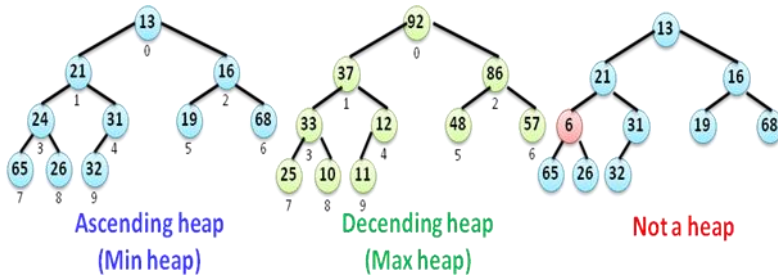


## Lab 10 : Heap &amp; Lab 11 : Sort

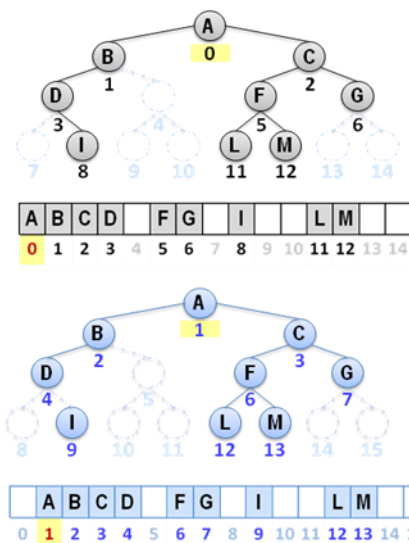
วัตถุประสงค์ ฝึกหัดเขียน Sort แบบต่างๆ



**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 \_\_\_\_\_



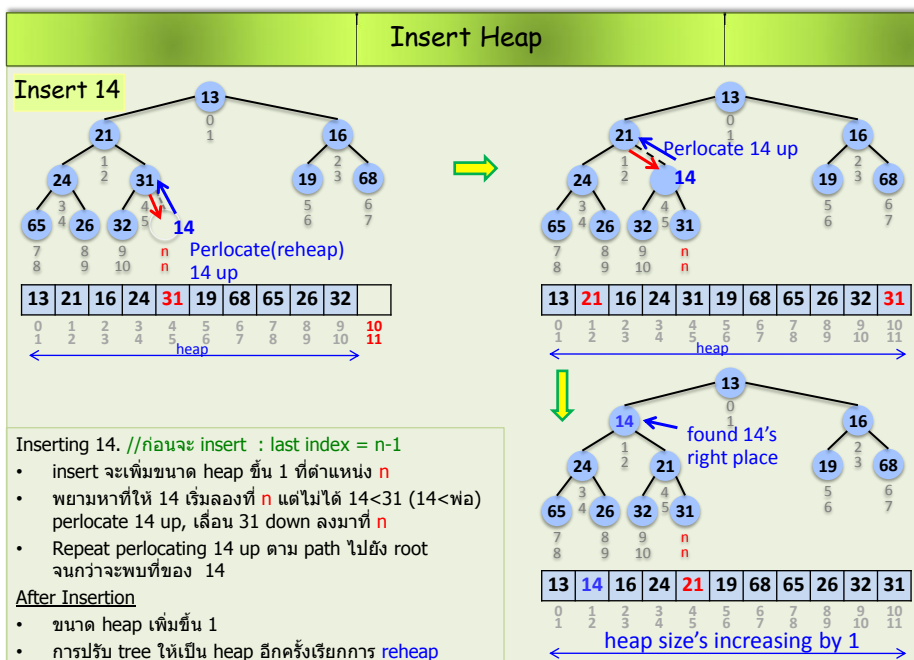
ซึ่งโครงสร้างแบบ implicit array (sequential array) นี้ เราจะนับตำแหน่งของ node นิยมทำกับ 2 แบบ คือ ให้ root เริ่มที่ 0 (รูปซ้าย) และ ให้ root เริ่มที่ 1 (รูปขวา) นับตำแหน่งไปเรื่อยๆ เรียงตามลำดับที่ละ level จากซ้ายไปขวา นับไม่เว้น node ที่ไม่มี (ในรูปเป็นเส้นประ) แต่ละ node จึงมีตำแหน่งกำกับดังรูป และใช้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

