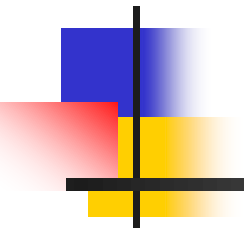


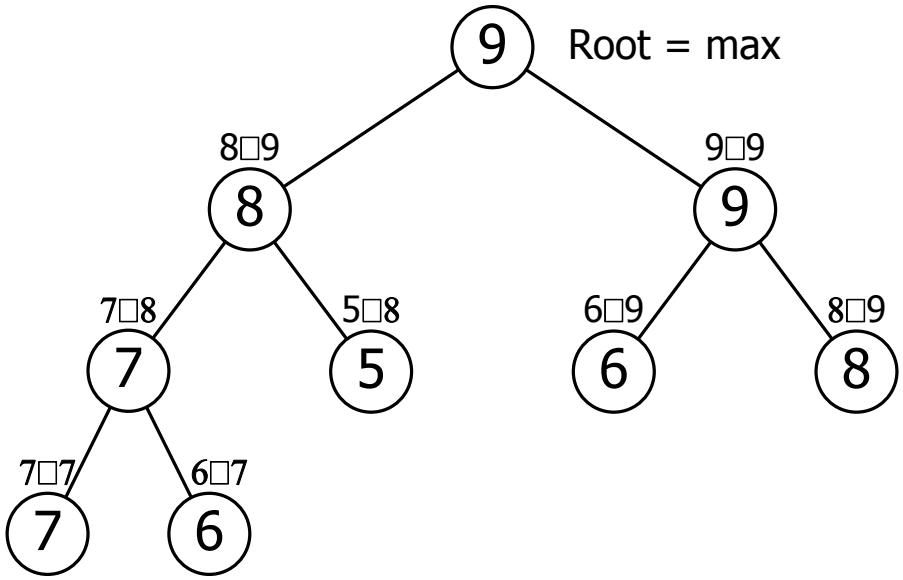
Chapter 07

Heap Tree

Heap Tree

Heap Tree

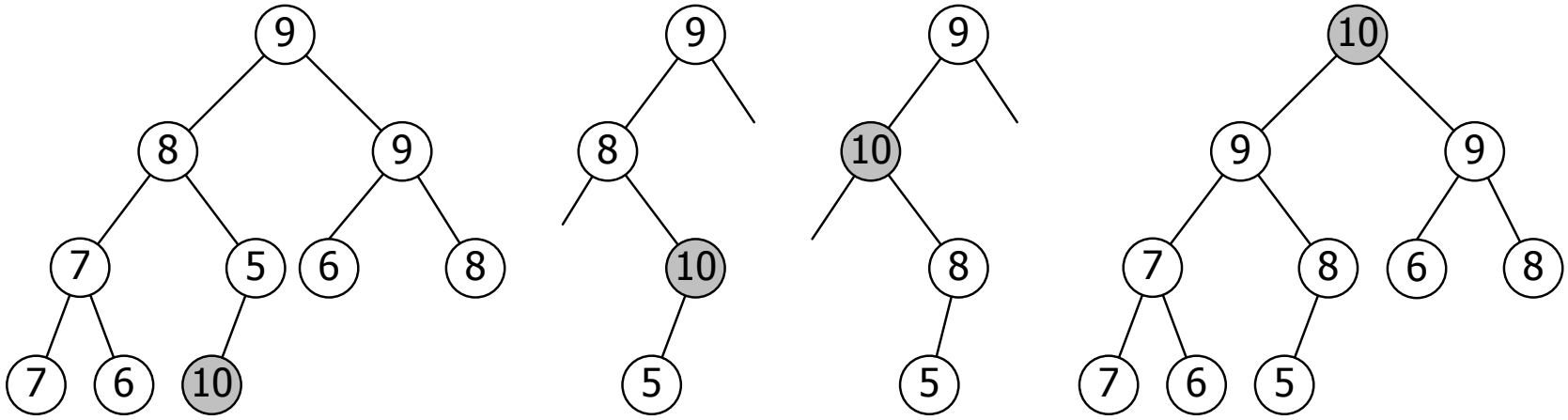
- ต้นไม้ต้องเป็น **complete binary tree** (การเกิดโหนดไล่จากซ้ายไปขวาจนเต็มทีละชั้น) ทำให้สามารถใช้ **ARRAY** มา **implement** ได้
- ค่าของโหนด **N[i]** ใด ๆ จะต้อง**มากกว่าหรือเท่ากับ** ค่าของโหนดลูก **N[2i] , N[2i+1]**



