Module5—Priority Queue(Heap)

5.0 Priority Queue(คิวแบบมีลำดับความสำคัญ)

เป็นชนิดข้อมูลนามธรรม**คิว**ที่มี

- การจัดเก็บข้อมูล ตามลำดับความสำคัญ จากมากไปน้อย แทนที่จะเป็นลำดับเวลาที่เข้ามาก่อนหลัง
- ข้อมูลที่มีลำดับความสำคัญสูงสุดอยู่ที่หัวคิว
- ข้อมูลที่มีความสำคัญต่ำสุดอยู่ที่ท้ายคิว
- การนำข้อมูลเข้า Priority Queue จะกระทำที่ท้ายคิว เช่นเดียวกับคิวปกติ

ตัวอย่าง

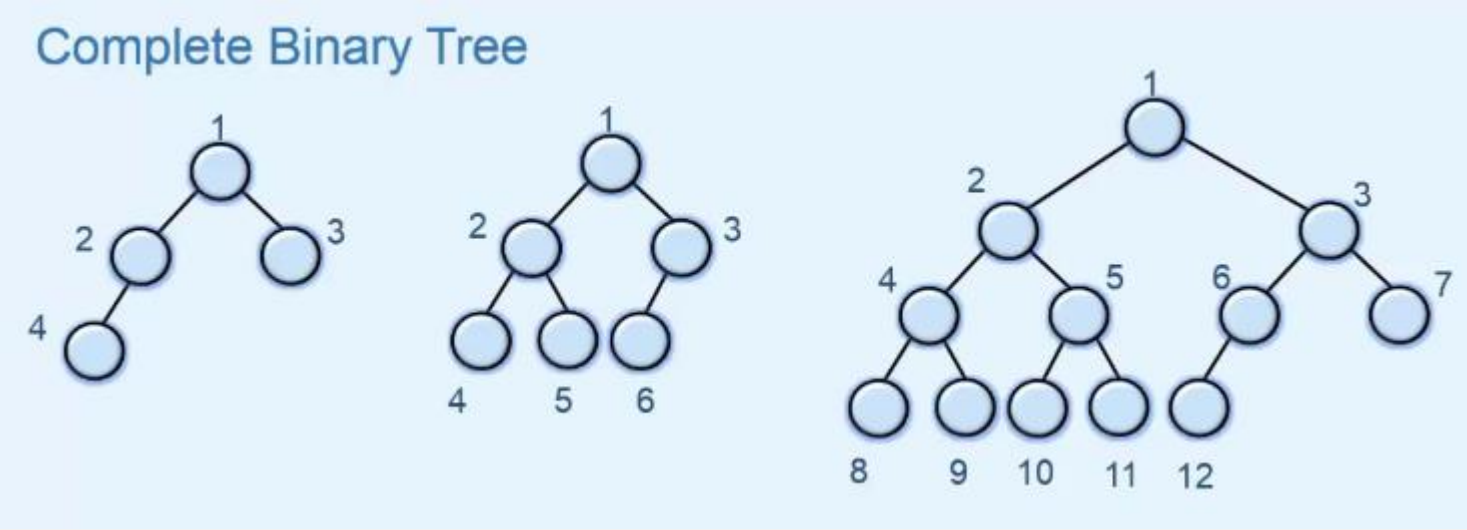
การให้การรักษาผู้ป่วยที่มีอาการหนักก่อน แม้จะมาทีหลัง

การสั่งงานใน printer ส่วนกลาง

การสร้าง priority queue ในบทนี้จะใช้โครงสร้างข้อมูล แบบ Heap

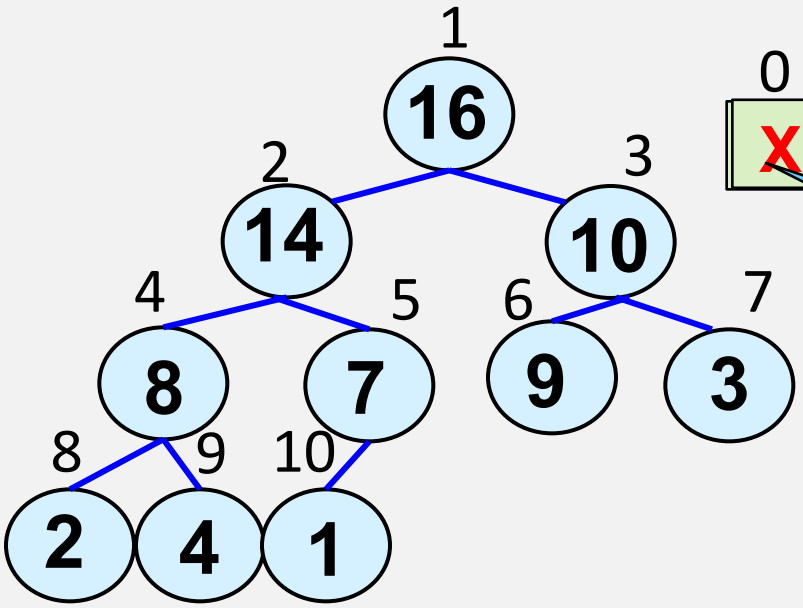
Complete Binary Tree คือ

- ไบนารีทรี
- ที่แต่ละสมาชิกจะต้องมีสมาชิกครบทั้งโหนดลูกทางซ้าย (Left Child Node) และโหนดลูกทางขวา (Right Child Node)
- ยกเว้นโหนดในระดับใบ (Leaves) ที่สามารถมีสมาชิกไม่ครบได้



5.1 Heap

Definition The (binary) heap data structure is an array object that can be viewed as a complete binary tree. Each node of the tree corresponds to an element of the array that stores the value in the node.



