บทที่ 9-2 Sorting

บทเรียนย่อย

9.8 Shell Sort Algorithm

9.9 Shell Sort Implementation

9.10 Merge Sort Algorithm

9.11 Merge Sort Implementation

9.12 Quick Sort Algorithm

9.13 Quick Sort Implementation

บทที่ 9-2 Sorting

บทเรียนย่อย

9.8 Shell Sort Algorithm

9.9 Shell Sort Implementation

9.10 Merge Sort Algorithm

9.11 Merge Sort Implementation

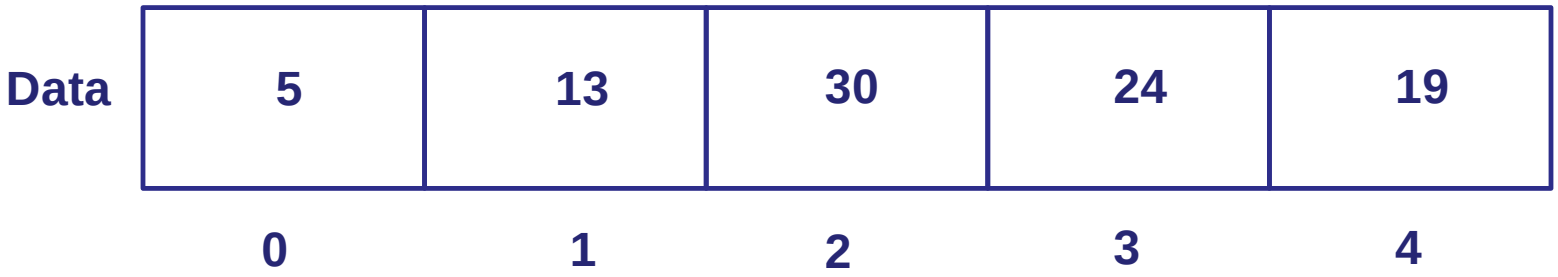
9.12 Quick Sort Algorithm

9.13 Quick Sort Implementation

9.8 Shell Sort Algorithm

Shell Sort หรือ Diminishing Increment Sort เป็นวิธีการจัดเรียงที่พัฒนามาจาก Insertion Sort โดยทำการหลีกเลี่ยงการทำการเปรียบเทียบที่มากครั้ง หลักการของ Shell Sort จะเริ่มต้นจากเปรียบเทียบข้อมูลที่อยู่ห่างกัน (gap) และเปรียบเทียบห่างน้อยลง จนเหลือห่างกัน (gap = 1) แบบทำ insertion sort

