External Merge Sort

เสนอ

อาจารย์สุรีย์พัชร มุสิกะภวัต

สมาชิก

63102105112 นายอัครพล พิกุลศรี

63102105136 นายสิทธิพร วงศ์บาตร

63102105140 นายชลสิทธิ์ สีสถาน

63102105141 นายธนวัฒน์ สารินทร์

External Sort

การเรียงลำดับภายนอก คือ อัลกอรึธึมการเรียงลำดับที่สามารถ**จัดการข้อมูล จำนวนมหาศาลได้** เมื่อข้อมูลจัดเรียงไม่พอดี (fit) กับ**หน่วยความจำหลัก**ของคอมพิวเตอร์ (RAM) และจะต้องอยู่ในหน่วยความจำภายนอกที่ช้ากว่า (Hard drive) แทน

การเรียงลำดับภายนอก**มักใช้อัลกอริทึมแบบไฮบริด**ในการเรียงลำดับส่วนของข้อมูลที่มีขนาดเล็ก ข้อมูลจะถูกใส่ลงในหน่วยความจำหลัก จะถูกอ่าน จัดเรียง และเขียนลงในไฟล์ชั่วคราว ซึ่งในขั้นตอนการรวม**ไฟล์ย่อย**ที่เรียงลำดับจะถูก**รวมเป็นไฟล์ใหญ่**ไฟล์เดียว

ขั้นตอนวิธีไฮบริดเป็นอัลกอริทึมที่รวมสองหรือขั้นตอนวิธีการอื่น ๆ ที่แก้ปัญหาเดียวกันเลือกหนึ่งอย่างใดอย่างหนึ่ง

(ขึ้นอยู่กับข้อมูล) หรือสลับระหว่างพวกเขาในช่วงเวลาของขั้นตอนวิธี โดยทั่วไปจะทำเพื่อรวมคุณลักษณะที่ต้องการของแต่ละรายการ เพื่อให้อัลกอริธึมโดยรวมดีกว่าส่วนประกอบแต่ละรายการ

รายการเรียงลำดับ merge sort

จะเป็นลักษณะ การตัด array ออกเป็น 2 ส่วนใน แต่ละส่วนก็จะเอาไป recursion ตัดออกเป็นชิ้นย่อย ๆ ลงไปอีกจนเหลือขนาดเล็ก ที่สามารถ sort ได้

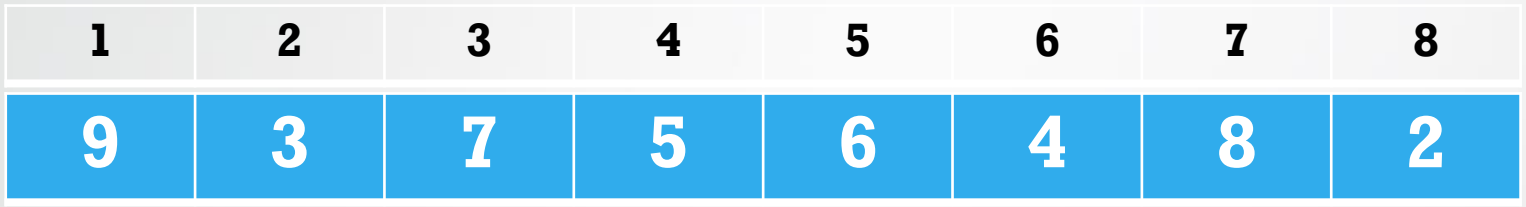
ก็จะจัดการ sort ชิ้นเล็กๆ ให้เสร็จแล้วค่อยนำชิ้นเล็กๆ ที่ sort เสร็จมาต่อกันอีกที

