



*Descending Order*

# Sorting

Kiatnarong Tongprasert

## Sorts

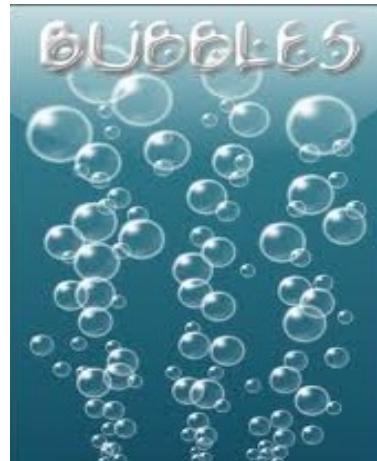
- Bubble Sort
- Selection Sort
- Insertion Sort
- Shell Sort
- Heap Sort
- Merge Sort
- Quick Sort

# Sorting การจัดลำดับ



# Bubble Sort

Ascending Order จากน้อย --> มาก



Key idea :

Bubble ตัวใหญ่สุด,

- Scan จากซ้าย , สลับที่คู่ที่ไม่ถูกลำดับ  
แต่ละ pass (scan)
  - ตัวใหญ่สุดจะลอยขึ้นไปทางขวาไปยังที่มั่นควรจะอยู่
  - ทำให้จำนวนข้อมูลลดลง 1 ตำแหน่ง
- Repeat bubbling

ต่อมาก็ตัวใหญ่ลำดับต่อมา

Original File
8 3 9 4 5
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

From 1 <sup>st</sup> pass
3 8 4 5 9
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

From 2 <sup>nd</sup> pass

After 3<sup>rd</sup> pass: 3<sup>rd</sup> biggest 5 floats up

# Bubble Sort

## Questions:

1. ทำทั้งหมดกี่ pass  $n-1$

2. เปรียบเทียบทั้งหมดกี่ครั้ง :

- pass #1  $n-1$

- pass #2  $n-2$

- ...

- Pass #  $n-1$  1

- pass ที่  $i$  เปรียบเทียบ ( $n-i$ ) comparisons.

- รวม :  $(n-1) + (n-2) + \dots + 1$

- $= O(\underline{n^2})$

3. Pass #4.

- ต้องมี pass #4 ? ไม่

- ทำไม? ข้อมูลเรียงแล้ว

- รู้ได้อย่างไร? ไม่มีการสลับที่ใน pass ที่แล้ว

Ascending Order จากน้อย --> มาก

Original File					
8	3	9	4	5	
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

From 1 <sup>st</sup> pass					
3	8	4	5	9	
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

From 2 <sup>nd</sup> pass					
3	4	5	8	9	
					After 3 <sup>rd</sup> pass: 3 <sup>rd</sup> biggest 5 floats up

## Bubble Sort Pseudocode

```

last_index = n-1 //array size = n
swaped = true
loop(last_index >= 1 && swaped)
    swaped = false
    i = 0
    loop(i < last_index)
        if (a[i] > a[i+1])
            swap(a[i],a[i+1])
            swaped = true
        i += 1
    last_index -= 1

```

Inner loop					
	0	1	2	3	4
p0	8	3	9	4	5
P1	3	8	9	4	5
P2	3	8	9	4	5
P3	3	8	4	9	5
P4	3	8	4	5	9

Outer loop					
	0	1	2	3	4
p0	8	3	9	4	5
P1	3	8	4	5	9
P2	3	4	5	8	9
P3	3	4	5	8	9
P4	3	4	5	8	9

$$1+2+3+\dots+(n-3)+(n-2)+(n-1)$$

$n * (n-1)/2$

array size = n  
 Outer loop ทำตั้งแต่ last [n-1,1]  
 inner loop : i [0 , last-1] = last ครั้ง

$$\sum_{last=n-1}^1 last = \sum_{i=n-1}^1 i = \sum_{i=1}^{n-1} i$$

$$= 1+2+3+\dots+(n-1) = n * (n-1)/2 = O(n^2)$$

## Bubble Sort Code : Python

```
def bubble(l):  
    for last in range(len(l)-1, 0, -1):#from last index to 0  
        swaped = False  
        for i in range(last):  
            if l[i] > l[i+1]:  
                l[i], l[i+1] = l[i+1], l[i] #swap is true  
                swaped = True  
        if not swaped:  
            break  
  
l = [5,6,2,3,0,1,4]  
bubble(l)
```

Outer loop					
P0	8	3	9	4	5
P1	3	8	4	5	9
P2	3	4	5	8	9
P3	3	4	5	8	9
P4	3	4	5	8	9

Inner loop

8	3	9	4	5
3	8	9	4	5
3	8	9	4	5
3	8	4	9	5
3	8	4	5	9

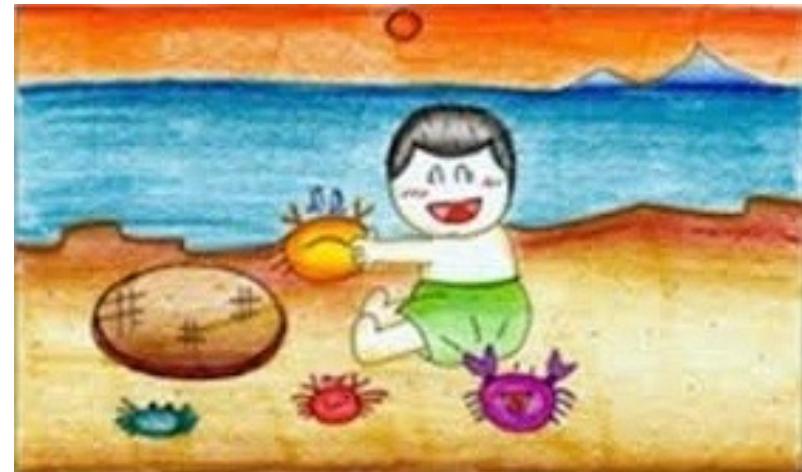
0    1    2    3    4

array size = n  
Outer loop ทำตั้งแต่ last [n-1,1]  
inner loop : i [0 , last-1] = last ครั้ง

$$\sum_{last=n-1}^1 \sum_{i=n-1}^1 i = \sum_{i=1}^{n-1} i = 1 + 2 + 3 + \dots + (n-1) = n * (n-1)/2 = O(n^2)$$

# Straight Selection Sort (Pushed Down Sort)

6	9	8	5	4	Original data : 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 : After Pass <u>4</u>



I'll choose the biggests first.

Ascending Order เรียงจากน้อย --> มาก

Algorithm :

- Scan เพื่อ เลือก ตัวใหญ่สุด(หรือตัวเล็กสุด) สลับกับ ตัวสุดท้าย (หรือตัวแรก)  
ในแต่ละครั้งที่ scan
  - ตัวใหญ่สุด(หรือตัวเล็กสุด) ไปอยู่ที่ตำแหน่งท้ายของกลุ่ม และ
  - file สั้นลง 1 ตัว
- ทำ 1 ซ้ำ

ตัวอย่าง pass แรก เลือกตัวใหญ่สุด ได้ 9 สลับกับตำแหน่งสุดท้ายคือ 4 ดังนั้นจะได้ 9 ไปอยู่ที่ตำแหน่งท้ายของกลุ่ม pass 2 เลือกตัวใหญ่สุด ได้ 8 สลับกับตำแหน่งสุดท้ายคือ 5 ดังนั้นจะได้ 8 ไปอยู่ที่ตำแหน่งท้ายของกลุ่ม

## Straight Selection Sort Analysis



6	9	8	5	4	Original data : 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 : After Pass _____

Ascending Order เรียงจากน้อย --> มาก

Data size =  $n$ .

How many passes would make the ordered list?  $\underline{\hspace{2cm} n-1 \hspace{2cm}}$  passes.

Pass #1,                          # comparisions =  $\underline{\hspace{2cm} n-1 \hspace{2cm}}$

Pass #2,                          # comparisions =  $\underline{\hspace{2cm} n-2 \hspace{2cm}}$

Pass #3,                          # comparisions =  $\underline{\hspace{2cm} n-3 \hspace{2cm}}$

.....

...

...

Last Pass #  $\underline{\hspace{2cm} n-1 \hspace{2cm}}$                           # comparisions =  $\underline{\hspace{2cm} 1 \hspace{2cm}}$

Total #comparisions =  $\underline{\hspace{2cm} 1+2+3+\dots+(n-1) \hspace{2cm}}$  =  $\underline{\hspace{2cm} n*(n-1)/2 \hspace{2cm}}$  =  $O(\underline{\hspace{2cm} n^2 \hspace{2cm}})$

# Straight Selection Sort Pseudocode

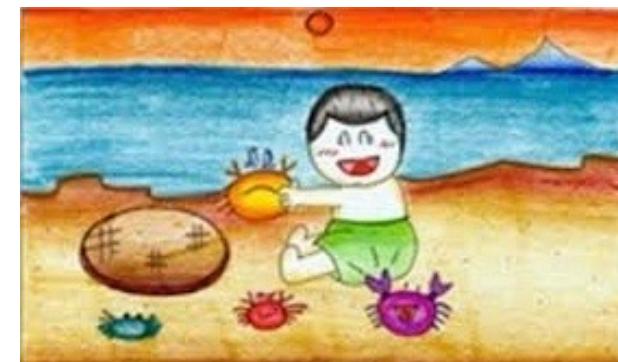
```

last = n-1 //array size = n
loop(last >= 1) //last > 0
    biggest = a[0]
    big_i = 0
    i = 1
    loop (i <= last)
        if (a[i] > biggest)
            biggest = a[i]
            big_i = i
        i += 1
    swap(a[big_i], a[last])
    last -= 1

```

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



array size = n  
Outer loop ทำตั้งแต่ last [n-1,1]  
inner loop : i [1 , last] = last ครั้ง

$$\sum_{last=n-1}^1 last = \sum_{i=n-1}^1 i = \sum_{i=1}^{n-1} i = 1+2+3+\dots+(n-1) = n * (n-1)/2 = O(n^2)$$

## Straight Selection Sort Code : Python

```
def selection(l):  
    for last in range(len(l)-1, 0, -1): #from last index to 0  
        biggest = l[0] # set biggest = first element  
        biggest_i = 0 # set index = first index  
        for i in range(1, last+1): # from 1 to last find biggest  
            if l[i] > biggest:  
                biggest = l[i]  
                biggest_i = i  
        l[last], l[biggest_i] = l[biggest_i], l[last] # swap biggest at the last
```

