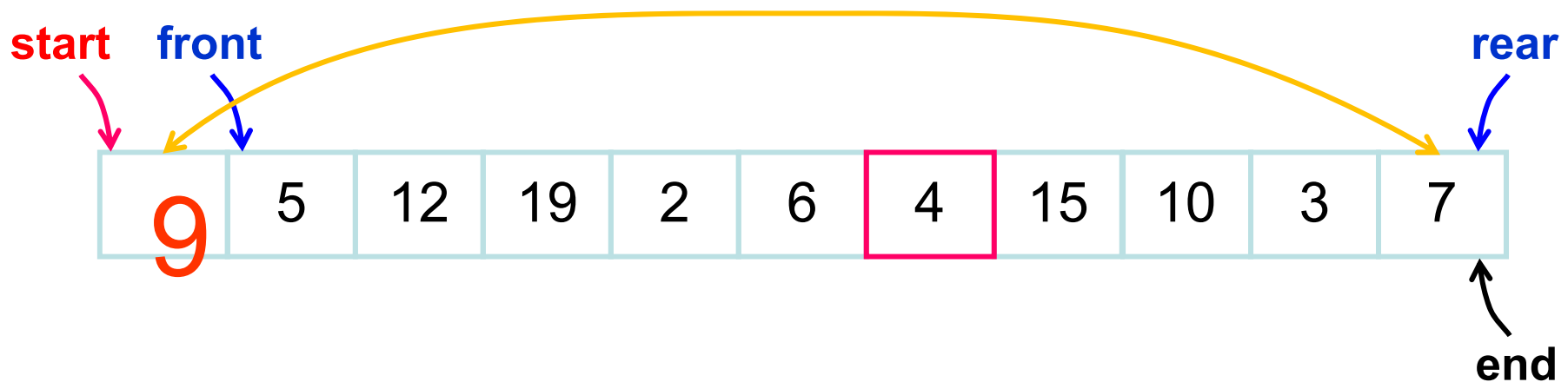
由小到大排序完成之資料 2, 3, 4, 5, 6, 7, 9, 10, 12, 15, 19



快速排序法 – $O(n \log n)$ 5/8

未排序資料 9, 5, 12, 19, 2, 6, 4, 15, 10, 3, 7

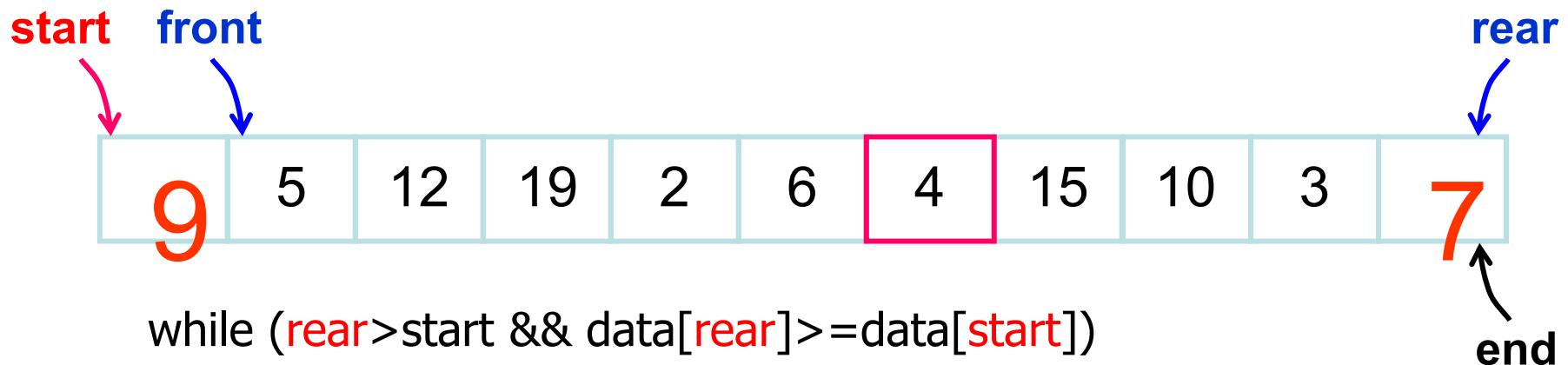
由小到大排序完成之資料 2, 3, 4, 5, 6, 7, 9, 10, 12, 15, 19



快速排序法 – $O(n \log n)$ 5/8

未排序資料 9, 5, 12, 19, 2, 6, 4, 15, 10, 3, 7

由小到大排序完成之資料 2, 3, 4, 5, 6, 7, 9, 10, 12, 15, 19



```
while (rear > start && data[rear] >= data[start])  
    rear--;
```

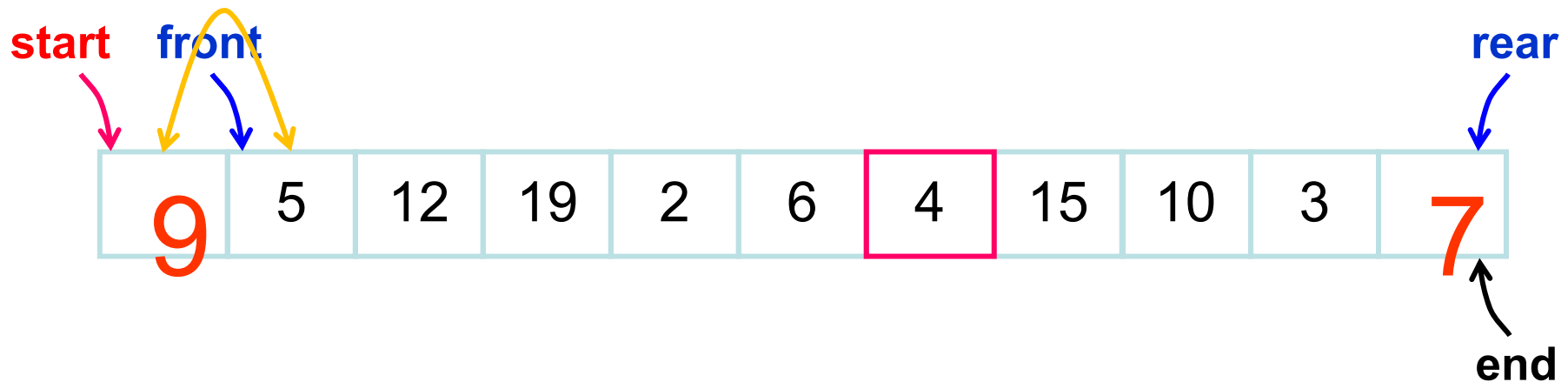
★rear 最多只會到達 start

★rear 停下來如果大於 start, data[rear] 一定小於 data[start]

快速排序法 – $O(n \log n)$ 5/8

未排序資料 9, 5, 12, 19, 2, 6, 4, 15, 10, 3, 7

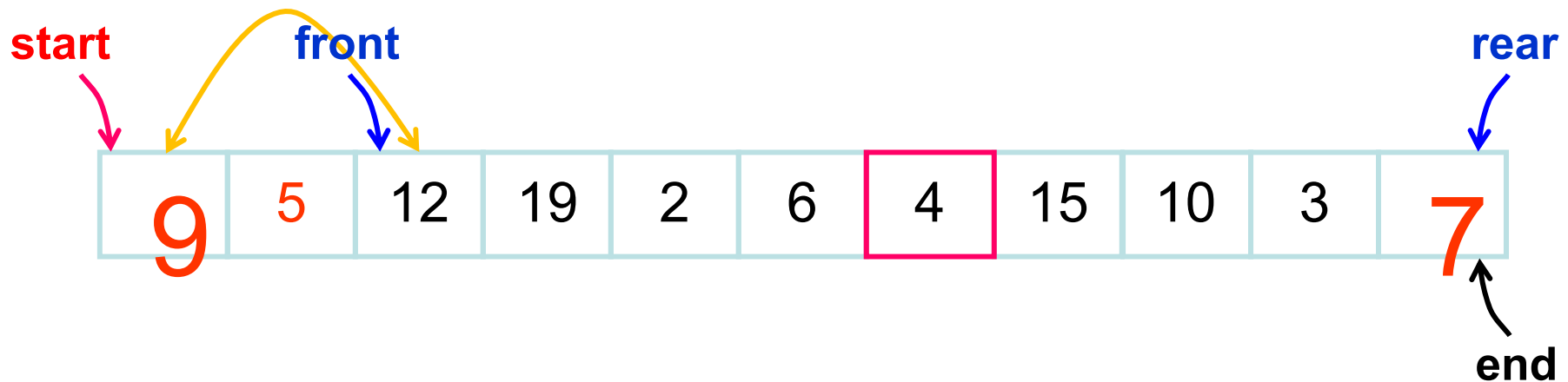
由小到大排序完成之資料 2, 3, 4, 5, 6, 7, 9, 10, 12, 15, 19



