L02.03
Heap (II)

50.004 Introduction to Algorithm
Gemma Roig
(slides adapted from Dr. Simon LUI)
ISTD, SUTD

What is this lecture about

- It is a simpler version of Heap I (L02.02)
- With step by step examples

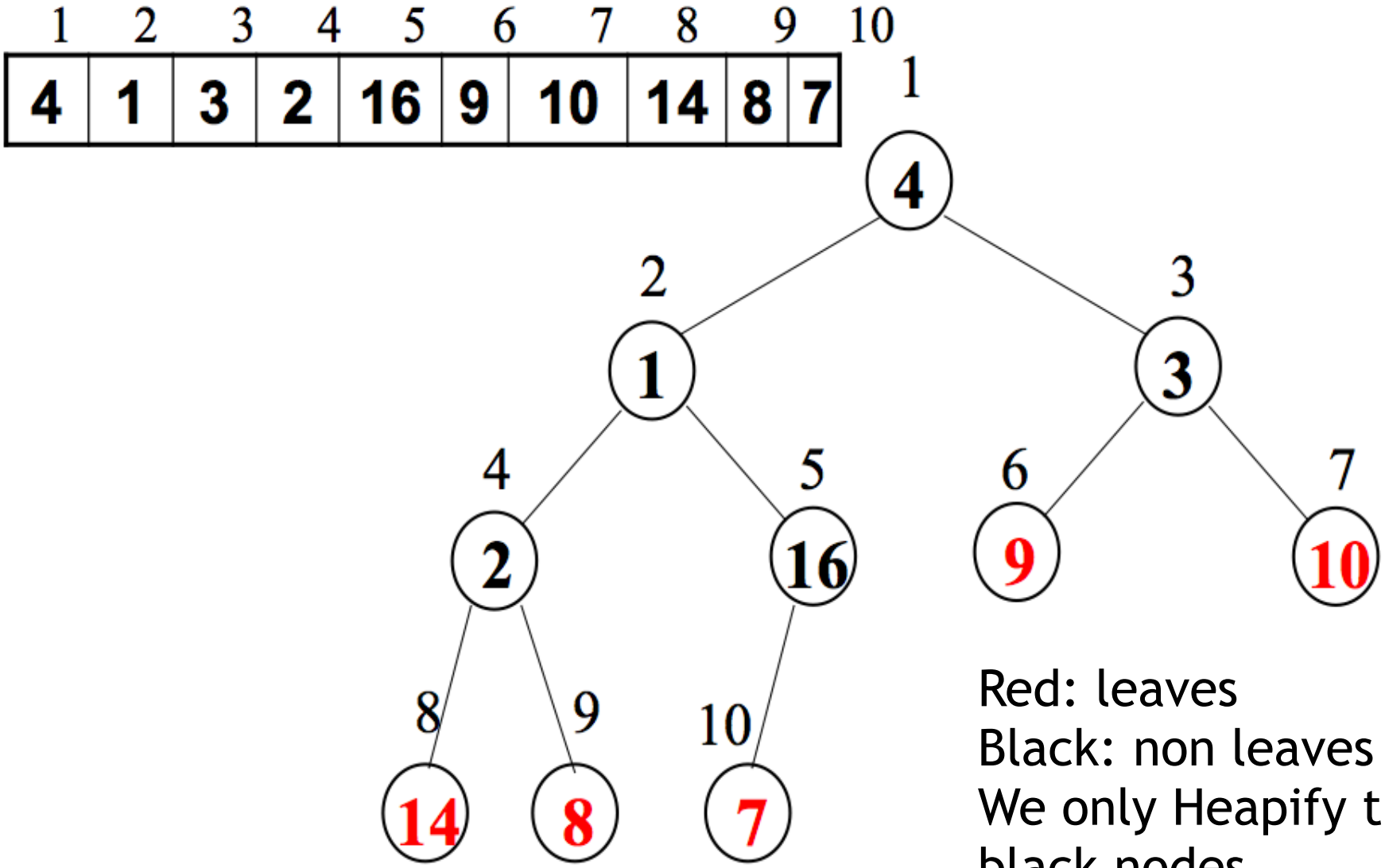
Build Max Heap

Build Max Heap

Build-Max-Heap(A)

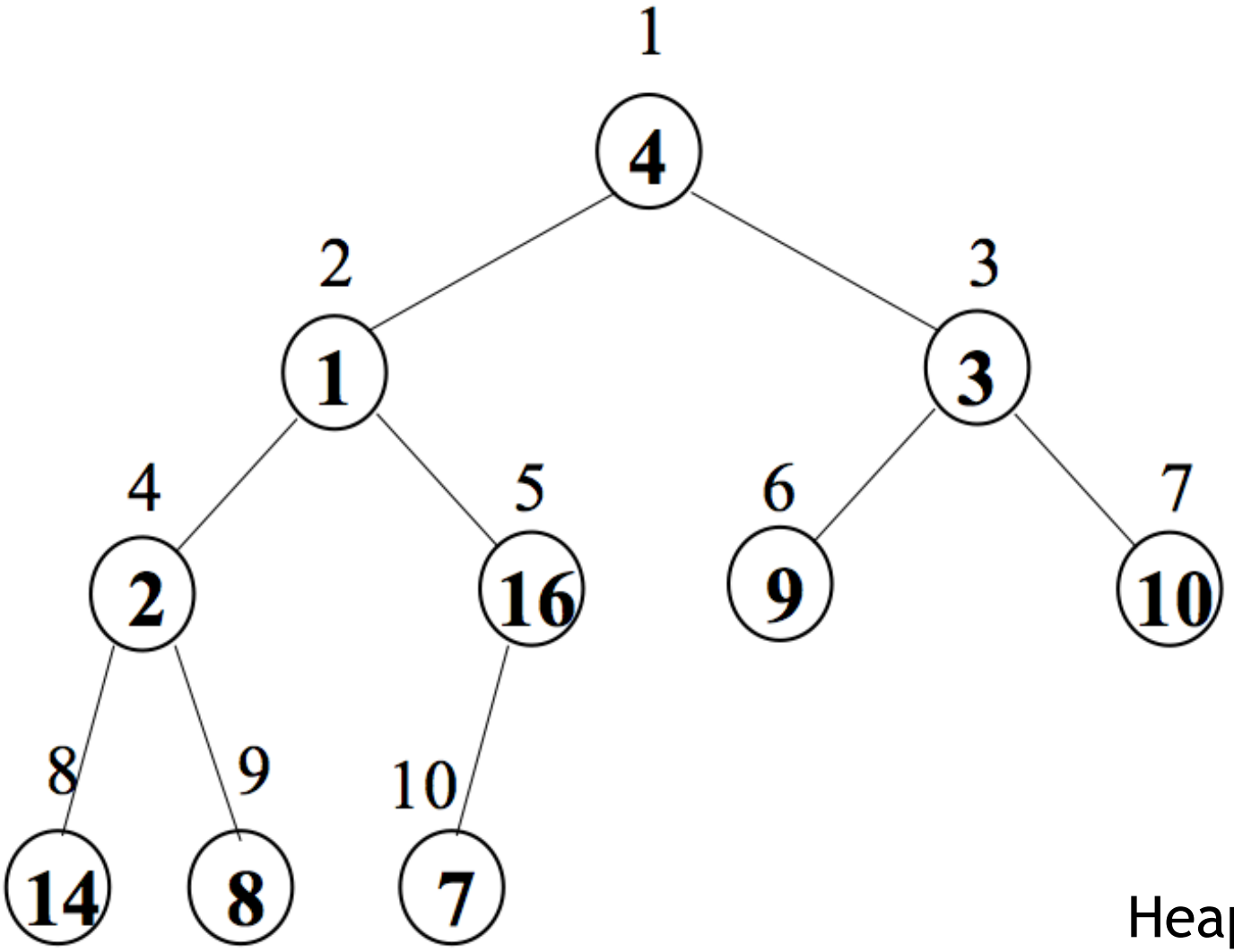
1. *heap-size*[A] ← *length*[A]
2. for i ← ⌊*length*[A]/2⌋ downto 1
3. do Max-Heapify(A,i)

Example: Build_Max_Heap



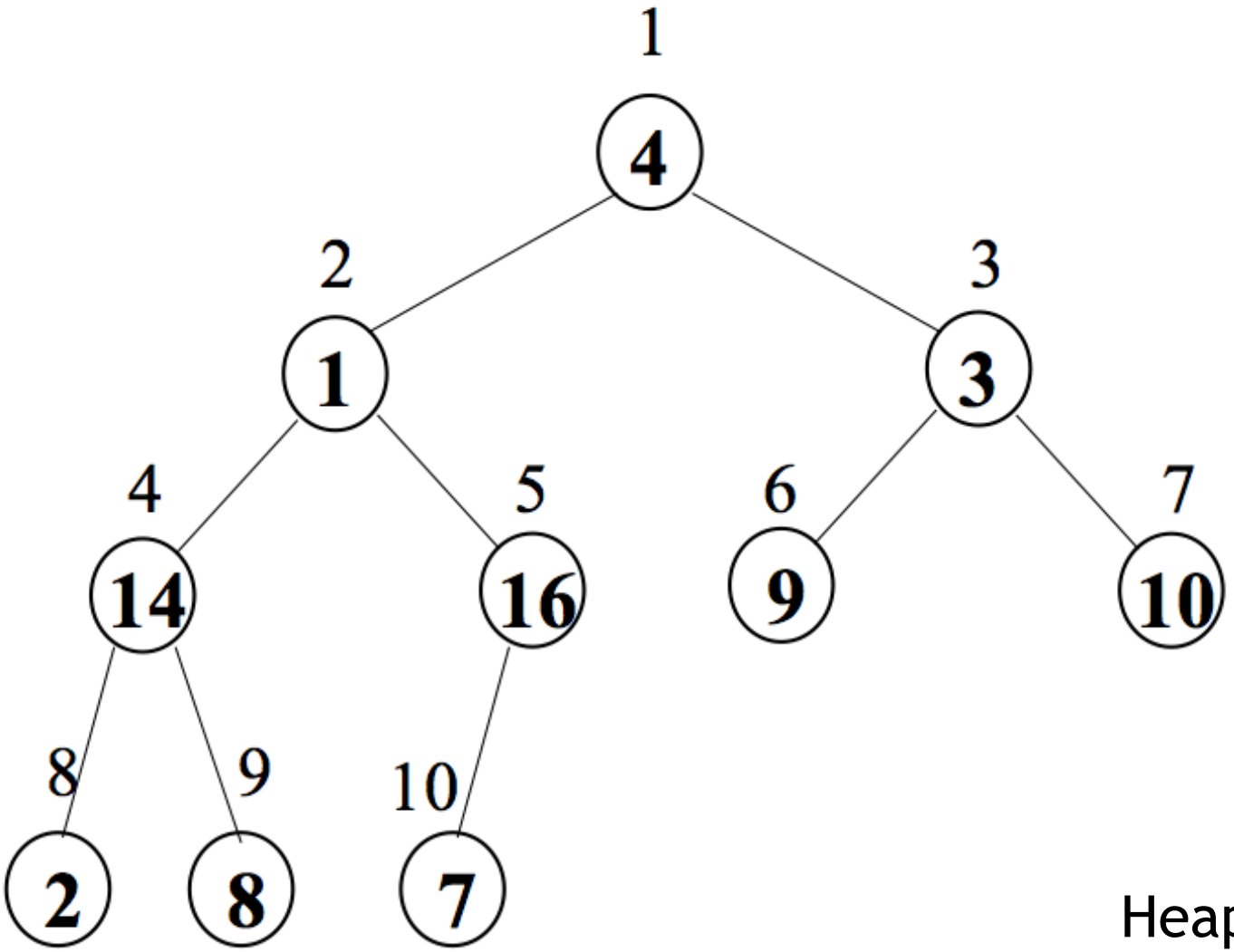
Red: leaves
Black: non leaves
We only Heapify the black nodes