6	9	8	5	4
6	4	8	5	9
6	4	5	8	9
5	4	6	8	9
4	5	6	8	9

Ascending Order  
เรียงจากน้อย --> มาก

array size = n  
Outer loop from last [n-1,1]  
inner loop : i [1 , last] = last ครั้ง

$$\sum_{last=n-1}^1 last = \sum_{i=n-1}^1 i = \sum_{i=1}^{n-1} i = 1+2+3+\dots+(n-1) = n * (n-1)/2 = O(n^2)$$

# Insertion Sort



Algorithm : ให้นำก์เมื่อนหยิบไพ่ขึ้นมาทีละตัว เอาใส่ในมือที่เป็นไปที่เรียงไว้แล้ว ในตัวอย่างไฟน์มือเป็นสีส้ม

1. Scan เพื่อเลือกใส่ (insert) ตัวใหม่เข้าที่ในข้อมูลที่เรียงแล้ว insert โดยเทียบไปทีละตัว (ในตัวอย่างเทียบจากขวาไปซ้าย)  
หากตัวในมือมากกว่าให้เลื่อนมันออกมากทางขวา 1 ตำแหน่ง
2. ทำ 1 ซ้ำ จนใส่ไฟหมด

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 --> มาก

ตัวอย่าง ครั้งแรกที่หยิบไฟใบแรก ไม่ต้องทำอะไร เพราะมีเพียง 1 ใบ

1. pass ที่ 1 เอาไฟใบที่ 2 คือ 6 เข้าไปในมือ (หากให้ 6 อญ)  $6 < 8$  เลื่อน 8 ออกมา เทียบสุดแล้ว ใส่ 6 ในที่เดิมของ 8
2. pass ที่ 2 เอาไฟใบที่ 3 คือ 7 เข้าไปในมือ (หากให้ 7 อญ)  $8 < 7$  เลื่อน 8 ออกมา 6 ไม่น้อยกว่า 7 พับที่สำหรับ 7 ใส่ 7 ในที่เดิมของ 8 สุดแล้ว ใส่ 5 ในที่เดิมของ 6
3. pass ที่ 3 เอาไฟใบที่ 4 คือ 5 เข้าไปในมือ (หากให้ 5 อญ)  $8 < 5$  เลื่อน 8 ออกมา  $7 < 5$  เลื่อน 7 ออกมา  $6 < 5$  เลื่อน 6 ออกมา สุดแล้ว ใส่ 5 ในที่เดิมของ 6

## Insertion Sort

```
//ascending order, array size = n
```

```
// loop inserting i-th element
i = 1 // start from 2nd element
loop (i < n)
    insertEle = a[i];
    // find insert position ip
    ip = i ;
    loop (ip > 0 && a[ip-1] > insertEle)
        a[ip] = a[ip-1]; //shift out other data
        ip -= 1           // to make place for
    a[ip] = insertEle; // insertElement
    i += 1;            // for next insertElement
```



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

จำนวนอย -->มาก

array size = n

Outer loop ทำตั้งแต่ i [1,n-1]

inner loop : ip[1 , i] = i ครั้ง

$$= \sum_{i=1}^{n-1} i$$

$$= 1+2+3+\dots+(n-1)$$

$$= n * (n-1)/2$$

$$= O(n^2)$$

## Insertion Sort



8	6	7	5	9	Original data : size = n
6	8	7	5	9	After pass 1
6	7	8	5	9	After pass 2
5	6	7	8	9	After pass 3
5	6	7	8	9	Sorted : after pass 4

Data size = **n**, how many passes? **n-1**

Pass #1,                  # comparisions: minimum = **1** maximum = **1**

Pass #2,                  # comparisions: minimum = **1** maximum = **2**

Last Pass # **n-1** # comparisions: minimum = **1** maximum = **n-1**

Total #comparisions

- Worst case = **1+2+3+...+(n-1)** = **n\*(n-1)/2** = O(**n<sup>2</sup>**) When? **Reverse Ordered List**

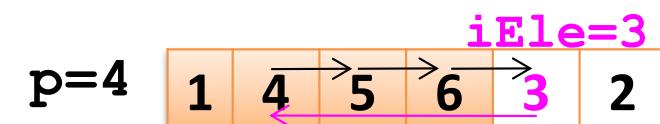
- Best case = **1+1+...+1** = **(n-1)** = O(**n**) When? **Ordered List**

## Insertion Sort Code : Python

```
def insertion(l):
    for i in range(1, len(l)): #from index 1 to last index
        iEle = l[i]      #assign insertElement
        for j in range(i, -1, -1):
            if j > 0 and l[j-1] > iEle :
                l[j] = l[j-1]
            else:
                l[j] = iEle
                break
```

```
l1 = [32,26,2,15,264,-183,10,0,20,-142,-1]
insertion(l1)
print(l1)
```

p1	4	1	6	5	3	2
p2	1	4	6	5	3	2
p3	1	4	6	5	3	2
p4	1	4	5	6	3	2
p5	1	3	4	5	6	2
p6	1	2	3	4	5	6



← j

array size = n  
Outer loop ทำตั้งแต่ i [1,n-1]  
inner loop : ip[i , 1] = i ครั้ง

$$\sum_{i=1}^{n-1} i = 1+2+3+\dots+(n-1) = n * (n-1)/2 = O(n^2)$$

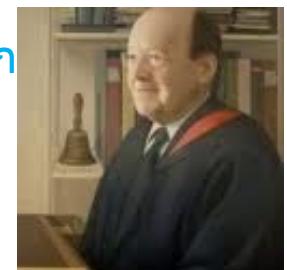
# Shell Sort (Diminishing Increment Sort)

## Shell Sort

- กำหนด incremental Sequence ของ interger กี่ตัวก็ได้ ตัวแรกเป็น 1 เพิ่มจากน้อยไปมาก

$i_1, i_2, \dots, i_{max}$  เช่น **1 3 5**

- แบ่ง file ใหญ่ เป็น file ย่อย  $n$  files ตามค่า  $i$  ใน incremental Sequence  
ครั้งแรกใช้  $i_{max}$  ตัวที่มากที่สุดแบ่ง ดังนั้นถ้าใช้ **1 3 5** ต้องใช้ 5 มาทำการแบ่งก่อน

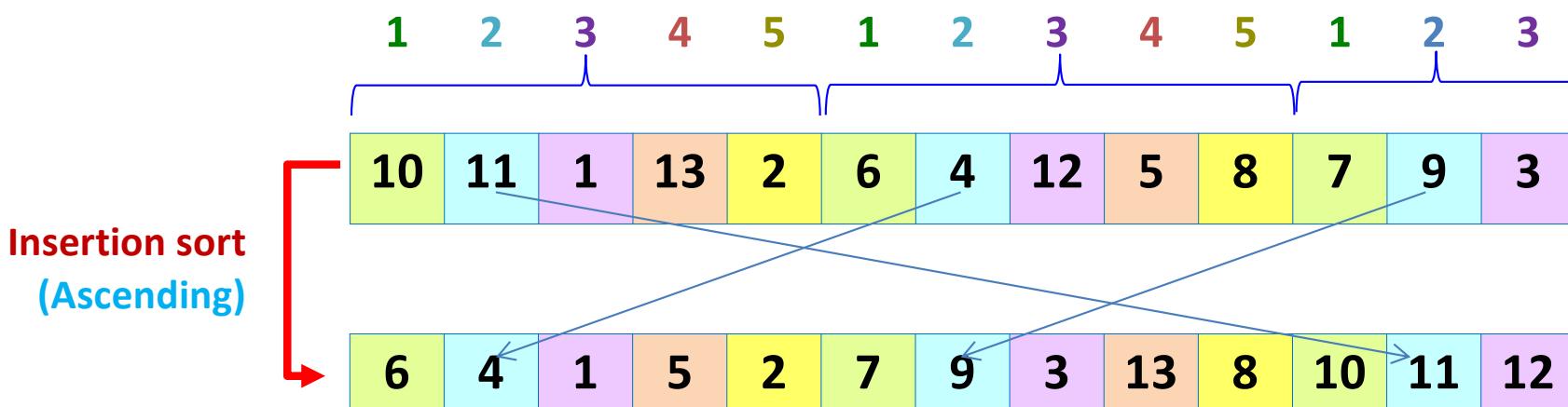


Donald Shell

- วิธีแบ่ง file ใหญ่ เป็น  $n$  file ย่อย นับข้อมูลทีละตัวไล่ไปตามลำดับตั้งแต่ 1 ถึง  $n$   
เมื่อครบ  $n$  ให้เริ่มนับ 1 ถึง  $n$  ใหม่ ทำเช่นนี้ไปเรื่อยๆ จะได้ข้อมูล  $n$  กลุ่ม ตามหมายเลขที่นับ  
ใช้ 5 แบ่งได้ 5 files ย่อย ตามสี ดังแสดงข้างล่าง

- Insertion sort แต่ละ file ย่อยทุก file ได้ผลลัพธ์อยู่ในตำแหน่งของแต่ละ file ย่อย  
จะเห็นว่าในครั้งแรก data วิ่งไกล

- ทำขั้นตอน 2-4 ใหม่โดยใช้ค่า  $i$  ตัวมากrongลงมา เมื่อทำการแบ่ง file โดยใช้  $i = 1$  ในที่สุดก็  
จะทำ insertion sort กับข้อมูลทุกตัวพร้อมกัน จึงเรียก Diminishing Increment Sort เพราะ  
ใช้ incremental sequence ตัวที่ลดลงเรื่อยๆ



