ฮีพทวิภาค
Binary heap

คิวแบบมีลำดับความสำคัญ (priority queue)

- queue ปกติจะมีลำดับตามการเข้า
- priority queue มีการปรับลำดับตามความสำคัญ
  - ความสำคัญสูงอยู่ที่หัว queue
  - ความสำคัญต่ำอยู่ที่ปลาย queue
- เมื่อมีการ enqueue item จะถูกจัดไปอยู่ในลำดับตามความสำคัญ
- ง่ายสุดในการสร้างคือ insert ด้วย O(n) แล้วจัดลำดับความสำคัญด้วยการ sort ใช้เวลา O(nlogn)
- ดีกว่านั้นคือใช้ ฮีพทวิภาค
  - enqueue ใช้ O(logn)
  - dequeue ใช้ O(logn)

ฮีพทวิภาค

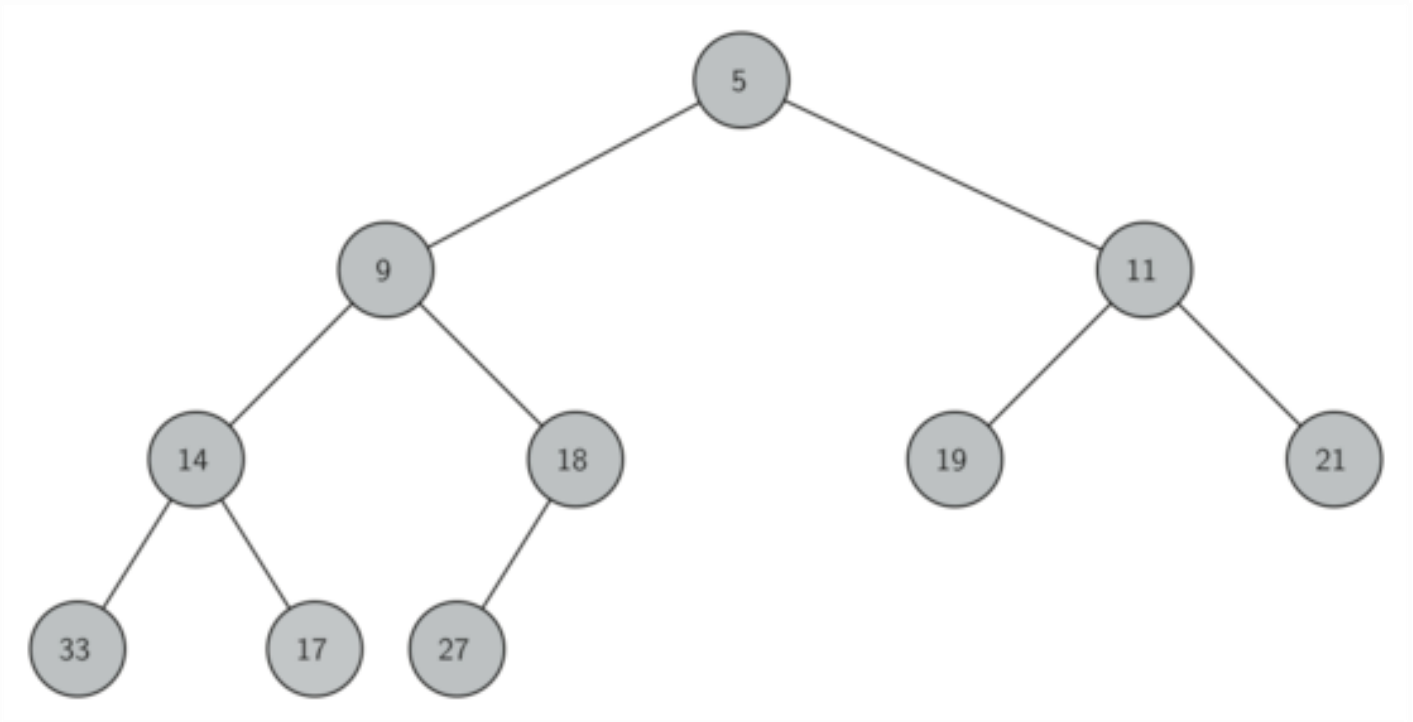
- มีความคล้ายกับต้นไม้
- การสร้างจริงๆ ใช้แค่ list
- มี 2 ประเภทคือ
  - max heap ความสำคัญสูดสุดคือมีค่า key มากสุด
  - min heap ความสำคัญสูงสุดคือมีค่า key น้อยสุด