0	1	2	3	4	5	6	7	8	9	10
X	16	14	10	8	7	9	3	2	4	1

ช่องที่ 0 ไม่ใช้เก็บข้อมูล

ตอบคำถาม

1. Priority Queue คืออะไร
2. Heap คืออะไร
3. ถ้ากำหนด int A[30]; เก็บ heap
   ค่า 30 คือ length หรือ heapsize

A is an array represent heap

attribute

- int A[30];

30

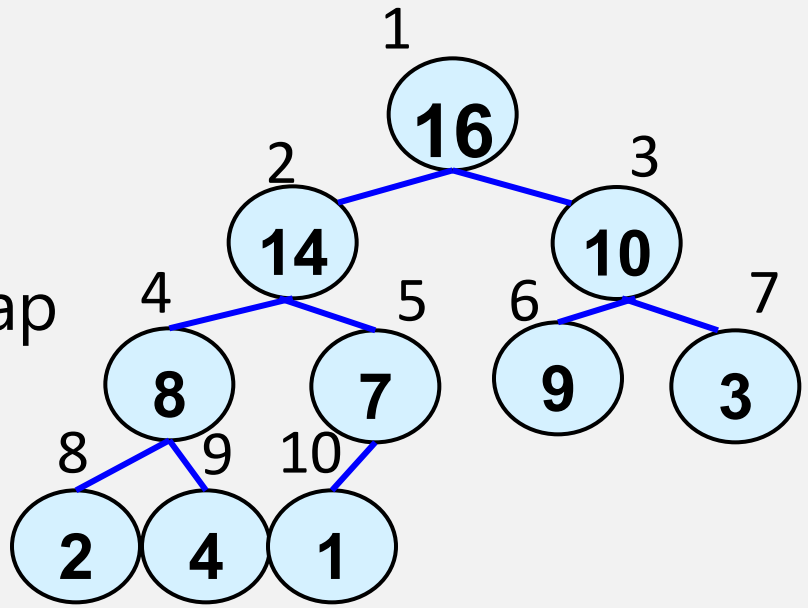
- length ขนาดของ A

10

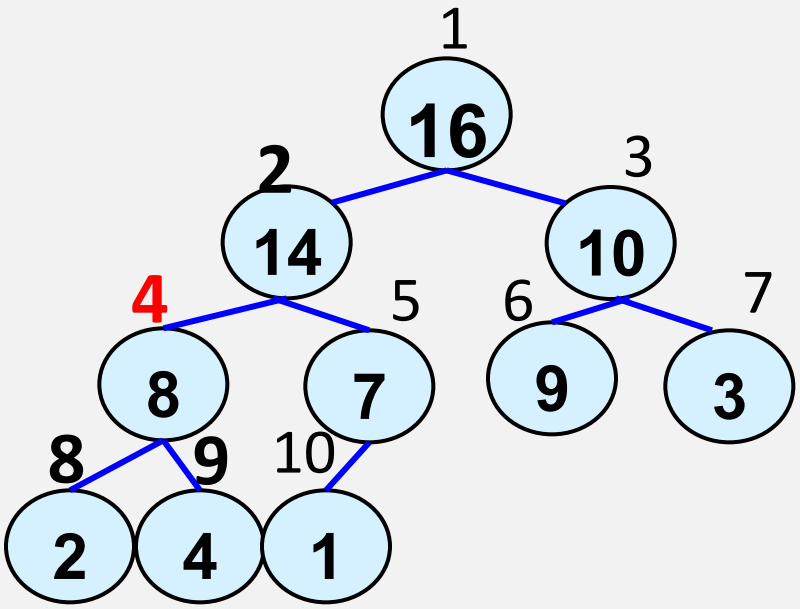
- heapsize จำนวนสมาชิกใน heap

- heapsize <= length

0	1	2	3	4	5	6	7	8	9	10	..
X	16	14	10	8	7	9	3	2	4	1	



	1	2	3	4	5	6	7	8	9	10
X	16	14	10	8	7	9	3	2	4	1



i=4

parents(i)
return i/2

Parents =2

left(i)
return 2i

Left = 8

right(i)
return 2i+1

right = 9

	1	2	3	4	5	6	7	8	9	10
X	16	14	10	8	7	9	3	2	4	1

i=2

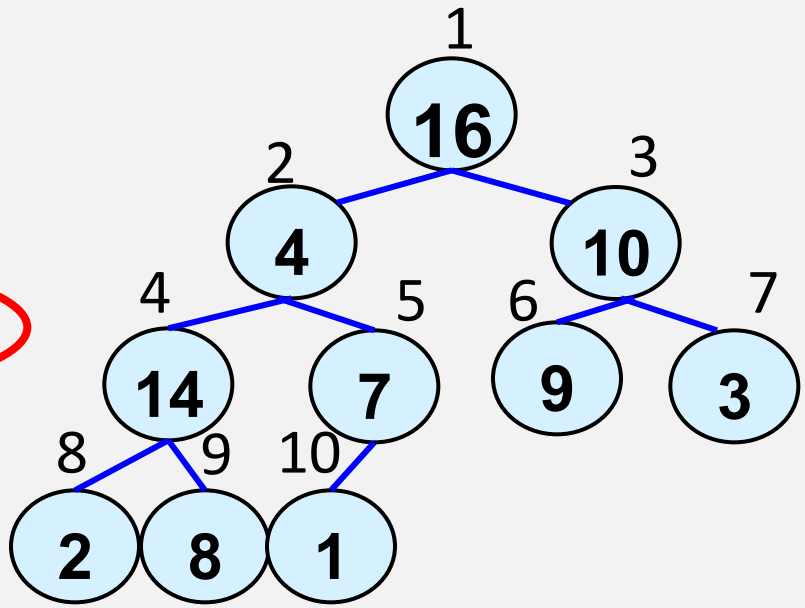
parent คือ?
left คือ ?
right คือ ?

