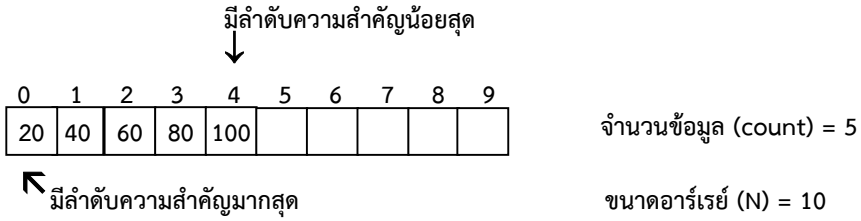
บทที่ 6

คิวแบบมีลำดับความสำคัญ

คิวแบบมีลำดับความสำคัญ (Priority Queue) เป็นชนิดข้อมูลนามธรรมคิวที่มีการจัดเก็บข้อมูลตามลำดับความสำคัญ (Priority) จากมากไปน้อย ข้อมูลที่มีลำดับความสำคัญสูงสุดอยู่ที่หัวคิว และข้อมูลที่มีความสำคัญต่ำสุดอยู่ที่ท้ายคิว การกำหนดว่าข้อมูลใดมีความสำคัญมากกว่า จะขึ้นอยู่กับประเภทของงานหรือลักษณะของข้อมูล ตัวอย่างการจัดการคิวแบบนี้ในชีวิตประจำวัน เช่น การเข้าคิวเพื่อรอรับบริการขององค์กรไปรษณีย์ เมื่อมีคนพิการมารอเข้าคิวเพื่อรับบริการอาจใช้การจัดการคิวแบบให้ลำดับความสำคัญแก่ผู้พิการให้สามารถรับบริการได้ก่อนโดยไม่ต้องต่อท้ายคิว การจัดการคิวงานของเครื่องพิมพ์ (Print Queue) ที่จัดเก็บข้อมูลตามขนาดของเอกสารที่ส่งมาพิมพ์ การจัดการการส่งแพ็กเกจ (Packet) ในระบบเครือข่าย (Network) ที่อาจให้ความสำคัญในการจัดเก็บข้อมูลตามสถานะความเร่งด่วนของแพ็กเกจ เป็นต้น

6.1 การดำเนินการของคิวแบบมีลำดับความสำคัญ

การนำข้อมูลเข้า Priority Queue จะกระทำที่ท้ายคิว เช่นเดียวกับคิวปกติ แต่จะมีการตรวจสอบลำดับความสำคัญของข้อมูลที่ต้องการเพิ่มเข้าคิวก่อนจึงจะรู้ตำแหน่งที่ต้องการเพิ่มข้อมูลในคิว ส่วนการนำข้อมูลออกจากคิวก็ต้องนำข้อมูลออกที่มีลำดับความสำคัญมากที่สุดออกจากคิวก่อนในหัวข้อนี้จะขอกำหนดให้ ข้อมูลที่มีค่าน้อยจะมีลำดับความสำคัญมากกว่าข้อมูลที่มีค่ามาก ดังนั้นข้อมูลที่ถูกจัดเก็บในตำแหน่งแรกของคิวจะมีค่าน้อยที่สุด (มีลำดับความสำคัญสูงสุด) และข้อมูลที่ถูกจัดเก็บในตำแหน่งท้ายคิวจะมีค่ามากที่สุด (มีลำดับความสำคัญต่ำสุด) ดังรูปที่ 6.1



รูปที่ 6.1 ตัวอย่างโครงสร้างของคิวแบบมีลำดับความสำคัญ

จากรูปที่ 6.1 พบว่าข้อมูล 20 เป็นข้อมูลในตำแหน่งแรกของคิวแสดงว่ามีลำดับความสำคัญมากที่สุด เพราะหากมีการนำข้อมูลออกจากคิวจะนำข้อมูล 20 ออกจากคิวเป็นอันดับแรกและ 100 เป็นข้อมูลตัวสุดท้ายในคิวแสดงว่ามีลำดับความสำคัญต่ำที่สุด

ตัวอย่างการกำหนดเมท็อดของการดำเนินการในคิวแบบมีลำดับความสำคัญ แสดงดังตารางที่ 6.1 โดยลักษณะการเปลี่ยนแปลงของค่าข้อมูลในคิว และผลลัพธ์หลังการเรียกใช้การดำเนินการต่างๆ ในคิวแบบมีลำดับความสำคัญ แสดงดังตารางที่ 6.2

ตารางที่ 6.1 ตัวอย่างการดำเนินการในคิวแบบมีลำดับความสำคัญ

การดำเนินการ	หน้าที่
insert(x)	นำข้อมูลใหม่ (ค่า x) จัดเก็บเพิ่มในคิว
removeMin()	ลบข้อมูลที่มีค่าน้อยที่สุดในคิว ซึ่งเป็นการนำข้อมูลที่มีลำดับความสำคัญมากที่สุดออกจากคิวนั่นเอง
min()	ค้นคืนข้อมูลที่มีค่าน้อยที่สุด (มีลำดับความสำคัญมากที่สุด) ในคิว
clear()	ลบข้อมูลทั้งหมดในคิว
size()	นับและคืนค่าจำนวนข้อมูลที่จัดเก็บในคิว
isEmpty()	ตรวจสอบว่าคิวว่างหรือไม่ หากคิวว่างให้