# Shell Sort (Diminishing Increment Sort)

## Shell Sort

1. กำหนด incremental Sequence  $i_1, i_2, \dots, i_{\max}$  เช่น

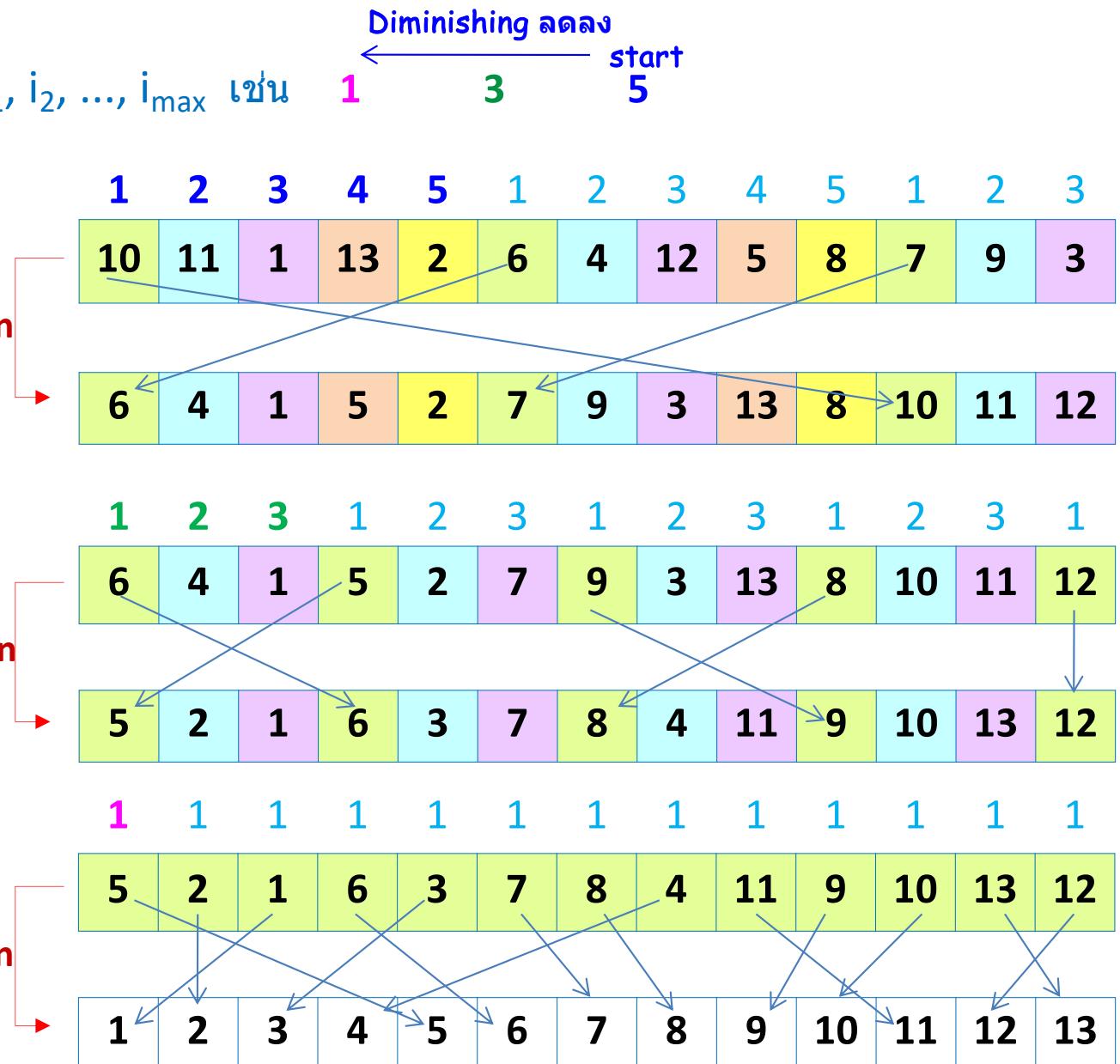
2. for  $n$  in range( $\max, 0, -1$ )
  - แบ่ง file ใหญ่ เป็น  $n$  file ย่อย
  - insertion sort แต่ละ file ย่อย

$n$  มาก  $\rightarrow$  file เล็ก  
insertion sort  $O(n^2)$  ok  
 $n$  น้อย  $\rightarrow$  ค่อนค่าง sorted  
insertion sort ดี  $O(n)$

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

### Insertion Sort

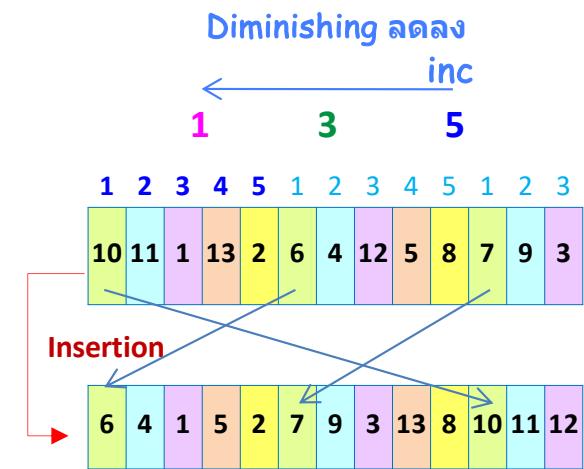
Best Case :  
ordered list  $O(n)$



[home](#)

## Shell Sort (Donald Shell)

```
def shell(l, dIncrements):  
    for inc in dIncrements: #for each diminishing increment  
        for i in range(inc, len(l)): #insertion sort  
            iEle = l[i] #inserting element  
            for j in range(i, -1, -inc):  
                if l[j-inc] > iEle and j >= inc:  
                    l[j] = l[j-inc]  
                else:  
                    l[j] = iEle  
                    break  
  
l = [10,11,1,13,2,6,4,12,5,8,7,9,3]  
dIncrements = [5,3,1]  
shell(l, dIncrements)  
print(l)
```



Shell :  $1, 2, 4, 8, 16, \dots, 2^i$

$O(n^2)$

Hibbard :  $1, 3, 7, 15, \dots, 2^{i-1}$

$O(n^{3/2})$

Sedgewick  $1, 8, 23, 77, 281, \dots, 4^{i+1} + 3 * 2^i + 1$

$O(n^{4/3})$



Donald Shell



Binary Heap

Complete Binary Tree

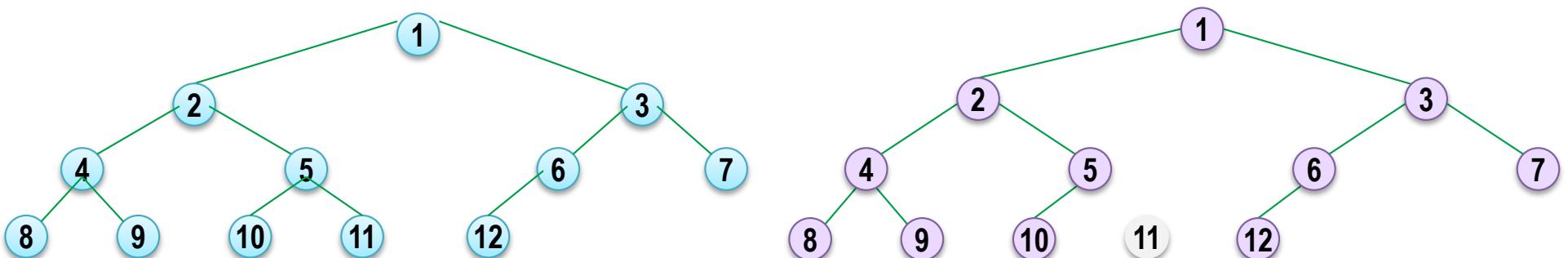
# Review : Complete Binary Tree

## Complete Binary Tree

$$H = \log_2(N+1) - 1$$

$$= \lfloor \log_2 N \rfloor \rightarrow O(\log_2(N))$$

$N$  ระหว่าง  $[2^H, 2^{H+1} - 1]$

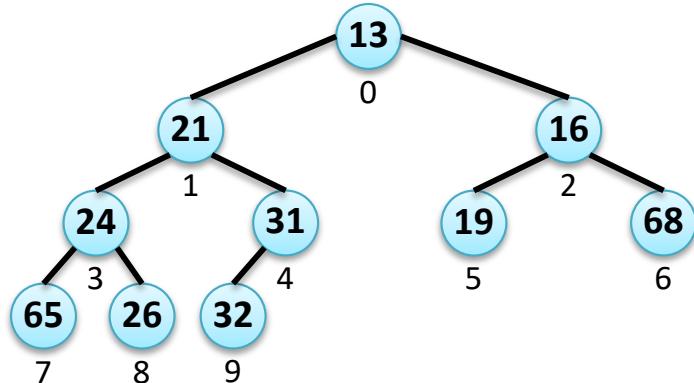


## Complete Binary Tree

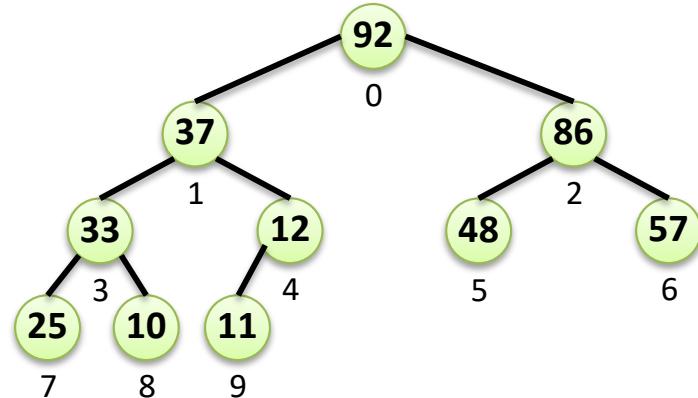
Every level is completely filled,  
except possibly the last, &  
All nodes are as far left as possible

Without node 11 :  
Not a complete binary tree

# Binary Heap



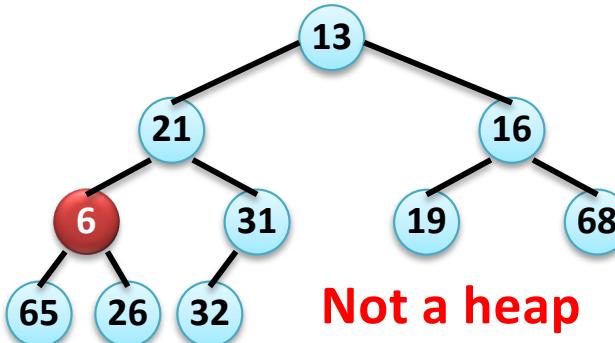
Ascending heap  
(Min heap)



Descending heap  
(Max heap)