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

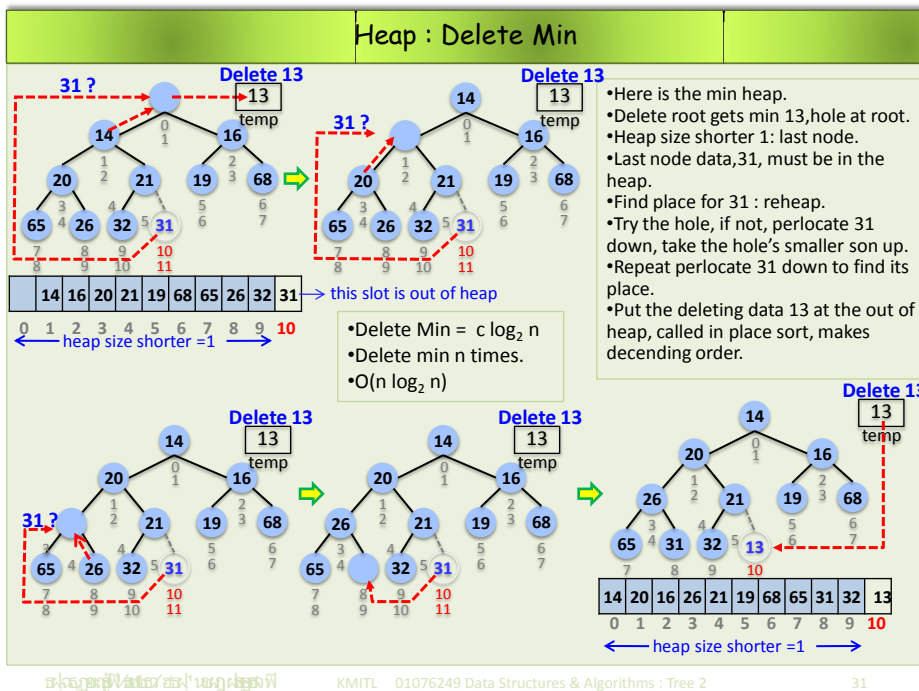
เริ่ม root ที่ index 0      เริ่ม root ที่ index 1

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



min heap ข้อมูลตัวที่น้อยที่สุดอยู่ที่ \_\_\_\_\_

การ delete min คือการดึง root ออกมา และทำการ reheap (จัดข้อมูลที่เหลือให้เป็น heap ใหม่) ข้อมูลที่น้อยเป็นลำดับถัดมาจะขึ้นไปอยู่ที่ root ดังนั้น หากวน delete min ออกมาเรื่อยๆ จนหมด heap เราจะได้ข้อมูลที่ sorted จากน้อยไปมาก ascending order การทำ **in place sort** คือ delete min แล้วนำมาใส่ไว้ใน array ตัวเดิม โดยใส่ไว้ที่ index ตัวสุดท้ายของ heap ซึ่งจะเป็นตำแหน่งที่หายจาก heap เมื่อ delete data ออก หาก delete min ออกจนหมด การทำ in place sort บน min heap จะให้ array ที่เรียงแบบ \_\_\_\_\_ order จากมากไปน้อย

**การทดลอง 1** Heap Sort (in place) เขียนฟังก์ชันต่อไปนี้โดยคิด parameter เอง พร้อมทั้งทดสอบความถูกต้องด้วย

1) เพื่อทดสอบความถูกต้องของ heap เขียนฟังก์ชันเพื่อพิมพ์ tree

- พิมพ์ list l
- print90 ( ) พิมพ์รูป tree ของ list l ในรูปหมุนซ้าย 90 องศา

กำหนด input list l พร้อมรันฟังก์ชันในข้อ 1 เพื่อทดสอบ

l = [68, 65, 32, 24, 26, 21, 19, 13, 16, 14]

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

2) กำหนด heap h = [] เขียนฟังก์ชัน insert() เพื่อ insert data เข้าไปใน heap h ให้วน loop insert data จาก list l เข้าไปใน heap h ทีละตัวจนหมด แต่ครั้งที่ insert ให้พิมพ์ heap h เพื่อทดสอบ

