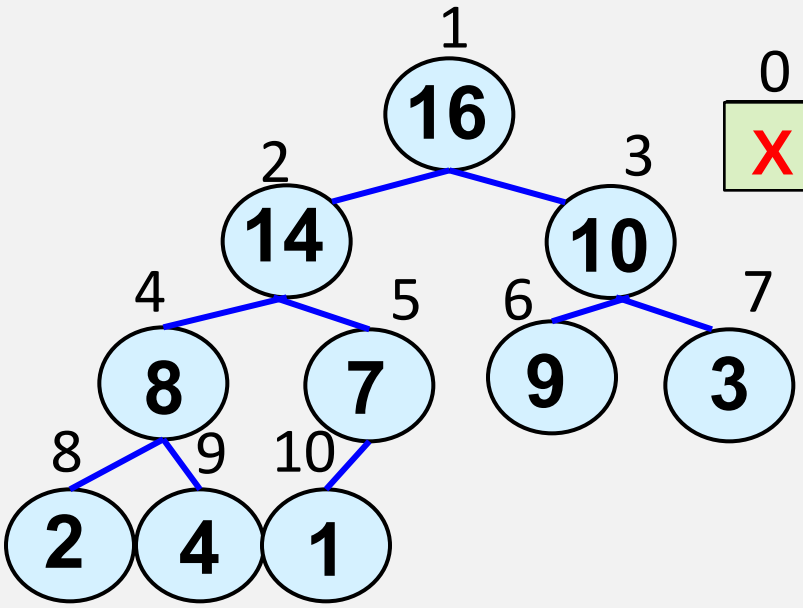
Module5—Heap

Heap : array ที่ view

5.1 Heap * กรอง

* โครงสร้างข้อมูล array ที่ถูกมองเป็น complete binary tree

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

1. ไม่เก็บข้อมูลที่ index 0

* มีคุณสมบัติเหมือน tree

A is an array represent heap

attribute

- int A[30];

30

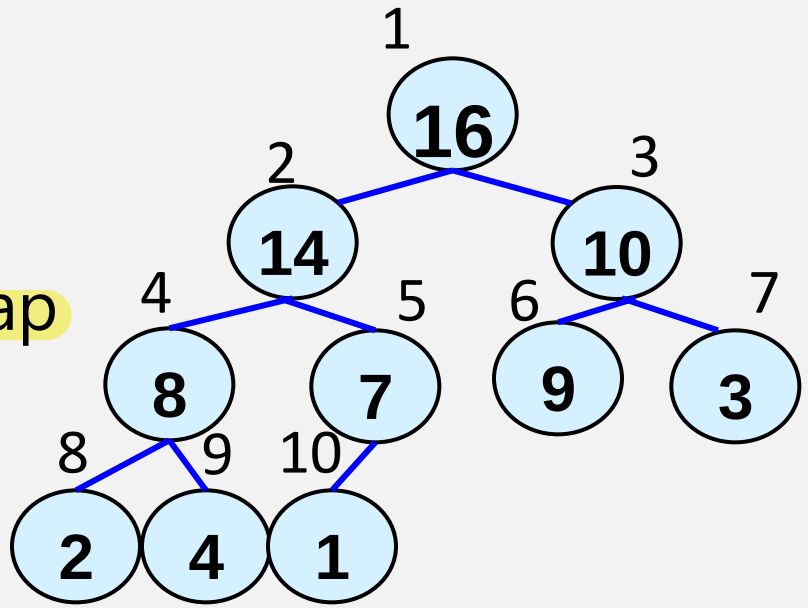
- length ขนาดของ A

10

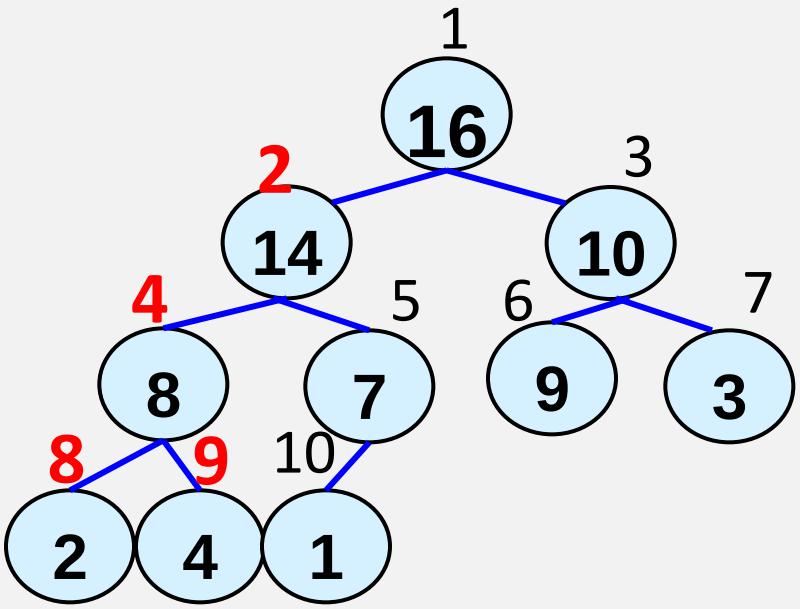
- heapsize จำนวนสมาชิกใน heap

- 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

คำถาม ถ้า i=3?