5.1.1 Heap property :

Heaps also satisfy the heap property: for every node i other than the root,

A[parent(i)] >= A[i]

ตัวใดที่ขาดคุณสมบัติ



การเปลี่ยน Array ธรรมดาให้เป็น heap ด้วย algorithm heapify

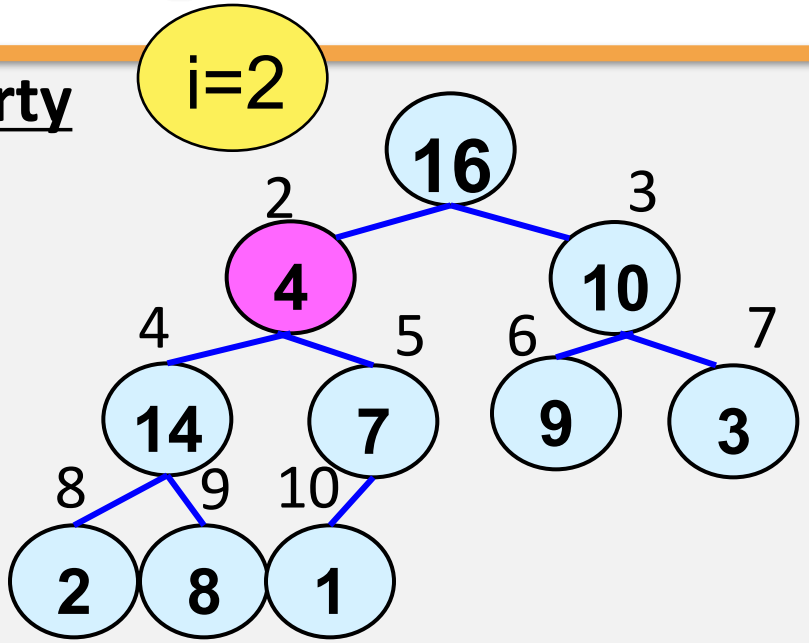
5.1.2 Maintaining t eap property

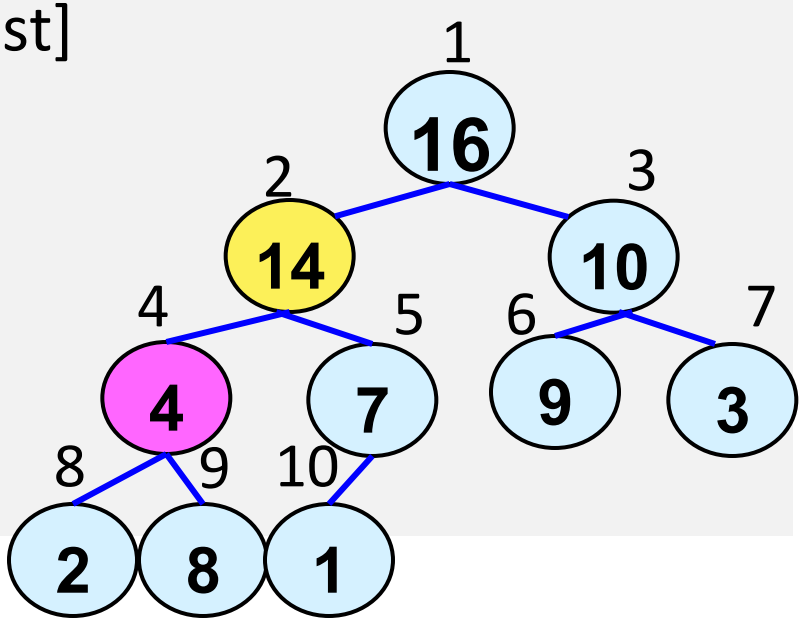
```
Heapify(A,i)
  l=left(i)
  r=right(i)
  if l <= heapsize and  A[l] > A[i]
        then    largest = l
        else    largest = i
  if r <= heapsize and  A[r] > A[largest]
        then largest =r
  if largest != i
        then
            exchange(A[i],A[largest])
            Heapify(A,largest)
```