ตารางที่ 6.1 ตัวอย่างการเรียกใช้การดำเนินการของคิวแบบมีลำดับความสำคัญ

การดำเนินการ	ข้อมูลในคิว	ผลลัพธ์	คำอธิบาย
1) create()	ว่าง	-	สร้างคิวแบบมีลำดับความสำคัญ (มีสถานะเป็นคิวว่าง)
2) insert(5)	5	-	เพิ่มข้อมูล 5 ในคิวว่าง
3) insert(9)	5 9	-	เพิ่มข้อมูล 9 ต่อท้ายคิว เนื่องจากมีค่ามากกว่า 5
4) insert(2)	2 5 9	-	เพิ่มข้อมูล 9 ไว้ที่ตำแหน่งแรกในคิว เนื่องจากมีค่าน้อยที่สุด
5) insert(7)	2 5 7 9	-	เพิ่มข้อมูล 7 แทรกไว้ระหว่าง 5 และ 9 ตามลำดับความสำคัญ
6) min()	2 5 7 9	2	ค้นคืนค่าข้อมูลที่มีค่าน้อยที่สุด (มีลำดับความสำคัญมากที่สุด)
7) removeMin()	5 7 9	-	นำข้อมูลที่มีค่าน้อยที่สุดออกจากคิว
8) size()	5 7 9	3	นับและคืนค่าจำนวนข้อมูลที่ถูกจัดเก็บในคิว
9) min()	5 7 9	5	ค้นคืนค่าข้อมูลที่มีค่าน้อยที่สุด (ในขณะนี้ คือ 5)
10) removeMin()	7 9	-	นำข้อมูลที่มีค่าน้อยที่สุดออกจากคิว
11) removeMin()	9	-	นำข้อมูลที่มีค่าน้อยที่สุดออกจากคิว (ในขณะนี้ คือ 7)
12) removeMin()	-	-	นำข้อมูลที่มีค่าน้อยที่สุดออกจากคิว (ในขณะนี้ คือ 9)
13) isEmpty()	-	True	ตรวจสอบว่าคิวว่างหรือไม่
14) removeMin()	-	“Error”	เนื่องจากเป็นคิวว่างจึงไม่มีข้อมูลให้อ้างอิงถึง
15) size()	-	0	นับและคืนค่าจำนวนข้อมูลที่ถูกจัดเก็บในคิว

6.2 ลักษณะการใช้งานคิวแบบมีลำดับความสำคัญ

การใช้งานคิวแบบมีลำดับความสำคัญ จะมีความแตกต่างจากคิวในลักษณะวงกลมตรงที่ต้องจัดการในลักษณะตามลำดับความสำคัญ โดยหากกำหนดให้ค่าข้อมูลน้อยมีลำดับความสำคัญสูง และค่าข้อมูลที่มีค่ามากมีลำดับความสำคัญต่ำ การนำข้อมูลเข้าคิวหรือออกจากคิวจะจัดการตามลำดับความสำคัญของข้อมูล ตารางที่ 6.2 แสดงการเปรียบเทียบประสิทธิภาพในการทำงานของการใช้งานคิวแบบมีลำดับความสำคัญตามลักษณะการใช้งาน ดังนี้

1) การจัดการแบบไม่เรียงลำดับ

- การเพิ่มข้อมูลในคิวที่ไม่ต้องจัดเก็บแบบเรียงลำดับตามค่าข้อมูล จะสามารถใส่ในคิวที่ตำแหน่งท้ายคิวได้ทันที ดังนั้นไม่ว่าจะใช้งานคิวที่สร้างด้วยอาร์เรย์หรือลิงค์ลิสต์ เวลาในการทำงานของการดำเนินการ insert() จึงเท่ากับ O(1) เหมือนกัน
- การนำข้อมูลออกจากคิวแบบมีลำดับความสำคัญ จะต้องนำข้อมูลที่มีลำดับความสำคัญมากที่สุด (เป็นข้อมูลที่มีค่าน้อยที่สุด) ออกจากคิวก่อน แต่ข้อมูลไม่ได้จัดเก็บในคิวแบบเรียงลำดับ จึงต้องใช้เวลาในการค้นหาตำแหน่งข้อมูลที่มีค่าน้อยสุดในคิวก่อน ดังนั้นไม่ว่าจะใช้งานคิวที่สร้างด้วยอาร์เรย์หรือลิงค์ลิสต์ เวลาในการทำงานของการดำเนินการ removeMin() จึงเท่ากับ O(n) เหมือนกัน

2) การจัดการแบบเรียงลำดับ (Sorted List or Sorted Array)

- การเพิ่มข้อมูลในคิวแบบมีลำดับความสำคัญและมีการจัดการกับข้อมูลในคิวแบบเรียงลำดับ หากนำเข้าข้อมูลที่มีค่าน้อย จะแทรกอยู่ที่หัวคิวส่วนข้อมูลที่มีค่ามากจะวางไว้ที่ท้ายคิว ทำให้เวลานำข้อมูลเข้าคิวจึงต้องค้นหาตำแหน่งที่จะแทรกข้อมูลในคิวตามลำดับความสำคัญของข้อมูล ดังนั้นเวลาในการทำงานของการดำเนินการ insert() จึงเท่ากับ O(n) ไม่ว่าจะเป็นการใช้งานคิวที่สร้างด้วยอาร์เรย์หรือลิงค์ลิสต์
- การนำข้อมูลออกจากคิวที่จัดเก็บข้อมูลเรียงตามลำดับความสำคัญของข้อมูล เมื่อมีการนำข้อมูลออกจากคิวสามารถนำข้อมูลที่มีค่าน้อยสุดในตำแหน่งแรกของคิวออกจากคิวได้ทันที ดังนั้นไม่ว่าจะเป็นการใช้งานคิวที่สร้างด้วยอาร์เรย์หรือลิงค์ลิสต์ เวลาในการทำงานของการดำเนินการ removeMin() จึงเท่ากับ O(1) เหมือนกัน

ตารางที่ 6.2 การเปรียบเทียบประสิทธิภาพในการทำงานของการใช้งานคิวแบบมีลำดับความสำคัญ

การดำเนินการ	คิวที่สร้างด้วยลิงค์ลิสต์		คิวสร้างด้วยอาร์เรย์	
	แบบไม่เรียงลำดับ	แบบเรียงลำดับ	แบบไม่เรียงลำดับ	แบบเรียงลำดับ
1) insert()	O(1)	O(n)	O(1)	O(n)
2) removeMin()	O(n)	O(1)	O(n)	O(1)

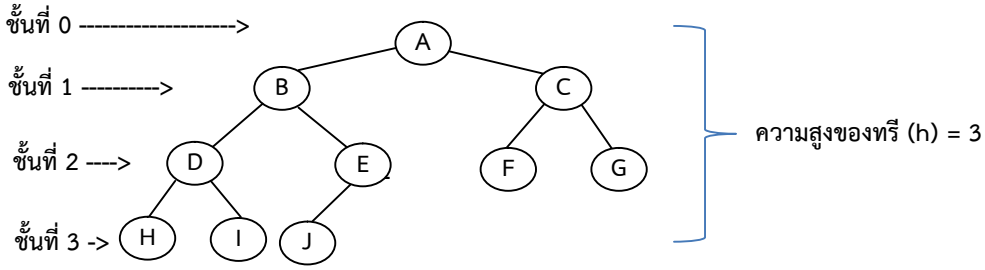
นอกจากนี้คิวที่สร้างด้วยลิงค์ลิสต์จะมีการใช้หน่วยความจำแบบพลวัต (Dynamic Memory) จึงควรคำนึงถึงการคืนพื้นที่หน่วยความจำเมื่อเลิกใช้งาน มิฉะนั้นจะไม่สามารถนำหน่วยความจำส่วนที่เลิกใช้งานแล้วนี้กลับมาใช้งานใหม่ได้ ซึ่งเป็นลักษณะปัญหาการรั่วของพื้นที่หน่วยความจำ (Memory Leak) การใช้งานคิวแบบมีลำดับความสำคัญ ไม่ว่าจะสร้างด้วยอาร์เรย์หรือลิงค์ลิสต์ จะใช้เวลาในการดำเนินการกับข้อมูลในคิวเป็น O(n) เหมือนกัน