快速排序法 – $O(n \log n)$ 5/8

未排序資料 9, 5, 12, 19, 2, 6, 4, 15, 10, 3, 7

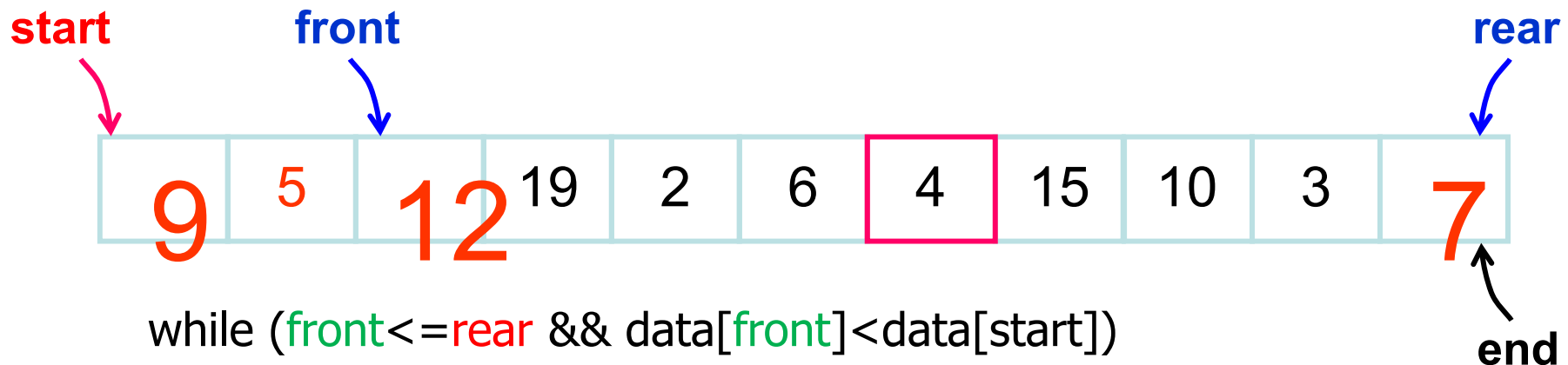
由小到大排序完成之資料 2, 3, 4, 5, 6, 7, 9, 10, 12, 15, 19



快速排序法 – $O(n \log n)$ 5/8

未排序資料 9, 5, 12, 19, 2, 6, 4, 15, 10, 3, 7

由小到大排序完成之資料 2, 3, 4, 5, 6, 7, 9, 10, 12, 15, 19



```
while (front <= rear && data[front] < data[start])  
    front++;
```

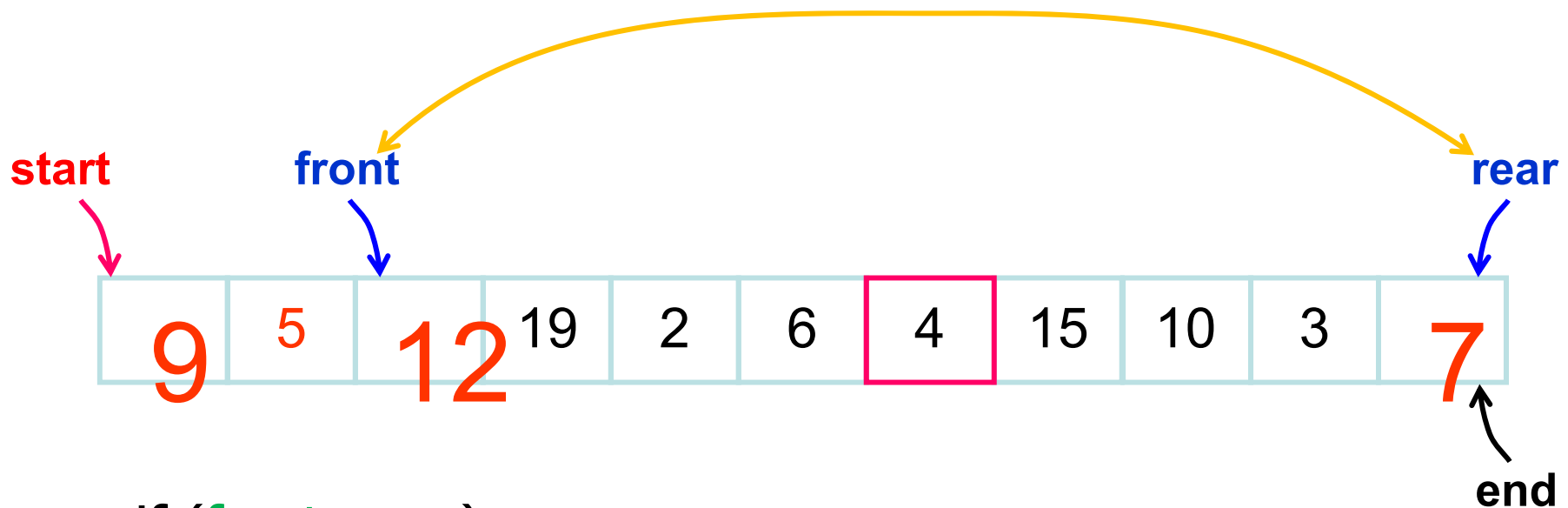
★front 最多會到達 end+1

★front 停下來如果小於等於 end, data[front] 一定大於等於 data[start]

快速排序法 – $O(n \log n)$ 5/8

未排序資料 9, 5, 12, 19, 2, 6, 4, 15, 10, 3, 7

由小到大排序完成之資料 2, 3, 4, 5, 6, 7, 9, 10, 12, 15, 19

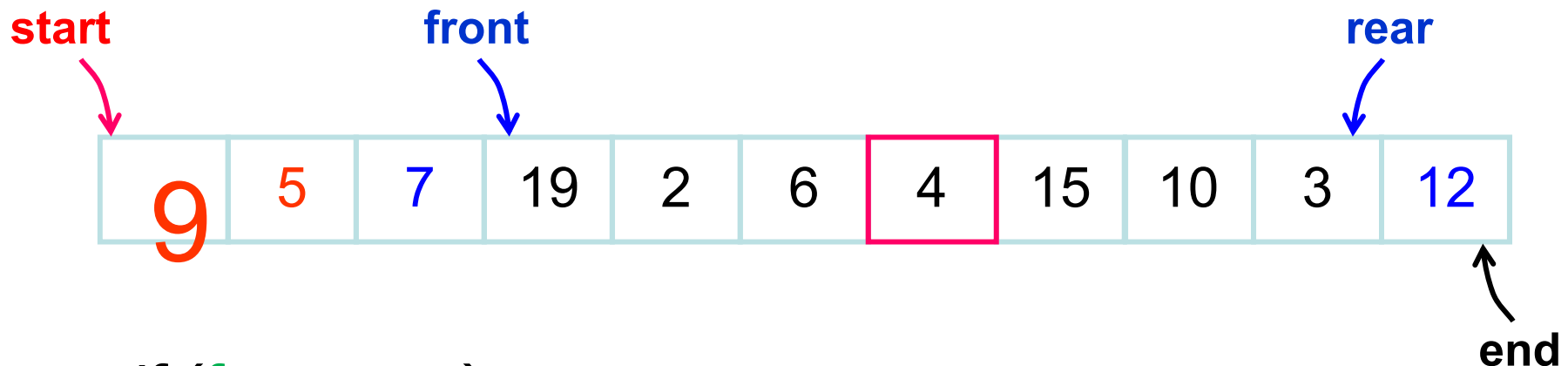


```
if (front < rear)
    swap(&data[front++], &data[rear--]);
```

快速排序法 – $O(n \log n)$ 5/8

未排序資料 9, 5, 12, 19, 2, 6, 4, 15, 10, 3, 7

由小到大排序完成之資料 2, 3, 4, 5, 6, 7, 9, 10, 12, 15, 19

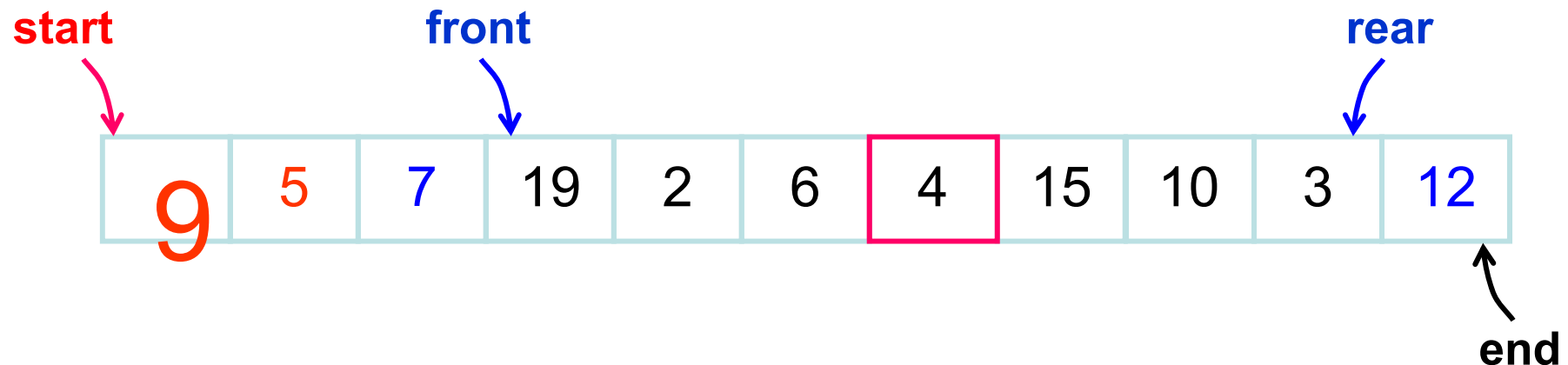


```
if (front < rear)
    swap(&data[front++], &data[rear--]);
```