Insert node into heap

- การเพิ่มโหนดเข้าไปใน Heap
  - นำโหนดใหม่ที่ต้องการเพิ่มไปสร้างเป็นโหนดสุดท้าย
  - เปรียบเทียบค่าของโหนดใหม่กับโหนดที่อยู่ข้างบน ถ้ามีค่าน้อยกว่าให้หยุด แต่ถ้ามีค่ามากกว่า ให้สลับค่ากัน แล้วทำต่อขึ้นไปเรื่อยๆ

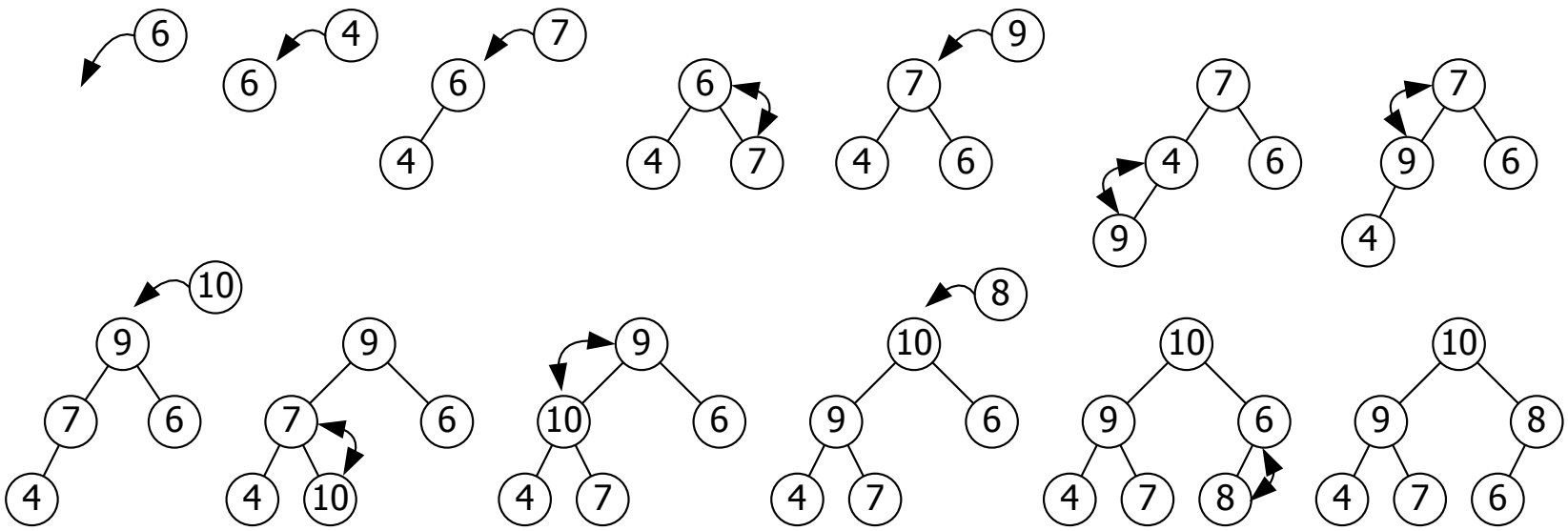
- Example เพิ่มโหนด 10 เข้าไปในฮีป



Constructing heap tree

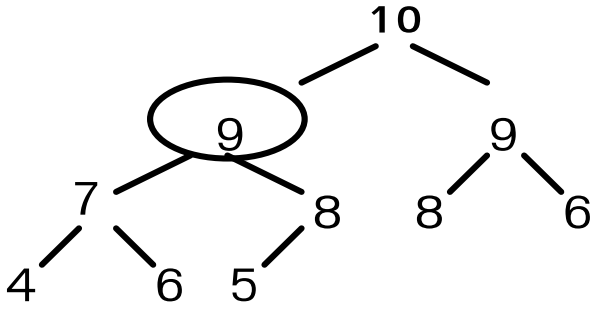
- การสร้างต้นไม้แบบฮีป
  - นำโหนดที่ต้องการมาเพิ่มเข้าไปในต้นไม้ แล้วเปรียบเทียบเพื่อสลับค่ากับโหนดที่อยู่ข้างบน(Parent) จนครบทุกโหนด