Example: Build_Max_Heap



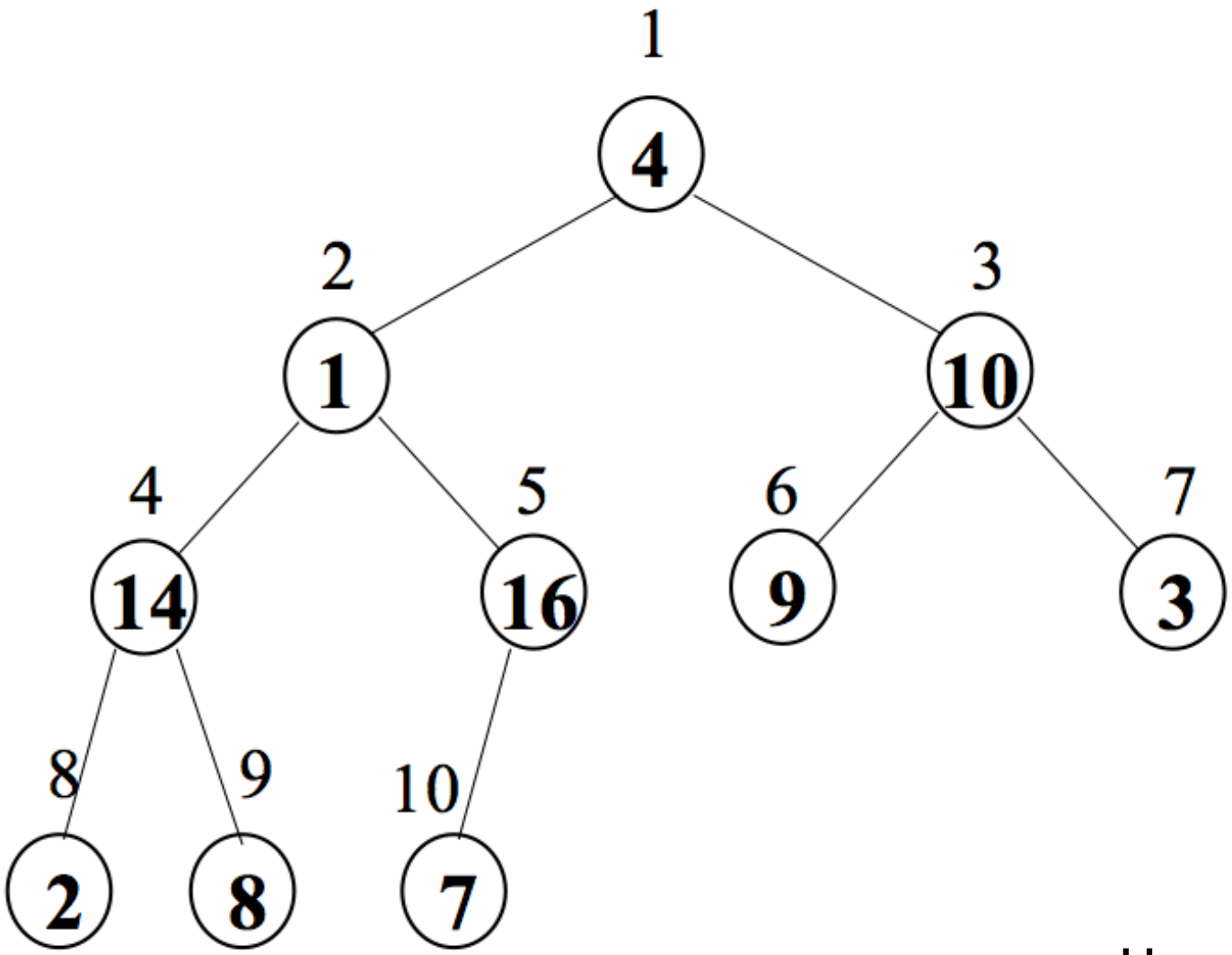
Heapify(A,5)

Example: Build_Max_Heap



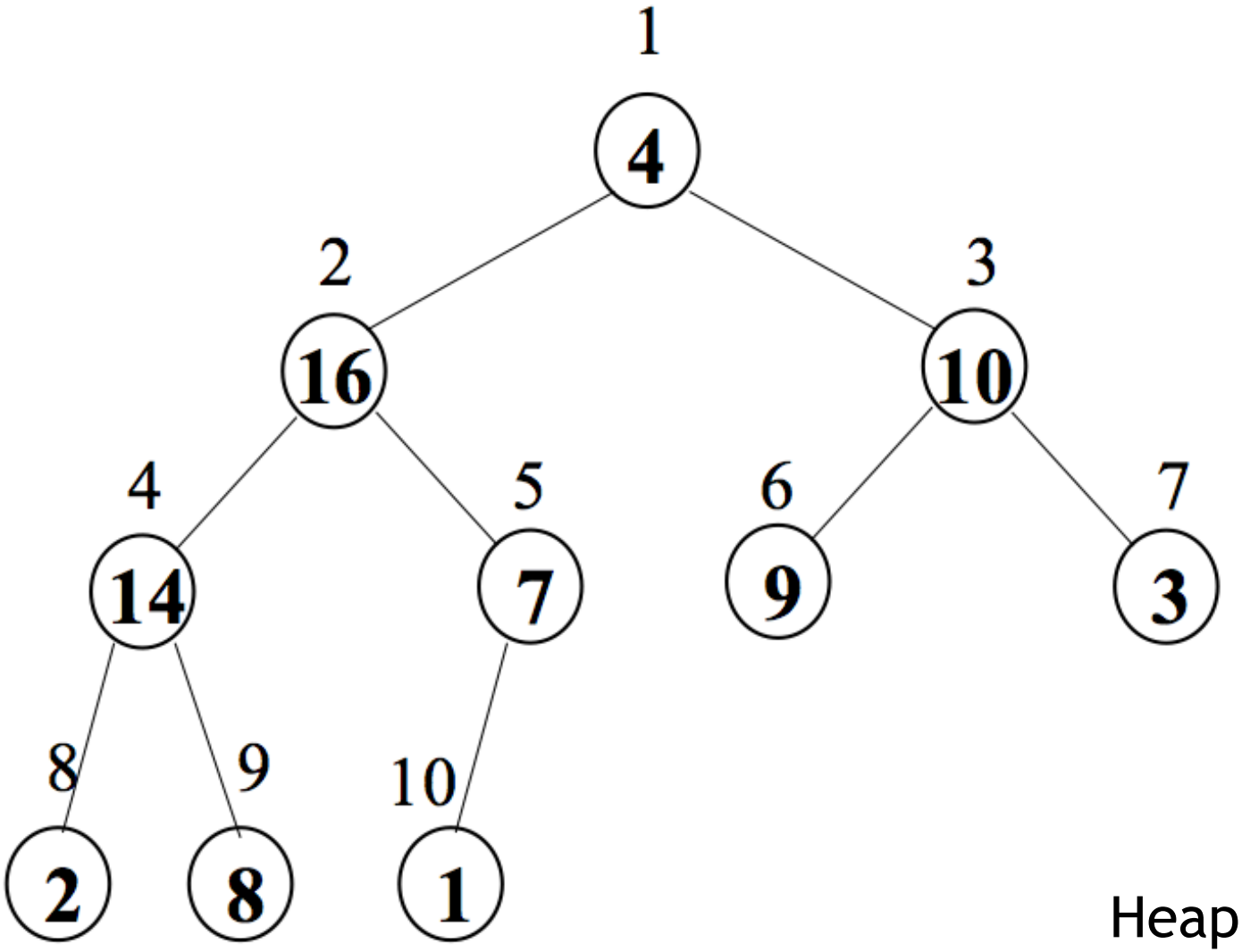
Heapify(A,4)

