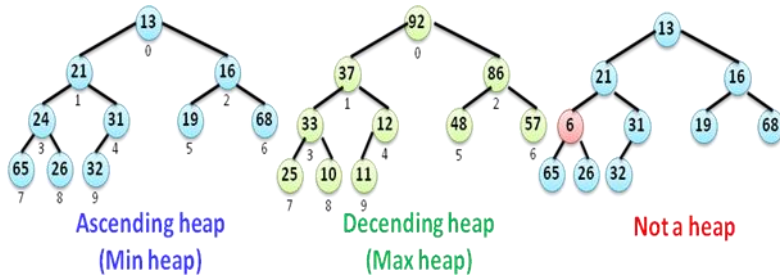


## Lab 8 : Heap &amp; Lab 9 : Sort

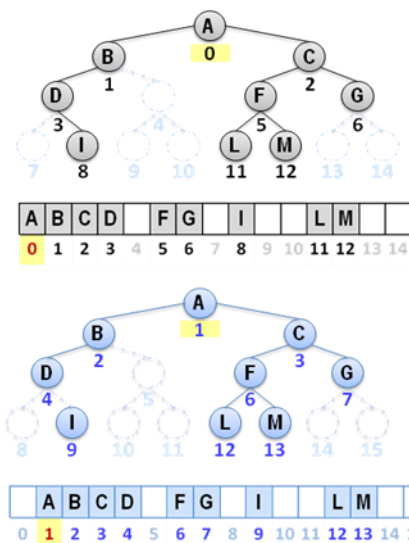
วัตถุประสงค์ ฝึกหัดเขียน Sort แบบต่างๆ



**Binary Heap** : complete Binary Tree ซึ่ง key ของ node ใดๆ

1.  $\leq$  key ของ decendents ของ มัน : **Min heap**  
เช่น 13 น้อยกว่าลูกหลานทั้งหมดของมัน
2.  $\geq$  key ของ decendents ของ มัน : **Max heap**  
เช่น 37 มากกว่าลูกหลานทั้งหมดของมัน

เนื่องจากเป็น complete binary tree จึงควร implement ด้วย data structure แบบ \_\_\_\_\_



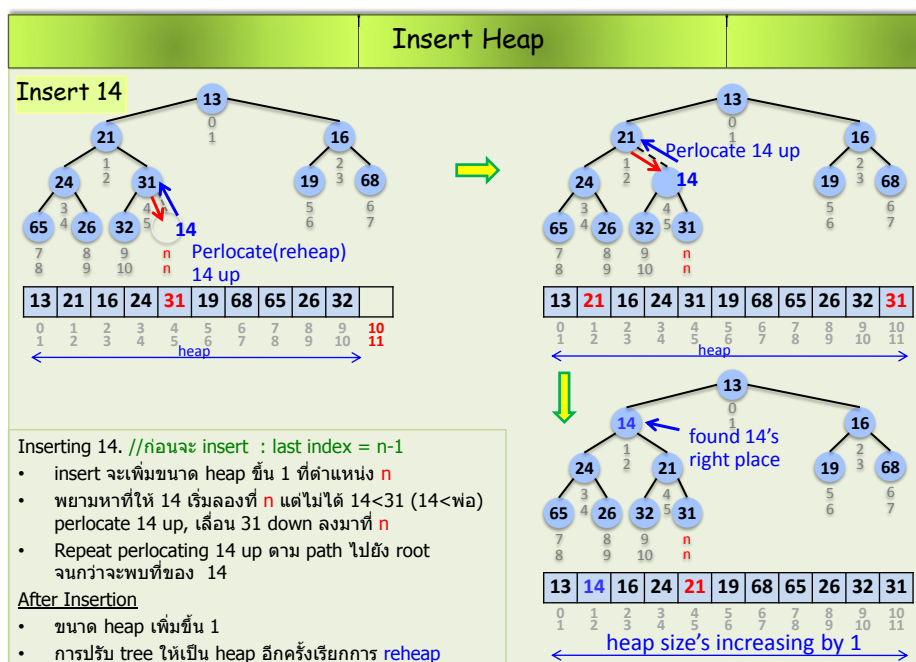
ซึ่งโครงสร้างแบบ implicit array (sequential array) นี้ เราจะนับตำแหน่งของ node นิยมทำกับ 2 แบบ คือ ให้ root เริ่มที่ 0 (รูปซ้าย) และ ให้ root เริ่มที่ 1 (รูปขวา) นับตำแหน่งไปเรื่อยๆ เรียงตามลำดับที่ละ level จากซ้ายไปขวา นับไม่เว้น node ที่ไม่มี (ในรูปเป็นเส้นประ) แต่ละ node จึงมีตำแหน่งกำกับดังรูป และใช้ implicit array (sequential array) เก็บข้อมูล data ของแต่ละ node ใน index ตามตำแหน่งของมัน

