



Sorting

Lecturers :

Boontee Kruatrachue	Room no. 913
Kritawan Siriboon	Room no. 913

Data Structures & Algorithm in Python

Sorts

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

Linked List & Tree Sorts

- Sequential Search
(Linear Search)
- Binary Search Tree
- AVL (Hight Balanced) Tree
- B-Tree
- Heap

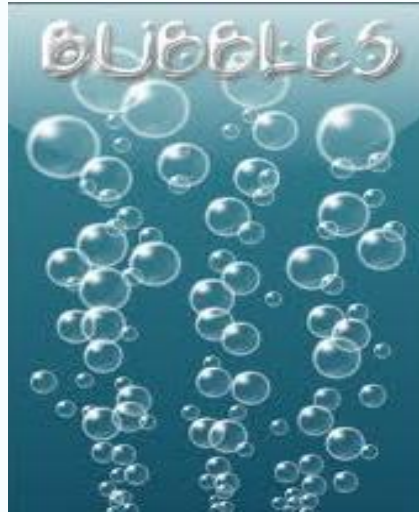
Comparison-based sorting

ใช้การเปรียบเทียบในการจัดลำดับ

- Best case $O(n \log n)$



Bubble Sort



Ascending Order จากน้อย --> มาก

Key idea :

Bubble ตัวใหญ่สุด,

- Scan จากซ้าย , สลับที่คู่ที่ไม่ถูกลำดับ

แต่ละ pass (scan)

- ตัวใหญ่สุดจะลอยขึ้นไปทางขวาไปยังที่ที่มันควรจะอยู่
- ทำให้ **file** สั่นลง 1 ตำแหน่ง
- Repeat bubbling

ต่อมาก็ตัวใหญ่ลำดับต่อมา

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 st pass: the biggest,9,floats up

3	8	4	5	9	From 1 st pass
3	8	4	5	9	
3	4	8	5	9	
3	4	5	8	9	After 2 nd pass: 2nd biggest,8, floats up

3	4	5	8	9	From 2 nd pass
					After 3 rd pass: 3 rd biggest 5 floats up

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 : $(n-1) + (n-2) + ... + 1$
 - $= O(\underline{n^2})$

3. Pass #4.

- ต้องมี pass #4 ? ไม่
- ทำไม? file เรียงแล้ว
- รู้ได้อย่างไร ? ไม่มีการสลับที่ใน pass ที่แล้ว

Ascending Order จากน้อย --> 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 st pass: the biggest,9,floats up

3	8	4	5	9	From 1 st pass
3	8	4	5	9	
3	4	8	5	9	
3	4	5	8	9	After 2 nd pass: 2nd biggest,8, floats up

3	4	5	8	9	From 2 nd pass
					After 3 rd pass: 3 rd biggest 5 floats up

Bubble Sort Pseudocode

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

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

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
	0	1	2	3	4

1+2+3+... +(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+...+(n-1) = n * (n-1)/2 = O(n²)

Bubble Sort Code : Python

```
def bubble(l):  
    for last in range(len(l)-1, -1,-1):# จาก last ind ถึง ind 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  
                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

0 1 2 3 4

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

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

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



I'll choose the biggest first.

Algorithm :

1. Scan เพื่อเลือกตัวใหญ่สุด(หรือตัวเล็กสุด) สลับที่กับตัวสุดท้าย (หรือตัวแรก)

ในแต่ละครั้งที่ scan

- ตัวใหญ่สุด(หรือตัวเล็กสุด) ไปอยู่ที่ๆ ที่ และ
- file สิ้นลง 1 ตัว

2. ทำ 1 ซ้ำ

ตัวอย่าง pass แรก เลือกตัวใหญ่สุด ได้ 9 สลับกับตำแหน่งสุดท้ายคือ 4 ดังนั้นจะได้ 9 ไปอยู่ที่ๆ

pass 2 เลือกตัวใหญ่สุด ได้ 8 สลับกับตำแหน่งสุดท้ายคือ 5 ดังนั้นจะได้ 8 ไปอยู่ที่ๆ



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

Data size = n .

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

Pass #1,	# comparisons = $\underline{n-1}$
Pass #2,	# comparisons = $\underline{n-2}$
Pass #3,	# comparisons = $\underline{n-3}$

.....

Last Pass # $\underline{n-1}$	# comparisons = $\underline{1}$
Total #comparisions = $\underline{1+2+3+...+(n-1)}$	= $\underline{n*(n-1)/2} = O(\underline{n^2})$

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++
    swap(a[big_i], a[last])
    last--
```

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, -1, -1):# จาก last ind ถึง ind 0  
        biggest = l[0] # ค่าใหญ่สุด  
        biggest_i = 0 #ตำแหน่ง ของค่าใหญ่สุด  
        for i in range(1, last+1): # จากตำแหน่ง 1 ถึง last หาค่าใหญ่สุด  
            if l[i] > biggest: # if ที่ i ใหญ่กว่าค่าเดิม  
                biggest = l[i] # เปลี่ยน ค่าใหญ่ที่สุด เป็น ค่าใหม่ที่ i  
                biggest_i = i # เก็บ index ค่าใหญ่อันใหม่  
        #swap elements biggest and last element  
        l[last], l[biggest_i] = l[biggest_i], l[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 ทำตั้งแต่ last [n-1,1]
inner loop : i [1 , last] = last ครั้ง

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



Algorithm : ให้นึกเหมือนหยิบไพ่ขึ้นมาทีละตัว เอาใส่ในมือที่เป็นไพ่ที่เรียงไว้แล้ว ในตัวอย่างไพ่ในมือเป็นสีส้ม

1. Scan เพื่อเลือกใส่(insert) ตัวใหม่เข้าที่ใน file ที่เรียงแล้ว
insert โดยเทียบไปที่ละตัว (ในตัวอย่างเทียบจากขวาไปซ้าย)
หากตัวในมือมากกว่าให้เลื่อนมันออกมาทางขวา 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 --> มาก

ตัวอย่าง ครั้งแรกที่หยิบไพ่ใบแรก ไม่ต้องทำอะไร เพราะมีเพียง 1 ใบ

1. pass ที่ 1 เอาไพ่ใบที่ 2 คือ 6 เข้าไปในมือ (หาที่ให้ 6 อยู่) $6 < 8$ เลื่อน 8 ออกมา เทียบสุดแล้ว ใส่ 6 ในที่เดิมของ 8
2. pass ที่ 2 เอาไพ่ใบที่ 3 คือ 7 เข้าไปในมือ (หาที่ให้ 7 อยู่) $8 < 7$ เลื่อน 8 ออกมา 6 ไม่น้อยกว่า 7 พบที่สำหรับ 7 ใส่ 7 ในที่เดิมของ 8
3. pass ที่ 3 เอาไพ่ใบที่ 4 คือ 5 เข้าไปในมือ (หาที่ให้ 5 อยู่) $8 < 5$ เลื่อน 8 ออกมา $7 < 5$ เลื่อน 7 ออกมา $6 < 5$ เลื่อน 6 ออกมา สุดแล้ว ใส่ 5 ในที่เดิมของ 6

//ascending order, array size = n

// loop inserting i-th element

i = 1 // start form 2nd element

loop (i < n)

insertEle = a[i];

// find insert position ip

ip = i ;

loop (ip > 0 && insertEle <= a[ip-1])

a[ip] = a[ip-1]; //shift out other data

ip-- // to make place for

a[ip] = insertEle; // insertElement

i++; // 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

จากน้อย --> มาก

array size = n

Outer loop ทำตั้งแต่ i [1,n-1]

inner loop : ip[1 , i] = i ครั้ง

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

$$= 1+2+3+...+(n-1)$$

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

$$= O(n^2)$$

Insertion Sort



8	6	7	5	9	Original File : size = n
6	8	7	5	9	After pass 1
6	7	8	5	9	After pass 2
5	6	7	8	9	
5	6	7	8	9	Sorted file : after pass __