Example: Build_Max_Heap



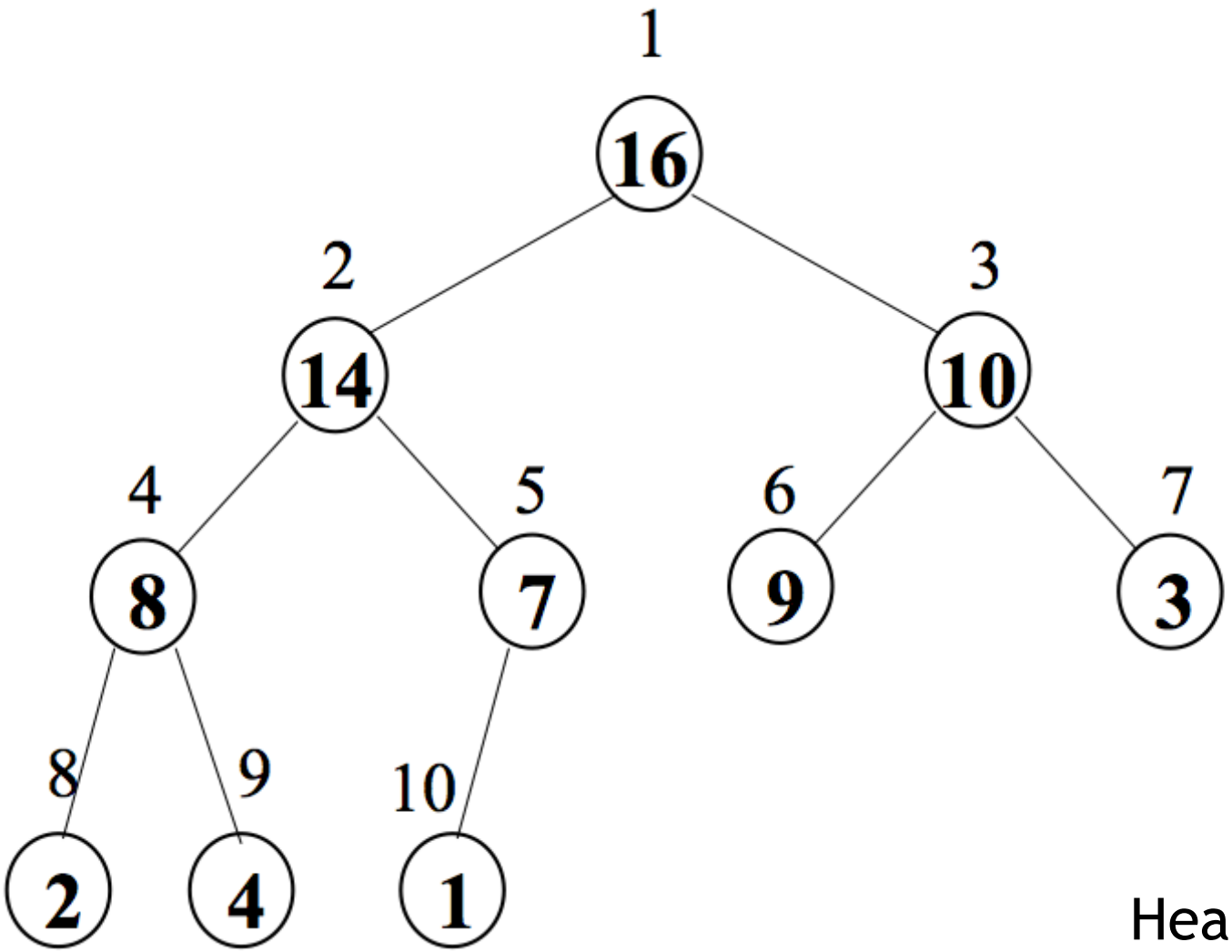
Heapify(A,3)

Example: Build_Max_Heap



Heapify(A,2)

Example: Build_Max_Heap



Heapify(A,1)

Heapify

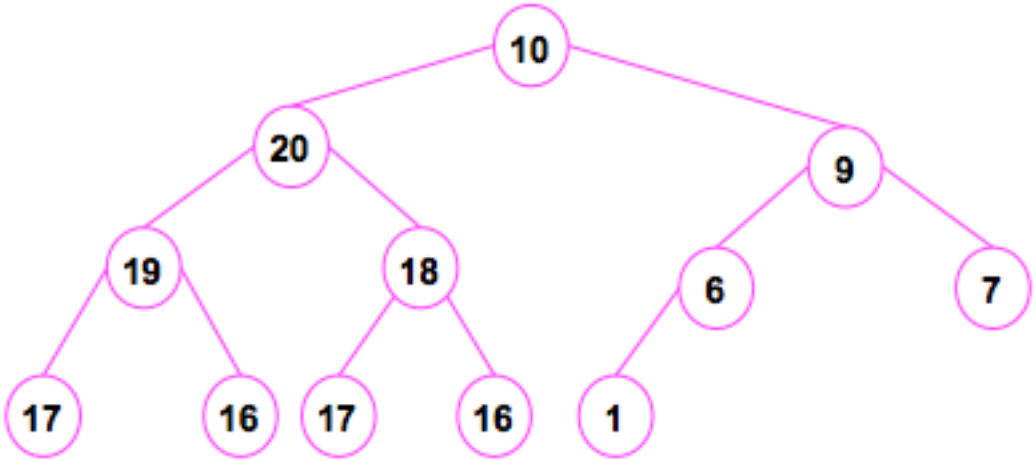
Heapify

Max-Heapify(A,i)

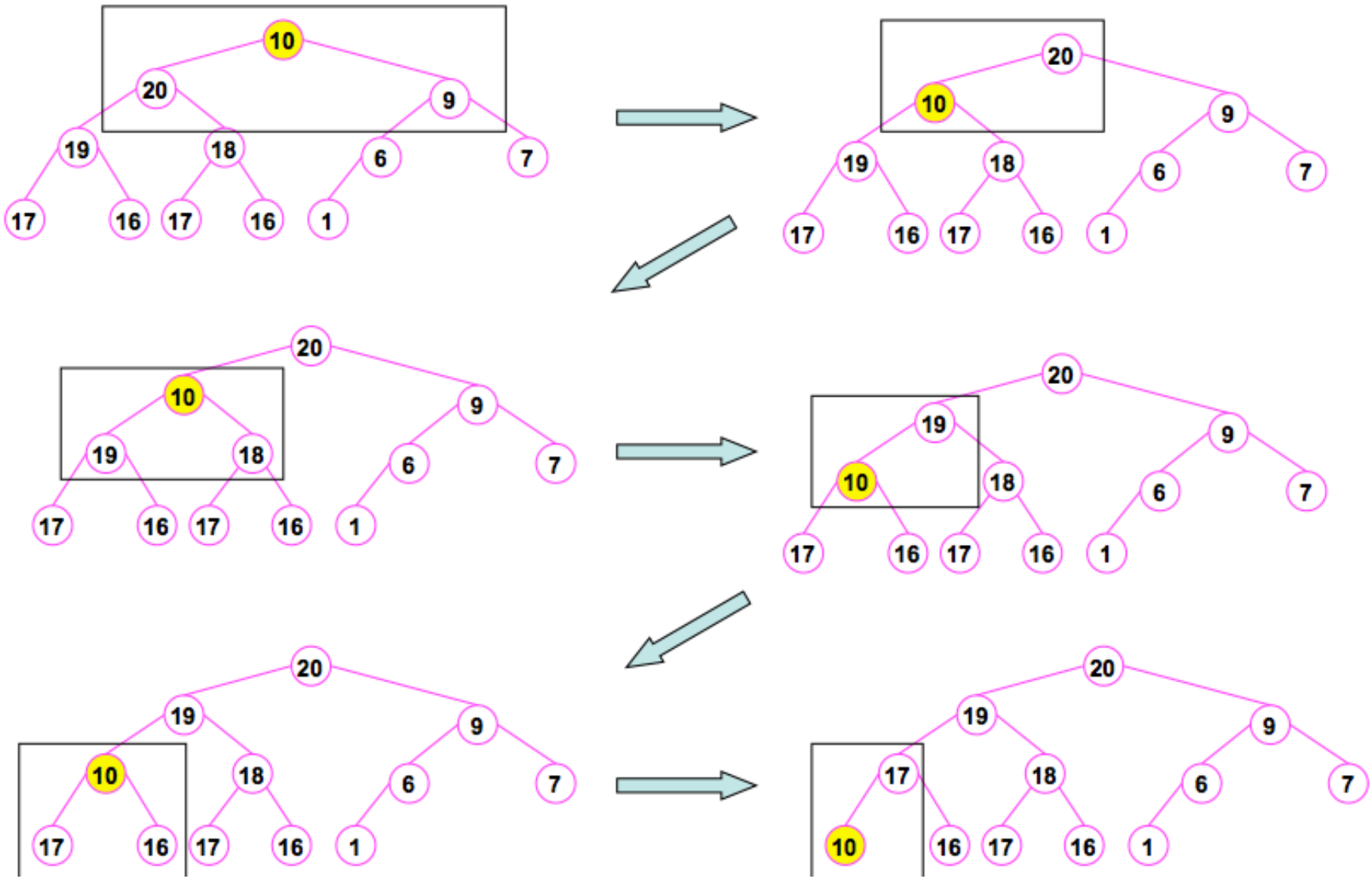
```
1.  l ← Left(i)
2.  r ← Right(i)
3.  if l ≤ heap-size[A] and A[l] > A[i]
4.      then largest ← l
5.      else largest ← i
6.  if r ≤ heap-size[A] and A[r] > A[largest]
7.      then largest ← r
8.  if largest ≠ i
9.      then swap( A[i], A[largest])
        Max-Heapify(A, largest)
```

Example: Heapify

- Node(1) violates the max heap rule



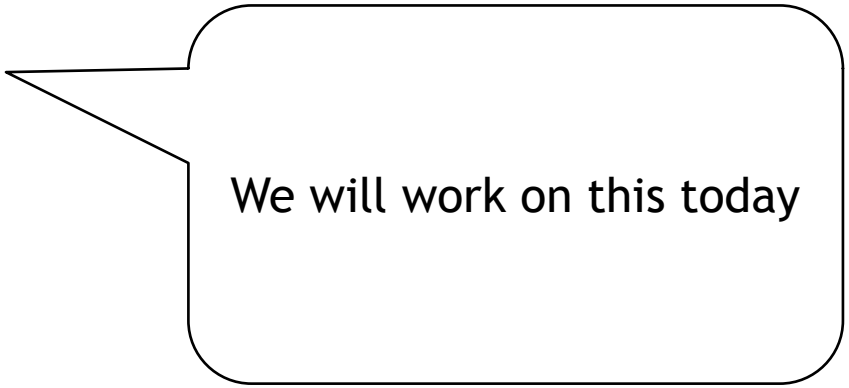
Example: Heapify



Priority Queue

Priority Queue

- There are many kinds of priority Queue
  - Array Priority Queue
  - List Priority Queue
  - Heap Priority Queue



Priority Queue operations

Insert(A,key): insert a key, then heapify and maintain the heap structure

Max(A): returns the largest key

Extract-Max(A): returns the largest key, remove it, then heapify and maintains the heap property

Increase-Key(A,i,key): Increase the key of index i

Priority Queue – the algorithm

Max (A)

1. **return** A[1]

Extract-Max(A)

