

# เอกสารประกอบการอบรม ส่วนที่ ๑ วิชาโครงสร้างข้อมูล

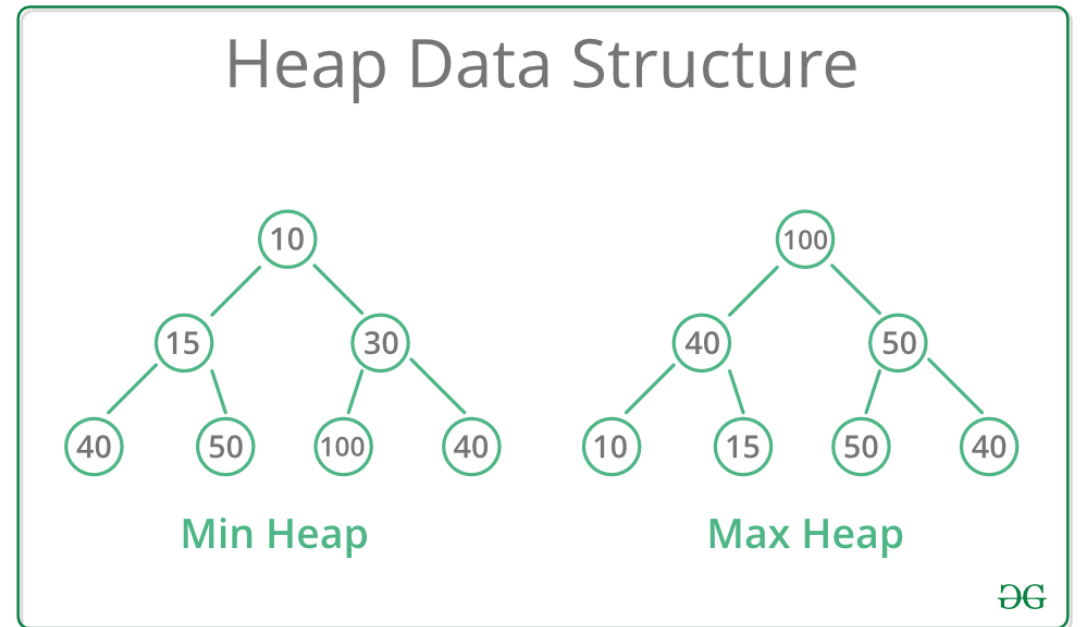
ค่ายคอมพิวเตอร์โอลิมปิก สอวน. ค่าย ๒ ๒/๒๕๖๖  
ศูนย์โรงเรียนสามเสนวิทยาลัย - มหาวิทยาลัยธรรมศาสตร์  
ระหว่างวันที่ ๑๘ มีนาคม - ๓ เมษายน ๒๕๖๗

โดย

สาขาวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์และเทคโนโลยี  
มหาวิทยาลัยธรรมศาสตร์

# Non Linear Data Structure

- Recursion
- Tree
- Binary Tree, Binary Search Tree
- Binary Heaps
- Graph
- Hashing



# Binary Heap





## Priority Queues (Binary Heaps)

- Priority Queue คือโครงสร้างข้อมูลประกอบไปด้วย ๒ operations หลัก คือ การแทรกข้อมูล (insert) และการลบข้อมูลตัวที่น้อยที่สุด (deleteMin)
- โครงสร้างข้อมูลที่สามารถสร้าง priority queue เช่น ลิงค์ลิสต์ อะเรย์ หรือต้นไม้ไบนารี ที่เรียกว่า binary heaps

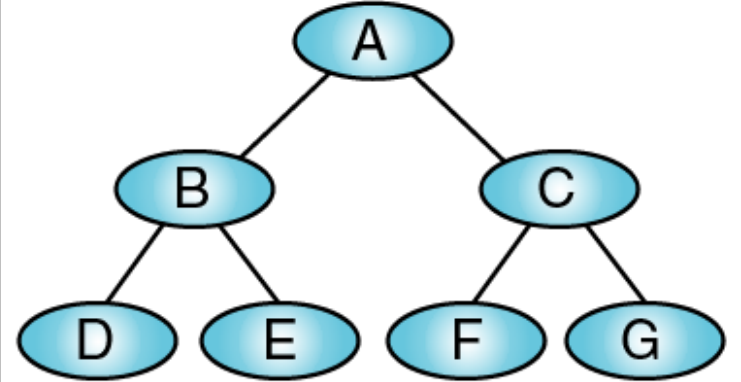
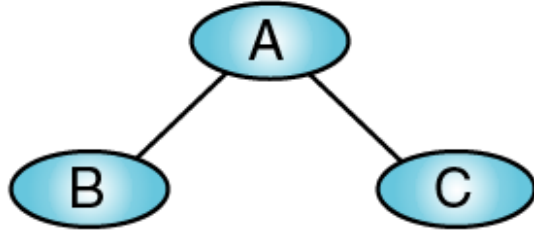
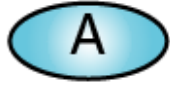




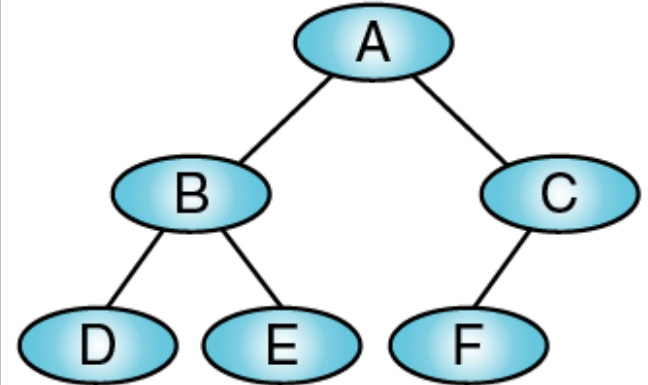
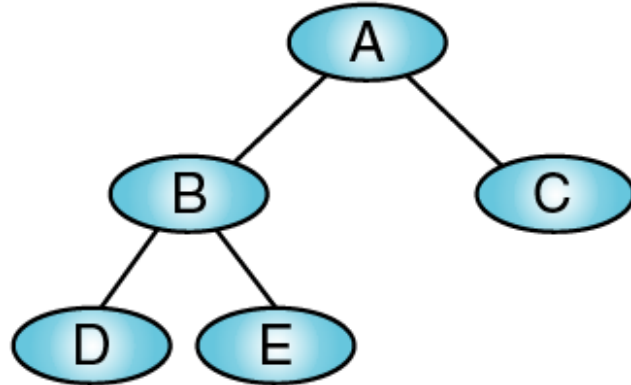
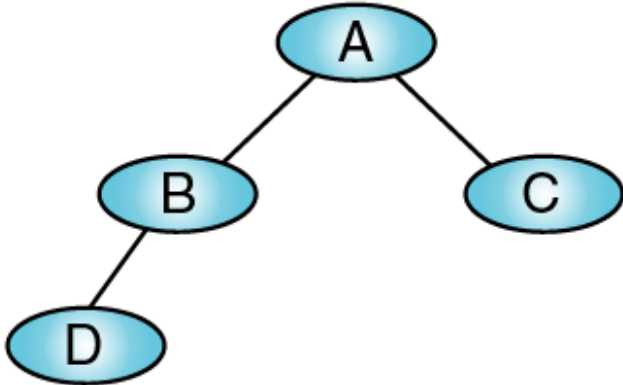
# คุณสมบัติทางด้านโครงสร้าง