4

5

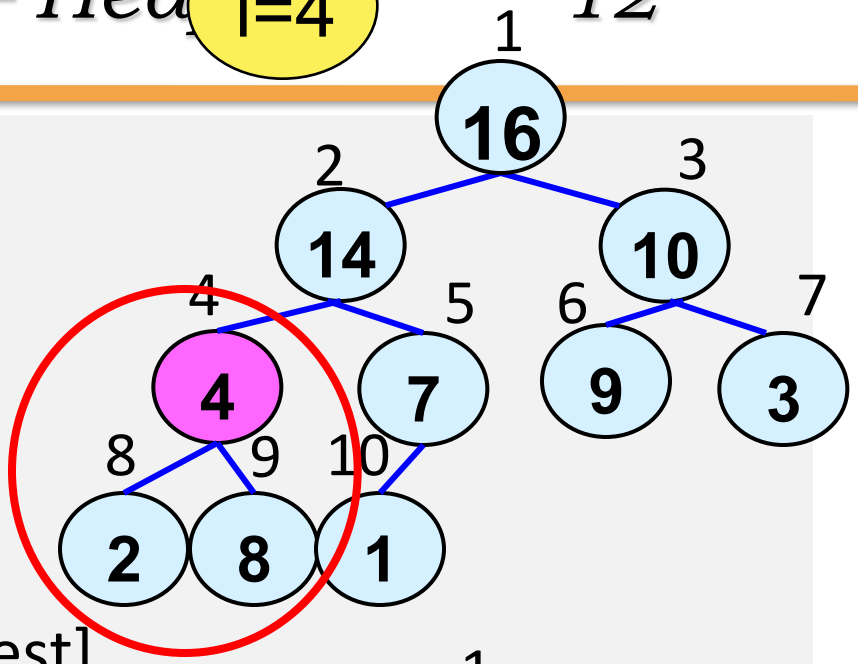
4, 14



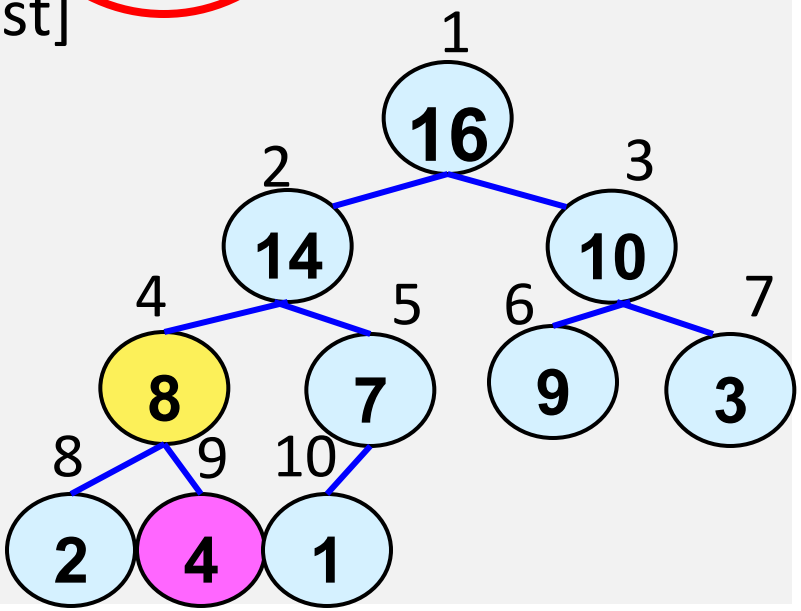


i=4

```
Heapify(A,i)
  l=left(i)
  r=right(i)
  if l <= heapsize and  A[l] > A[i]
        then    largest = l
        else    largest = i
  if r <= heapsize and  A[r] > A[largest]
        then largest =r
  if largest != i
        then
            exchange(A[i],A[largest])
            Heapify(A,largest)
```

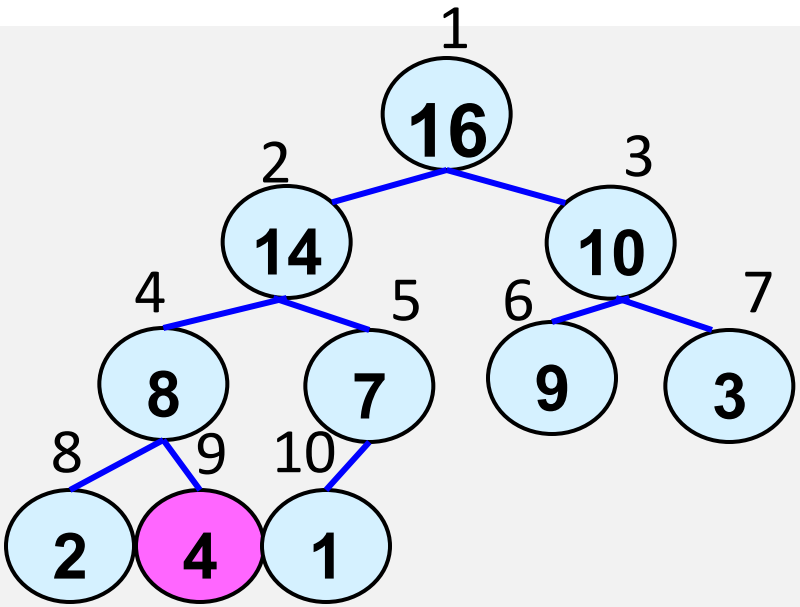


4, 8



i=9

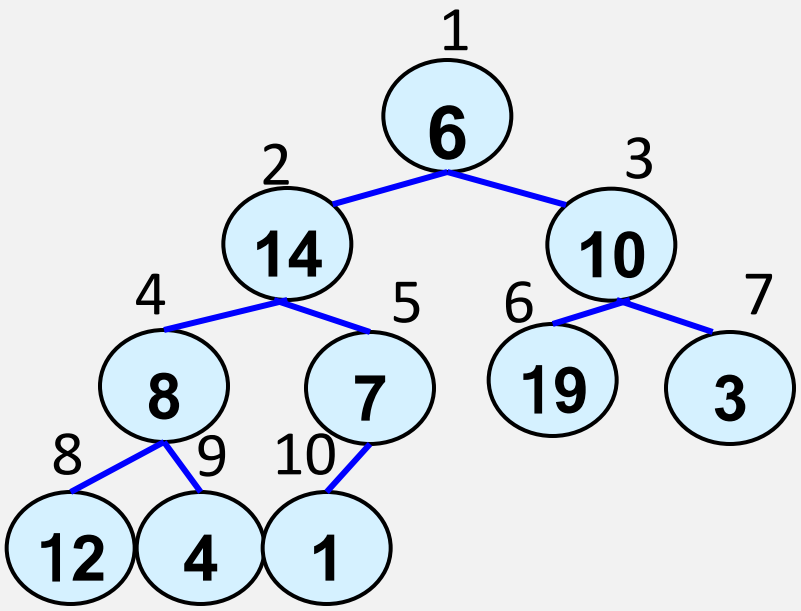
```
Heapify(A,i)
  l=left(i)
  r=right(i)
  if l <= heapsize and  A[l] > A[i]
        then    largest = l
        else    largest = i
  if r <= heapsize and  A[r] > A[largest]
        then largest =r
  if largest != i
        then
           exchange(A[i],A[largest])
           Heapify(A,largest)
```