快速排序法 – $O(n \log n)$ 5/8

未排序資料 9, 5, 12, 19, 2, 6, 4, 15, 10, 3, 7

由小到大排序完成之資料 2, 3, 4, 5, 6, 7, 9, 10, 12, 15, 19

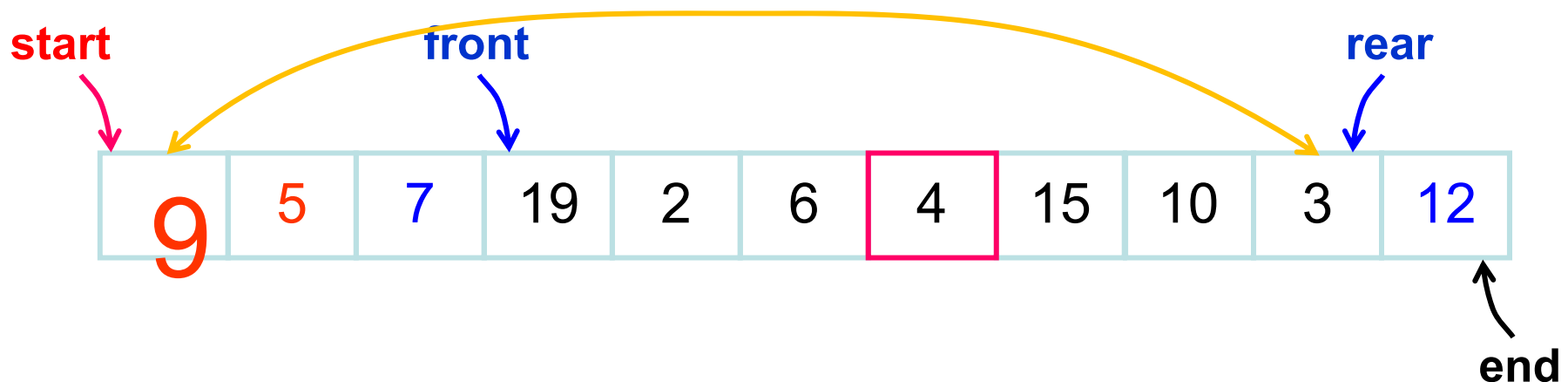


while (front ≤ rear) { ... }

快速排序法 – $O(n \log n)$ 5/8

未排序資料 9, 5, 12, 19, 2, 6, 4, 15, 10, 3, 7

由小到大排序完成之資料 2, 3, 4, 5, 6, 7, 9, 10, 12, 15, 19

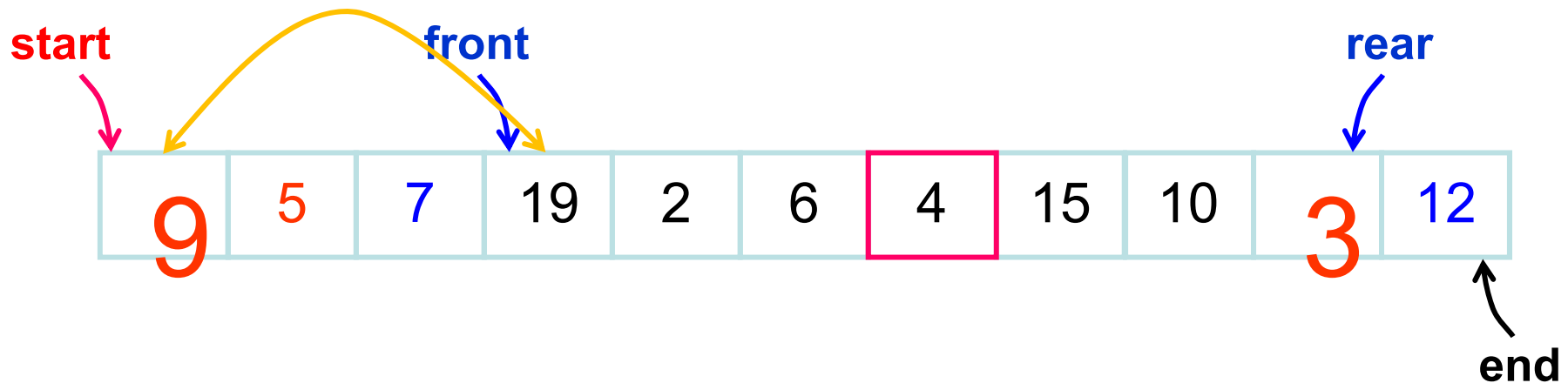


while (front ≤ rear) { ... }

快速排序法 – $O(n \log n)$ 5/8

未排序資料 9, 5, 12, 19, 2, 6, 4, 15, 10, 3, 7

由小到大排序完成之資料 2, 3, 4, 5, 6, 7, 9, 10, 12, 15, 19

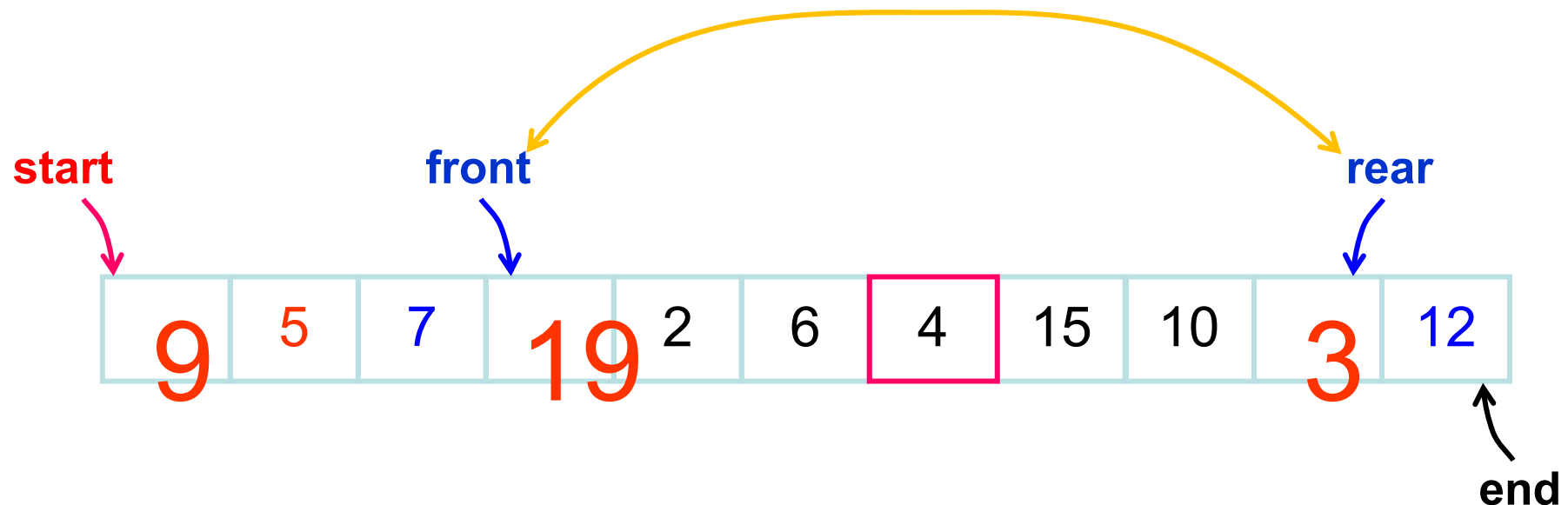


while (front <= rear) { ... }

快速排序法 – $O(n \log n)$ 5/8

未排序資料 9, 5, 12, 19, 2, 6, 4, 15, 10, 3, 7

由小到大排序完成之資料 2, 3, 4, 5, 6, 7, 9, 10, 12, 15, 19

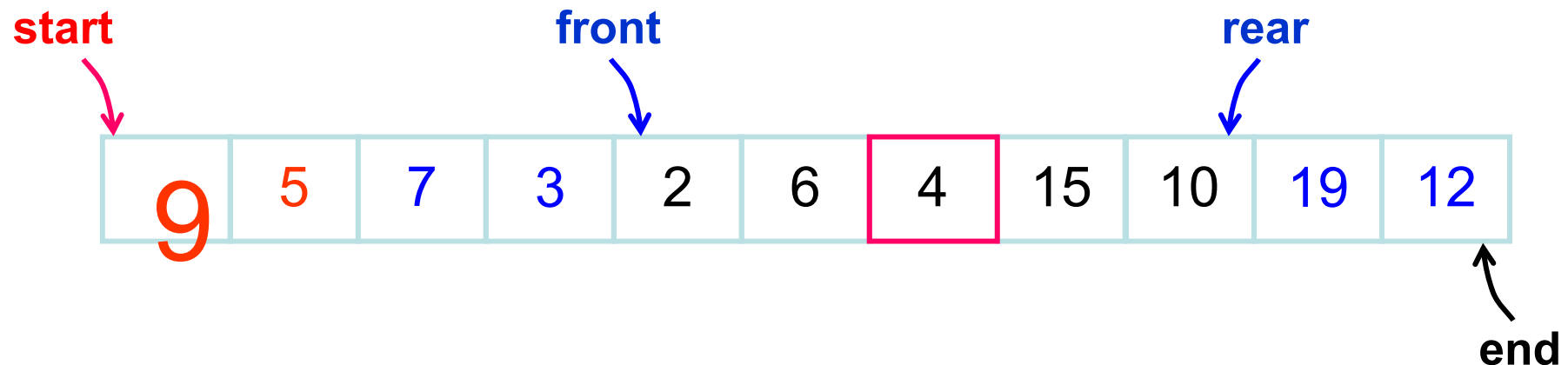


while (front ≤ rear) { ... }

快速排序法 – $O(n \log n)$ 5/8

未排序資料 9, 5, 12, 19, 2, 6, 4, 15, 10, 3, 7

由小到大排序完成之資料 2, 3, 4, 5, 6, 7, 9, 10, 12, 15, 19

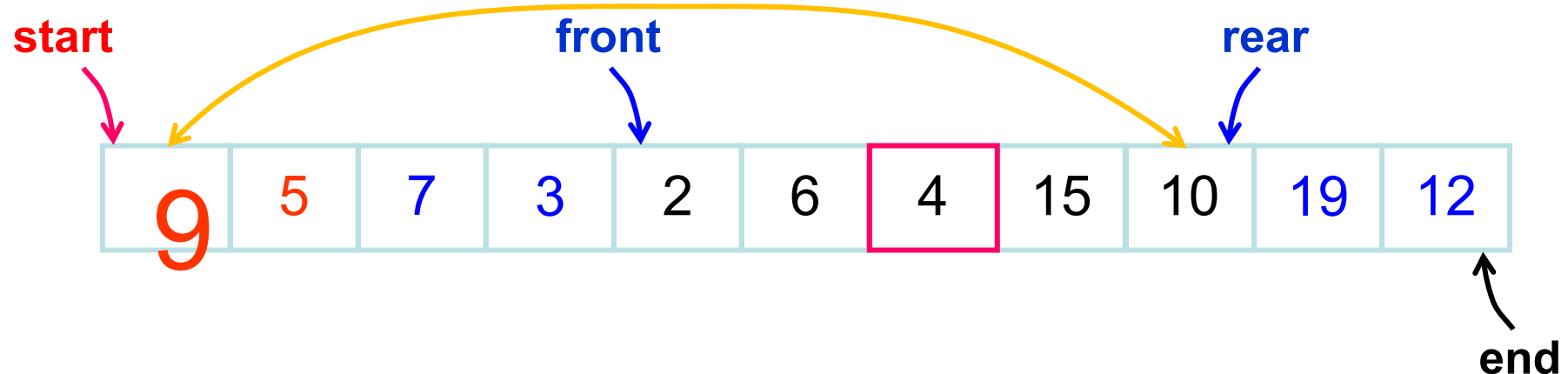