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

Pass #1, # comparisons: minimum = 1 maximum = 1

Pass #2, # comparisons: minimum = 1 maximum = 2

Last Pass # $n-1$ # comparisons: minimum = 1 maximum = $n-1$

Total #comparisons

- Worst case = $1+2+3+...+(n-1)$ = $\frac{n*(n-1)}{2}$ = $O(n^2)$ 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]          #insert element  
        for j in range(i, -1, -1):  
            if iEle < l[j-1] and j > 0:  
                l[j] = l[j-1]  
            else:  
                l[j] = iEle  
                break
```

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

					iEle=3	
p=4	1	4	5	6	3	2
					← j	

array size = n


Outer loop ทำตั้งแต่ i [1,n-1]

inner loop : ip[i , 1] = i ครั้ง

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

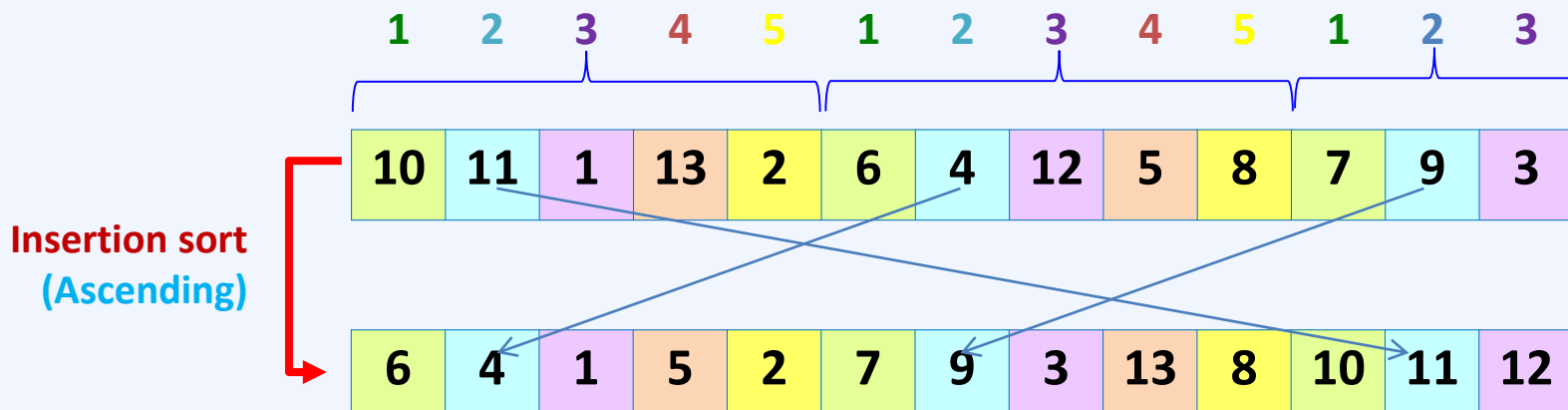
Shell Sort (Diminishing Increment Sort)

Shell Sort

- กำหนด incremental Sequence ของ interger ก็ตัวก็ได้ ตัวแรกเป็น 1 ไล่เพิ่มน้อยไปมาก
 $i_1, i_2, \dots, i_{\max}$ 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
- วิธีแบ่ง file ใหญ่ เป็น n file ย่อย นับข้อมูลที่ไล่ตัวไปตามลำดับตั้งแต่ 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 วังไกล
- ทำขั้นตอน 2-4 ใหม่โดยใช้ค่า i ตัวที่มากรองลงมา เมื่อทำการแบ่ง file โดยใช้ $i = 1$ ในที่สุด
 ก็จะทำ insertion sort กับข้อมูลทุกตัวพร้อมกัน จึงเรียก Diminishing Increment Sort เพราะ
 ใช้ incremental sequence ตัวที่ลดลงเรื่อยๆ



Donald Shell



Shell Sort (Diminishing Increment Sort)

Shell Sort

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

Diminishing ลดลง
start 5
3
1

2. for (int n = max; n >= 1; n--)

- แบ่ง file ใหญ่ เป็น n file ย่อย
- insertion sort แต่ละ file ย่อย

n มาก → file เล็ก
insertion sort $O(n^2)$ ok
n น้อย → ค่อนข้าง 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

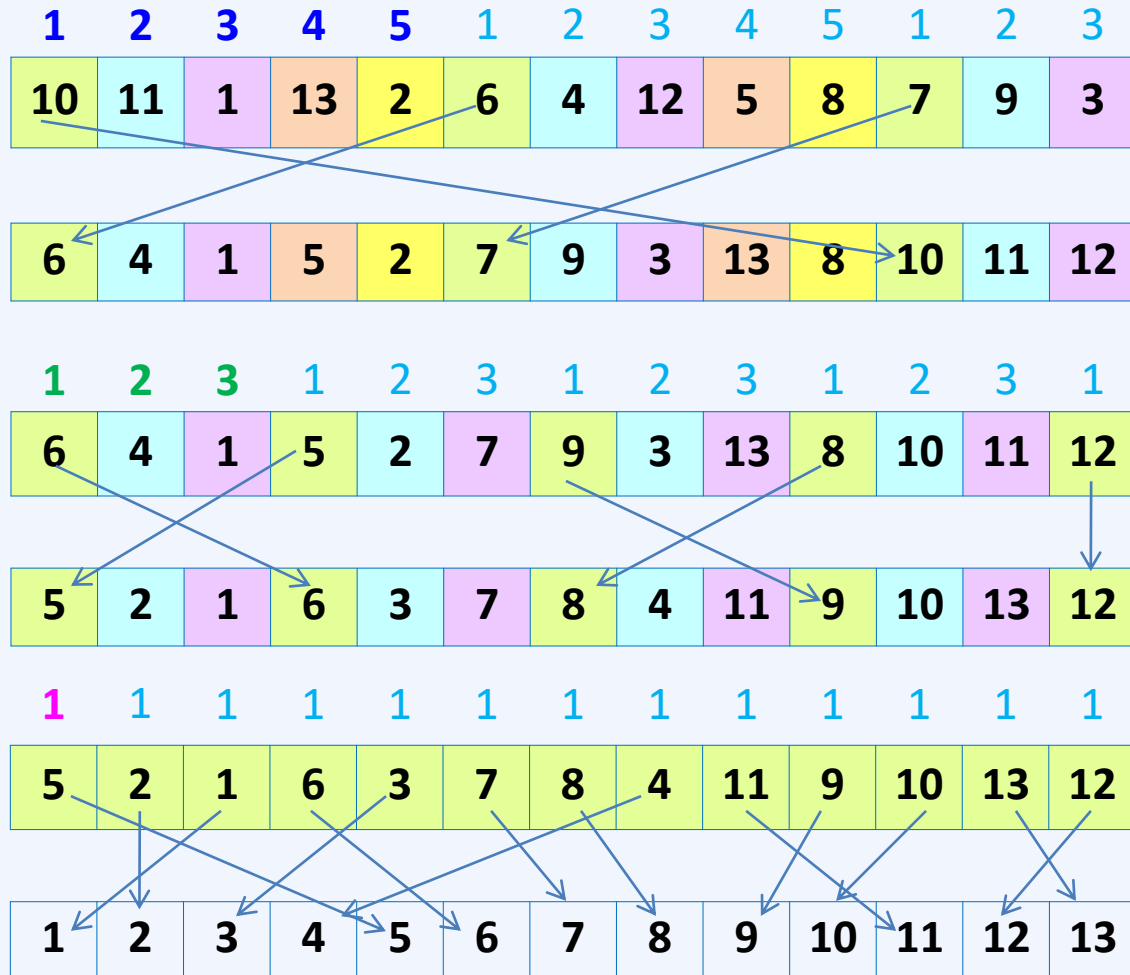
Insertion Sort

Best Case :
ordered list $O(n)$

Insertion

Insertion

Insertion



Shell Sort (Donald Shell)