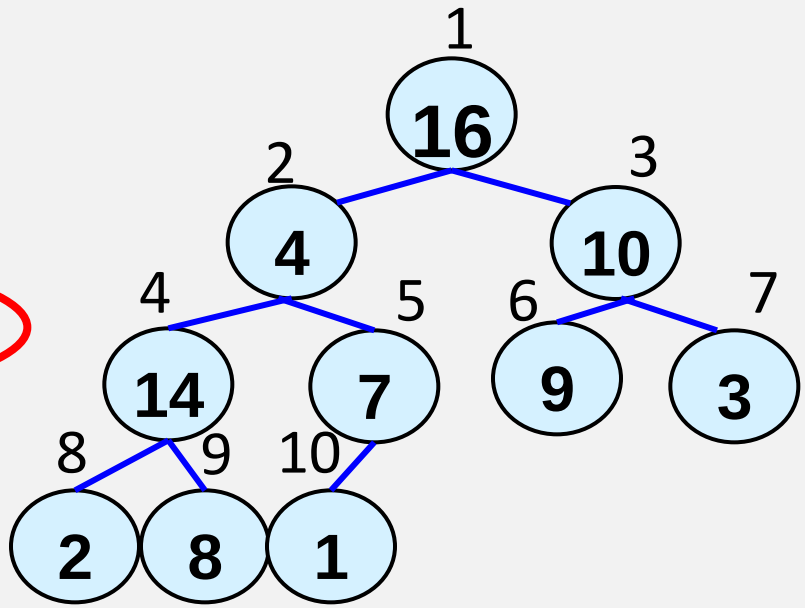
parents = 1 left = 6 right = 7

5.1.1 Heap property :

สำหรับทุกโหนด i ที่ไม่ใช่ root

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

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



ตัวใดที่ขาดคุณสมบัติ

Ans 2

binary search มีคุณสมบัติ left ≤ root ≤ right

heap มีคุณสมบัติ left ≤ root ≥ right

} heap ≠ binary search

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

6

5.1.2 Maintaining t eap property

4

เปลี่ยน array → heap

```
Heapify(A,i)
  l=left(i)
  r=right(i)
```

5

14 > 4

เช็คพ่อแม่ กับ ลูก

```
  if l <= heapsize and A[l] > A[i]
        then   largest = l
        else   largest = i
```