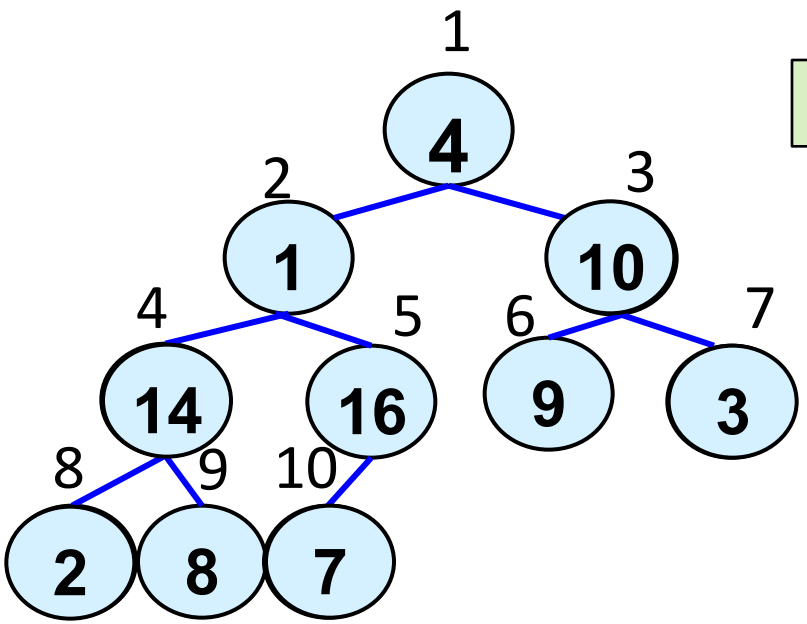


algorithm heapify เป็นการสร้าง array ให้เป็น heap ทีละโหนด

คำถาม

ถ้า heapify i=1 แล้ว array นี้จะมีคุณสมบัติเป็น heap ทั้ง array หรือไม่





0	1	2	3	4	5	6	7	8	9	10
X	4	1	3	2	16	9	10	14	8	7

X	4	1	3	2	16	9	10	14	8	7
X	4	1	3	14	16	9	10	2	8	7
X	4	1	10	14	16	9	3	2	8	7
X										
X										

```
Build_heap(A)
{    length=heapsize
     for(i= length/2 downto 1)
          Heapify(A,i)
}
```

คำถาม

1. Heapify 1 ค่า มี bigO เท่าใด
2. Build head มี bigO เท่าใด

- เมื่อ Array มีคุณสมบัติเป็น heap ข้อมูลตัวที่มากที่สุดจะอยู่ด้านหน้าสุด

- สามารถนำคุณสมบัตินี้มาใช้ในการเรียงข้อมูลในอะเรย์ เช่นการเรียงข้อมูลจากไปไปน้อย

ต้องการเรียงข้อมูลจากน้อยไปมาก

Heap

Heap

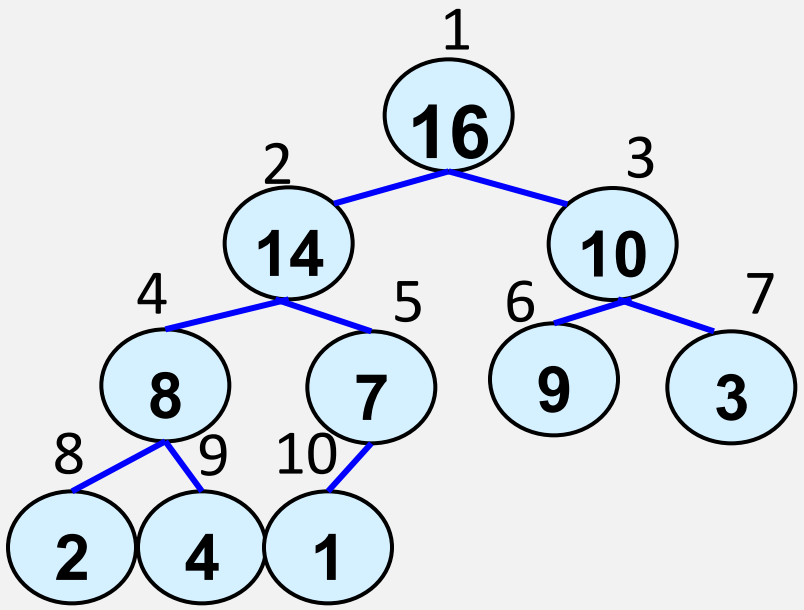
Heap

5.1.4 Heap sort :

	1	2	3	4	5	6	7	8	9	10
X	16	14	10	8	7	9	3	2	4	1

```
Heapsort(A)
  Build_heap(A)
  for i = length  down to 2
  do
     exchange (A[1], A[i])
     heapsize = heapsize -1;
      Heapify(A,1)
```

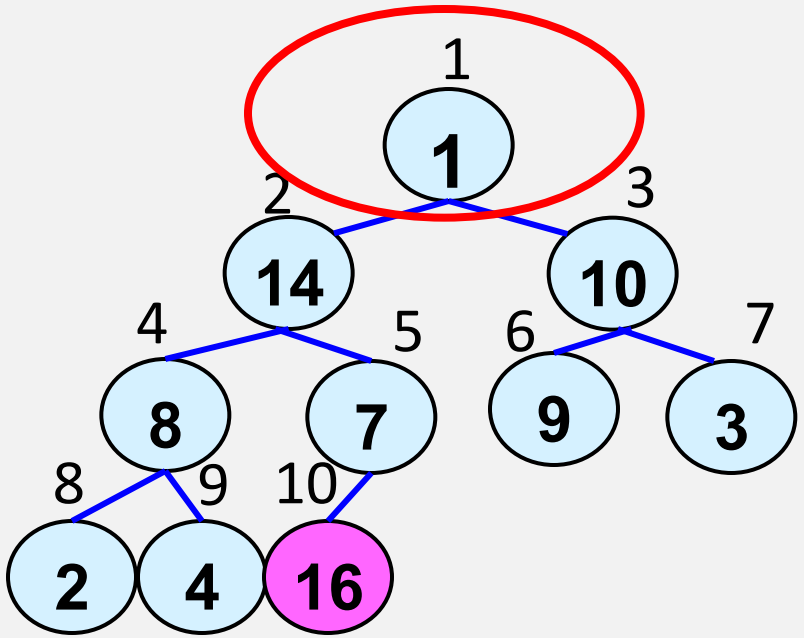
i=10



	1	2	3	4	5	6	7	8	9	10
X	1	14	10	8	7	9	3	2	4	16

```
Heapsort(A)
  Build_heap(A)
  for i = heapsize  down to 2
  do
      exchange (A[1], A[i])
      heapsize = heapsize -1;
       Heapify(A,1)
```