จะเห็นว่า data structure แบบ implicit array นี้ เก็บเฉพาะ data ไม่จำเป็นต้องเก็บ link เนื่องจากสามารถคำนวณได้

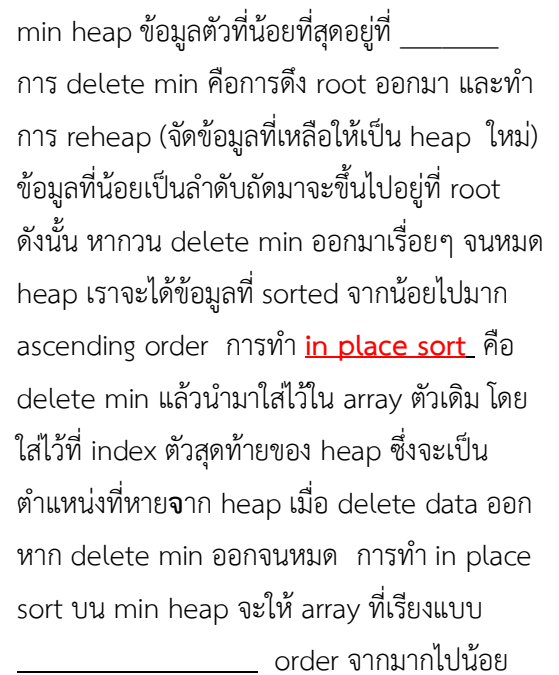
เริ่ม root ที่ index 0      เริ่ม root ที่ index 1

ลูกข้างซ้ายของ node ที่ index i อยู่ที่ index \_\_\_\_\_

„ ขวา „ \_\_\_\_\_



Algorithm ในการสร้าง heap ใช้วิธี insert data เข้าไปใน heap ที่ละตัว ซึ่งทำให้ size ของ heap เพิ่มขึ้นจาก index สุดท้ายไป 1 ตัว ในรูปกำลัง insert 14 index ที่เพิ่มขึ้น n คือ 10 (root เริ่มที่ index 0) / 11 (root เริ่มที่ index 1) โดยดูว่าสามารถ insert data เข้าที่ตำแหน่ง n ได้หรือไม่ (ผิดกฎของ heap หรือไม่) หากไม่ได้ จะทำการ perlocate data up ขึ้นไปตามทางพ่อของมันไปยัง root ทำให้การ insert data 1 ตัวทำงานอย่างมากเท่ากับความสูงของ tree คือ  $\log_2 n$  ดังนั้น insert n ตัว มีลำดับเป็น



1) เพื่อทดสอบความถูกต้องของ array heap เขียนฟังก์ชันเพื่อพิมพ์ tree

- print ( ) พิมพ์ array a ของ type T ใดๆ ตั้งแต่ index 0 ถึง i
- print90 ( ) พิมพ์รูป tree ของ array a ของ type T ใดๆ ตั้งแต่ index 0 ถึง i ในรูปหมุนซ้าย 90 องศา หากนศ.ใช้โครงสร้างที่ root อยู่ที่ index 1 ปรับฟังก์ชันให้พิมพ์ตั้งแต่ index 1