---

- Binary Heaps เป็นต้นไม้ไบนารีที่โหนดถูกเติมเต็มลงมาทีละระดับจากซ้ายไปขวา หรือเป็นต้นไม้ไบนารีที่เกือบสมบูรณ์ (nearly complete binary tree)



**(a) Complete trees (at levels 0, 1, and 2)**

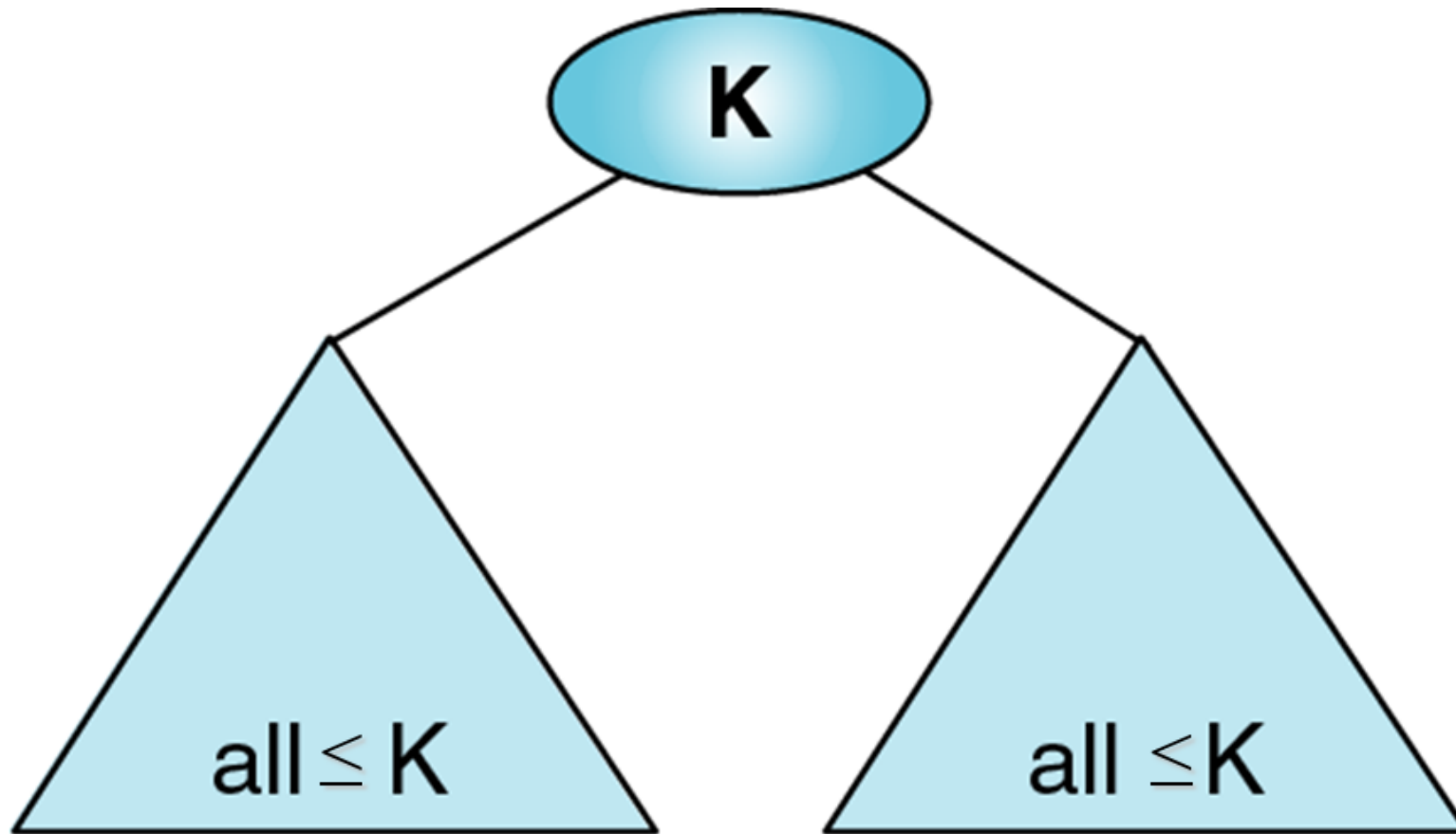


**(b) Nearly complete trees (at level 2)**

## คุณสมบัติทางด้าน ลำดับของข้อมูล

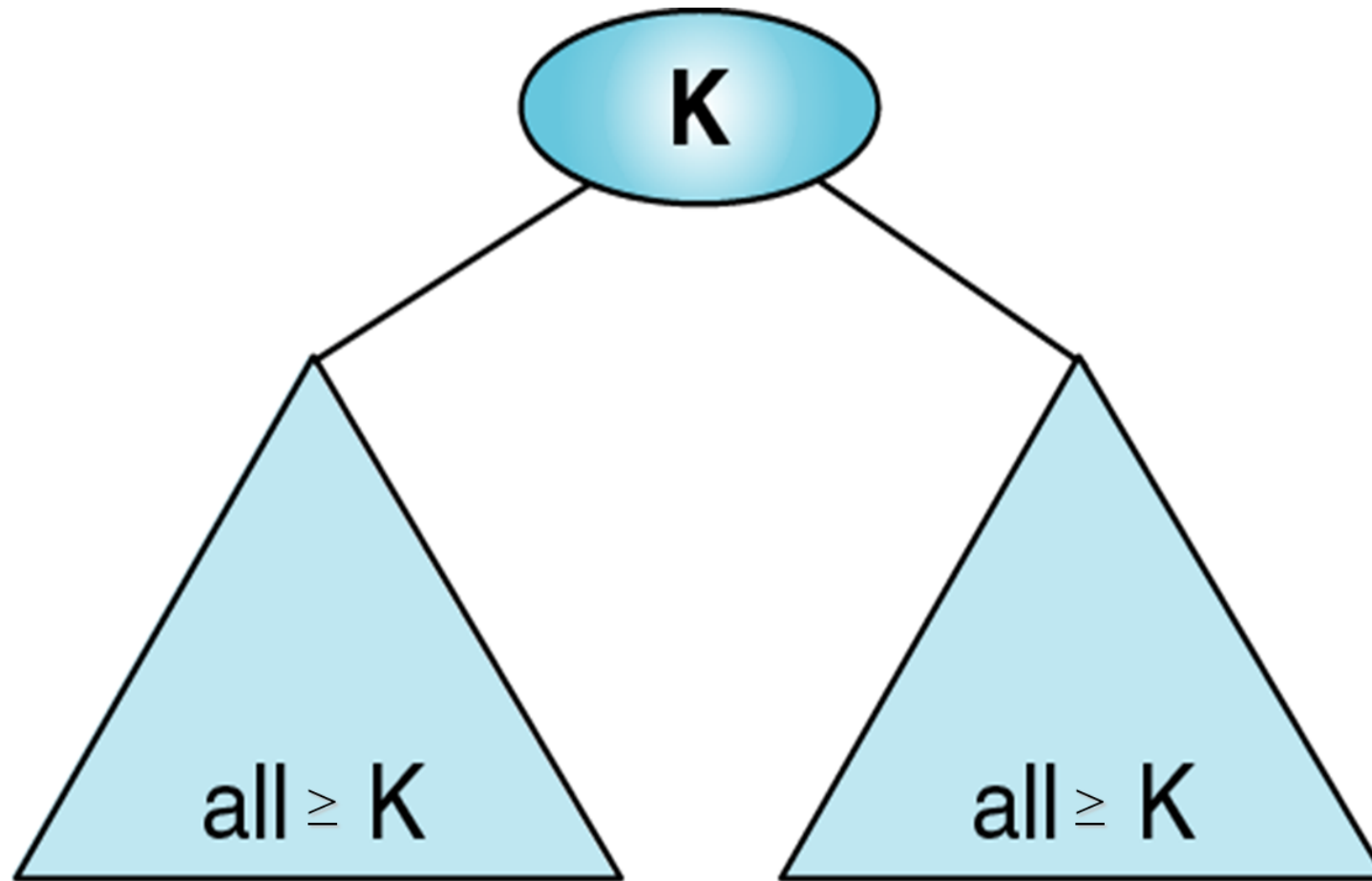
- ข้อมูลใน heaps จะถูกจัดเรียงเพื่อให้การลบข้อมูลได้สะดวก ง่ายและรวดเร็ว โดย
  - ข้อมูลตัวที่เล็กที่สุด จะอยู่ที่โหนดรากหรือ root node และต้นไม้ย่อยต้องมีคุณสมบัติเป็น heaps ด้วยคือ ทุกๆ โหนดจะต้องมีค่าน้อยกว่าโหนดลูกโหนดหลานของตัวเอง  
→ Min heap
  - ข้อมูลตัวที่ใหญ่ที่สุด จะอยู่ที่โหนดรากหรือ root node และต้นไม้ย่อยต้องมีคุณสมบัติเป็น heaps ด้วยคือ ทุกๆ โหนดจะต้องมีค่ามากกว่าโหนดลูกโหนดหลานของตัวเอง  
→ Max heap