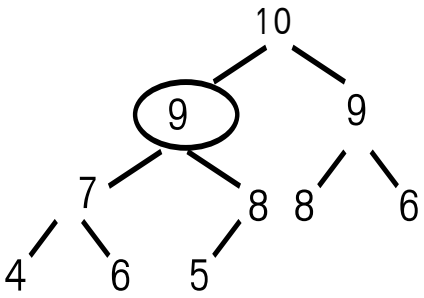
- Example สร้างฮีปด้วยข้อมูล 6 4 7 9 10 8



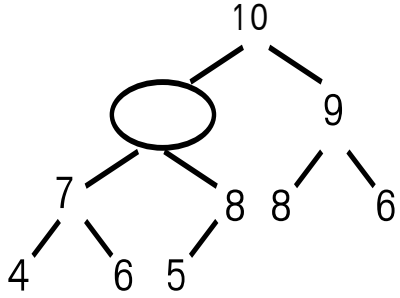
Delete node in heap tree

- การลบโหนดออกจากฮีป
  - นำค่าจากโหนดสุดท้ายมาแทนที่โหนดที่ต้องการลบ และลบโหนดสุดท้ายทิ้ง
  - ปรับต้นไม้ที่โหนดนั้น ให้เป็นฮีปอีกครั้ง ด้วยการเปรียบเทียบค่าที่โหนดนั้น กับค่าของโหนดลูก เพื่อสลับตำแหน่งที่เหมาะสม เรียกวิธีการนี้ว่า **Sift down**
- **Example ลบโหนด 9 อันแรก ออกจากฮีป**

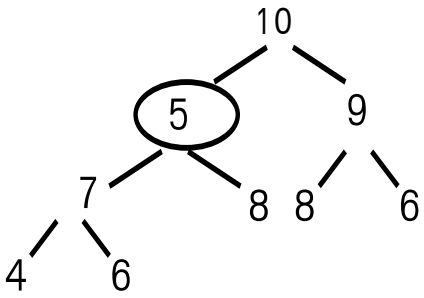




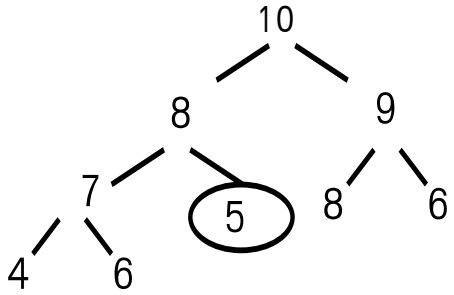
(1)



(2)