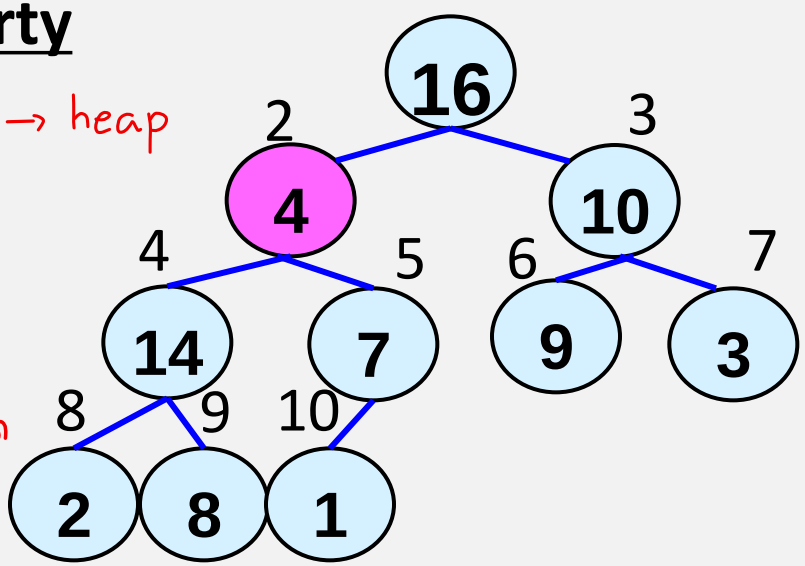
เก็บ index ที่ใหญ่ที่สุด

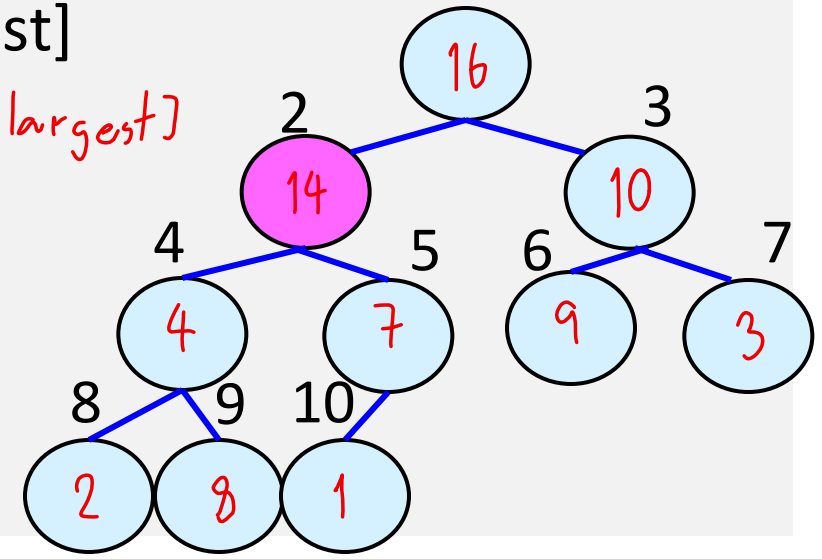
7 > 14

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

key ขวา กับ key[largest]

4, 14





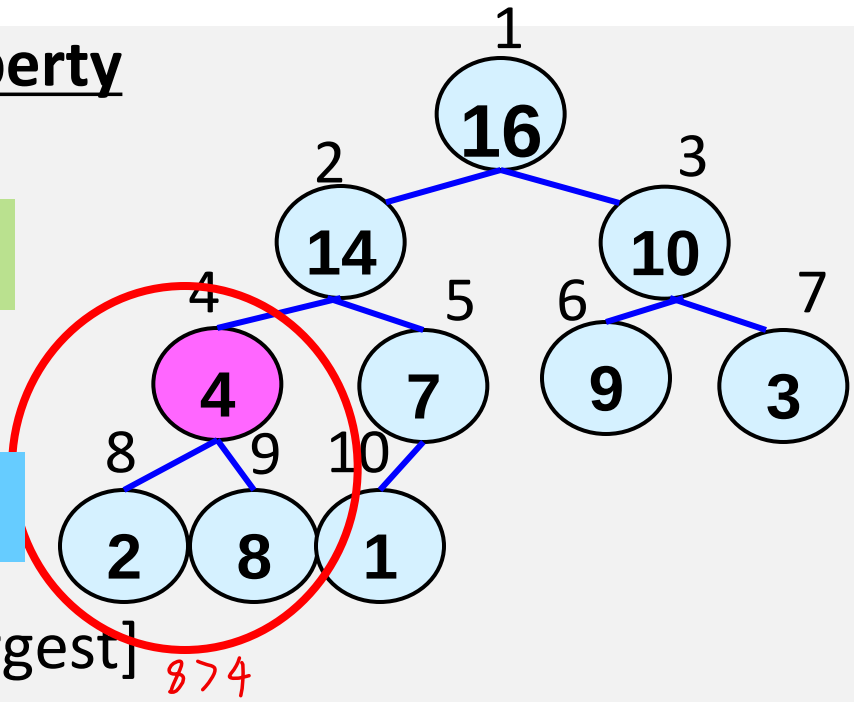
5.1.2 Maintaining the heap property

```
Heapify(A,i)   i = 4
  l=left(i)    l = 8
  r=right(i)   r = 9
  if l <= heapsize and  A[l] > A[i]   F
        then    largest = l
        else    largest = i          4
  if r <= heapsize and  A[r] > A[largest]
        then largest =r              9
  if largest != i   T
        then
            exchange(A[i],A[largest])
            Heapify(A,largest)
```