สมมติ **list Alpha** ขึ้นมา ขนาด **8** มี **element** ดังภาพ

List Alpha

1	2	3	4	5	6	7	8	index
9	3	7	5	6	4	8	2	element

List Alpha



low

height

start

end

ใช้ 2 ตัวแปรเป็น พารามิเตอร์คือ low, height

List Alpha

1	2	3	4	5	6	7	8	index
9	3	7	5	6	4	8	2	element

low

height

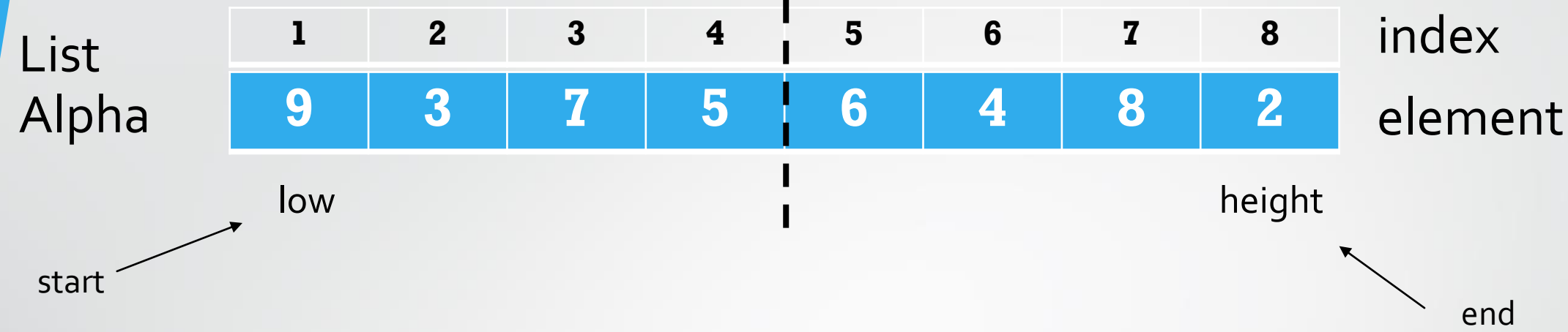
start

end

ใช้ 2 ตัวแปรเป็น พารามิเตอร์คือ **low, height**

ได้อัลกอริทึมดังนี้

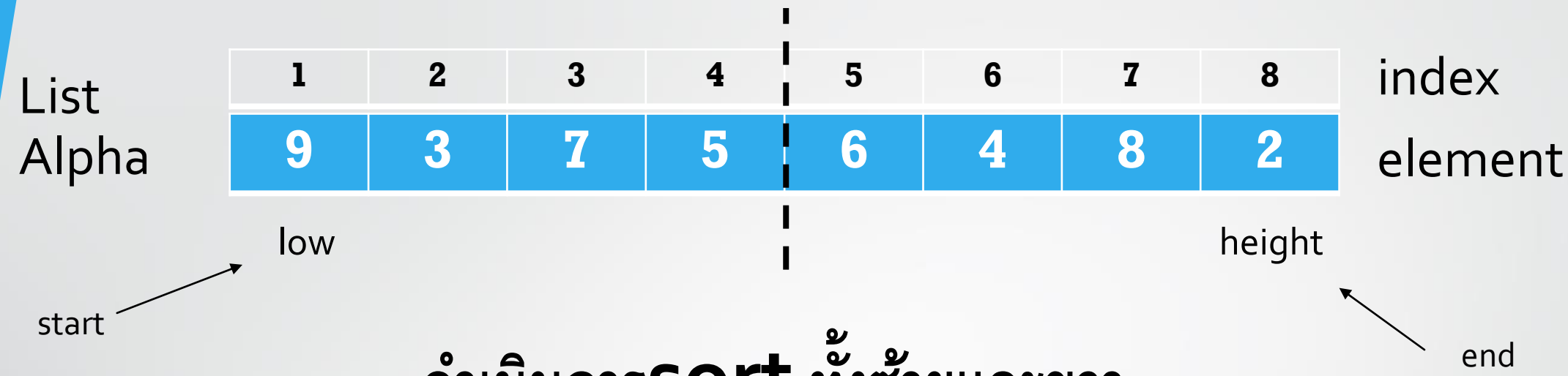
1. Algorithm merge sort(low, height)
2. if (low < heigh):
3. mid = (low + height)/2
4. merge sort(low, mid)
5. merge sort(mid + 1, height)
6. merge (low, mid, height)



มองเหมือนว่า**list**ข้างต้นเป็นปัญหาที่ใหญ่มาก เราต้องการที่จะแก้ เราจึงแบ่งเป็นปัญหาย่อยๆ โดยหาค่า **mid** นั่นเอง

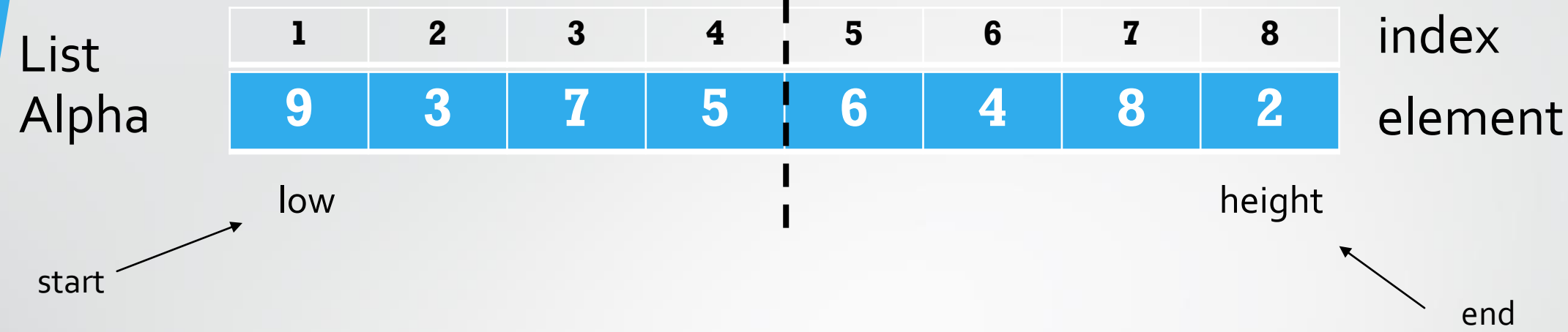
1. Algorithm merge sort(low, height)
2. if (low < heigh):
3. **mid = (low + height)/2**
4. merge sort(low, mid)
5. merge sort(mid + 1, height)
6. merge (low, mid, height)