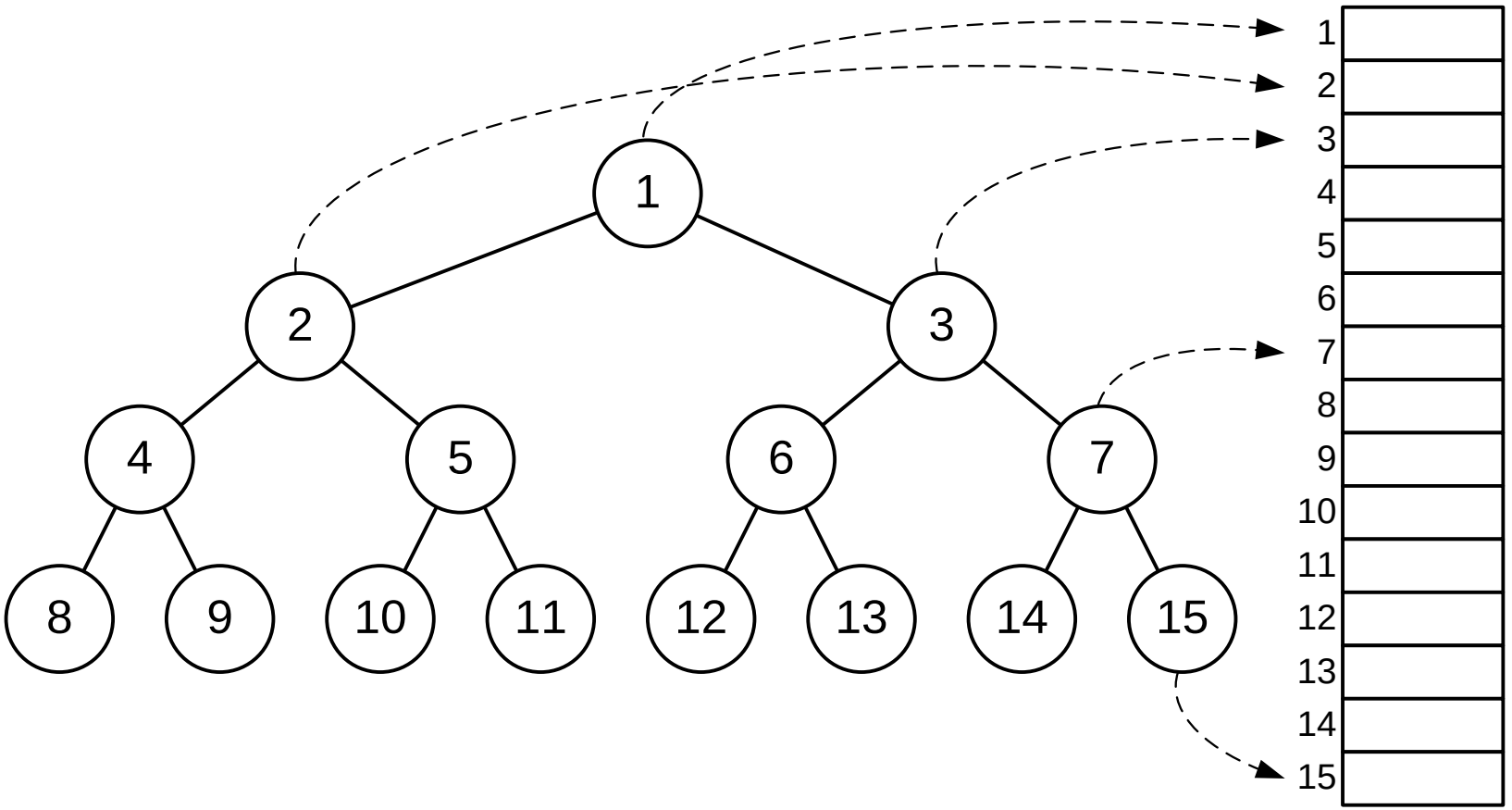
(3)



(4)