2 > 4

8 > 4

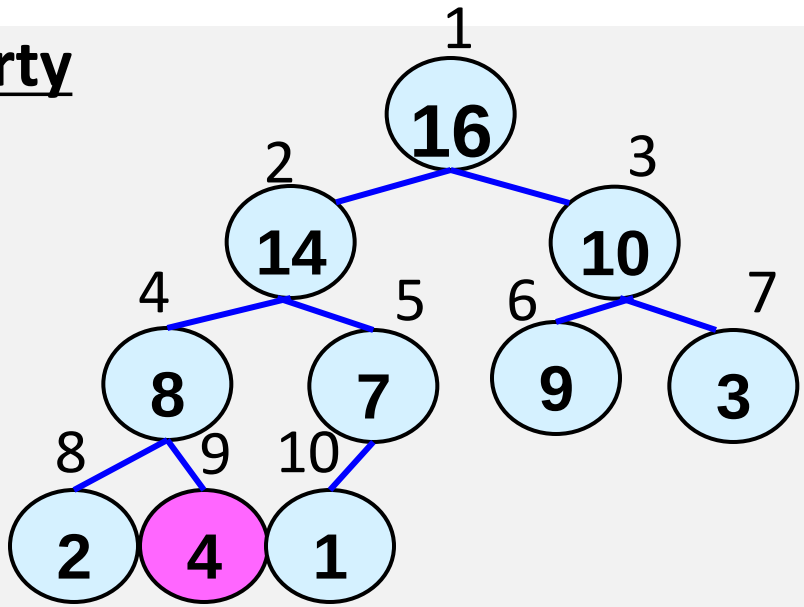
สลับ ตำแหน่ง

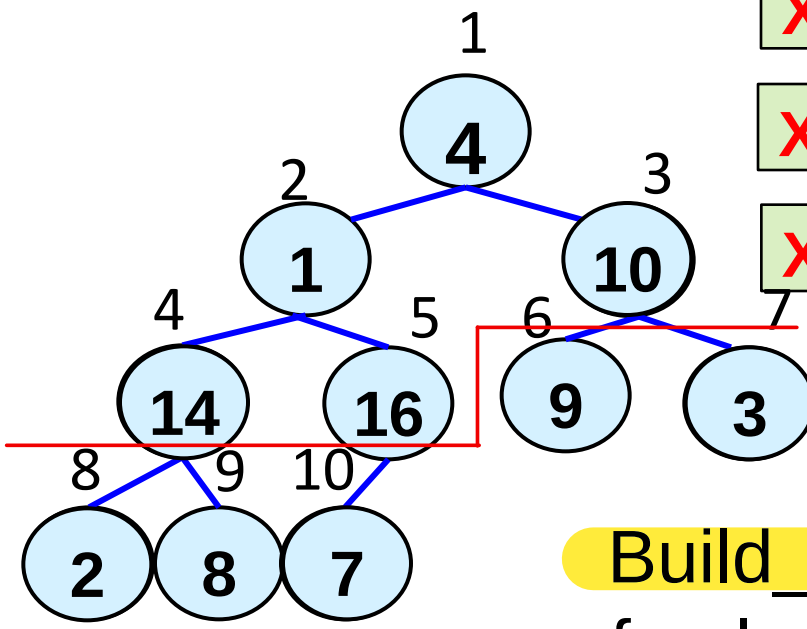


5.1.2 Maintaining the heap property

Heapify(A,i) Big O(log N)

```
l=left(i)  2i
r=right(i)  2i + 1
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)
```

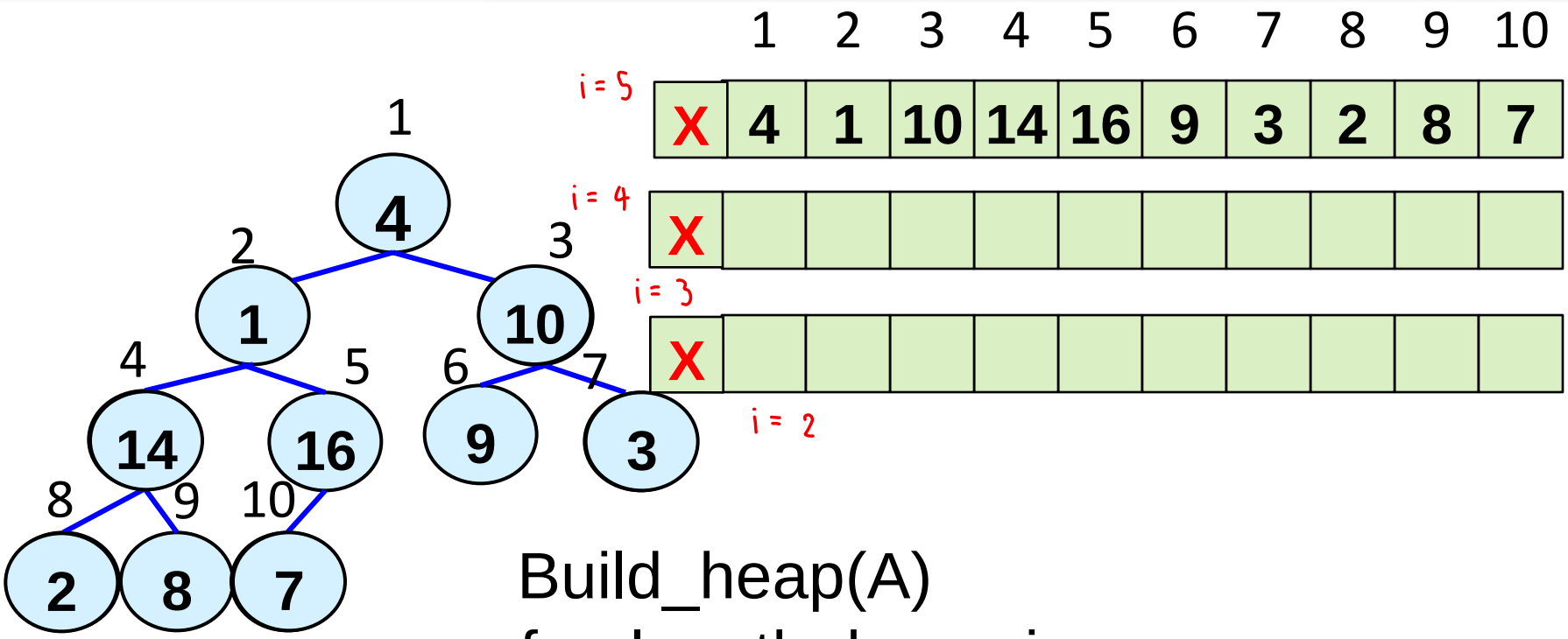




	0	1	2	3	4	5	6	7	8	9	10
	X	4	1	3	2	16	9	10	14	8	7
	X	4	1	3	14	16	9	10	2	8	7
	X	4	1	10	14	16	9	3	2	8	7

Home Work

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




```
Build_heap(A)
{     length=heapsize
      for(i= length/2 downto 1)
            Heapify(A,i)  5 จนถึง 1
}
```