while (front ≤ rear) { ... }

快速排序法 – $O(n \log n)$ 5/8

未排序資料 9, 5, 12, 19, 2, 6, 4, 15, 10, 3, 7

由小到大排序完成之資料 2, 3, 4, 5, 6, 7, 9, 10, 12, 15, 19

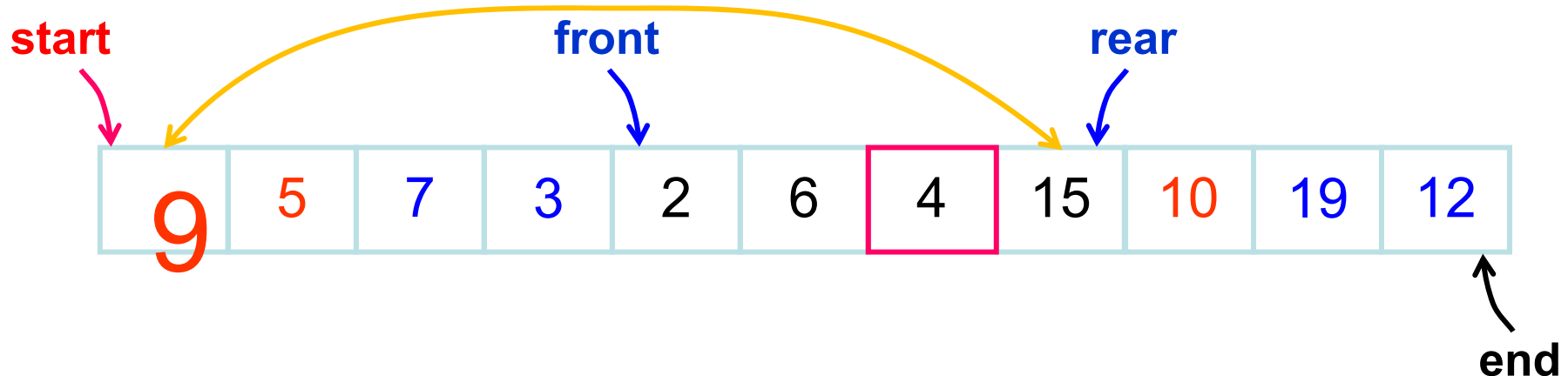


while (front ≤ rear) { ... }

快速排序法 – $O(n \log n)$ 5/8

未排序資料 9, 5, 12, 19, 2, 6, 4, 15, 10, 3, 7

由小到大排序完成之資料 2, 3, 4, 5, 6, 7, 9, 10, 12, 15, 19

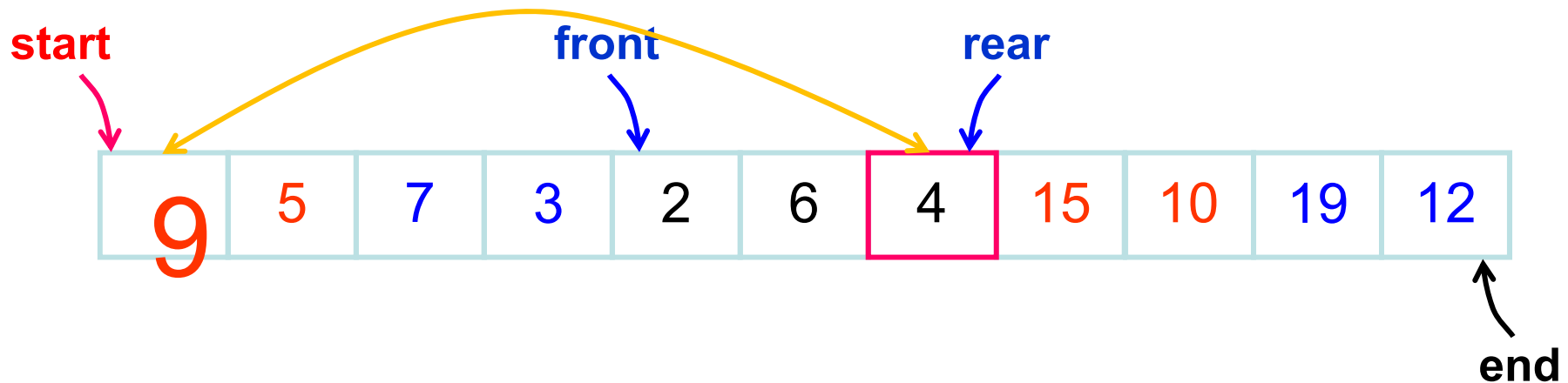


while (front ≤ rear) { ... }

快速排序法 – $O(n \log n)$ 5/8

未排序資料 9, 5, 12, 19, 2, 6, 4, 15, 10, 3, 7

由小到大排序完成之資料 2, 3, 4, 5, 6, 7, 9, 10, 12, 15, 19

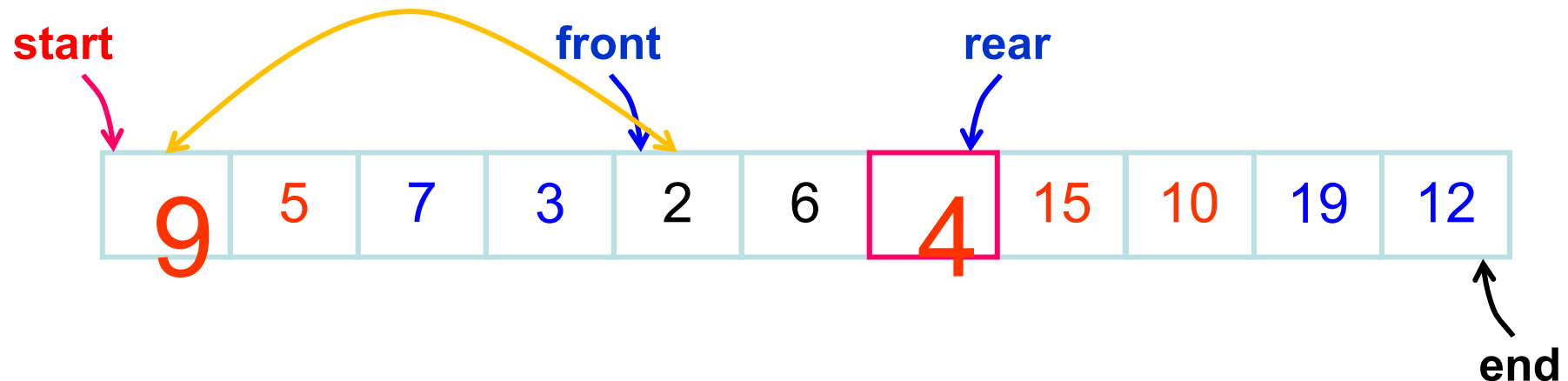


while (front ≤ rear) { ... }

快速排序法 – $O(n \log n)$ 5/8

未排序資料 9, 5, 12, 19, 2, 6, 4, 15, 10, 3, 7

由小到大排序完成之資料 2, 3, 4, 5, 6, 7, 9, 10, 12, 15, 19

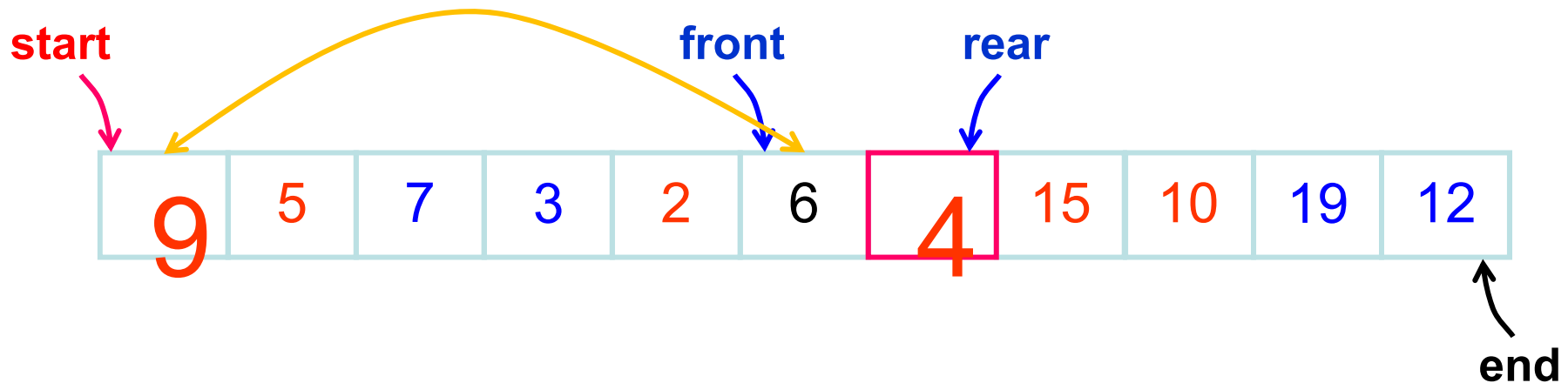


while (front ≤ rear) { ... }

快速排序法 – $O(n \log n)$ 5/8

未排序資料 9, 5, 12, 19, 2, 6, 4, 15, 10, 3, 7

由小到大排序完成之資料 2, 3, 4, 5, 6, 7, 9, 10, 12, 15, 19