```
def shell(l, dIncrements):
    for inc in dIncrements: #for each deminishing increment
        for i in range(inc,len(l)): #insertion sort
            iEle = l[i]    #inserting element
            for j in range(i, -1, -inc):
                if iEle<l[j-inc] 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)
```

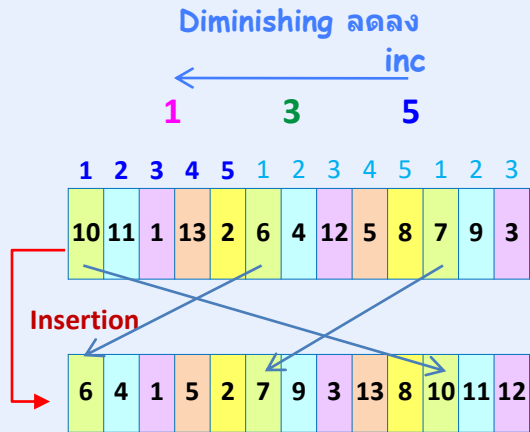
popular (but poor) increment (suggested by Shell)
$$h_t = \lfloor n/2 \rfloor \qquad h_k = \lfloor h_{k+1}/2 \rfloor$$

การเลือก **increment** สำคัญต่อ **performance** มาก

Sedgewick ได้เสนอ **increment sequence** ที่ดีไว้หลายอัน
หนึ่งในนั้นคือ **1, 5, 19, 41, 109, ...**

คือ **term** ในรูป $9 * 4^i - 9 * 2^i + 1$ หรือ $4^i - 3 * 2^i + 1$

ในทางปฏิบัติ **performance** เป็นที่ยอมรับได้ แม้ว่า **n** จะมีค่ามากเป็นหมื่นก็ตาม



Donald Shell



Binary Heap

Complete Binary Tree

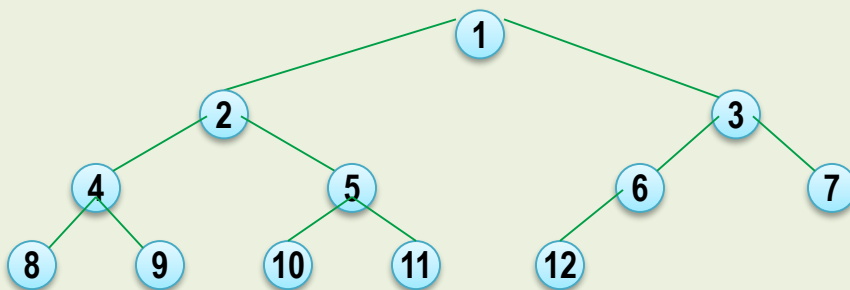
Review : Complete Binary Tree

Complete Binary Tree

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

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

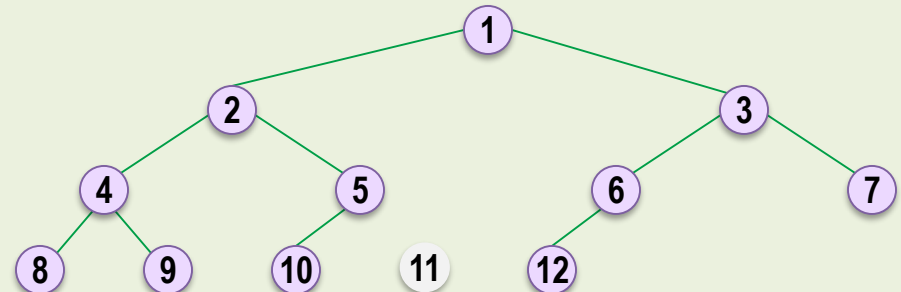
$$N \text{ ระหว่าง } [2^H, 2^{H+1} - 1]$$



Complete Binary Tree

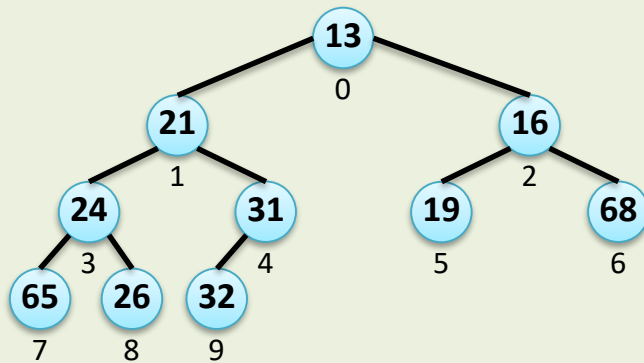
Every level is completely filled,
except possibly the last, &

All nodes are as far left as possible

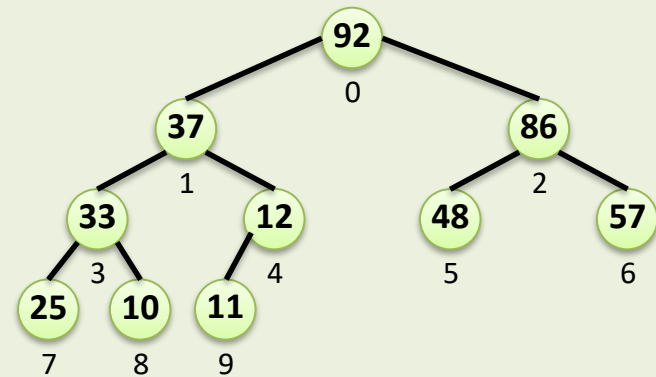


Without node 11 :
Not a complete binary tree

Binary Heap



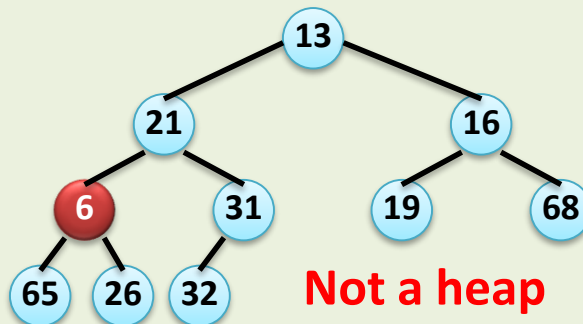
Ascending heap
(Min heap)



Decending heap
(Max heap)

Binary Heap : complete Binary Tree ซึ่ง key ของ node ใดๆ

