Ch11: Heap and Heap Sort

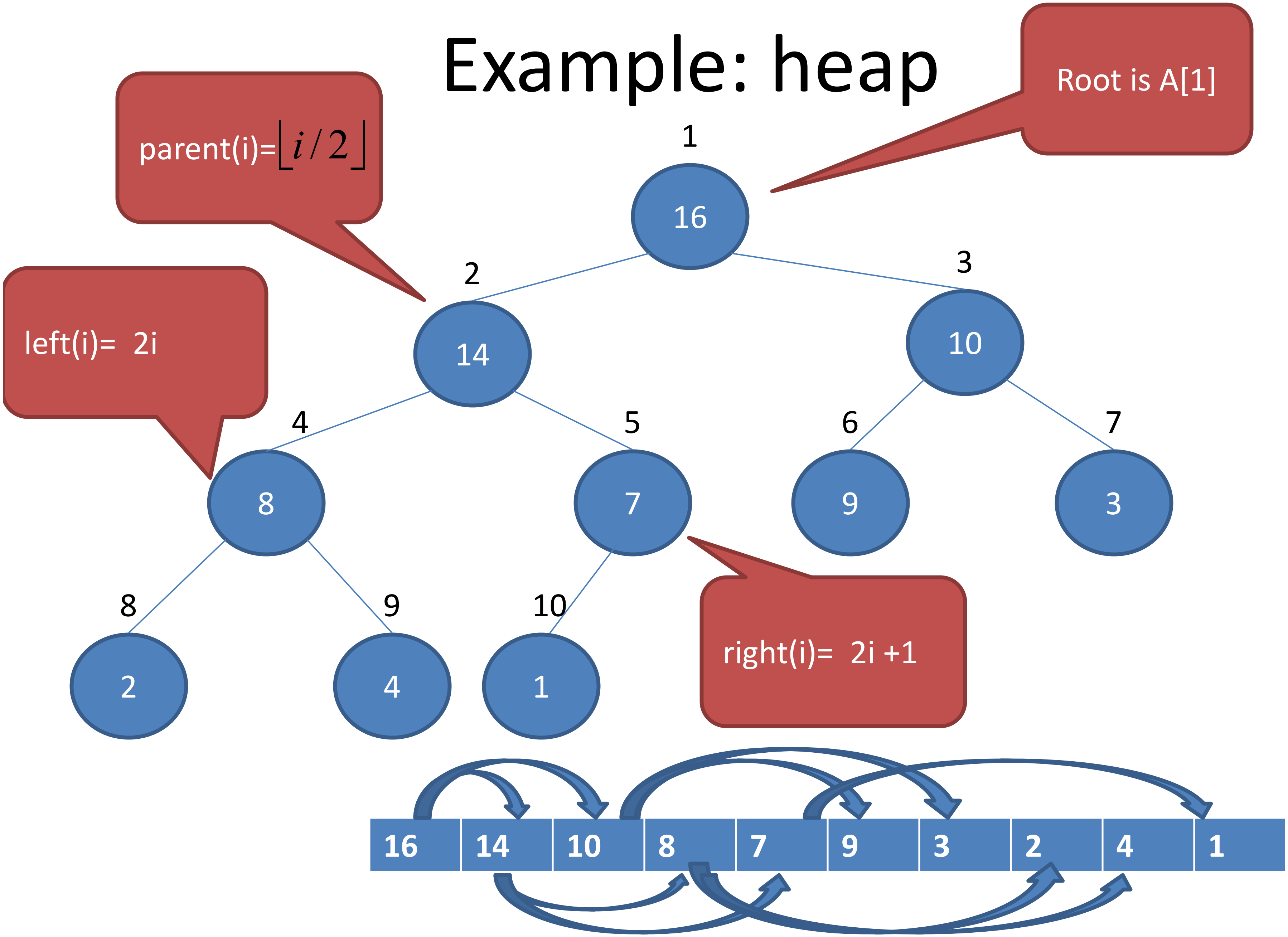
305234
Algorithm Analysis and Design

Jiraporn Pooksook
Naresuan University

What is the (binary) heap?

- The (binary) heap data structure is an array object that can be viewed as a nearly complete binary tree. Each node of the tree corresponds to an element of the array that stores the value in the node.
- An array A that represents a heap is an object with two attributes:
  – length[A] is the number of elements in the array
  – heap-size[A] is the number of elements in the heap stored within array A.

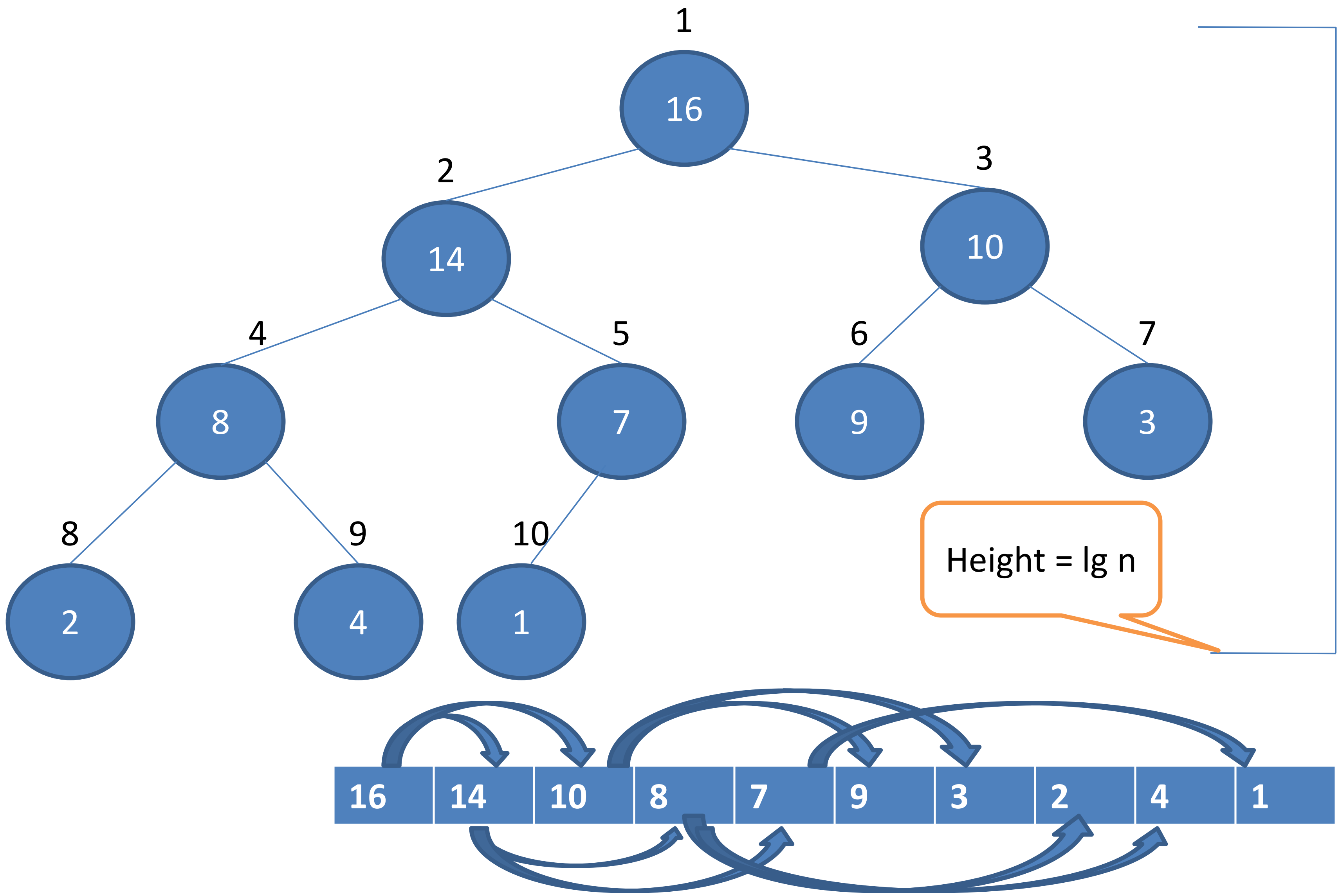
Example: heap



Binary Heap

- There are two kinds of binary heaps:
- Max-heaps
  - For every node i other than the root, A[parent(i)] $\geq$ A[i]
- Min-heaps
  - For every node i other than the root, A[parent(i)] $\leq$ A[i]

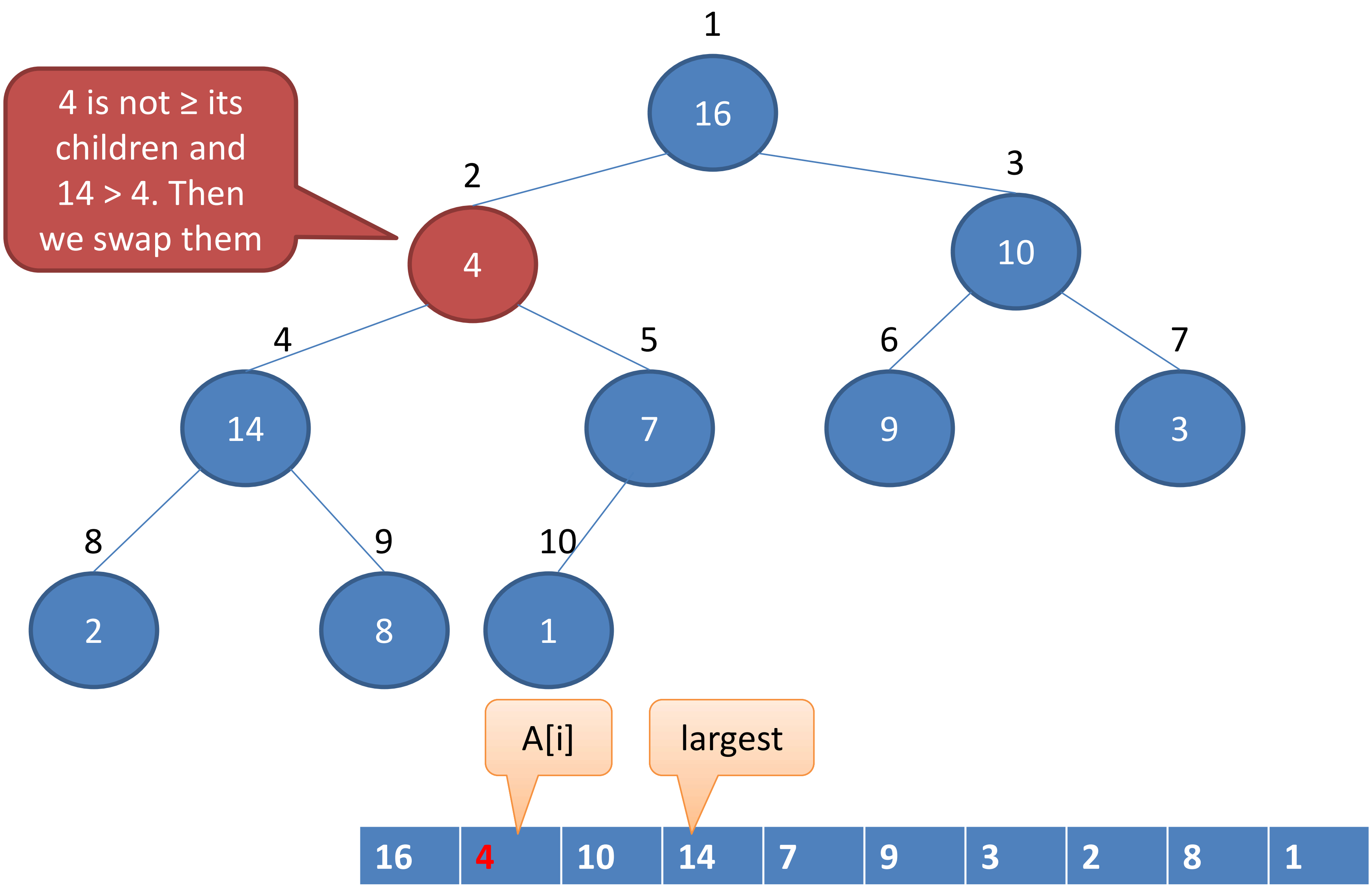
Example: Max-heap



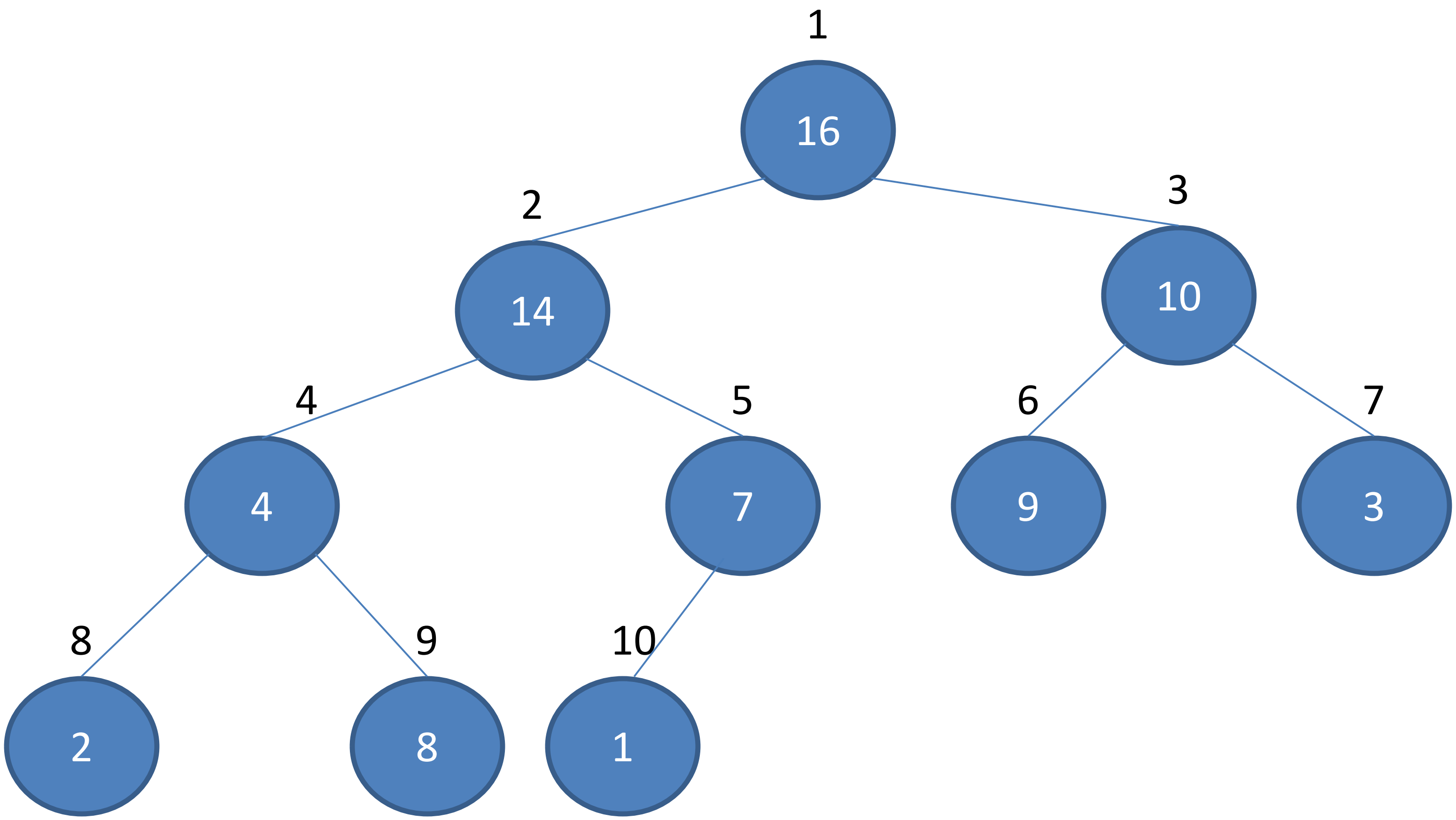
Max-Heapify(A,i)

```
l = left(i)
r = right(i)
if l ≤ heap-size[A] and A[l] > A[i]
        then largest = l
else    largest = i

if  r ≤ heap-size[A]  and A[r] > A[largest]
        then largest = r

If largest != i
        then exchanged A[i] and A[largest]
             Max-Heapify(A,largest)
```

Example: Max-Heapify(A,2)

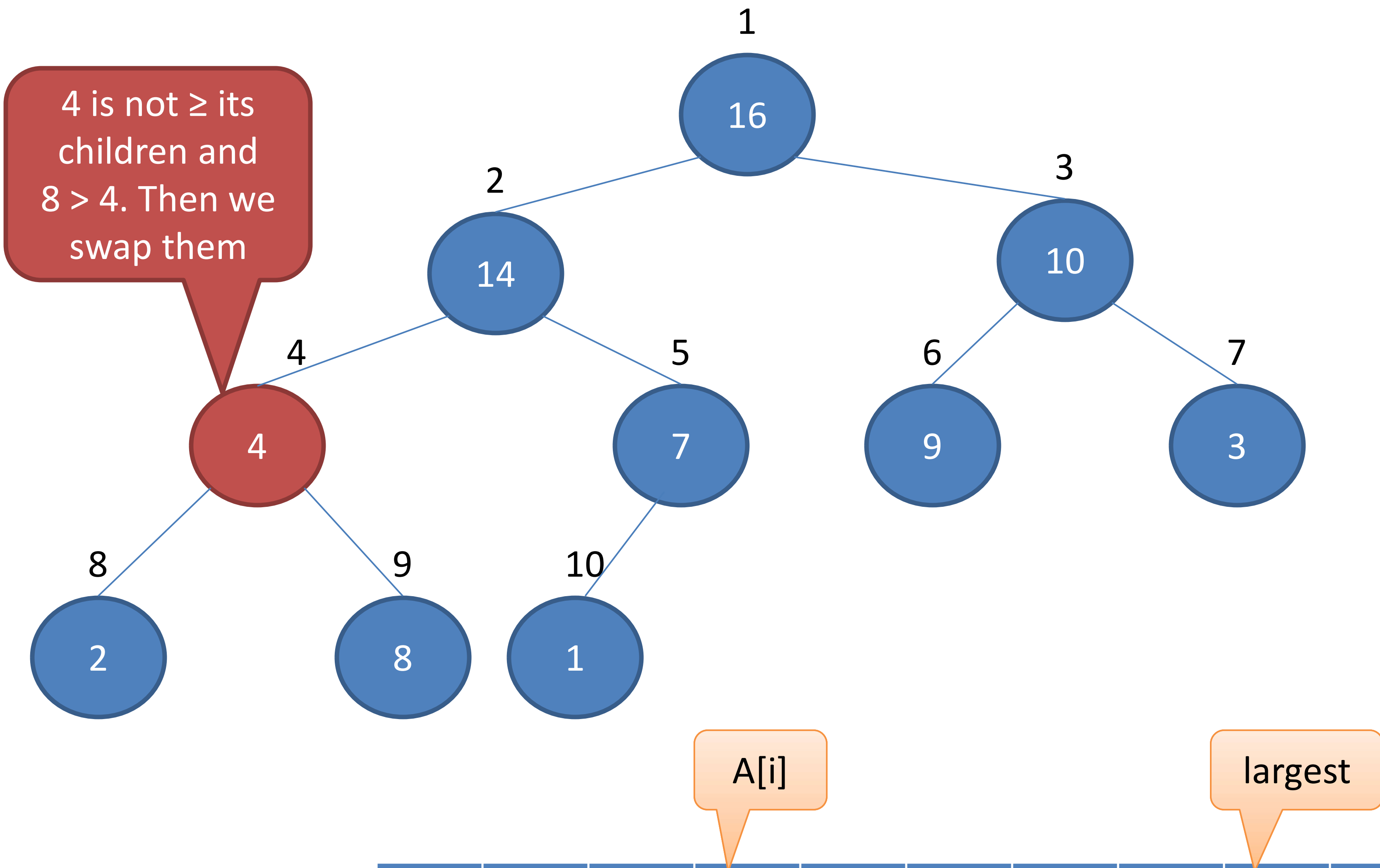


Example: Max-Heapify(A,2)