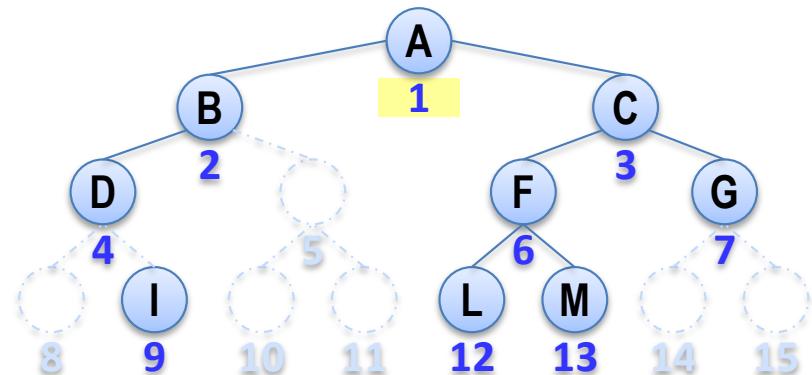
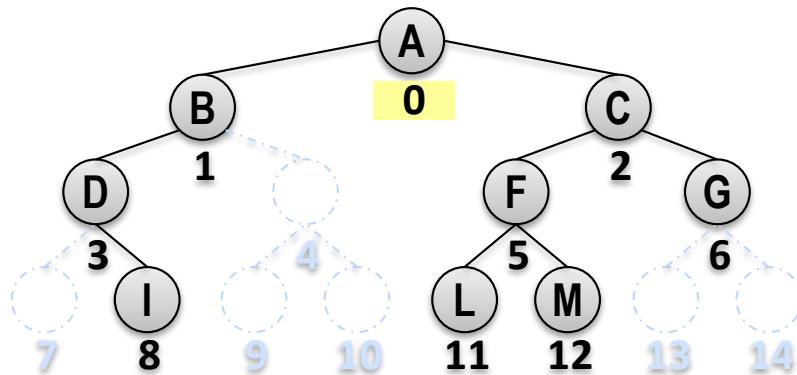
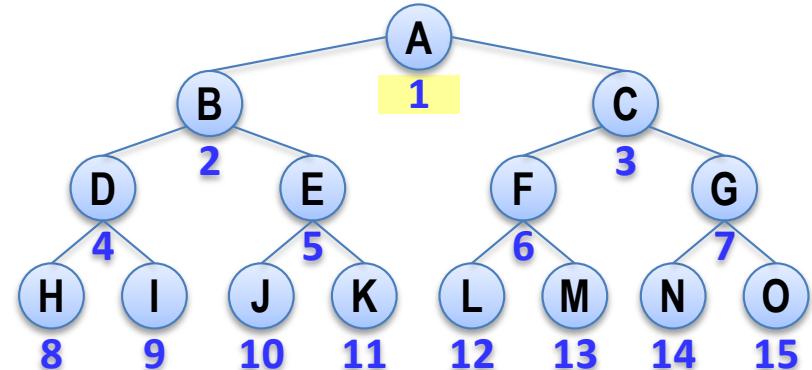
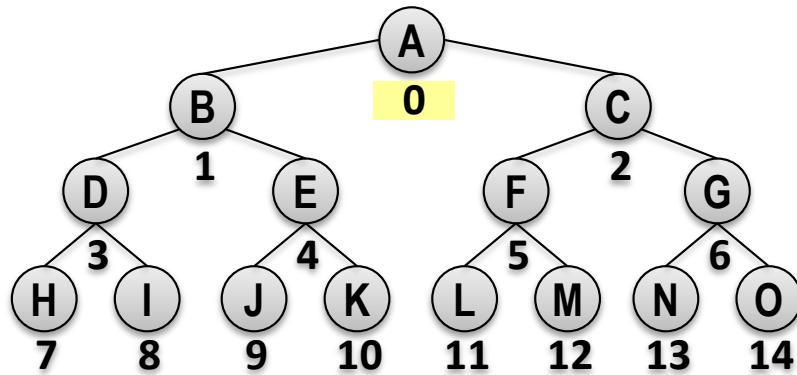
Binary Heap : complete Binary Tree ชี้ไป key ของ node ได้

1. key  $\leq$  ของ descendants ของ มัน : **Min heap** เช่น 13 น้อยกว่าลูกหลานทั้งหมดของมัน
2. key  $\geq$  ของ descendants ของ มัน : **Max heap** เช่น 92 มากกว่าลูกหลานทั้งหมดของมัน

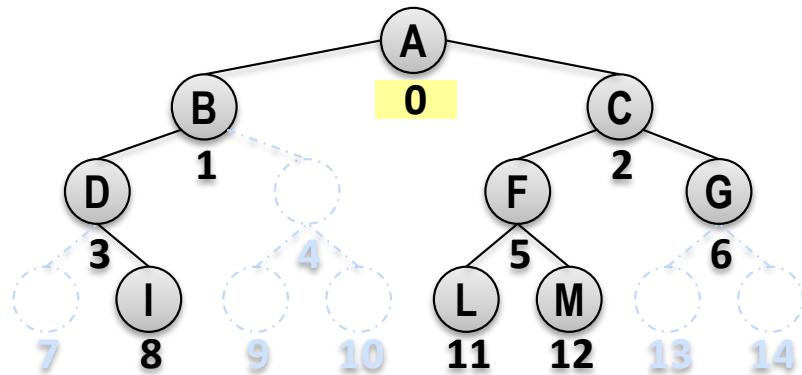


Not a heap

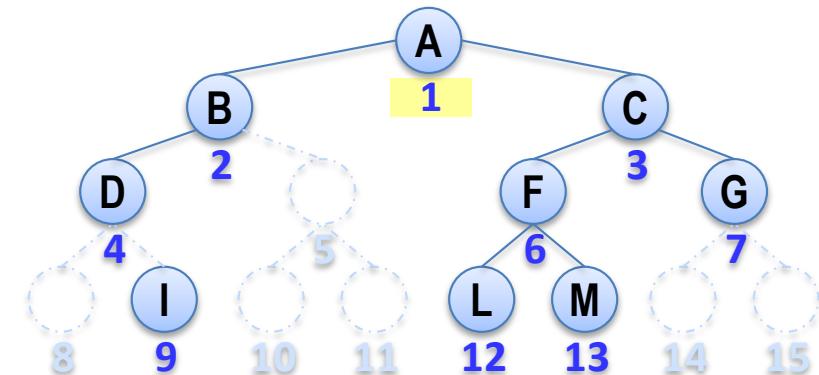
# Review : (Sequential) Implicit Array



# (Sequential) Implicit Array



A	B	C	D		F	G		I		L	M			
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14



	A	B	C	D		F	G	I		L	M			
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Where is the **root** ?

The node at index $i$ 's	left son ?
	right son ?
	father ?

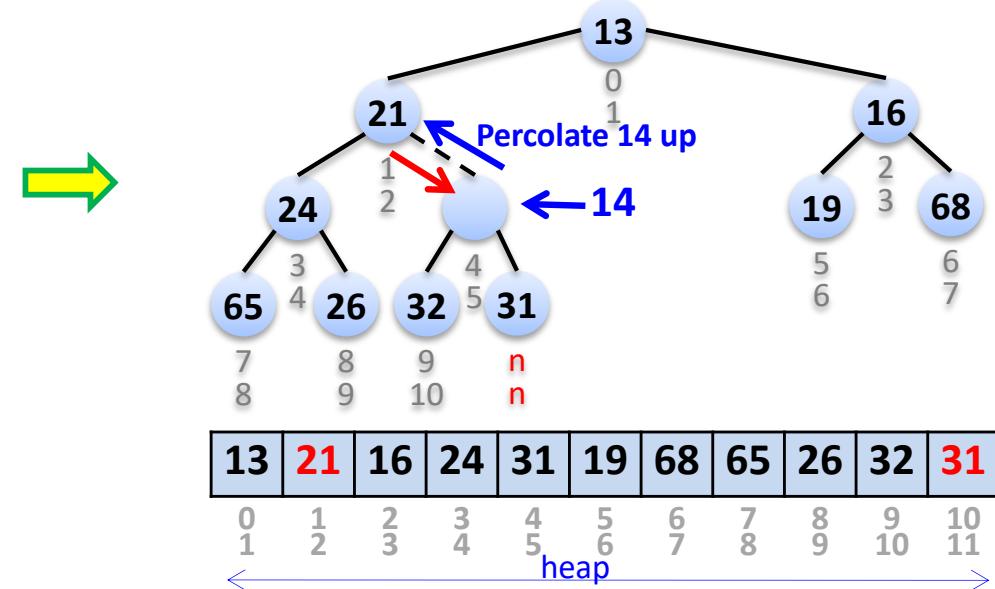
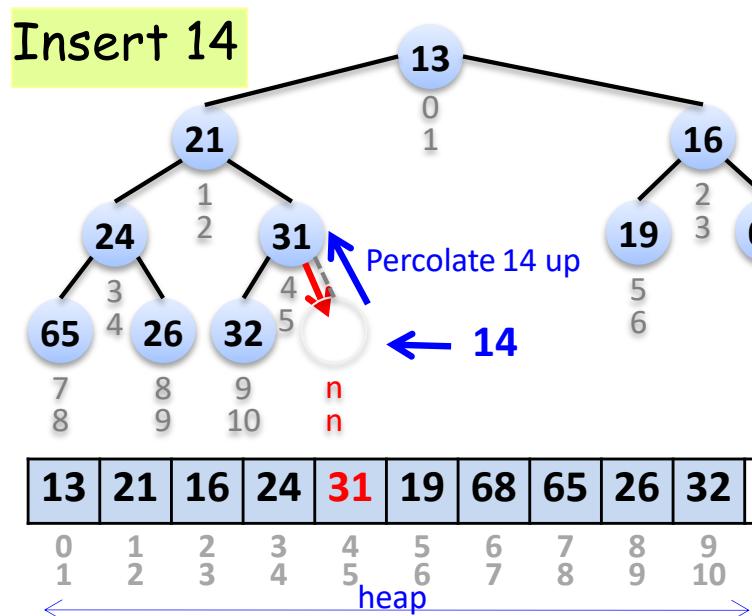
Start at index 0