1. if *heap-size*[A] < 1
2. **then error** "heap underflow"
3. *max* ← A[1]
4. A[1] ← A[*heap-size*[A]]
5. *heap-size*[A] ← *heap-size*[A] - 1
6. Max-Heapify(A,1)
7. **return** *max*

Priority Queue – the algorithm

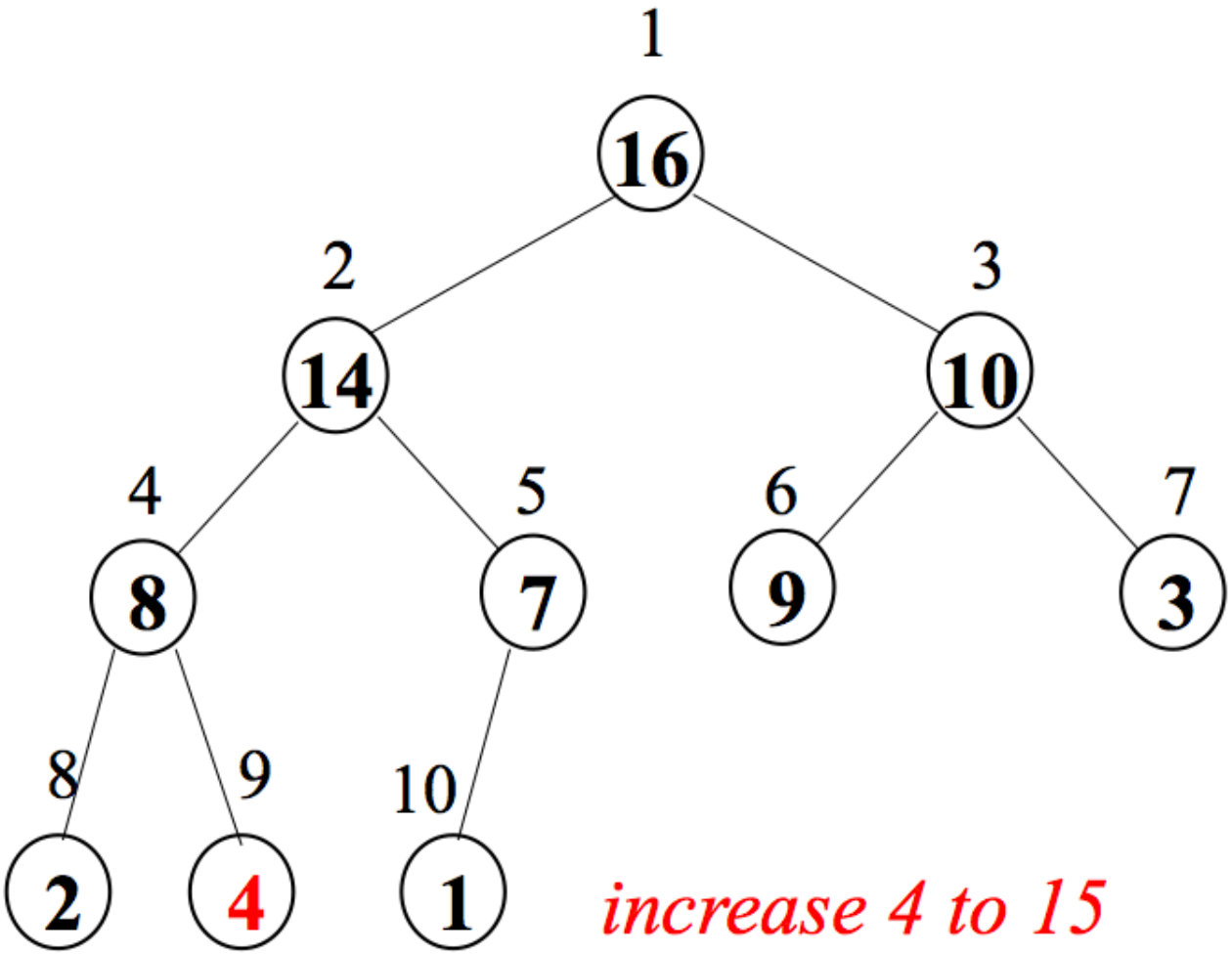
Increase-Key(A,i,key)

1. **if** key < A[i]
2. **then error** "new key is smaller than current key"
3. A[i] ← key
4. **while** i > 1 and A[Parent(i)] < A[i]
5. **do** exchange A[i] ←→ A[Parent(i)]
6. i ← Parent(i)

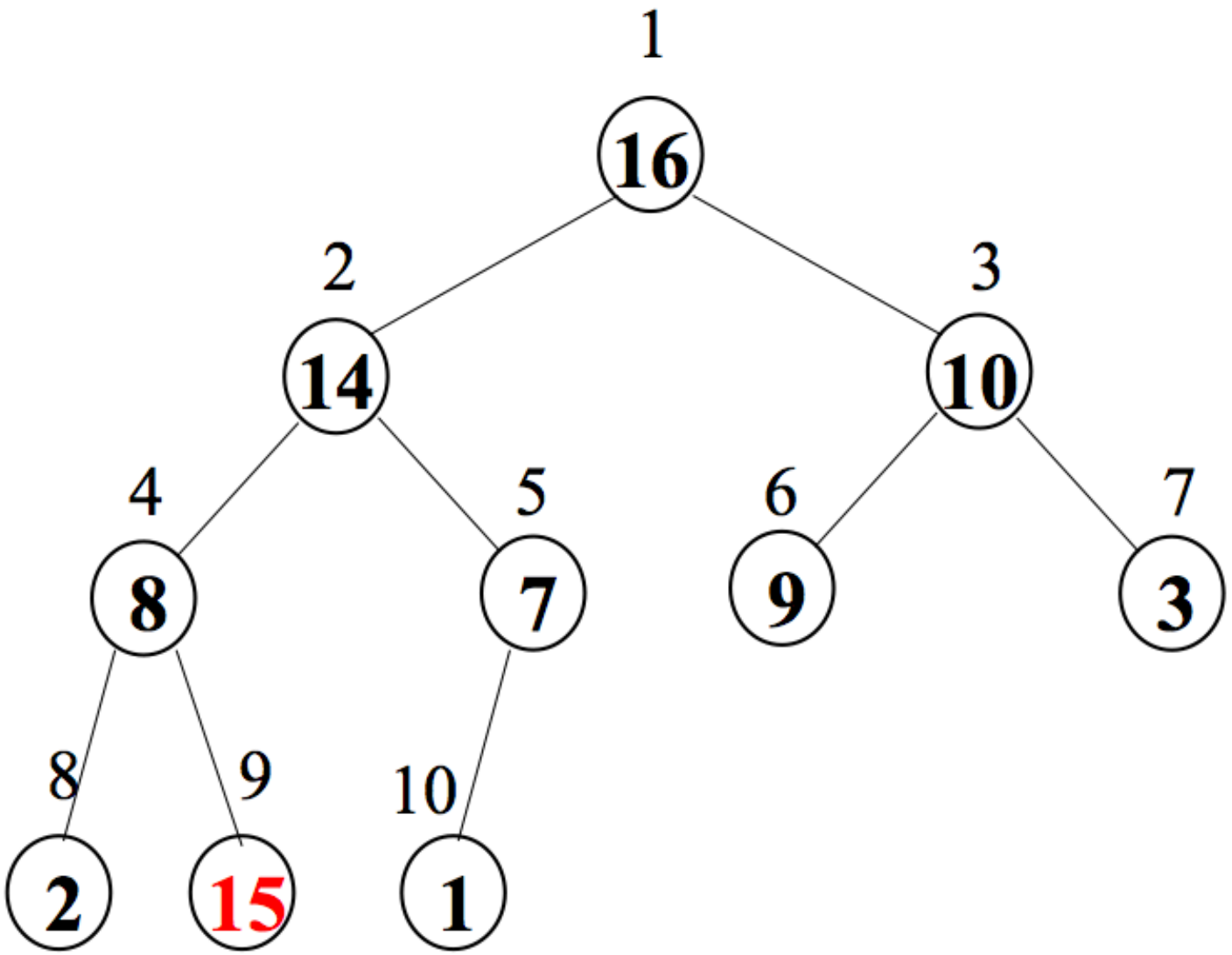
Insert(A,key)

1. *heap-size*[A] ← *heap-size*[A]+1
2. A[*heap-size*[A]] ← $-\infty$
3. Heap-Increase-Key(A, *heap-size*[A], key)