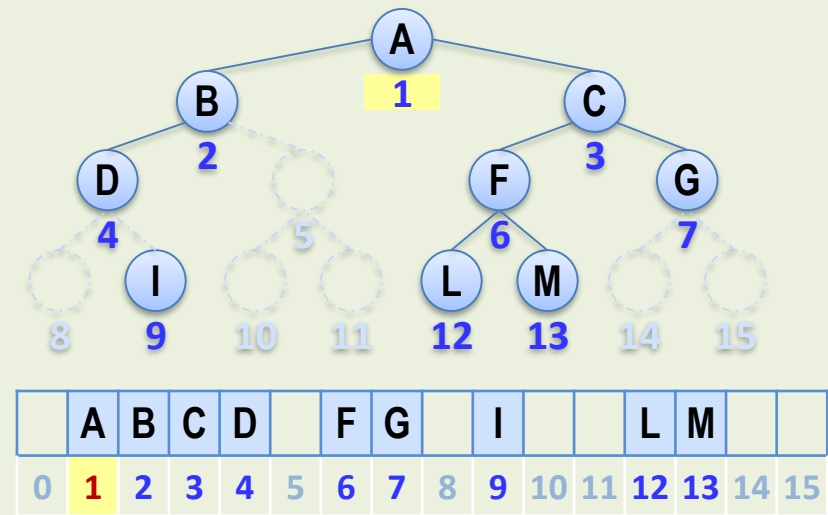
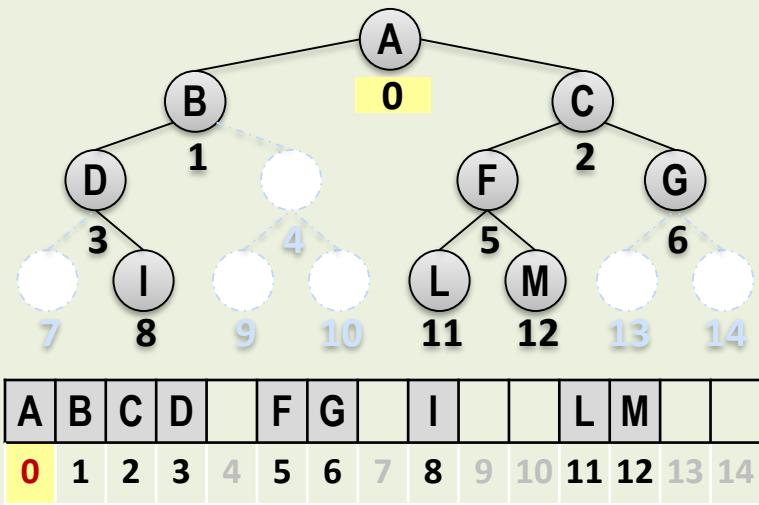
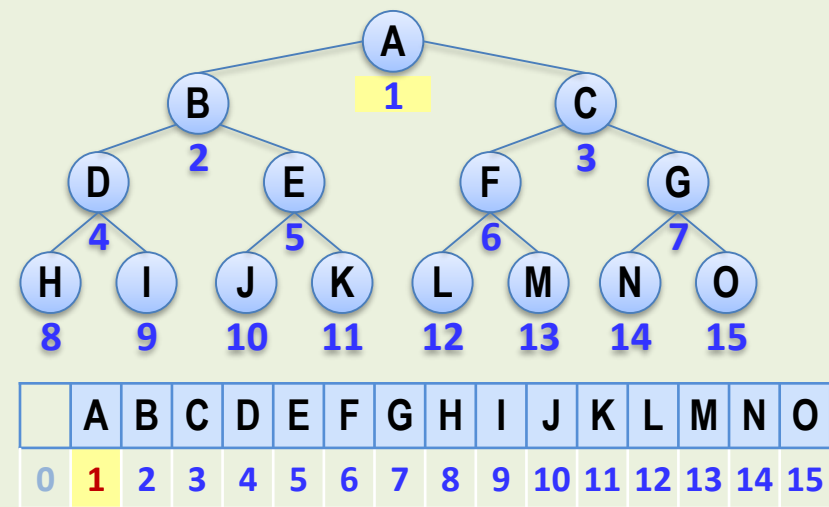
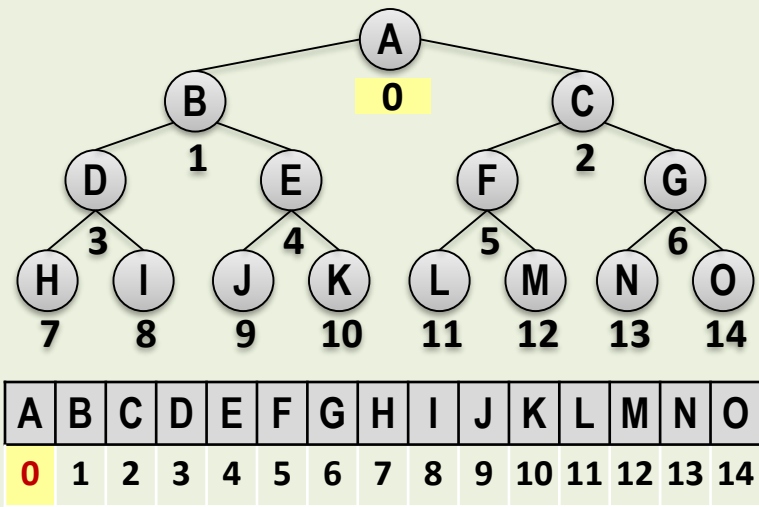
1. \leq key ของ decendents ของ มัน : **Min heap** เช่น 13 น้อยกว่าลูกหลานทั้งหมดของมัน
2. \geq key ของ decendents ของ มัน : **Max heap** เช่น 37 มากกว่าลูกหลานทั้งหมดของมัน



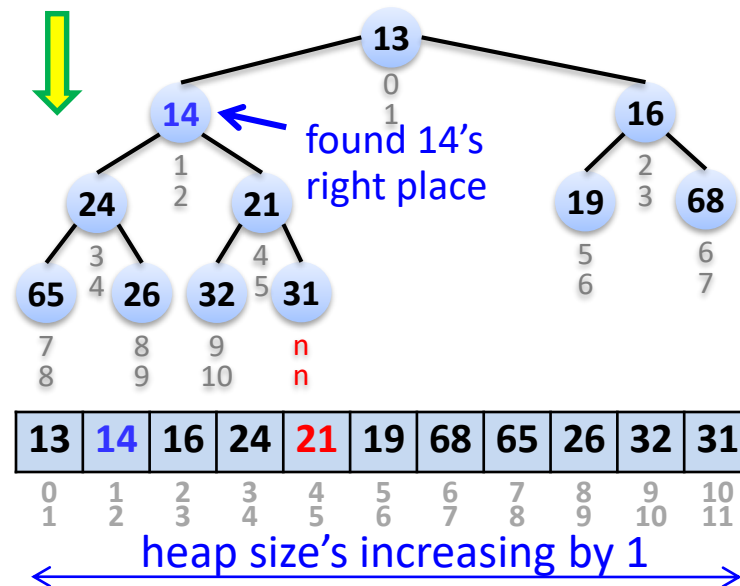
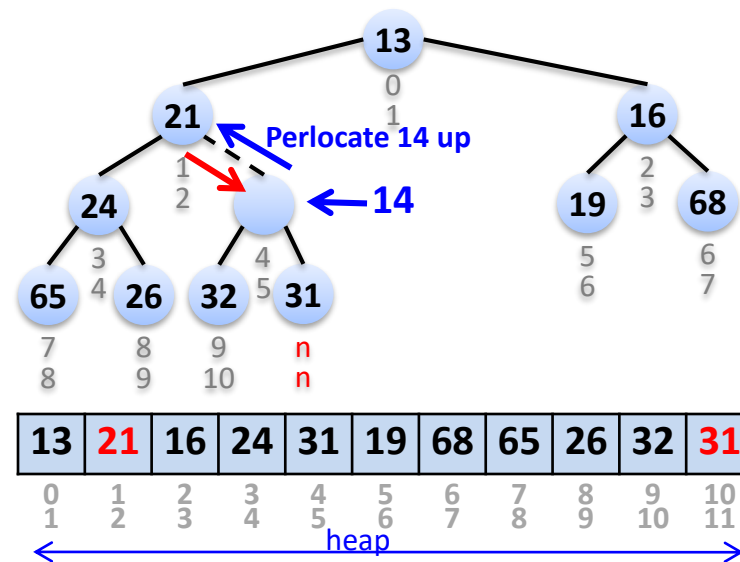
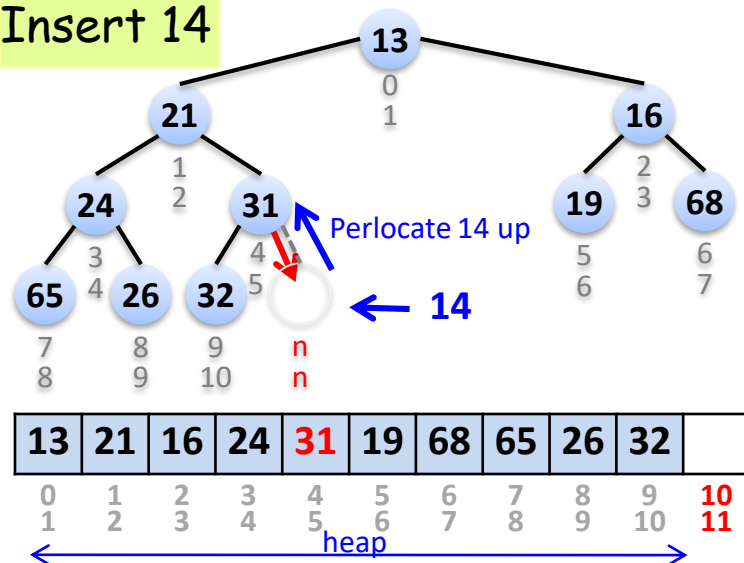
Not a heap

[home](#)

Review : (Sequential) Implicit Array



Insert 14



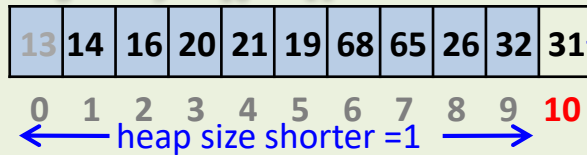
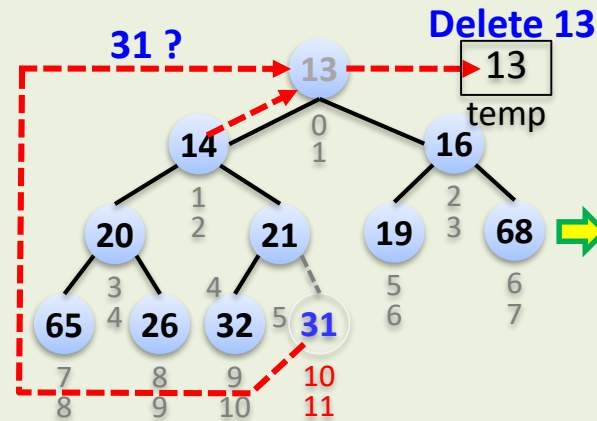
Inserting 14.

- ก่อนจะ insert : last index = $n-1$ (data 32)
- insert จะเพิ่มขนาด heap ขึ้น 1 ที่ตำแหน่ง n
- พยายามทำให้ 14 เริ่มลองที่ n
- ถ้าไม่ได้ perlocate 14 up ($14 < 31$) ($14 < \text{พ่อ}$), เลื่อนพ่อของมัน (31) down
- Repeat พยายามทำให้ 14 ตาม path ไปยัง root จนกว่าจะพบที่ของ 14

After Insertion

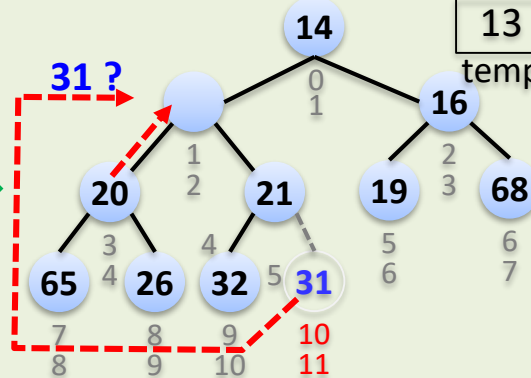
- ขนาด heap เพิ่มขึ้น 1
- การปรับ tree ให้เป็น heap อีกครั้งเรียกว่าการ reheap

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



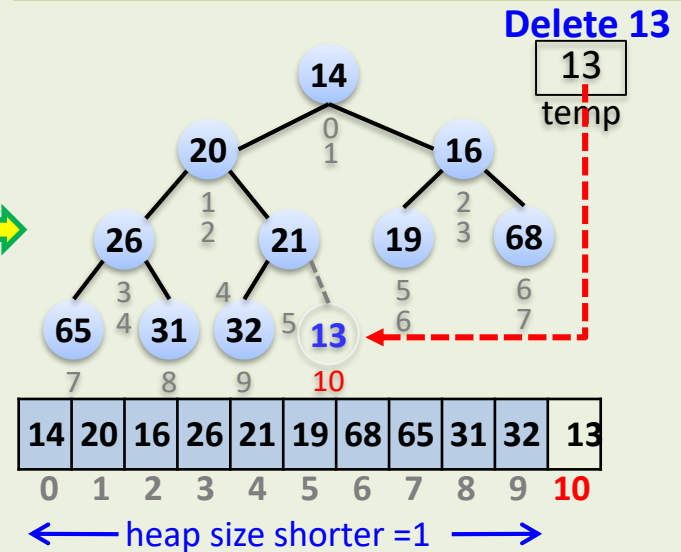
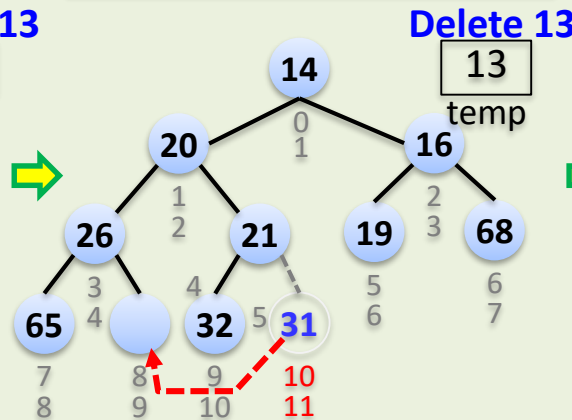
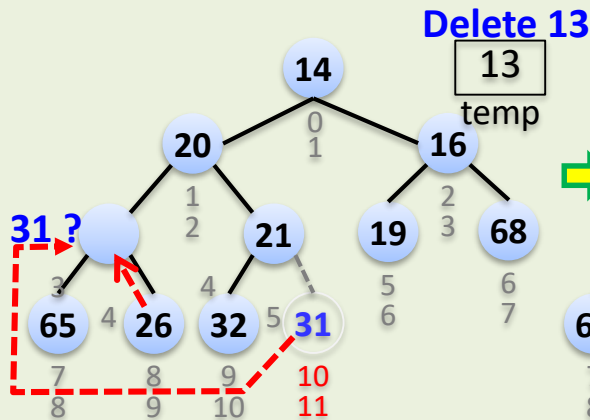
- Delete Min = $c \log_2 n$
- Delete min n times.
- Heap sort $\rightarrow O(n \log_2 n)$

Delete 13

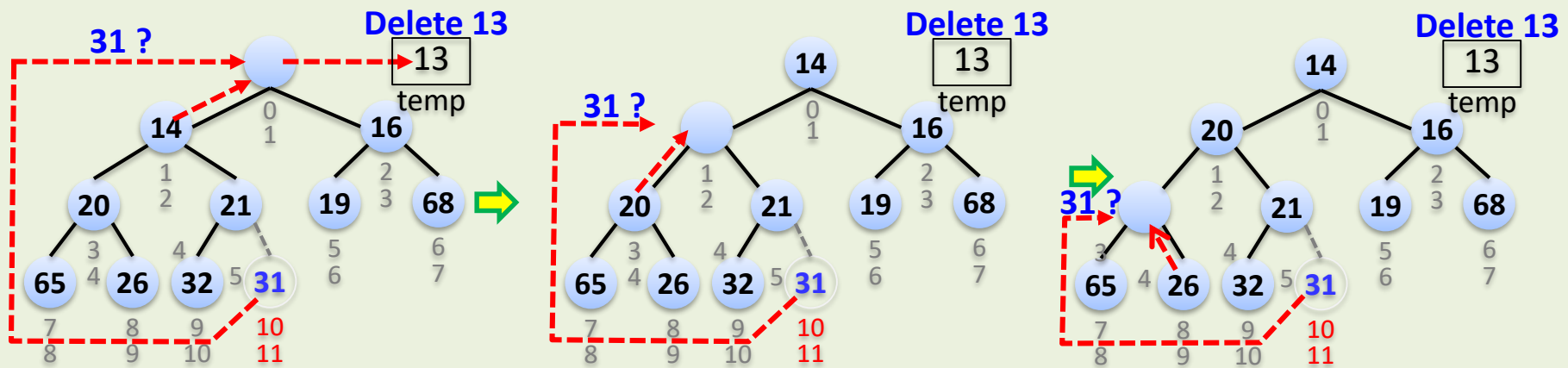


- Start at min heap from index : 0 to 10
- **Delete min** (root 13) \rightarrow hole at root.
- To delete, heap size shorter 1 (**last node**).
- Last node data, 31, must be in the heap.