0
$2i + 1$
$2i + 2$
$(i - 1) \text{ div } 2$

Start at index 1

1
$2i$
$2i + 1$
$i \text{ div } 2$

# Insert Heap

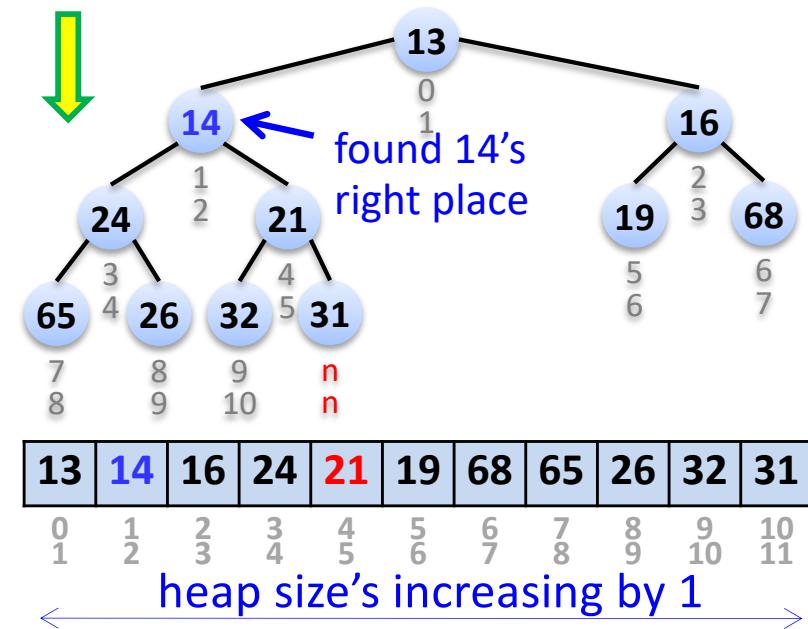


Inserting 14.

- ก่อนจะ insert : last index = n-1 (data 32)
- insert จะเพิ่มขนาด heap ขึ้น 1 ที่ตำแหน่ง n
- หาที่ให้ 14 เริ่มลงที่ n
- ถ้าไม่ได้ percolate 14 up ( $14 < 31$ ) ( $14 <$ พ่อ),  
เลื่อนพ่อของมัน (31) down
- Repeat หาที่ให้ 14 ตาม path ไปยัง root  
จนกว่าจะพบที่ของ 14

After Insertion

- ขนาด heap เพิ่มขึ้น 1
- การปรับ tree ให้เป็น heap อีกครั้งเรียกว่า reheap



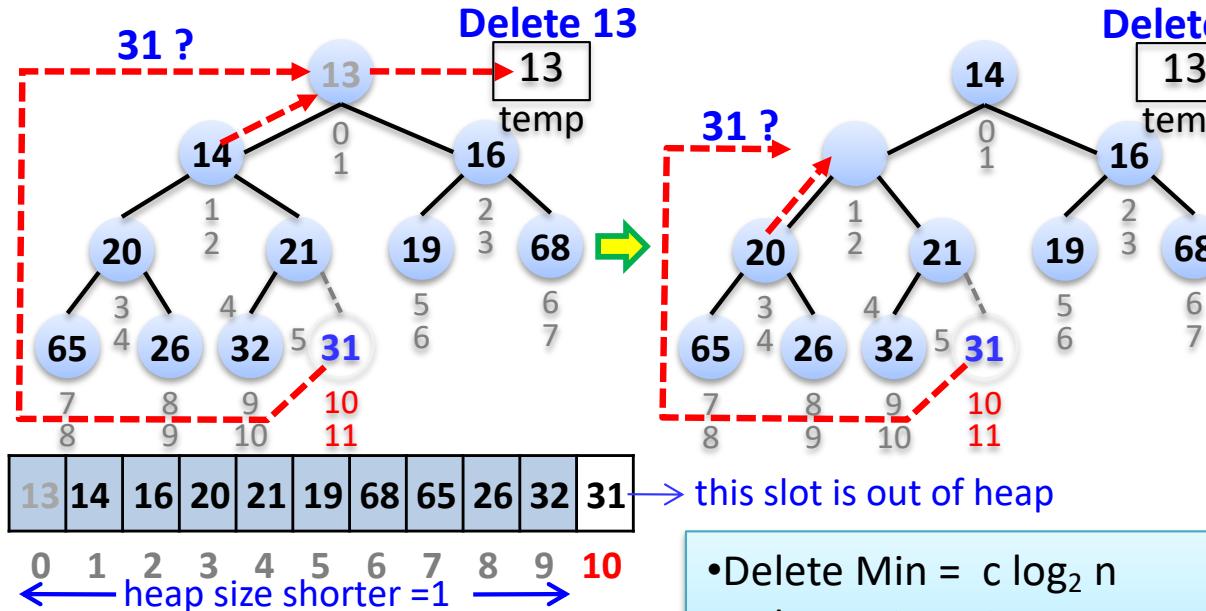
## Insert()

```
from math import log2
from math import floor
def print90(h, i, max_i):
    if i < max_i:
        indent = floor(log2(i+1))
        print90(h, (i*2)+2, max_i)
        print('    ' * indent, h[i])
        print90(h, (i*2)+1, max_i)

def insertMinHeap(h, i):
    """insertMinHeap"""
    print('insert', h[i], 'at index', i, end = '      ')
    print(h)
    insertEle = h[i]
    fi = (i-1)//2    #
    while i > 0  and insertEle < h[fi] :
        h[i] = h[fi]
        i = fi
        fi = (i-1)//2
    h[i] = insertEle

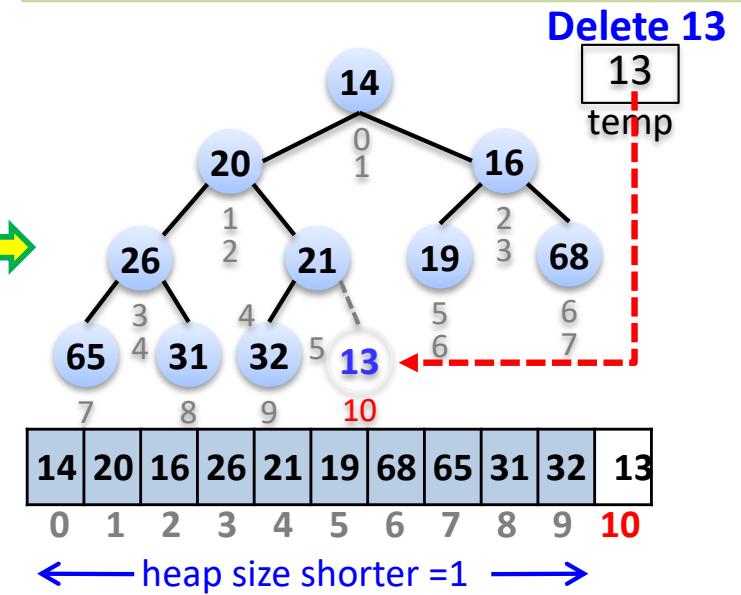
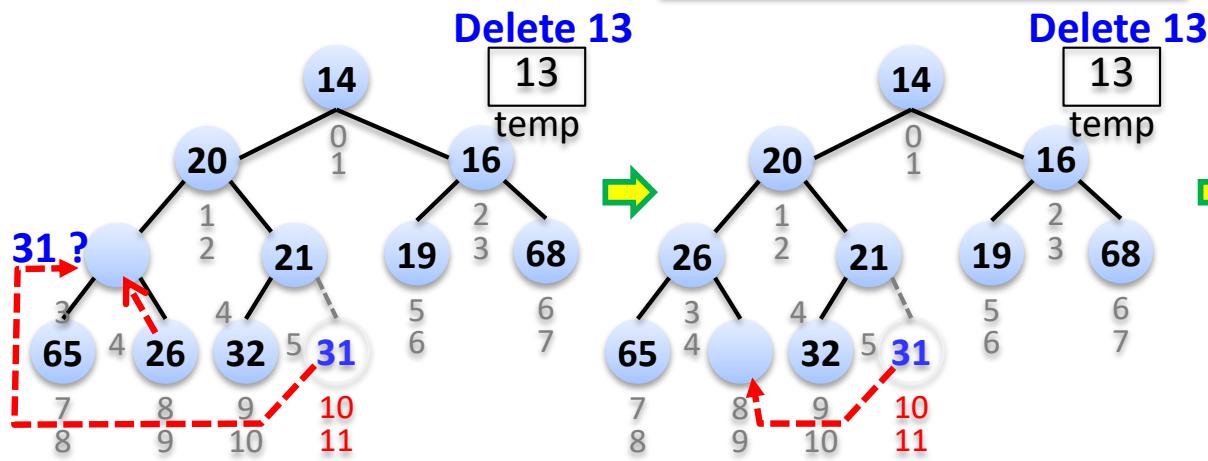
h = [30, 85, 97, 100, 200]
for i in range(1, len(h)):
    insertMinHeap(h, i)
    print(h)
    print90(h, 0, i)
    print('-----\n')
```

# Delete Min (ie. Delete root) : to sort data



- Start at min heap from index : 0 to 10
- Delete min (root 13) -> hole at root.
- To delete, heap size shorter 1 (last node)
- Last node data, 31, must be in the heap.
- Find place for 31(reheap). Try the hole.
- if not, perlocate 31 down, take the hole's smaller son up.
- Repeat finding place for 31.
- Put the deleting data 13 at previous-31-place in the same array is called **in place sort**, makes descending order.
- Now 13 is out of heap.