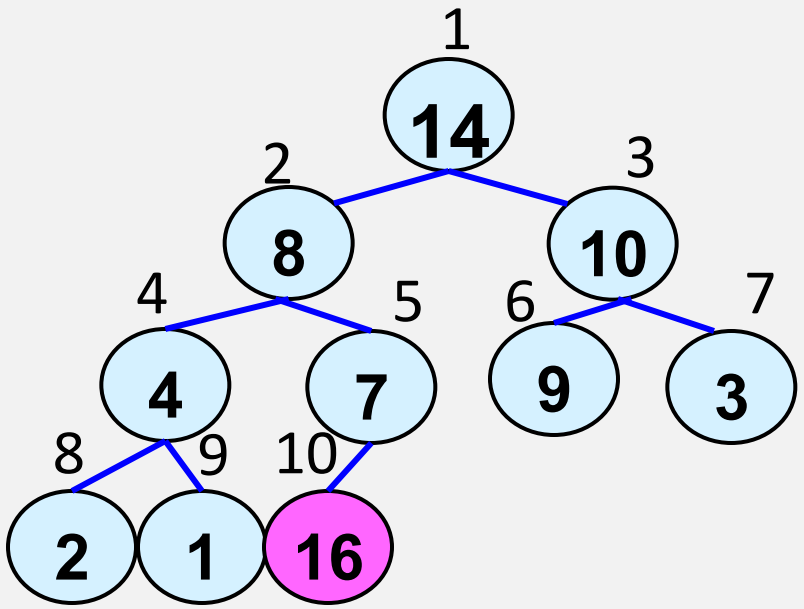
A[1-9]



	1	2	3	4	5	6	7	8	9	10
X	14	8	10	4	7	9	3	2	1	16

```
Heapsort(A)
  Build_heap(A)
  for i = length  down to 2
  do
     exchange (A[1], A[i])
     heapsize = heapsize -1;
      Heapify(A,1)
```

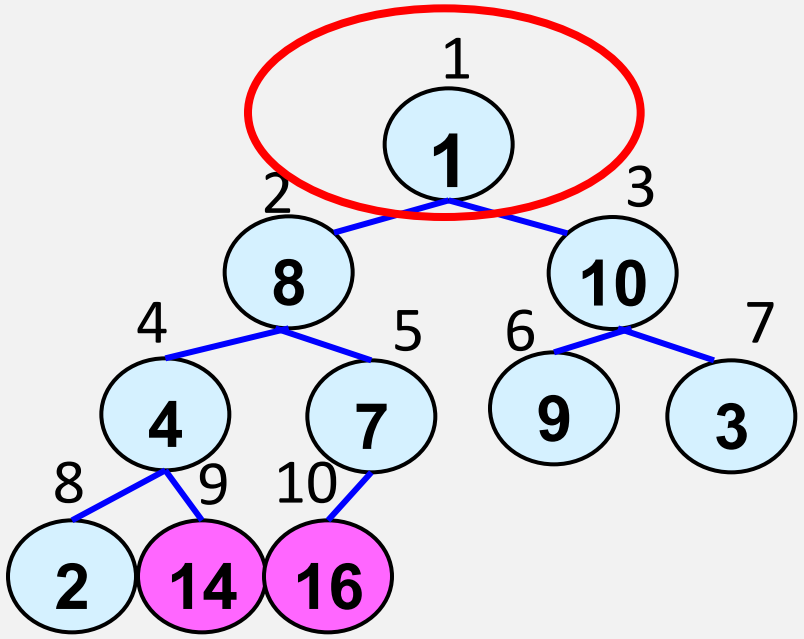
i=9



	1	2	3	4	5	6	7	8	9	10
X	1	8	10	4	7	9	3	2	14	16

```
Heapsort(A)
  Build_heap(A)
  for i = length  down to 2
  do
     exchange (A[1], A[i])
     heapsize = heapsize -1;
      Heapify(A,1)
```

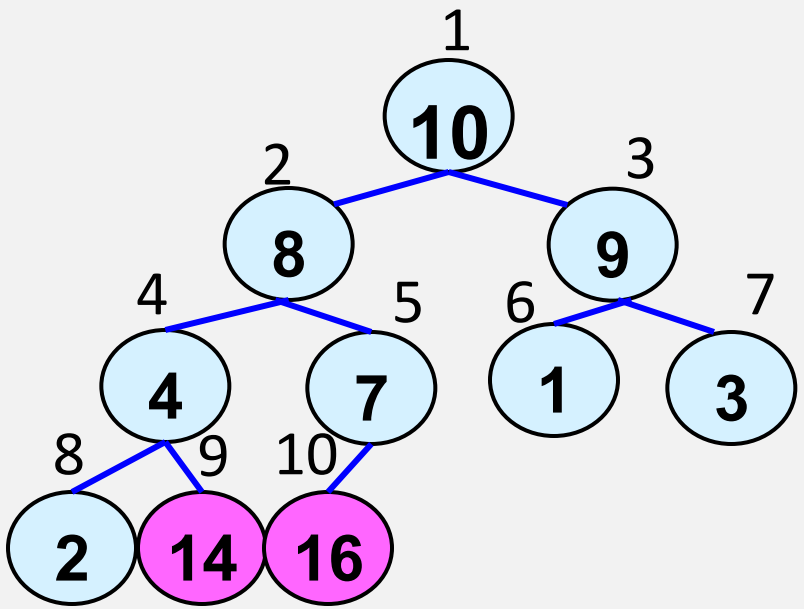
A[1-8]



	1	2	3	4	5	6	7	8	9	10
X	1	8	10	4	7	9	3	2	14	16

```
Heapsort(A)
  Build_heap(A)
  for i = length  down to 2
  do
      exchange (A[1], A[i])
      heapsize = heapsize -1;
       Heapify(A,1)
```

i=8



5.1.5 Priority queues

Priority queues : is a data structure for maintaining a set S of elements, each with a associated value called a key. A priority supports the following operations.

1) Insert(S, x) : insert the element x into the set S. This operation could be written as S <- S U { x }

2) Maximum(S) : returns the elements of S with the largest key;

3) Extract_Max(S) : return the elements of S with the largest key.