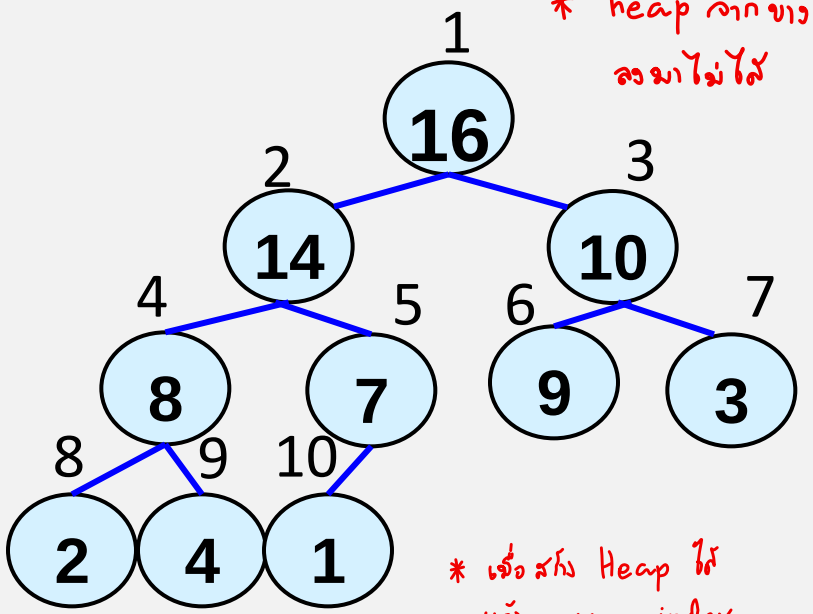
5.1.4 Heap sort :

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

* heap จากข้างบน ลงมาไม่ได้

```
Heapsort(A)
  Build_heap(A)        i=10
  for i = length  down to 2
  do
      exchange (A[1], A[i])
      heapsize = heapsize -1;
       Heapify(A,1)
```

สลับ หัว กับ ท้าย



* เมื่อสร้าง Heap ได้แล้ว array index ที่ 1 จะมากที่สุด ต้อง จริง! เพราะ root จะมีค่ามากที่สุด

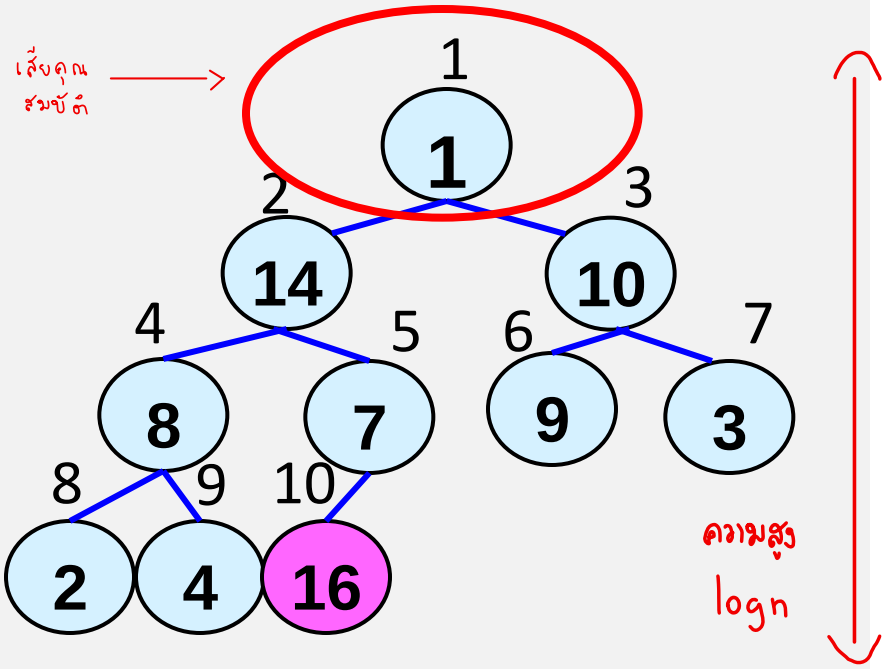
* แต่ index สองสุดท้าย น้องสุดจริง มีผล ต้อง ไม่จริง! เพราะ root ต้องมากกว่า child