# โครงสร้างของ Max-heap

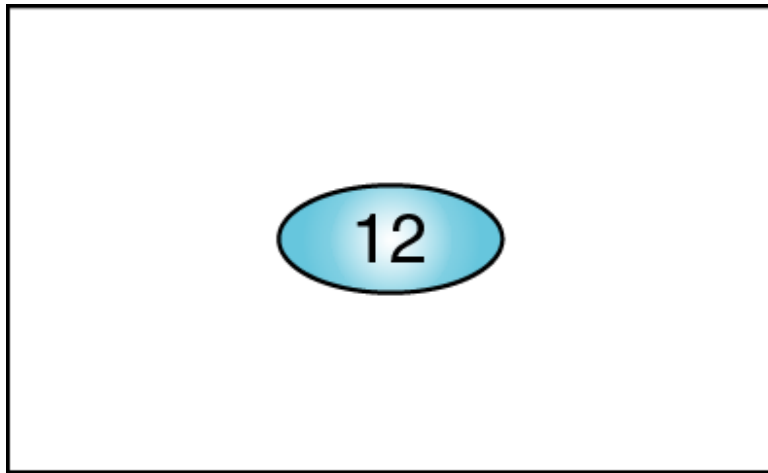




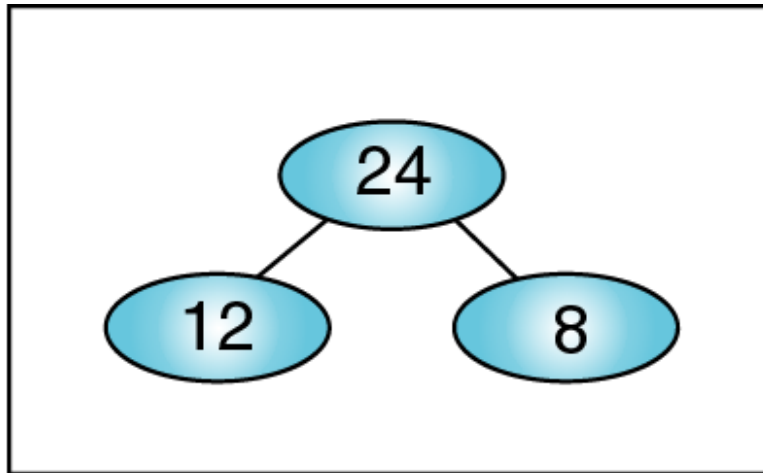
# โครงสร้างของ Min-heap



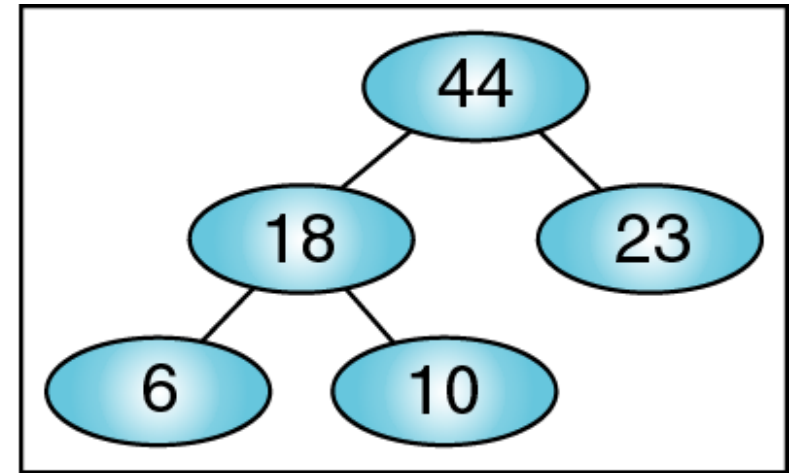
# ตัวอย่าง Heap ที่ถูกต้อง



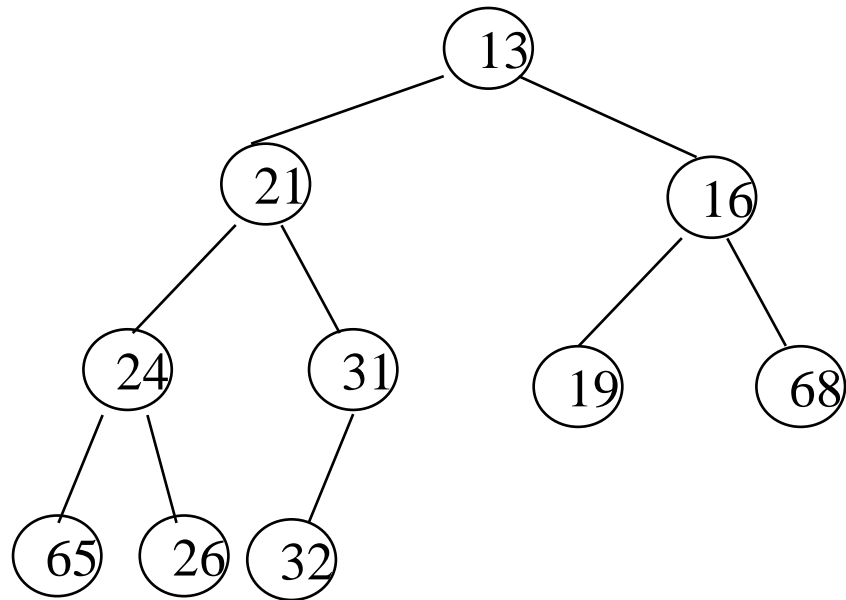
**(a) Root-only heap**



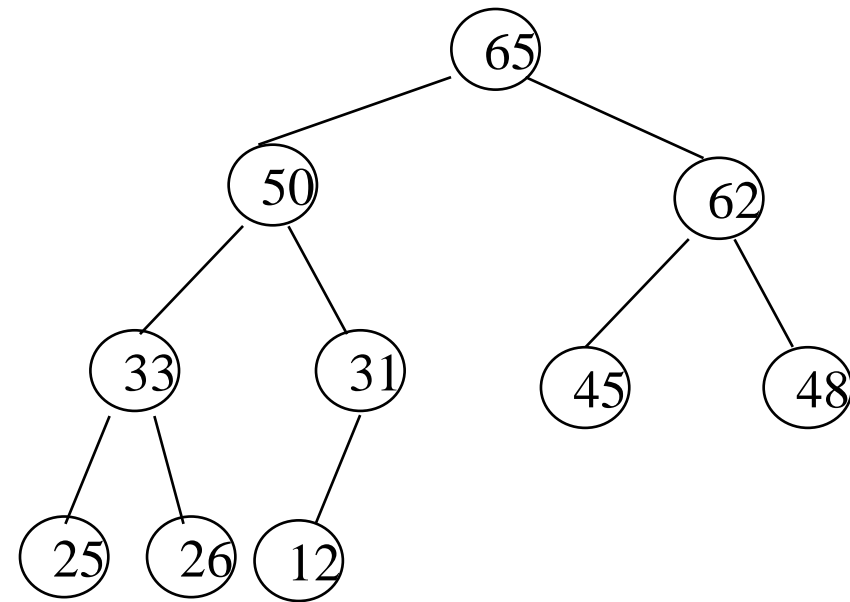