Priority queues operators :

1. Insert O(logn)
2. Maximum O(1)
3. Extract_Max ??

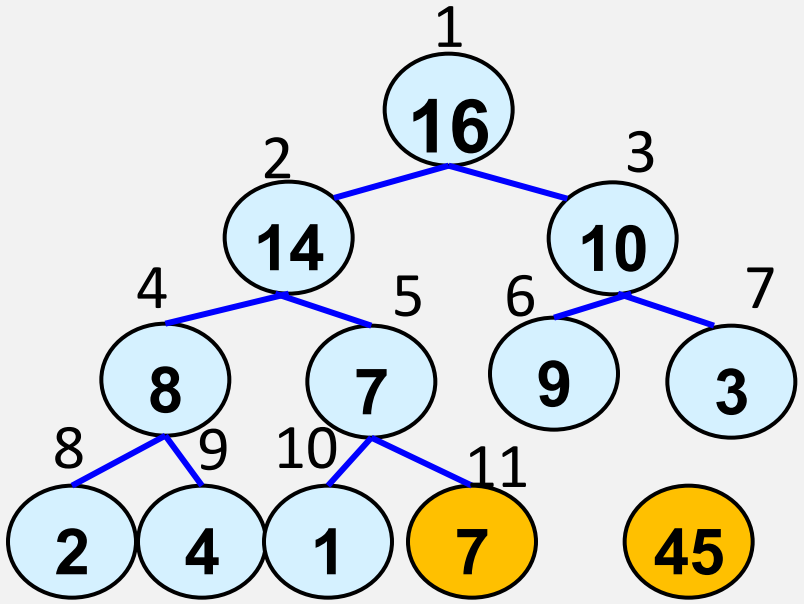
	1	2	3	4	5	6	7	8	9	10	11
	X	16	14	10	8	7	9	3	2	4	1	7
45	X	16	14	10	8	45	9	3	2	4	1	7

Heap_Insert(A, key)

heapsize = heapsize+1

i = heapsize i=

```
while i >1&& A[parent(i)]<key
    A[i] = A[parent(i)]
    i = parent(i)   i=
   A[i] = key
```



	1	2	3	4	5	6	7	8	9	10	11
	X	16	45	10	8	14	9	3	2	4	1	7

Heap_Insert(A, key) 45

heapsize = heapsize+1

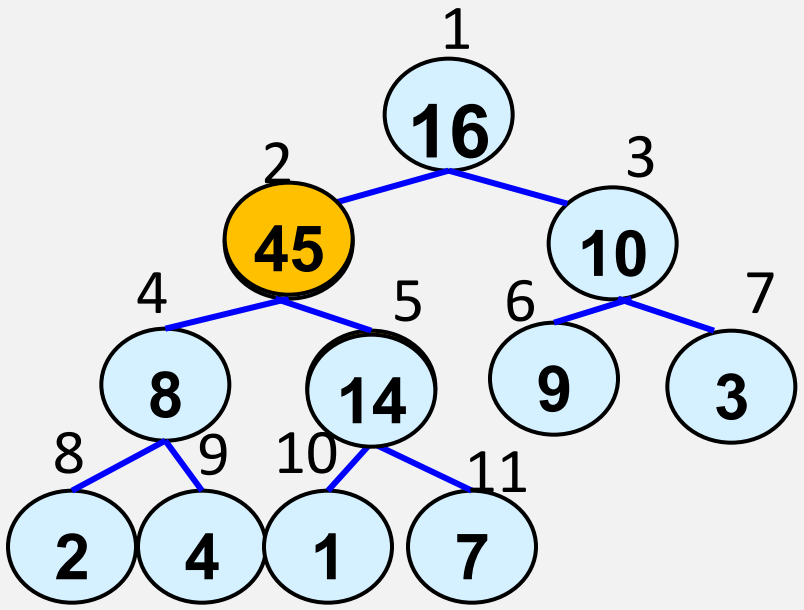
i = heapsize i=

while i >1&& A[parent(i)]<key

A[i] = A[parent(i)]

i = parent(i) i=

A[i] = key



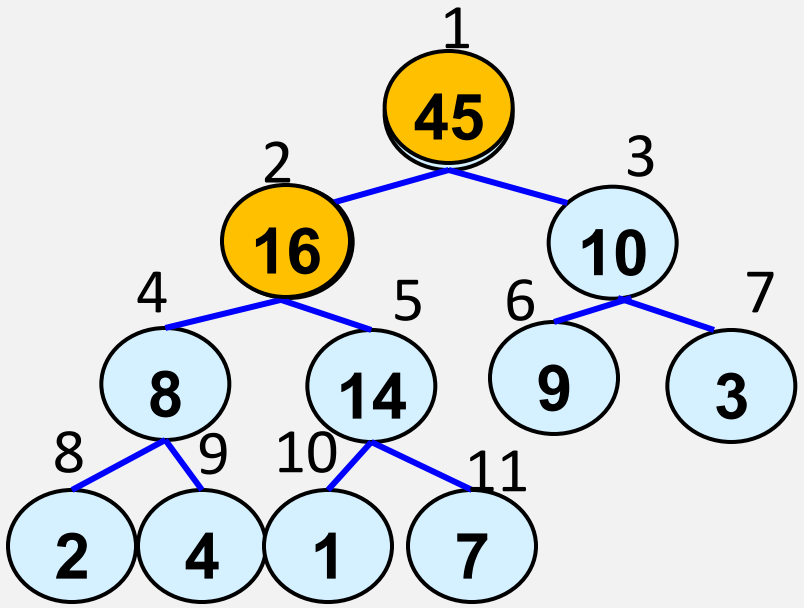
1	2	3	4	5	6	7	8	9	10	11
X	45	16	10	8	14	9	3	2	4	1	7

```
Heap_Insert(A, key)
heapsize = heapsize+1
i = heapsize
while i >1&& A[parent(i)]<key
    A[i] = A[parent(i)]
    i = parent(i)
   A[i] = key
```