- Find place for 31 (**reheap**). Try the hole.
- if not, **perlocate 31 down**, take the hole's smaller son up.
- Repeat finding place for 31.

- Put the deleting data 13 at previous-31-place in the same array is called **in place sort**, makes decending order.
- Now 13 is out of heap.

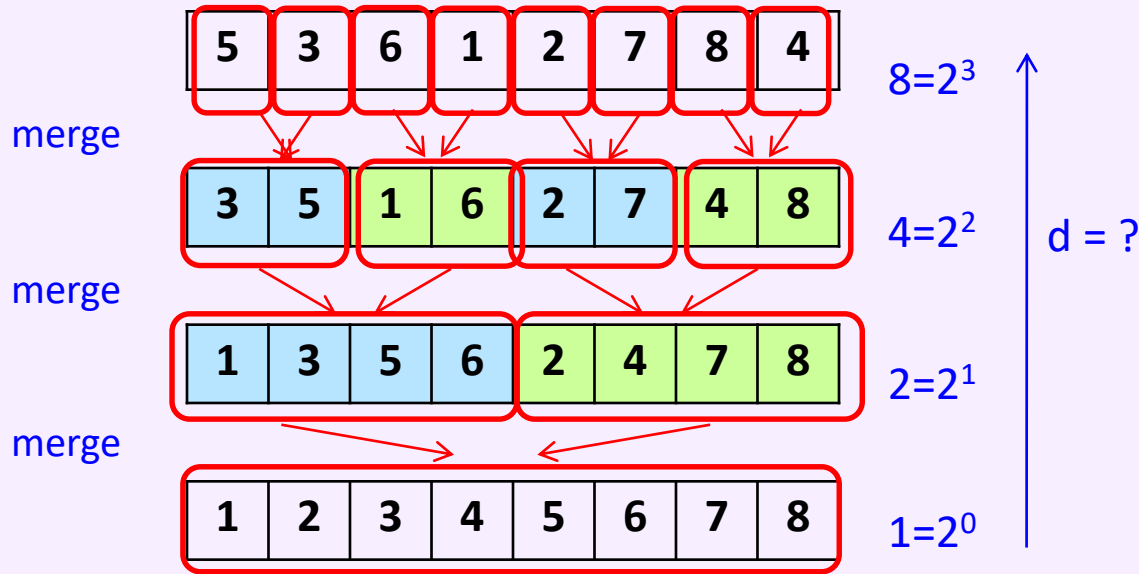


Heap Sort Analysis



- 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 แทบไม่มี extra space เลย หากทำ inplace sort
- avg case ดีไม่เท่า avg case ของ quick sort แต่
worst case แย่กว่า avg case ของ quick sort นิดเดียว
(worst case ของ quick sort = $O(n^2)$)
- จากการศึกษพบว่าโดยทั่วไป heap sort กินเวลาประมาณ 2 เท่าของ quick sort
- ในทางปฏิบัติพบว่าช้ากว่า Shell Sort ที่ใช้ incremental sequence ของ Sedgewick

Merge Sort

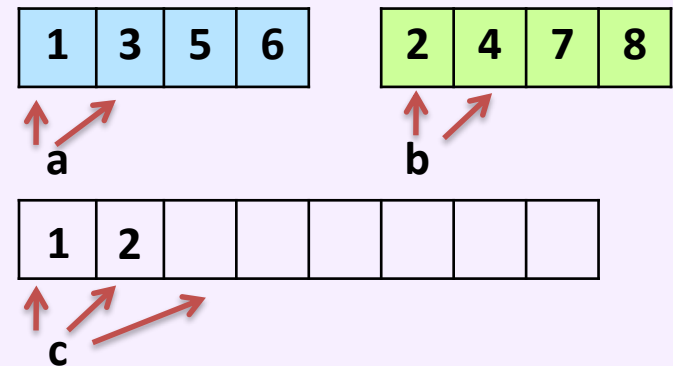


#comparisons = ____
#assignments = ____
Merge = $O(\text{____})$

Key idea :

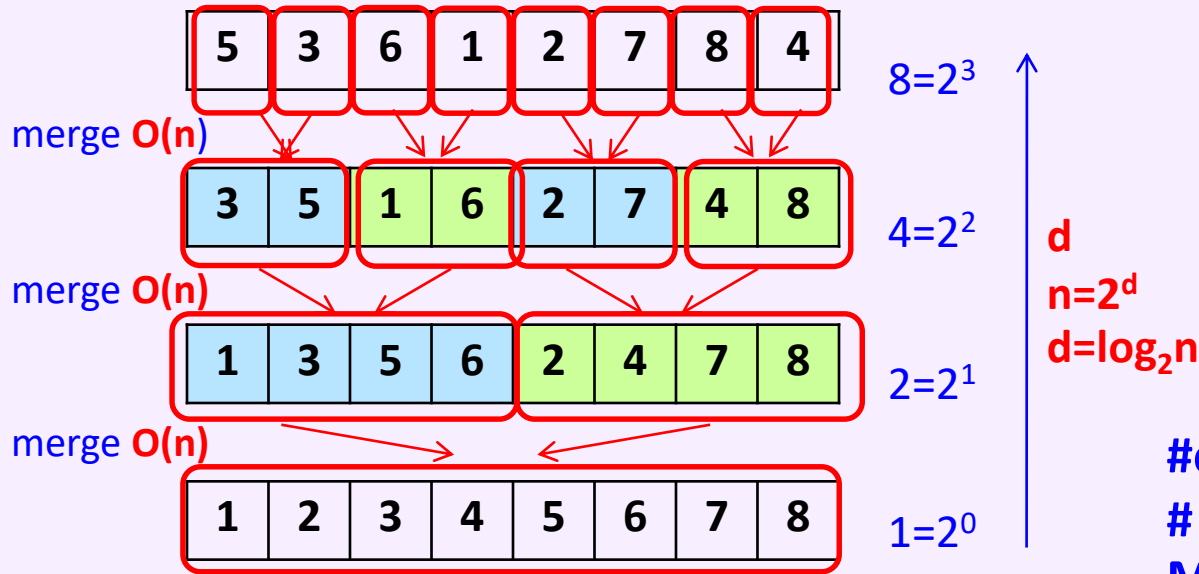
Merge : sorts successive pair to be sorted bigger one.

- Merge size 1 successive pair → sorted size 2
- Merge size 2 successive pair → sorted size 4
- Merge size 4 successive pair → sorted size 8



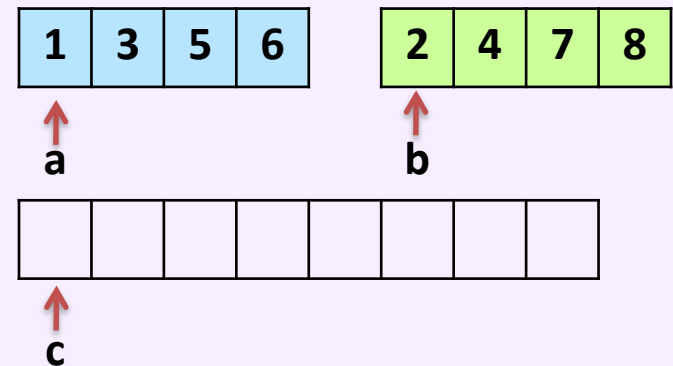
[home](#)

Merge Sort



#comparisons $\leq n$
assignments = n
Merge = $O(n)$

$$\text{mergeSort} = \text{merge } O(N) * d = N * \log_2 N$$



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)

$$T(1) = 1$$

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

$$\rightarrow \underline{T(N/2)}$$

$$\rightarrow \underline{T(N/2)}$$

$$\rightarrow \underline{N}$$

//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++