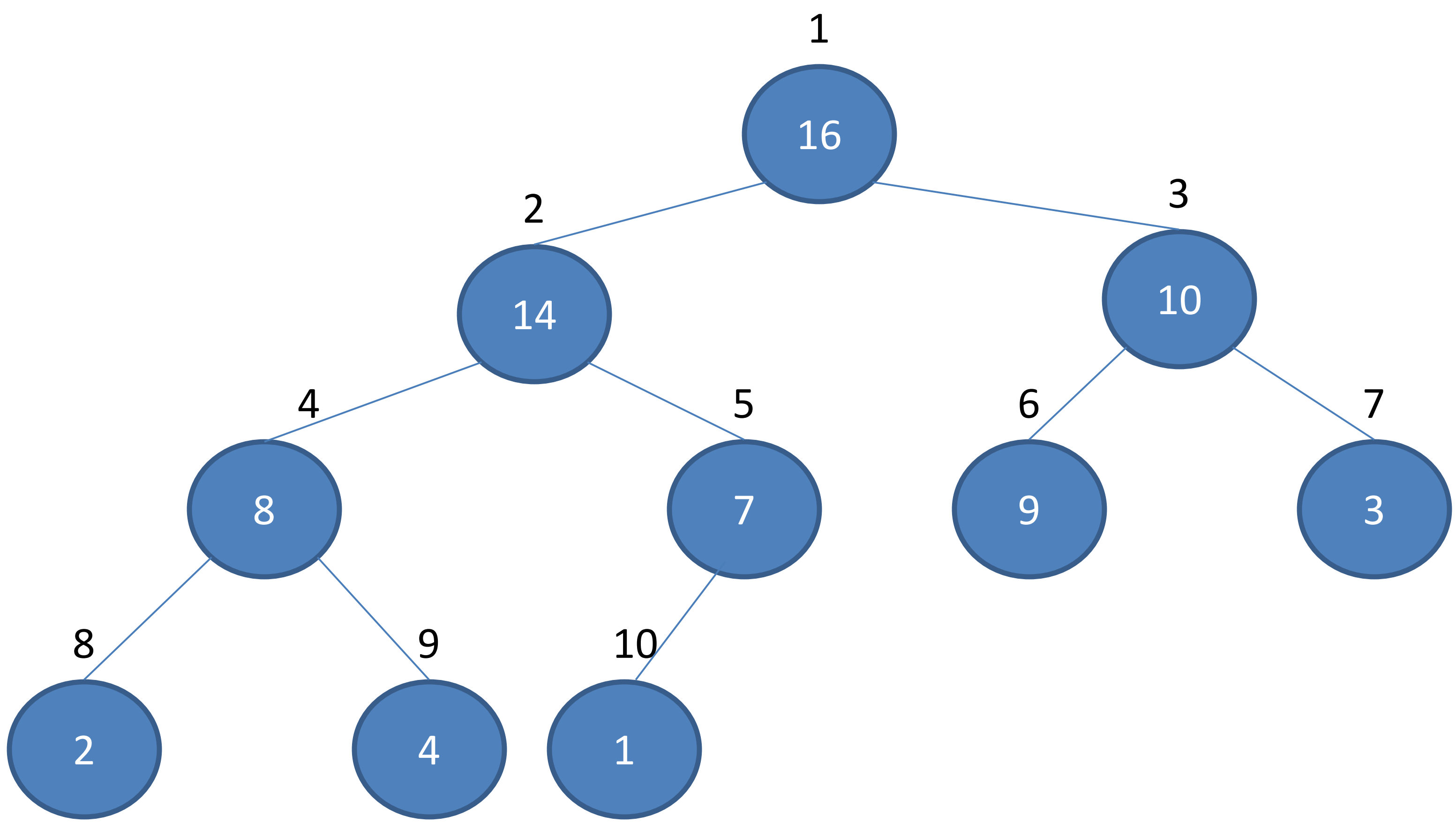
16	**14**	**10**	**4**	**7**	**9**	**3**	**2**	**8**	**1**

Example: Max-Heapify(A,4)



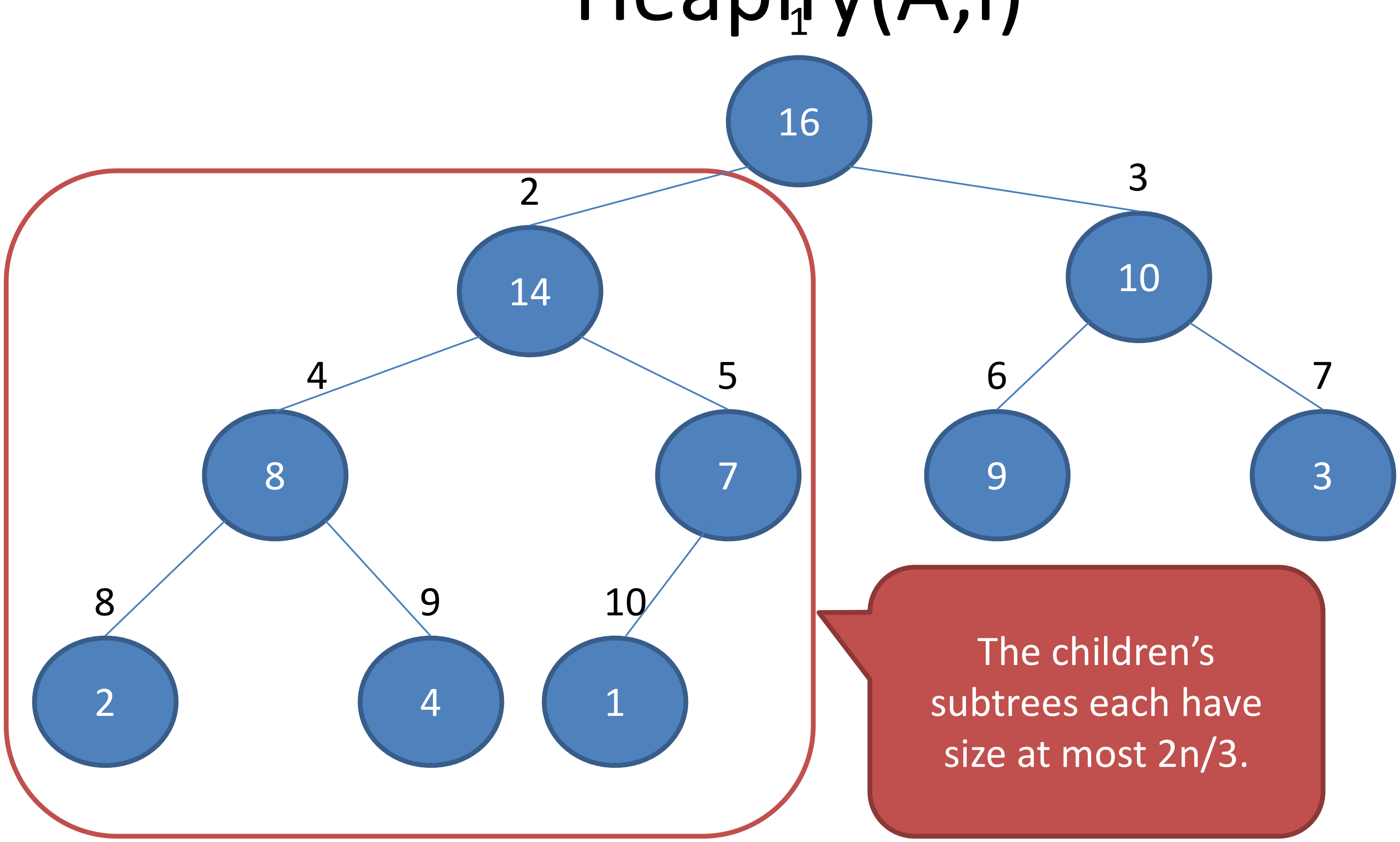
16	14	10	4	7	9	3	2	8	1

Example: Max-Heapify(A,4)



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

Analyze: Running time of Max-Heapify(A,i)



The children's subtrees each have size at most 2n/3.

$$T(n) \leq T(2n/3) + \Theta(1)$$

Analyze: Running time of Max-Heapify(A,i)

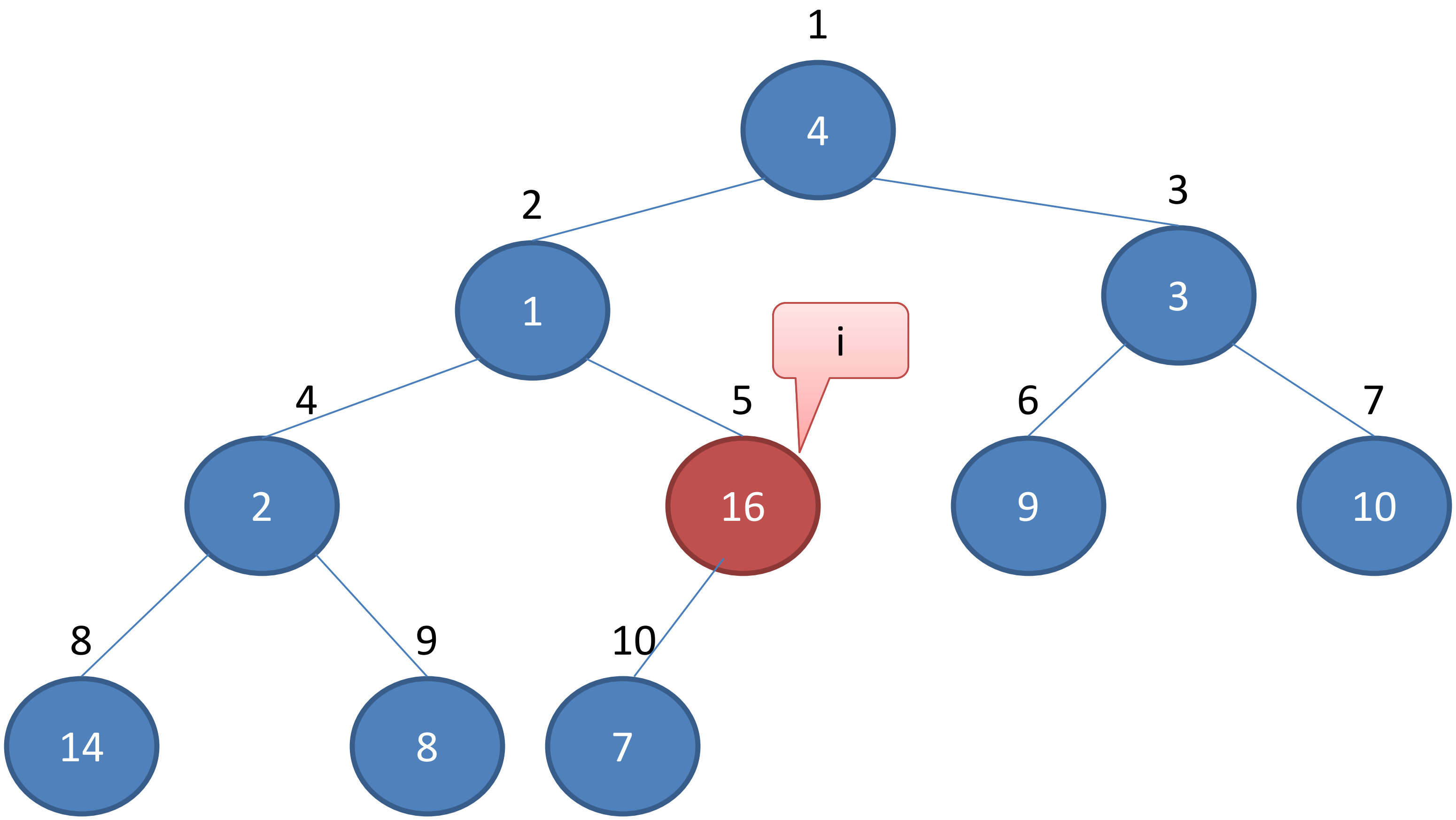
- We have $T(n) = T(2n/3) + 1$
- Determine which case of the master theorem applies:
- We have a=1, b=3/2, f(n)= 1
- Thus we have $n^{\log_b a} = n^{\log_{3/2} 1} = n^0 = 1$

  ข้อสังเกต 1 = n^0 , case 2
- Since $f(n) = \Theta(n^0) = \Theta(1)$ we can apply case 2 of the master theorem and conclude that the solution is $T(n) = \Theta(n^{\log_b a} \lg n) = \Theta(\lg n)$

Build-Max-Heap(A)

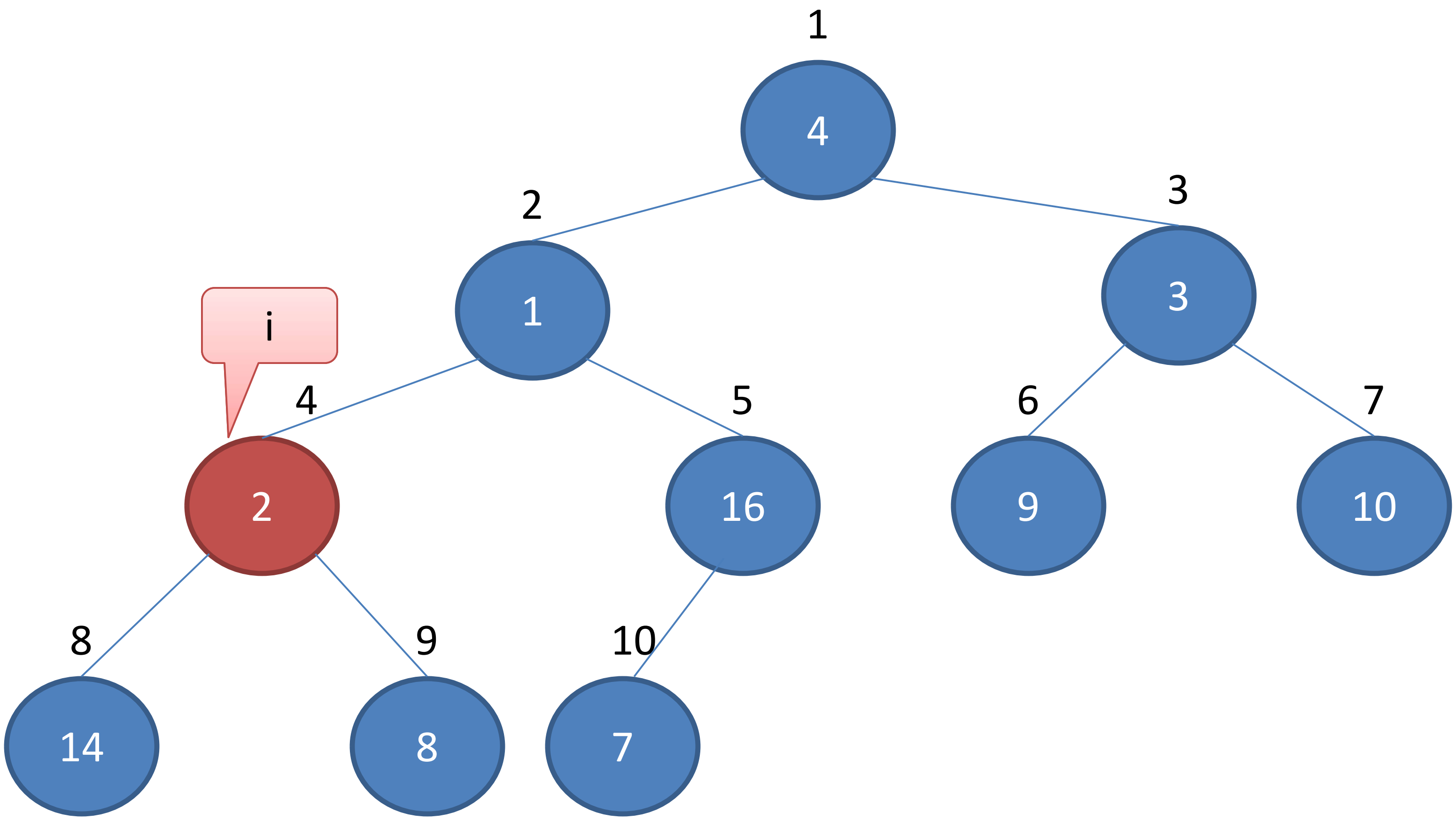
```
heap-size[A] = length[A]
for i = ⌊length[A]/2⌋ downto 1
        do Max-Heapify(A,i)
```

Example: Build-Max-Heap(A)



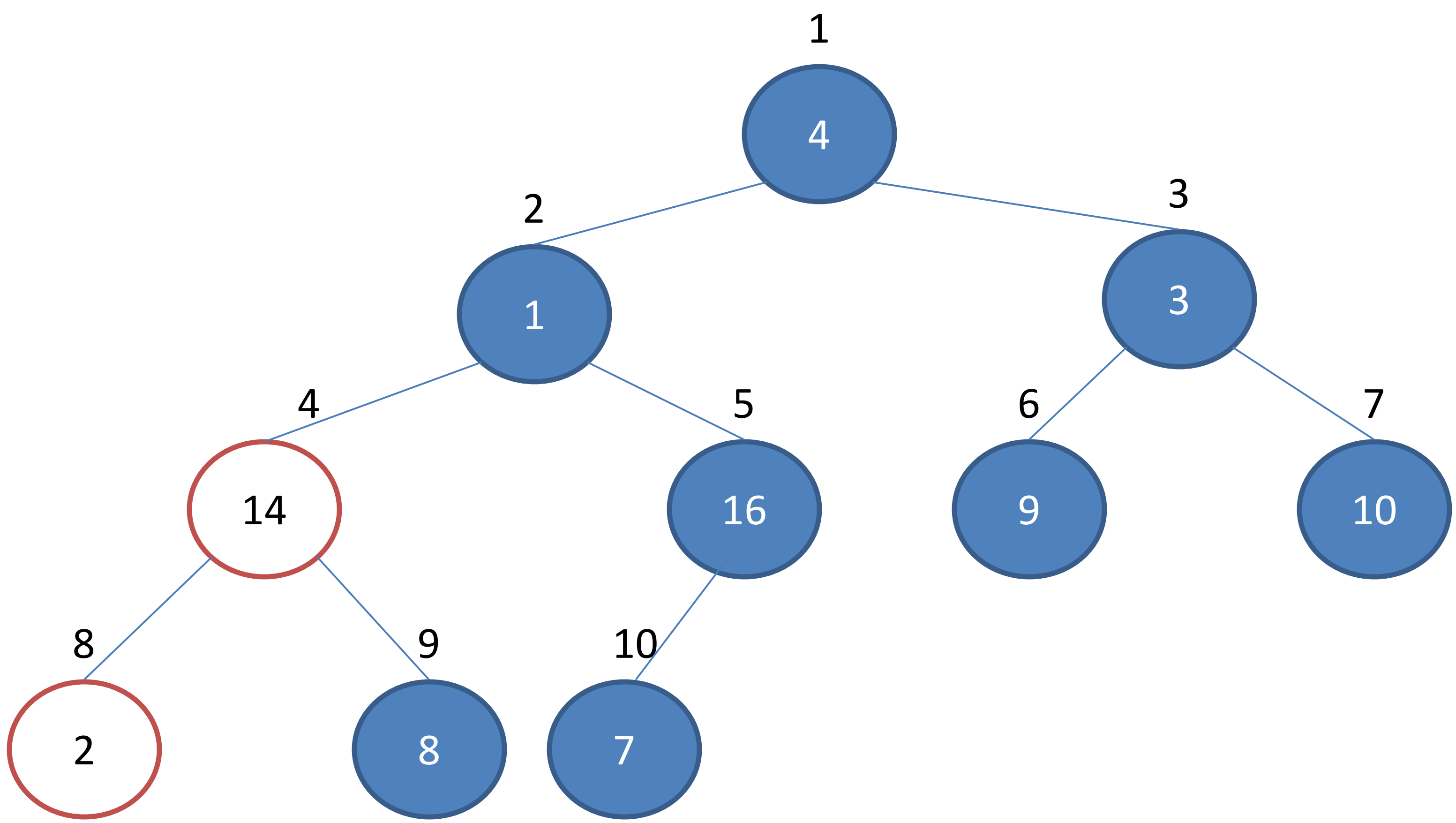
4	**1**	**3**	**2**	**16**	**9**	**10**	**14**	**8**	**7**