method ของฮีพทวิภาค

- BinaryHeap() สร้างฮีพว่าง
- insert(k) เพิ่ม item ใน ฮีพ
- findMin() คืนค่า item ที่มีค่า key น้อยที่สุด
- delMin() คืนค่า item ที่มีค่า key น้อยที่สุดแล้วลบออกด้วย
- isEmpty() ถามว่าฮีพว่างหรือไม่
- size() คืนค่าจำนวน item ในฮีพ
- buildHeap(list) สร้างฮีพใหม่จาก list

โครงสร้าง

- คล้ายต้นไม้ที่รักษาความสมดุล
  - ปม (node) ฝั่งซ้ายและฝั่งขวามีจำนวนเท่ากัน
  - ใช้ต้นไม้สมบูรณ์
    - แต่ละลำดับชั้นมี node เต็ม
    - ชั้นสุดท้ายเรียกว่าใบเรียงวางจากซ้ายไปขวา

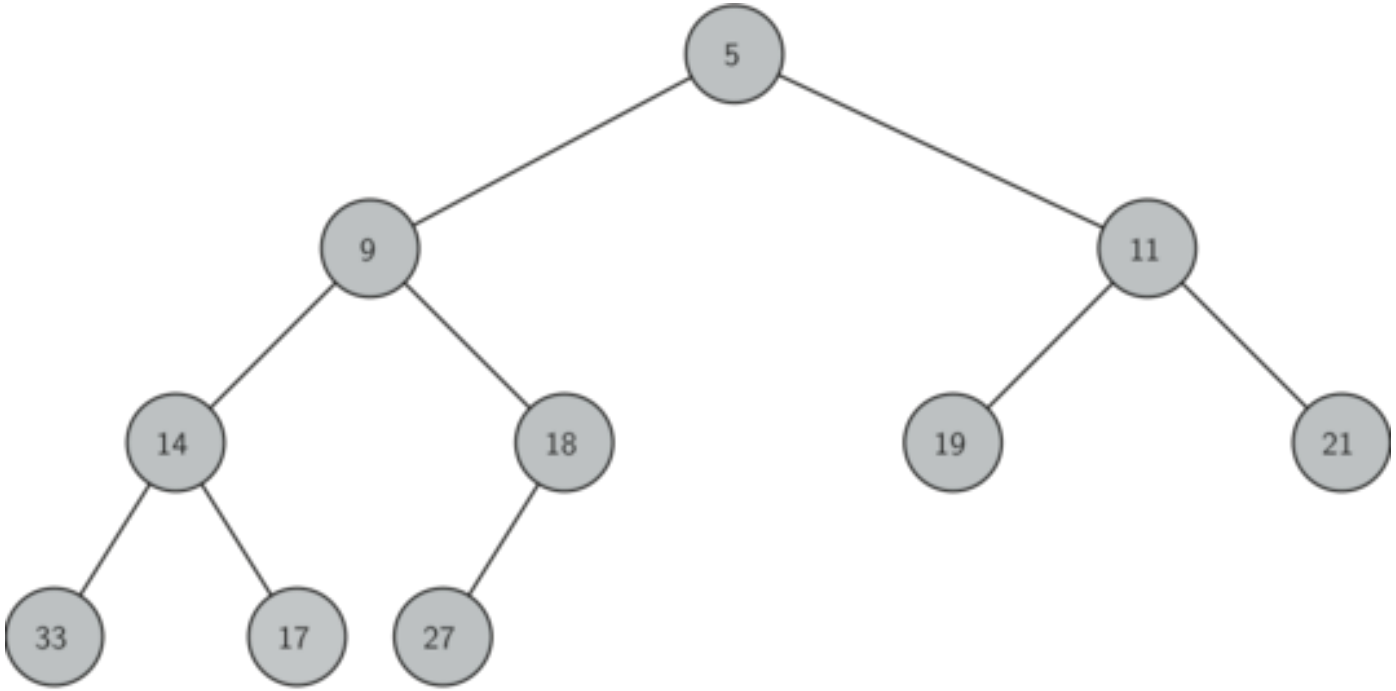


ต้นไม้สมบูรณ์

- สร้างได้ด้วย list
- ที่ parent node p
  - left child อยู่ที่ตำแหน่ง 2p
  - right child อยู่ที่ตำแหน่ง 2p + 1
- ที่ node p ใดๆ parent อยู่ที่ p // 2

คุณสมบัติของฮีพ

- key ที่ parent มีค่าน้อยกว่าหรือเท่ากับ children เสมอ



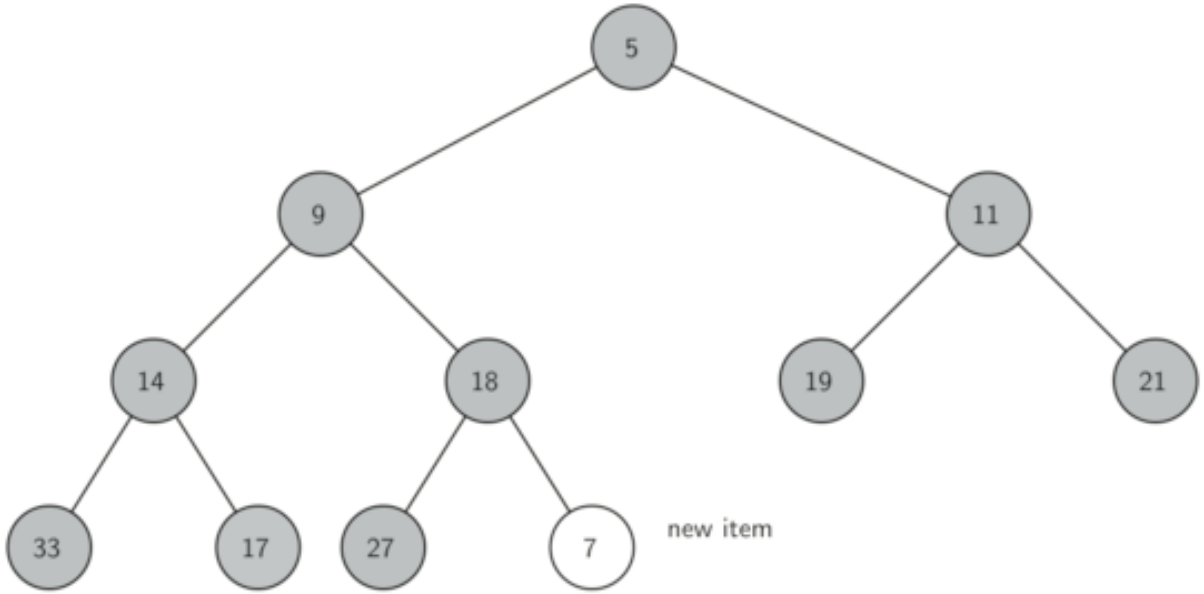
0	5	9	11	14	18	19	21	33	17	27	
0	1	2	3	4	5	6	7	8	9	10	11