// sample (1+8)/2 = 4.5 หมายถึง **index** ที่ **4 element = 5**

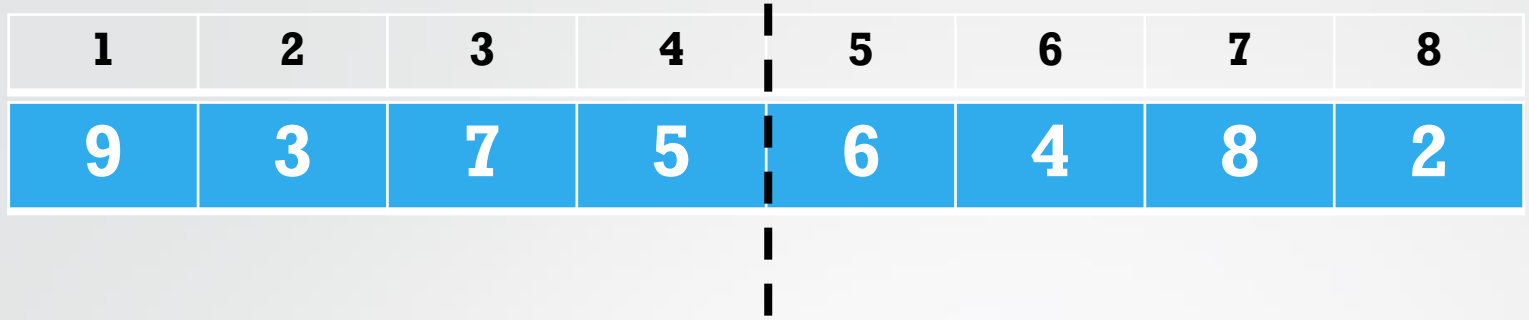


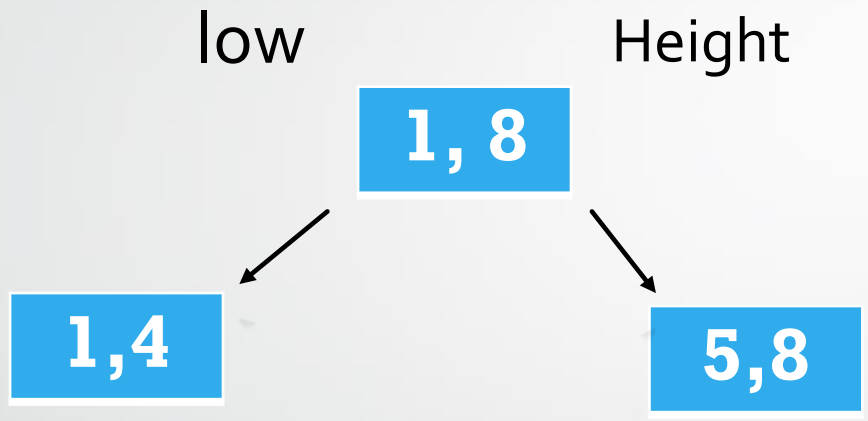
ดำเนินการsort ทั้งซ้ายและขวา

1. Algorithm merge sort(low, height)
2. if (low < heigh):
3. mid = (low + height)/2 // sample (1+8)/2 = 4.5 หมายถึง index ที่ 4 element = 5
4. **merge sort(low, mid)** // จาก low คือ ด้าน ซ้าย ไปถึงตรงกลาง
5. **merge sort(mid + 1, height)** // จาก mid + 1 คือ index 5 element = 6 เป็นต้นไป
6. merge (low, mid, height) จนถึงตำแหน่ง height



1. Algorithm merge sort(low, height)
2. if (low < heigh):
3. mid = (low + height)/2
4. merge sort(low, mid) // จาก low คือ ด้าน ซ้าย ไปถึงตรงกลาง
5. merge sort(mid + 1, height) // จาก mid + 1 คือ index 5 element = 6 เป็นต้นไปจนถึงตำแหน่ง height
6. **merge (low, mid, height)** // เมื่อทำการเรียงลำดับฝั่งซ้ายและขวา เราจะทำการ merge เข้าด้วยกัน