|             |                      |                            |                               |
|-------------|----------------------|----------------------------|-------------------------------|
| insert 68   | insert 26            | insert 13                  | insert 14                     |
| 68          | 24 26 65 68 32       | 13 19 21 26 32 65 24 68    | 13 14 21 19 16 65 24 68 26 32 |
| 68          | 65                   | 24                         | 24                            |
| -----       | 24                   | 21                         | 21                            |
| insert 65   | 32                   | 65                         | 65                            |
| 65 68       | 26                   | 13                         | 13                            |
| 65          | 68                   | 32                         | 16                            |
| 68          | -----                | 19                         | 32                            |
| insert 32   | insert 21            | 26                         | 14                            |
| 32 68 65    | 21 26 24 68 32 65    | 68                         | 26                            |
| 65          | 24                   | insert 16                  | 19                            |
| 32          | 65                   | 13 16 21 19 32 65 24 68 26 | 68                            |
| 68          | 21                   | 24                         | -----                         |
| insert 24   | 26                   | 21                         |                               |
| 24 32 65 68 | 68                   | 65                         |                               |
| 65          | insert 19            | 13                         |                               |
| 24          | 19 26 21 68 32 65 24 | 32                         |                               |
| 32          | 24                   | 16                         |                               |
| 68          | 21                   | 19                         |                               |
|             | 65                   | 68                         |                               |
|             | 32                   |                            |                               |
|             | 26                   |                            |                               |