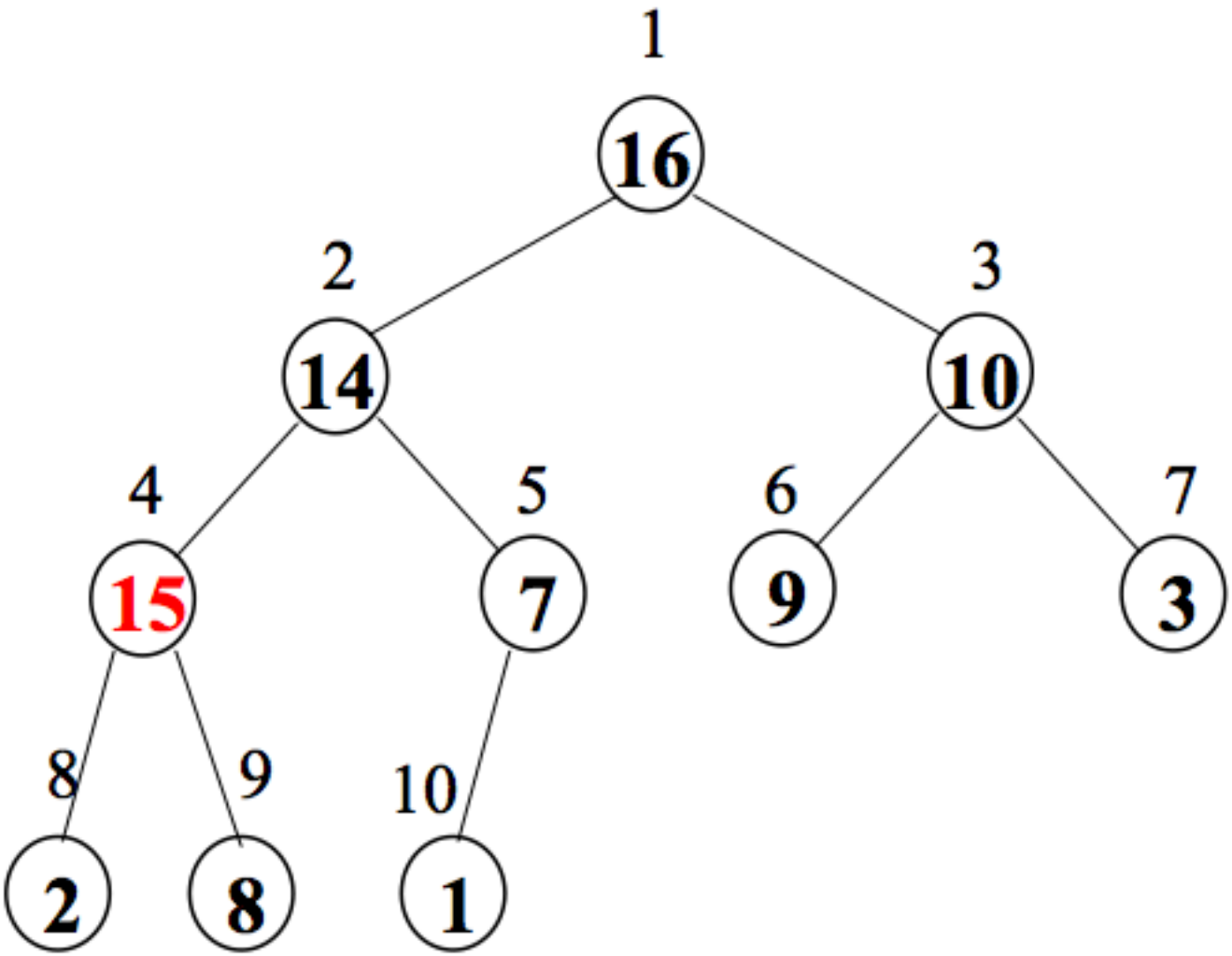
Example: increase key



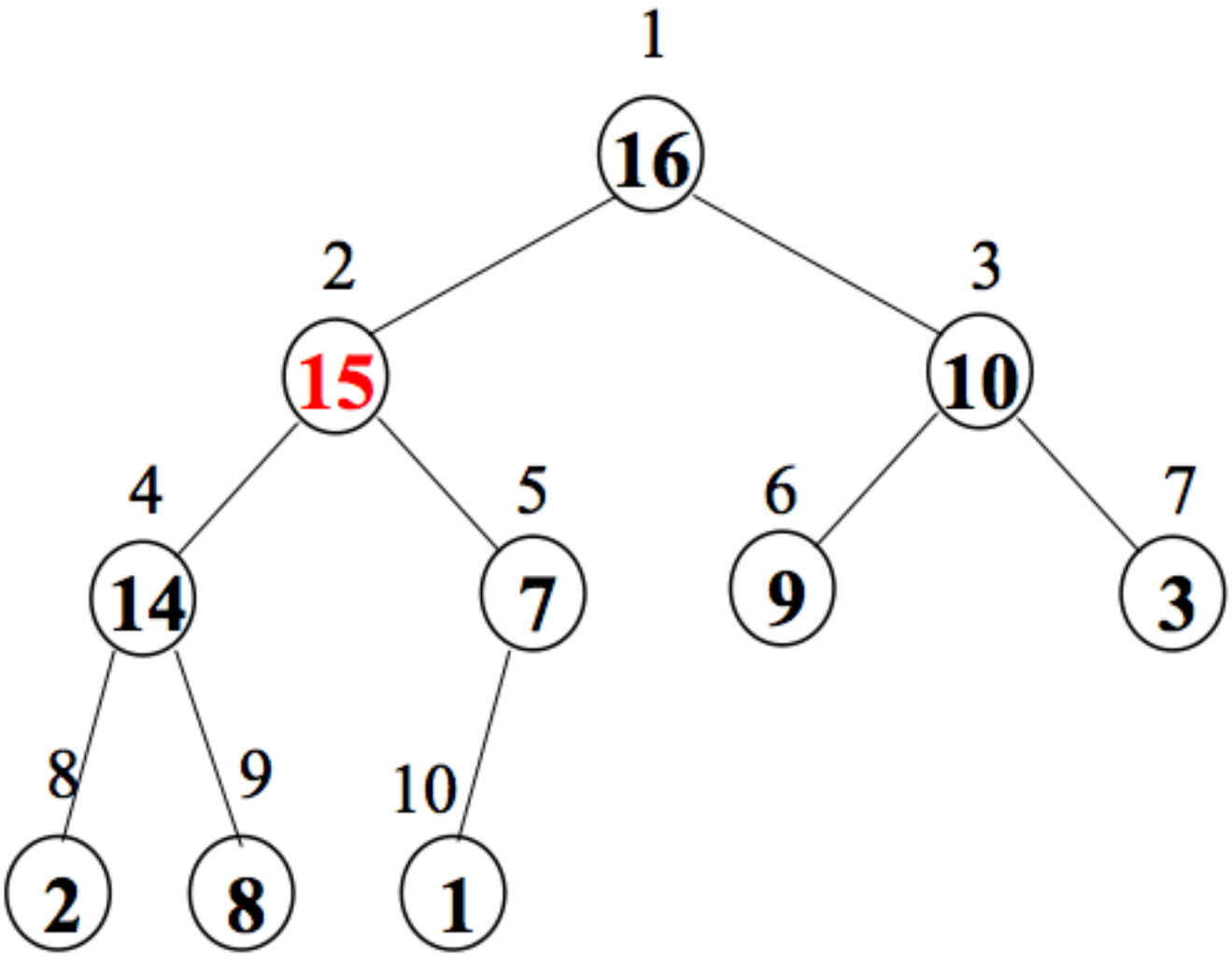
Example: increase key



Example: increase key



Example: increase key



Heap sort

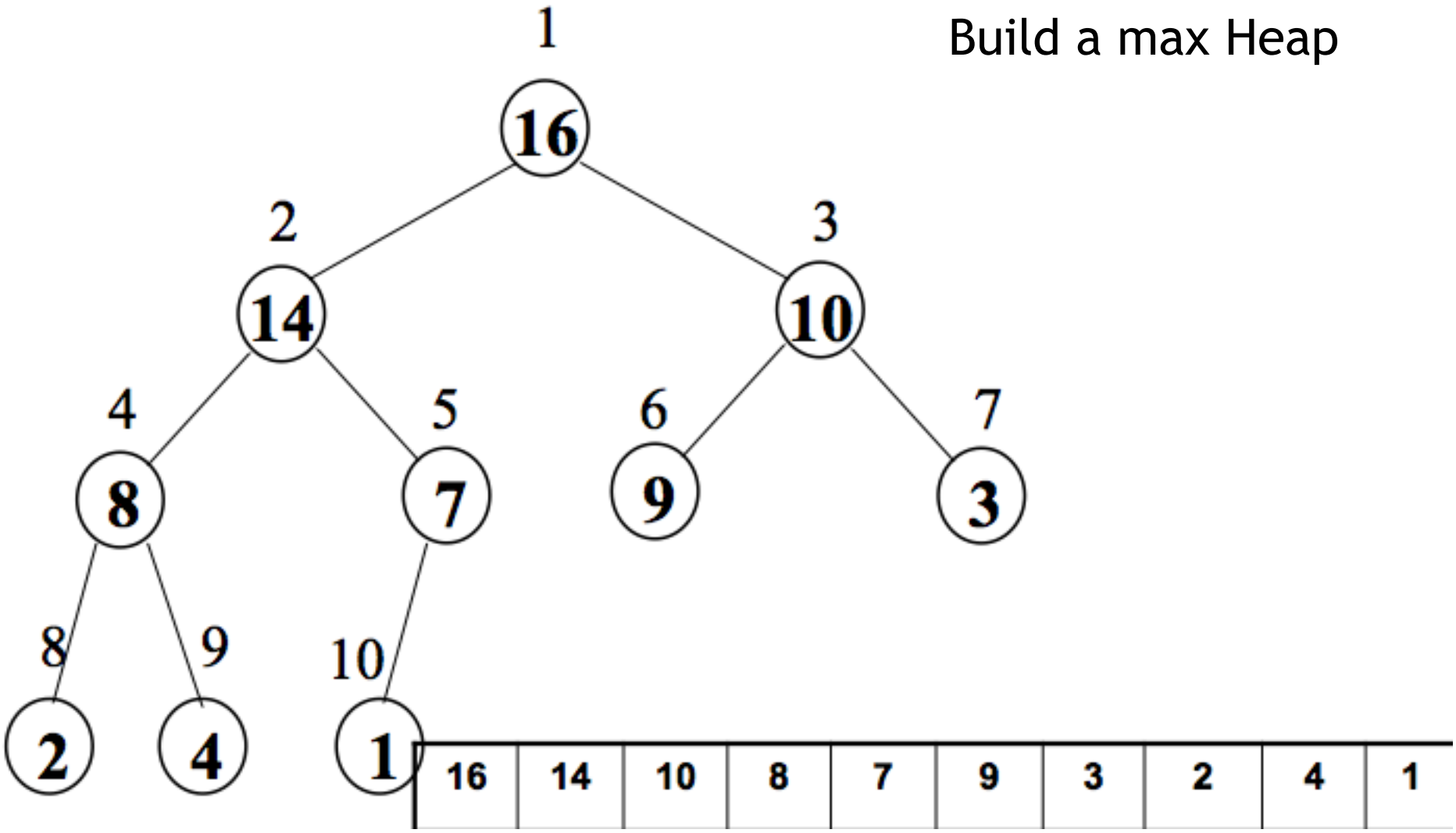
Heap sort

Heapsort(A)

1. Build-Max-Heap(A)
2. for i ← *length*[A] **downto** 2
3. **do** exchange A[1] ↔ A[i]
4. *heap-size*[A] ← *heap-size*[A]-1
5. Max-Heapify(A,1)