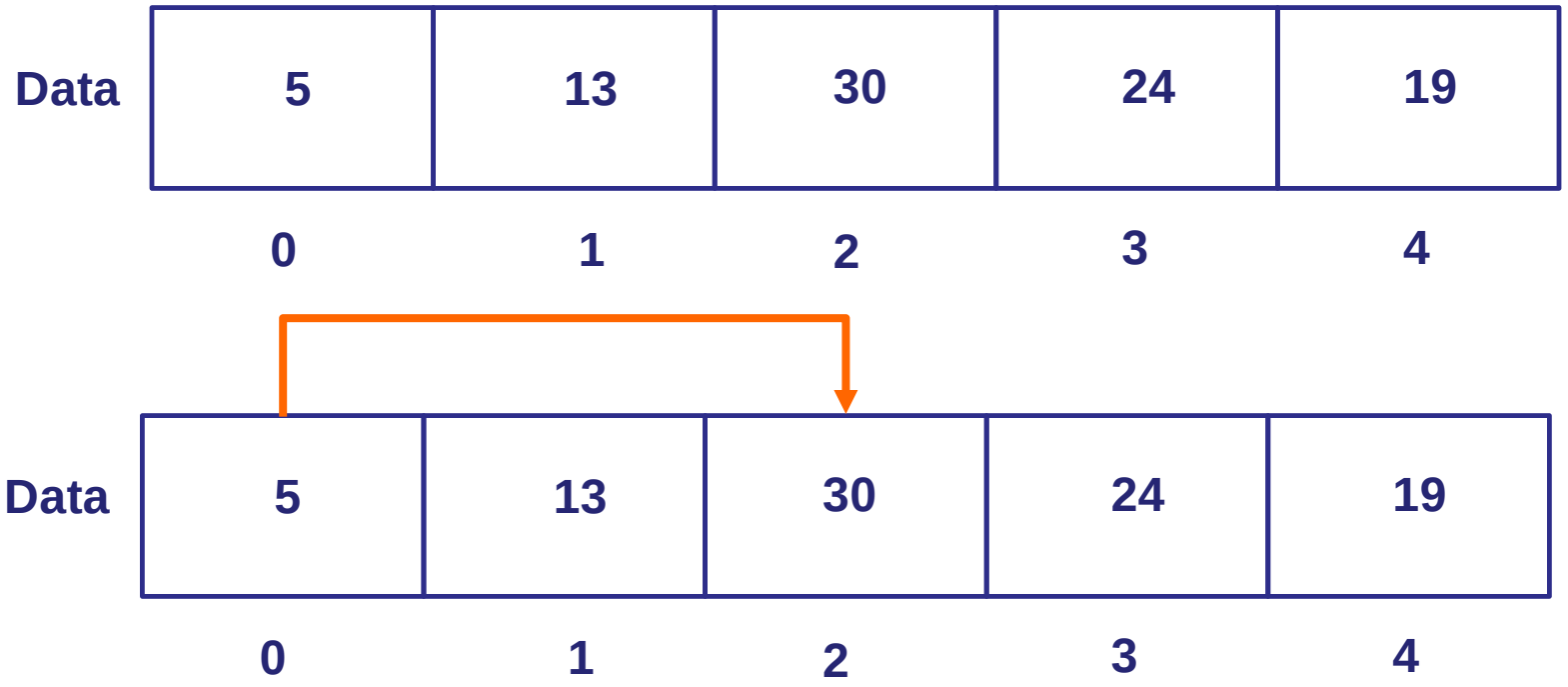
ลักษณะการดำเนินการของ Shell Sort



1. Gap = N / 2

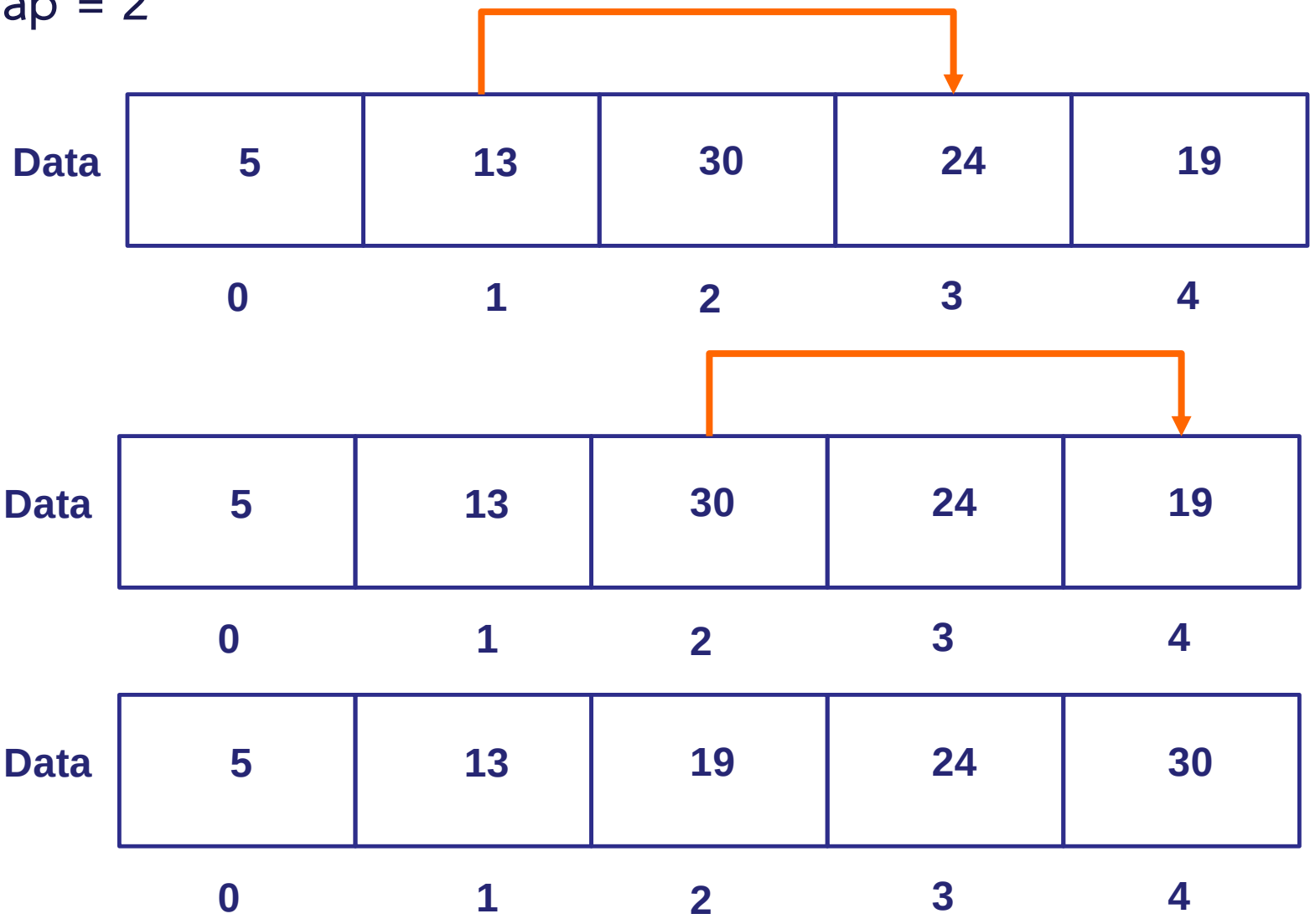
ลักษณะการดำเนินการของ Shell Sort

Gap = 2



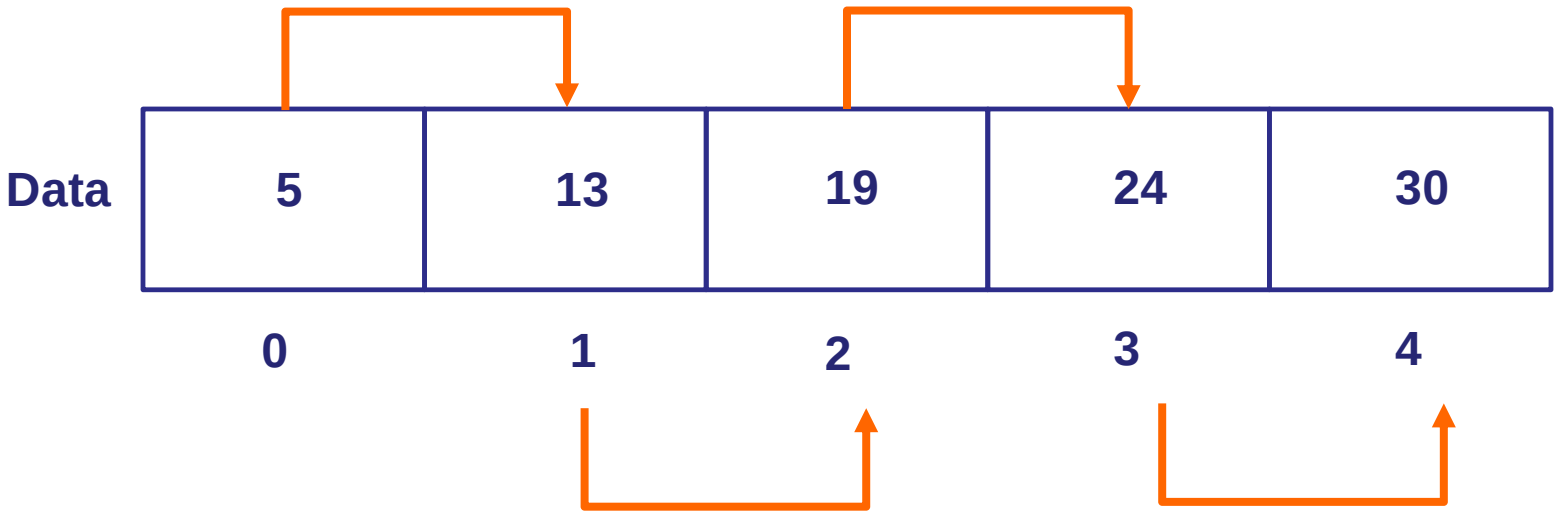
ลักษณะการดำเนินการของ Shell Sort

Gap = 2



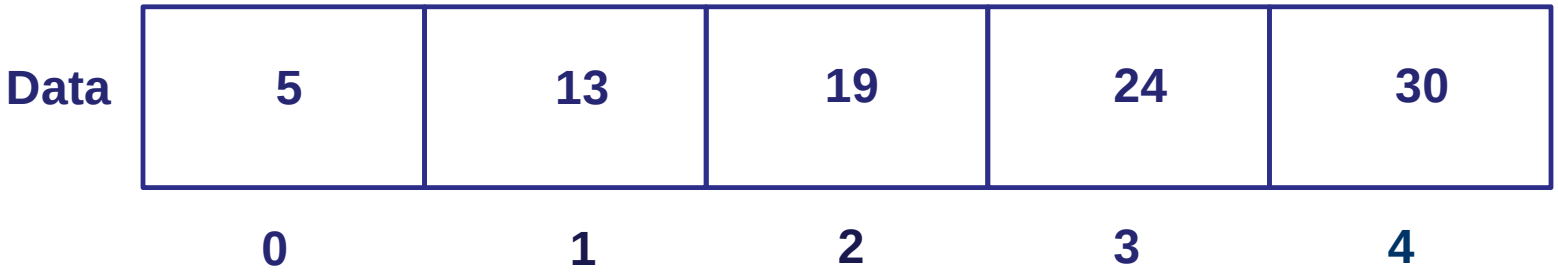
ลักษณะการดำเนินการของ Shell Sort

Gap = 1



ลักษณะการดำเนินการของ Shell Sort

ผลลัพธ์



Time :: Best: O(n log n) Worst: $O(n^2)$

Average: depends on gap sequence

บทที่ 9-2 Sorting

บทเรียนย่อย

9.8 Shell Sort Algorithm

9.9 Shell Sort Implementation

9.10 Merge Sort Algorithm

9.11 Merge Sort Implementation

9.12 Quick Sort Algorithm

9.13 Quick Sort Implementation

9.9 Shell Sort Implementation

Algorithm : Shell Sort

Pre : data[] (int) , size (int)

Post : ข้อมูลถูกจัดเรียงลำดับจากน้อยไปมาก

1. เริ่มต้น
2. ประกาศตัวแปรชื่อ gap , temp , i และ j เป็นเลขจำนวนเต็ม
3. ทำซ้ำ ตั้งแต่ gap = size / 2 จนมีค่าเท่ากับ gap < 0
    - 3.1 ทำซ้ำ ตั้งแต่ i = gap จนมีค่าเท่ากับ i > size
        - 3.1.1 กำหนดให้ j มีค่าเท่ากับ i - 1

.....

9.9 Shell Sort Implementation

.....