- Delete Min =  $c \log_2 n$
- Delete min  $n$  times.
- Heap sort ->  $O(n \log_2 n)$

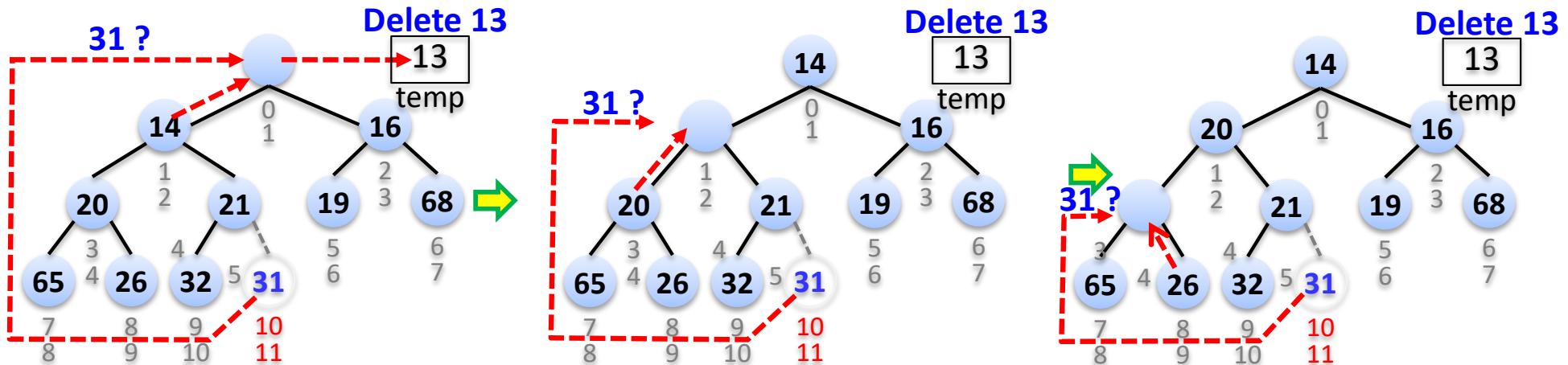


## DeleteMin()

```
def delMin(h, last):
    print('delMin', h[0], 'last index = ', last, end = '      ')
    print(h)
    insertEle = h[last]
    h[last] = h[0] #inplace sort the root
    hole = 0
    ls = hole*2+1 # left son indices
    found = False
    while ls < last and not found:
        rs = ls if ls+1 >= last else ls+1
        min = ls if h[ls] < h[rs] else rs # minson index
        if h[min] < insertEle:
            h[hole] = h[min] # promote small son up to hole
            hole = min # going down the tree
            ls = hole*2+1
        else:
            found = True
    h[hole] = insertEle

h = [13,14,16,24,21,19,68,65,26,32,31]
for last in range(len(h)-1, -1, -1):
    delMin(h, last)
    print(h)
    print90(h, 0, last)
    print('-----\n')
```

# Heap Sort Analysis

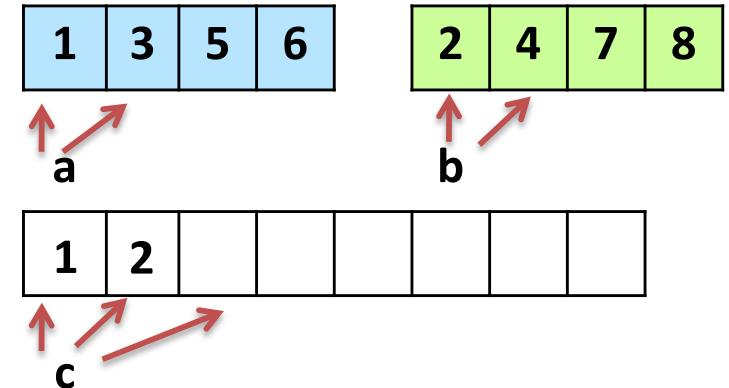
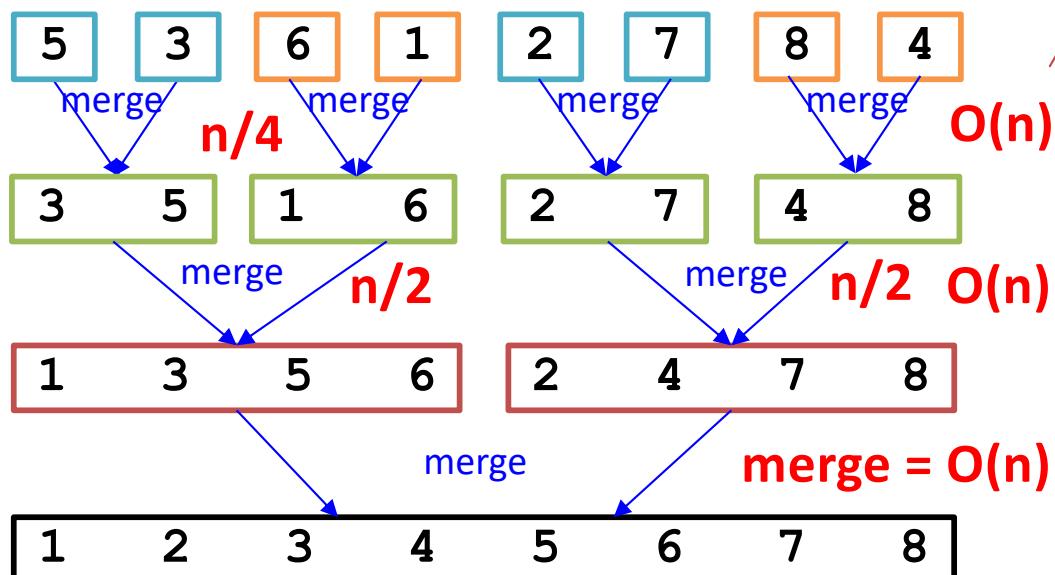


- ไม่ดีเมื่อ  $n$  น้อย (ต้องเสียเวลาในการปรับให้เป็น heap)
- Maximum delete min 1 ตัว =  $O(\log_2 n)$  เพราะ depth ของ heap เป็น  $\log_2 n$   
ดังนั้น Heap Sort = deleteMin  $n-1$  ครั้ง =  $O(n \log_2 n)$
- ใช้ space น้อยมาก แต่ไม่มี extra space เลย หากทำ inplace sort
- จากการศึกษาพบว่าโดยทั่วไป heap sort กินเวลาประมาณ 2 เท่าของ quick sort
- ในทางปฏิบัติพบว่าช้ากว่า Shell Sort ที่ใช้ incremental sequence ของ Sedgewick

# Merge Sort

Key idea **Merge** : sorts 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



$8=2^3=2^d$    #comparisons  $\leq \underline{n}$   
#assignments =  $\underline{n}$   
Merge =  $O(\underline{n})$

$$4=2^2$$

$$2=2^1$$

$$n=2^d$$

$$d=\log_2 n$$

$$1=2^0$$

Merge Sort = merge  $O(n) \times d = O(n \log_2 n)$

[home](#)

# Merge Sort Pseudocode

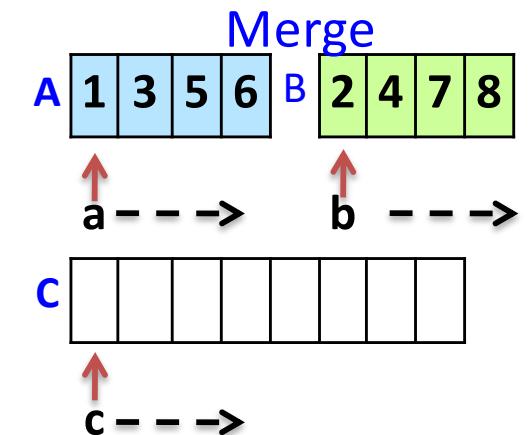
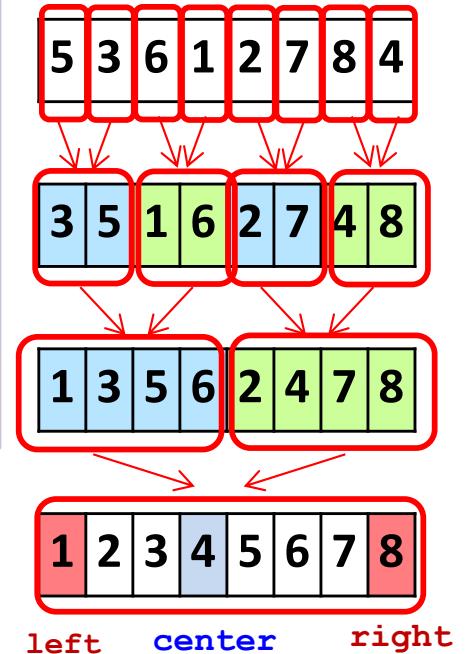
```
//mergeSort : sort a from left to right
mergeSort(a, left, right)
    if (left < right)
        center = (left + right) / 2
        mergeSort(a, left, center)
        mergeSort(a, center+1, right )
        merge(a, left, center, center+1, right)
```

$$\begin{aligned} T(1) &= 1 \\ T(N) &= 2 * T(N/2) + N \\ &\rightarrow \underline{T(N/2)} \\ &\rightarrow \underline{T(N/2)} \\ &\rightarrow \underline{N} \end{aligned}$$

```
//merging ordered a [iA,endA] & b [iB,endB]
//to be ordered using temp_list C [iA,endB]
```

```
Merge(a, iA, endA, iB, endB)
    c = C[iA]
    loop(iA < endA and iB < endB)
        if ((a[iA] < a[iB])
            C[c] = a[iA], iA++
        else C[c] = a[iB], iB++
        c++
    append C with the remaining list
    copy C back to a
```

$$\begin{aligned} \# \text{comparisons} &= \underline{<N} \\ \# \text{assignments} &= \underline{N} \\ O(\underline{N}) \end{aligned}$$



# Recurrence Relation (Recurrence Equation)

```
//mergeSort : sort a from left to right
mergeSort(a, left, right)           T(1) = 1
    if (left < right)               T(N) = 2 * T(N/2) + N
        center = (left + right) / 2
        mergeSort(a, left, center)   → T(N/2)
        mergeSort(a, center+1, right ) → T(N/2)
        merge(a, left, center, center+1, right) → N
```

$$T(n) = 2 * T(n/2) + n$$

- ความสัมพันธ์ทางคณิตศาสตร์ข้างบน เรียกว่า recurrence relation หรือ recurrence equation  
เนื่องจากมี function  $T()$  อยู่ทั้งด้านซ้ายและขวาของสมการ
- ที่เป็นดังนี้ เพราะ merge sort เป็น recursive algorithm  
เราสามารถใช้ recurrence relation หา complexity ของ algorithm ได้
- สำหรับกรณี merge sort =  $O(n \log_2 n)$  ซึ่งจะได้เห็นใน slide ถัดๆไป

# Merge Sort Analysis : Telescoping a sum

$$\begin{aligned} T(1) &= 1 && \dots (1) \\ T(N) &= 2*T(N/2) + N && \dots (2) \end{aligned}$$

$$\frac{T(N)}{N} = \frac{2*T(N/2)}{N} + \frac{N}{N} \quad \dots (3) : (2)/N$$

$$\frac{T(N)}{N} = \frac{T(N/2)}{N/2} + 1 \quad \dots (*) : (\text{from 3})$$

$$\frac{T(N/2)}{N/2} = \frac{T(N/4)}{N/4} + 1 \quad \dots (*) : \text{any } N=2^i$$