6.3 การประยุกต์ใช้งานคิวแบบมีลำดับความสำคัญ

ในหัวข้อนี้จะกล่าวถึงการประยุกต์ใช้งานคิวแบบมีลำดับความสำคัญที่สร้างด้วยโครงสร้างข้อมูลฮีป (Heap) ที่มีลักษณะการจัดเก็บข้อมูลแบบไบนารีทรี (Binary Tree) ซึ่งทำให้ประสิทธิภาพในการใช้งานคิวเพิ่มขึ้น คือ จะใช้เวลาในการดำเนินการกับข้อมูลในคิวเท่ากับความสูงของทรี คือ log(n) ซึ่งเป็นฟังก์ชันเวลาที่โตช้ากว่า O(n)

6.6.1 ลักษณะของฮีป

โครงสร้างข้อมูลไบนารีฮีปทรี (Binary Heap Tree) หรือเรียกสั้นๆ ว่าฮีป จะจัดเก็บข้อมูลในลักษณะไบนารีทรีแบบบริบูรณ์ (Complete Binary Tree) คือ ไบนารีทรีที่แต่ละสมาชิกจะต้องมีสมาชิกครบทั้งโหนดลูกทางซ้าย (Left Child Node) และโหนดลูกทางขวา (Right Child Node) ยกเว้นโหนดในระดับใบ (Leaves) ที่สามารถมีสมาชิกไม่ครบได้ ดังรูปที่ 6.2



รูปที่ 6.2 ตัวอย่างโครงสร้างข้อมูลไบนารีทรีแบบบริบูรณ์

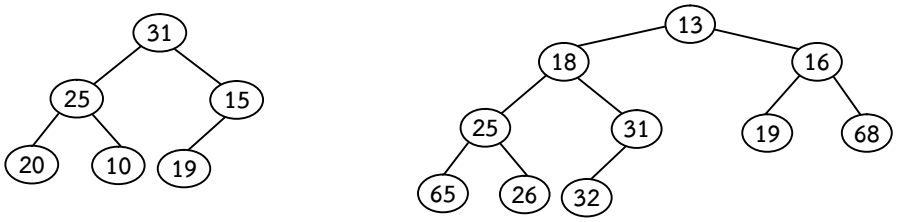
สมาชิกในฮีปจะเรียกว่าโหนด จากรูป 6.2 พบว่าโหนด A มีโหนดลูกครบทั้ง 2 โหนด คือ มีโหนด B เป็นโหนดลูกทางซ้าย และโหนด C เป็นโหนดลูกทางขวา ดังนั้นจะกล่าวได้ว่าโหนด B และโหนด C มีโหนด A เป็นโหนดพ่อนั่นเอง ในขณะที่โหนด E มีโหนดลูกไม่ครง คือ มีโหนดลูกทางซ้ายเพียงโหนดเดียว คือ โหนด J

ไบนารีทรีแบบสมบูรณ์ที่มีความสูงเท่ากับ h จะมีจำนวนโหนดอยู่ระหว่าง 2^h ถึง $2^{(h+1)}$-1 โหนด เช่น หากทรีมีความสูงเท่ากับ 3 จะมีจำนวนโหนดในทรีระหว่าง 2^3 ถึง 2^4- 1 คือ 8 ถึง 15 โหนด ดังนั้นหากทราบว่าจำนวนข้อมูลในไบนารีทรีทั้งหมด n โหนด จะสามารถคำนวณหาค่าความสูงของไบนารีทรีได้จากสมการ $\lfloor \log(n) \rfloor$ เช่น

- n เท่ากับ 8 จะได้ความสูงของทรีเท่ากับ $\lfloor \log(8) \rfloor = \lfloor \log(2^3) \rfloor = 3$
- n เท่ากับ 8 จะได้ความสูงของทรีเท่ากับ $\lfloor \log(15) \rfloor = \lfloor 3.91 \rfloor = 3$
- n เท่ากับ 16 จะได้ความสูงของทรีเท่ากับ $\lfloor \log(16) \rfloor = \lfloor \log(2^4) \rfloor = 4$

ฮีปมีโครงสร้างการจัดเก็บข้อมูล 2 รูปแบบ คือ Max-heap และ Min-heap โดยมีลักษณะความสัมพันธ์ระหว่างโหนดพ่อแม่กับโหนดลูก ดังนี้

- Max-heap คือ ฮีปที่ข้อมูลของโหนดพ่อแม่จะมีค่ามากกว่าข้อมูลของโหนดลูก ดังแสดงในรูป 6.2 (ก) ดังนั้นข้อมูลที่มีค่ามากที่สุดจะอยู่ที่ตำแหน่งรากของทรี
- Min-heap คือ ฮีปที่ข้อมูลของโหนดพ่อแม่จะมีค่าน้อยกว่าข้อมูลของโหนดลูก ดังแสดงในรูป 6.2 (ข) ดังนั้นข้อมูลที่มีค่าน้อยที่สุดจะอยู่ที่ตำแหน่งรากของทรี