Example: Build-Max-Heap(A)



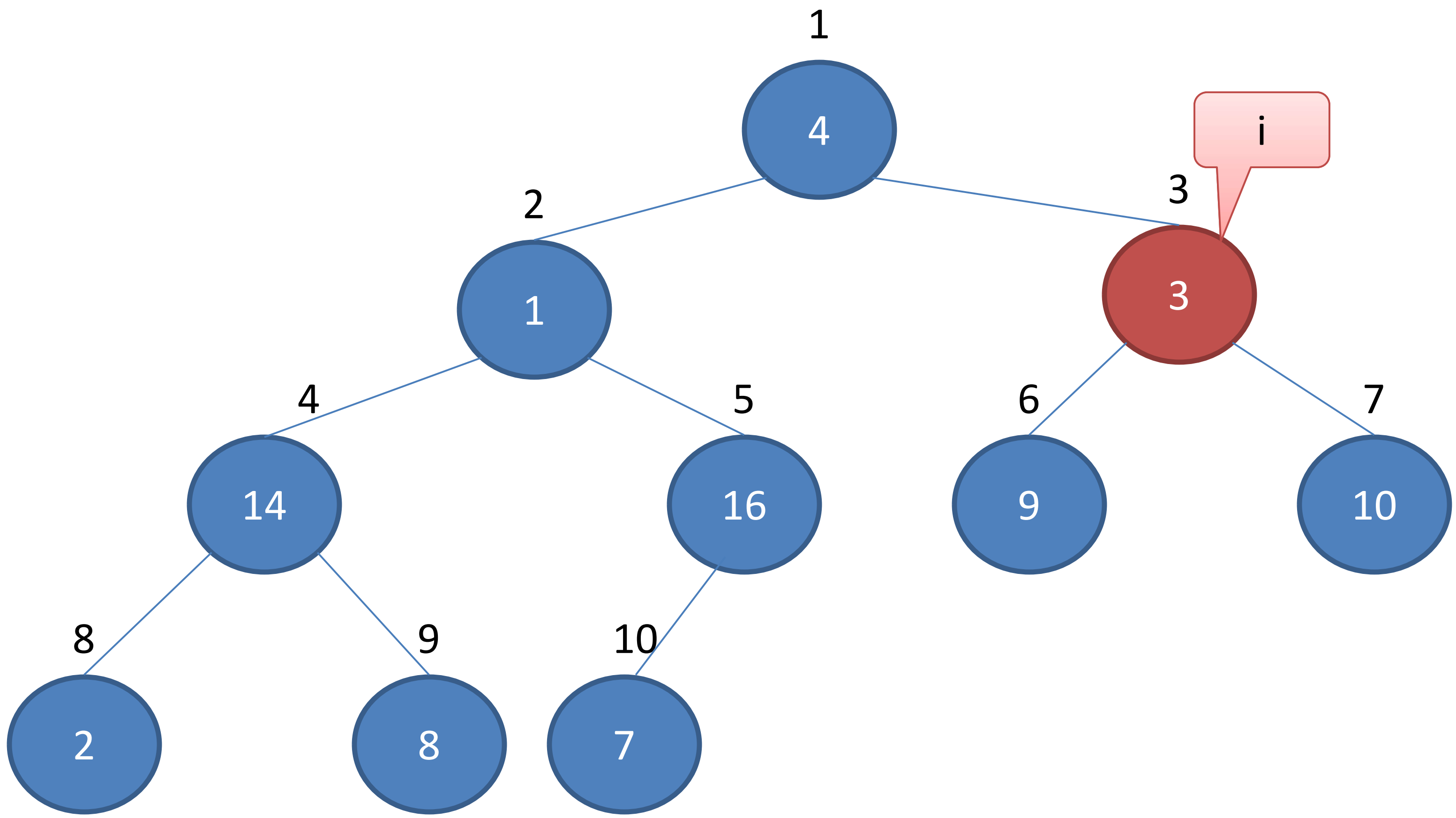
4	**1**	**3**	**2**	**16**	**9**	**10**	**14**	**8**	**7**

Example: Build-Max-Heap(A)



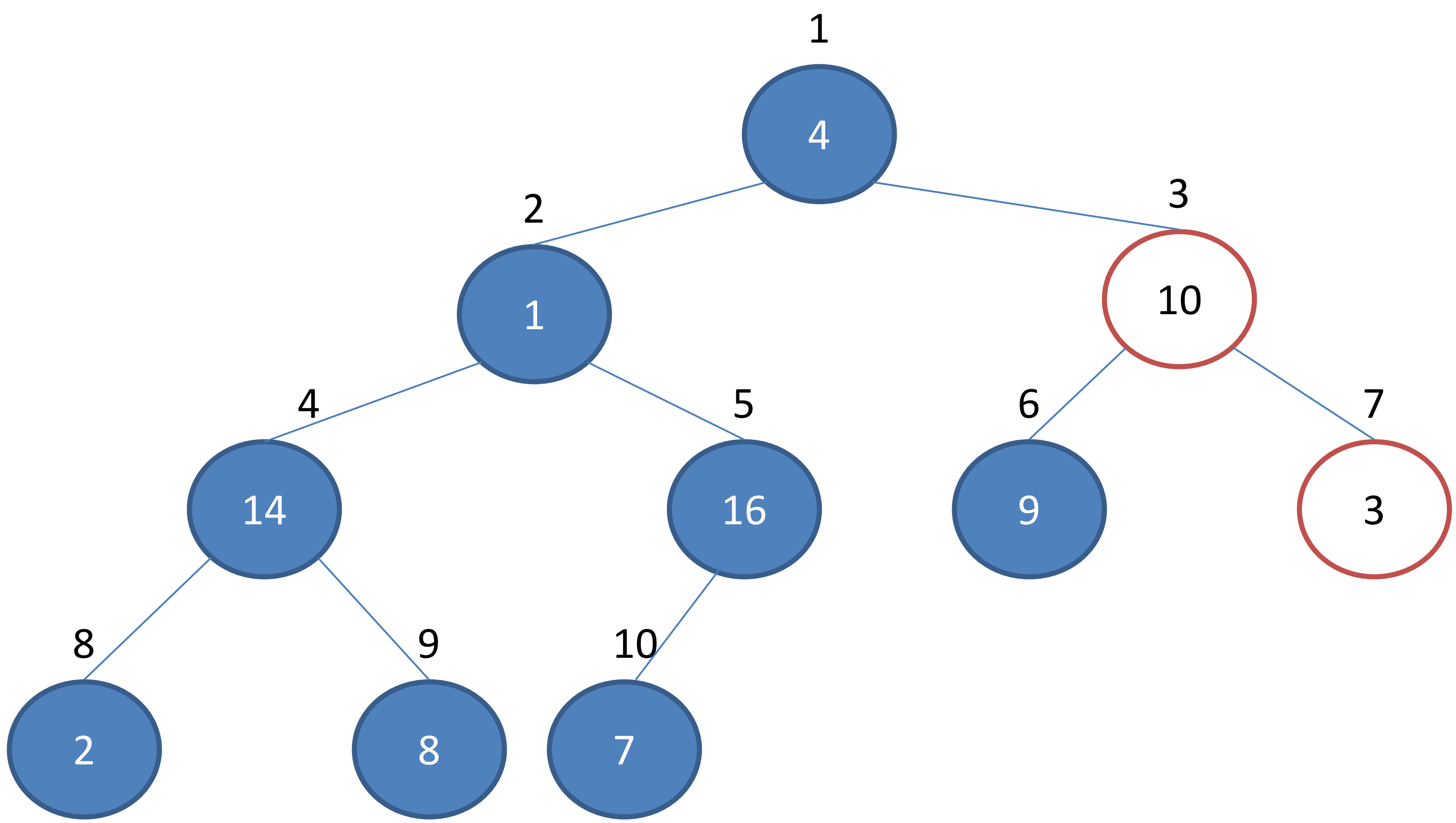
4	**1**	**3**	**14**	**16**	**9**	**10**	**2**	**8**	**7**

Example: Build-Max-Heap(A)



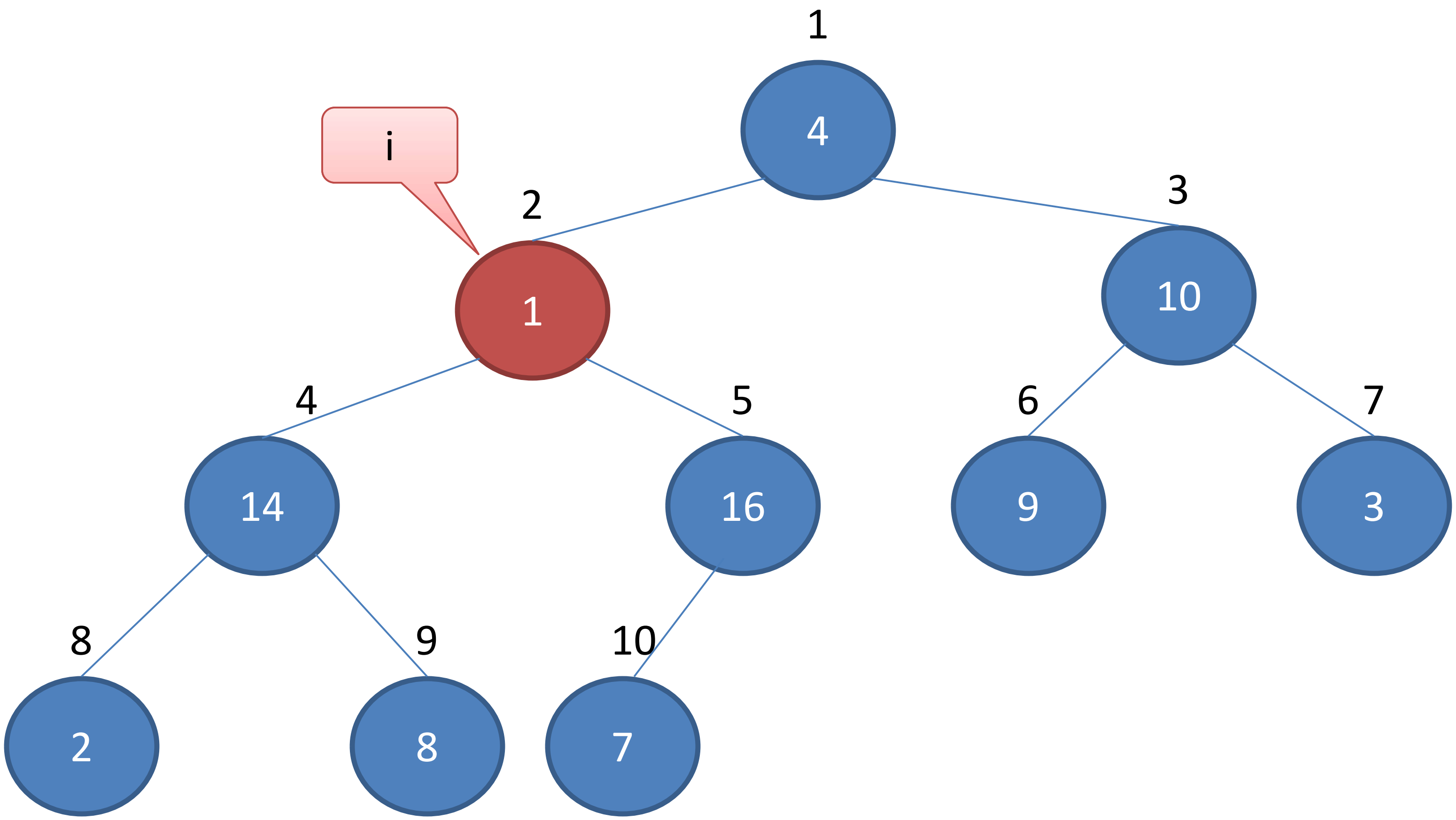
4	**1**	**3**	**14**	**16**	**9**	**10**	**2**	**8**	**7**

Example: Build-Max-Heap(A)



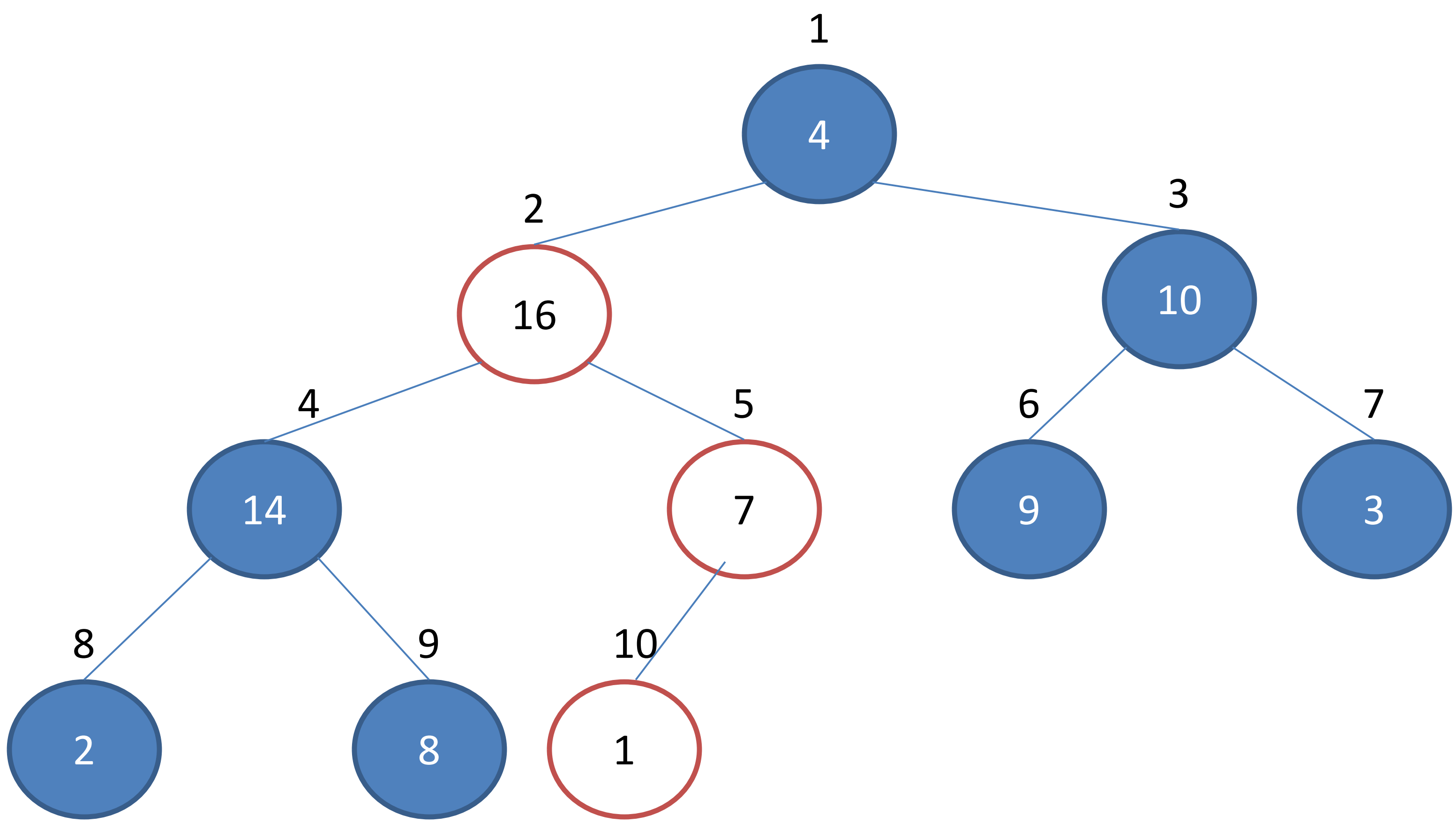
4	**1**	**10**	**14**	**16**	**9**	**3**	**2**	**8**	**7**

Example: Build-Max-Heap(A)



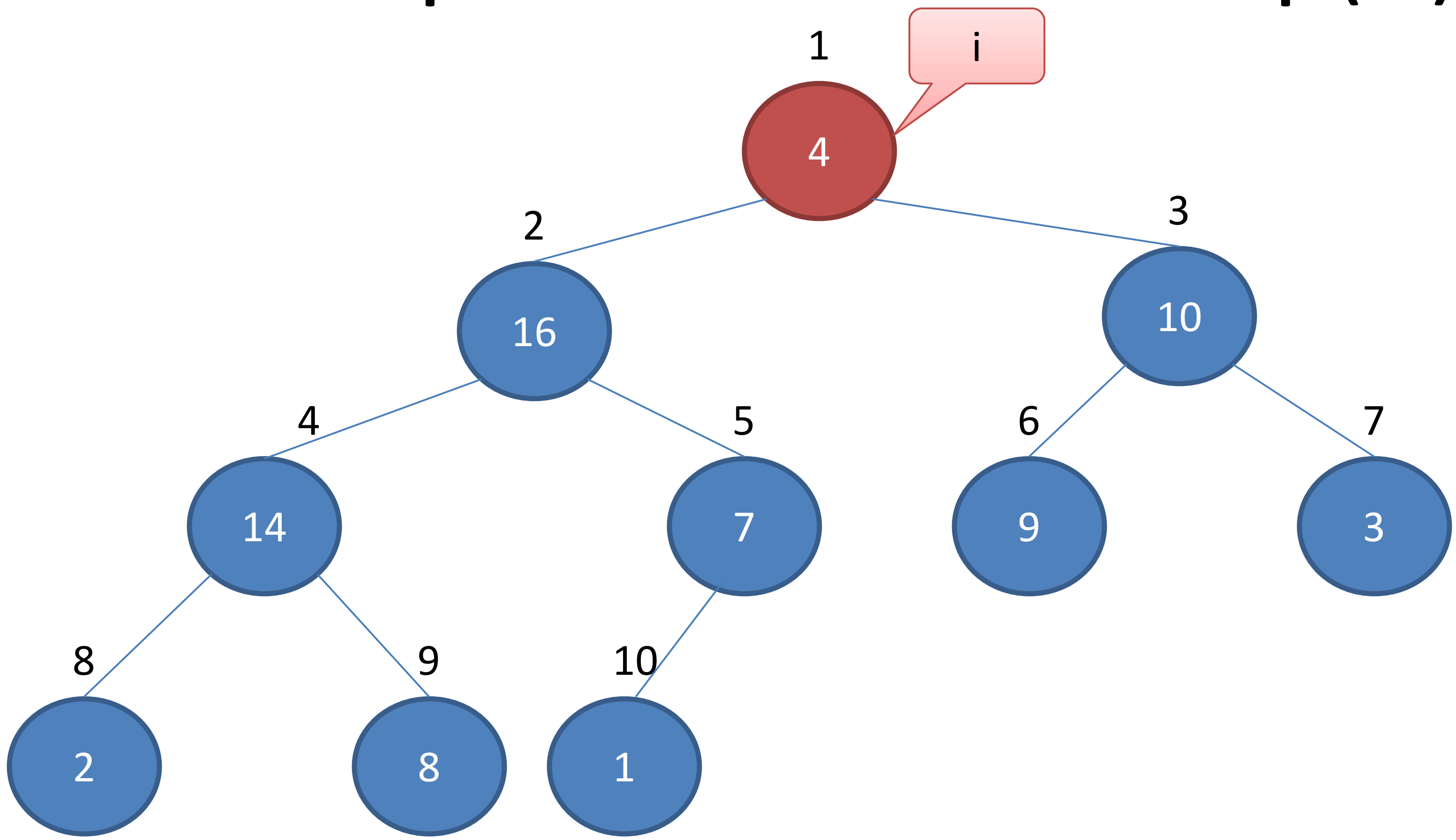
4	**1**	**10**	**14**	**16**	**9**	**3**	**2**	**8**	**7**

Example: Build-Max-Heap(A)