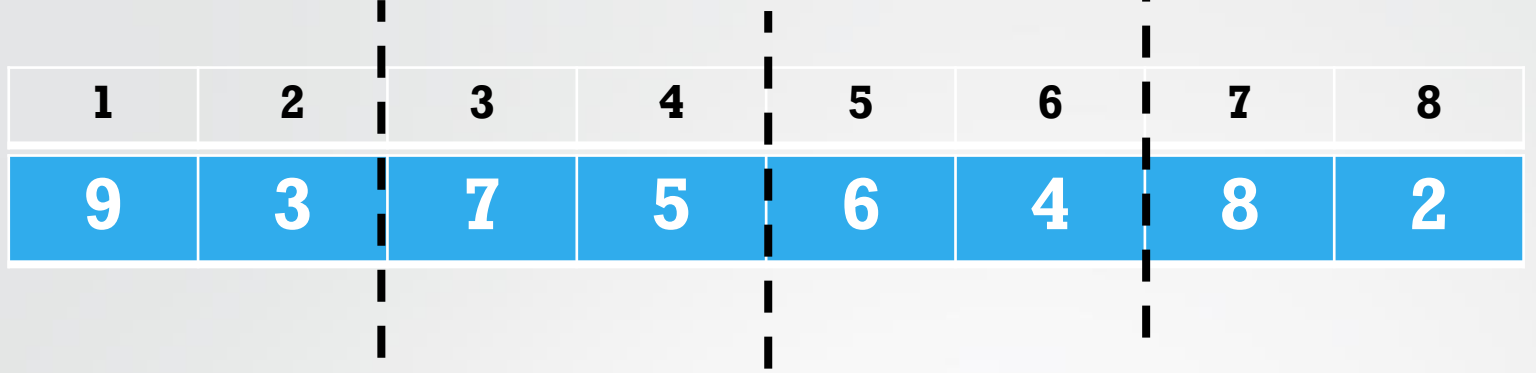


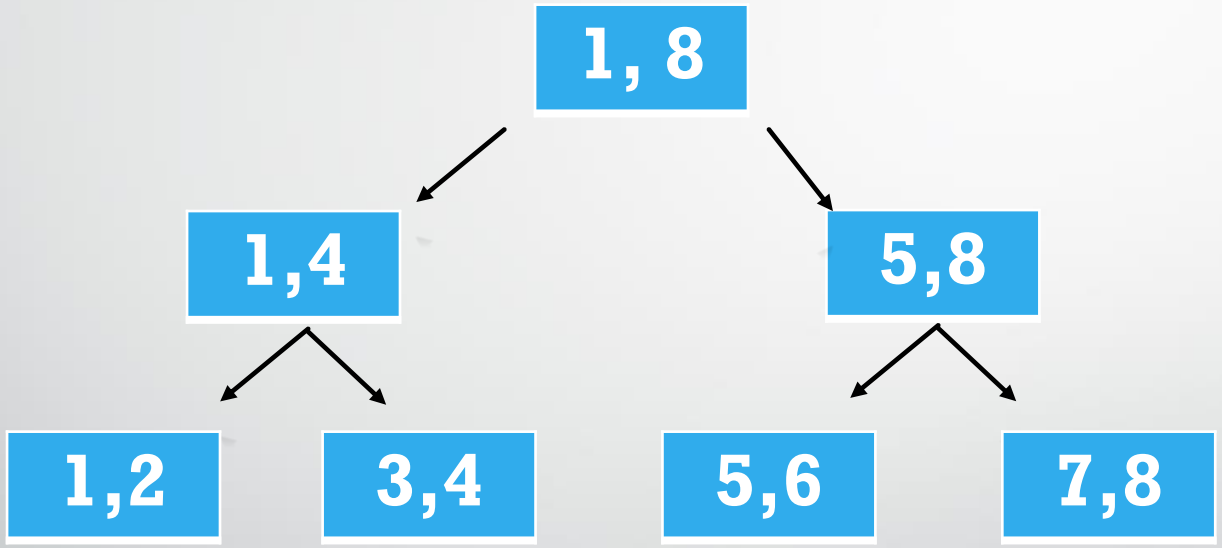

```
1. Algorithm merge sort(low, height)
2.     if (low < heigh):
3.         mid = (low + height)/2
4.         merge sort(low, mid)
5.         merge sort(mid + 1, height)
6.         merge (low, mid, height)
```

Low < height ?

Yes

True !



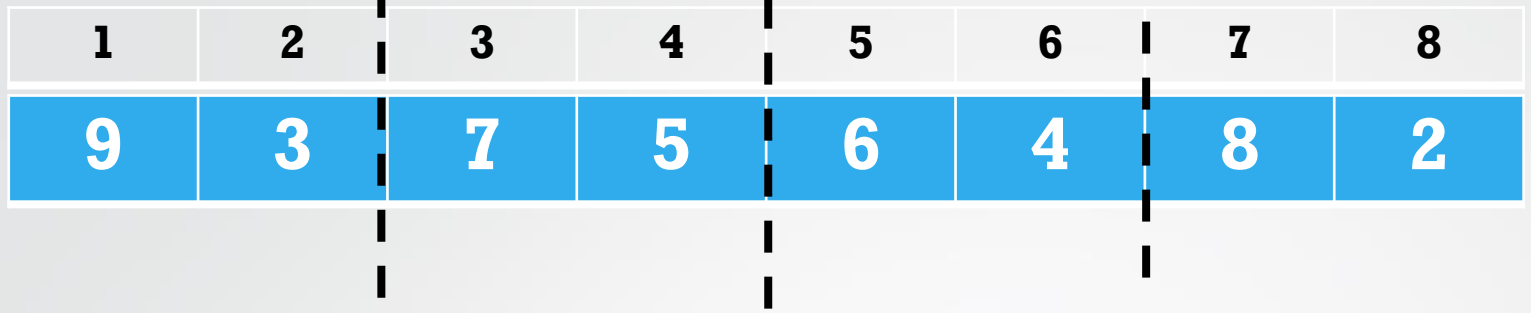


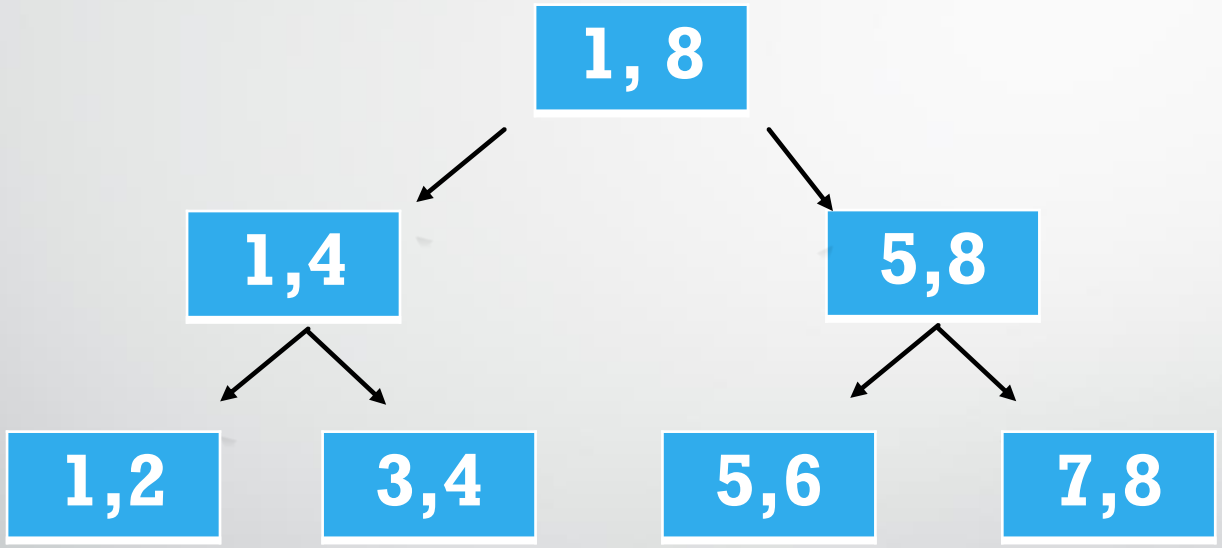

```
1. Algorithm merge sort(low, height)
2.     if (low < heigh):
3.         mid = (low + height)/2
4.         merge sort(low, mid)
5.         merge sort(mid + 1, height)
6.         merge (low, mid, height)
```

Low < height ?

Yes

True !