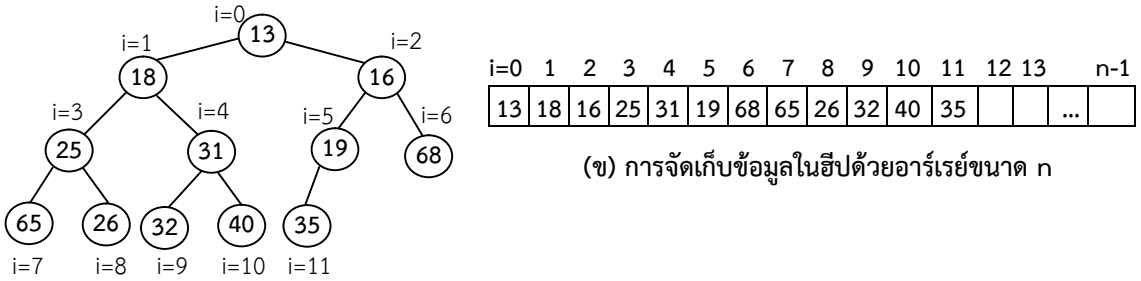
(ก) Max-heap **(ข) Min-heap**

รูปที่ 6.2 ตัวอย่างโครงสร้างข้อมูลฮีป

(1) การสร้างฮีปด้วยอาร์เรย์

โครงสร้างข้อมูลแบบไบนารีฮีป สามารถสร้างด้วยอาร์เรย์ ดังรูปที่ 6.3 เป็นตัวอย่างฮีปแบบ Min-heap ที่มีจำนวนสมาชิกทั้งหมด 12 ค่า ดังรูปที่ 6.3 (ก) แต่ละสมาชิกจะถูกจัดเก็บในอาร์เรย์ขนาด N ในตำแหน่งที่ i = 0 ... 11 ดังรูปที่ 6.3 (ข) ซึ่งมีคุณสมบัติสำหรับสมาชิกตำแหน่งที่ i ดังนี้

- โหนดลูกด้านซ้าย (Left Child) ของสมาชิกตำแหน่งที่ i คือ สมาชิกตำแหน่งที่ 2i +1
- โหนดลูกด้านขวา (Right Child) ของสมาชิกตำแหน่งที่ i คือ สมาชิกตำแหน่งที่ 2i+2
- โหนดแม่ (Parent) ของสมาชิกตำแหน่งที่ i คือ สมาชิกตำแหน่งที่ $\lfloor (i-1)/2 \rfloor$



(ก) โครงสร้างข้อมูลฮีป

(ข) การจัดเก็บข้อมูลในฮีปด้วยอาร์เรย์ขนาด n

รูปที่ 6.3 ตัวอย่างการจัดเก็บข้อมูลในฮีปด้วยอาร์เรย์

จากรูปที่ 6.3 สมาชิกในฮีปของโหนดข้อมูล 31 ซึ่งเก็บในอาร์เรย์ตำแหน่งที่ i = 4 จะมี

- ข้อมูลของโหนดลูกด้านซ้ายจะเก็บในตำแหน่งที่ 2i+1 คือ โหนดของค่าข้อมูล 32 ที่เก็บในอาร์เรย์ตำแหน่งที่ (2*i+1) = (2*4+1) = 9
- ข้อมูลของโหนดลูกด้านขวาจะเก็บในตำแหน่งที่ 2i+2 คือ โหนดของค่าข้อมูล 40 ที่เก็บในอาร์เรย์ตำแหน่งที่ (2*i+2) = (2*4+2) = 10
- ข้อมูลของโหนดพ่อแม่จะเก็บในตำแหน่งที่ $\lfloor (i-1)/2 \rfloor$ คือ โหนดของค่าข้อมูล 18 ที่เก็บในอาร์เรย์ตำแหน่งที่ $\lfloor (4-1)/2 \rfloor = \lfloor 1.5 \rfloor = 1$

(2) การลบข้อมูลออกจากฮีป

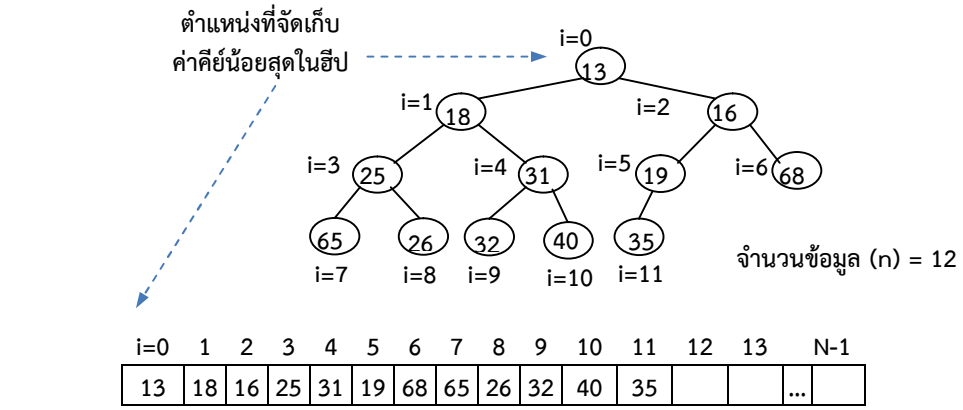
ขั้นตอนในการลบข้อมูลออกจากฮีป แสดงดังรูปที่ 6.4 และ 6.5 มีรายละเอียดดังนี้

1) คืนค่า (Return) ของสมาชิกที่อยู่ในตำแหน่งรากของทรีพร้อมทั้งลบข้อมูลนั้นออกจากโครงสร้างฮีป เนื่องจากฮีปที่กล่าวถึงในหัวข้อนี้เป็น Min-heap ข้อมูลที่น้อยที่สุดในฮีปจะอยู่ที่ตำแหน่งรากของทรี ดังรูปที่ 6.4 (ก) การลบข้อมูลออกจากฮีป จึงเป็นการดำเนินการ deleteMin() คือจะทำการอ่านค่าสมาชิกที่น้อยที่สุดในฮีป แล้วลบออกจากฮีป ซึ่งทำให้เกิด

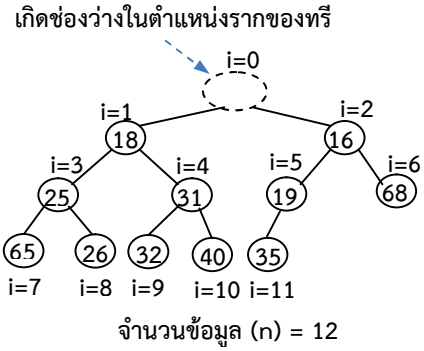
โหนดว่างที่ตำแหน่งรากของฮีป ดังรูปที่ 6.4 (ข)

2) ทำการหาค่าสมาชิกที่มีค่าคีย์น้อยที่สุดในลำดับถัดไป มาไว้ที่ตำแหน่งรากที่ว่างอยู่นี้แทน มีขั้นตอนดังนี้