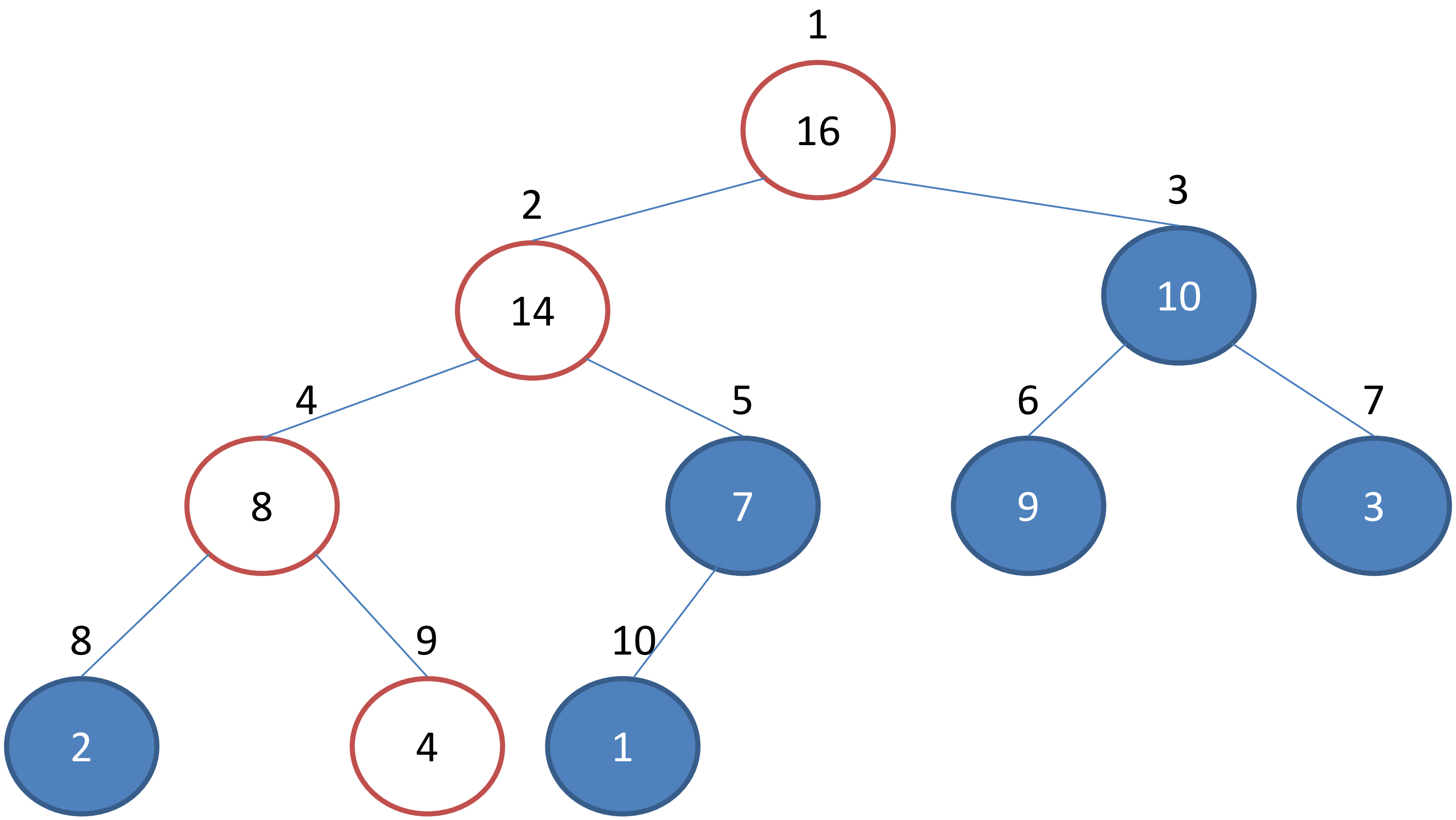
4	**16**	**10**	**14**	**7**	**9**	**3**	**2**	**8**	**1**

Example: Build-Max-Heap(A)



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

Example: Build-Max-Heap(A)



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

Analyze: Build-Max-Heap(A)

heap-size[A] = length[A]
for i = $\lfloor length[A]/2 \rfloor$ downto 1
        do Max-Heapify(A,i)

loop invariant =
at the start of each iteration of the for loop of lines 2-3, each node i+1, i+2,…,n is the root of a max-heap.

Initialization:
Before running loop 1 , i = $\lfloor n/2 \rfloor$. Each node $\lfloor n/2 \rfloor$+1, $\lfloor n/2 \rfloor$+2, …, n is a leaf and is the root of a trivial max-heap. (True!!)

Maintenance:
if children of node i are numbered higher than I, they are both roots of max-heaps.
The condition required for the call Max-Heapify(A,i) to make node I a max-heap root.
Decrementing in the for loop update reestablishes the loop invariant for the next loop. (True!!)

Termination: at termination, i =0. each node 1,2,….,n Is the root of a max-heap. Node 1 is. (True!!)

Analyze running time: Build-Max-Heap(A)

heap-size[A] = length[A]
for i = $\lfloor length[A]/2 \rfloor$ downto 1
do Max-Heapify(A,i)

Times
1
n/2+1
n/2.O(lg n)

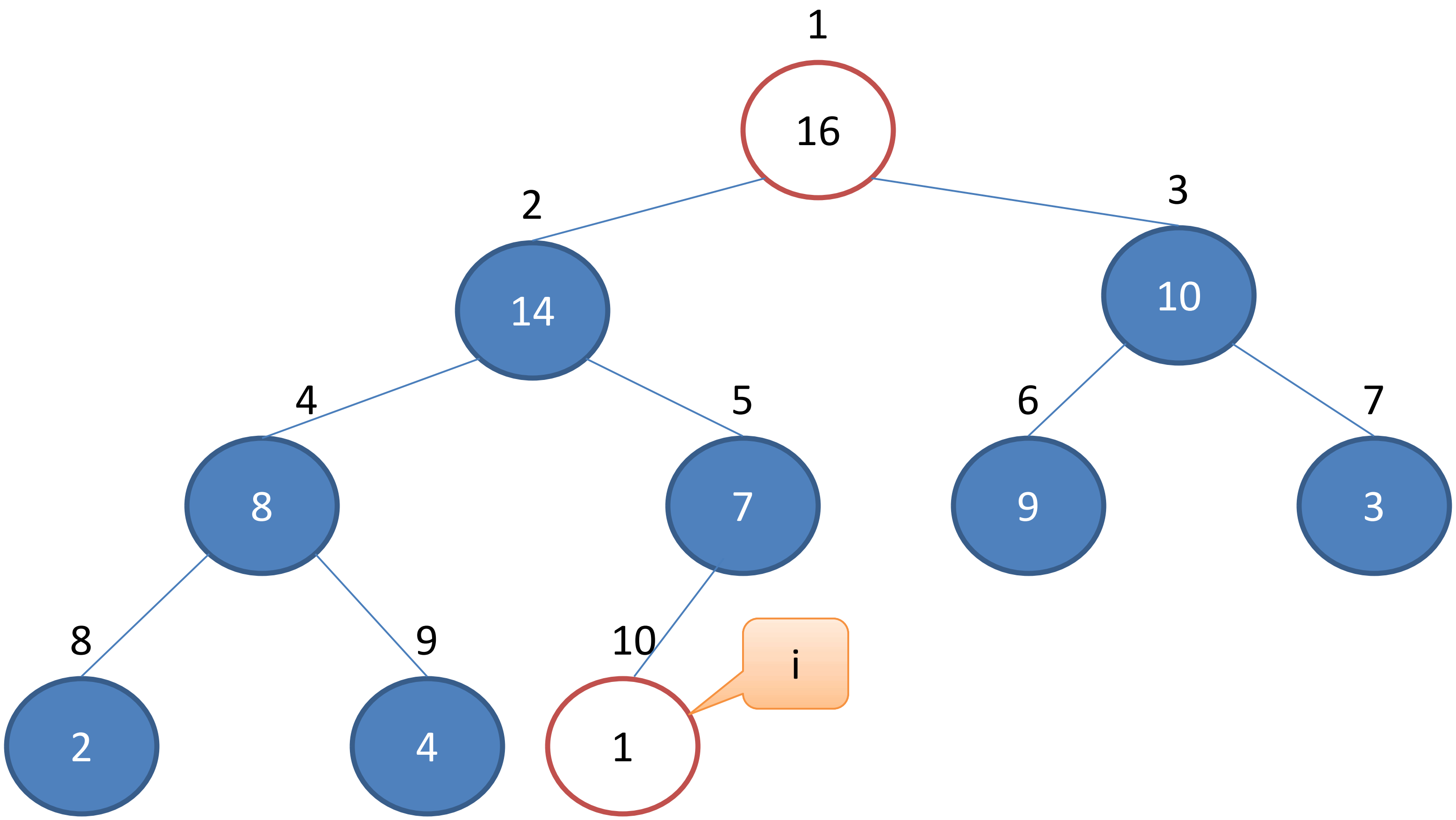
T(n) = O(n lg n)

Tight Analysis: an n-element heap has height = $\lfloor \lg n \rfloor$ and at most $\lceil n/2^{h+1} \rceil$ nodes of any height h.
The time required by Max-Heapify when called on a node of height h is O(h). Thus running time can be bounded as O(n)

Heapsort(A)

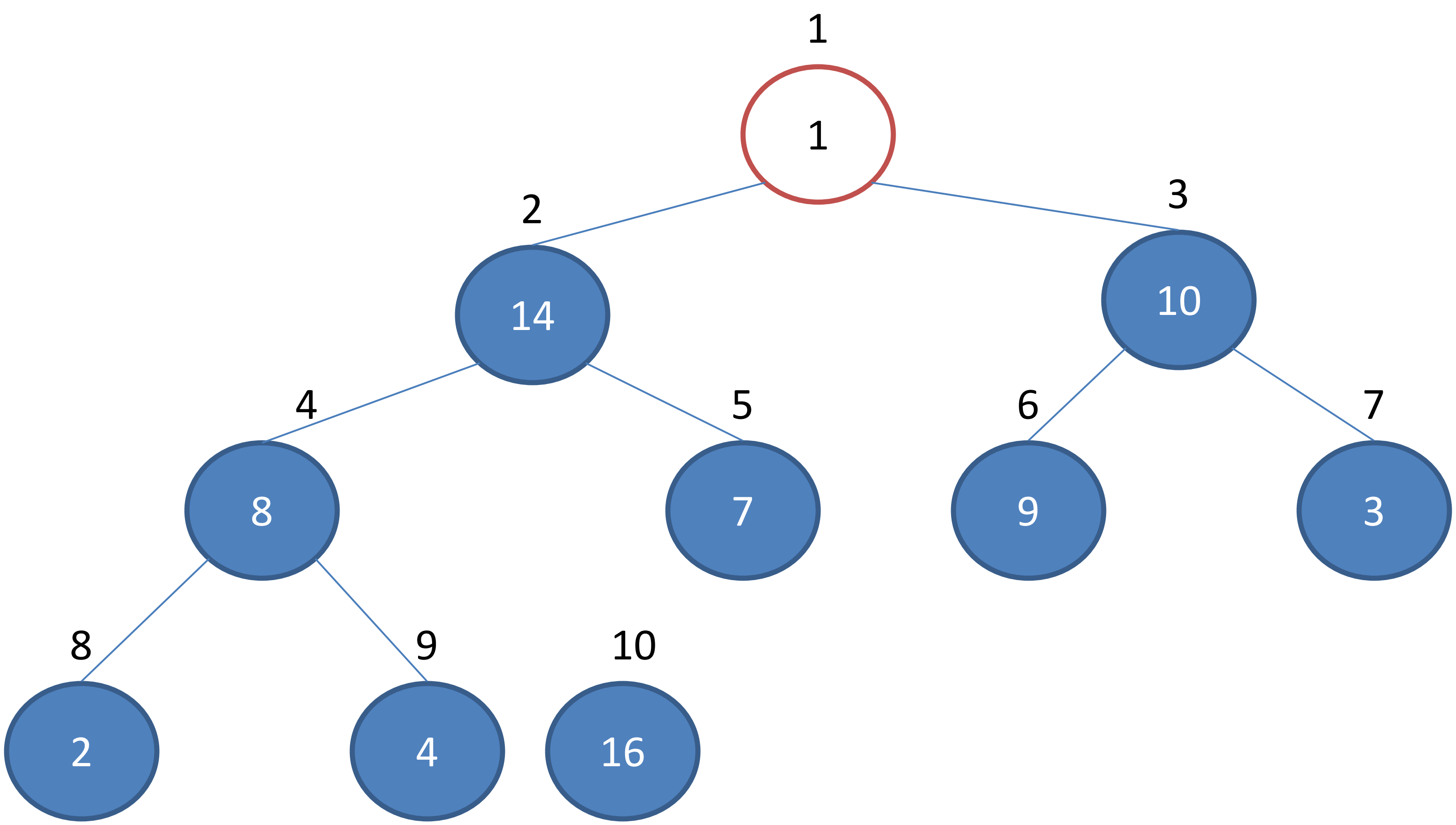
```
Build-Max-Heap(A)
for i = length[A]  downto 2
        do exchange A[1] and A[i]
        heap-size[A] = heap-size[A] -1
        Max-Heapify(A,1)
```

Example: Heapsort(A)



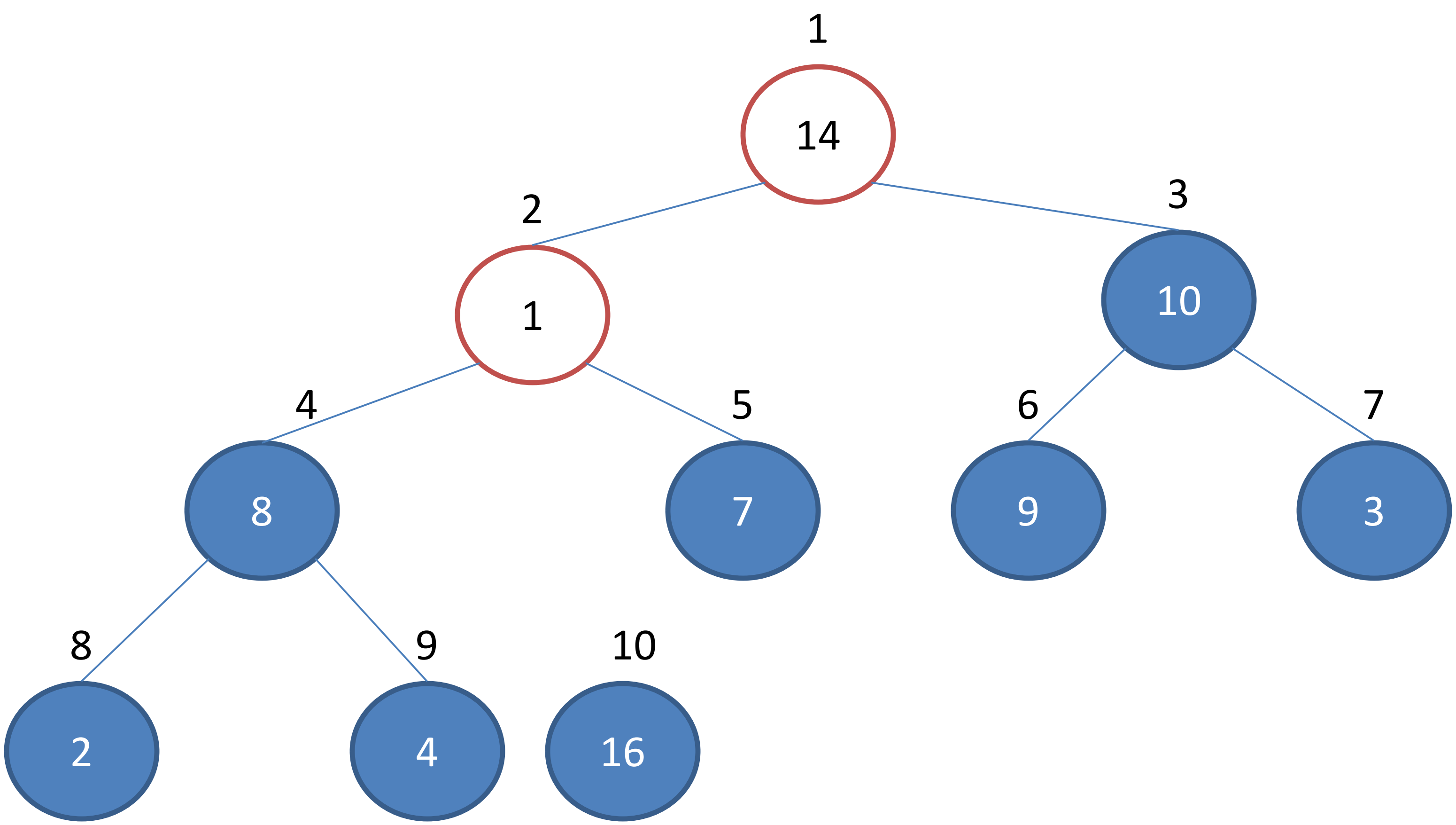
16	**14**	**10**	**8**	**7**	**9**	**3**	**2**	**4**	**1**

Example: Heapsort(A)



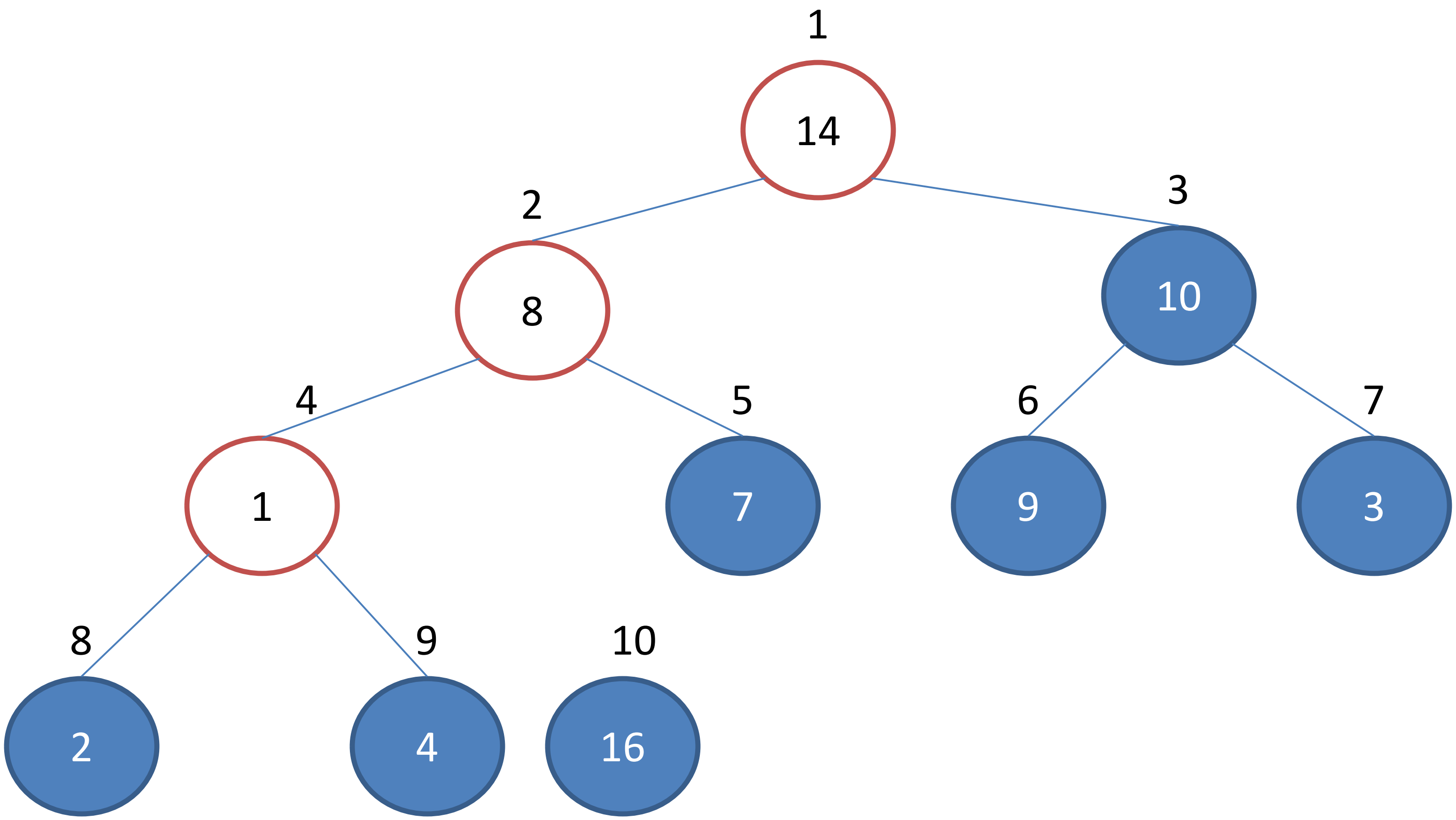
1	**14**	**10**	**8**	**7**	**9**	**3**	**2**	**4**	**16**