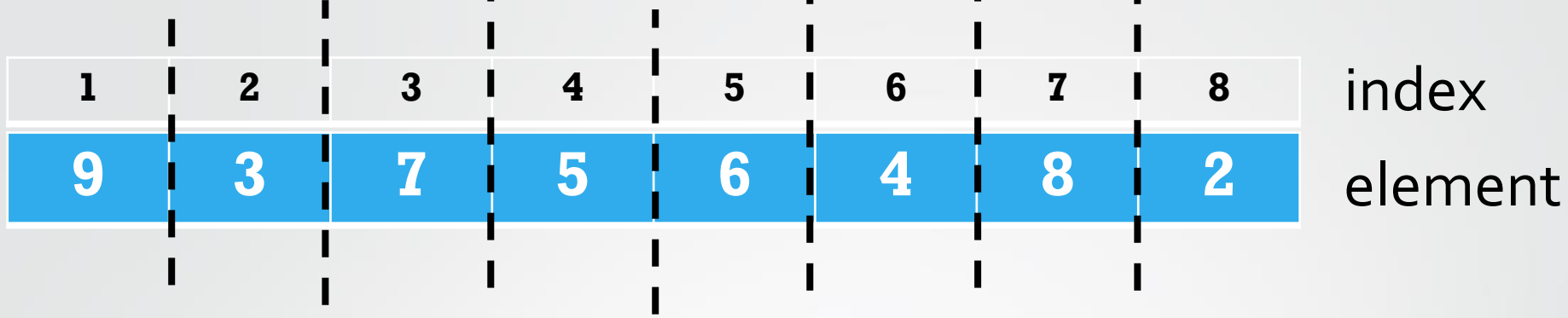

```
1. Algorithm merge sort(low, height)
2.     if (low < heigh):
3.         mid = (low + height)/2
4.         merge sort(low, mid)
5.         merge sort(mid + 1, height)
6.         merge (low, mid, height)
```

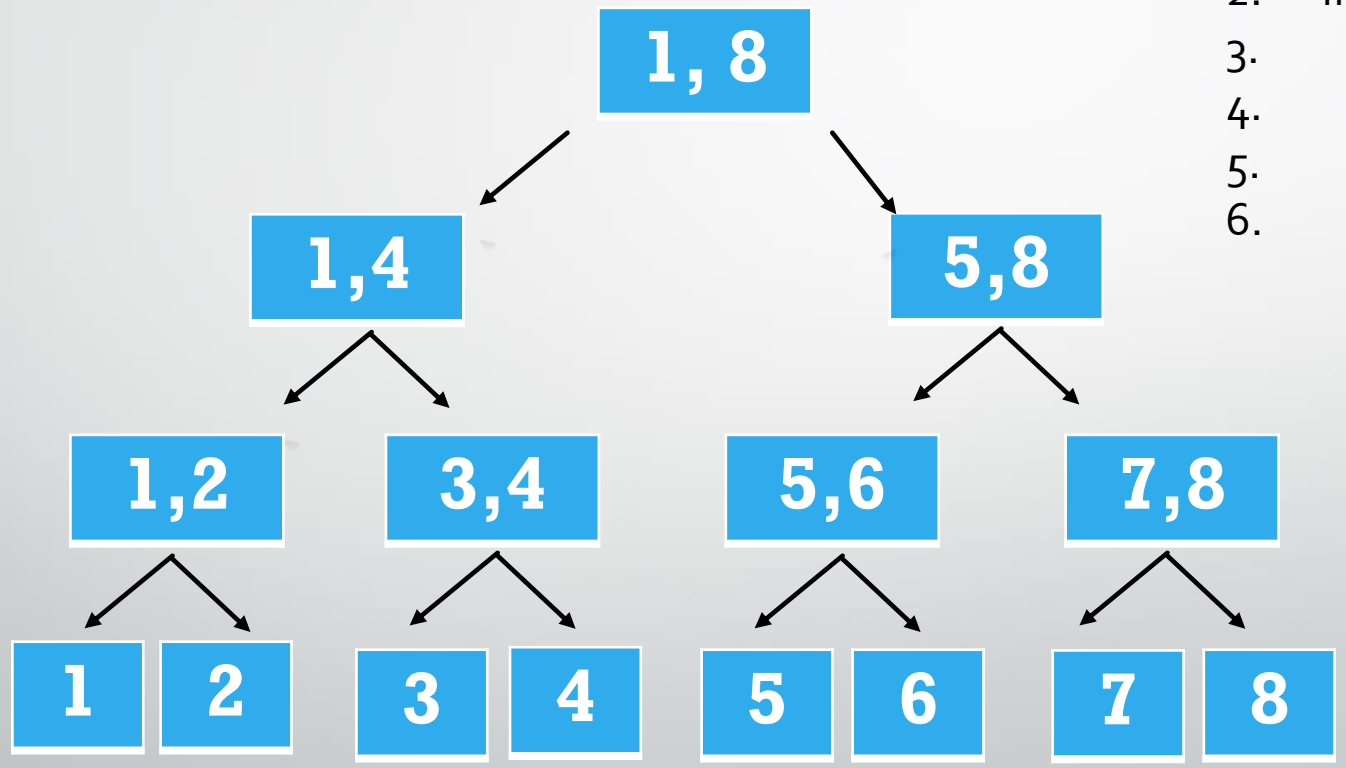
เมื่อเราแบ่งครึ่งแล้ว เนื่องจากยังเข้าเงื่อนไข **low < height** เราจะทำการแบ่งครึ่งอีกครั้ง

Low < height ?