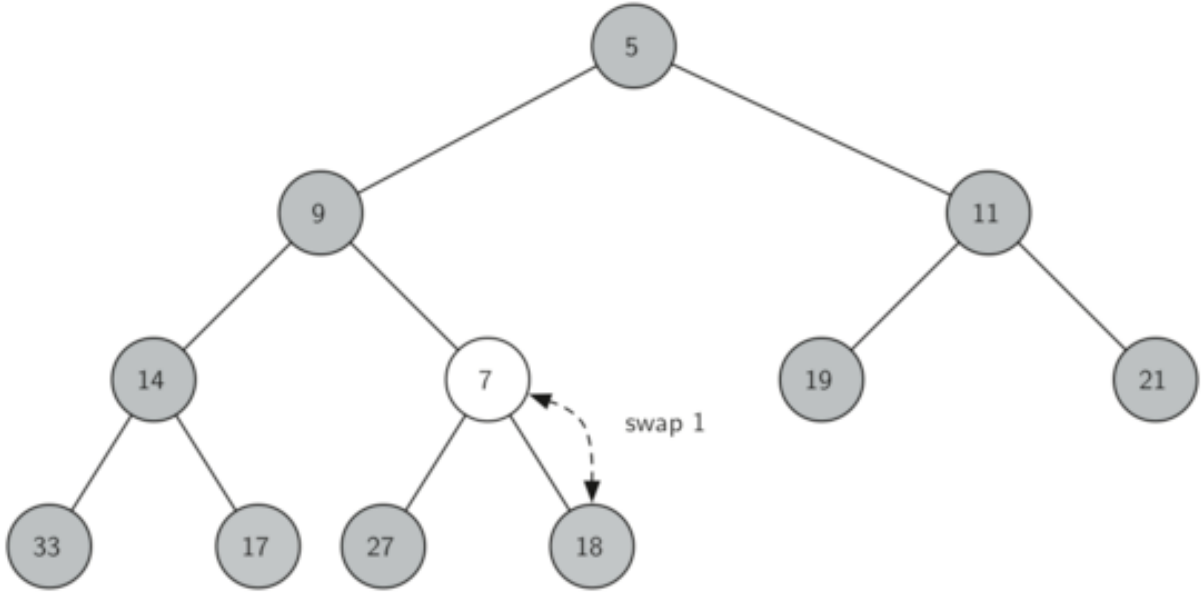
การสร้าง class ของฮีพ

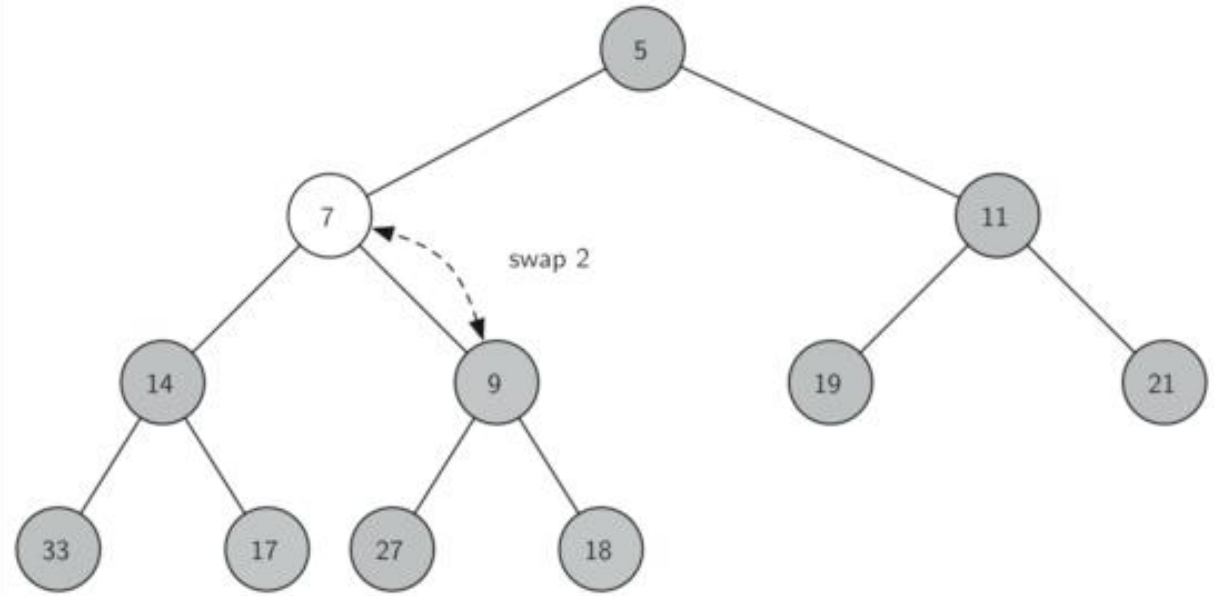
```
class BinHeap:
    def __init__(self):
        self.heapList = [0]
        self.currentSize = 0
```

insert

- ทำโดย append item เข้าไปใน list
- รักษาสภาพความสมบูรณ์เสมอ
- ดัน item กลับขึ้นด้านบนเพื่อให้อยู่ในลำดับที่ถูกต้อง (percolate up)
  - parent มี key น้อยกว่าหรือเท่ากับ children เสมอ