Example: Heapsort(A)



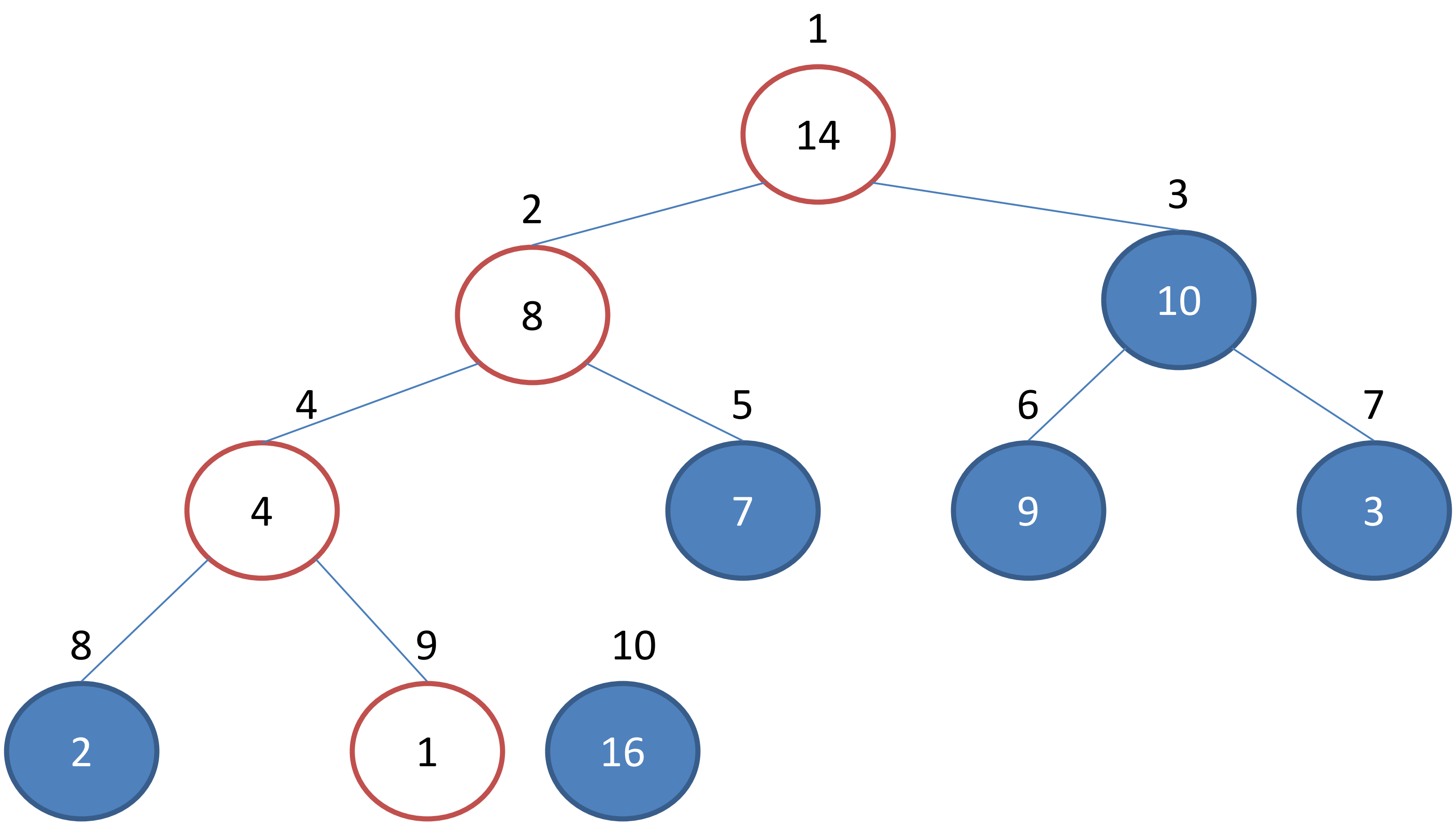
14	1	10	8	7	9	3	2	4	16

Example: Heapsort(A)



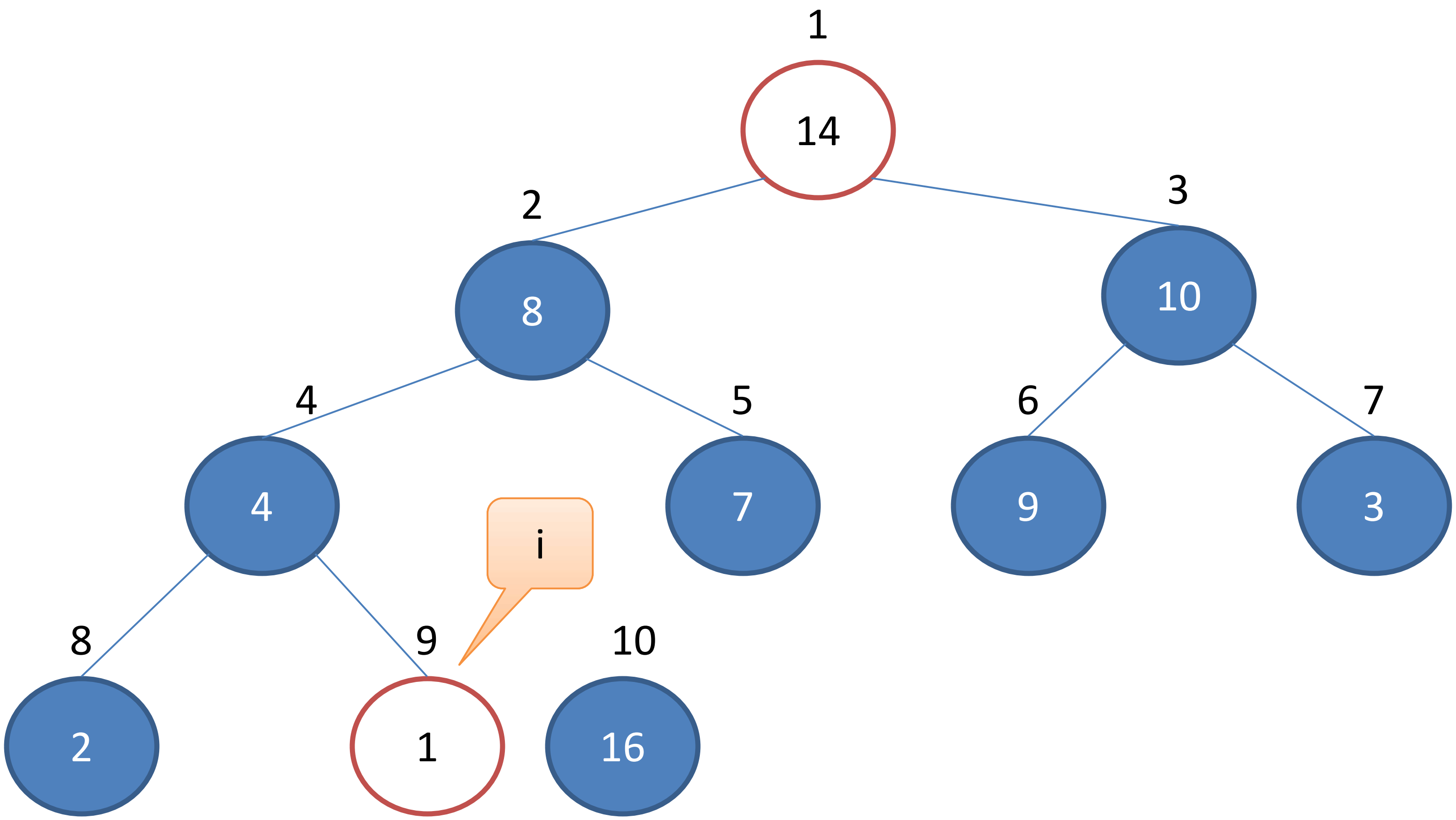
14	8	10	1	7	9	3	2	4	16

Example: Heapsort(A)



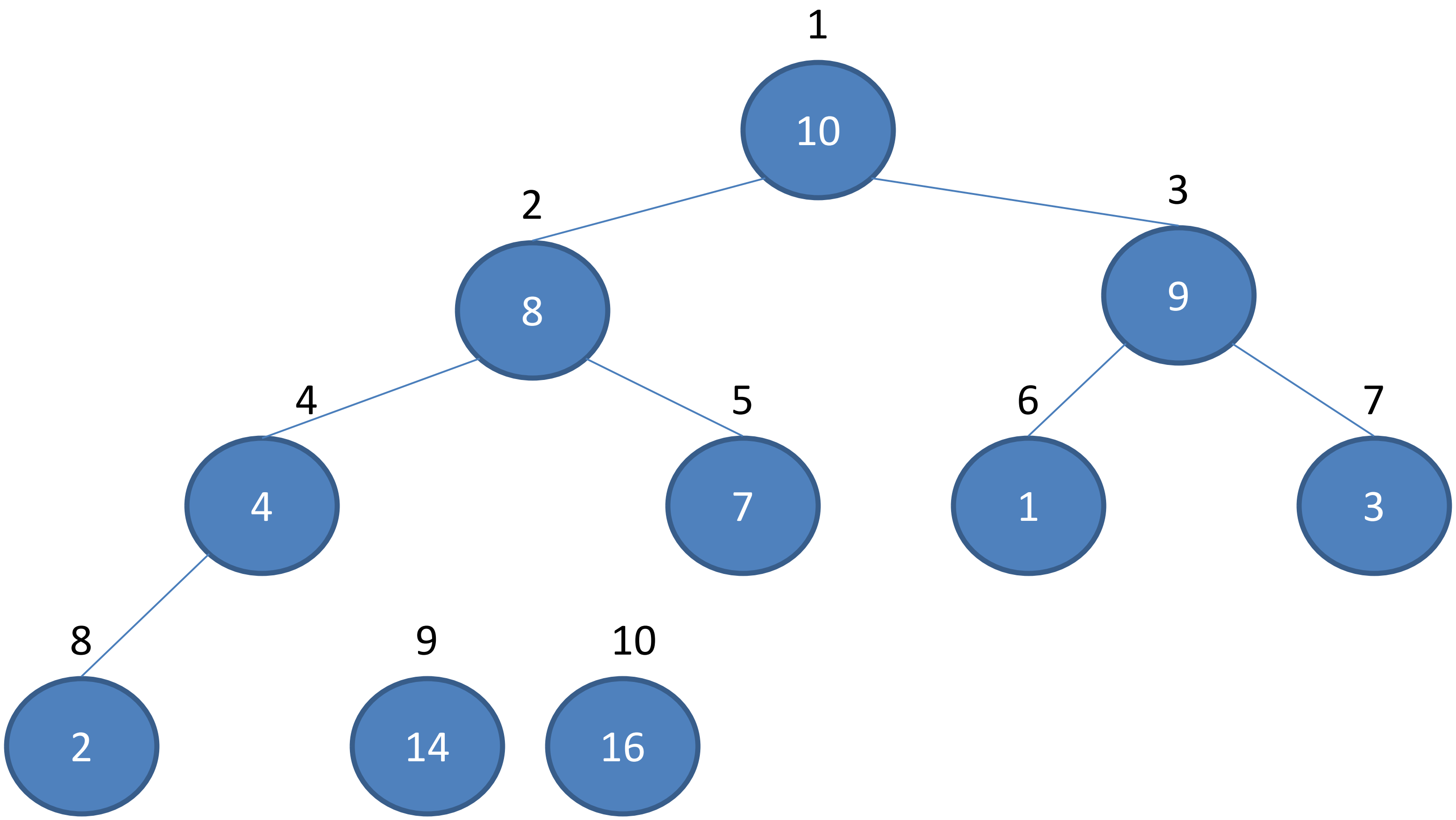
14	**8**	**10**	**4**	**7**	**9**	**3**	**2**	**1**	**16**

Example: Heapsort(A)



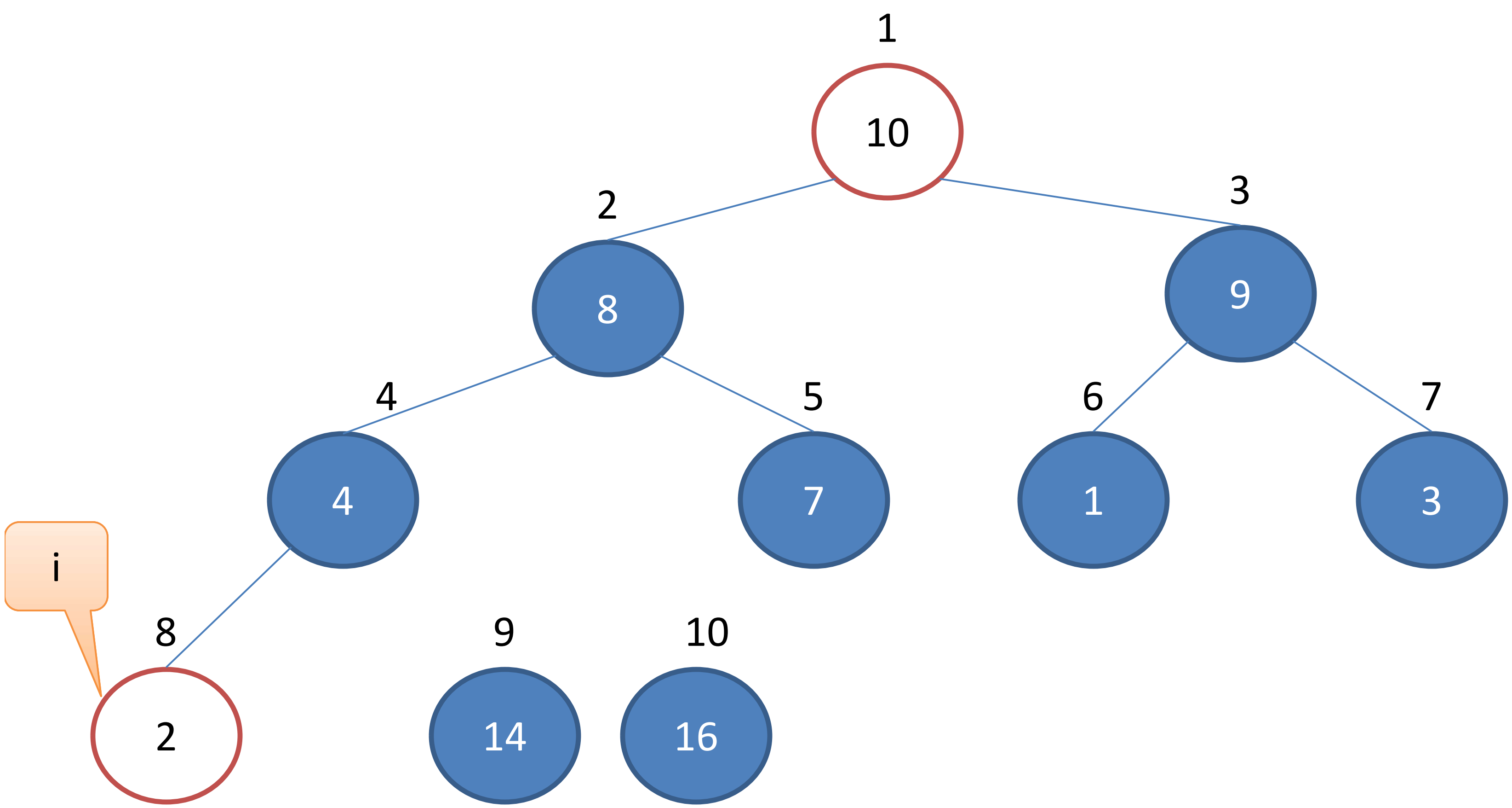
14	**8**	**10**	**4**	**7**	**9**	**3**	**2**	**1**	**16**

Example: Heapsort(A)



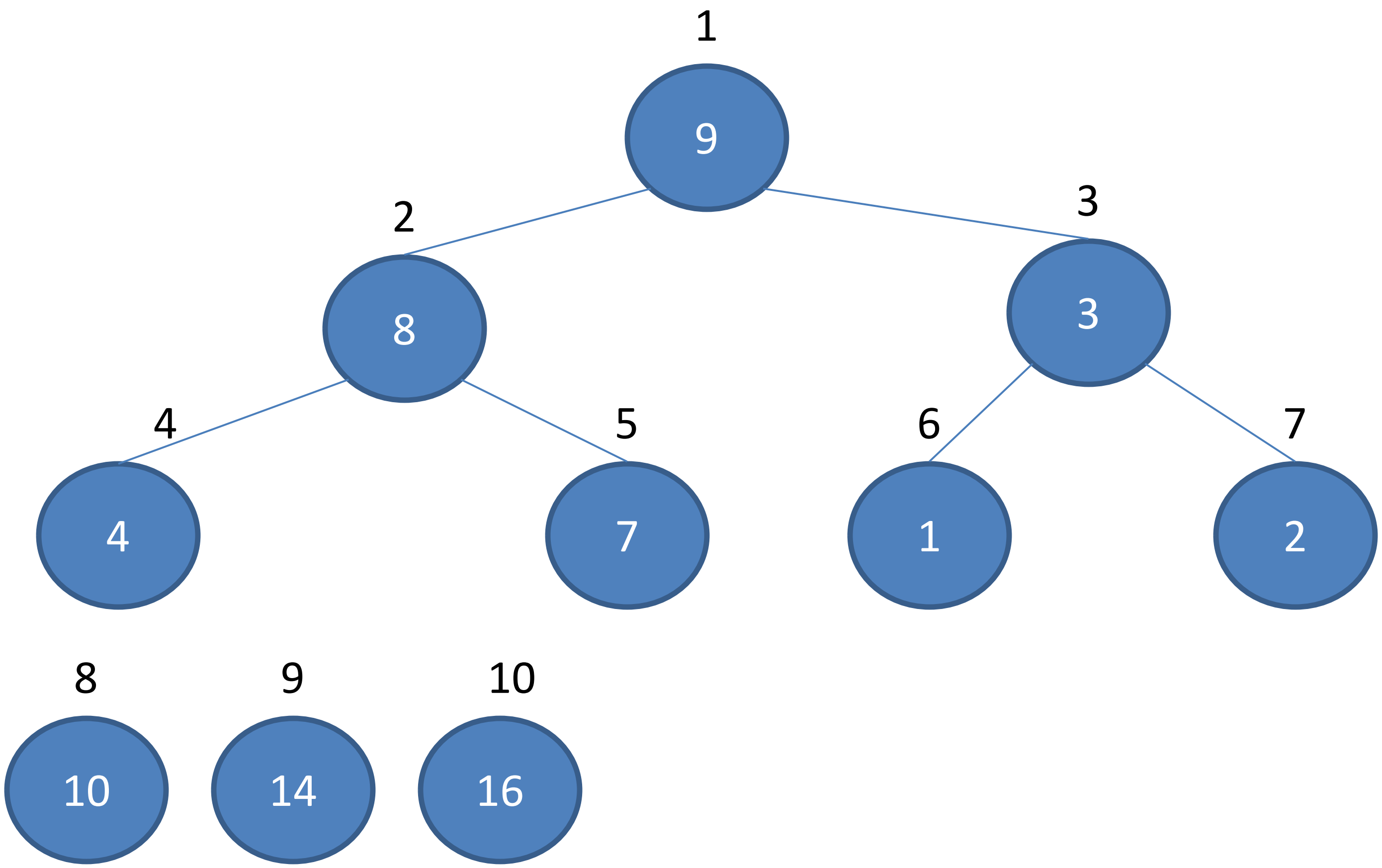
10	8	9	4	7	1	3	2	14	16

Example: Heapsort(A)