3.1.2 ทำซ้ำจนกว่า j มีค่าน้อยกว่า 0 หรือ data ตำแหน่งที่ j

จะมีค่าไม่มากกว่า temp

3.3.1 กำหนดให้ data ตำแหน่งที่ j + 1 มีค่าเท่ากับ data ตำแหน่งที่ j

3.3.2 กำหนดให้ j มีค่าเท่ากับ j – 1

4. กำหนดให้ data ตำแหน่งที่ j + 1 มีค่าเท่ากับ temp

5. จบการทำงาน

9.9 Shell Sort Implementation

Algorithm การจัดเรียงข้อมูลแบบ Shell Sort

```
1   List of data, have the amount of data equal N, list[N]
2   gap=round(N/2) /* round() : to be  an integer that ends with one or more zeroes */
3   while gap > 0 do
4       for index = gap to N increment by 1
5           temp = list[index]
6           subindex = index
7           while subindex >= gap and list[subindex - gap] > temp do
8               list[subindex] = list[subindex - gap]
9               subindex = subindex - gap
10          end while
11          list[subindex] = temp
12      end for
13      gap = round(gap/2)
14  end while
```

บทที่ 9-2 Sorting

บทเรียนย่อย

9.8 Shell Sort Algorithm

9.9 Shell Sort Implementation

9.10 Merge Sort Algorithm

9.11 Merge Sort Implementation

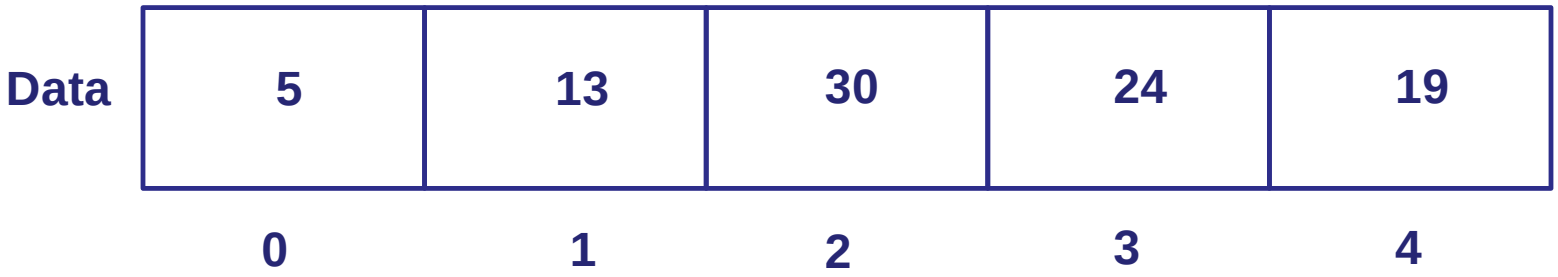
9.12 Quick Sort Algorithm

9.13 Quick Sort Implementation

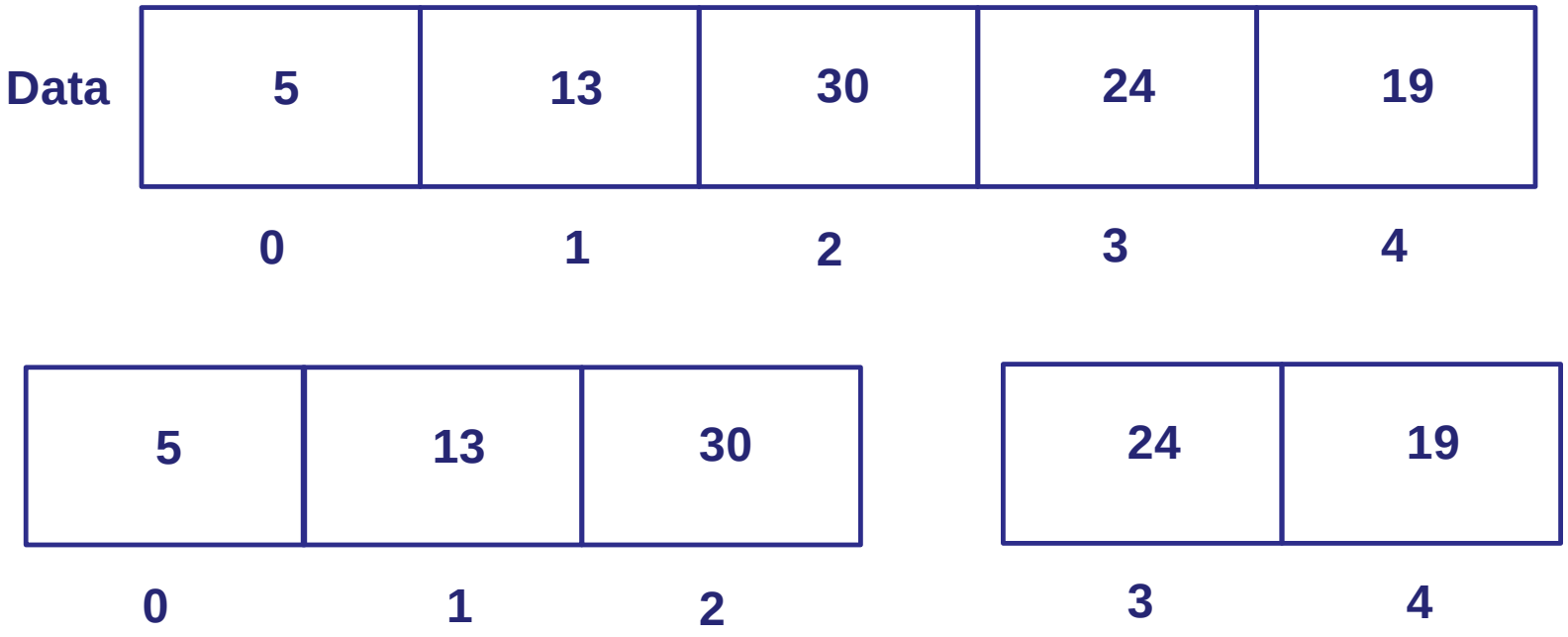
9.10 Merge Sort Algorithm

Merge Sort (การเรียงลำดับแบบผสาน) เป็นขั้นตอนวิธีในการเรียงลำดับที่อาศัยการเปรียบเทียบ และยังเป็นตัวอย่างขั้นตอนวิธีที่ใช้หลักการ divide and conquer มีหลักการก็คือ ให้แบ่งข้อมูลออกเป็น 2 ส่วนก่อน ซึ่งแต่ละส่วนก็แบ่งออกเป็นอีก 2 ส่วนอีกต่อไปเรื่อยๆ จนกระทั่งไม่สามารถแบ่งได้อีก แล้วจึงค่อยทำการจัดเรียงข้อมูลในส่วนย่อย จากนั้นนำข้อมูลส่วนย่อยดังกล่าวมารวมกันใหม่อีกครั้ง