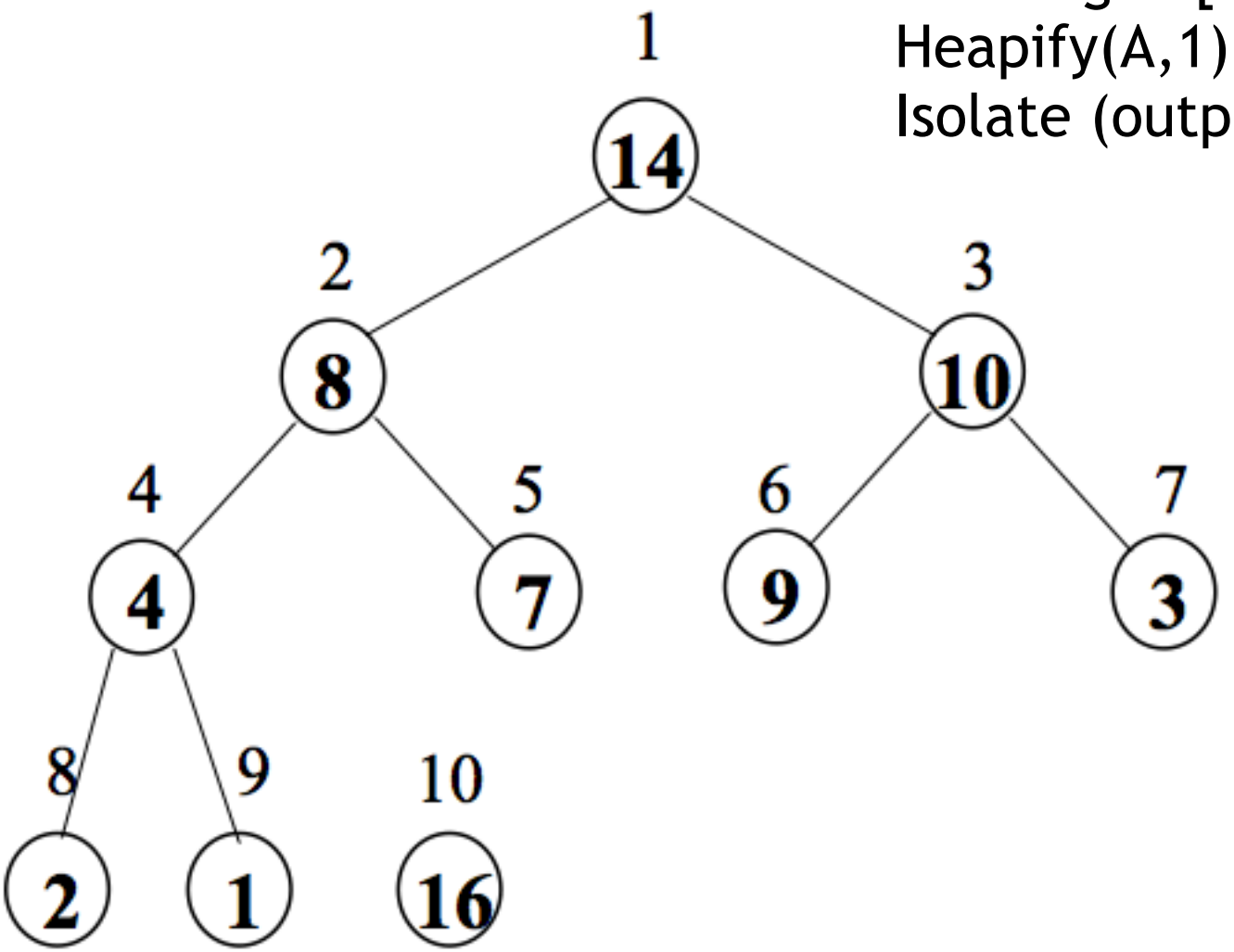
Example: Heap sort

Build a max Heap



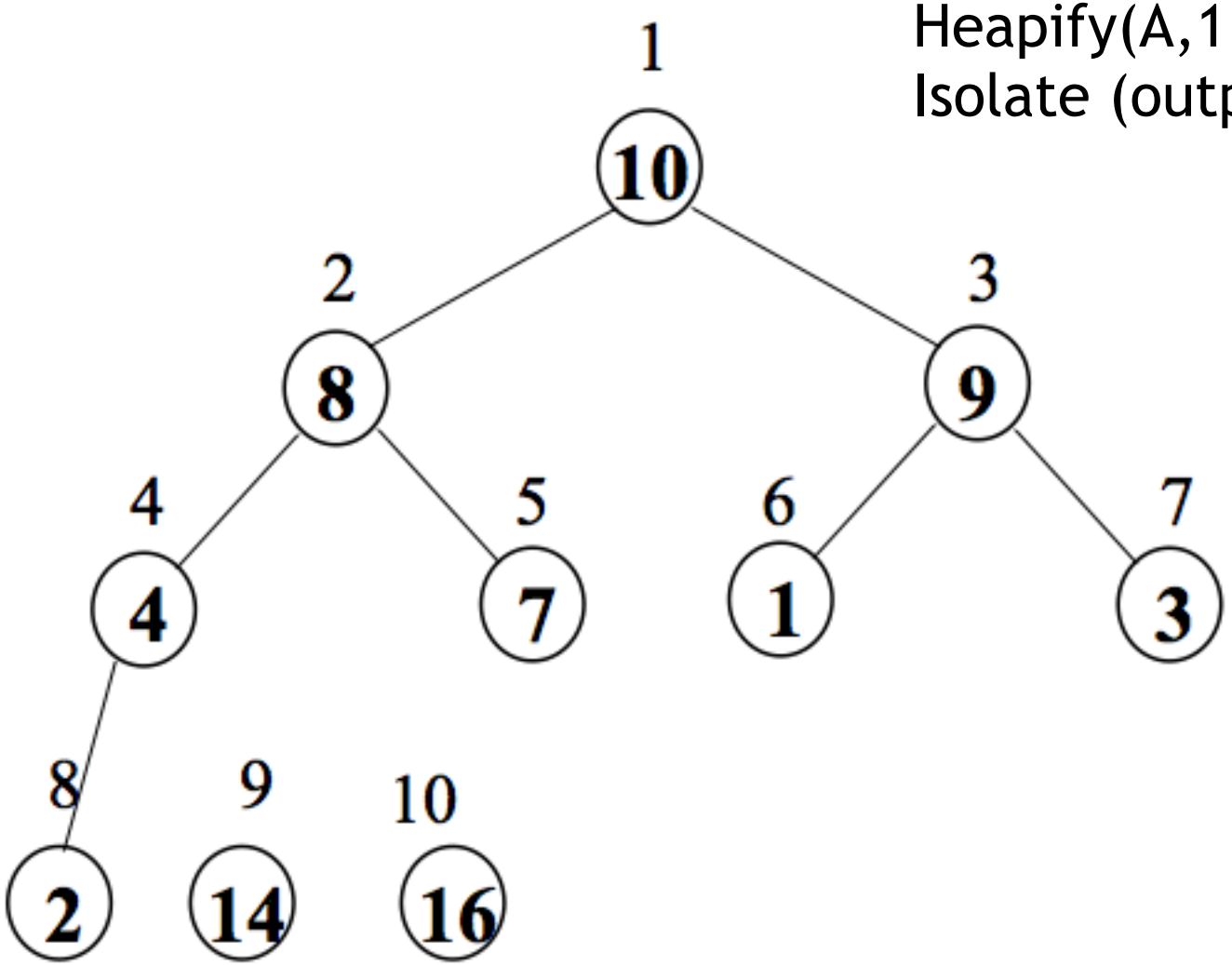
Example: Heap sort

Exchange A[1] and A[10]
Heapify(A,1)
Isolate (output) A[10]



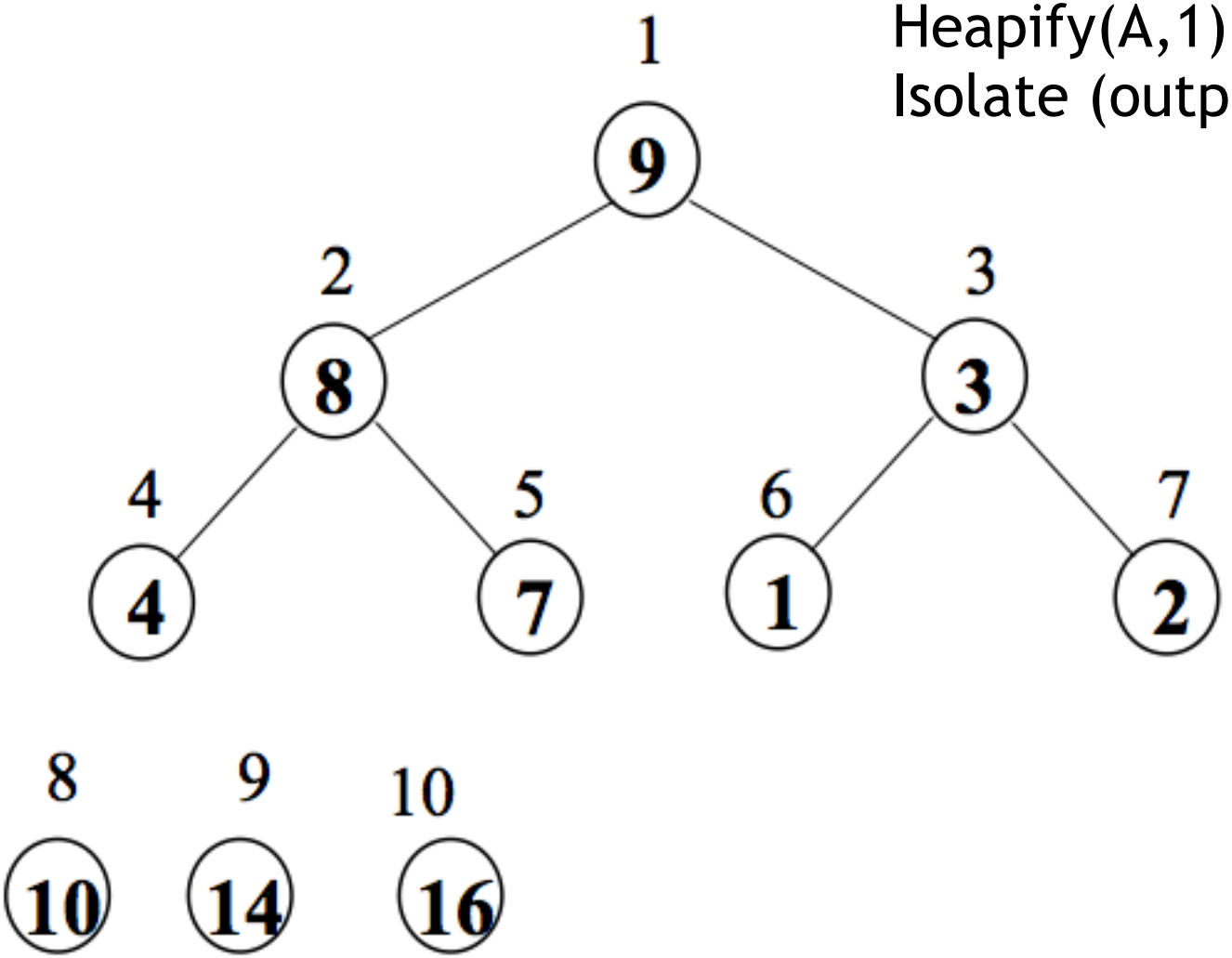
Example: Heap sort

Exchange A[1] and A[9]
Heapify(A,1)
Isolate (output) A[9]