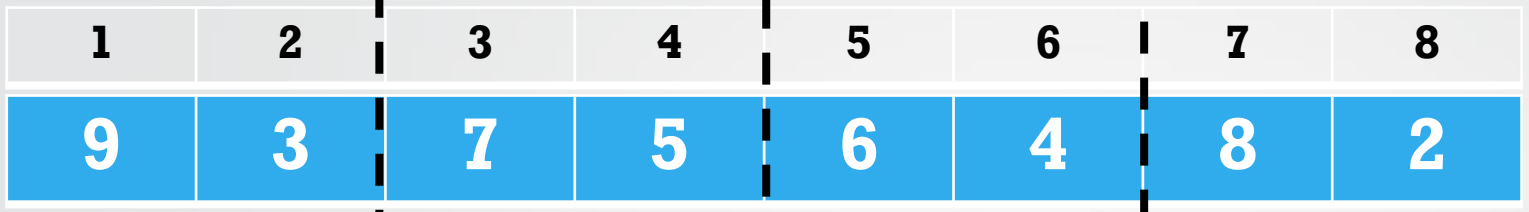
Yes

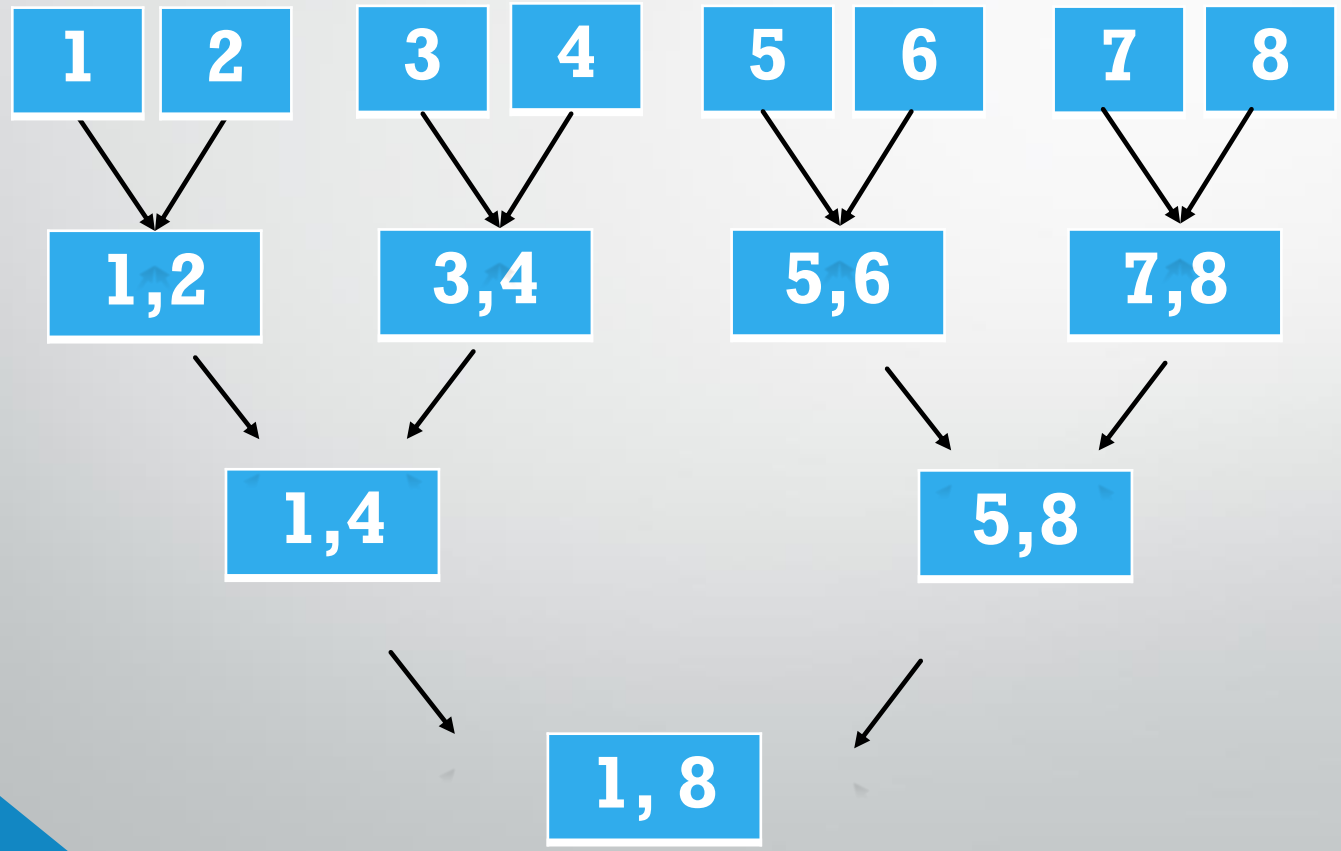
True !