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

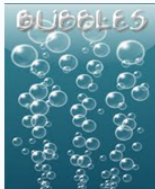
```
deleteMin =19 FindPlaceFor 24
21 26 24 68 32 65 19 16 14 13
  19
    24
      65
        21
          32
            13
              26
                14
                  68
                    16
deleteMin =21 FindPlaceFor 65
24 26 65 68 32 21 19 16 14 13
  19
    65
      21
        24
          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 ลงใส่ code print จำนวนการ compare data ในแต่ละ pass ในกรณีต่างๆ best case และ worst case หากทำไม่ได้ดูเฉลย code ในภาคทฤษฎี

## Bubble Sort



Ascending Order จากน้อย --&gt; 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 ตัวใหญ่สุด,

- Scan จากซ้าย , สลับที่คู่ที่ไม่ถูกลำดับ

แต่ละ pass (scan)

- ตัวใหญ่สุดจะลอยขึ้นไปทางขวาไปยังที่ที่มันควรจะอยู่
- ทำให้ file สั้นลง 1 ตำแหน่ง
- Repeat bubbling

ต่อมาก็ตัวใหญ่อันดับต่อมา

ทศ.ดร.บุญธีร์ เกียรติธราวุธ ทศ.ภกตวัน ศิริบุญรอด KMITL 01076249 Data Structures & Algorithms : Sorting [home](#) 4

## Straight Selection Sort (Pushed Down Sort)

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

Ascending Order เรียงจากน้อย --&gt; มาก

Algorithm :

1. Scan เพื่อเลือกตัวใหญ่สุด(หรือตัวเล็กสุด) สลับที่กับตัวสุดท้าย (หรือตัวแรก) ในแต่ละครั้งที่ scan
  - \* ตัวใหญ่สุด(หรือตัวเล็กสุด) ไปอยู่ที่ j ที่ และ
  - \* file สั้นลง 1 ตัว

2. ทำ 1 ซ้ำ

ตัวอย่าง pass แรก เลือกตัวใหญ่สุด ได้ 9 สลับกับตำแหน่งสุดท้ายคือ 1 ดังนั้นจะได้ 9 ไปอยู่ที่ j

pass 2 เลือกตัวใหญ่สุด ได้ 8 สลับกับตำแหน่งสุดท้ายคือ 5 ดังนั้นจะได้ 8 ไปอยู่ที่ j



I'll choose the biggest first.

ทศ.ดร.บุญธีร์ เกียรติธราวุธ ทศ.ภกตวัน ศิริบุญรอด KMITL 01076249 Data Structures & Algorithms : Sorting [home](#) 5

## Insertion Sort



Algorithm : ให้จินตนาการเหมือนหยิบไพ่ขึ้นมาทีละตัว เอาใส่ในมือที่เป็นไพ่ที่เรียงไว้แล้ว ในตัวอย่างไพ่ในมือเป็นสีส้ม

1. Scan เพื่อเลือกใส่(insert) ตัวใหม่เข้าที่ใน file ที่เรียงแล้ว

insert โดยเทียบไปทีละตัว (ในตัวอย่างเทียบจากขวาไปซ้าย)

หากตัวใหม่มีค่ามากกว่าให้เลื่อนมันออกมาทางขวา 1 ตำแหน่ง

2. ทำ 1 ซ้ำ จนใส่ไพ่หมด

|   |   |   |   |   |
|---|---|---|---|---|
| 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 |

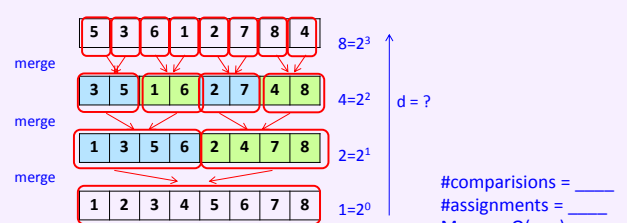
Ascending Order จากน้อย --&gt; มาก

ตัวอย่าง ครั้งแรกที่หยิบไพ่ในแรก ไม่ต้องทำอะไร เพราะมีเพียง 1 ใบ

1. pass ที่ 1 เอาไพ่ใบที่ 2 คือ 6 เข้าไปในมือ (หาที่ใส่ 6 อยู่) 6 < 8 เลื่อน 8 ออกมา เห็นเลข 6 แล้ว 6 < 5 ในที่เดิมของ 8
2. pass ที่ 2 เอาไพ่ใบที่ 3 คือ 7 เข้าไปในมือ (หาที่ใส่ 7 อยู่) 8 < 7 เลื่อน 8 ออกมา 6 < 7 ไม่ขยับกว่า 7 พบที่สำหรับ 7 ในที่เดิมของ 8
3. pass ที่ 3 เอาไพ่ใบที่ 4 คือ 5 เข้าไปในมือ (หาที่ใส่ 5 อยู่) 8 < 5 เลื่อน 8 ออกมา 7 < 5 เลื่อน 7 ออกมา 6 < 5 เลื่อน 6 ออกมา เลข 5 ในที่เดิมของ 6

ทศ.ดร.บุญธีร์ เกียรติธราวุธ ทศ.ภกตวัน ศิริบุญรอด KMITL 01076249 Data Structures & Algorithms : Sorting [home](#) 12

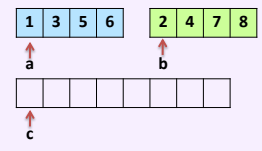
## Merge Sort



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



ทศ.ดร.บุญธีร์ เกียรติธราวุธ ทศ.ภกตวัน ศิริบุญรอด KMITL 01076249 Data Structures & Algorithms : Sorting [home](#) 28

## Quick Sort

Key idea :

1. Choose the pivot

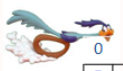
- 1<sup>st</sup> element
- median of 3
- average

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



|   |   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 5 | 1 | 4 | 9 | 6 | 3 | 8 | 2 | 7 | 0 |
| 3 | 1 | 4 | 0 | 2 | 5 | 8 | 6 | 7 | 9 |
| 3 | 1 | 4 | 0 | 2 | 5 | 8 | 6 | 7 | 9 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

Ordered List

Quick Sort Partition : 1<sup>st</sup> element

//sort a[left,right]

QuickSort(a, left, right)

if a's size &gt; 1

pivot = first element

i = index of second element

j = index of last element;

loop (i &lt; j)

right scan i until element &gt; pivot

left scan j until element &lt; pivot

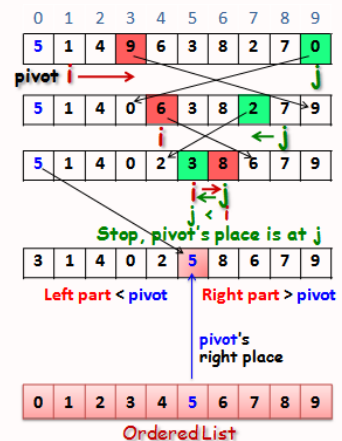
if (i &lt; j) swap elements at i and j

//if i &gt; j means all list is scanned

swap pivot to right pos at j

QuickSort left half a[left,j-1]

QuickSort right half a[j+1,right]



ทศ.ดร.บุญธีร์ เกียรติธราวุธ ทศ.ภกตวัน ศิริบุญรอด KMITL 01076249 Data Structures & Algorithms : Sorting [home](#) 33

ทศ.ดร.บุญธีร์ เกียรติธราวุธ ทศ.ภกตวัน ศิริบุญรอด KMITL 01076249 Data Structures & Algorithms : Sorting [home](#) 34