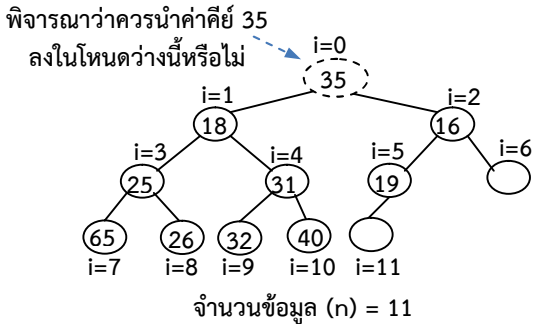
2.1) ให้นำค่าคีย์ที่อยู่ในลำดับสุดท้ายของฮีปมาพิจารณา นั่นคือ หากมีค่าคีย์ในฮีปn จำนวน จะนำค่าคีย์ในตำแหน่งที่ n-1 ของอาร์เรย์ (สมมุติว่า คือ ค่า x) มาใส่ไว้ในตำหน่งโหนดว่างนี้แทนค่าเดิมที่ลบออกไป (ในขณะนี้ คือ ตำแหน่งราก) ดังนั้นจำนวนค่าคีย์ในฮีปจะลดลง 1 ค่า ดังรูปที่ 6.4 (ค) แสดงตัวอย่างกรณี n เท่ากับ 12 ค่าคีย์ที่ต้องนำมาพิจารณาว่าสามารถใส่ในช่องว่างของฮีปได้หรือไม่ คือ ค่าคีย์ 35 ในตำแหน่งที่ 11 ซึ่งมีผลให้จำนวนค่าคีย์ในฮีปลดลงเหลือ 11 ค่า



(ก) ตำแหน่งในการลบข้อมูลออกจากโครงสร้างฮีปที่สร้างด้วยอาร์เรย์ขนาด N



(ข) หลังลบคีย์ที่มีค่าน้อยสุดออกจากฮีป



(ค) หาตำแหน่งในการวางค่าคีย์จากโหนดสุดท้ายในฮีป

รูปที่ 6.4 ตัวอย่างการลบข้อมูลออกจากฮีป

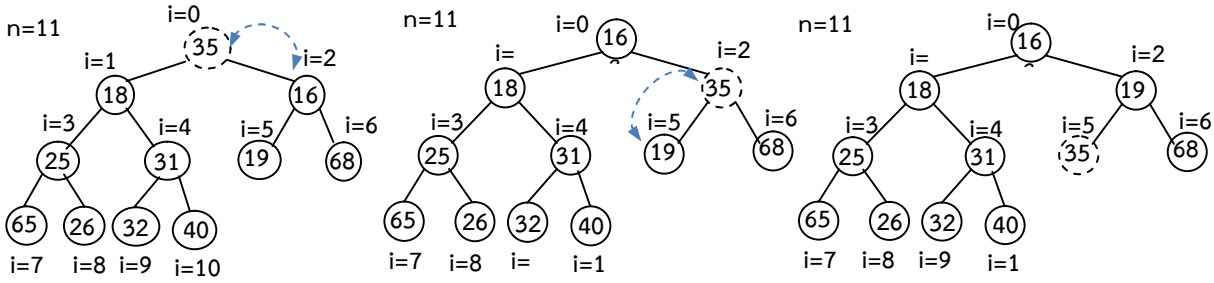
2.2) จากนั้นให้พิจารณาว่า หากใส่ค่า x ในตำแหน่งโหนดว่างนี้แล้ว ดังรูปที่ 6.4 (ค) จะทำให้คุณสมบัติของฮีปเสียไปหรือไม่ ดังนี้

- หากค่าคีย์ในโหนดลูกทางซ้ายและทางขวาของโหนดที่ว่าง มีค่าคีย์ที่มากกว่าค่า x ทั้งคู่ แสดงว่าข้อมูล x สามารถวางในตำแหน่งโหนดว่างนี้ได้ ให้จบการทำงาน
- แต่หากค่าคีย์ของโหนดลูกทางซ้ายหรือทางขวาน้อยกว่าค่า x แสดงว่าจะต้องนำค่าคีย์ของโหนดลูกที่มีค่าน้อยสุดมาใส่ในโหนดว่าง ซึ่งมีผลให้โหนดลูกที่ถูกขยับค่าคีย์ขึ้นทำหน้าที่เป็นโหนดว่างแทนโหนดเดิม

- หากโหนดว่างนี้ยังมีโหนดลูก ให้วนทำซ้ำข้อ 2.2 จนกว่าจะพบตำแหน่งที่เหมาะสมสำหรับค่า x

รูปที่ 6.5 แสดงตัวอย่างการปรับโครงสร้างข้อมูลฮีปด้วยวิธีการซึมผ่านลง (Percolate Down) สามารถอธิบายขั้นตอนการทำงานได้ ดังนี้

- จากรูปที่ 6.5 (ก) พบว่าโหนดว่างที่ i=0 ไม่สามารถใส่ค่า 35 ได้ เนื่องจากค่าคีย์ 16 ของโหนดลูกทางขวามีค่าน้อยกว่า 35 จึงต้องนำค่า 16 มาใส่ในโหนดว่าง ดังนั้นโหนดลูกทางขวา (ที่ i=2) จะทำหน้าที่เป็นโหนดว่างแทน
- จากรูปที่ 6.5 (ข) พบว่าโหนดว่างที่ i=2 ไม่สามารถใส่ค่า 35 ได้เช่นกัน เนื่องจากค่าคีย์ 19 ของโหนดลูกทางซ้าย (ที่ i=5) มีค่าน้อยกว่า 35 จึงต้องนำค่า 19 มาใส่ในโหนดว่าง ดังนั้นโหนดลูกทางซ้ายจะทำหน้าที่เป็นโหนดว่างแทน
- จากรูปที่ 6.5 (ค) พบว่าโหนดว่างที่ i=5 ไม่มีโหนดลูกจึงสามารถใส่ค่า 35 ได้ที่โหนดว่างนี้ได้ จึงหยุดการปรับโครงสร้างฮีปด้วยวิธีการซึมผ่านลง และได้โครงสร้างข้อมูลฮีปที่ตรงตามคุณสมบัติ Min-heap ดังเดิม