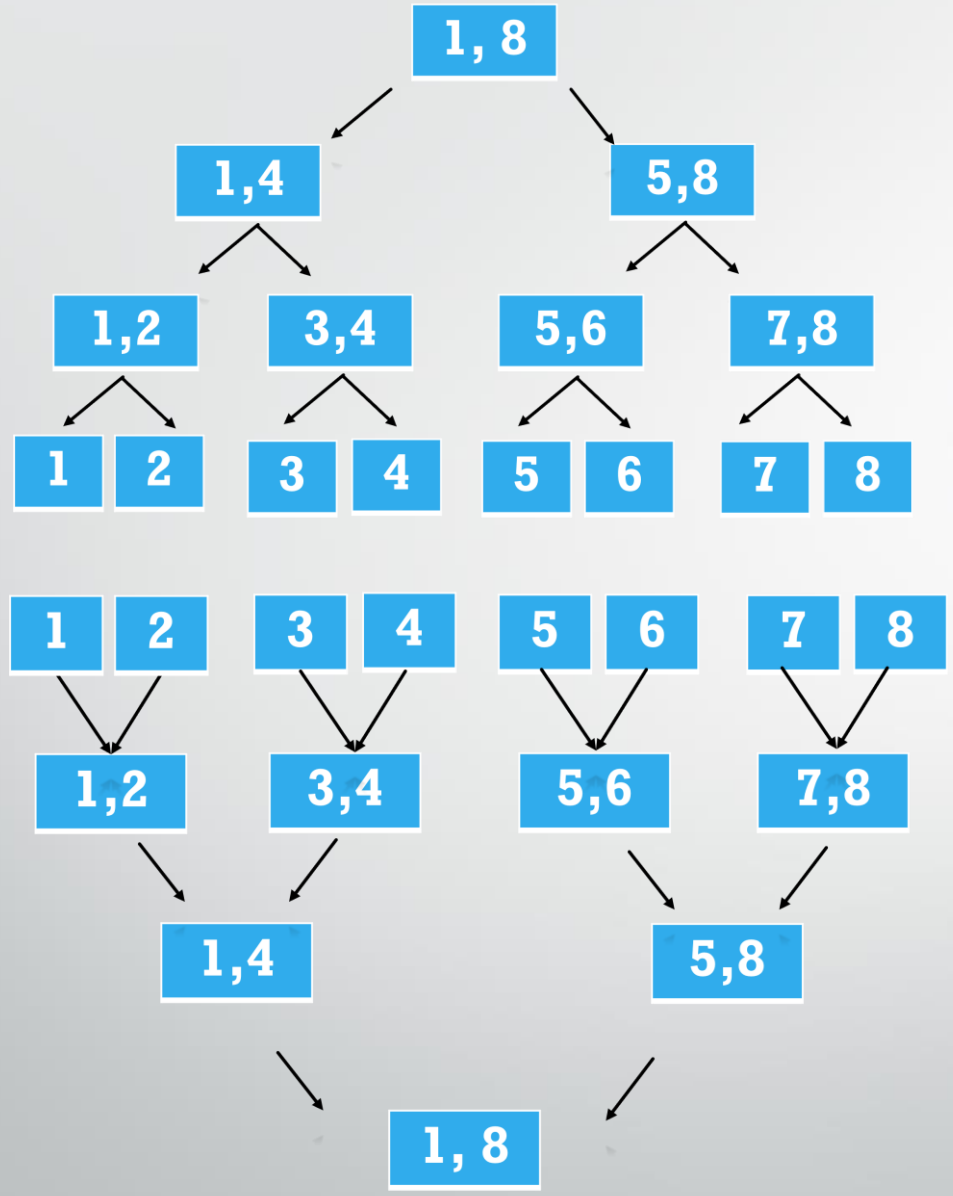


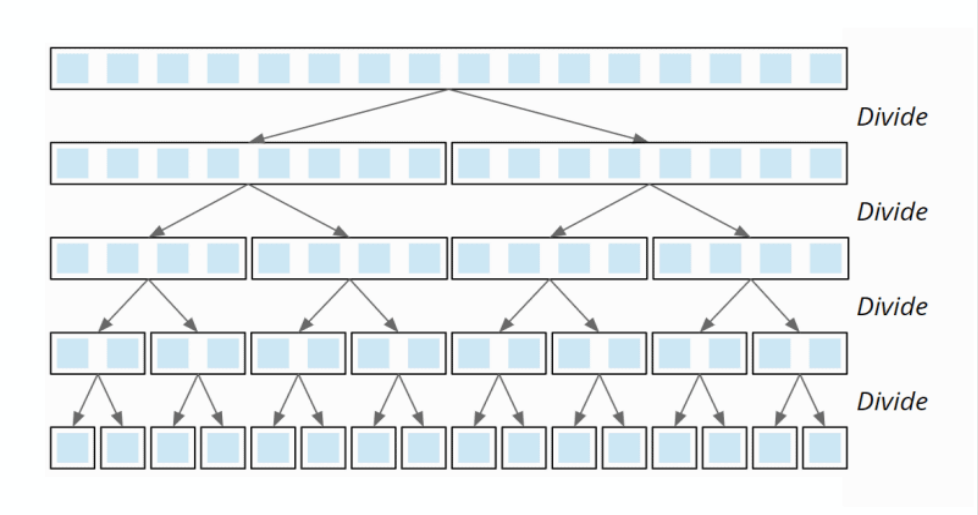

```
1. Algorithm merge sort(low, height)
2.     if (low < heigh):
3.         mid = (low + height)/2
4.         merge sort(low, mid)
5.         merge sort(mid + 1, height)
6.         merge (low, mid, height)
```