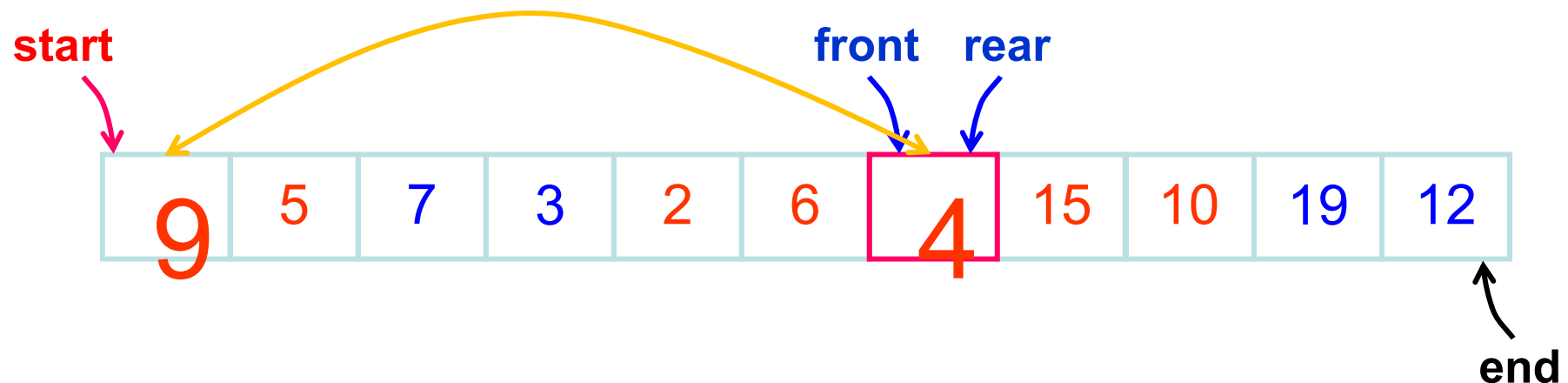


while (front ≤ rear) { ... }

快速排序法 – $O(n \log n)$ 5/8

未排序資料 9, 5, 12, 19, 2, 6, 4, 15, 10, 3, 7

由小到大排序完成之資料 2, 3, 4, 5, 6, 7, 9, 10, 12, 15, 19

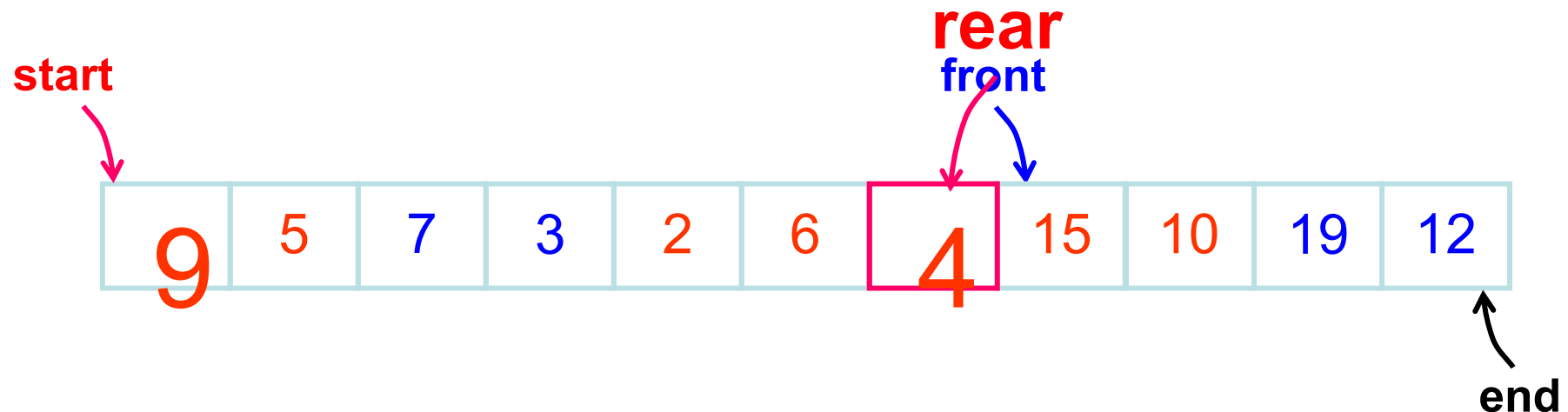


while (front <= rear) { ... }

快速排序法 – $O(n \log n)$ 5/8

未排序資料 9, 5, 12, 19, 2, 6, 4, 15, 10, 3, 7

由小到大排序完成之資料 2, 3, 4, 5, 6, 7, 9, 10, 12, 15, 19

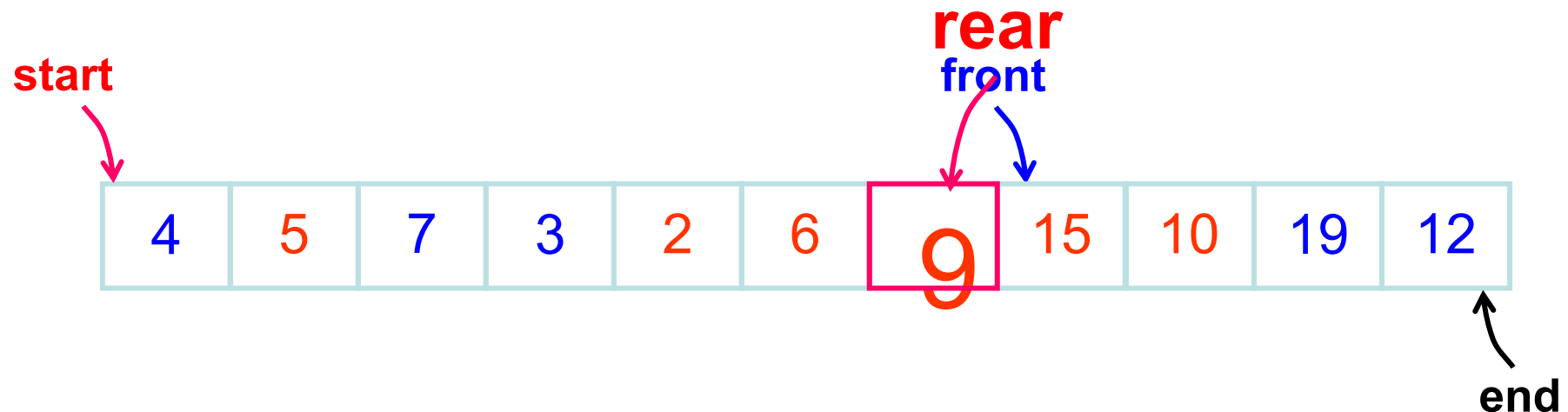


while (front ≤ rear) { ... }

快速排序法 – $O(n \log n)$ 5/8

未排序資料 9, 5, 12, 19, 2, 6, 4, 15, 10, 3, 7

由小到大排序完成之資料 2, 3, 4, 5, 6, 7, 9, 10, 12, 15, 19



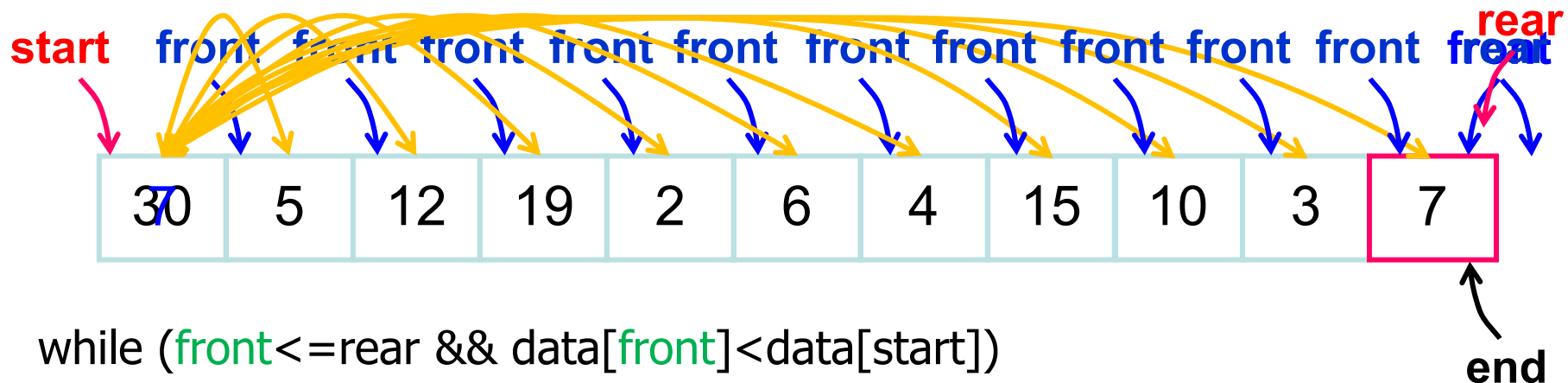
```
if (start < rear) swap(&data[start], &data[rear]);
```

得到兩個長度比較短的排序問題

快速排序法 – $O(n \log n)$ 6/8

未排序資料 30, 5, 12, 19, 2, 6, 4, 15, 10, 3, 7

由小到大排序完成之資料 2, 3, 4, 5, 6, 7, 10, 12, 15, 19, 30



```
while (front <= rear && data[front] < data[start])  
    front++;
```