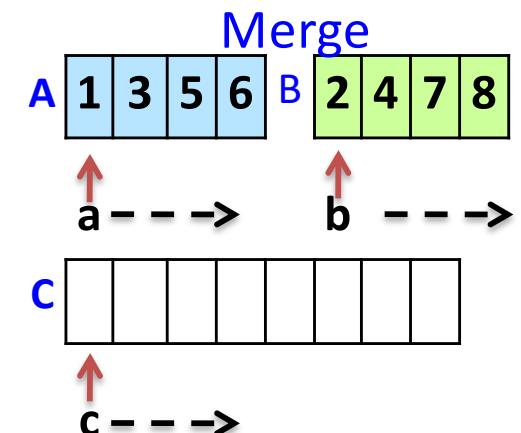
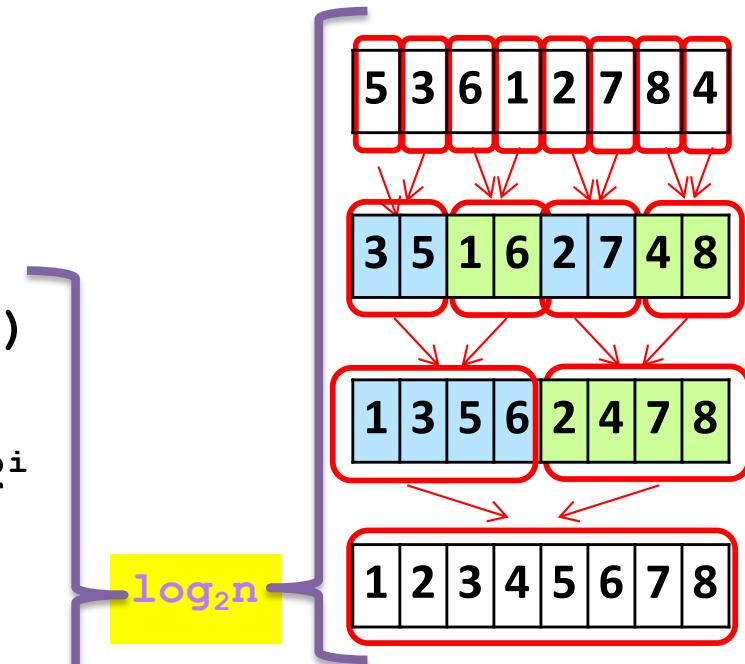
$$\frac{T(N/4)}{N/4} = \frac{T(N/8)}{N/8} + 1 \quad \dots (*) \quad ,$$

...

$$\frac{T(2)}{2} = \frac{T(1)}{1} + 1 \quad \dots (*) \quad ,$$

$$\frac{T(N)}{N} = \frac{T(1)}{1} + 1 * \log_2 N // \text{sum*} = \text{Telescoping a Sum}$$

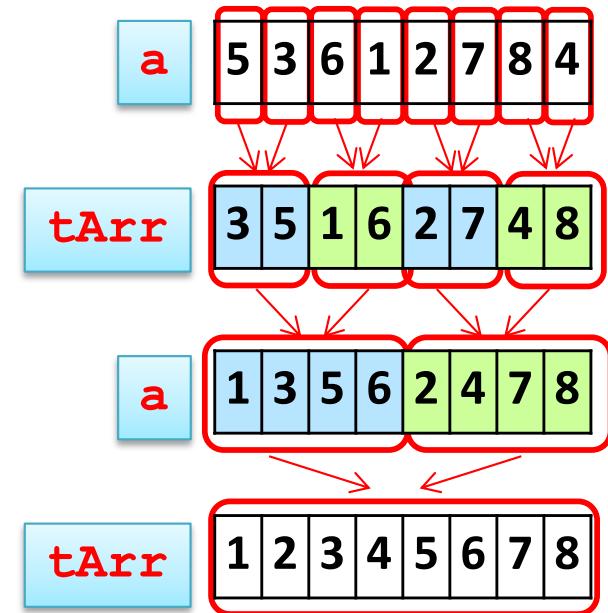
$$T(N) = N + N * \log_2 N = O(N \log N)$$



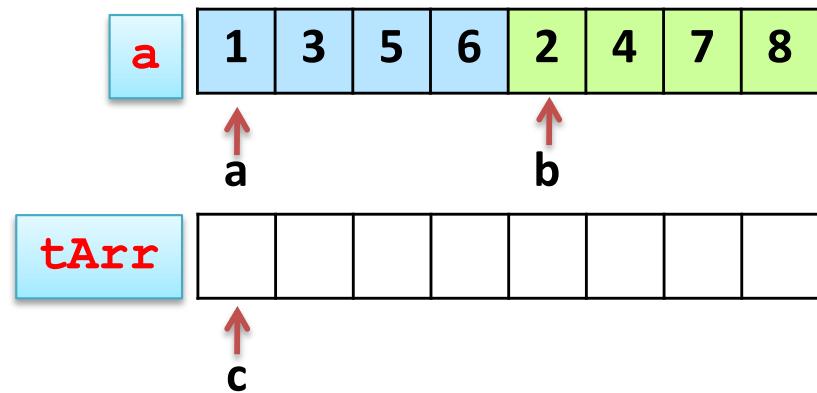
# Merge Sort Code : Python

```
def mergeSort(l, left, right):
    center = (left+right)//2
    if left < right:
        mergeSort(l, left, center)
        mergeSort(l, center+1, right)
        merge(l, left, center+1, right)

l = [5,3,6,1,2,7,8,4]
print(l)
mergeSort(l,0, len(l)-1)
print(l)
```



**merge**

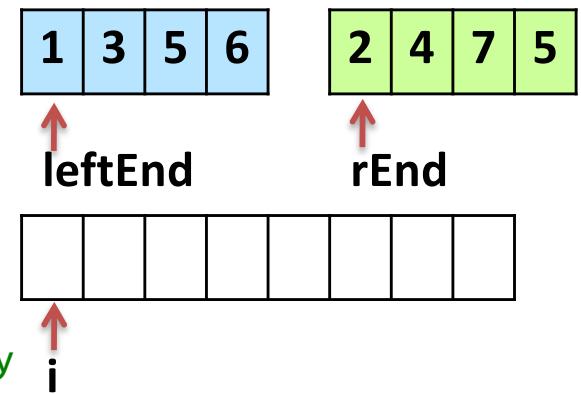


# Merge Code : Python

```
def merge(l, left, right, rightEnd):
    start = left
    leftEnd = right-1
    result = []
    while left <= leftEnd and right <= rightEnd:
        if l[left] < l[right]:
            result.append(l[left])
            left += 1
        else:
            result.append(l[right])
            right += 1
    while left <= leftEnd: # copy remaining left half if any
        result.append(l[left])
        left += 1
    while right <= rightEnd: # copy remaining right half if any
        result.append(l[right])
        right += 1

    for ele in result: # copy result back to list l
        l[start] = ele
        start += 1
        if start > rightEnd:
            break
```

Merge



# Quick Sort

Key idea :

## 1. Choose the pivot

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

## 2. Partition :

Pivot partitions data into 2 halves

- Left half < pivot
- Right half > pivot
- Pivot goes to its right place

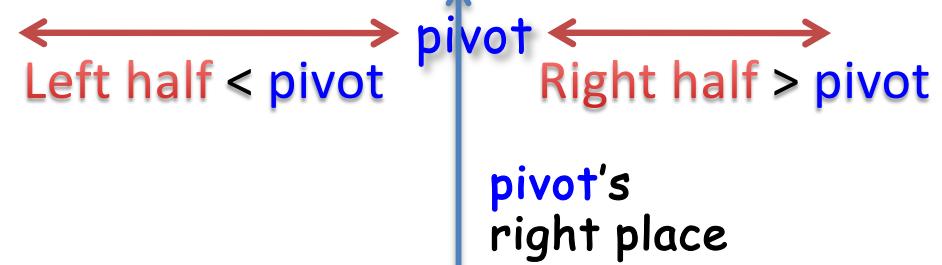
## 3. Repeat partitioning both left & right half



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

pivot

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



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

Ordered List

[home](#)

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

//sort a[left,right]

QuickSort(a, left, right)

```

if a's size > 1
pivot = first element
i = index of second element
j = index of last element;

```

```

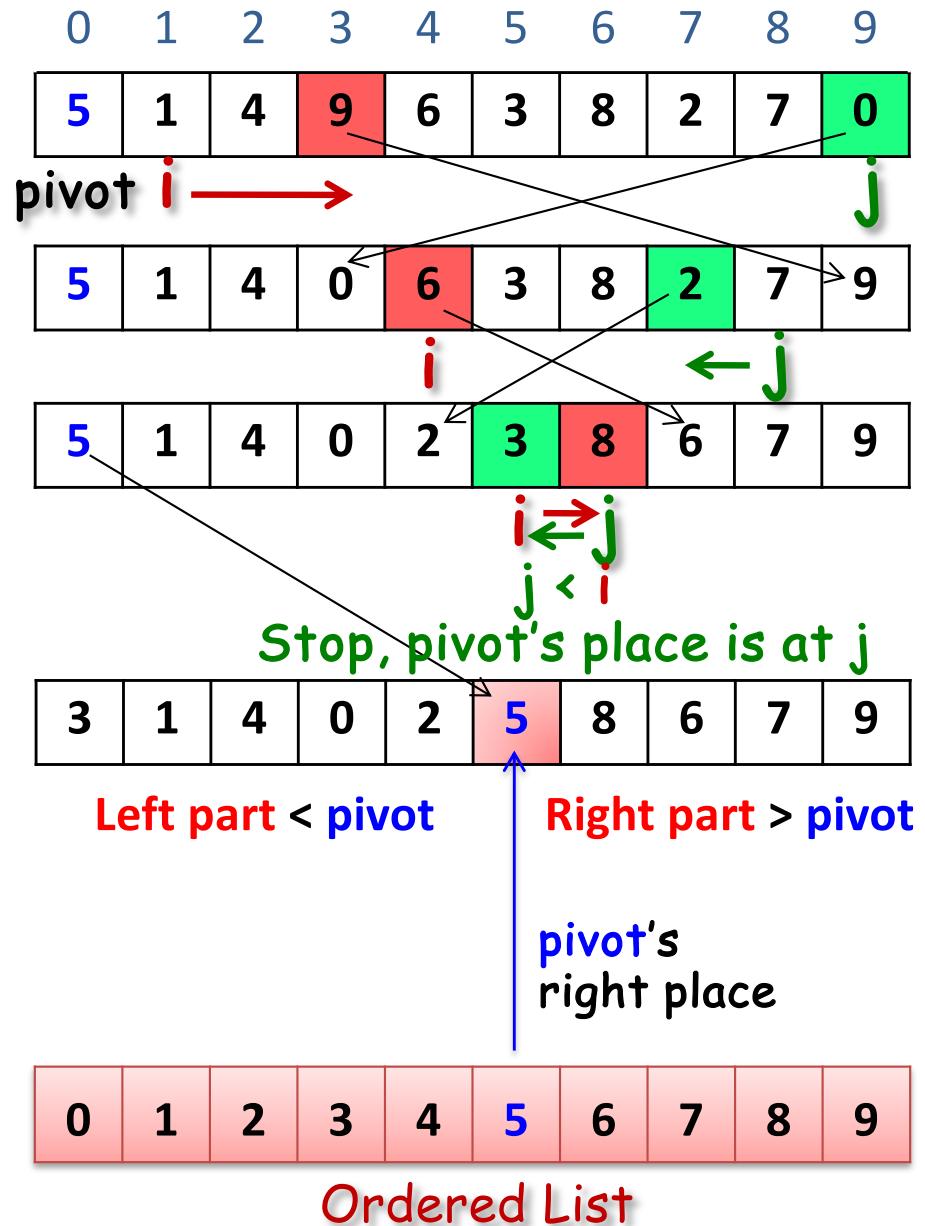
loop(i < j)
    right scan i until element > pivot
    left scan j until element < pivot
    if(i < j) swap elements at i and j