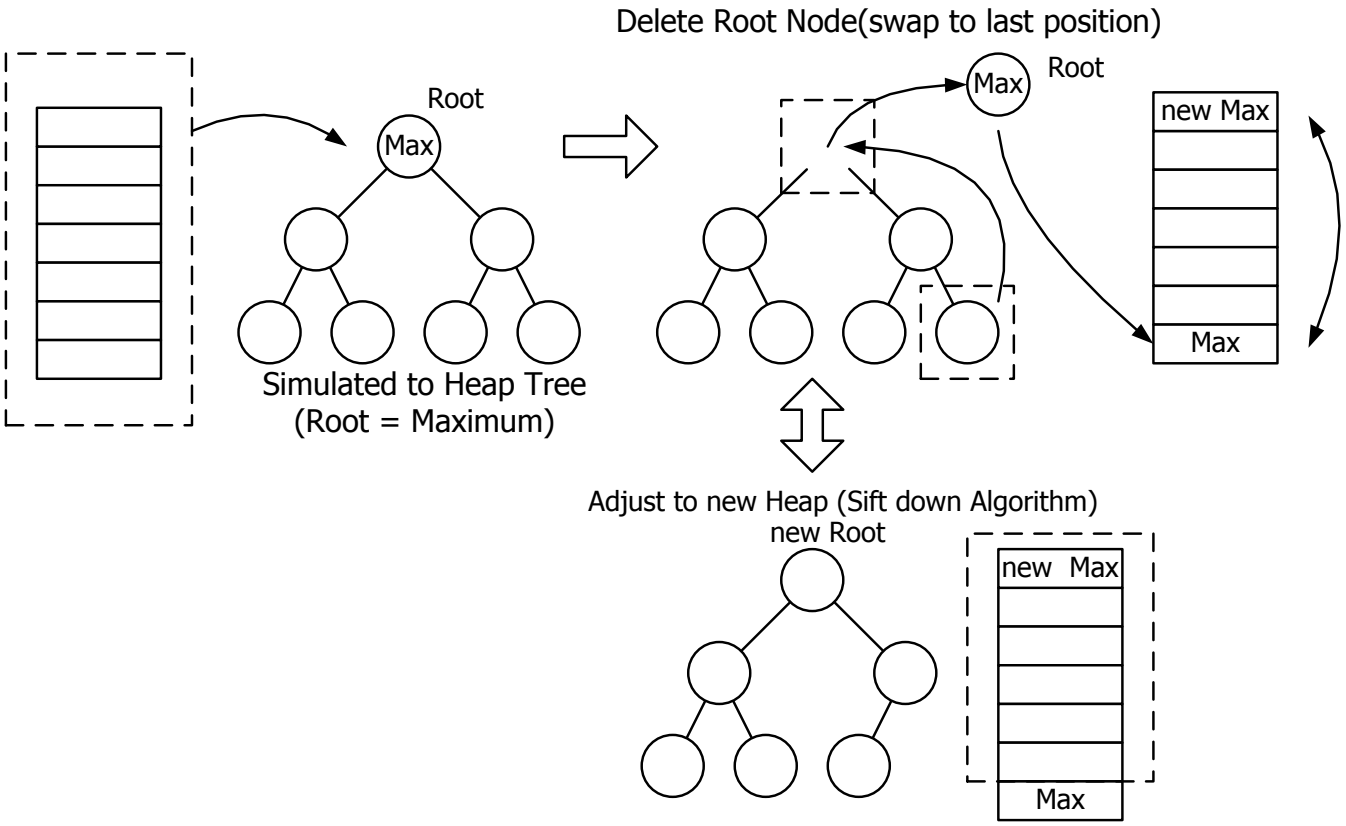
Representation heap with memory array

- การแทนที่ตำแหน่งของโหนดใน **Complete Binary Tree** โดยใช้อาร์เรย์



Heap Sort in memory array

- การเรียงลำดับข้อมูลในอาร์เรย์โดยใช้เทคนิคฮีป
  - สมมุติค่าที่อยู่ในอาร์เรย์เป็นต้นไม้แบบฮีป
  - ดึงค่าที่อยู่ใน **root** ไปเก็บไว้เป็นค่าที่เรียงลำดับแล้ว**(Sorted Array)**
  - ปรับต้นไม้ที่เหลือให้เป็น **heap** แล้วเริ่มต้นใหม่ จนกว่าจะหมด