**(b) Two-level heap**



**(c) Three-level heap**

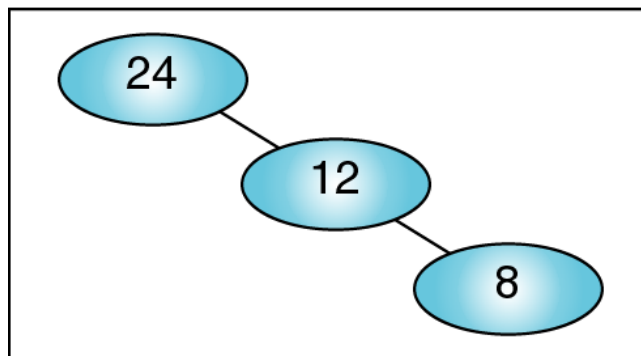


**minHeap**

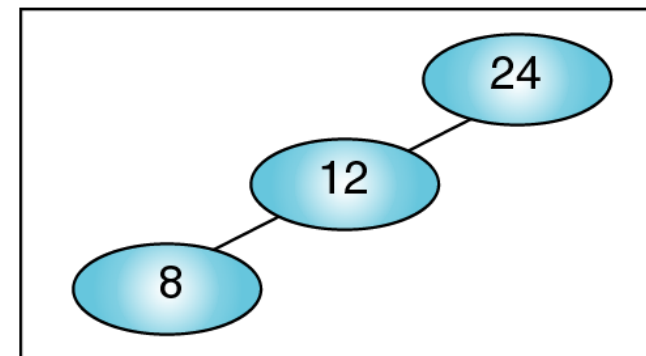


**maxHeap**

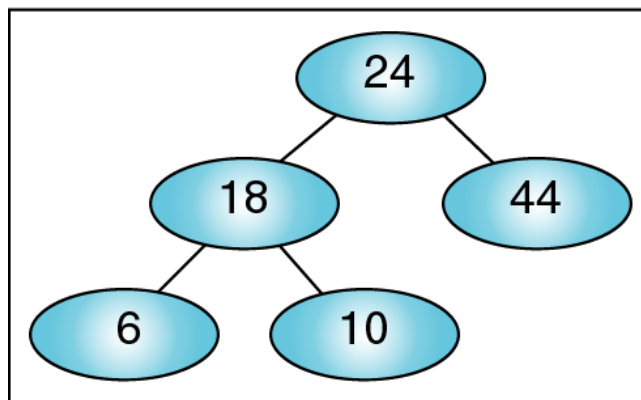
ตัวอย่าง Tree ที่  
ไม่ใช่ Heap



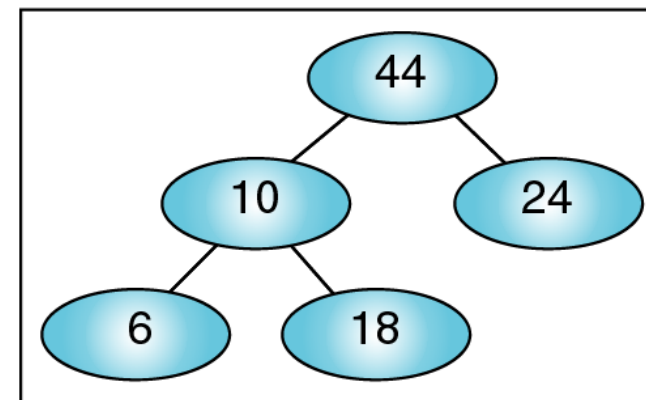
(a) Not nearly complete  
(rule 1)



(b) Not nearly complete  
(rule 1)



(c) Root not largest  
(rule 2)