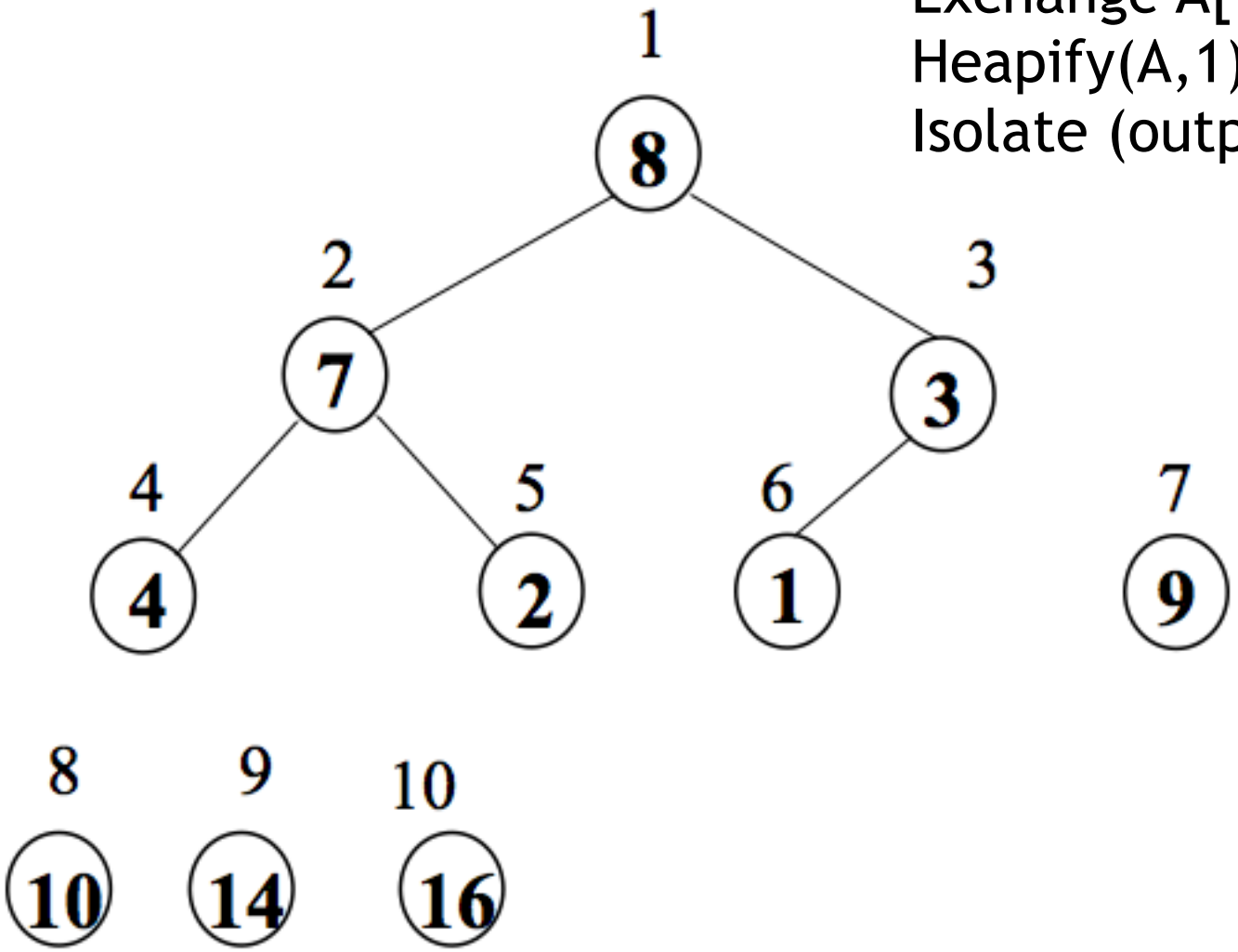
Example: Heap sort

Exchange A[1] and A[8]
Heapify(A,1)
Isolate (output) A[8]



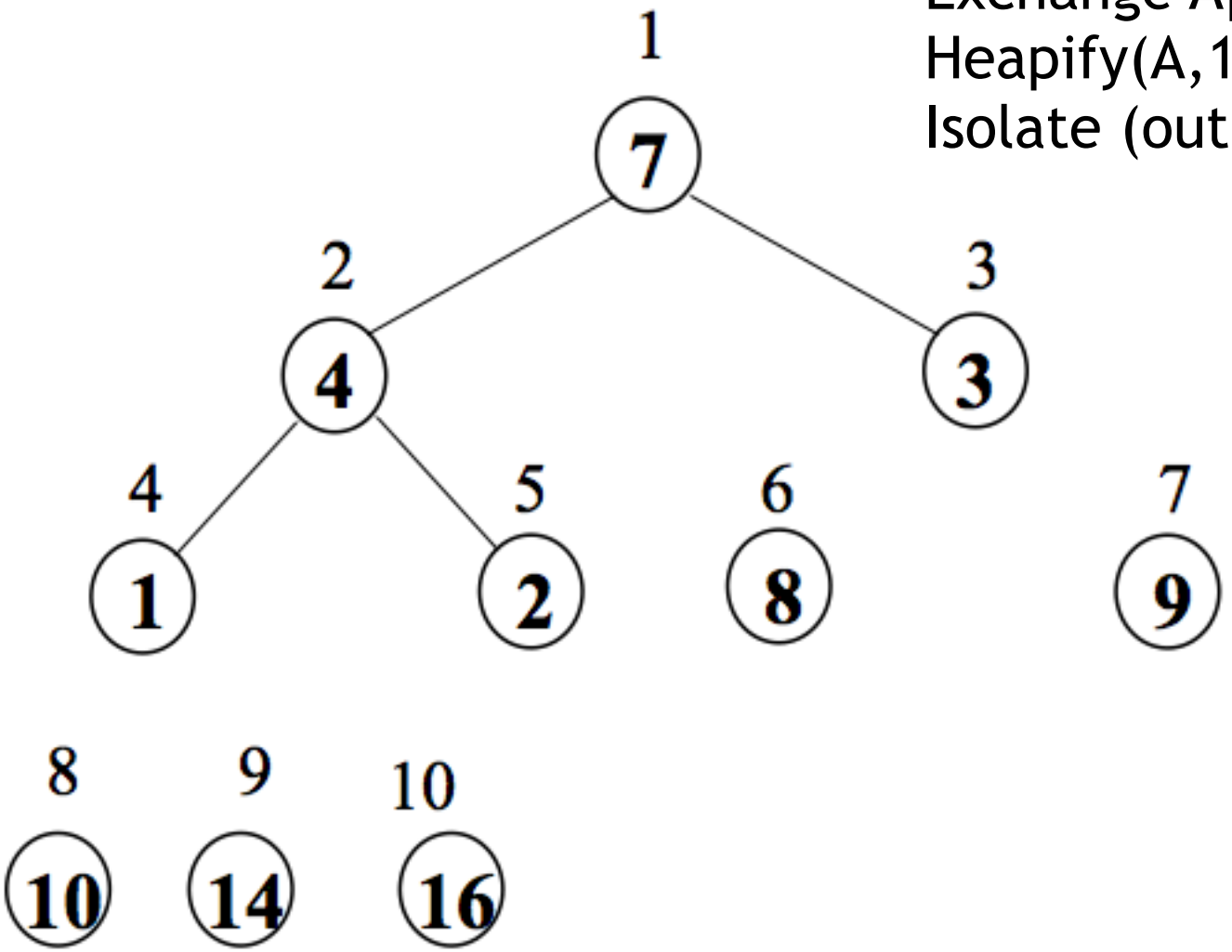
Example: Heap sort

Exchange A[1] and A[7]
Heapify(A,1)
Isolate (output) A[7]