Algorithm Heap Sort

- เรียงลำดับข้อมูลที่อยู่ในอาร์เรย์โดยใช้เทคนิค Heap Sort

1. สมมุติอาร์เรย์เป็น Complete Binary Tree แล้วปรับย้ายตำแหน่งข้อมูลโดยการ sift down จนกระทั่งมีคุณสมบัติเป็น Heap Tree *(Make Heap)*

```
for (i = n/2; i >=1; i--)
    siftDown(i, n);
```

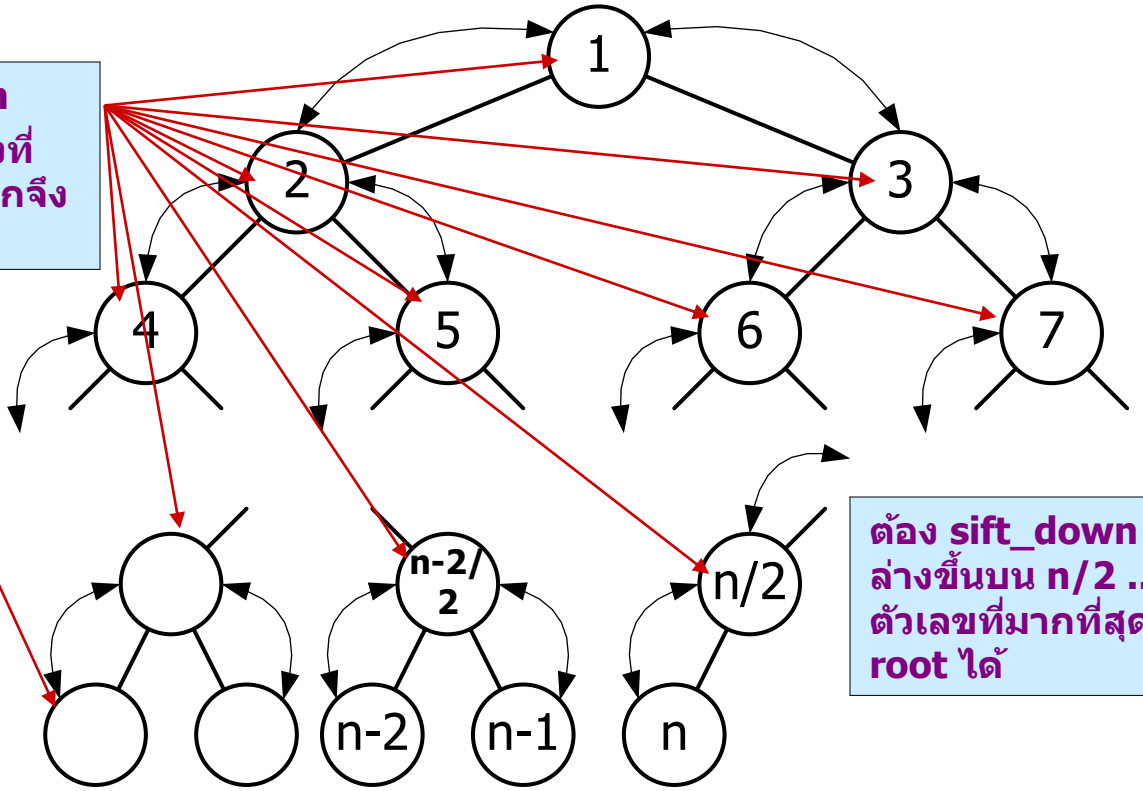
2. ดึง(ลบ)ข้อมูลที่ตำแหน่ง root node(ตำแหน่งแรก ซึ่งจะมีค่าข้อมูลสูงสุด)
   - สลับตำแหน่ง root กับตำแหน่งสุดท้าย*(Swap)* จะทำให้ตำแหน่งสุดท้ายเป็นข้อมูลที่มีค่ามากที่สุด
   - ลดขนาดของ tree ลง 1 โหนด ทำให้ตำแหน่งสุดท้ายไม่อยู่ใน tree (ถือเป็นตำแหน่งที่เรียงลำดับแล้ว) แล้วปรับ tree ที่เหลือให้กลายเป็น Heap Tree ใหม่ *(ทำ Sift Down ใหม่)*
3. ทำซ้ำขั้นตอนที่ 2 จนกระทั่งสำเร็จ (เหลือข้อมูลตัวเดียว)

```
for (i = n; i > 1; i--)
   { swap(i, 1);
     siftDown(1,i-1); }
```