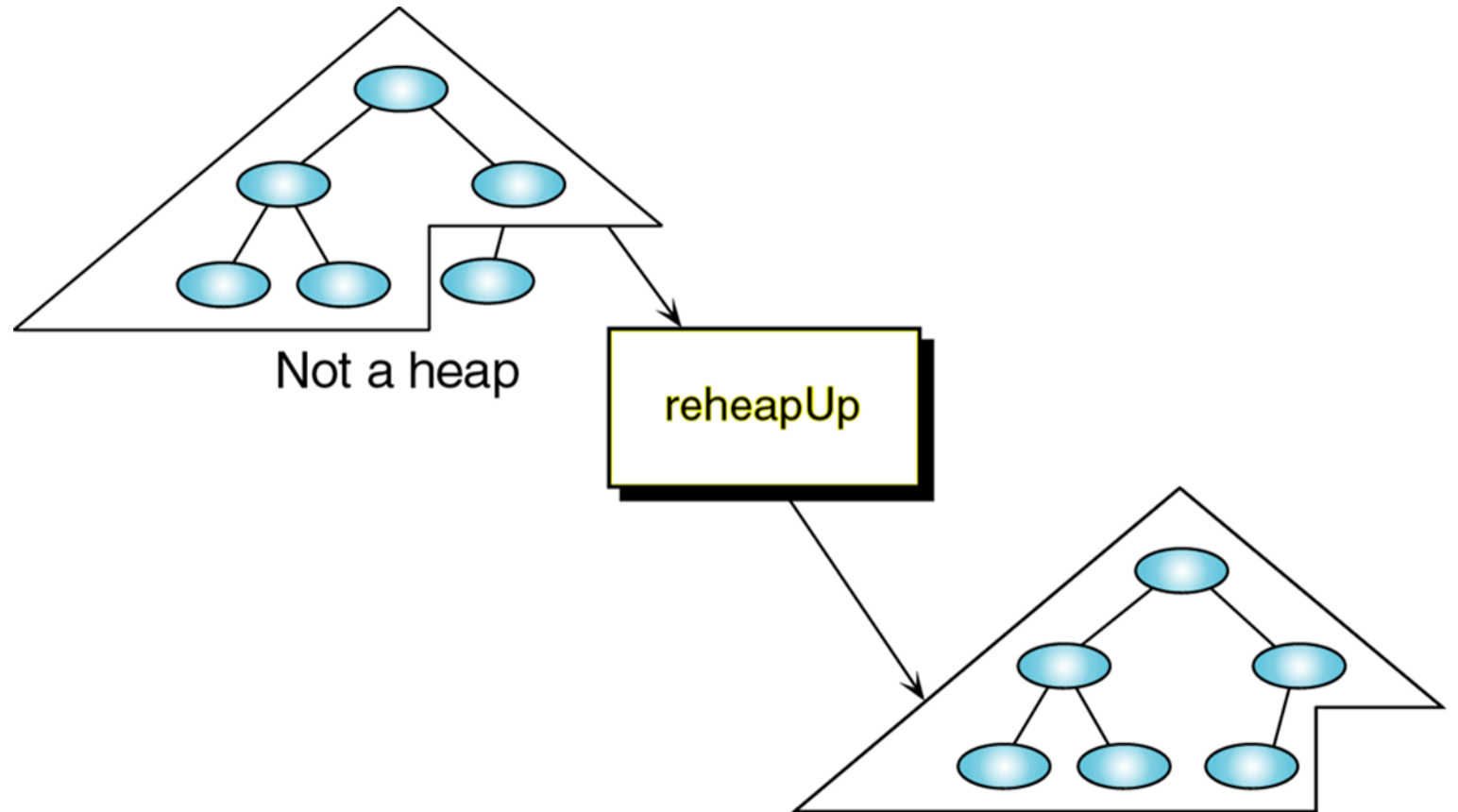
(d) Subtree 10 not a heap  
(rule 2)



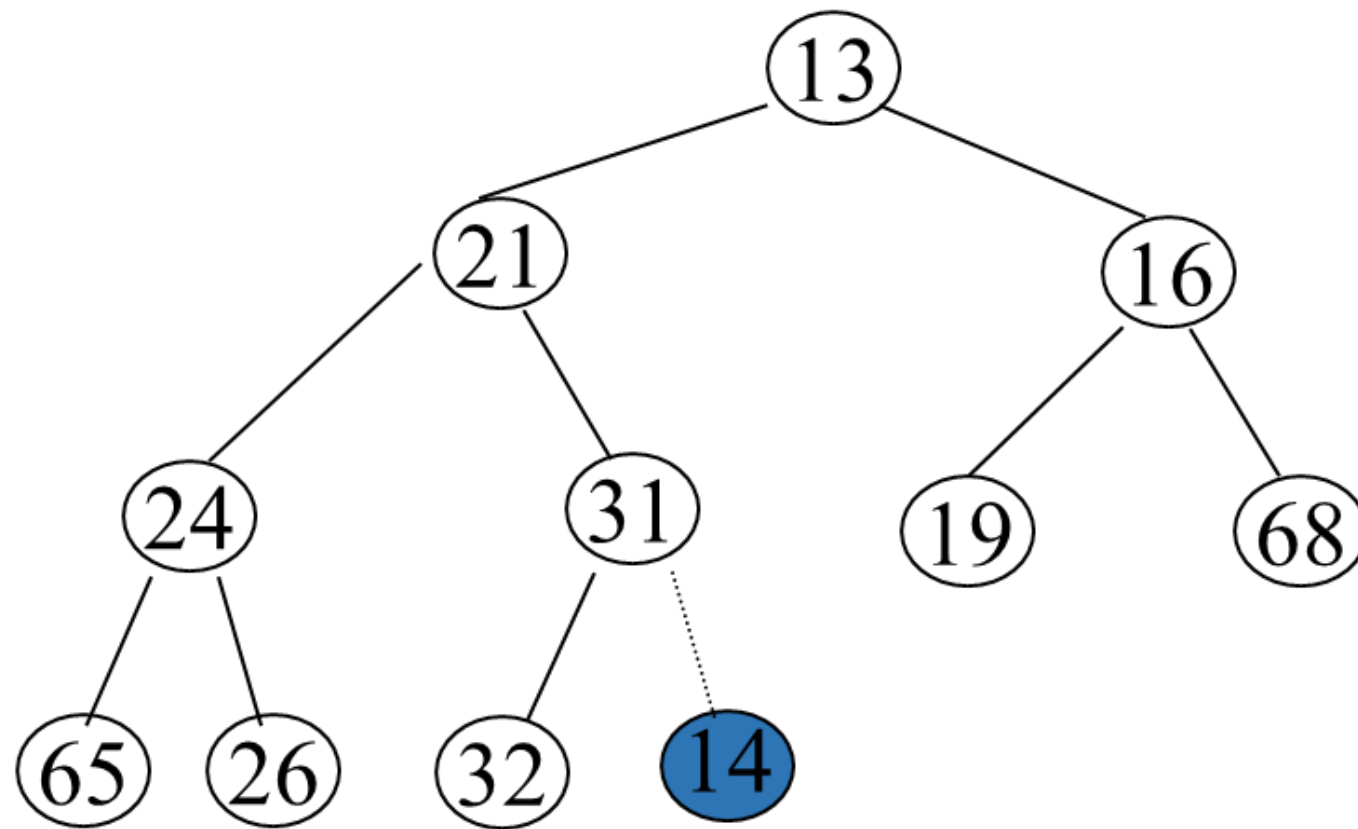
## Heap Operation

- **Insertion:** เพิ่มข้อมูลเข้าไปใน heap
- **Deletion:** ลบข้อมูลออกจาก heap และจัดโครงสร้างให้ยังคงคุณสมบัติของ heap