68	65	32	24	26	21
	19				
	32				
	21				
68					
	26				
		14			
65					
		16			
	24				
		13			

- 2) ทำ Heap Sort โดยเขียนฟังก์ชัน deleteMin() เพื่อทำการดึงค่า root ของ heap ในตัวอย่างทำ inplace sort heap ปรับ reheap พร้อมทั้งลดค่า last index ของ heap ลง ทดสอบความถูกต้องโดยให้พิมพ์ array heap ทั้งหมด วน loop deleteMin จนหมดทุกตัวมาใส่ที่ array a พิมพ์ a จะได้ ascending list พิมพ์ h จะได้ decending list ข้างล่างเป็นตัวอย่าง output

```
***deleteMin***
input heap:
13 14 21 19 16 65 24 68 26 32
  24
    21
      65
        13
          16
            32
              14
                26
                  19
                    68
===== heap sort =====
deleteMin =13 FindPlaceFor 32
14 16 21 19 32 65 24 68 26 13
  24
    21
      65
        14
          32
            13
              16
                26
                  19
                    68
```

```
deleteMin =14 FindPlaceFor 26
16 19 21 26 32 65 24 68 14 13
  24
    21
      65
        16
          32
            13
              19
                14
                  26
                    68
deleteMin =16 FindPlaceFor 68
19 26 21 68 32 65 24 16 14 13
  24
    21
      65
        19
          32
            13
              26
                14
                  68
                  16
```

```
deleteMin =19 FindPlaceFor 24
21 26 24 68 32 65 19 16 14 13
  19
    24
      65
        21
          32
            13
              26
                14
                  68
                    16
deleteMin =21 FindPlaceFor 65
24 26 65 68 32 21 19 16 14 13
  19
    65
      21
        24
          32
            13
              26
                14
                  68
                    16
```

```
deleteMin =24 FindPlaceFor 32
26 32 65 68 24 21 19 16 14 13
  19
    65
      21
        26
          24
            13
              32
                14
                  68
                    16
deleteMin =26 FindPlaceFor 68
32 68 65 26 24 21 19 16 14 13
  19
    65
      21
        32
          24
            13
              68
                14
                  26
                    16
```

```
deleteMin =32 FindPlaceFor 65
65 68 32 26 24 21 19 16 14 13
  19
    32
      21
        65
          24
            13
              68
                14
                  26
                    16
deleteMin =65 FindPlaceFor 68
68 65 32 26 24 21 19 16 14 13
  19
    32
      21
        68
          24
            13
              65
                14
                  26
                    16
===== Sorting a =====
13 14 16 19 21 24 26 32 65 68
```

[การทดลอง 2](#) เขียน Sorting แบบต่างๆ ตาม algorithm ที่เรียน ทดลองโดยกำหนด data เองให้ครอบคลุมทุกกรณี หากทำไม่ได้ ดู Pseudocode คิดตาม algorithm ถึง complexity ของแต่ละ sort ตามที่เรียน ลองใส่ code print จำนวนการ compare data ในแต่ละ pass ในกรณีต่างๆ best case และ worst case

### Bubble Sort



**Ascending Order** จาบน้อย -> มาก

8	3	9	4	5	Original File
3	8	9	4	5	
3	8	9	4	5	
3	8	4	9	5	
3	8	4	5	9	After 1 <sup>st</sup> pass: the biggest, 9, floats up
3	8	4	5	9	From 1 <sup>st</sup> pass
3	8	4	5	9	
3	4	8	5	9	
3	4	5	8	9	After 2 <sup>nd</sup> pass: 2nd biggest, 8, floats up
3	4	5	8	9	From 2 <sup>nd</sup> pass
					After 3 <sup>rd</sup> pass: 3 <sup>rd</sup> biggest 5 floats up

**Key idea :**  
**Bubble the largest,**

- Scan from the left, swap unordered successive elements.