(ก) พิจารณาโหนดว่างที่ i=0 ต้องสลับค่าคีย์ระหว่าง 35 และ 16

(ข) พิจารณาโหนดว่างที่ i=2 ต้องสลับค่าคีย์ระหว่าง 35 และ 19

(ค) พิจารณาโหนดว่างที่ i=5 สามารถใส่ค่าคีย์ 35 ที่โหนดว่างนี้ได้

รูปที่ 6.5 ตัวอย่างการปรับโครงสร้างข้อมูลฮีปด้วยวิธีการซึมผ่านลง

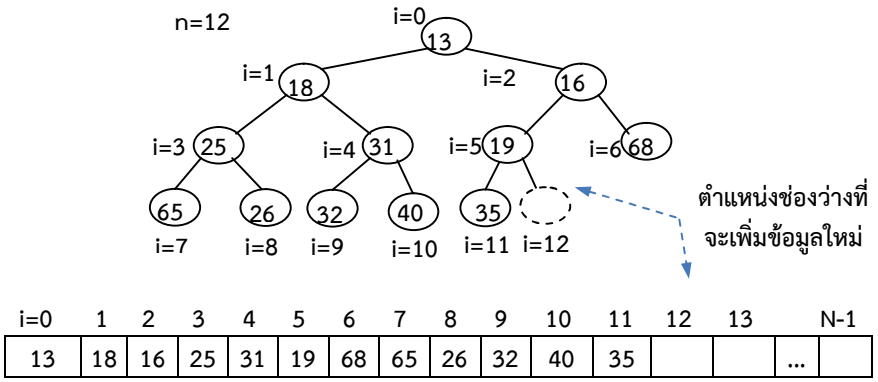
จากการทำงานของวิธีการลบข้อมูลออกจากโครงสร้างฮีปจะพบว่าเวลาในการทำงานสูงสุดจะไม่เกินค่าความสูงของทรี นั่นคือจะได้ค่า T(n) เท่ากับ log (n) เมื่อ n คือจำนวนข้อมูลที่จัดเก็บในทรี

(3) การเพิ่มข้อมูลหรือแทรกสมาชิกใหม่ในฮีป

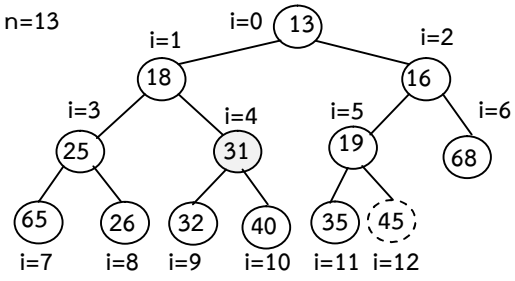
เมื่อต้องการนำค่าคีย์ใหม่จัดเก็บในโครงสร้างข้อมูลฮีป สำหรับการดำเนินการ Insert()มีลำดับการทำงาน ดังรูปที่ 6.6 มีรายละเอียดดังนี้

1) นำค่าคีย์ใหม่ใส่ในตำแหน่งถัดไปในฮีป นั่นคือ หากฮีปเก็บค่าคีย์ทั้งหมด n ค่า จะใส่ค่าคีย์ใหม่ในลำดับที่ n+1 ของฮีป ดังนั้นโหนดใหม่ที่เพิ่มนี้จะอยู่ในตำแหน่งโหนดใบของทรีเสมอ
2) ให้ดูว่าการเพิ่มโหนดข้อมูลใหม่นี้ ทำให้ฮีปยังมีคุณสมบัติของ Min-heap อยู่หรือไม่ ในที่นี้จำนวนคีย์ในฮีป คือ 12 ค่า ดังนั้นค่าคีย์ใหม่จะเก็บในลำดับที่ 13 ของฮีป หรือก็คือตำแหน่งช่องว่างที่ 12 ของอาร์เรย์ (เนื่องจากเริ่มต้นจากตำแหน่งที่ 0) ซึ่งจะต้องพิจารณาคุณสมบัติของฮีปจากโหนดพ่อแม่ของโหนดใหม่ที่เพิ่ม ซึ่งก็คือโหนดในตำแหน่งที่ $\lfloor(12-1)/2\rfloor$= 5 ซึ่งเก็บค่าคีย์ 19 อยู่ ดังรูปที่ 6.6(ก)

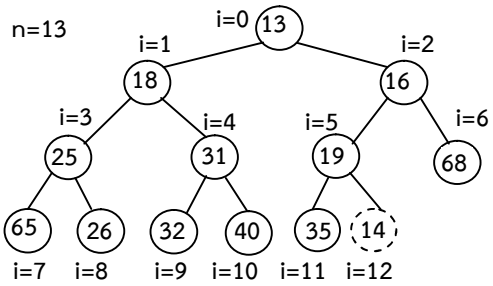
2.1) หากค่าคีย์ใหม่ที่แทรกเพิ่มในฮีป คือ 45 จะพบว่าคีย์ของโหนดพ่อแม่ (ค่า 19) มีค่าน้อยกว่า 45 แสดงว่าโครงสร้างข้อมูลฮีปนี้ยังตรงตามคุณสมบัติของ Min-heap อยู่ แสดงว่าสามารถใส่ค่าคีย์ 45 ไว้ที่โหนดใหม่ได้โดยไม่ต้องมีการปรับโครงสร้างข้อมูลฮีป ดังรูปที่ 6.6(ข)