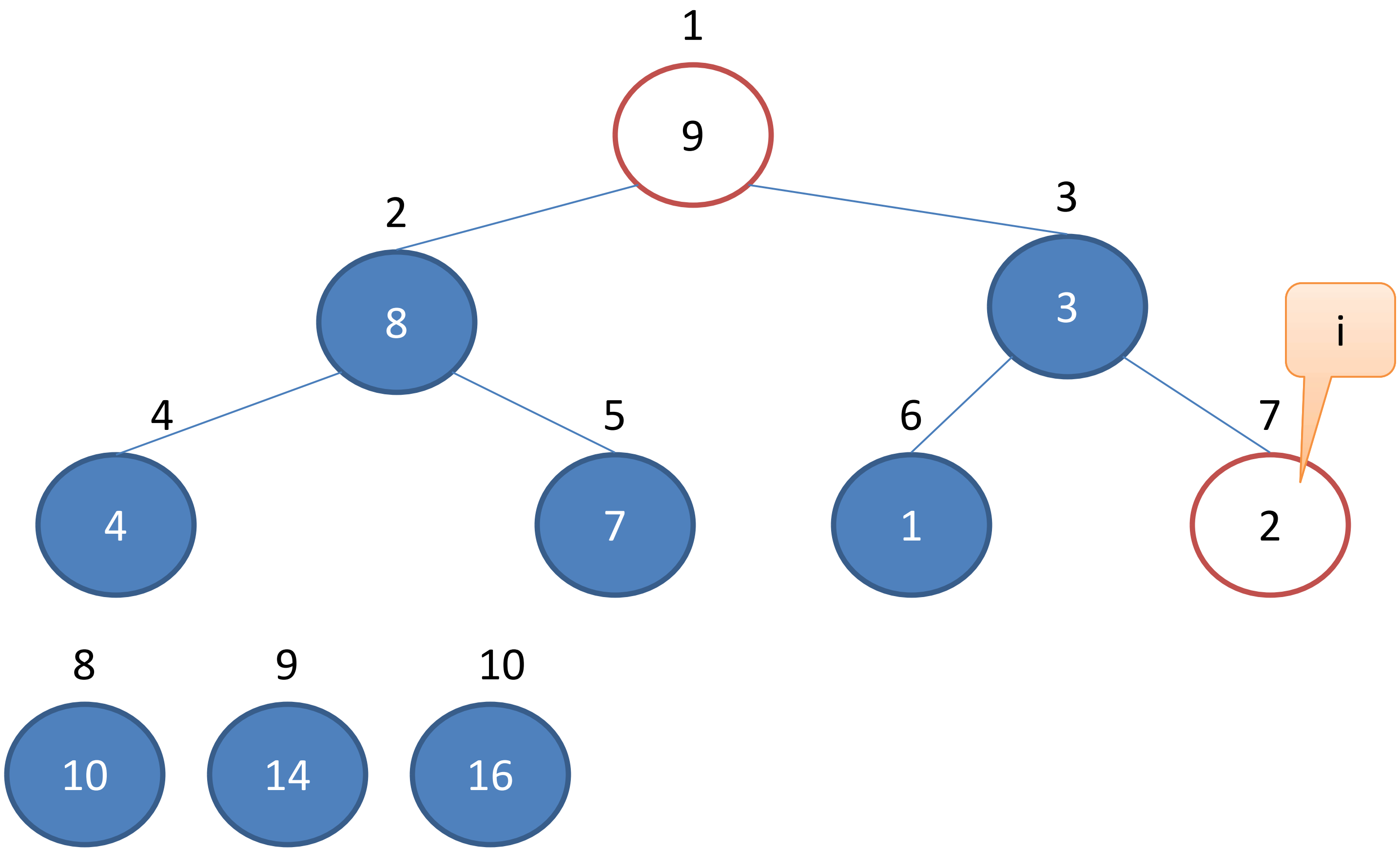
10	8	9	4	7	1	3	2	14	16

Example: Heapsort(A)



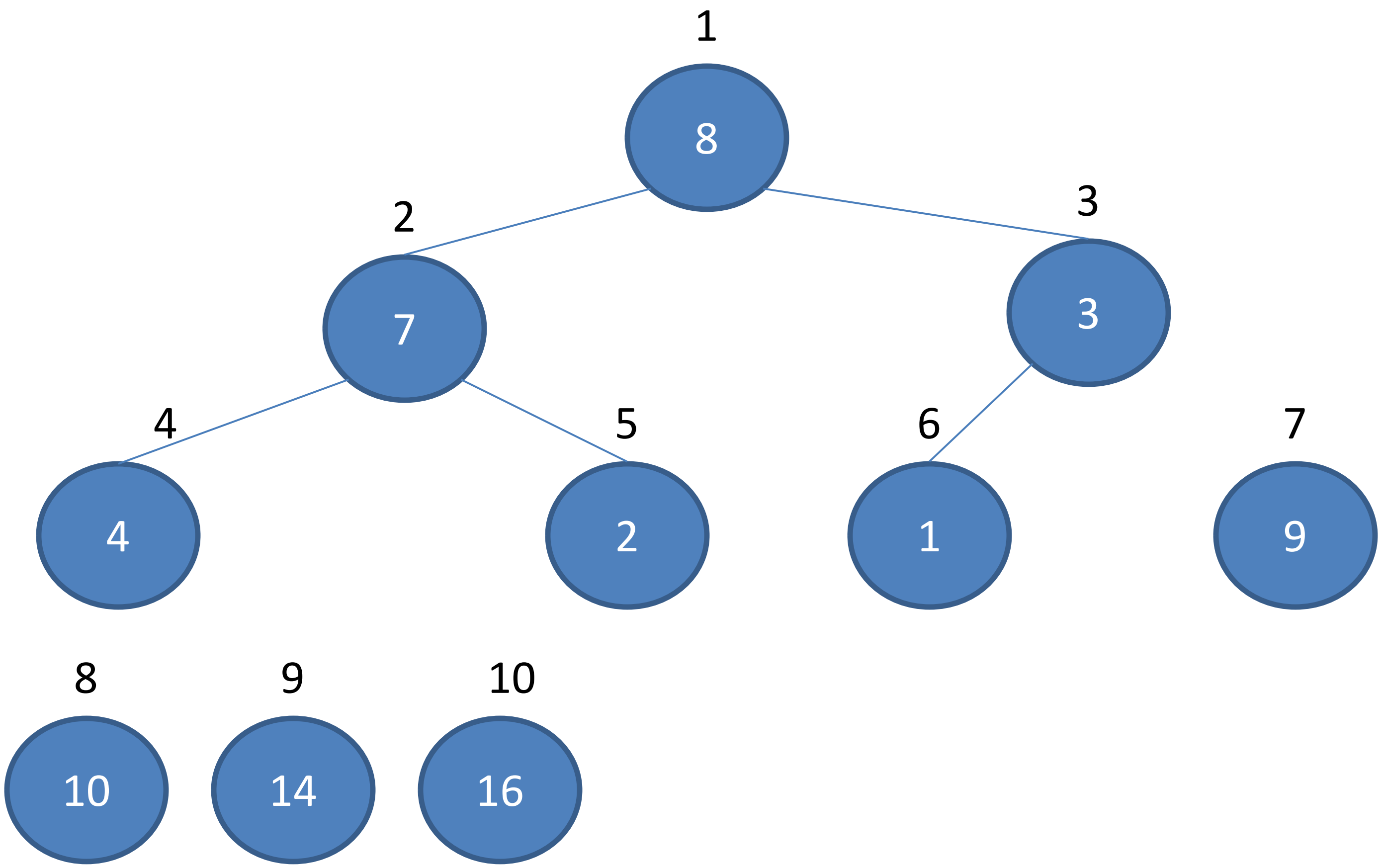
9	**8**	**3**	**4**	**7**	**1**	**2**	**10**	**14**	**16**

Example: Heapsort(A)



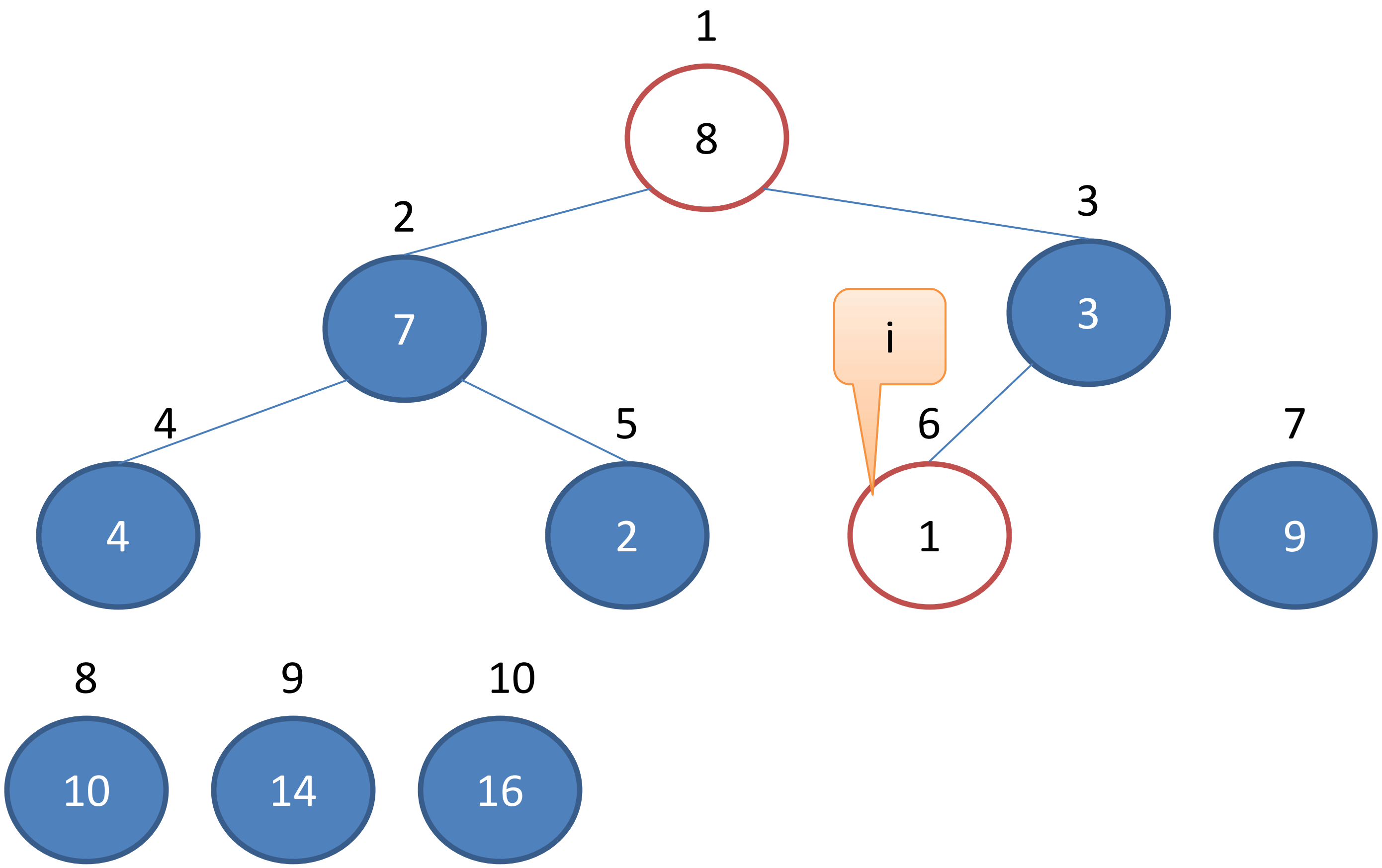
9	**8**	**3**	**4**	**7**	**1**	**2**	**10**	**14**	**16**

Example: Heapsort(A)



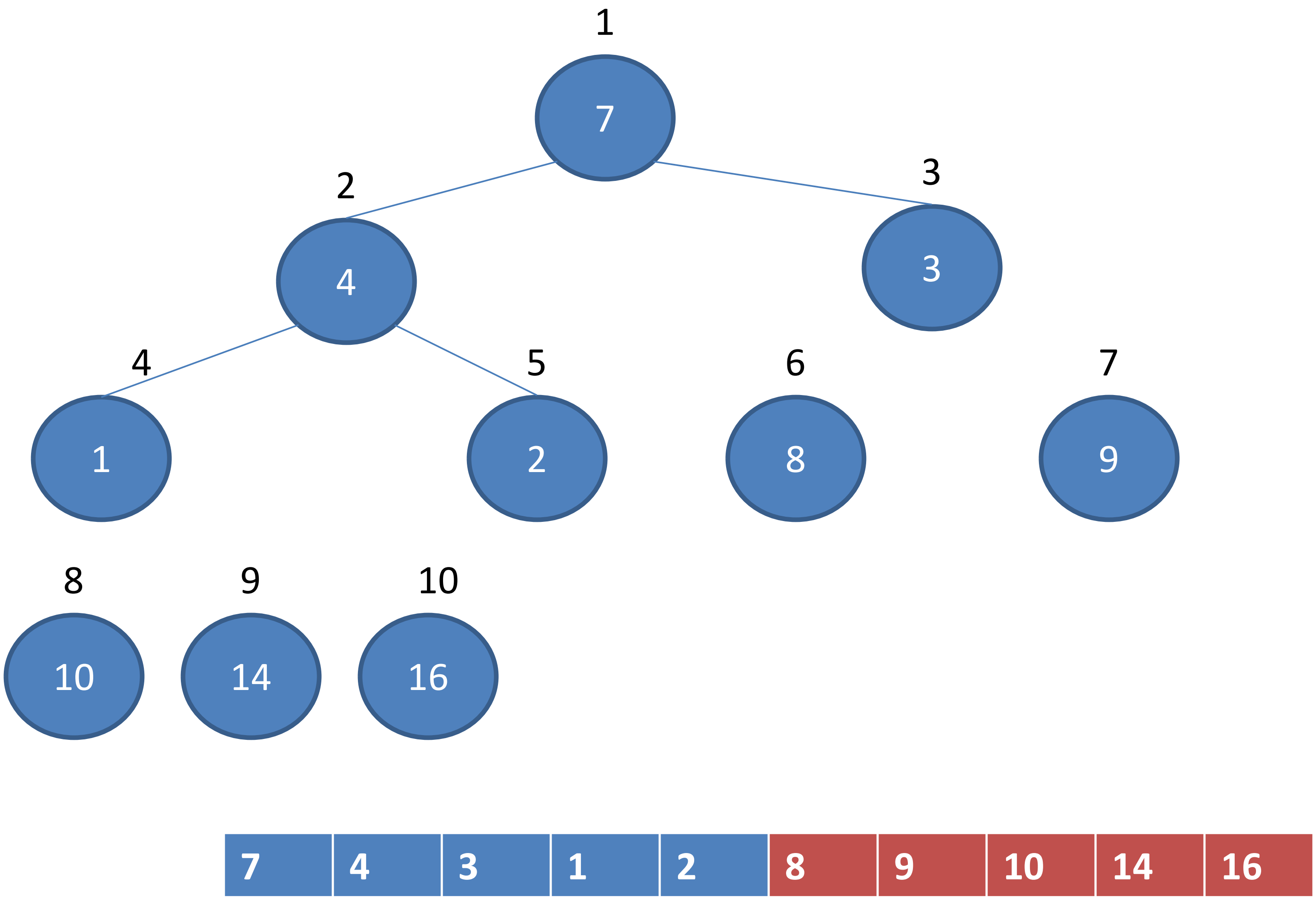
8	7	3	4	7	1	9	10	14	16

Example: Heapsort(A)

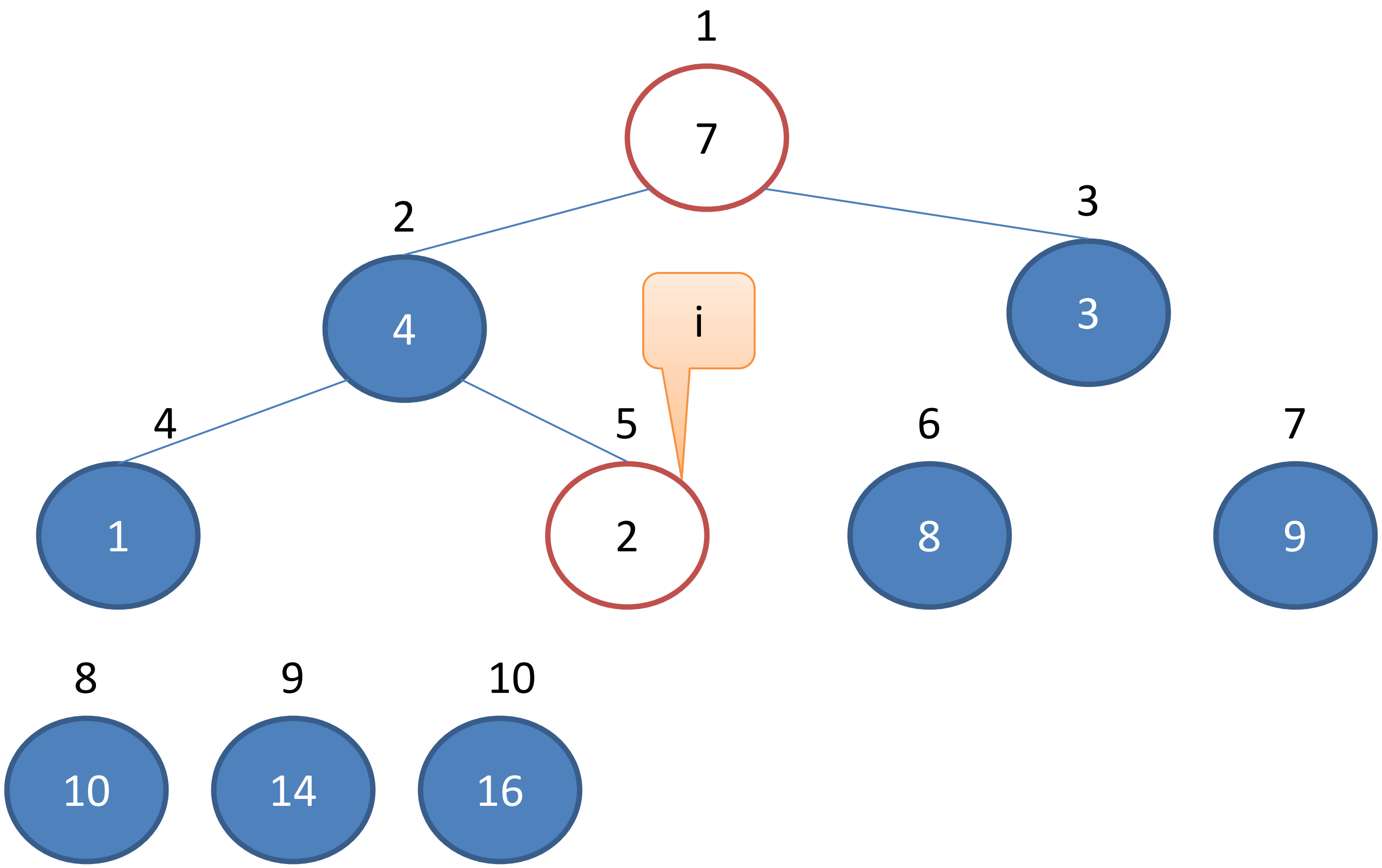


8	7	3	4	7	1	9	10	14	16

Example: Heapsort(A)