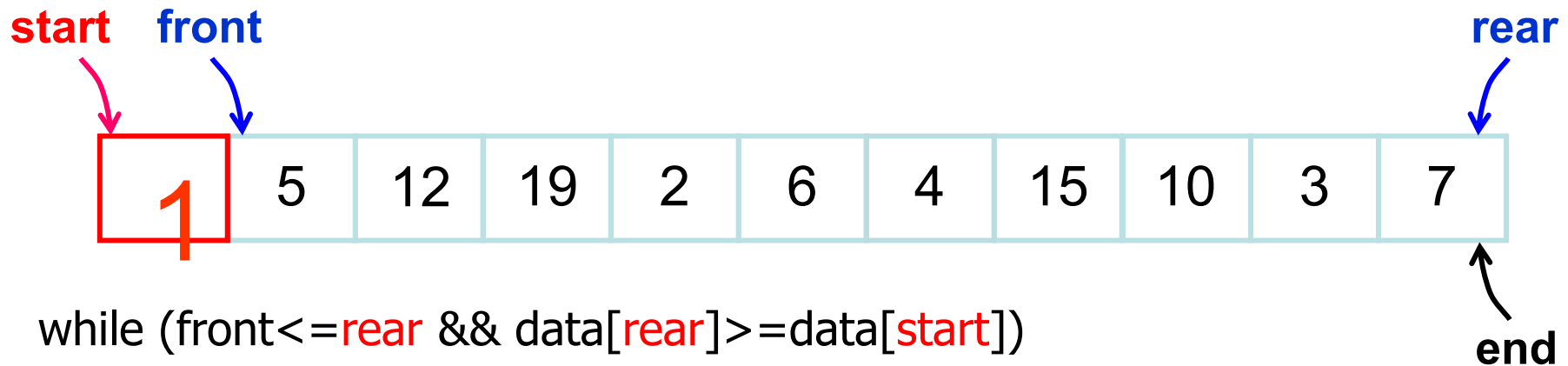
★front 最多會到達 end+1

★front 停下來時如果小於等於 end, data[front] 一定大於等於 data[start]

快速排序法 – $O(n \log n)$ 7/8

未排序資料 1, 5, 12, 19, 2, 6, 4, 15, 10, 3, 7

由小到大排序完成之資料 1, 2, 3, 4, 5, 6, 7, 10, 12, 15, 19



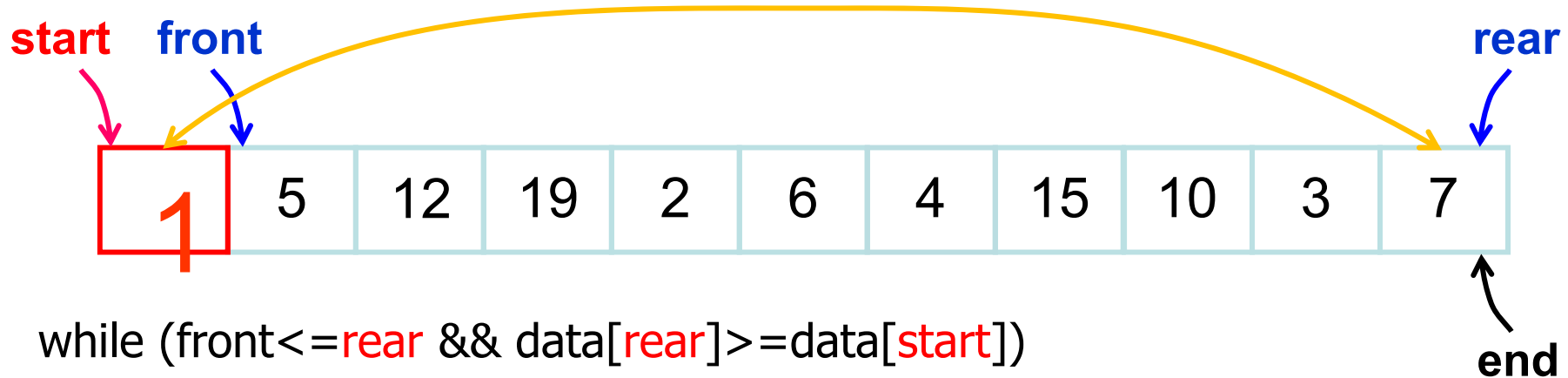
★rear 最多只會到達 start

★rear 停下來時如果大於 start, data[rear] 一定小於 data[start]

快速排序法 – $O(n \log n)$ 7/8

未排序資料 1, 5, 12, 19, 2, 6, 4, 15, 10, 3, 7

由小到大排序完成之資料 1, 2, 3, 4, 5, 6, 7, 10, 12, 15, 19



```
while (front <= rear && data[rear] >= data[start])  
    rear--;
```

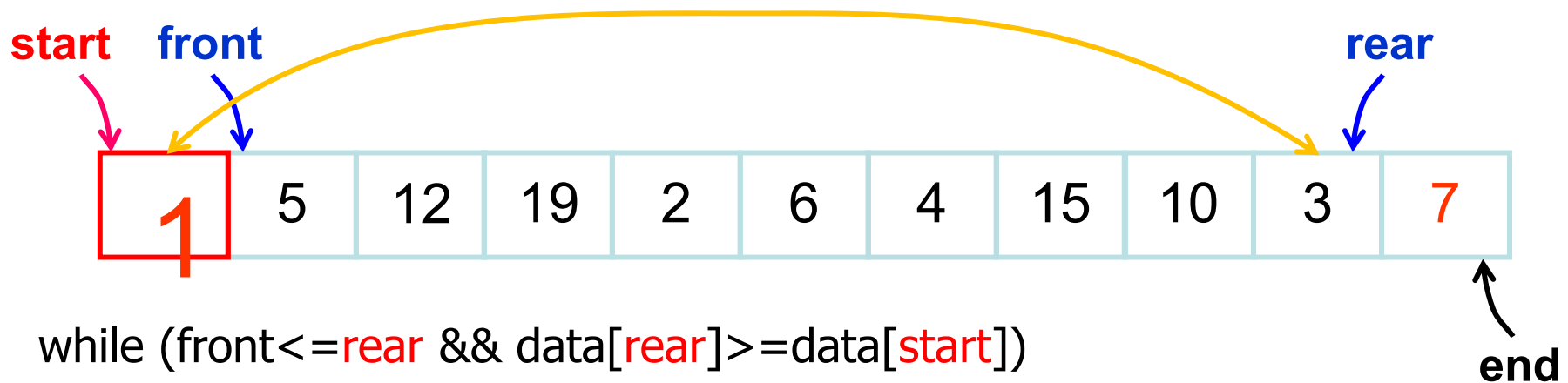
★rear 最多只會到達 start

★rear 停下來時如果大於 start, data[rear] 一定小於 data[start]

快速排序法 – $O(n \log n)$ 7/8

未排序資料 1, 5, 12, 19, 2, 6, 4, 15, 10, 3, 7

由小到大排序完成之資料 1, 2, 3, 4, 5, 6, 7, 10, 12, 15, 19



```
while (front <= rear && data[rear] >= data[start])  
    rear--;
```

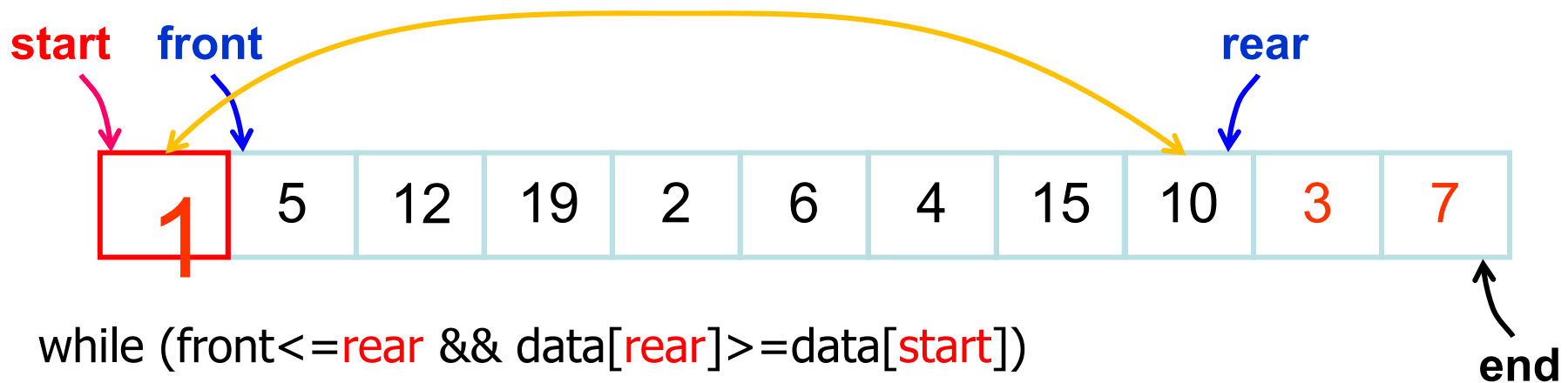
★rear 最多只會到達 start

★rear 停下來時如果大於 start, data[rear] 一定小於 data[start]

快速排序法 – $O(n \log n)$ 7/8

未排序資料 1, 5, 12, 19, 2, 6, 4, 15, 10, 3, 7

由小到大排序完成之資料 1, 2, 3, 4, 5, 6, 7, 10, 12, 15, 19



```
while (front <= rear && data[rear] >= data[start])  
    rear--;
```