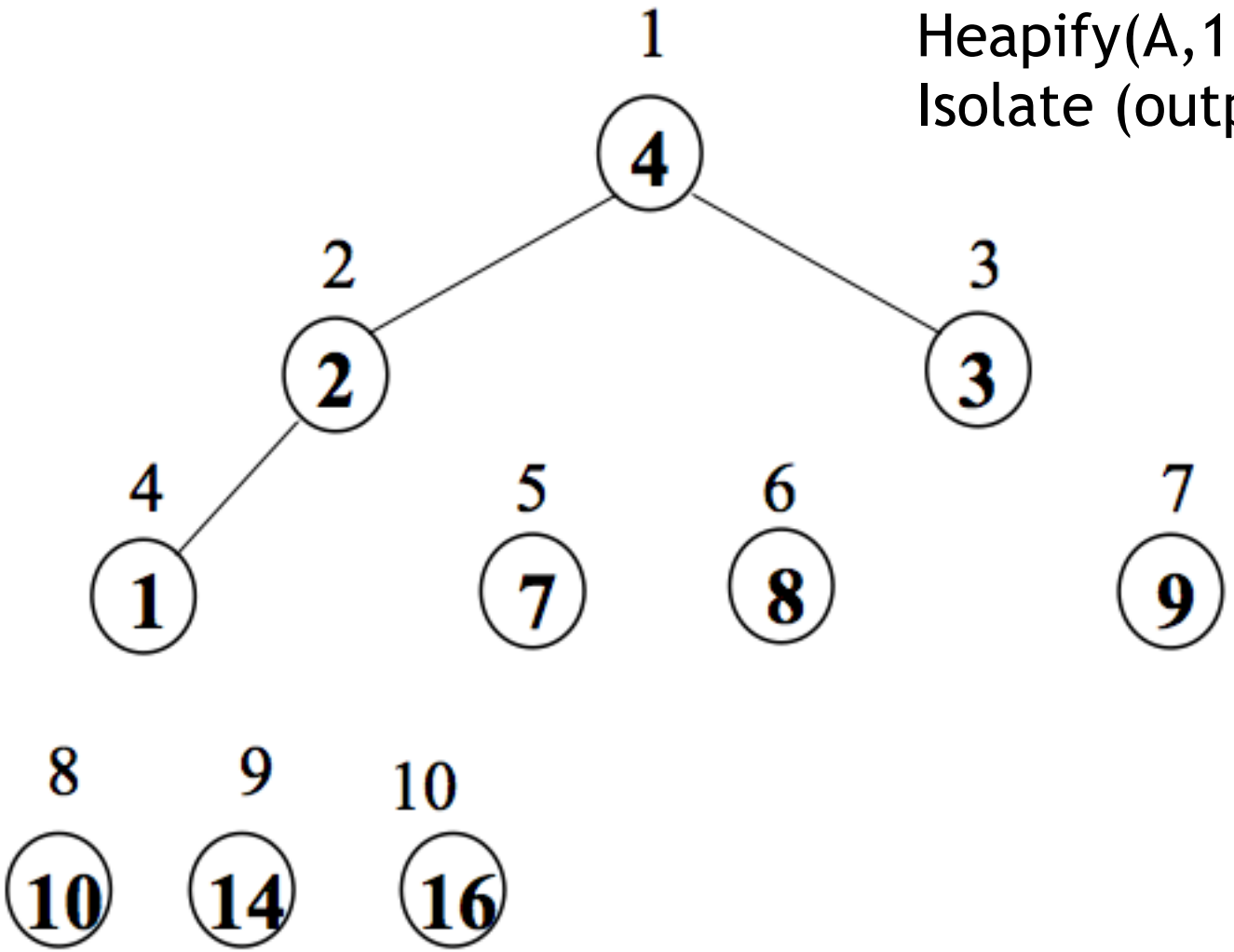
Example: Heap sort

Exchange A[1] and A[6]
Heapify(A,1)
Isolate (output) A[6]

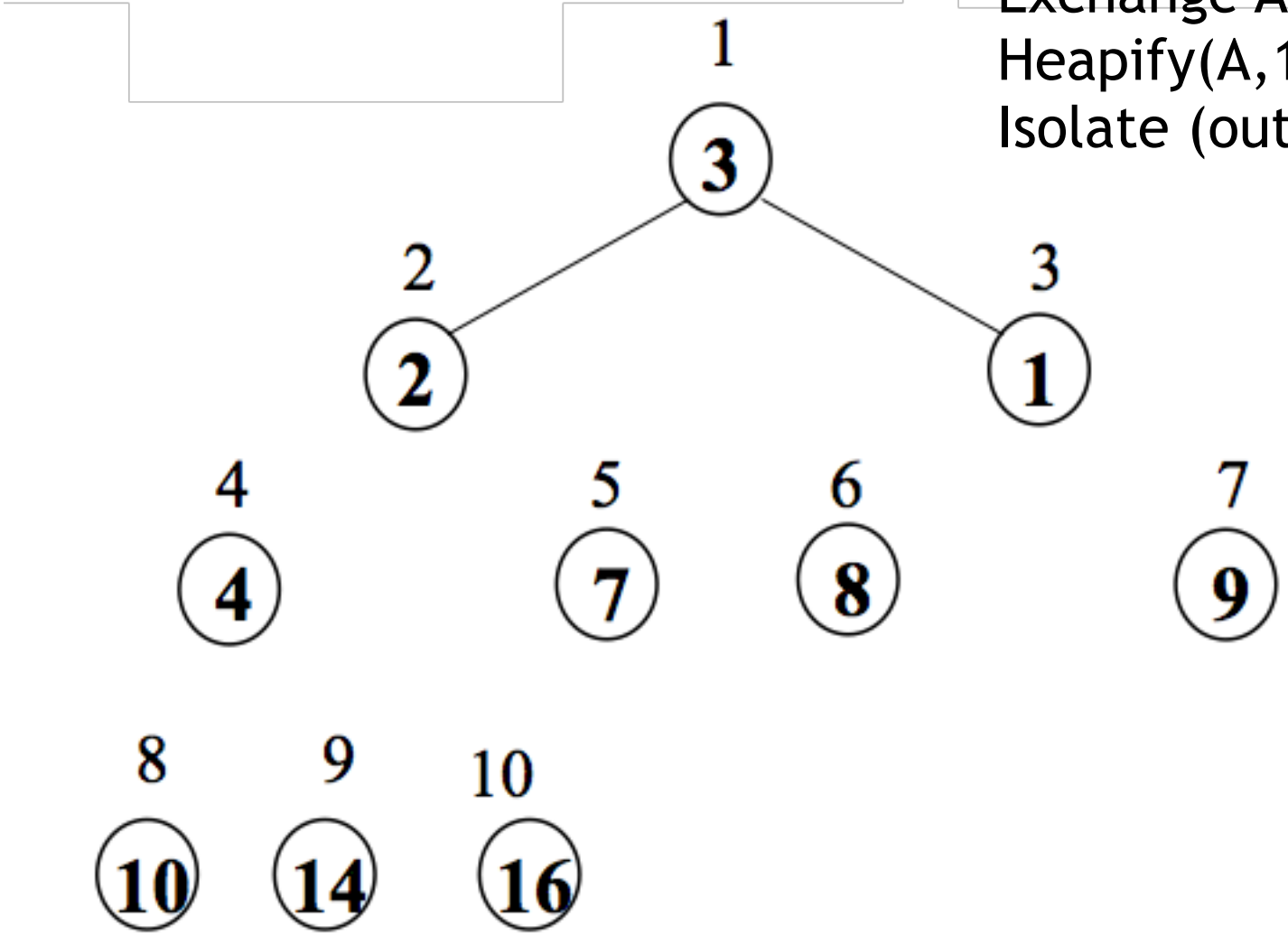


Example: Heap sort

Exchange A[1] and A[5]
Heapify(A,1)
Isolate (output) A[5]

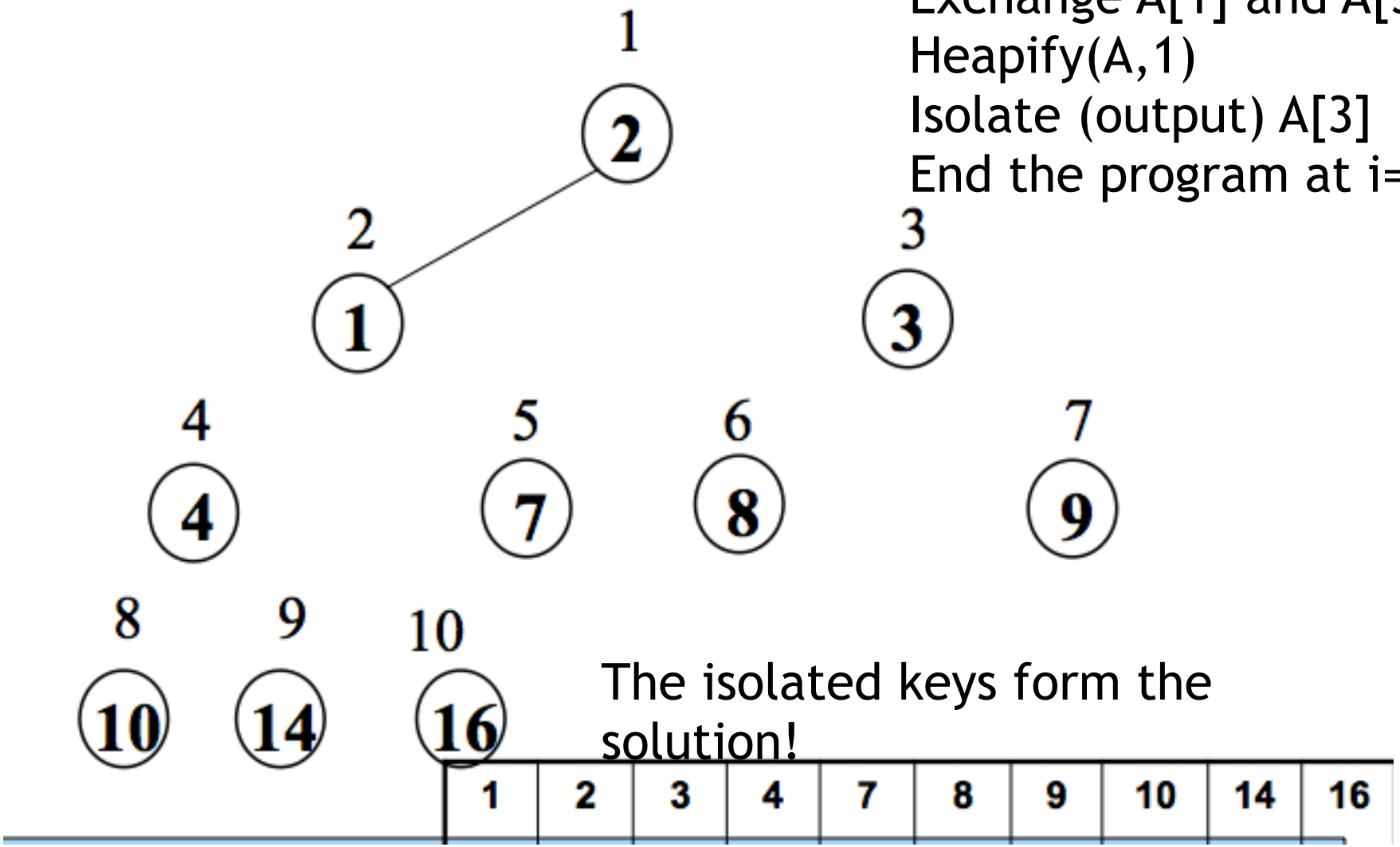


Example: Heap sort



Exchange A[1] and A[4]
Heapify(A,1)
Isolate (output) A[4]

Example: Heap sort



Exchange A[1] and A[3]
Heapify(A,1)
Isolate (output) A[3]
End the program at i=2

The isolated keys form the solution!

Application

Application of Priority Queue

- Used as stack
  - For Dijkstra's algorithm (find the shortest path from 1 node to another) – week 9
  - For Prim's algorithm (find the minimum spanning tree) – week 6

Application of Heap

- To implement a priority Queue
- To implement heapsort
- Computer virtual memory