5.1.4 Heap sort : Big O(nlogn)

	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)        10 ถึง 2
  for i = length  down to 2
  do
     exchange (A[1], A[i])
     heapsize = heapsize -1;   10 - 1
     Heapify(A,1)
```

A[1-9]



5.1.4 Heap sort :

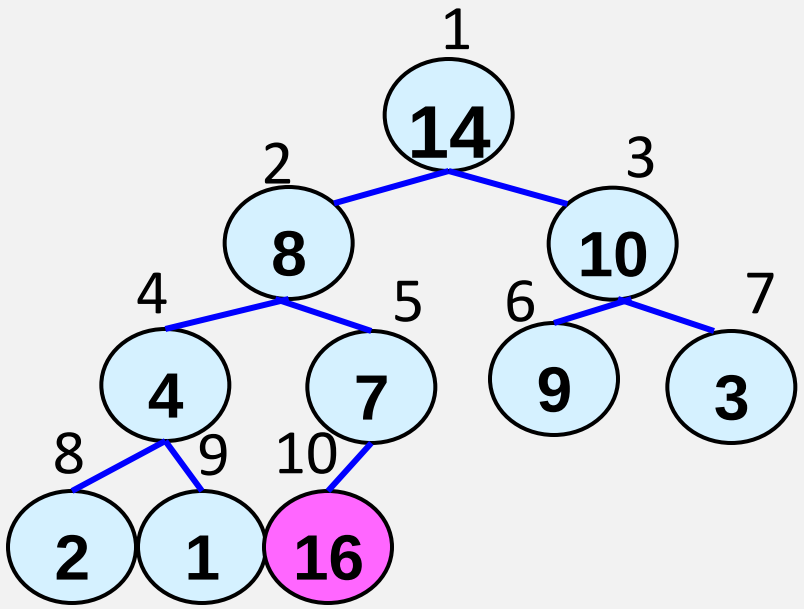
	1	2	3	4	5	6	7	8	9	10
X	14	8	10	4	7	9	3	2	1	16

```
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

9 -> 2

A[1] A[9]



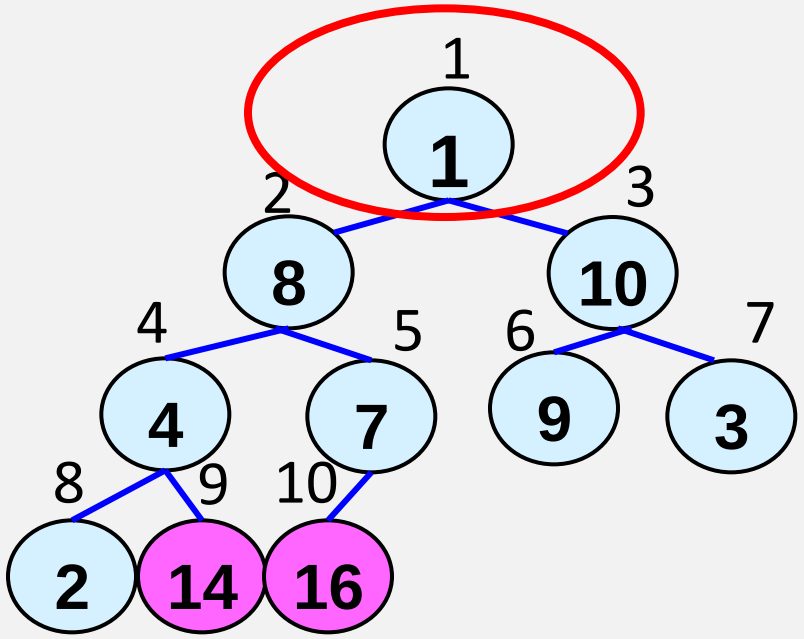
5.1.4 Heap sort :