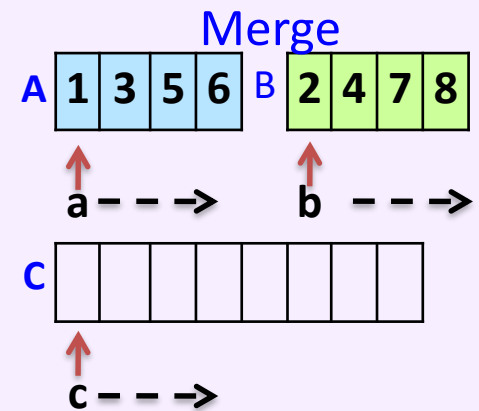
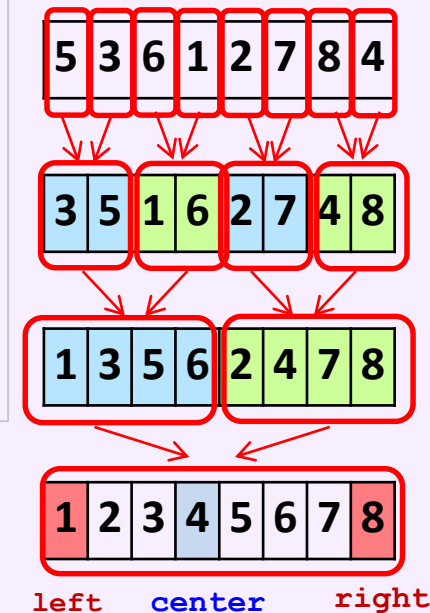
appending C with the remaining list

copy C back to a

$$\# \text{ comparisons} = \underline{<N}$$

$$\# \text{ assignments} = \underline{N}$$

$$O(\underline{N})$$



Merge Sort Analysis : Telescoping a sum

$$T(1) = 1 \quad \dots (1)$$

$$T(N) = 2 * T(N/2) + N \quad \dots (2)$$

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

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

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

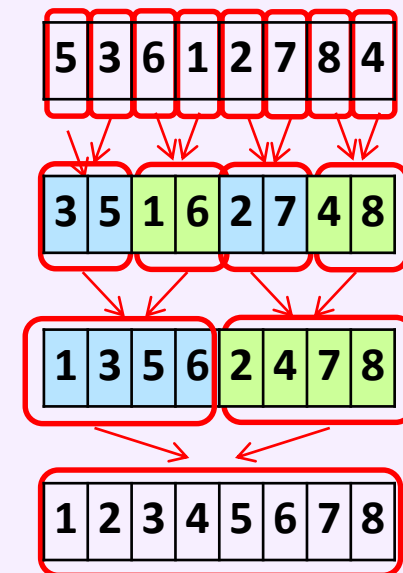
$$\frac{T(N/4)}{N/4} = \frac{T(N/8)}{N/8} + 1 \quad \dots (*) \quad "$$

...

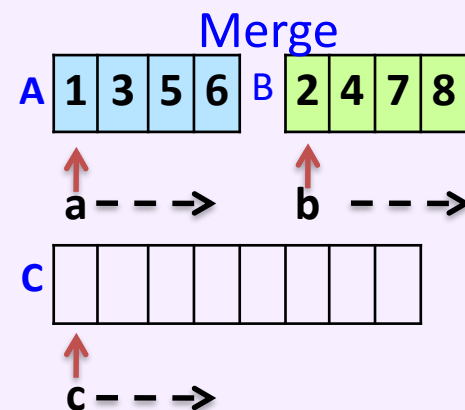
$$\frac{T(2)}{2} = \frac{T(1)}{1} + 1 \quad \dots (*) \quad "$$

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

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



$\log_2 n$



Merge Sort Analysis : Brute Force

$$T(1) = 1 \quad \dots (1)$$

$$T(N) = 2 * T(N/2) + N \quad \dots (2)$$

$$T(N) = 2 * (2 * T(N/4) + N/2) + N$$

$$T(N) = 4 * T(N/4) + 2N$$

$$T(N) = 4 * (2 * T(N/8) + N/4) + 2N$$

$$T(N) = 8 * T(N/8) + 3N$$

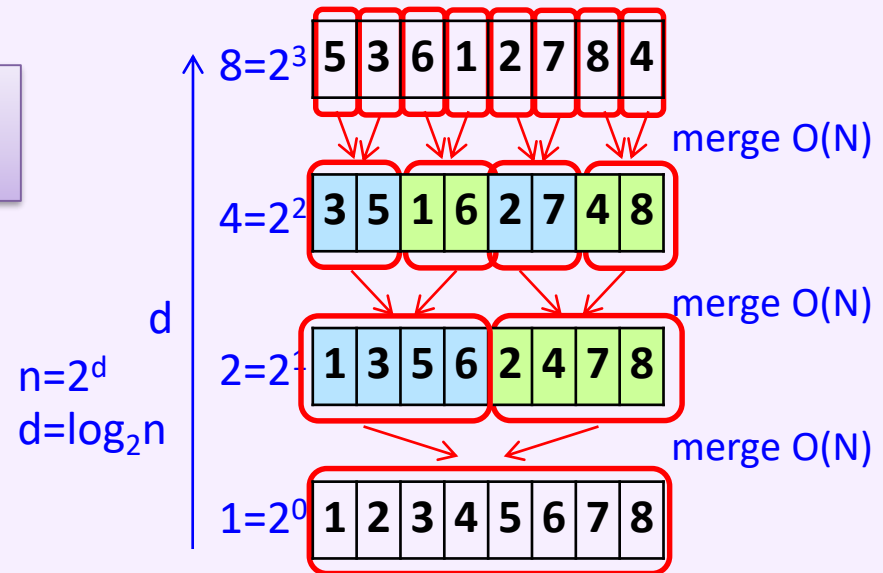
$$T(N) = 16 * T(N/16) + 4N$$

...

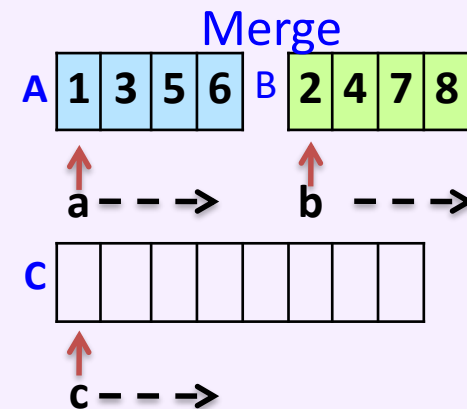
$$T(N) = 2^d * T(N/2^d) + dN \quad \dots (3)$$

$$T(N) = N + \log_2 N * N : d = \log_2 n, n = 2^d$$

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



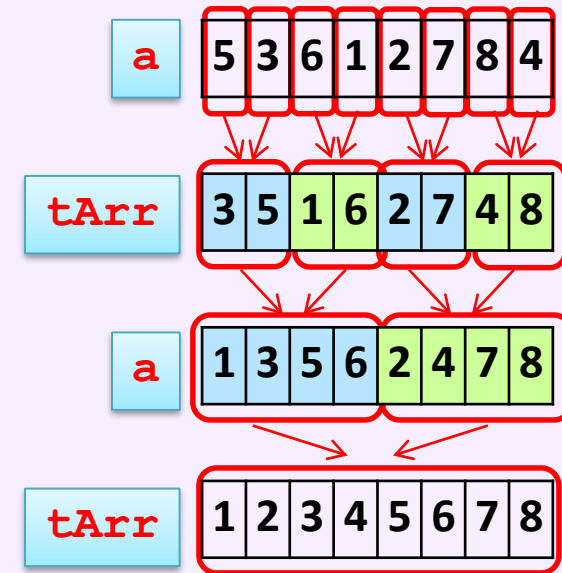
$$\text{mergeSort} = \text{merge } O(N) * d = N * \log_2 N$$



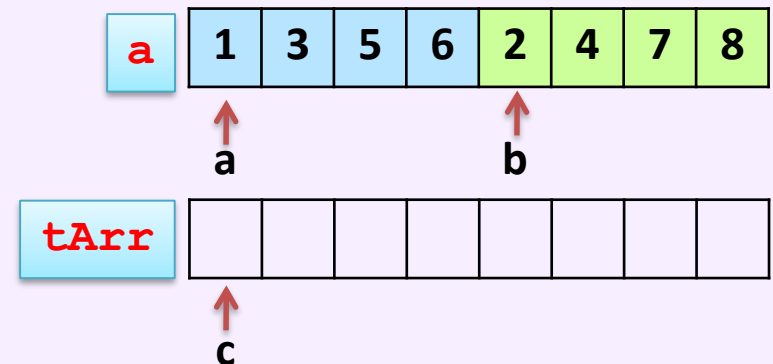
Merge Sort Code : Python