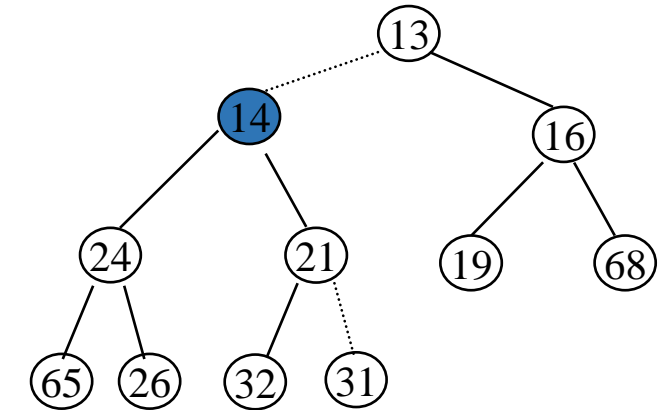
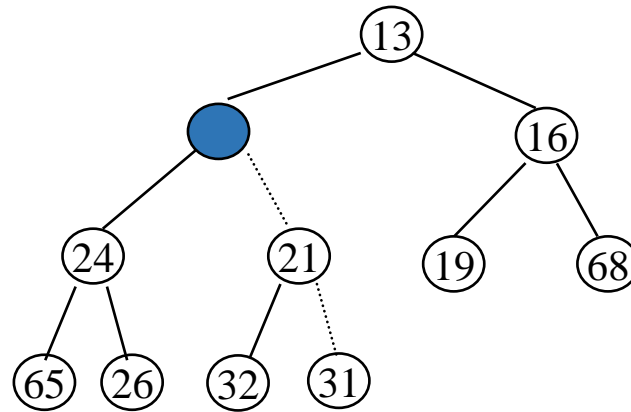
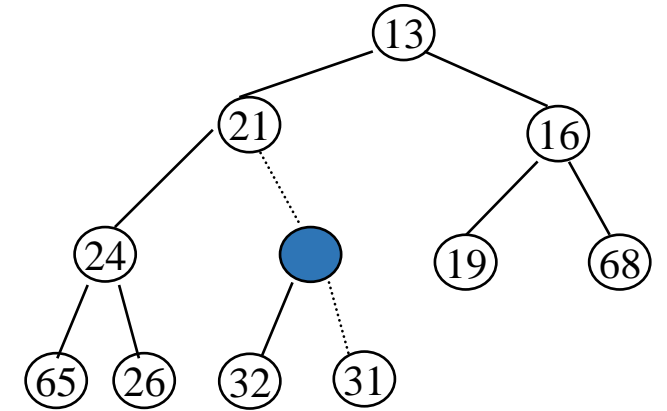
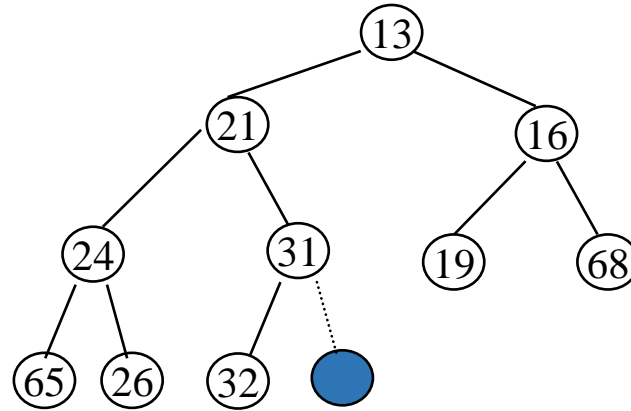
# Insertion