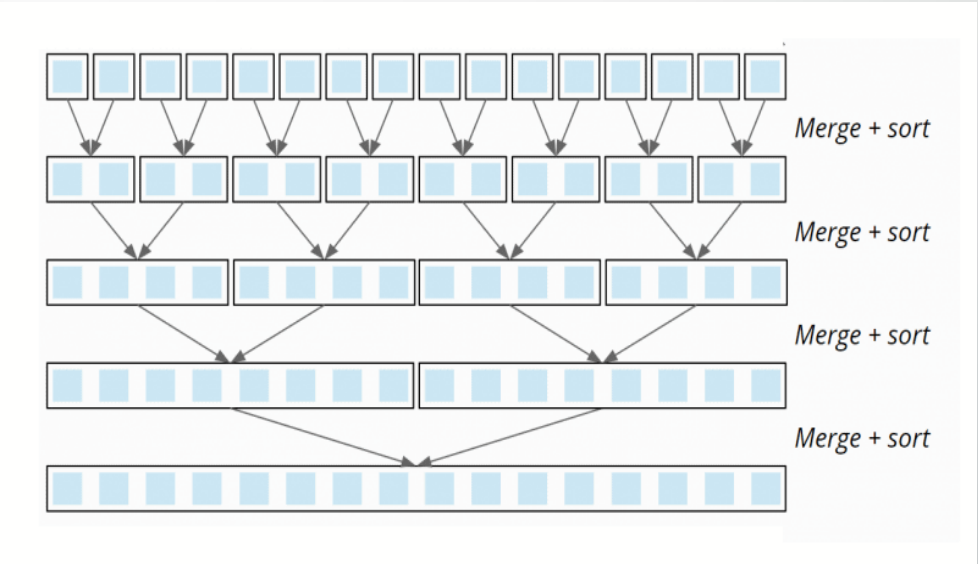


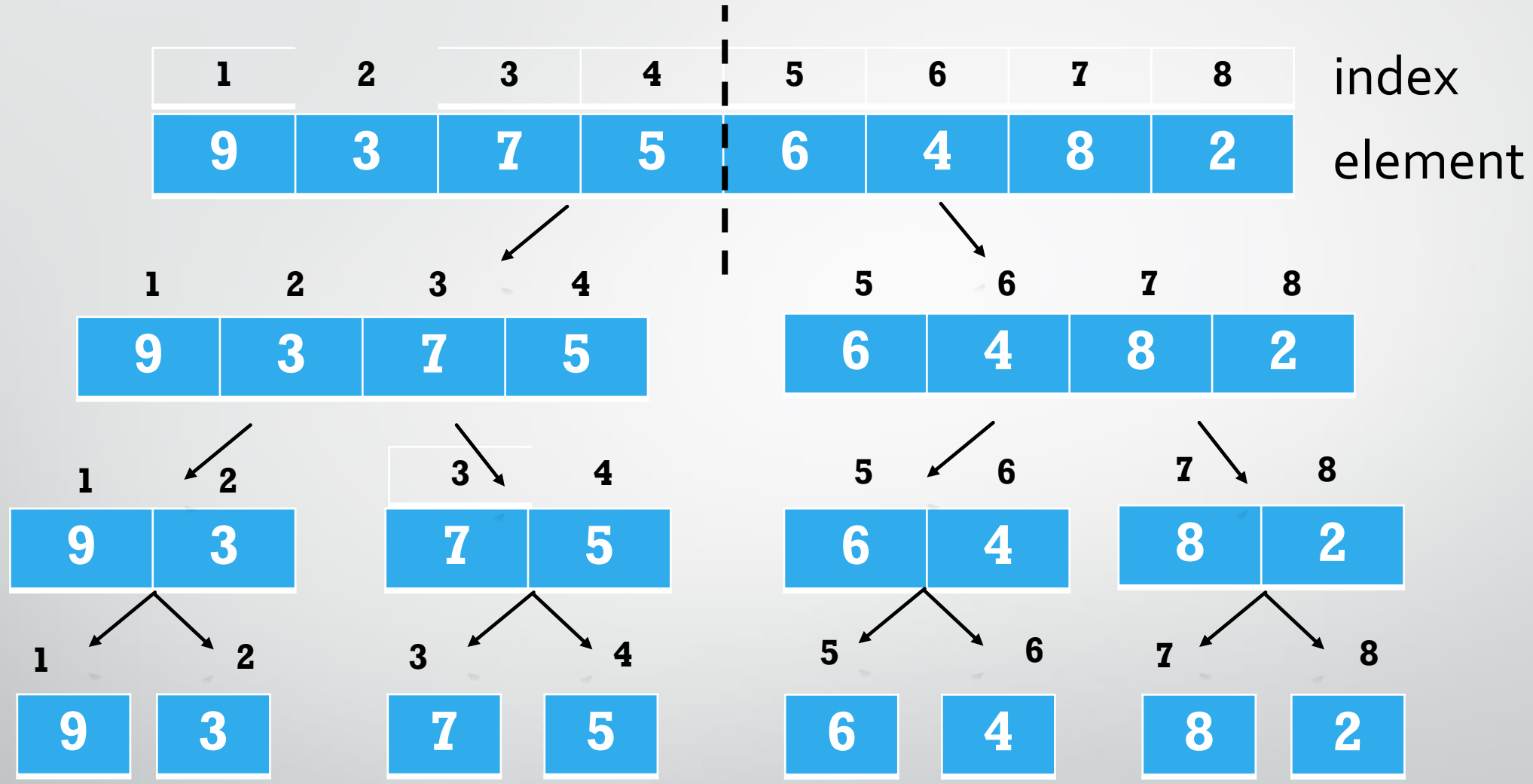



```
1. Algorithm merge sort(low, height)
2.     if (low < heigh):
3.         mid = (low + height)/2
4.         merge sort(low, mid)
5.         merge sort(mid + 1, height)
6.         merge (low, mid, height)
```


1, 8
1,4
5,8
1,2
3,4
5,6
7,8
1
2
3
4
5
6
7
8
1
2
3
4
5
6
7
8
1,2
3,4
5,6
7,8
1,4
5,8
1, 8


Divide
Divide
Divide
Divide


Merge + sort
Merge + sort
Merge + sort
Merge + sort


index
element
1 2 3 4 5 6 7 8
9 3 7 5 6 4 8 2
1 2 3 4
9 3 7 5
5 6 7 8
6 4 8 2
1 2
9 3
3 4
7 5
5 6
6 4
7 8
8 2
1
9
2
3
3
7
4
5
5
6
6
4
7
8
8
2

```
1. Algorithm merge sort(low, height)
2.     if (low < heigh):
3.         mid = (low + height)/2
4.         merge sort(low, mid)
5.         merge sort(mid + 1, height)
6.         merge (low, mid, height)
```