```
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)
```



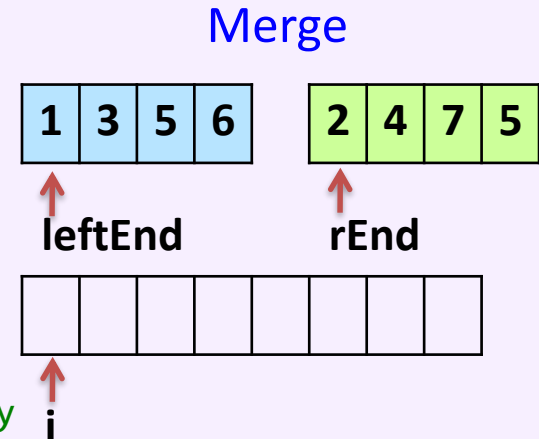
merge



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
```



Quick Sort



Key idea :

1. Choose the pivot

- 1st 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

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
---	---	---	---	---	---	---	---	---	---

Left half $<$ pivot pivot Right half $>$ pivot

pivot's
right place

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

Ordered List

3. Repeat partitioning both left & right half

Quick Sort Partition : 1st 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

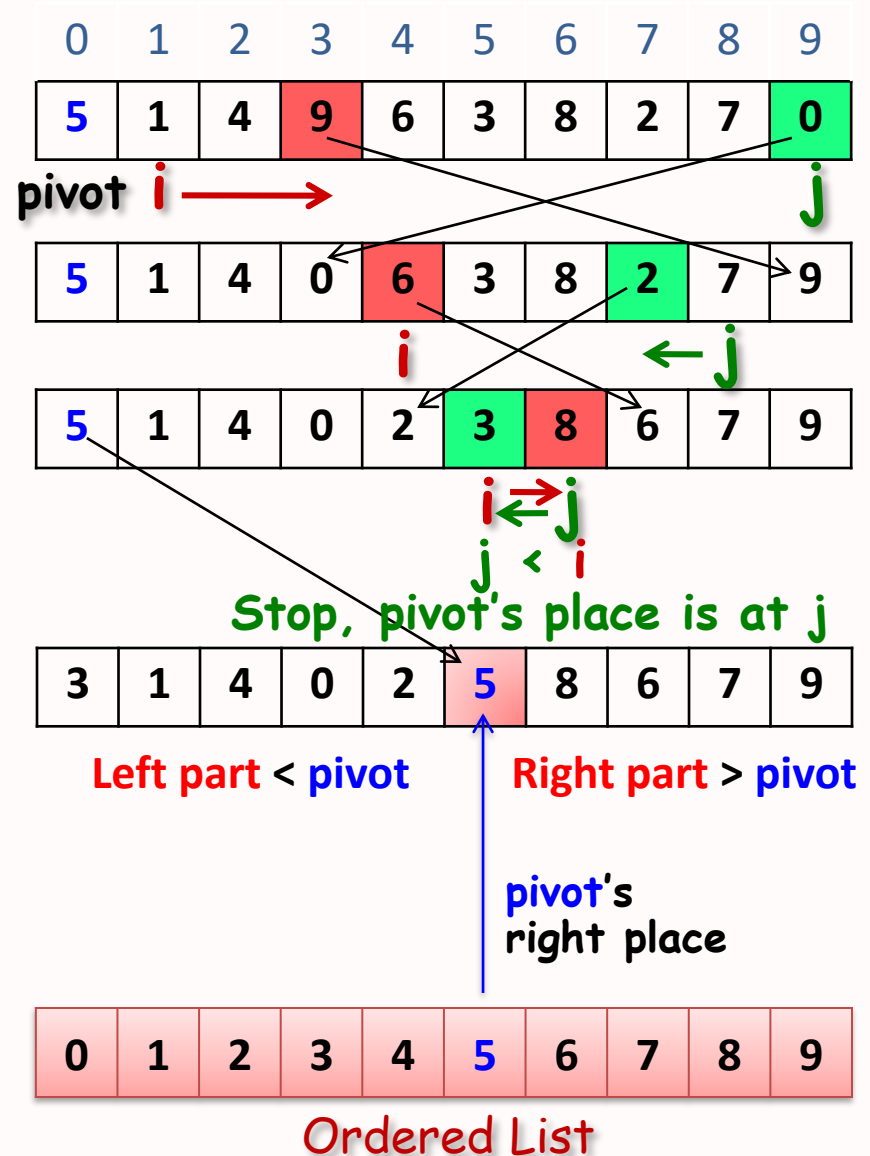
if(i < j) swap elements at i and j

//if i > j means all list is scanned

swap **pivot** to right pos at j

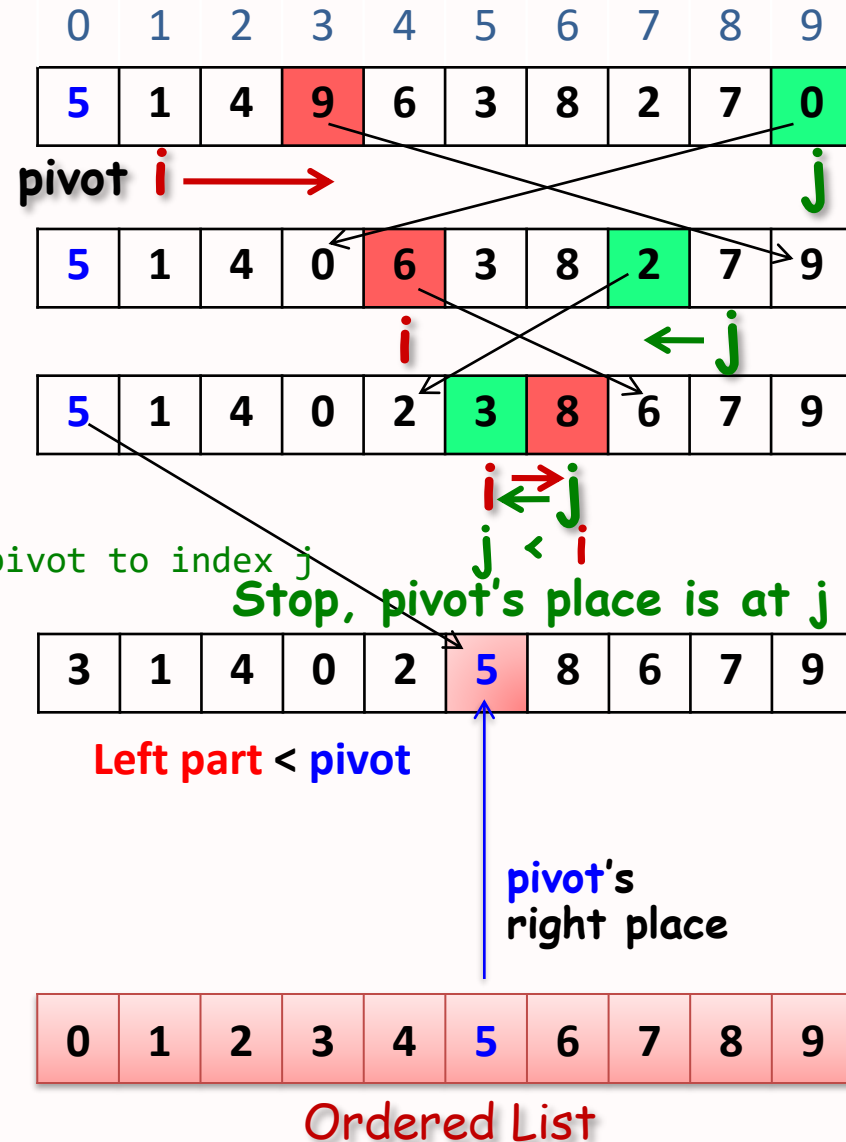
QuickSort left half a[left,j-1]