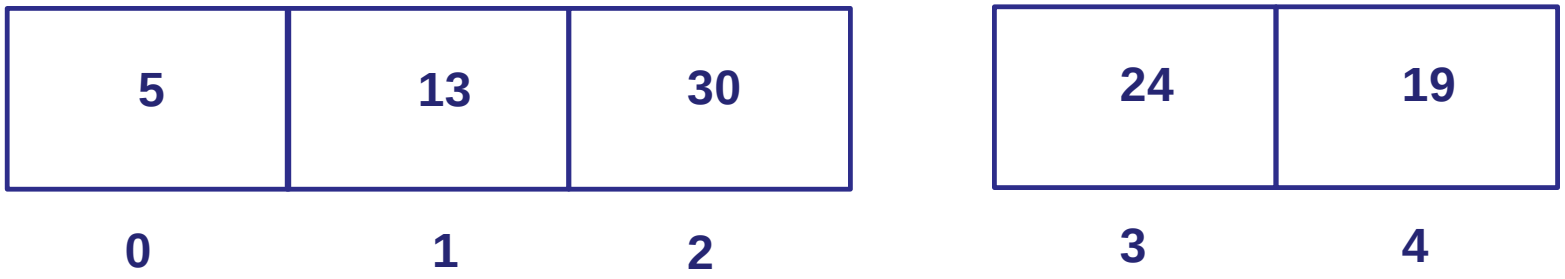
ลักษณะการดำเนินการของ Merge Sort



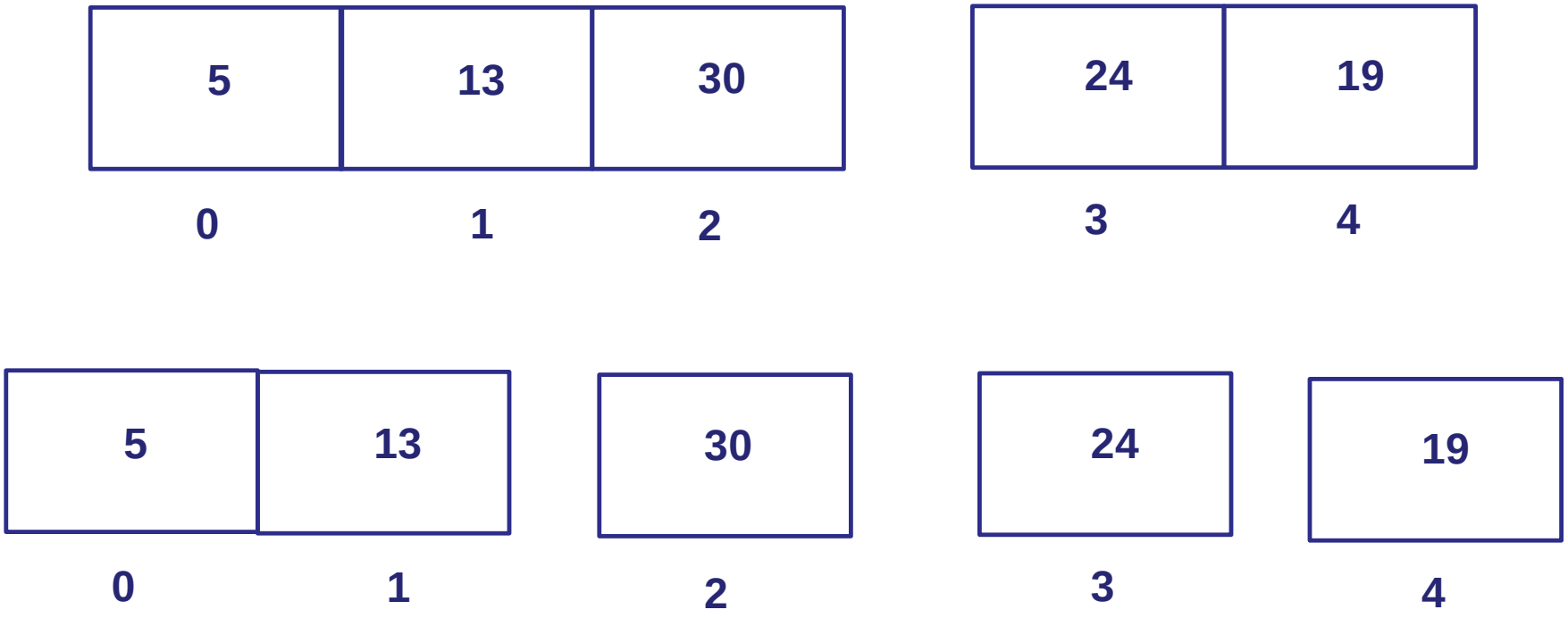
ลักษณะการดำเนินการของ Merge Sort



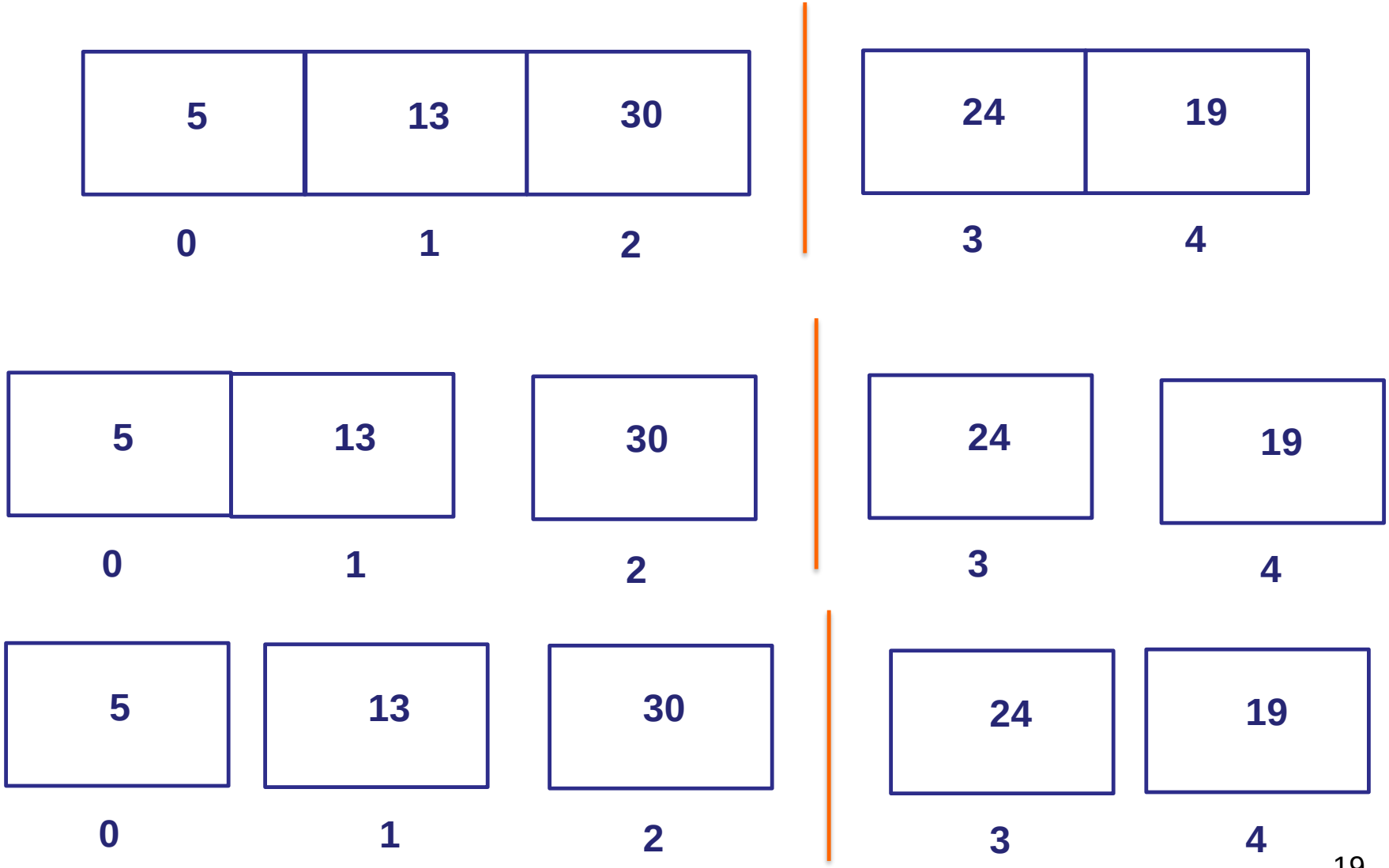
ลักษณะการดำเนินการของ Merge Sort



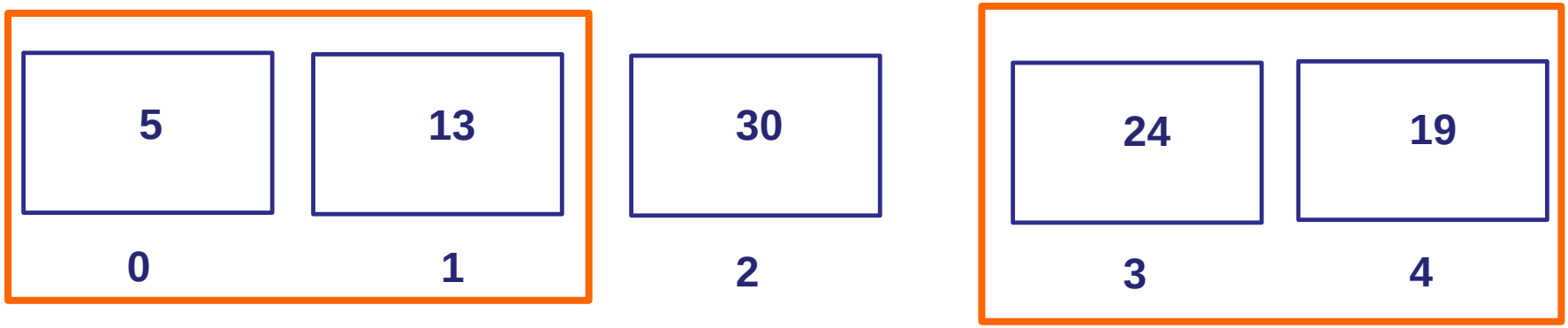
ลักษณะการดำเนินการของ Merge Sort



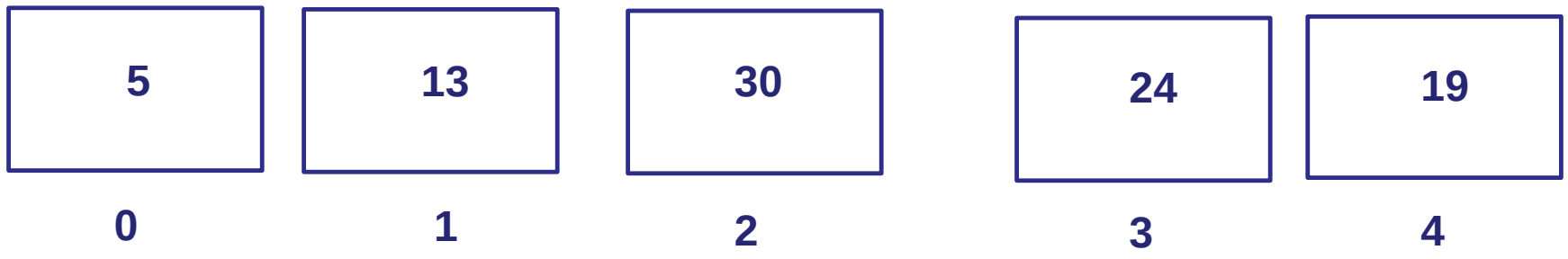
ลักษณะการดำเนินการของ Merge Sort

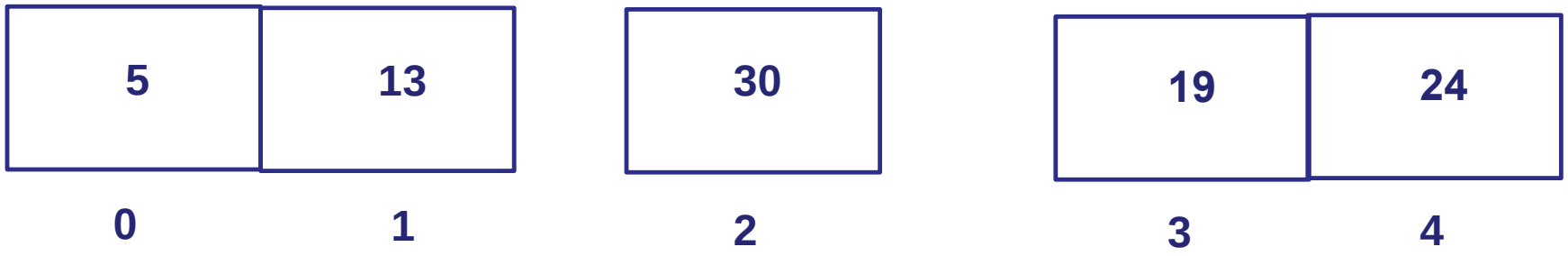


ลักษณะการดำเนินการของ Merge Sort

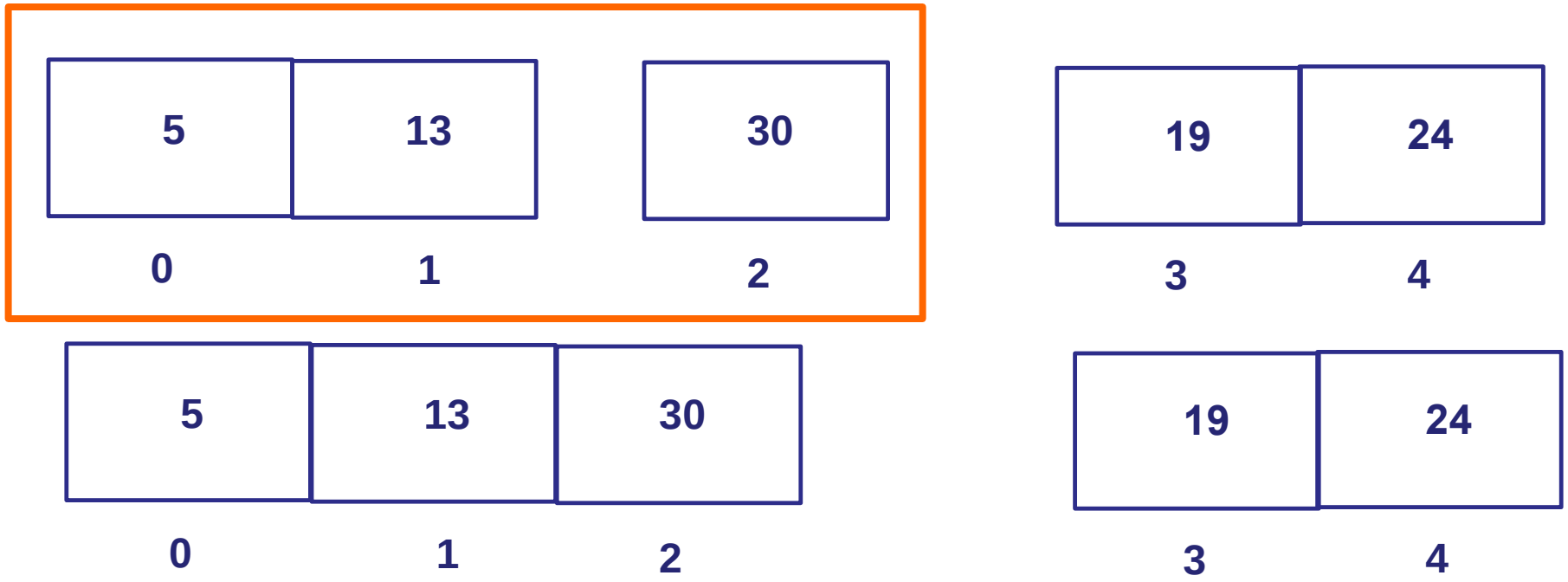