5
9
11
14
18
19
21
33
17
27
7
new item

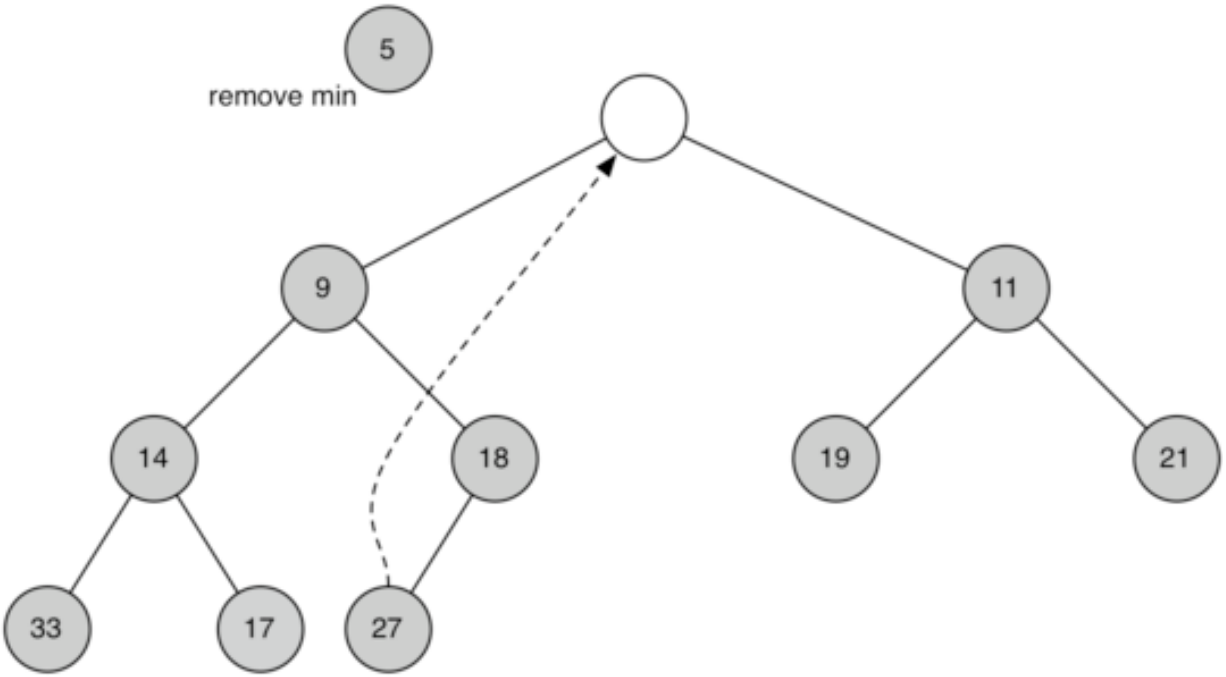

5
9
11
14
7
19
21
swap 1
33
17
27
18

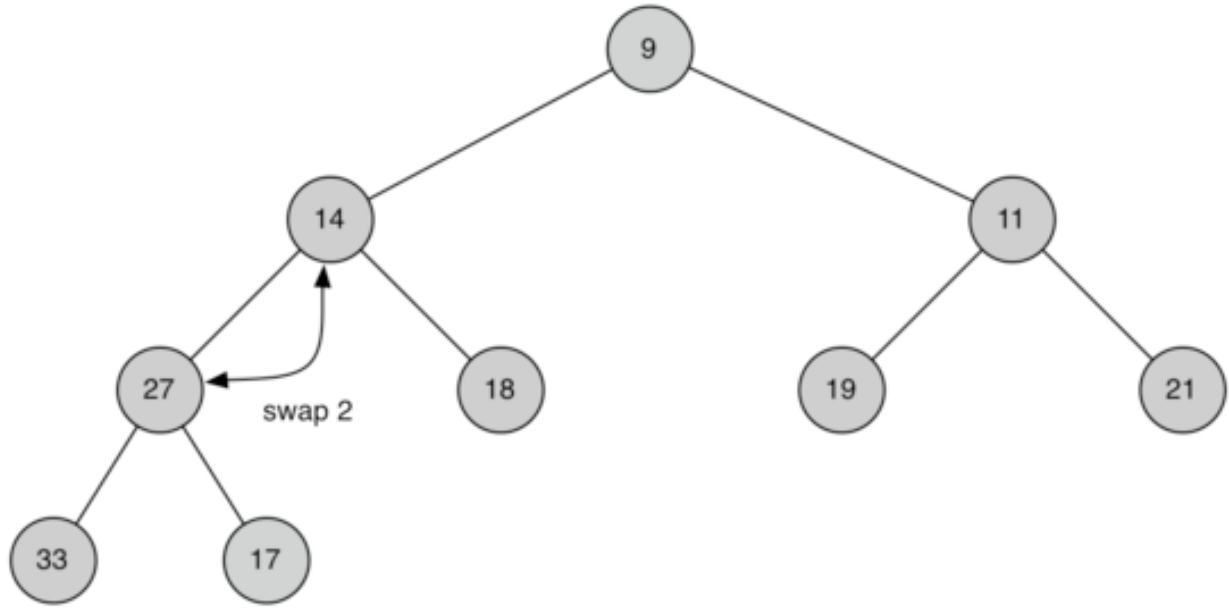

5
7
11
swap 2
14
9
19
21
33
17
27
18

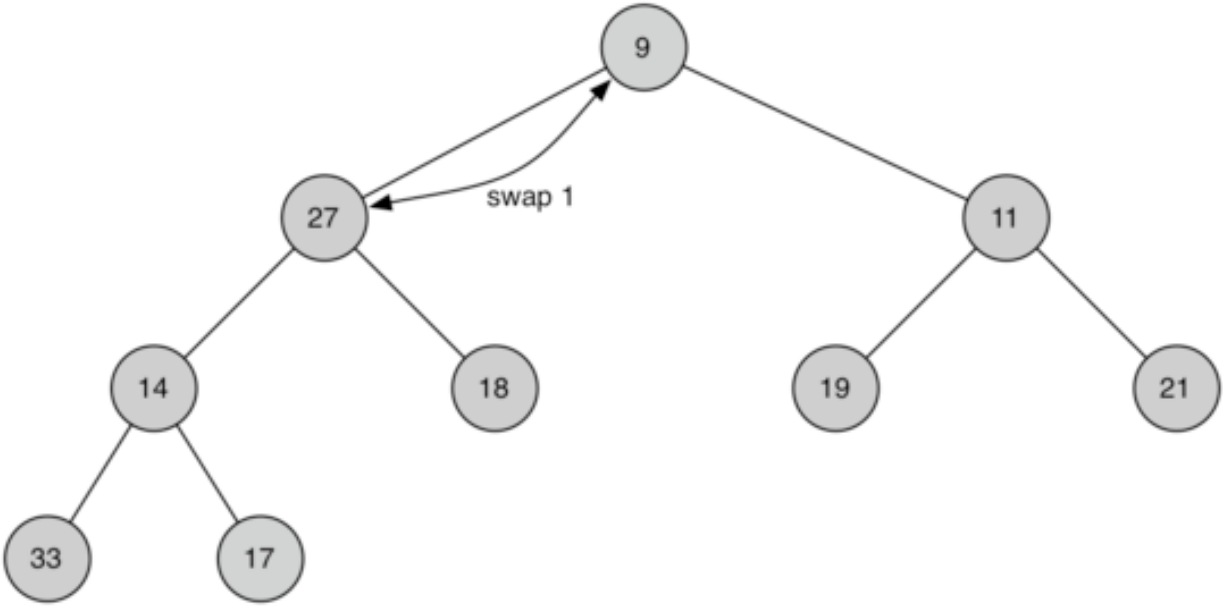
```
def percUp(self,i):
    while i // 2 > 0:
      if self.heapList[i] < self.heapList[i // 2]:
         tmp = self.heapList[i // 2]
         self.heapList[i // 2] = self.heapList[i]
         self.heapList[i] = tmp
      i = i // 2


def insert(self,k):
    self.heapList.append(k)
    self.currentSize = self.currentSize + 1
    self.percUp(self.currentSize)
```