QuickSort right half a[j+1,right]



Quick Sort Python Code : 1st element

```
def quick(l, left, right):  
    if left < right:  
        #partition  
        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  
        quick(l, left, j-1)  
        quick(l, j+1, right)  
  
l = [5,1,4,9,6,3,8,2,7,0]  
quick(l,0,len(l)-1)  
print(l)
```



Recursion of Quicksort

```
quick sort[0,9], pivot=5
0 1 2 3 4 5 6 7 8 9
[5 1 4 9 6 3 8 2 7 0]
      i                j i,j=3,9 swap(9,0)
[5 1 4 0 6 3 8 2 7 9]
      i      j      i,j=4,7 swap(6,2)
[5 1 4 0 2 3 8 6 7 9]
      j i      i,j=6,5 pvPos=5 swap(5,3)
[3 1 4 0 2 5 8 6 7 9]
```

```
quick sort[0,4], pivot=3
0 1 2 3 4
[3 1 4 0 2]
      i      j i,j=2,4 swap(4,2)
[3 1 2 0 4]
      j i i,j=4,3 pvPos=3 swap(3,0)
[0 1 2 3 4]
```

```
quick sort[0,2], pivot=0
0 1 2
[0 1 2]
j i      i,j=1,0 pvPos=0
[0 1 2]
```

```
quick sort[1,2], pivot=1
```

```
1 2
[ 1 2]
      j i i,j=2,1 pvPos=1
[ 1 2]
```

```
quick sort[6,9], pivot=8
```

```
6 7 8 9
[      8 6 7 9]
      j i i,j=9,8 pvPos=8
```

```
swap(8,7)
```

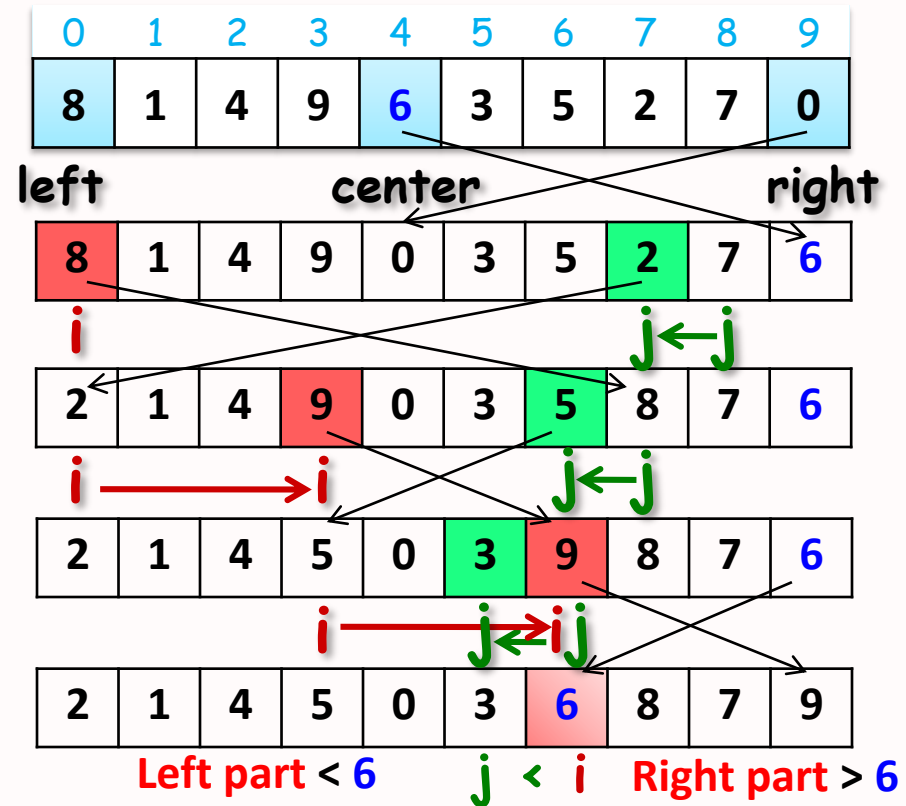
```
[      7 6 8 9]
```

```
quick sort[6,7], pivot=7
```

```
6 7
[      7 6]
      j i i,j=7,7 pvPos=7 swap(7,6)
[      6 7]
[0 1 2 3 4 5 6 7 8 9]
```

Quick Sort Partition : Median of Three 1

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



Stop, pivot's place is at i

pivot's right place

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

Ordered List

Quick Sort Partition : Median of Three : version 2

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

median of three(left,center,right)
make data in 3 pos. order
place pivot in pos. right-1

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

Left part < 6

j < i

Right part > 6

Stop, pivot's place is at i

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

Ordered List pivot's right place



Quick Sort (Previous Algorithm)

```
template <class T>
void quicksort(vector<T> & a,
               int left,int right){
```

```
    if( left + 10 <= right ) {
```

```
        T pivot = median3(a,left,right);
```

```
        //make data in 3 pos. Order
```

```
        //place pivot in pos. right - 1
```

```
        //Begin partitioning
```

```
        int i = left, j = right-1;
```

```
        for( ; ; ){
```

```
            while( a[++i] < pivot ) { }
```

```
            while( pivot < a[--j] ) { }
```

```
            if( i < j )
```

```
                swap(a[i],a[j]);
```

```
            else
```

```
                break;
```

```
        } //for
```

```
        //restore pivot
```

```
        swap(a[i],a[right-1]);
```

```
        quicksort(a,left,i-1);
```

```
        quicksort(a,i+1,right);
```

```
    } //if
```

```
    else
```

```
        // sort small size data
```

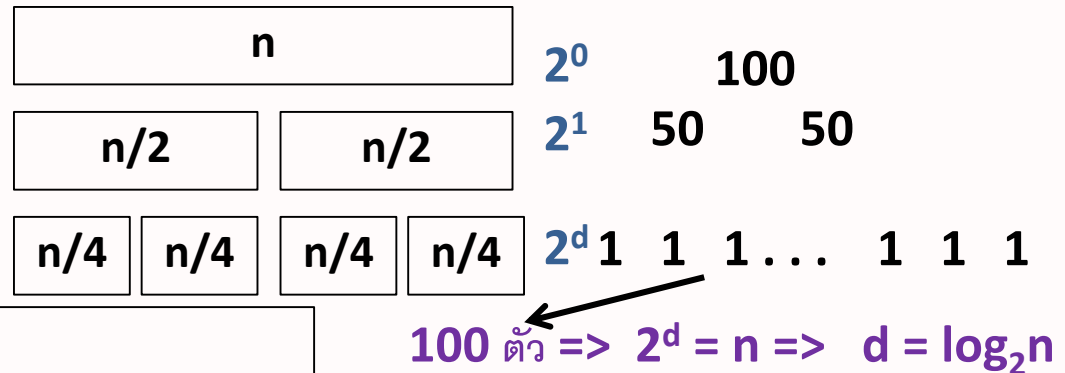
```
        insertionSort(a,
                      left,right);
```

```
}
```



Quick Sort Analysis

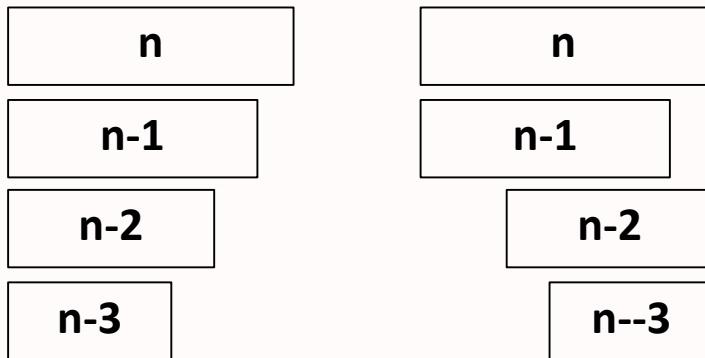
- แต่ละ pass compare n ครั้ง
ขึ้นกับ pivot เป็นอย่างไร
- Best Case แบ่งได้ 1/2 ทุกครั้ง



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

$$T(N) = cN \log N + N = O(N \log N)$$

- Worst Case = $n + (n-1) + (n-2) + \dots + 1 = O(n^2)$



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

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

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

...

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

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

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