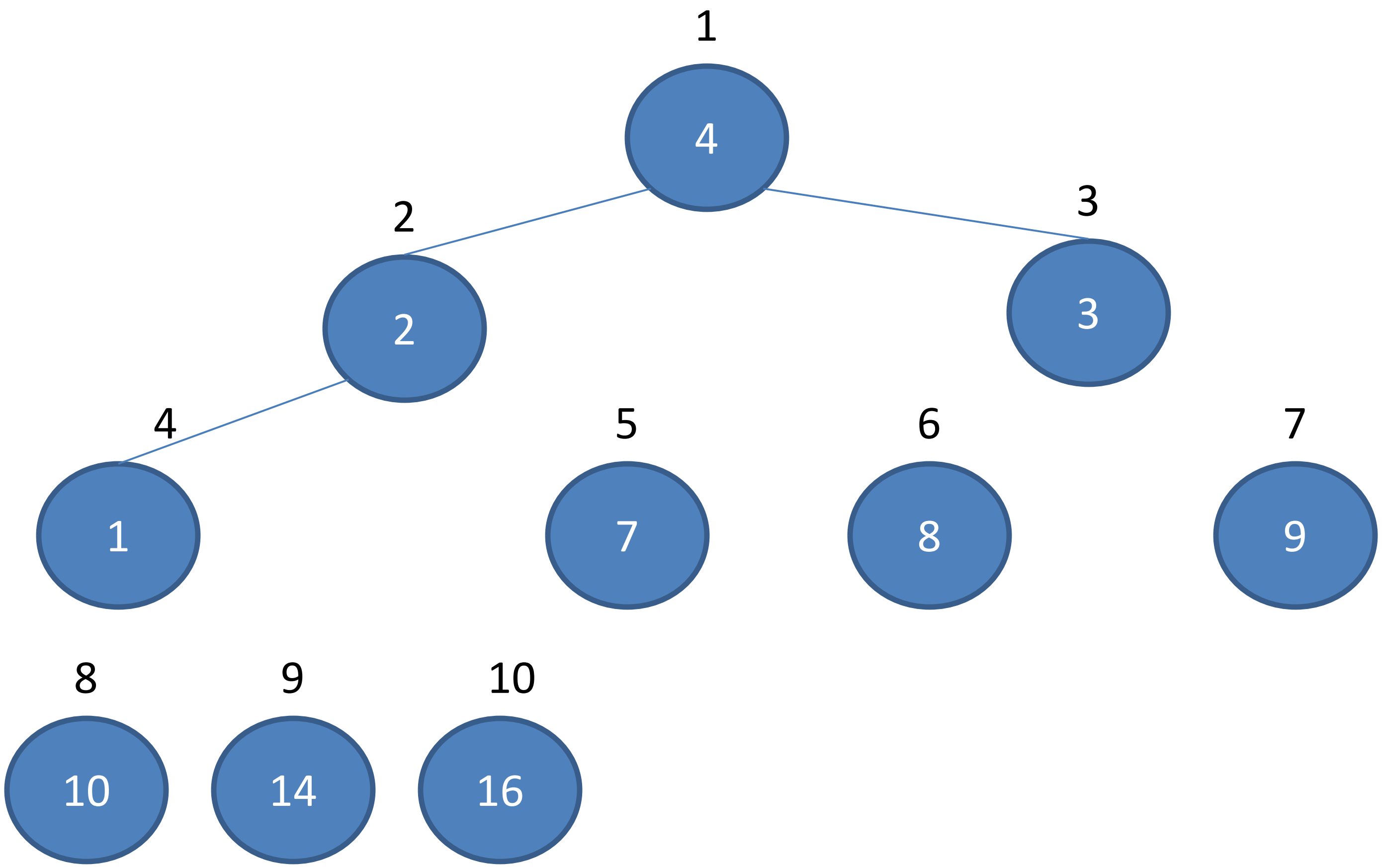


Example: Heapsort(A)



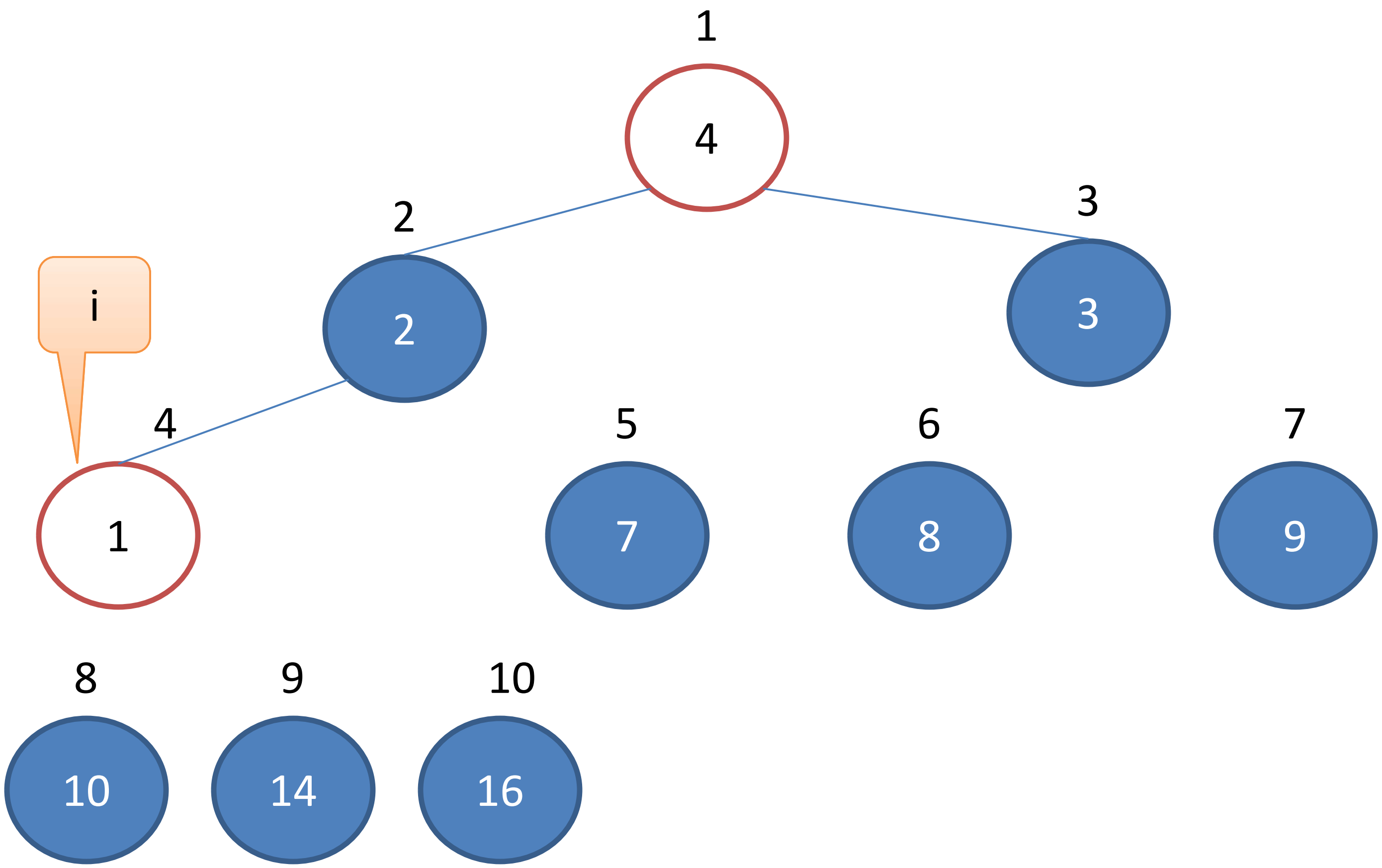
7	**4**	**3**	**1**	**2**	**8**	**9**	**10**	**14**	**16**

Example: Heapsort(A)



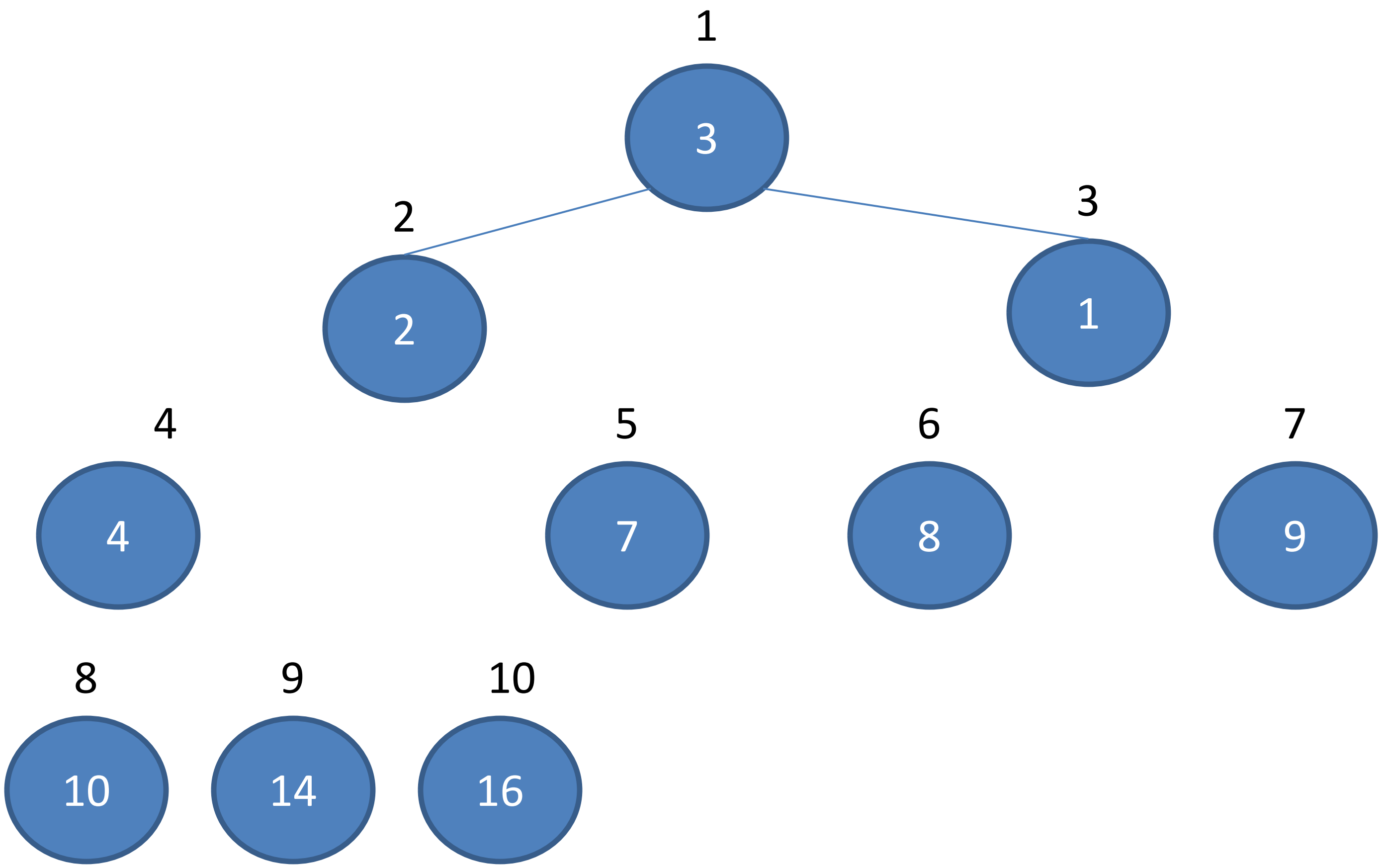
4	**2**	**3**	**1**	**7**	**8**	**9**	**10**	**14**	**16**

Example: Heapsort(A)



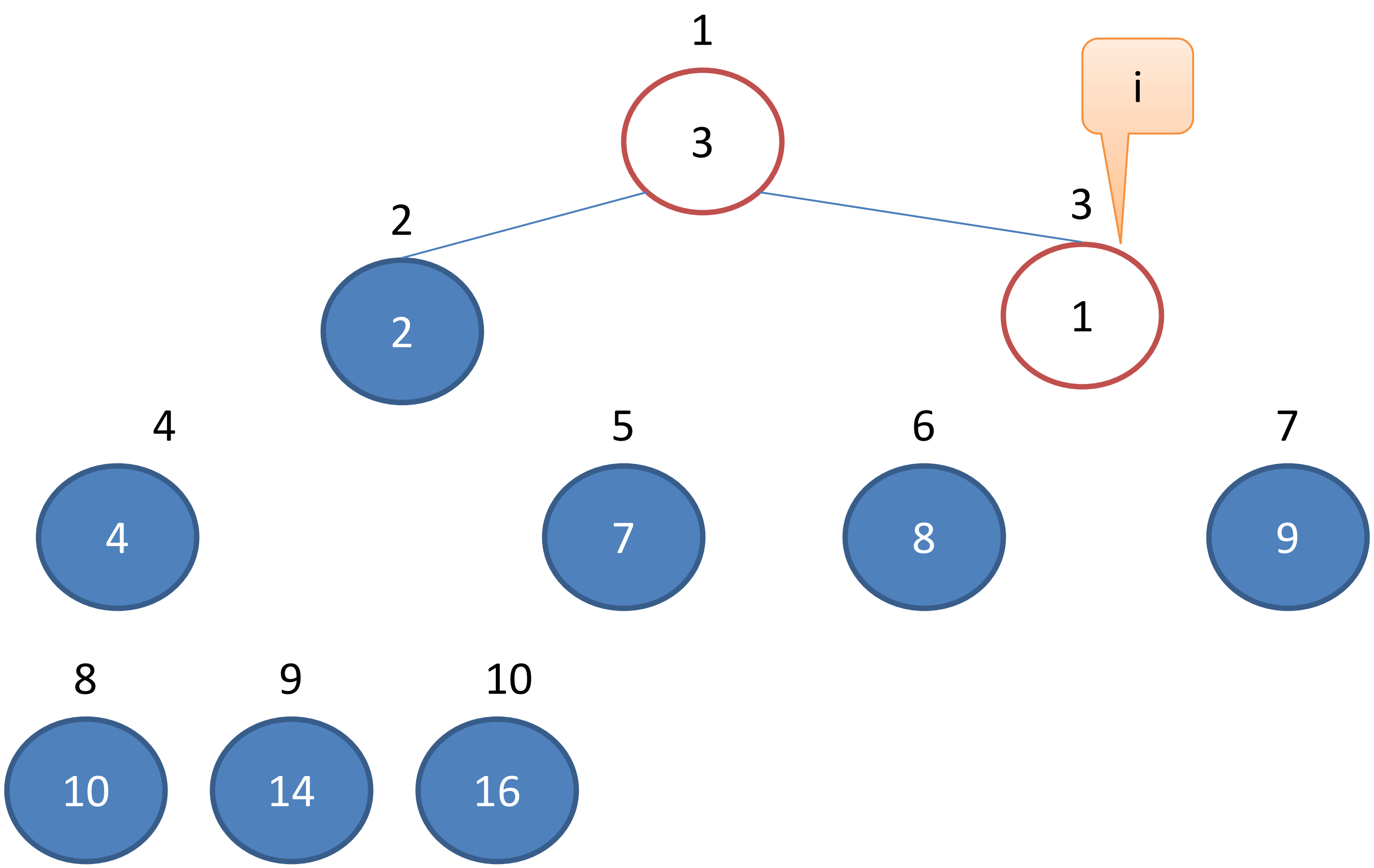
4	**2**	**3**	**1**	**7**	**8**	**9**	**10**	**14**	**16**

Example: Heapsort(A)



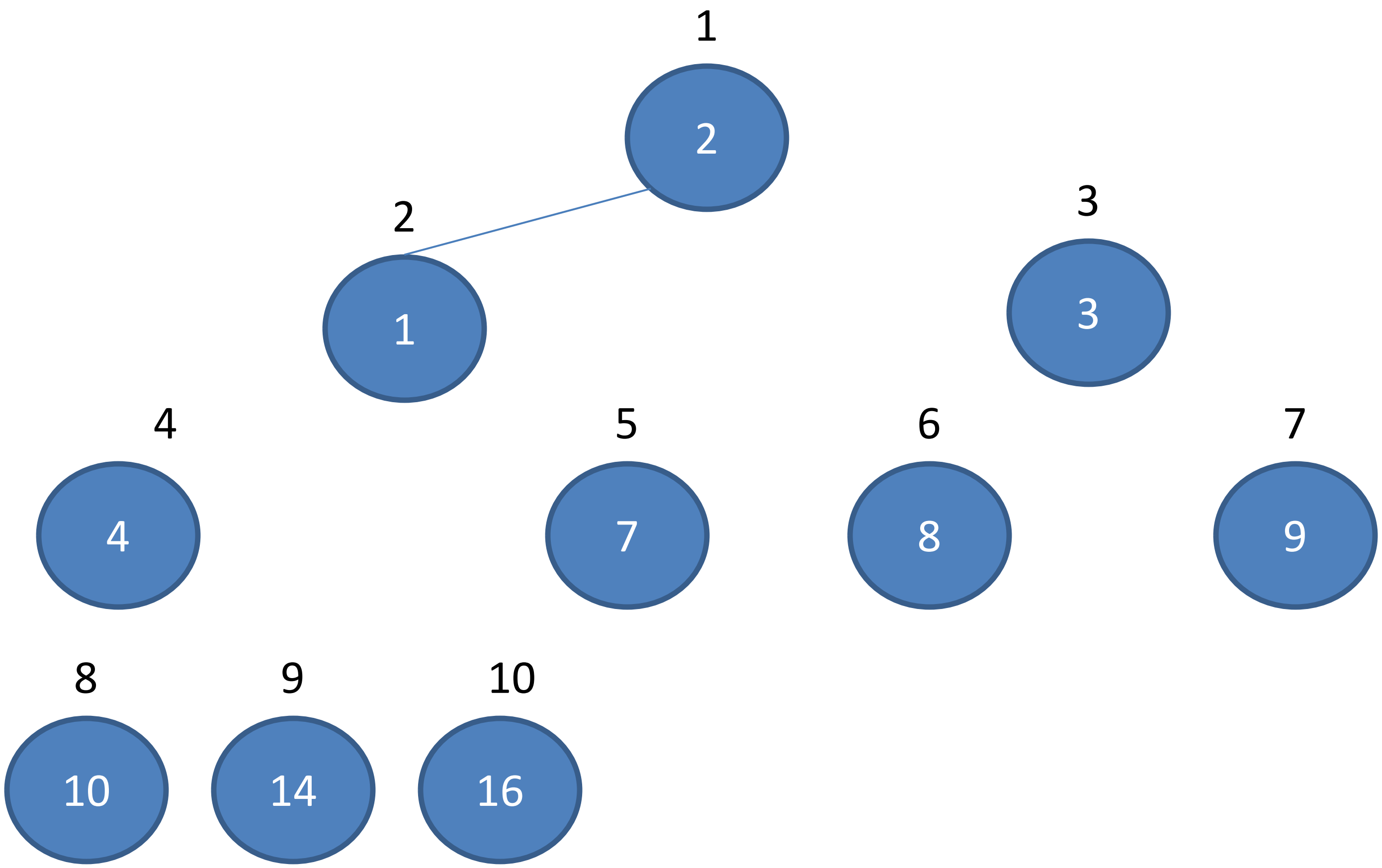
3	**2**	**1**	**4**	**7**	**8**	**9**	**10**	**14**	**16**

Example: Heapsort(A)