ลักษณะการดำเนินการของ Merge Sort

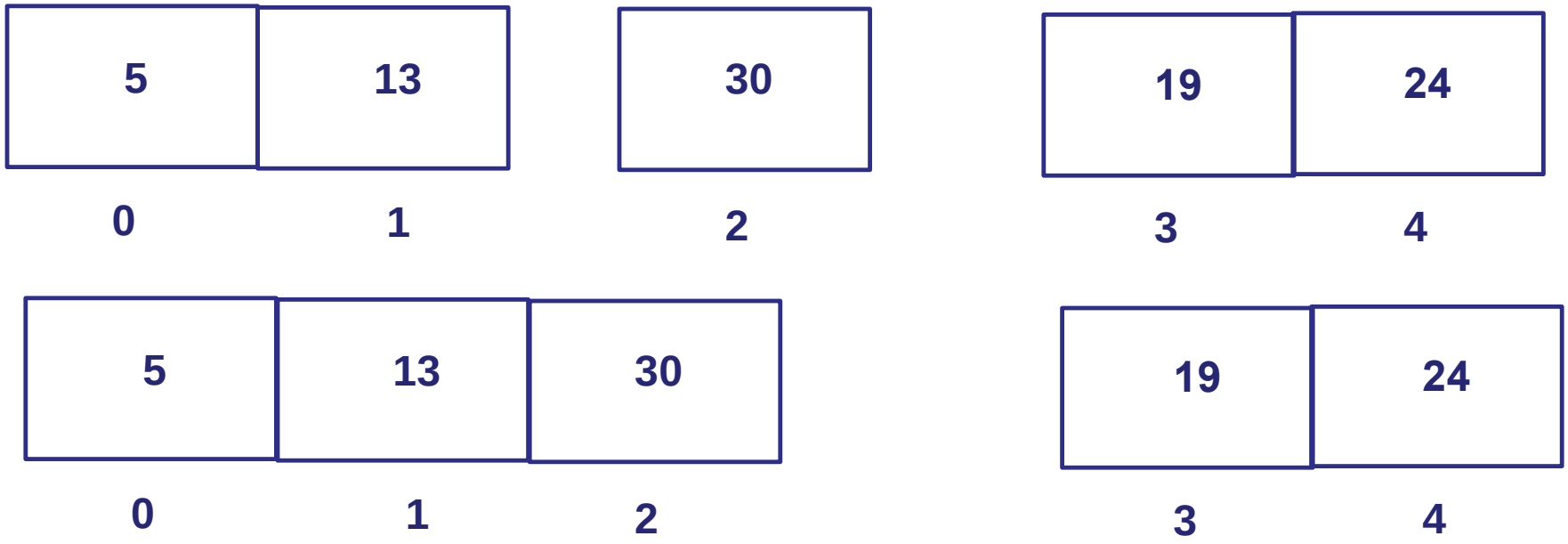




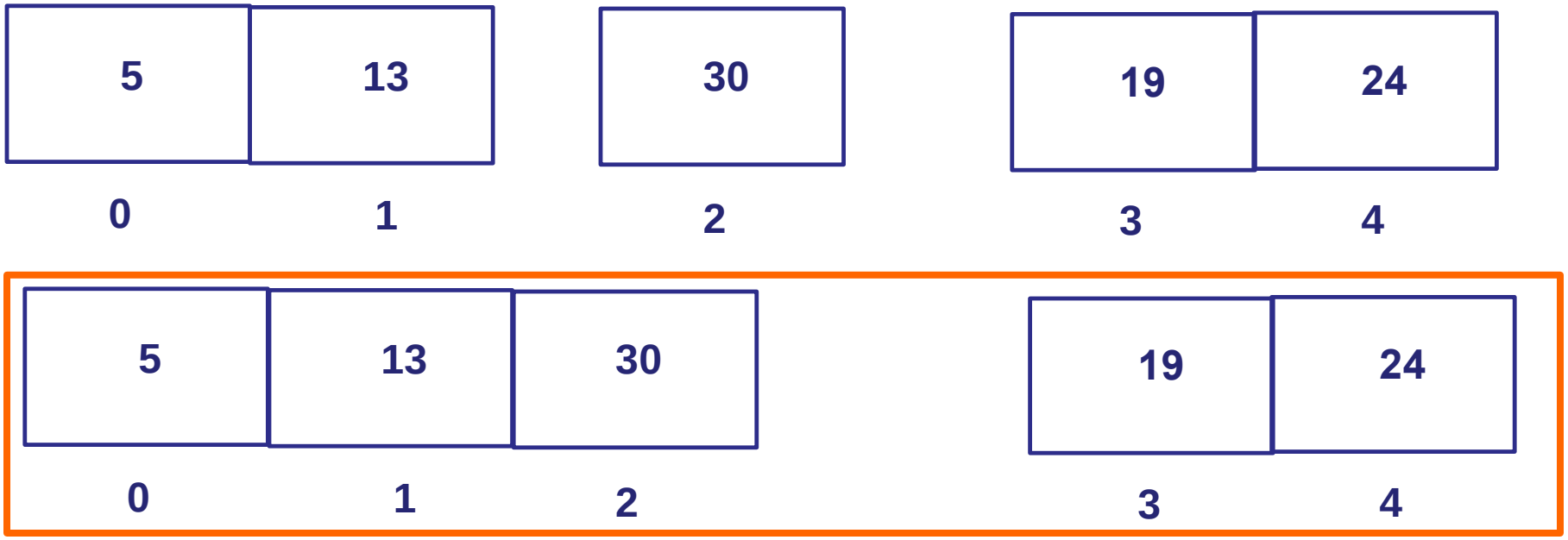
ลักษณะการดำเนินการของ Merge Sort



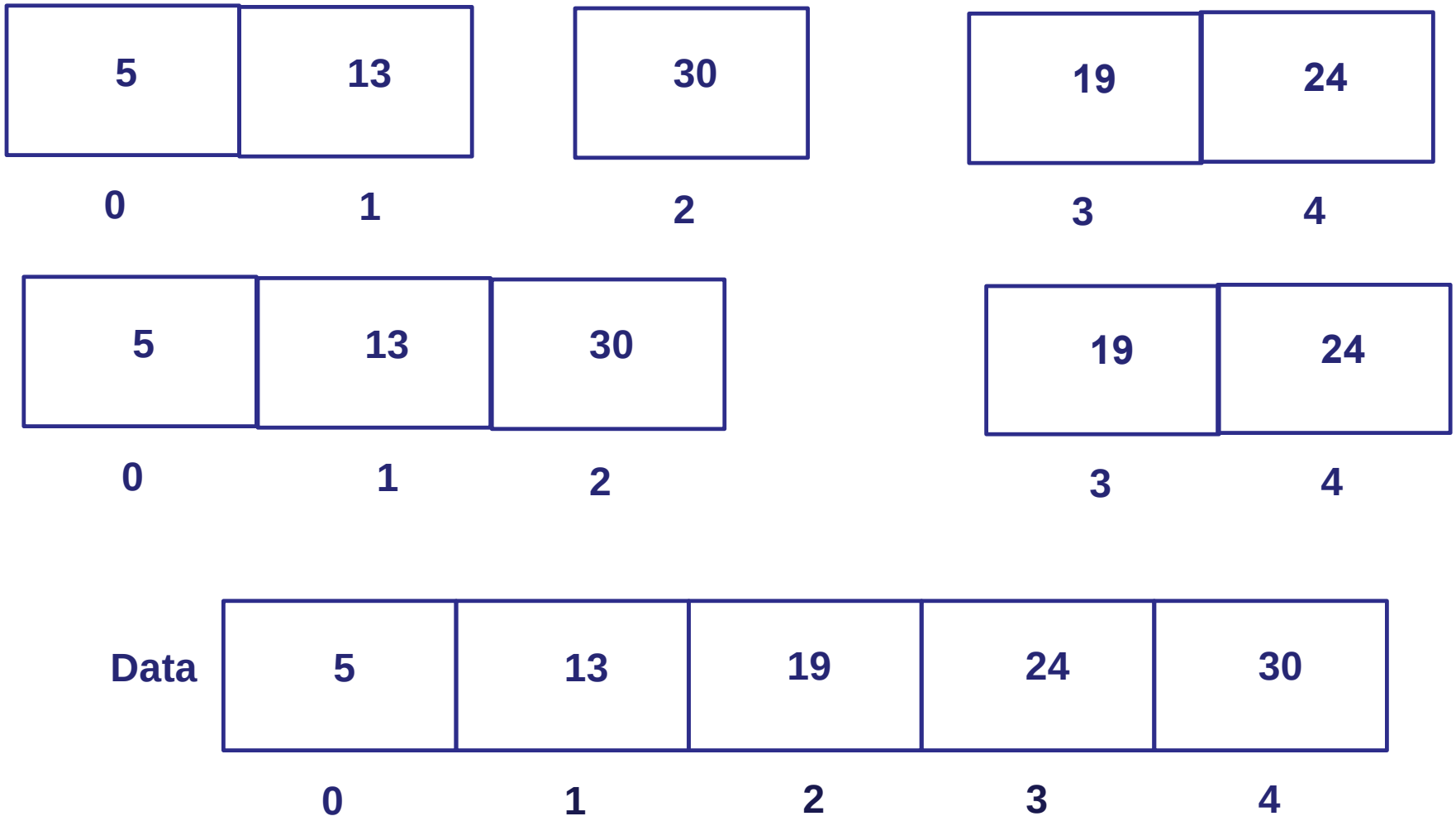
ลักษณะการดำเนินการของ Merge Sort



ลักษณะการดำเนินการของ Merge Sort

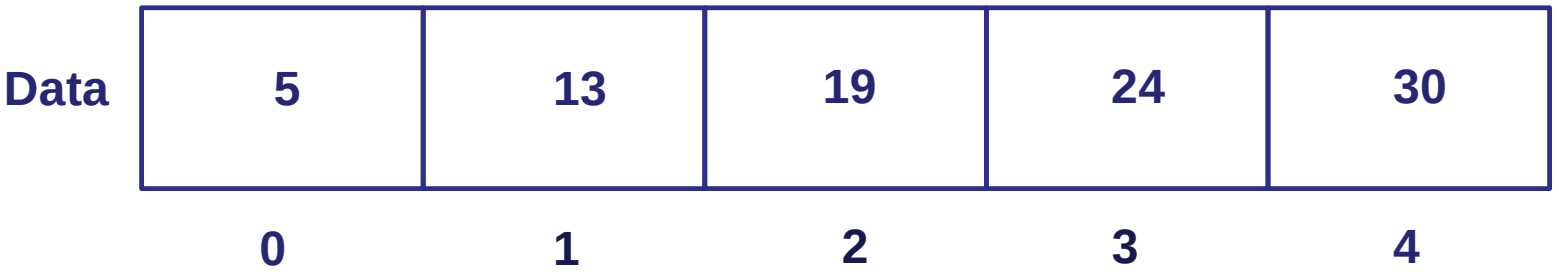


ลักษณะการดำเนินการของ Merge Sort



ลักษณะการดำเนินการของ Merge Sort

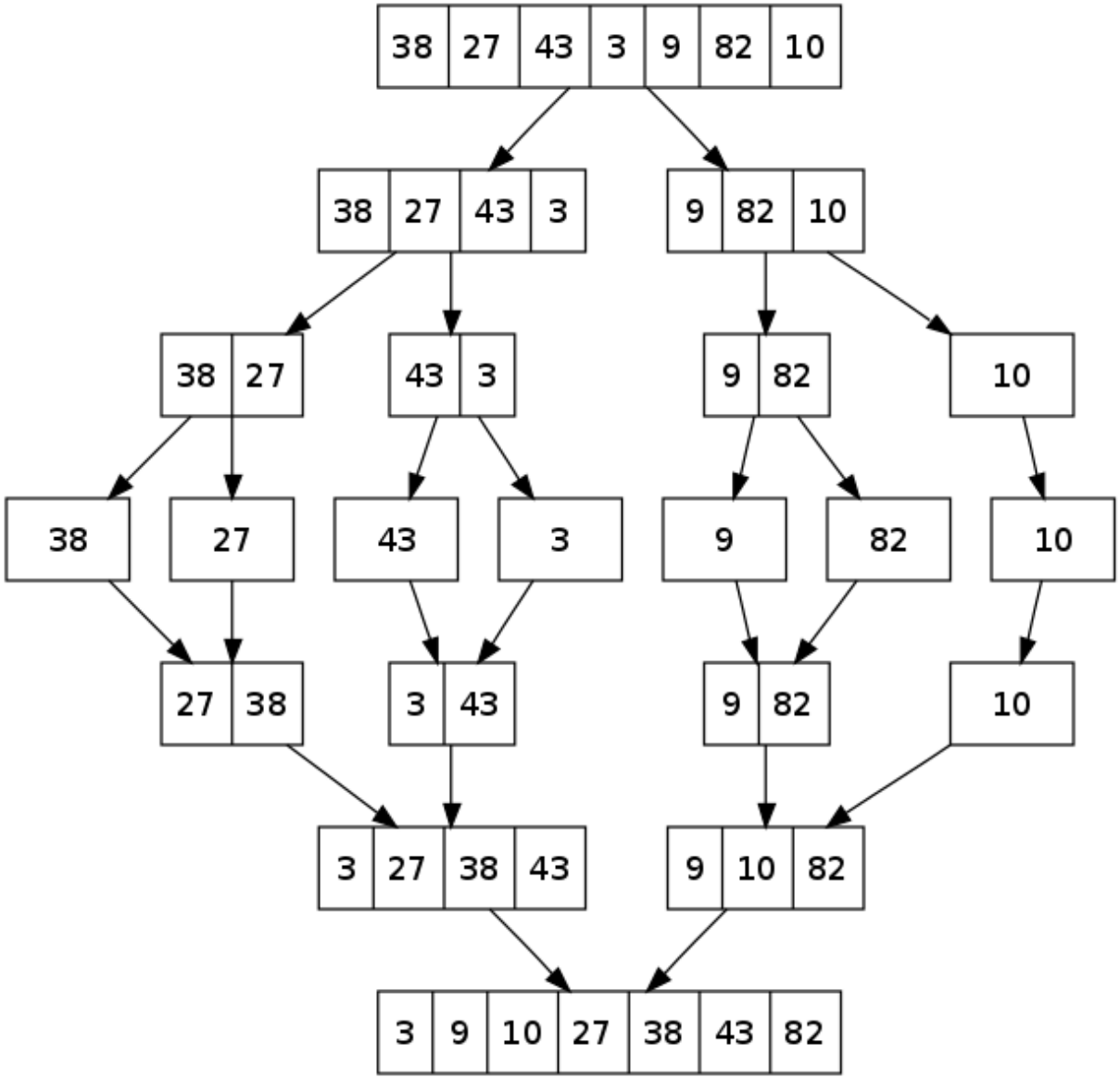
ผลลัพธ์



Time :: Best: O(n log n) Worst: O(n)

Average: O(n log n)

ลักษณะการดำเนินการของ Merge Sort



บทที่ 9-2 Sorting

บทเรียนย่อย

9.8 Shell Sort Algorithm

9.9 Shell Sort Implementation

9.10 Merge Sort Algorithm

9.11 Merge Sort Implementation

9.12 Quick Sort Algorithm

9.13 Quick Sort Implementation

9.11 Merge Sort Implementation

```
func mergesort( var a as array )
     if ( n == 1 ) return a

     var l1 as array = a[0] ... a[n/2]
     var l2 as array = a[n/2+1] ... a[n]

     l1 = mergesort( l1 )
     l2 = mergesort( l2 )

     return merge( l1, l2 )
end func
```

9.11 Merge Sort Implementation

```
func merge( var a as array, var b as array )
     var c as array

     while ( a and b have elements )
          if ( a[0] > b[0] )
               add b[0] to the end of c
               remove b[0] from b
          else
               add a[0] to the end of c
               remove a[0] from a
     while ( a has elements )
          add a[0] to the end of c
          remove a[0] from a
     while ( b has elements )
          add b[0] to the end of c
          remove b[0] from b
     return c
end func
```