**Each scan (pass) :**

- Floats the largest (or smallest) up in the right place.
- Remains 1-element-shorter file to process.
- Repeat bubbling.

**Then the next largest, etc.**

ศ.ดร.บุญธีร์ เครือเดระกู๋ ศ.ภกฤษณ์ ศิริบุรณ์ KMITL 01076249 Data Structures & Algorithms [home](#) 4

### Straight Selection Sort (Pushed Down Sort)

6	9	8	5	4	Original File : size n
6	4	8	5	9	After pass 1
6	4	5	8	9	After pass 2
5	4	6	8	9	...
4	5	6	8	9	Sorted File: After Pass ____

**Ascending Order** เรียงจากน้อย -> มาก


**Key idea :**

- Scan to select the biggest /smallest, swap with the 1<sup>st</sup> /last element.

**Each scan (pass) :**

- Floats the largest (or smallest) up in the right place.
- Remains 1-element-shorter file to process.


- Repeat selection.



**I'll choose the biggest first.**

ศ.ดร.บุญธีร์ เครือเดระกู๋ ศ.ภกฤษณ์ ศิริบุรณ์ KMITL 01076249 Data Structures & Algorithms [home](#) 8

### Insertion Sort



**Key idea :**

- Scan to insert the card in sorted file in hand.

**Each scan (pass) :**

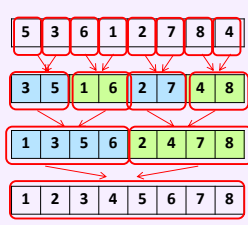
- Compare to insert the the right place to insert.
- If that is not the place, move that data next to its old place to give the room for the inserted card.
- Remains 1-element-shorter file to process.
- Repeat insertion.

8	6	7	5	9
6	8	7	5	9
6	7	8	5	9
5	6	7	8	9
5	6	7	8	9

**Ascending Order** จาบน้อย -> มาก

ศ.ดร.บุญธีร์ เครือเดระกู๋ ศ.ภกฤษณ์ ศิริบุรณ์ KMITL 01076249 Data Structures & Algorithms [home](#) 12

### Merge Sort



**Key idea :**

- Merge sorted successive pair to be sorted bigger one.
- Merge size 1 successive pair -> sorted size 2
- Merge size 2 successive pair -> sorted size 4
- Merge size 4 successive pair -> sorted size 8

**#comparisons = \_\_\_\_**  
**#assignments = \_\_\_\_**  
**Merge = O(\_\_\_\_)**

ศ.ดร.บุญธีร์ เครือเดระกู๋ ศ.ภกฤษณ์ ศิริบุรณ์ KMITL 01076249 Data Structures & Algorithms [home](#) 28

### Quick Sort

**Key idea :**

1. Choose the pivot
  - 1<sup>st</sup> element
  - median of 3
  - average
2. Partition :
  - Pivot partitions files into 2 halves
  - Left half < pivot
  - Right half > pivot
  - Pivot goes to its right place
3. Repeat partitioning both left & right half

The diagram illustrates the partitioning step of Quick Sort. It shows an initial array of 10 elements: [5, 1, 4, 9, 6, 3, 8, 2, 7, 0]. The first element, 5, is chosen as the pivot. The array is then partitioned into two halves: the left half [3, 1, 4, 0, 2] and the right half [8, 6, 7, 9]. The pivot (5) is moved to its correct position (index 5). The final sorted array is [0, 1, 2, 3, 4, 5, 6, 7, 8, 9].

Ordered List

รศ.ดร.บุญธีร์ บริรักษ์ราษฎร์

KMITL

01076249 Data Structures & Algorithms [home](#)

35