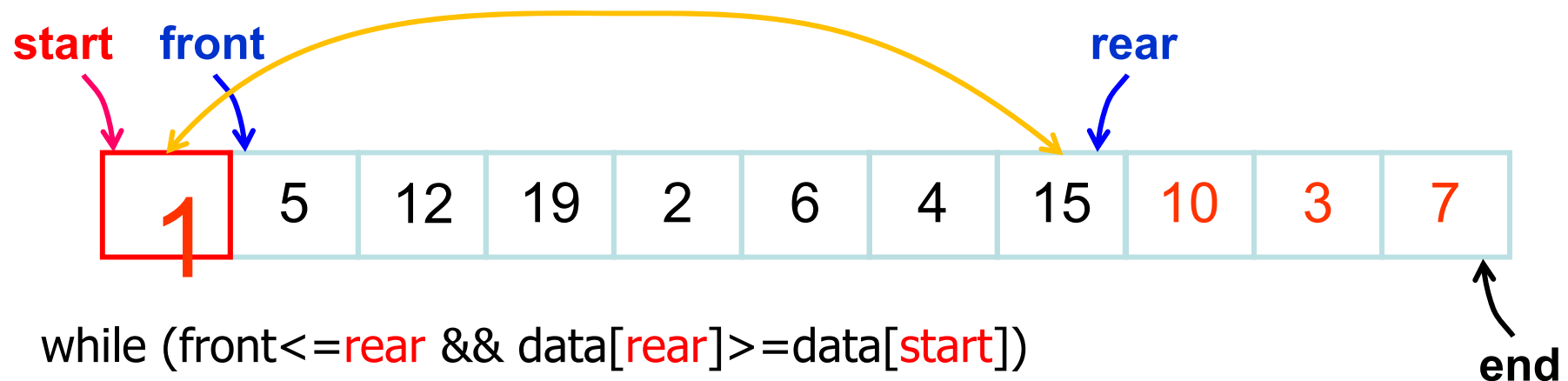
★rear 最多只會到達 start

★rear 停下來時如果大於 start, data[rear] 一定小於 data[start]

快速排序法 – $O(n \log n)$ 7/8

未排序資料 1, 5, 12, 19, 2, 6, 4, 15, 10, 3, 7

由小到大排序完成之資料 1, 2, 3, 4, 5, 6, 7, 10, 12, 15, 19



```
while (front <= rear && data[rear] >= data[start])  
    rear--;
```

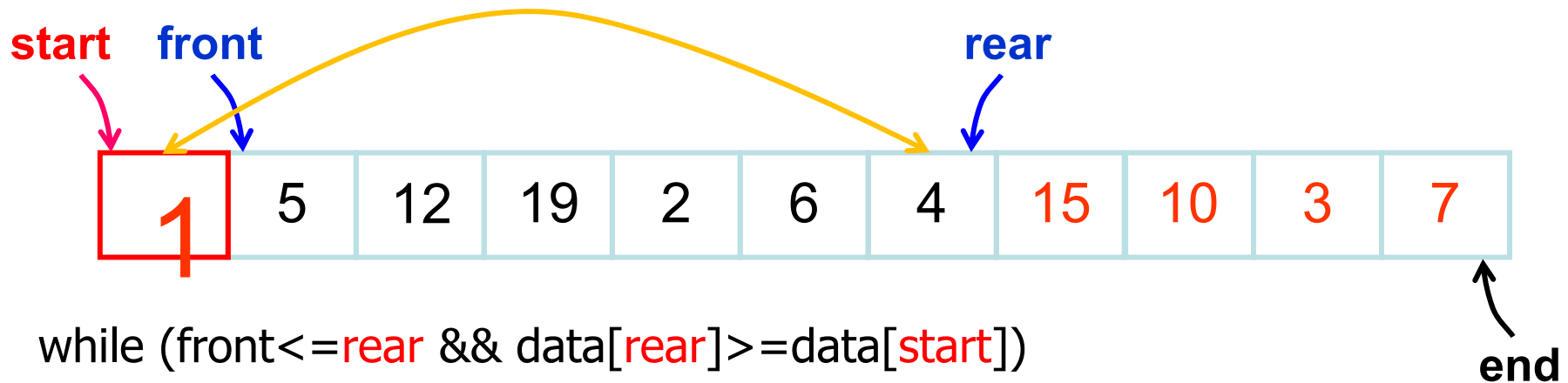
★rear 最多只會到達 start

★rear 停下來時如果大於 start, data[rear] 一定小於 data[start]

快速排序法 – $O(n \log n)$ 7/8

未排序資料 1, 5, 12, 19, 2, 6, 4, 15, 10, 3, 7

由小到大排序完成之資料 1, 2, 3, 4, 5, 6, 7, 10, 12, 15, 19



```
while (front <= rear && data[rear] >= data[start])  
    rear--;
```

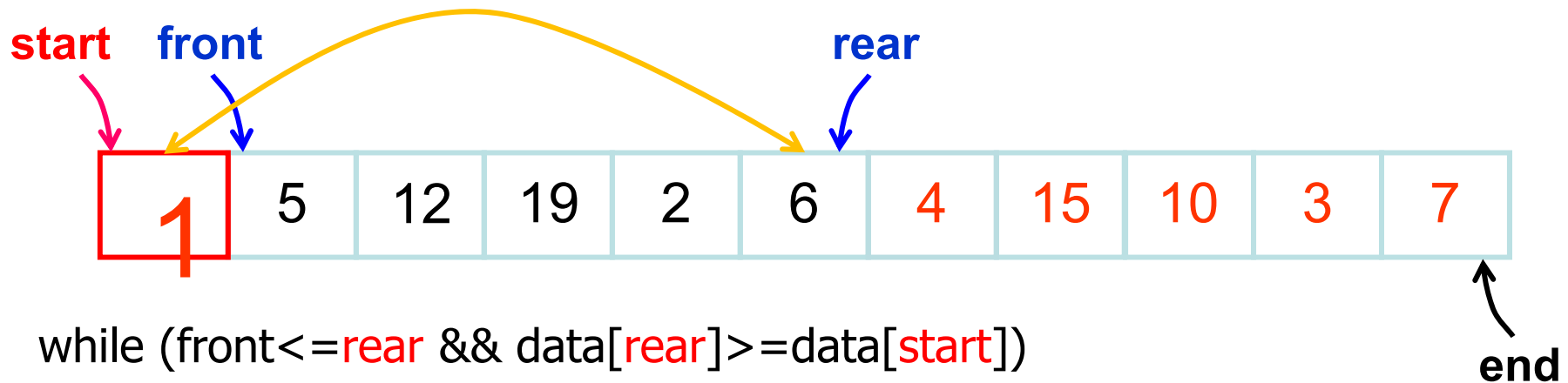
★rear 最多只會到達 start

★rear 停下來時如果大於 start, data[rear] 一定小於 data[start]

快速排序法 – $O(n \log n)$ 7/8

未排序資料 1, 5, 12, 19, 2, 6, 4, 15, 10, 3, 7

由小到大排序完成之資料 1, 2, 3, 4, 5, 6, 7, 10, 12, 15, 19



```
while (front <= rear && data[rear] >= data[start])  
    rear--;
```

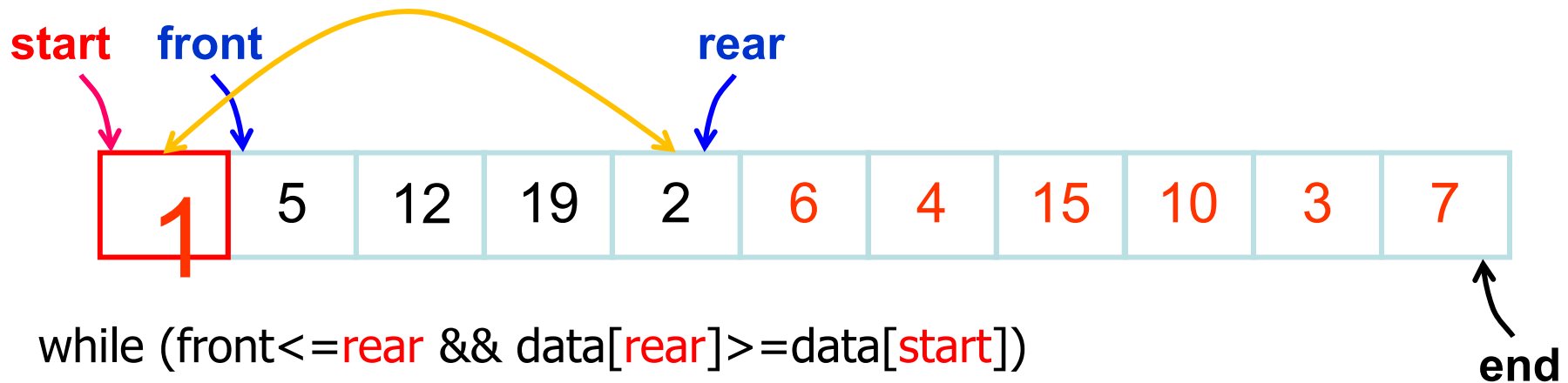
★rear 最多只會到達 start

★rear 停下來時如果大於 start, data[rear] 一定小於 data[start]

快速排序法 – $O(n \log n)$ 7/8

未排序資料 1, 5, 12, 19, 2, 6, 4, 15, 10, 3, 7

由小到大排序完成之資料 1, 2, 3, 4, 5, 6, 7, 10, 12, 15, 19



```
while (front <= rear && data[rear] >= data[start])  
    rear--;
```