```

```

//if i > 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]

```



# Quick Sort Python Code : 1<sup>st</sup> element

```

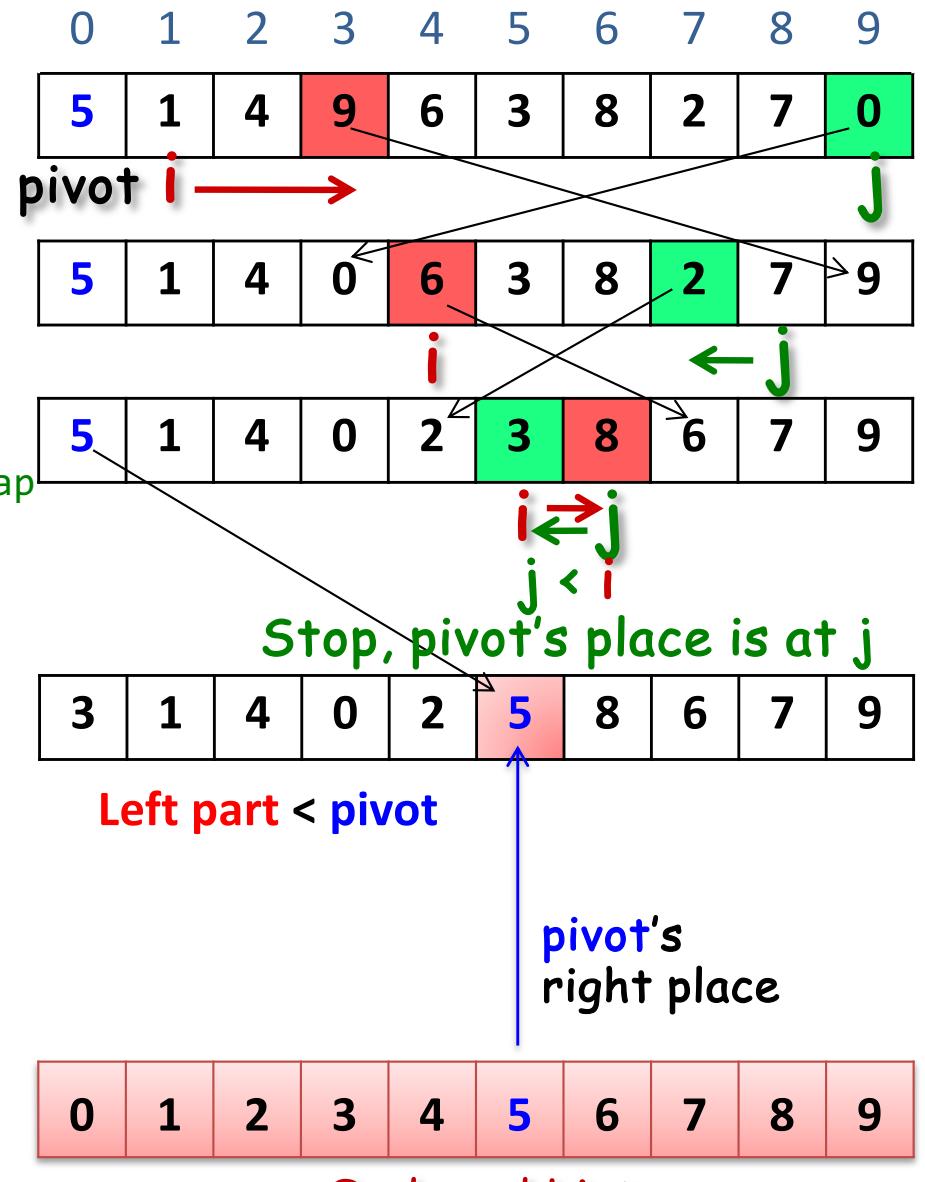
def quickSort(l) :
    qSort(l, 0, len(l)-1)

def qSort(l, left, right):
    if left < right :
        p = partition(l, left, right)
        qSort(l, left, p - 1)
        qSort(l, p + 1, right)

def partition(l, left, right):
    if left == right - 1 : #only 2 elements
        if l[left] > l[right] :
            l[left],l[right] = l[right],l[left] #swap
        return left
    pivot = l[left] #first element pivot
    i, j = left + 1, right
    while i<j:
        while i<right and l[i]<=pivot:
            i += 1
        while j>left and l[j]>=pivot:
            j -= 1
        if i<j:
            l[i], l[j] = l[j], l[i] #swap
    if left is not j:
        l[left], l[j] = l[j], l[left]
        # swap pivot to index j
    return j

l = [5,1,4,9,6,3,8,2,7,0]
quickSort(l)
print(l)

```



# Quick Sort Partition : Median of Three

left	center	right
8	6	0
0	6	8
Median = 6 = pivot		



L	0	1	2	3	C	4	5	6	7	8	R	9
	8	1	4	9	6	3	5	2	7	7		0

$C \leq L \leq R$

6	1	4	9	0	3	5	2	7	8		
i → i					j ← j						
6	1	4	2	0	3	5	9	7	8		
i → j → i											
6	1	4	2	0	3	5	9	7	8		
						j < i					
5	1	4	2	0	3	6	9	7	8		

Left part < 6

j < i Right part > 6

Stop, pivot's place is at j

pivot's right place

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

Ordered List

```
def partition(l, left, right) :
    if left == right - 1 : #only 2 elements
        if l[left] > l[right] :
            l[left],l[right] = l[right],l[left] #swap
    return left

    c = (left + right)//2 #order  #order c <= l <= r
    if l[left] < l[c] :
        l[left],l[c] = l[c],l[left]
    if l[right] < l[c] :
        l[c],l[right] = l[right],l[c]
    if l[right] < l[left] :
        l[left],l[right] = l[right],l[left]

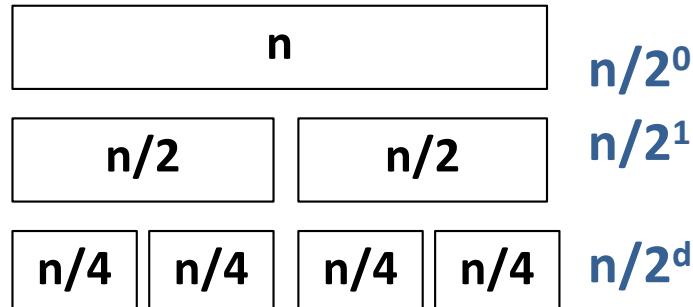
pivot = l[left] #first element pivot
i, j = left + 1, right
while i<j:
    while i<right and l[i]<=pivot:
        i += 1
    while j>left and l[j]>=pivot:
        j -= 1
    if i<j:
        l[i], l[j] = l[j], l[i] #swap
if left is not j:
    l[left], l[j] = l[j], l[left]
    # swap pivot to index j
return j
```

# Quick Sort Analysis

- แต่ละครั้งมีการ compare กับ pivot  $n-1$  ครั้ง
- Best Case pivot แบ่งครึ่งทุกครั้ง

$$T(N) = 2T(N/2) + cN$$

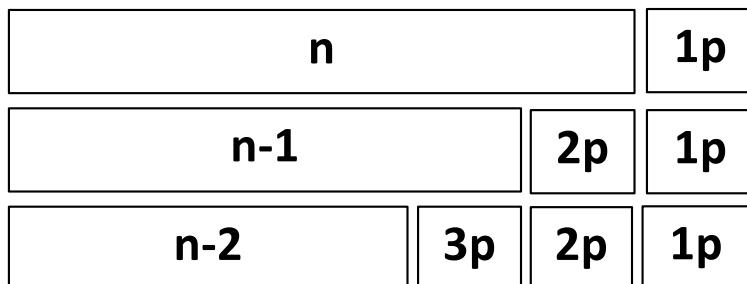
$$\therefore T(N) = O(N \log N)$$



- Worst Case pivot แยกสุดทุกครั้ง แบ่ง  $n-1$  กับ 1

$$T(N) = T(N-1) + cN$$

$$\therefore T(N) = \Theta(N^2)$$



$$T(N) = T(N-1) + cN$$

$$T(N-1) = T(N-2) + c(N-1)$$

$$T(N-2) = T(N-3) + c(N-2)$$

...

$$T(2) = T(1) + c(2)$$

$$T(N) = T(1) + c \sum_{i=2}^n i$$

$$\therefore T(N) = \Theta(N^2)$$

**Telescoping a Sum**

- Average Case  $\sim 1.386 N \log_2 N$