45

i=

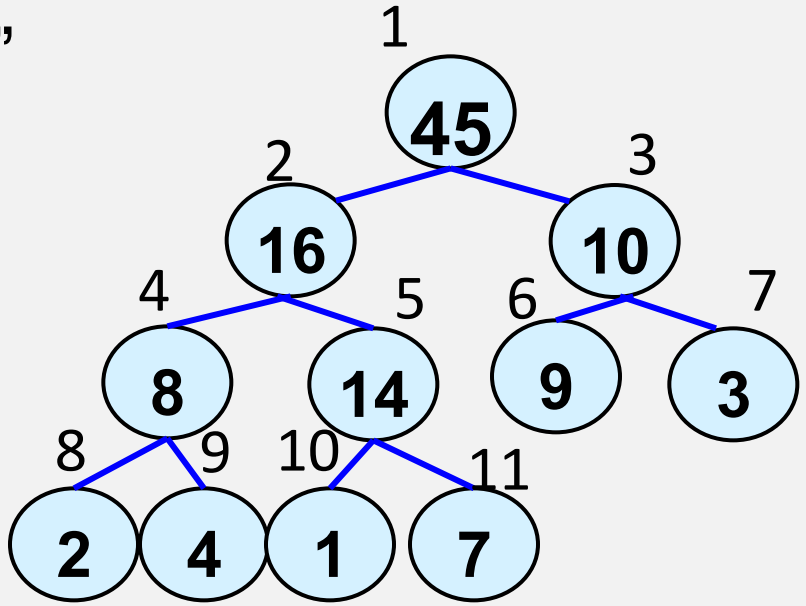
i=



```
Heap_Exact_Max(A)
if heap_size < 1
    then error “Heap underflow”
max = A[1]
A[1] = A[heapsize]
heapsize = heapsize-1
Heapify(A,1)
return max
```

max=45

	1	2	3	4	5	6	7	8	9	10	11
X	45	16	10	8	14	9	3	2	4	1	7



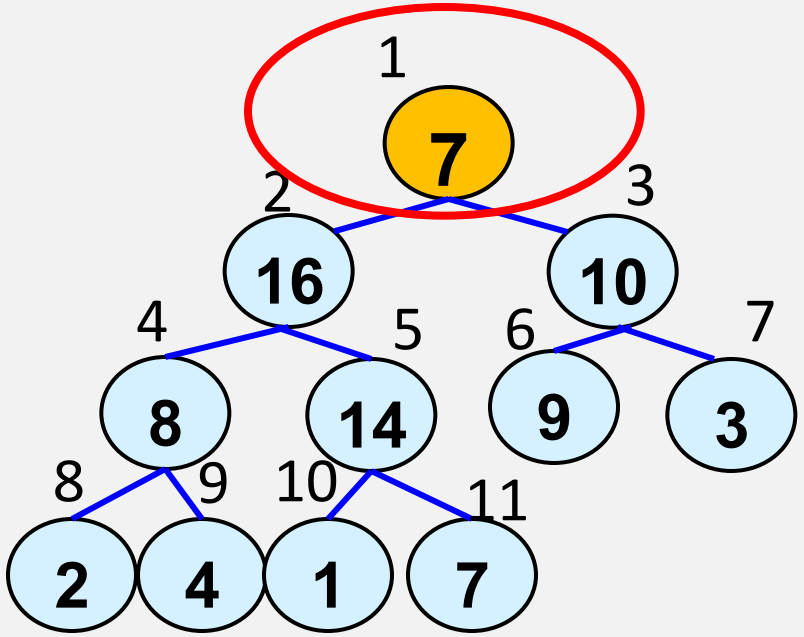
	1	2	3	4	5	6	7	8	9	10	11
	X	45	16	10	8	14	9	3	2	4	1	7

	X	7	16	10	8	14	9	3	2	4	1	7

```
Heap_Exact_Max(A)
if heap_size < 1
    then error “Heap underflow
max = A[1]
A[1] = A[heapsize]
heapsize = heapsize-1
Heapify(A,1)
return max
```

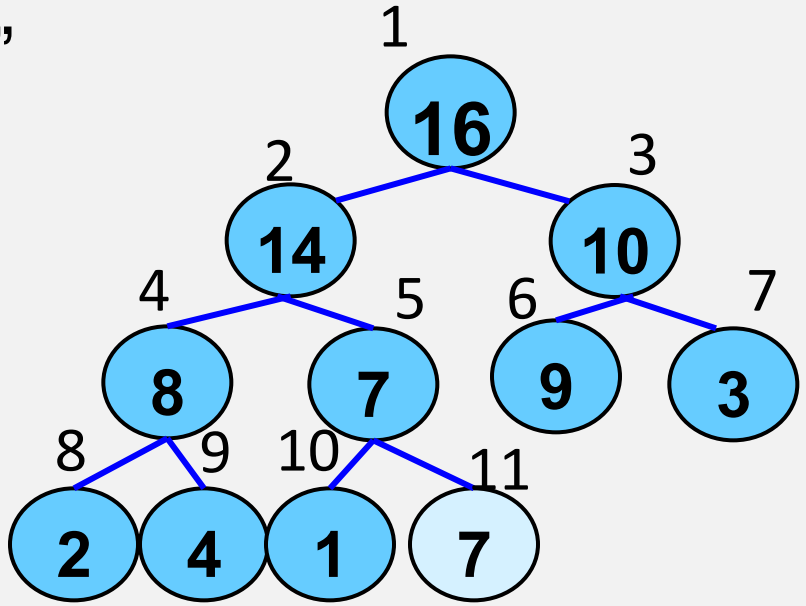
max=45

A[1-10]



```
Heap_Exact_Max(A)
if heap_size < 1
    then error “Heap underflow”
max = A[1]
A[1] = A[heapsize]
heapsize = heapsize-1
Heapify(A,1)
return max
```

	1	2	3	4	5	6	7	8	9	10	11
X	16	14	10	8	7	9	3	2	4	1	7



max=45