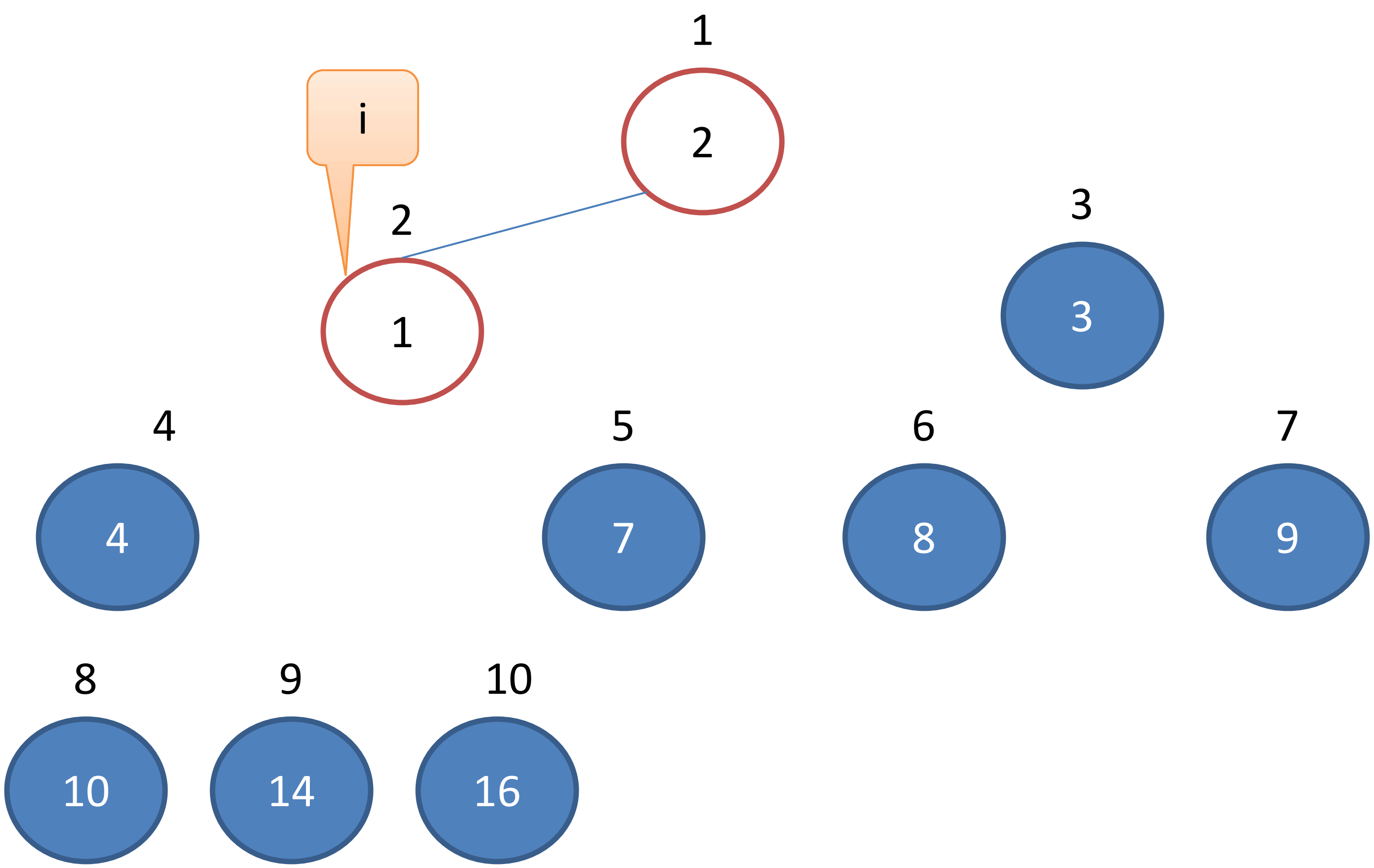
3	**2**	**1**	**4**	**7**	**8**	**9**	**10**	**14**	**16**

Example: Heapsort(A)



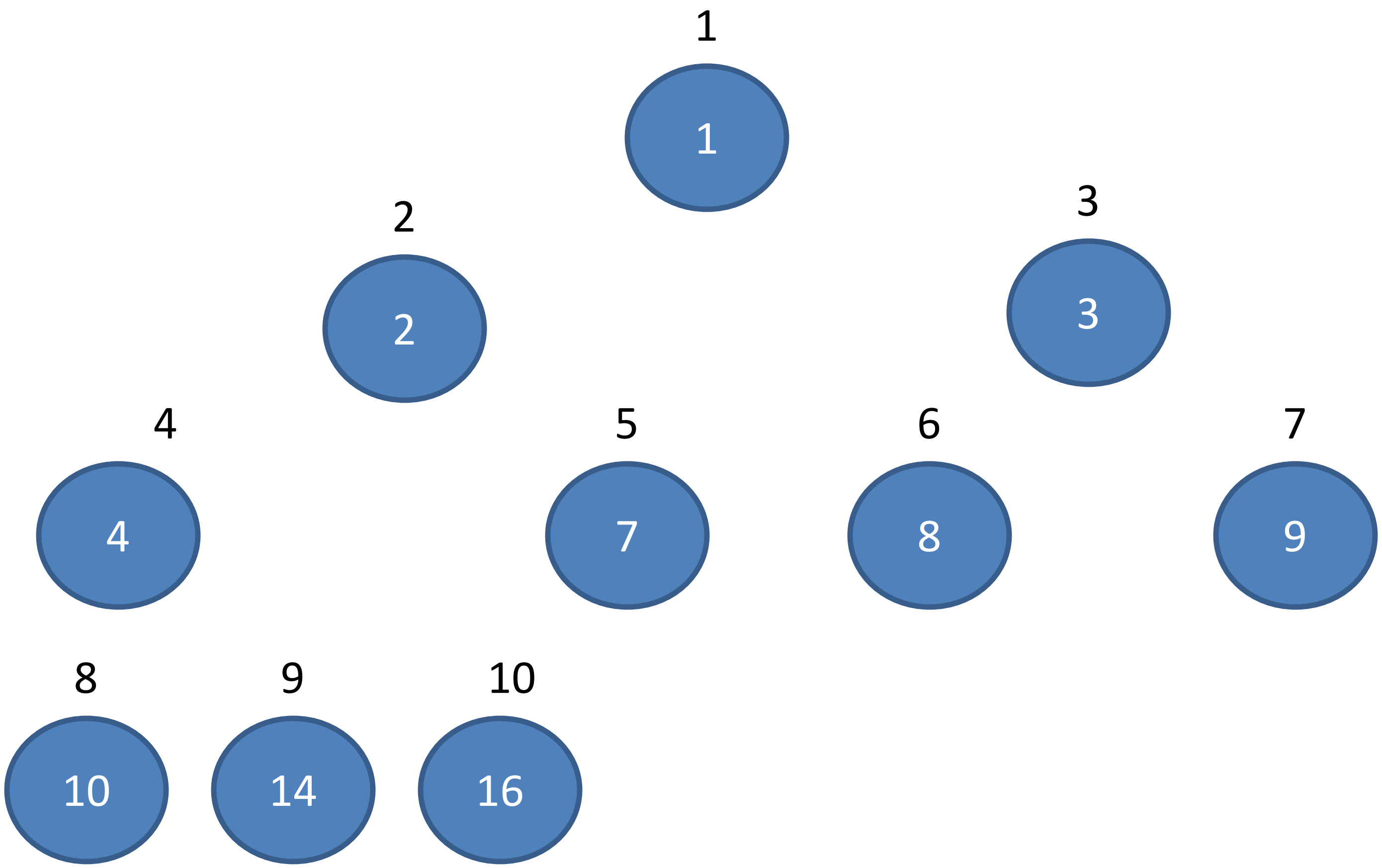
2	**1**	**3**	**4**	**7**	**8**	**9**	**10**	**14**	**16**

Example: Heapsort(A)



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

Example: Heapsort(A)



1	2	3	4	7	8	9	10	14	16

Analyze: Heapsort(A)

	Times
Build-Max-Heap(A)	O(n)
for i = length[A] downto 2	n
do exchange A[1] and A[i]	n-1
heap-size[A] = heap-size[A] -1	n-1
Max-Heapify(A,1)	n-1 . O(lg n)

T(n) = O(n lg n)

Practice: Heapsort

8	17	12	15	92	16	11	52	41

Priority Queues

- A priority queue is a data structure for maintaining a set S of elements, each with an associated value called a key.
- There are two kinds of priority queues:
- A max priority queue supports these operations:
  - Insert(S,x) -> inserts the element x into the set S.
  - Maximum(S) -> returns the element of S with the largest key.
  - Extract-Max(S) -> removes and returns the element of S with the largest key.
  - Increase-Key(S,x,k) -> increases the value of element x's key to the new value k which is assumed to be as large as x's current key value.
- A min priority queue supports these operations:
  - Insert(S,x) , Minimum(S) , Extract-Min(S), Decrease-Key(S,x,k)

Max-Priority Queue

Pseudo-code: Heap-Maximum(A)

```
return A[1]
```

Running time = O(1)

Max-Priority Queue

Pseudo-code: Heap-Extract-Max(A)

```
if heap-size[A] < 1
        then error "heap underflow"

max=A[1]
A[1] = A[heap-size[A] ]
heap-size[A] = heap-size[A] – 1
Max-Heapify(A,1)
return max
```

Running time = O(lg n)

Max-Priority Queue

Pseudo-code: Heap-Increase-Key(A,I,Key)

```
if key < A[i]
   then error "new key is smaller than current key"

A[i] = key
while  i > 1 and A[parent(i)] < A[i]
          do exchange A[i] and A[parent(i)]
          i = parent(i)
```

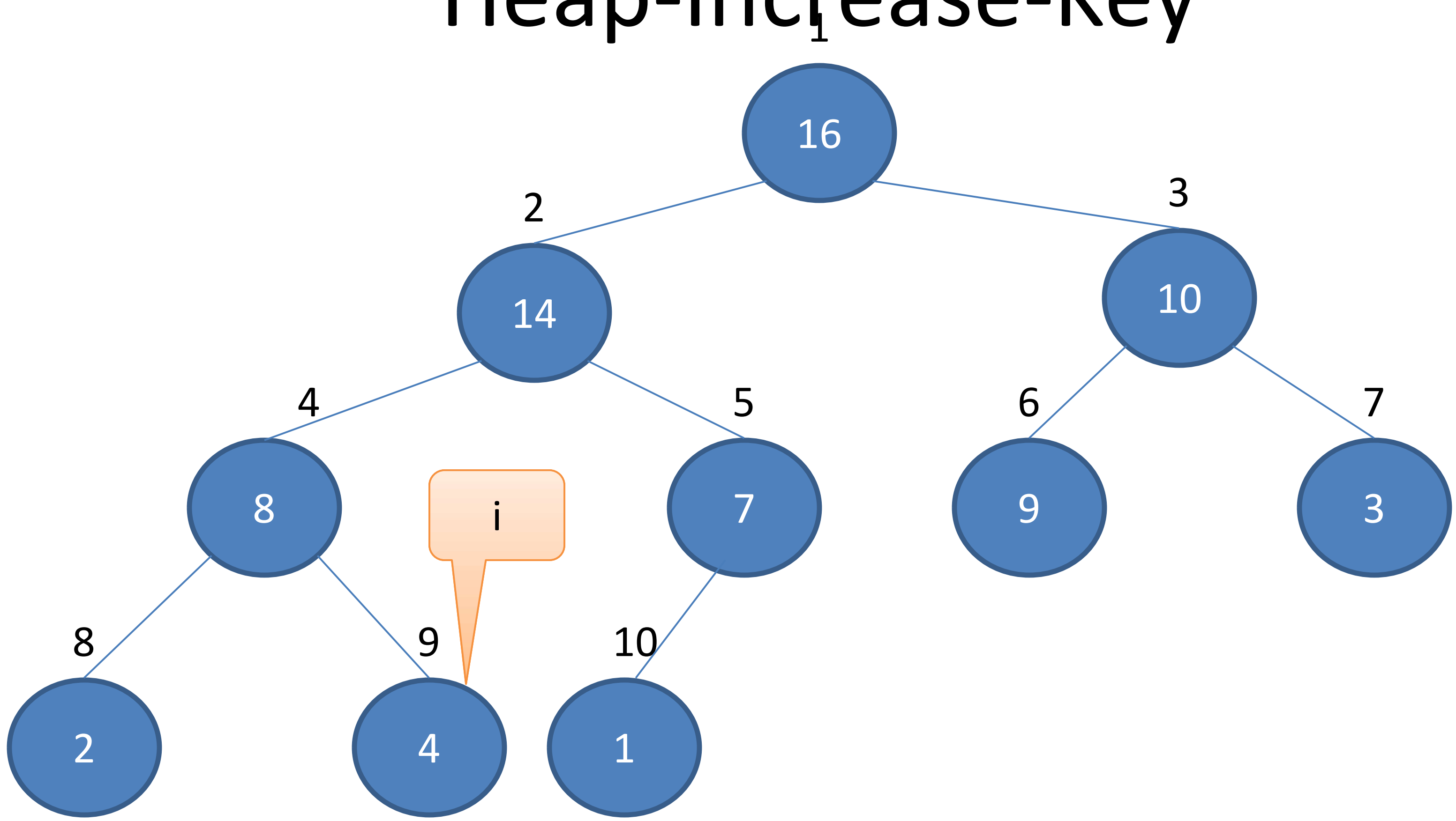
Running time = O(lg n)

Max-Priority Queue

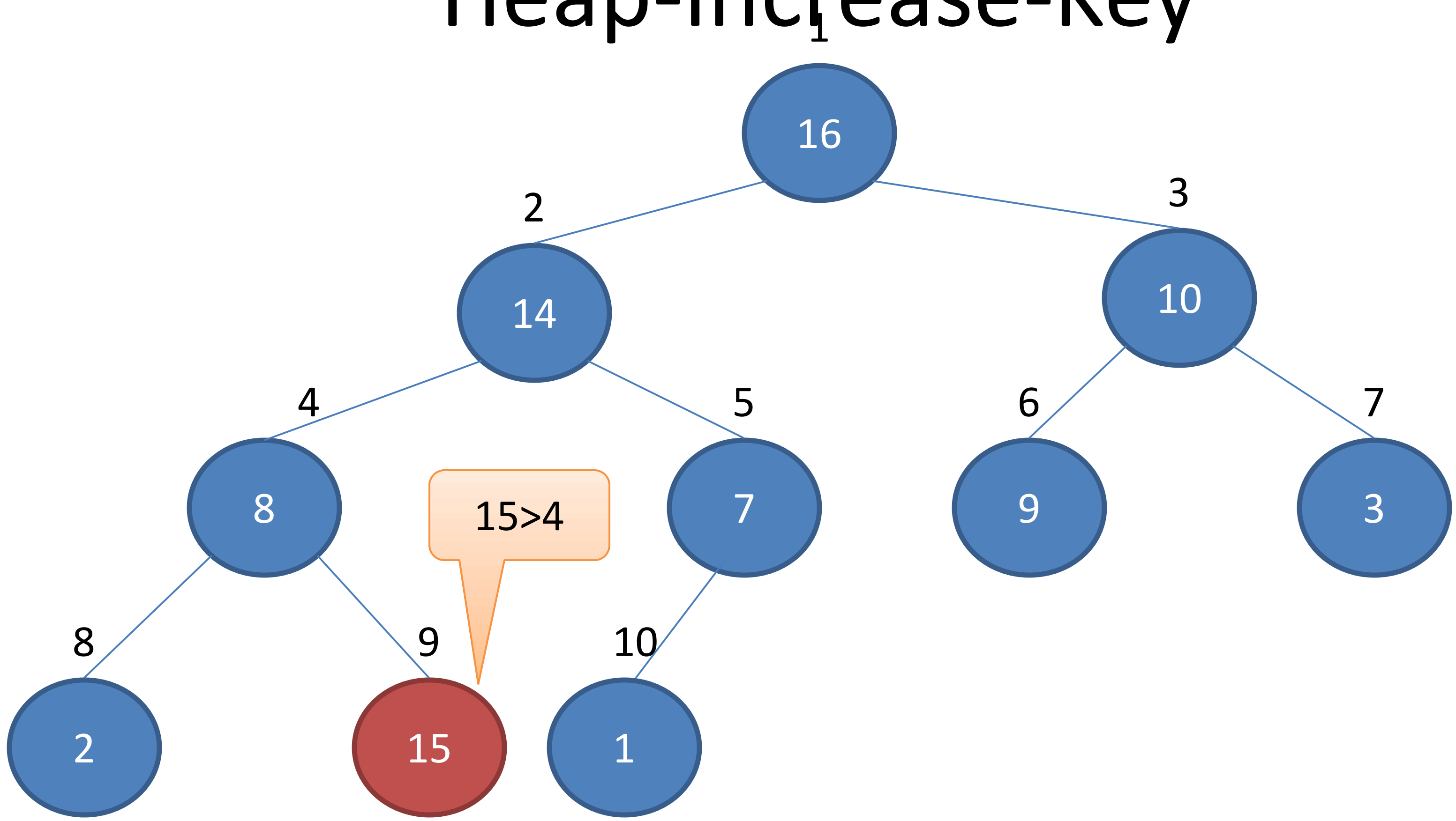
Pseudo-code: Max-Heap-Insert(A,key)
heap-size(A) = heap-size[A]+1
A[heap-size[A]] = $-\infty$
Heap-Increase-Key(A, heap-size[A], key)

Running time = O(lg n)

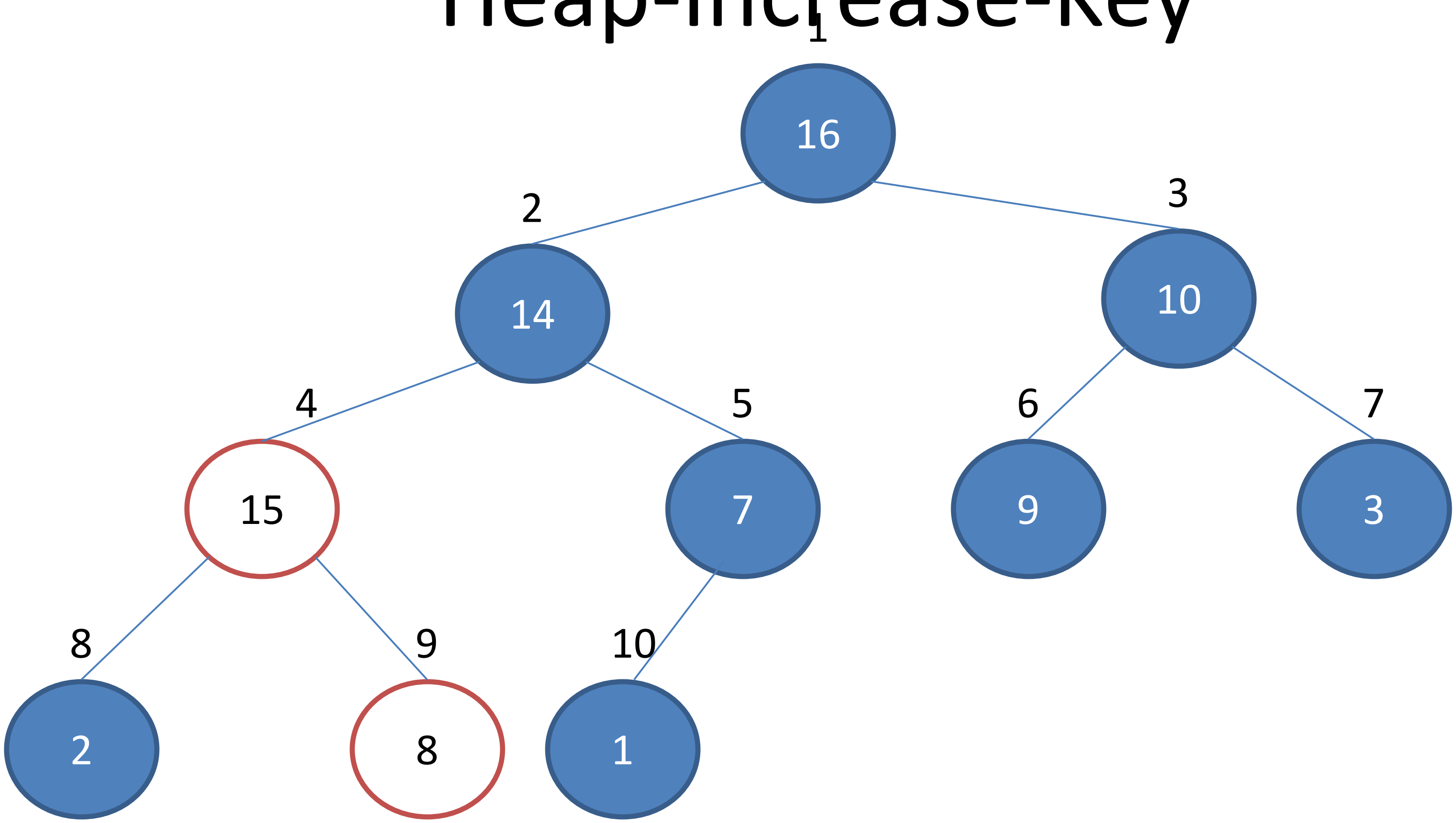
Example: Max-Priority Queue: Heap-Increase-Key



Example: Max-Priority Queue: Heap-Increase-Key



Example: Max-Priority Queue: Heap-Increase-Key



Example: Max-Priority Queue: Heap-Increase-Key