	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)
```

9 - 1

A[1-8]

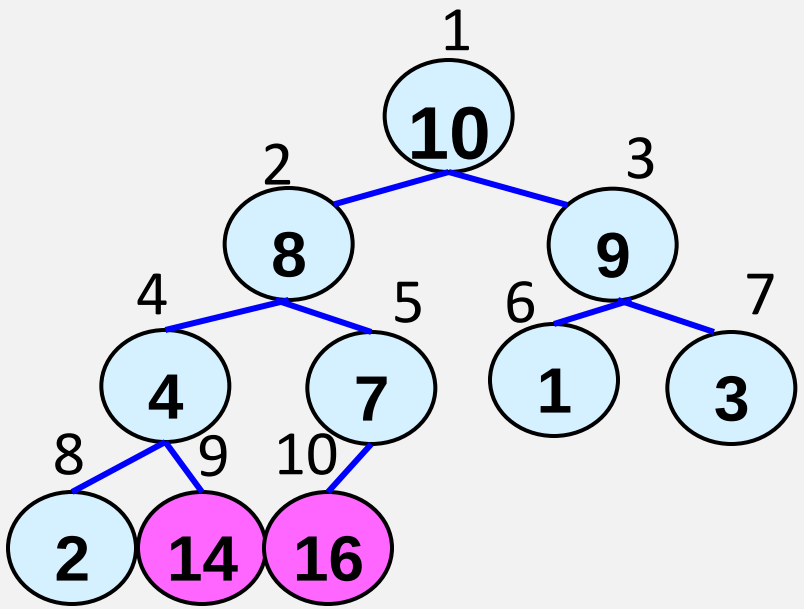


5.1.4 Heap sort :

	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 อภิสิทธิ์ (สิทธิพิเศษ)

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 } Big O(log n)

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.

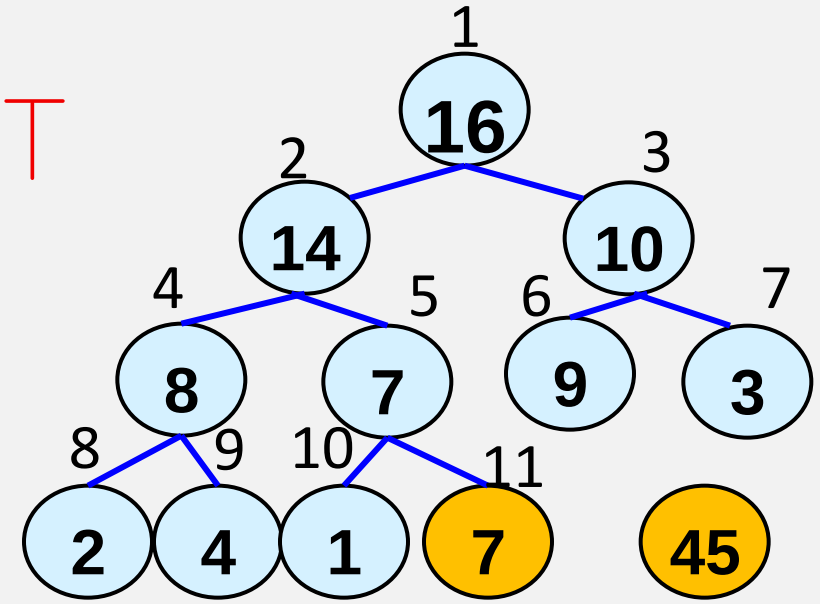
	1	2	3	4	5	6	7	8	9	10	11
	X	16	14	10	8	7	9	3	2	4	1	7
45	X	16	14	10	8	45	9	3	2	4	1	7

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

i=11

7<45

i=5

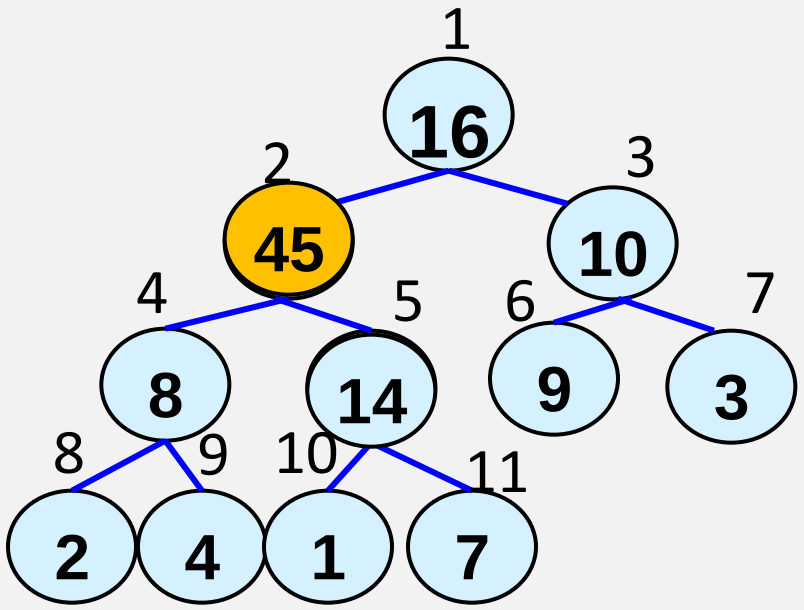


1	2	3	4	5	6	7	8	9	10	11
X	16	45	10	8	14	9	3	2	4	1	7

Heap_Insert(A, key) **45**
heapsize = heapsize+1
i = heapsize i= <

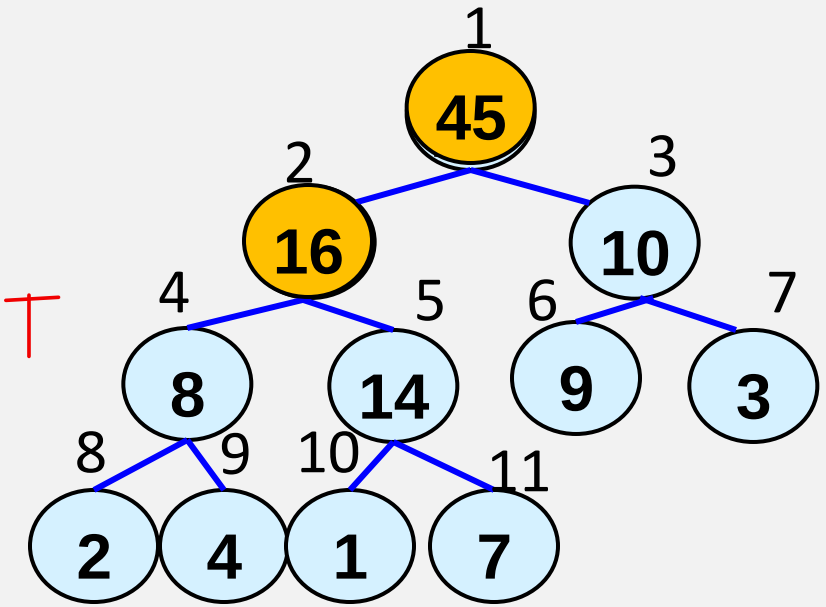
```
while i >1&& A[parent(i)]<key
    A[i] = A[parent(i)]
    i = parent(i)