ตัวอย่างการเพิ่ม  
ข้อมูลใน Min-  
Heap

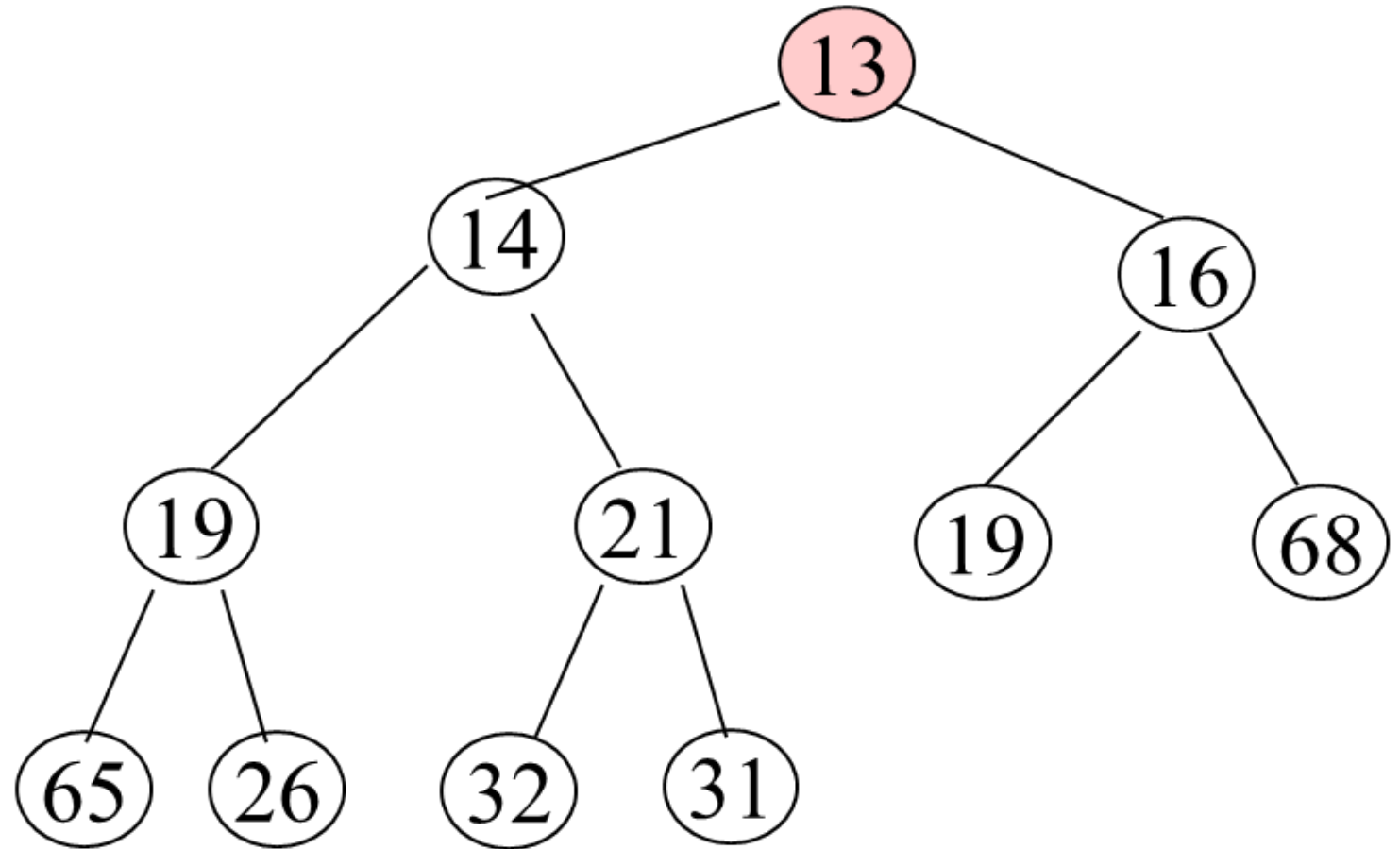


## Reheap Up

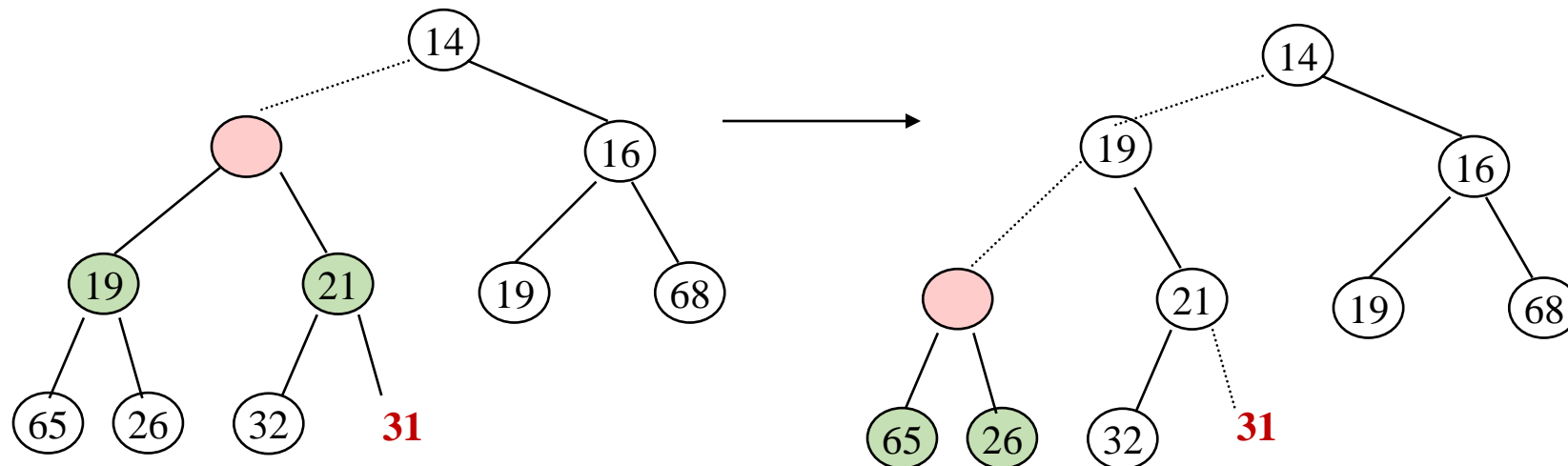
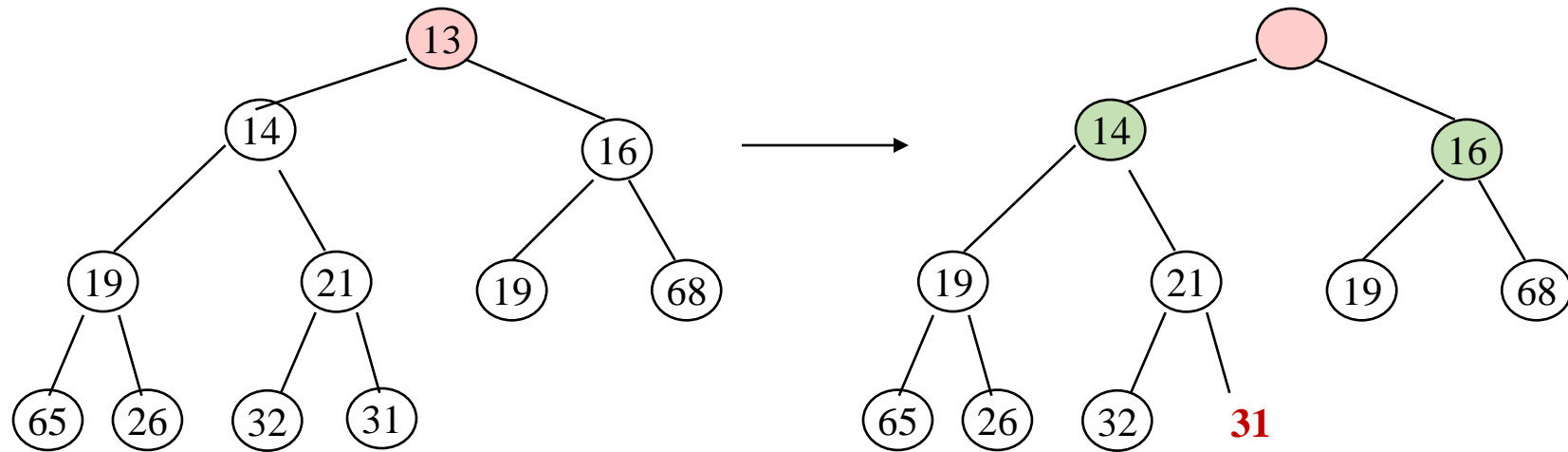