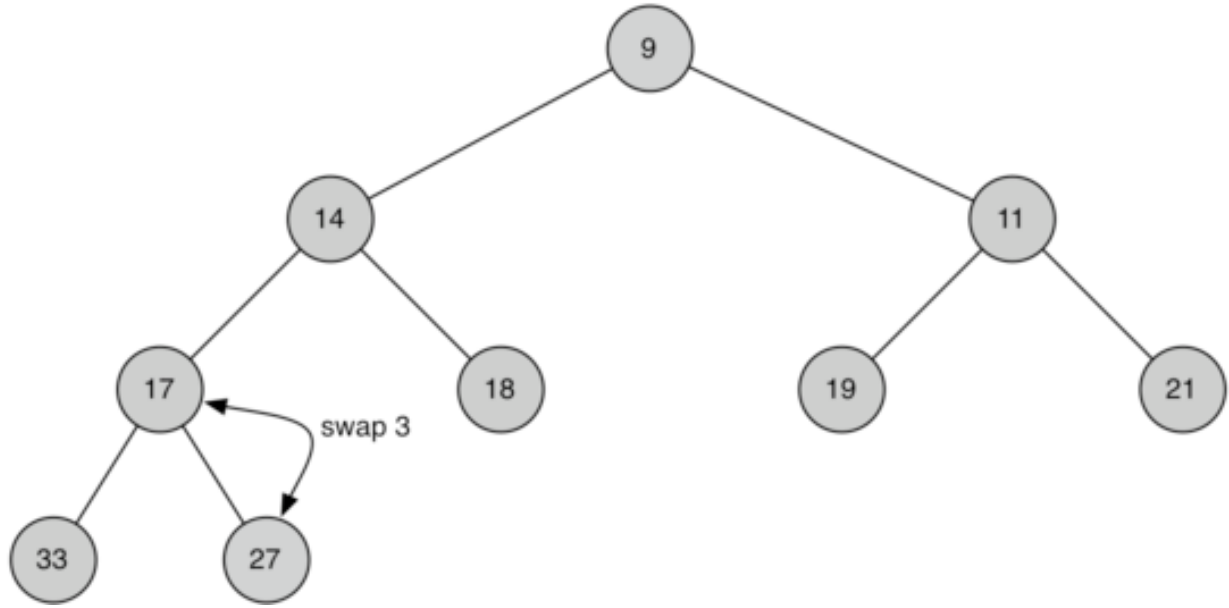
delMin

- ต้องหา node ที่มี key น้อยสุดขึ้นมาแทนที่
- เอา node สุดท้ายขึ้นมาก่อน
- ไล่กลับลงไปในตำแหน่งที่ (percolate down)