(ก) ตำแหน่งในการเพิ่มข้อมูลใหม่ในโครงสร้างฮีปที่สร้างด้วยอาร์เรย์ขนาด N



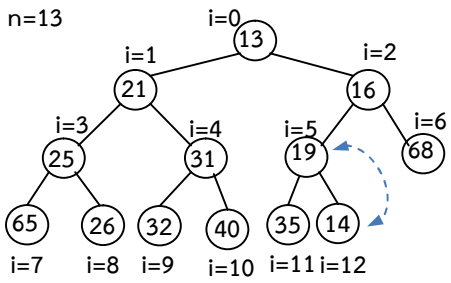
(ข) กรณีเพิ่ม 45



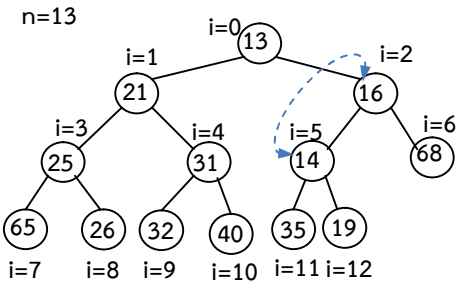
(ค) กรณีเพิ่ม 14

รูปที่ 6.6 ตัวอย่างการเพิ่มข้อมูลใหม่ในฮีป

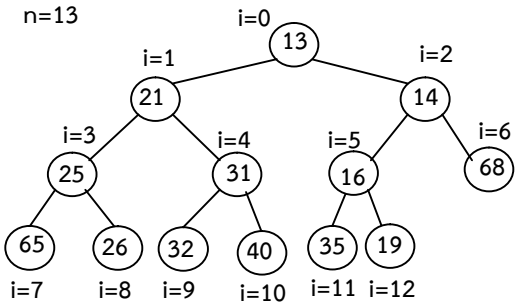
2.2) หากค่าคีย์ใหม่ที่แทรกเพิ่มในฮีป คือ 14 จะพบว่าคีย์ของโหนดพ่อแม่ (ค่า 19) มีค่ามากกว่าแสดงว่าโหนดใหม่ที่เพิ่มนี้มีผลให้โครงสร้างข้อมูลฮีปไม่ตรงตามคุณสมบัติของ Min-heap ดังรูปที่ 6.6(ค) ซึ่งปัญหานี้สามารถแก้ไขได้ด้วยการสลับค่าคีย์ระหว่างโหนดลูกกับโหนดพ่อแม่ จากนั้นเลื่อนขึ้นไปพิจารณาที่โหนดพ่อแม่นั้นต่อไป โดยสามารถเลื่อนขึ้นไปได้สูงสุดถึงระดับโหนดรากของทรี หรือจนกว่าจะเจอตำแหน่งโหนดที่ไม่ทำให้เสียคุณสมบัติของฮีป วิธีการนี้เรียกว่าการซึมผ่านขึ้นไป (Percolate Up) ดังตัวอย่างในรูปที่ 6.7(ก) – (ข) ตามลำดับ จากรูปที่ 6.7 (ค) จะพบว่าหลังจากทำการสลับค่า 16 และ 14 แล้ว เมื่อพิจารณาโหนดพ่อแม่ คือ โหนดค่าคีย์ 13 ต่อไป จะพบว่าเจอโหนดที่ไม่ทำให้เสียคุณสมบัติของ Min-heap จึงหยุดทำการสลับค่าคีย์ดังนั้นเวลาในการทำงานสำหรับการดำเนินการเพื่อเพิ่มค่าคีย์ใหม่ในฮีปจะใช้เวลาสูงสุดไม่เกินค่าความสูงของทรี ซึ่งเท่ากับ log(n) เมื่อ n คือจำนวนข้อมูลที่จัดเก็บในทรี



(ก) ที่โหนด i=12 ต้องสลับค่าคีย์ระหว่าง 14 และ 19



(ข) ที่ i=5 ต้องสลับค่าคีย์ระหว่าง 14 และ 16



(ค) ฮีปหลังปรับค่าคีย์ด้วยวิธีการซึมผ่านขึ้นไป

รูปที่ 6.7 ตัวอย่างการปรับโครงสร้างข้อมูลฮีปด้วยวิธีการซึมผ่านขึ้นไป

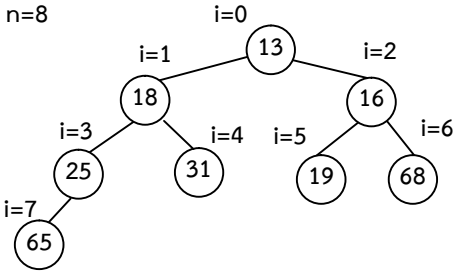
6.6.2 การประยุกต์ใช้งานโครงสร้างข้อมูลฮีป

ในหัวข้อนี้จะยกตัวอย่างการประยุกต์ใช้งานโครงสร้างข้อมูลฮีปในลักษณะของคิวแบบมีลำดับความสำคัญ สำหรับงานการเรียงลำดับข้อมูลที่เรียกว่า (Heap Sort) ประกอบด้วย 2 ขั้นตอนหลักคือ

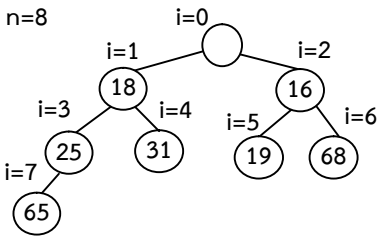
1) นำข้อมูลในอาร์เรย์มาสร้างเป็นโครงสร้างข้อมูลฮีปในลักษณะ Min-heap หรือ Max-heap
2) เรียกใช้การดำเนินการ deleteMin() หรือ deleteMax() เพื่อเรียงลำดับข้อมูลทีละ 1 ค่า และวนทำซ้ำจนครบทุกค่าข้อมูลในฮีป

ตัวอย่างการเรียงลำดับข้อมูลที่จัดเก็บด้วยโครงสร้างข้อมูลฮีปแบบ Min-heap แสดงดังรูปที่ 6.8 มีลำดับขั้นตอนในการเรียงลำดับ ดังนี้

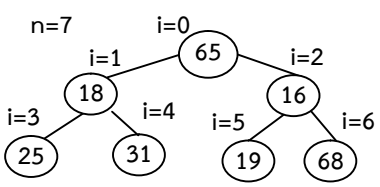
1) นำข้อมูลมาจัดเก็บในลักษณะ Min-heap ดังตัวอย่างในรูปที่ 6.8 (ก) แสดงโครงสร้างฮีปที่สร้างจากข้อมูลจำนวน 8 ค่า คือ 13 18 16 25 31 19 68 และ 65
2) เรียงลำดับข้อมูลทีละ 1 ค่าตามลำดับขั้นตอนต่อไปนี้
   2.1) นำข้อมูลต่ำสุดที่เก็บในฮีปมาเรียงลำดับ
   2.2) เรียกใช้การดำเนินการ deleteMin() เพื่อลบข้อมูลต่ำสุดออกจากฮีป โดยการนำค่าต่ำสุดในตำแหน่งราก (ที่ i=0) แล้วย้ายค่าจากตำแหน่งสุดท้ายในฮีปมาแทนที่ในตำแหน่งราก จากนั้นทำการปรับโครงสร้างฮีปให้ตรงตามคุณสมบัติ Min-heap คือ ทุกโหนดจะมีค่ามากกว่าโหนดพ่อแม่เสมอ
   2.3) วนทำซ้ำจนครบทุกค่าข้อมูลในฮีป



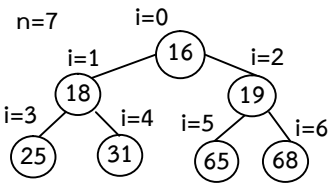
(ก) โครงสร้างข้อมูลฮีปแบบ Min-heap ขนาด 8



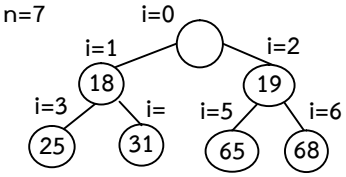
(ข) ที่ i =0, ลบค่าข้อมูล
ต่ำสุดออกจากฮีป