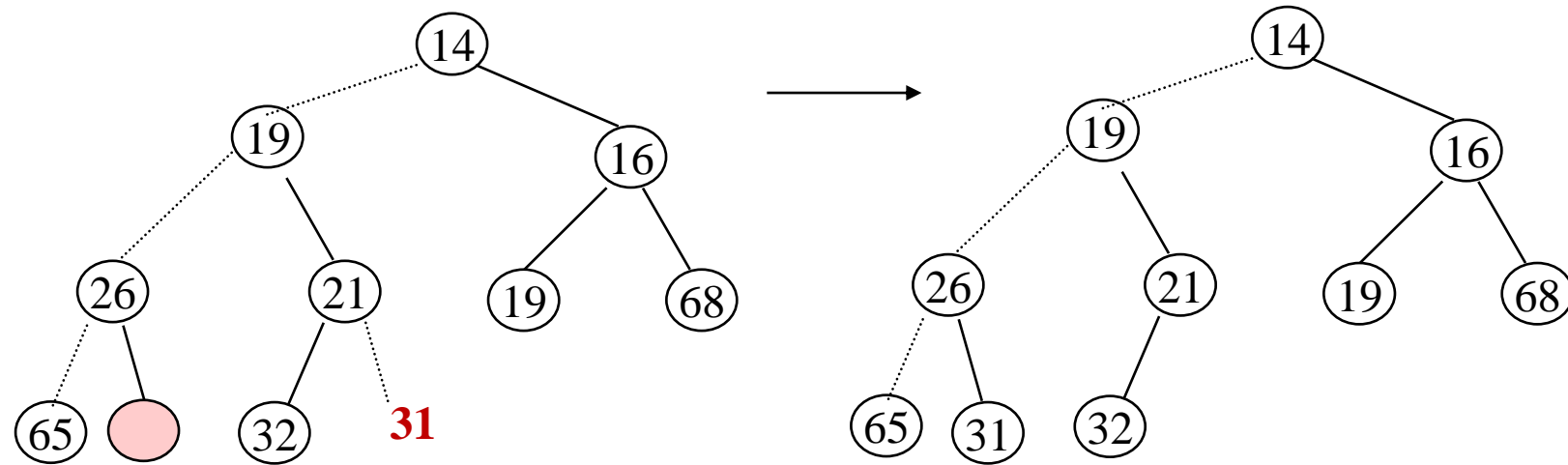
## การลบข้อมูลใน Heap



# การลบข้อมูลใน Heap



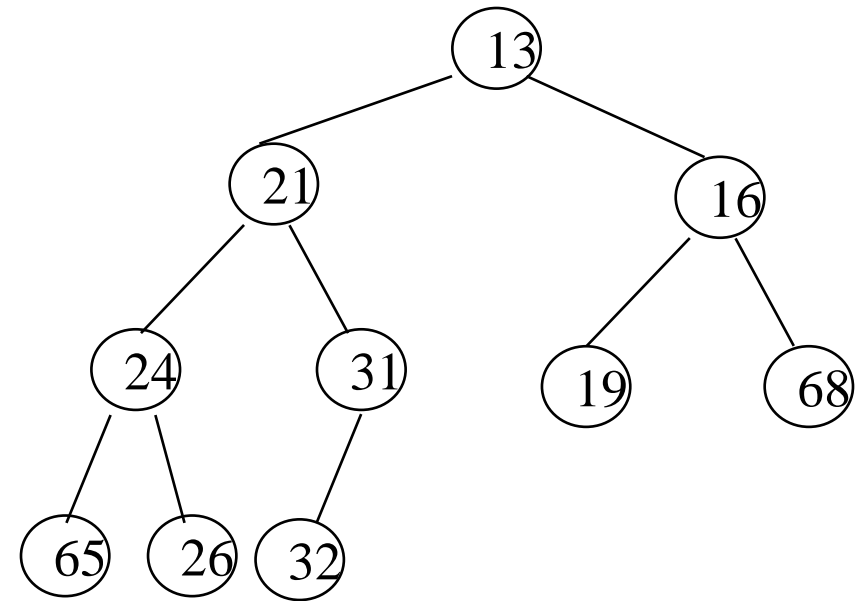
## Last two steps in *DeleteMin*



## การแทน heap ใน หน่วยความจำ

- การเก็บข้อมูลในไบนารีเหมาะกับการใช้อาร์เรย์ แบบมีลำดับ (Sequential Array Representation) เนื่องจากคุณสมบัติของการเป็น nearly complete ของตัวมันเอง จะทำให้ใช้เนื้อที่ได้เต็มประสิทธิภาพ
- ข้อมูลที่จัดเก็บลงในอาร์เรย์จะเรียงต่อกันไปตามลำดับจากซ้ายไปขวาของแต่ละ level ใน heap
- ความสัมพันธ์ของตำแหน่งข้อมูลใน heap คือ
  - parent ของโหนด  $i$  ใดๆ  $= i/2$  ถ้า  $i \neq 1$
  - left child ของโหนด  $i$  ใดๆ  $= 2i$
  - right child ของโหนด  $i$  ใดๆ  $= 2i + 1$





A heap in its logical form

0	1	2	3	4	5	6	7	8	9	10	11	12
	13	21	16	24	31	19	68	65	26	32		

A heap in an array

# Visualization

- ไปที่ <https://www.cs.usfca.edu/~galles/visualization/Heap.html>
- จงสร้าง min-heap จากข้อมูลต่อไปนี้

๒๓ ๑๕ ๑๗ ๑๓ ๓๑ ๑๐ ๒ ๔ ๒๙ ๑๔ ๕

A stack of books, bound with a white string, is visible on the left side of the slide. The books are of various thicknesses and are stacked in a slightly irregular manner.


## โครงสร้างของ Heap Node

```
heap    <array>           // ตัวแปรที่ใช้เก็บข้อมูล  
size    <integer>         // เก็บขนาดของ heap
```

# Heap Algorithms

- **insertHeap** แทรกโหนดใหม่ โดยเริ่มที่ตำแหน่งท้าย เปรียบเทียบ โหนดใหม่ กับ parent node ถ้าโหนดใหม่มีค่าน้อยกว่า จะมีการสลับค่ากับ parent node ทำไปเรื่อยๆ จนถึงตำแหน่งที่เหมาะสม
- **deleteHeap** ลบโหนดที่ root เลือกโหนดเพื่อมาแทน โดยการเปรียบเทียบ child node เพื่อหาค่าที่น้อยที่สุดลำดับถัดมาขึ้นมาแทน





**algorithm** insertHeap (val heap <array>, val size <integer>,  
val newVal <key>)

Given an array, rearrange data so that it forms heap

**Pre** heap is array containing a heap

size is the size of heap

newVal is new data to be inserted into heap

**Post** newVal has been inserted into heap in proper location

hole = size + 1

parent = hold/2

loop (heap[parent] >newVal)

heap[hold] = heap[parent]

hold = parent

parent = hole/2

heap[hold] = newVal

return

**end** insertHeap

**algorithm** deleteHeap (val heap <array>, val size <integer>)

**Deletes root of heap and passes data back to caller**

**Pre** heap is array containing a heap and size is heap size

**Return** return the key at root node

minElement = heap[1]

lastElement = heap[size]

size = size - 1

hold = 1

Loop (hold\*2 <= size)

    leftChild = hold \* 2

    rightChild = hold \* 2 + 1

    if (heap[rightChild] > heap[leftChild])

        child = heap[leftChild]

    else

        child = heap[rightChild]

    if (lastElement > heap[child])

        heap[hold] = heap[child]

    else

        break

heap[hold] = lastElement

return minElement

**end** deleteHeap

# การนำ Heap ไปแก้ปัญห

- การหาข้อมูลลำดับที่  $k$  ในลิสต์ของข้อมูลแบบไม่มีลำดับ
- วิธีการแก้ปัญห
  - เรียงลำดับข้อมูลในลิสต์ แล้วเลือกข้อมูลลำดับที่  $k$
  - สร้าง heap และลบข้อมูลออก  $k-1$  จำนวน เหลือข้อมูลตัวที่ต้องการที่ตำแหน่ง root แล้วค่อยแทรกข้อมูลตัวที่ลบออกกลับเข้าไปใน heap