5
remove min
9
11
14
18
19
21
33
17
27


9
14
11
27
swap 2
18
19
21
33
17


9
27
swap 1
11
14
18
19
21
33
17


9
14
11
17
18
19
21
swap 3
33
27

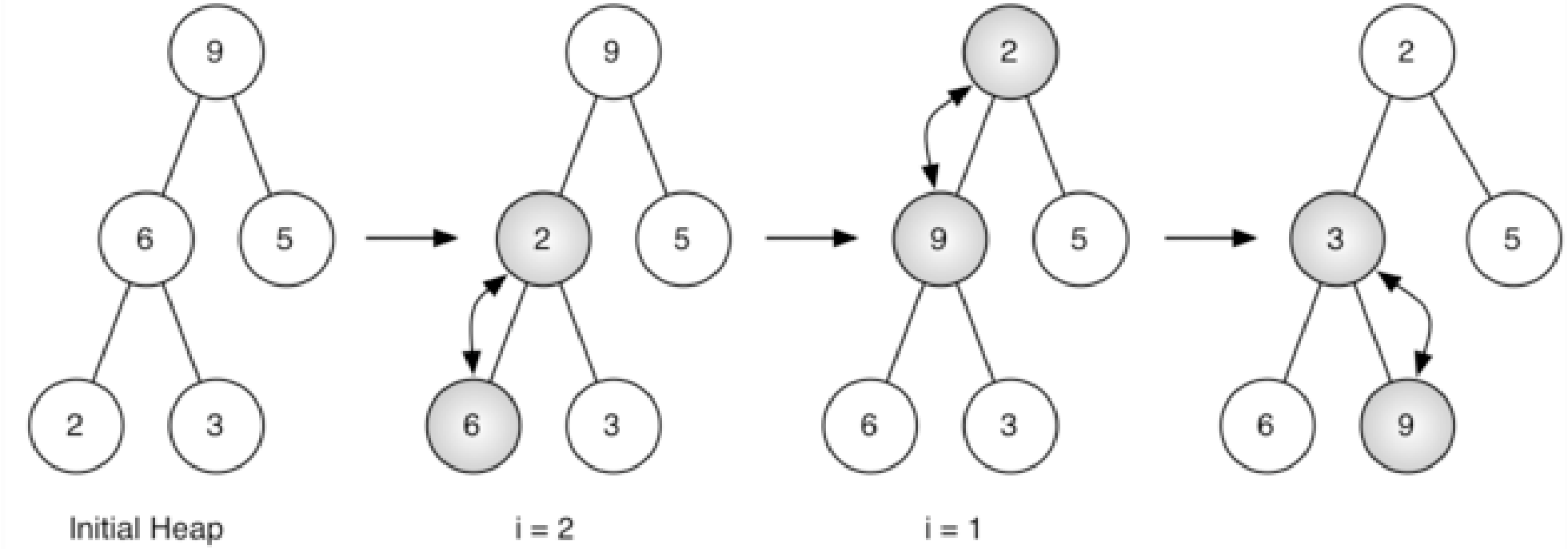
```
def percDown(self,i):
    while (i * 2) <= self.currentSize:
        mc = self.minChild(i)
        if self.heapList[i] > self.heapList[mc]:
            tmp = self.heapList[i]
            self.heapList[i] = self.heapList[mc]
            self.heapList[mc] = tmp
        i = mc
```

```
def minChild(self,i):
    if i * 2 + 1 > self.currentSize:
        return i * 2
    else:
        if self.heapList[i*2] < self.heapList[i*2+1]:
            return i * 2
        else:
            return i * 2 + 1
```

```
def delMin(self):
    retval = self.heapList[1]
    self.heapList[1] = self.heapList[self.currentSize]
    self.currentSize = self.currentSize - 1
    self.heapList.pop()
    self.percDown(1)
    return retval
```

สร้างฮีพจาก list

- ง่ายสุดคือค่อยๆ insert ทีละค่าใช้ O(nlogn)
- สร้างจาก list ทั้งหมดใช้ O(n)
- เริ่มจากครึ่งของ list
  - อีกครึ่งที่เหลือเป็น leaf node ที่ไม่มี child
- ขยับไปข้างหน้า list เรื่อยๆ (วิ่งขึ้นบนของ tree) แล้วไล่กลับลงมา




```
i = 2  [0, 9, 5, 6, 2, 3]

i = 1  [0, 9, 2, 6, 5, 3]

i = 0  [0, 2, 3, 6, 5, 9]
```

```
def buildHeap(self,alist):
    i = len(alist) // 2
    self.currentSize = len(alist)
    self.heapList = [0] + alist[:]
    while (i > 0):
        self.percDown(i)
        i = i - 1
```