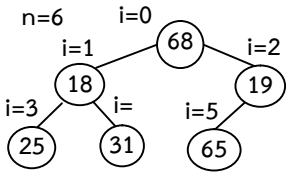
(ค) นำข้อมูลตำแหน่งสุดท้าย
ในฮีปไปแทนที่



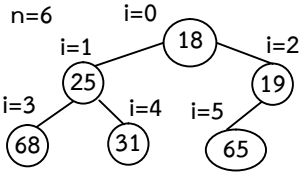
(ง) ปรับโครงสร้างฮีป
ให้เป็น Min-heap



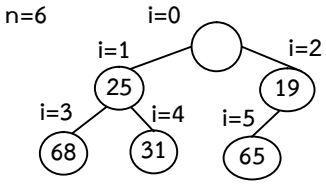
(จ) ที่ i =0, ลบค่าข้อมูล
ต่ำสุดออกจากฮีป



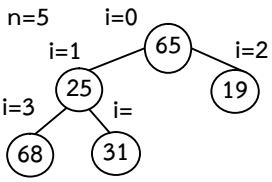
(ฉ) นำข้อมูลตำแหน่งสุดท้าย
ในฮีปไปแทนที่



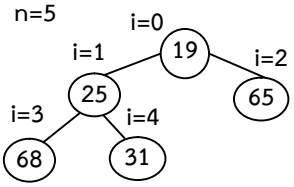
(ช) ปรับโครงสร้างฮีป
ให้เป็น Min-heap



(ซ) ที่ i =0, ลบค่าข้อมูล
ต่ำสุดออกจากฮีป



(ฌ) นำข้อมูลตำแหน่งสุดท้าย
ในฮีปไปแทนที่



(ญ) ปรับโครงสร้างฮีป
ให้เป็น Min-heap

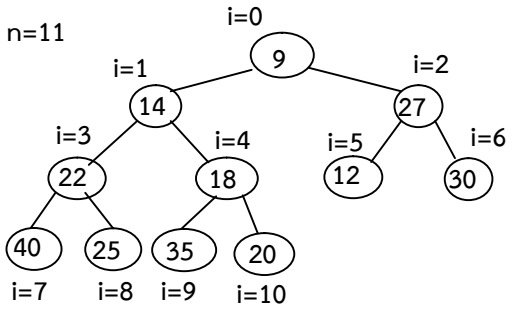
รูปที่ 6.8 ตัวอย่างการเรียงลำดับข้อมูลจากโครงสร้างฮีปจำนวน 3 ค่า คือ 13 16 18

จากรูปที่ 6.8 แสดงตัวอย่างการเรียงลำดับข้อมูลที่จัดเก็บด้วยโครงสร้างฮีปแบบ Min-heap จำนวน 3 ค่า คือ 13 16 18 ด้วยการลบข้อมูลน้อยที่สุดออกจากฮีปเพื่อนำมาเรียงลำดับจากน้อยไปมากทีละหนึ่งค่า จากรูปที่ 6.8 (ก) เมื่อต้องการนำค่าน้อยสุด คือ 13 มาเรียงลำดับ ดังรูปที่ 6.8 (ข)-(ง) การเรียกใช้การดำเนินการ removeMin() จะลบค่าน้อยสุดในตำแหน่งราก (ที่ i=0) ออกจากฮีปมาเรียงลำดับ ซึ่งมีผลให้จำนวนข้อมูลในฮีปลดลงหนึ่งค่า และเนื่องจากต้องนำข้อมูลในตำแหน่งสุดท้ายของฮีปมาวางแทนในตำแหน่งรากที่ว่างอยู่จึงต้องมีการปรับโครงสร้างฮีปให้ตรงตามคุณสมบัติ

Min-heap ทุกครั้งเสมอสำหรับขั้นตอนการนำค่าน้อยสุดตัวถัดไป คือ 16 และ 18 มาเรียงลำดับจะทำงานตามขั้นตอนในรูปที่ 6.8 (จ)-(ช) และ 6.8 (ซ)-(ญ) ตามลำดับ ดังนั้นหากนำข้อมูลน้อยสุดจากฮีปมาเรียงลำดับทีละค่าไปเรื่อยๆ จนครบทั้ง 8 ค่า จะได้ข้อมูลเรียงลำดับจากน้อยไปมาก ดังนี้ 13 16 18 19 25 31 65 68

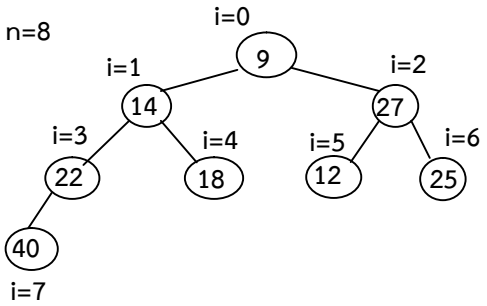
แบบฝึกหัด

1. จงยกตัวอย่างงานที่มีการนำโครงสร้างคิวแบบมีลำดับความสำคัญไปประยุกต์ใช้งาน
2. จงเปรียบเทียบเวลาในการทำงานของอัลกอริทึมสำหรับการดำเนินการ removeMin() และ insert() ของโครงสร้างข้อมูลคิวแบบมีลำดับความสำคัญ เมื่อนำไปใช้ด้วยโครงสร้างข้อมูลลิสต์ที่แตกต่างกันระหว่าง Pointer-based List (Linked list) และ Array-based List
3. จากตัวอย่างข้อมูลที่จัดเก็บอยู่ในหน่วยความจำโดยใช้โครงสร้างข้อมูล minHeap (ดังรูป)



หากเรียกใช้ Operation removeMin() จะได้ผลลัพธ์การทำงานอย่างไร จงเขียนรูปภาพแสดงการเปลี่ยนแปลงโครงสร้างข้อมูล minHeap ในแต่ละขั้นตอน

4. จากโครงสร้าง min-heap ที่กำหนดให้ จงแสดงผลลัพธ์ของโครงสร้าง min-heap ที่ได้จากการทำงานของแต่ละคำสั่งต่อไปนี้
   - X=Heap1.min()
   - Heap1.insert(X+10)



5. หากข้อมูลนำเข้า คือ 30 5 25 55 15 ตามลำดับ จงแสดงขั้นตอนการสร้างโครงสร้างข้อมูล min-heap และ max-heap