บทที่ 9-2 Sorting

บทเรียนย่อย

9.8 Shell Sort Algorithm

9.9 Shell Sort Implementation

9.10 Merge Sort Algorithm

9.11 Merge Sort Implementation

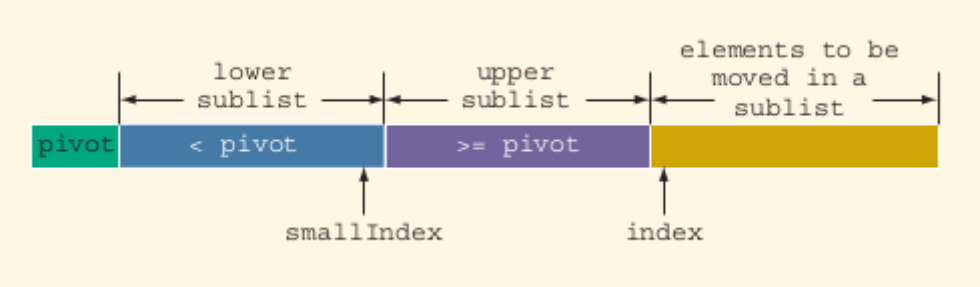
9.12 Quick Sort Algorithm

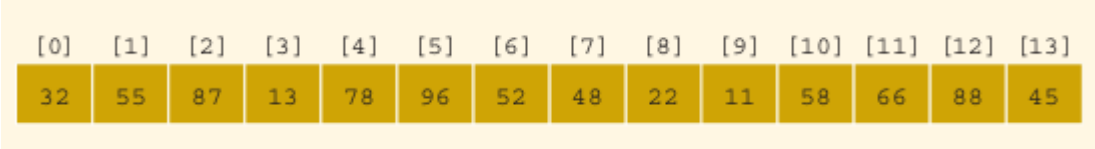
9.13 Quick Sort Implementation

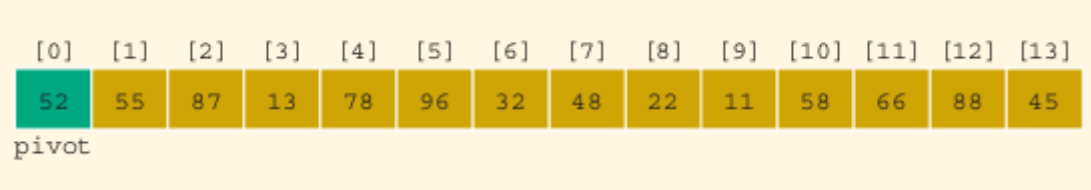
9.12 Quick Sort Algorithm

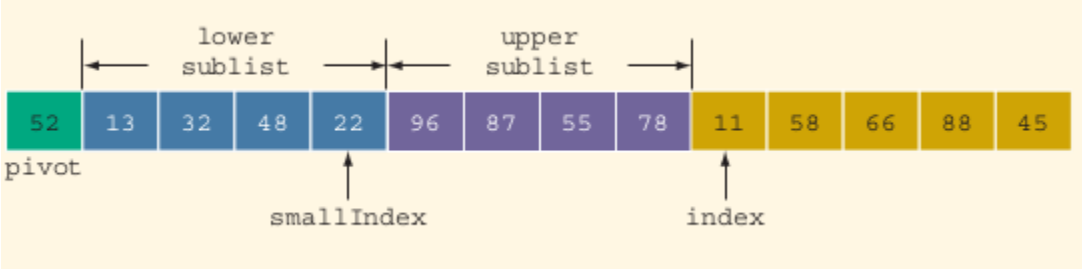
Quick Sort เป็นวิธีการเรียงลำดับที่อาศัยการทำงานแบบ divide and conquer โดยแบ่งข้อมูลออกเป็นสองชุดด้วยวิธีแบ่งส่วน (partition) โดยให้มีข้อมูล ฝั่งซ้าย ที่มีค่าข้อมูลน้อยกว่า ค่าที่เลือก (pivot) และฝั่งขวาเป็นฝั่งที่มีข้อมูลที่มีค่ามากกว่า ในการหาค่า pivot นิยมใช้ วิธีมัธยฐานจากข้อมูลสามตัว (median of three) [ค่าที่ตำแหน่ง((left + right) / 2)]

ลักษณะการดำเนินการของ Quick Sort

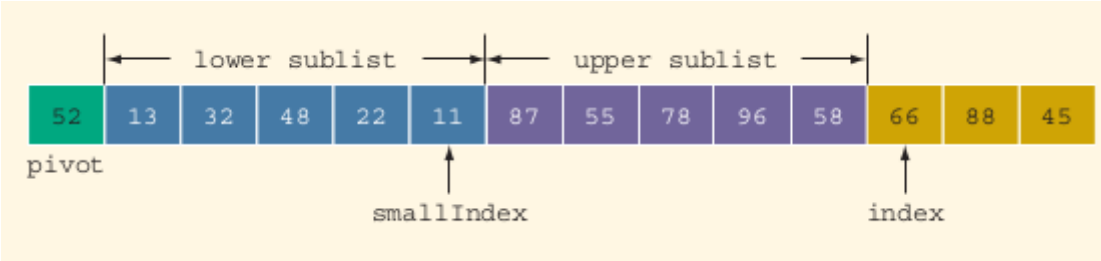


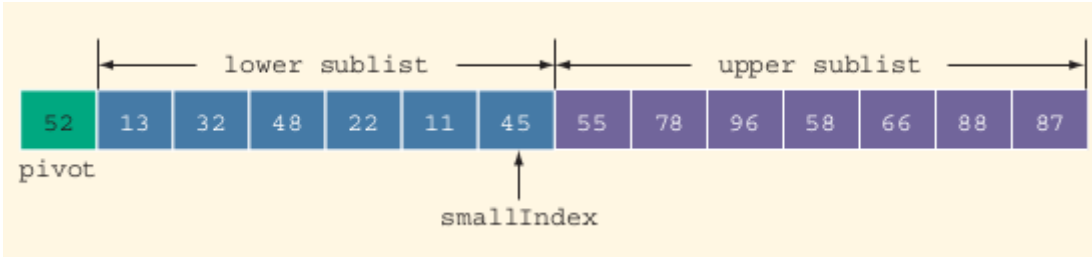


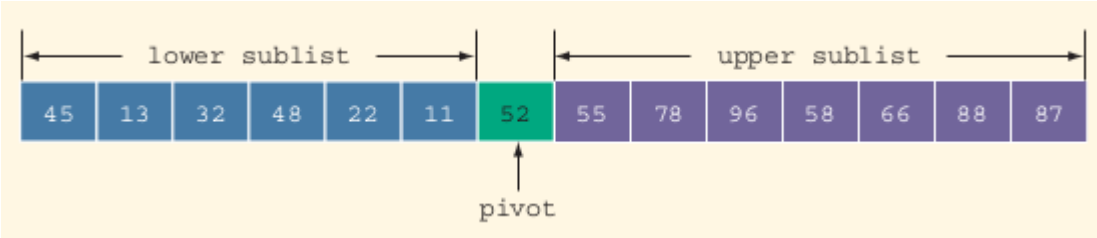




ลักษณะการดำเนินการของ Quick Sort







บทที่ 9-2 Sorting

บทเรียนย่อย

9.8 Shell Sort Algorithm

9.9 Shell Sort Implementation

9.10 Merge Sort Algorithm

9.11 Merge Sort Implementation

9.12 Quick Sort Algorithm

9.13 Quick Sort Implementation

9.13 Quick Sort Implementation

```
void quickSort(int arr[], int left, int right) {
      int i = left, j = right;
      int tmp;
      int pivot = arr[(left + right) / 2];

      /* partition */
      while (i <= j) {
            while (arr[i] < pivot)
                  i++;
            while (arr[j] > pivot)
                  j--;
            if (i <= j) {
                  tmp = arr[i];
                  arr[i] = arr[j];
                  arr[j] = tmp;
                  i++;
                  j--;
            }
      };

      /* recursion */
      if (left < j)
            quickSort(arr, left, j);
      if (i < right)
            quickSort(arr, i, right);
}
```