Make Heap

- การสร้าง heap tree ต้องทำให้โหนดบนมีค่ามากกว่าโหนดล่าง
  - ต้องสลับตัวเลขในโหนด(sift down) จากล่างขึ้นบนจนกว่าจะมีคุณสมบัติเป็น heap

โหนดที่ต้อง sift_down ตั้งแต่ 1 - n/2 ตำแหน่งที่เลย n/2 จะไม่มีโหนดลูกจึงไม่ต้อง sift down



ต้อง sift_down ไล่จากโหนดล่างขึ้นบน n/2 .. 1 เพื่อให้ตัวเลขที่มากที่สุดลอยขึ้นไปที่ root ได้

```
for (i = n/2; i >=1; i--)
      siftDown(i, n);
```

Sift_down

- Sift_down data at position i to n

```
void swap(int i, int j)
{long x ;
  x= data[i]; data[i] = data[j];  data[j] = x;
}
void siftDown( int i, int n)
{ int j , max;
  max = i; // j = i
  do { j = max;    // j is parent
       if ( (2*j <= n) &&(data[2*j] > data[max] )  )
          max = 2*j ; // 2*j is left child
       if ( (2*j+1 <= n) &&(data[2*j+1] > data[max] )  )
          max = 2*j+1; // 2*j+1 is right child
       if (j != max)
          swap(j, max);
     } while (j != max); // next heap
}
```

ใช้ max ในการจำตำแหน่งของโหนดลูกที่มีค่ามากที่สุด ระหว่าง j , 2*j , 2*j+1

ถ้าข้อมูลเก็บอยู่ที่ [0]..[n-1]

```
if ((2*j+1 <= n) && (data[2*j+1] > data[k]))
   k = 2*j+1;
if ((2 * j+2 <= n) && (data[2*j+2] > data[k]))
   k = 2*j+2;
```

สลับตำแหน่ง บน-ล่าง

ถ้ายังมีการเลื่อนโหนดแสดงว่ายังไม่เป็น heap ให้กลับไปตรวจสอบใหม่

Remove Root to Sorted Array

- สลับข้อมูลตัวแรก(root) กับตัวสุดท้าย(เทียบได้กับการดึงข้อมูลตัวที่มีค่ามากที่สุดออกมาจากต้นไม้)

```
void swap(int i, int j)
{ ..... }
void siftDown(int i, int n)
{ ..... }
void  heapSort ( ) //ข้อมูลอยู่ตำแหน่ง 0..count
{ int i ;
    for (i = count/2; i >=1; i--) /* Make Heap */
       siftDown(i, count);
    for (i = count; i > 1; i--)      /* Delete root */
      { swap(i, 1);                   /* Sort data[i] */
         siftDown(1, i-1); }
}
```

ถ้าข้อมูลเก็บอยู่ที่ [0]..[count-1]

```
for (i = count / 2; i >= 0; i--)
    siftDown(i, count-1);
for (i = count-1; i > 0; i--) {
    swap(i, 0);
siftDown(0, i - 1);
```