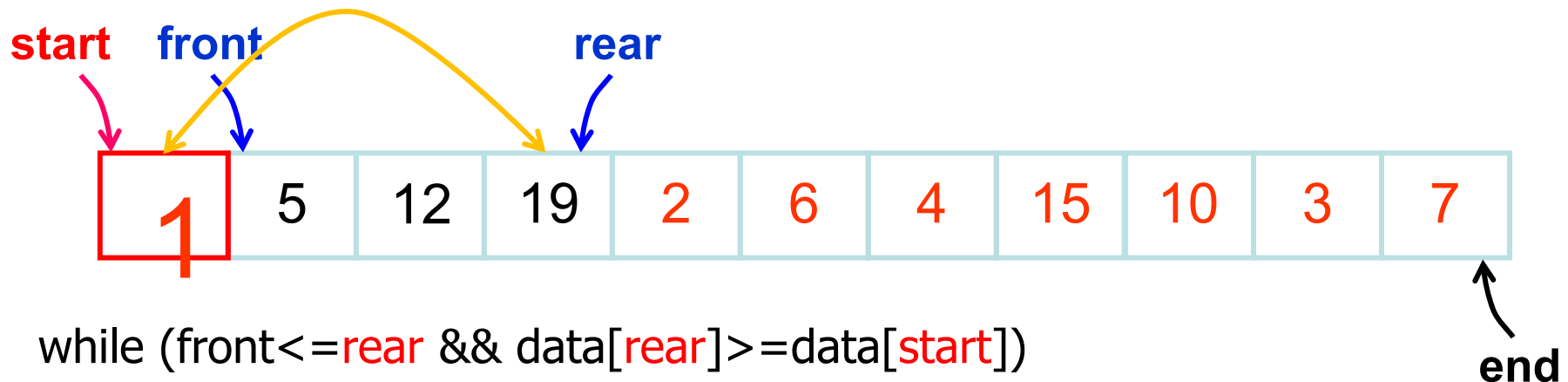
★rear 最多只會到達 start

★rear 停下來時如果大於 start, data[rear] 一定小於 data[start]

快速排序法 – $O(n \log n)$ 7/8

未排序資料 1, 5, 12, 19, 2, 6, 4, 15, 10, 3, 7

由小到大排序完成之資料 1, 2, 3, 4, 5, 6, 7, 10, 12, 15, 19



```
while (front <= rear && data[rear] >= data[start])  
    rear--;
```

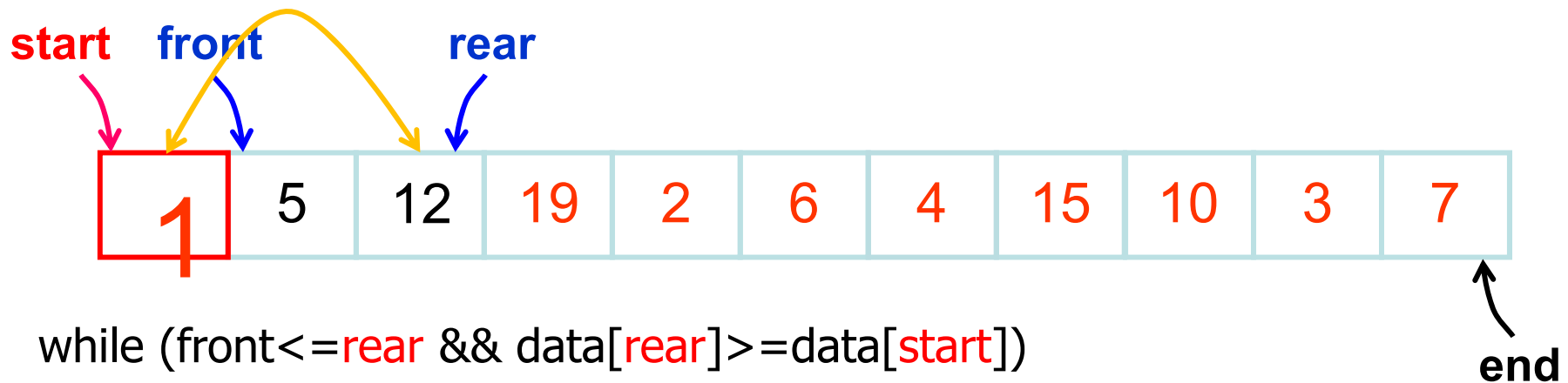
★rear 最多只會到達 start

★rear 停下來時如果大於 start, data[rear] 一定小於 data[start]

快速排序法 – $O(n \log n)$ 7/8

未排序資料 1, 5, 12, 19, 2, 6, 4, 15, 10, 3, 7

由小到大排序完成之資料 1, 2, 3, 4, 5, 6, 7, 10, 12, 15, 19



```
while (front <= rear && data[rear] >= data[start])  
    rear--;
```

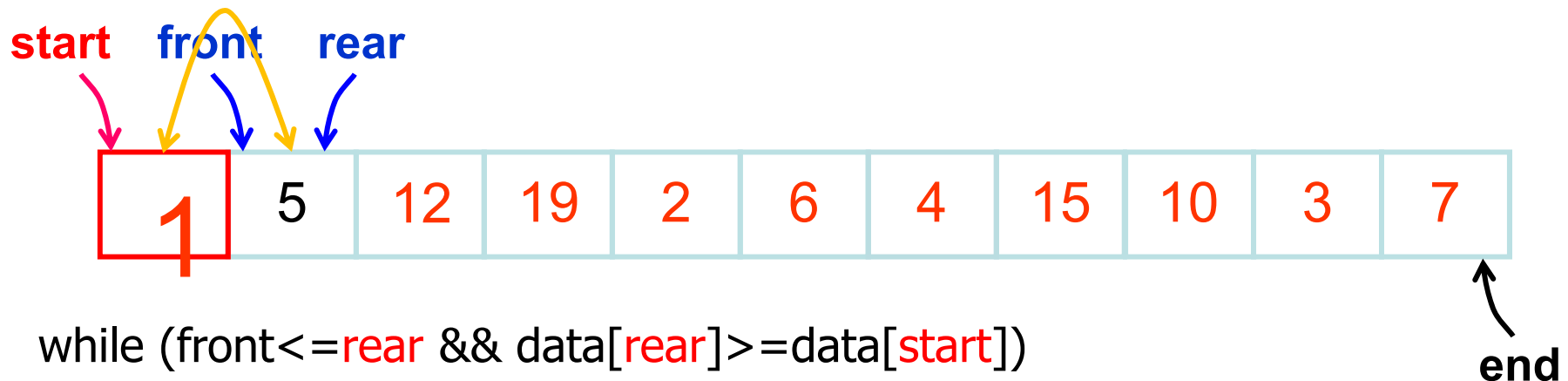
★rear 最多只會到達 start

★rear 停下來時如果大於 start, data[rear] 一定小於 data[start]

快速排序法 – $O(n \log n)$ 7/8

未排序資料 1, 5, 12, 19, 2, 6, 4, 15, 10, 3, 7

由小到大排序完成之資料 1, 2, 3, 4, 5, 6, 7, 10, 12, 15, 19



```
while (front <= rear && data[rear] >= data[start])  
    rear--;
```

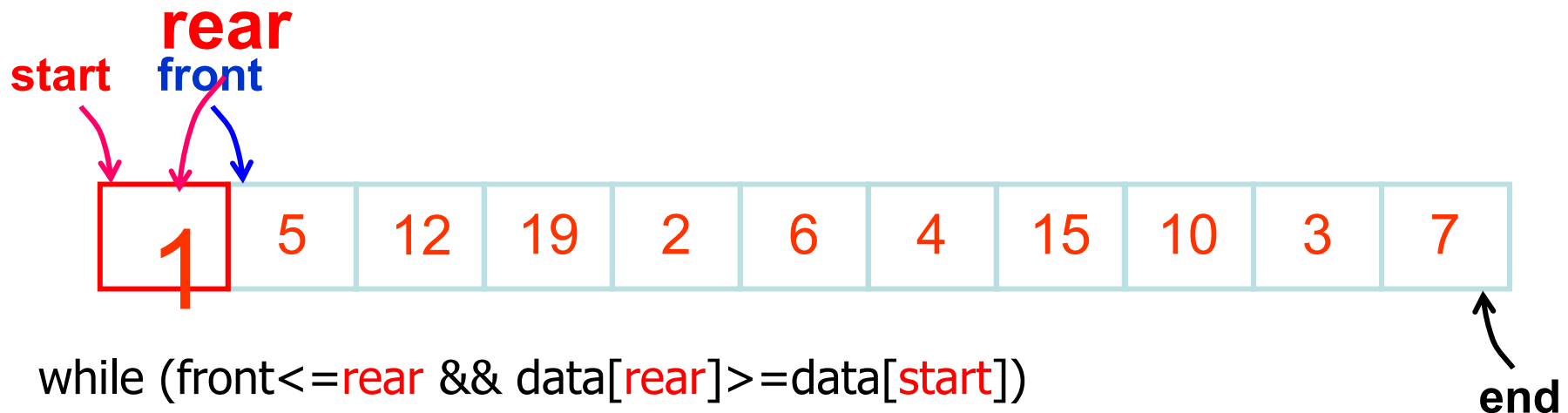
★rear 最多只會到達 start

★rear 停下來時如果大於 start, data[rear] 一定小於 data[start]

快速排序法 – $O(n \log n)$ 7/8

未排序資料 1, 5, 12, 19, 2, 6, 4, 15, 10, 3, 7

由小到大排序完成之資料 1, 2, 3, 4, 5, 6, 7, 10, 12, 15, 19



```
while (front <= rear && data[rear] >= data[start])  
    rear--;
```

★rear 最多只會到達 start

★rear 停下來時如果大於 start, data[rear] 一定小於 data[start]

快速排序法 – $O(n \log n)$ 8/8

➤ n 筆資料排序

- 分治法**平均**需要分割多少次：如果每次分割點都在正中間, $n \rightarrow n/2 \rightarrow n/4 \rightarrow \dots \rightarrow 2$, 需要 $\log n - 1$ 次
- 分治法**最差**需要分割多少次：如果每次分割點剛好在最前面或是最後面, $n \rightarrow n-1 \rightarrow n-2 \rightarrow \dots \rightarrow 2$, 需要 $n - 1$ 次
- 每一次尋找分割點需要的比較次數：front 由 $\text{start}+1$ 開始往後移, rear 由 end 開始往前移, 直到相遇 ($\text{front} > \text{rear}$) 為止, 共有 $n-1$ 個元素和 start 那個元素比較.... $O(n)$
 - ✦ 分割以後 $n \rightarrow a, b$ ($n = a + b + 1$), 變成兩個大小為 a 和 b 的問題, 兩個問題的比較次數都正比於 a 和 b , 把這兩個問題的比較次數加起來, 還是正比於 n

➤ $O(n \log n)$