   A[i] = key
```

i=



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

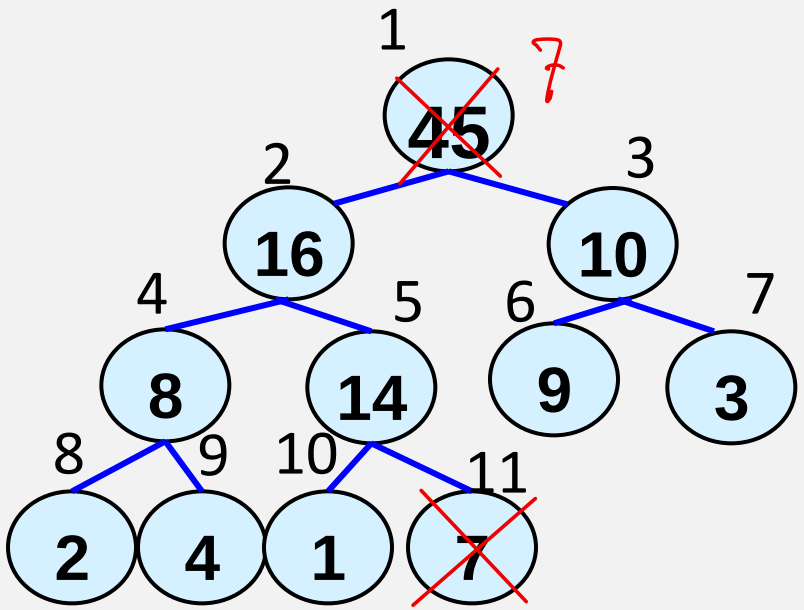
```
Heap_Insert(A, key)        45
heapsize = heapsize+1
i = heapsize               i=2        16<45
while i >1&& A[parent(i)]<key
    A[i] = A[parent(i)]    value สลับ ข่อแม่
    i = parent(i)          i=1
    A[i] = key
```



Heap_Exact_Max(A)
if heap_size < 1 ดึงข้อมูลออก
then error “Heap underflow”
max = A[1] max= 45 21
A[1] = A[heapsize] 7 13
heapsize = heapsize-1 12
Heapify(A,1)
return max

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

13



Big O (log n)

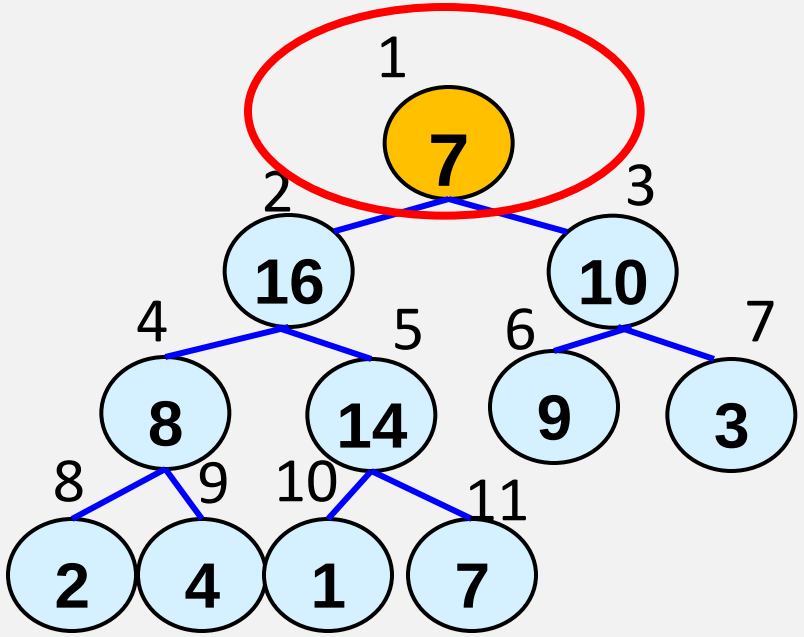
	1	2	3	4	5	6	7	8	9	10	11
X	45	16	10	8	14	9	3	2	4	1	7

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
```

max=45

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