Priority Queue

Heap

- A heap(binary) is a nearly complete binary tree.
- An Array A that represents a heap is an object with two attributes
  - A. *length* : gives the number of element in the array.
  - A.*heap-size* : the number elements that can be stored within array A.
  - 0 ≤ A.*heapsize* ≤ A.*length*.

There are two kinds of heap

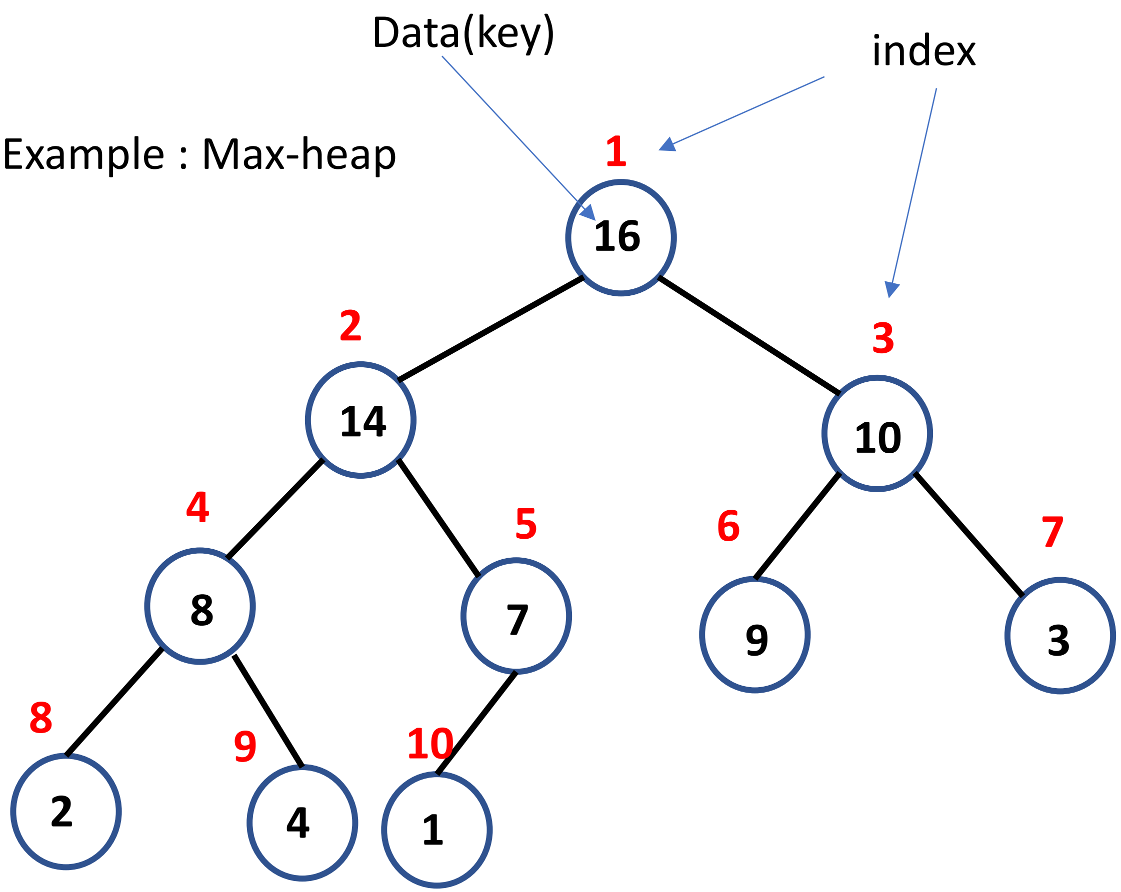
- Max-heap
- Min-heap

- A heap is said to be **max-heap**, if every node *i* other than root ***A[parent(i)] ≥ A[i]*** i.e., the value of a node is at most the value of its parent. i.e., the largest element in a max-heap is stored at the root , and the subtree rooted at a node contains values no larger than that contained at the node itself.

- A heap is said to be **min-heap**, if every node *i* other than root **A[parent(i)] ≤ A[i]** i.e., the smallest element is at the root.

Heaps are used to implement Priority Queue

Example : Max-heap



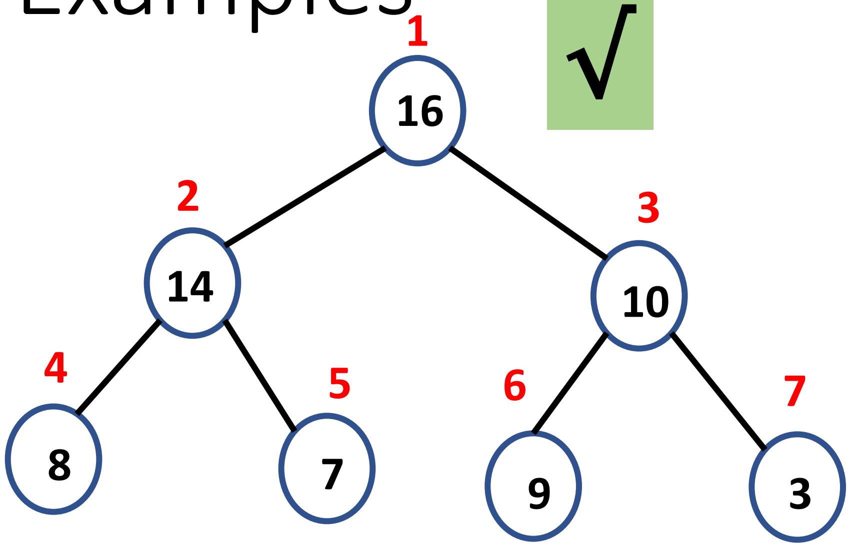
1	2	3	4	5	6	7	8	9	10
16	14	10	8	7	9	3	2	4	1

Parent (i) is stored at index ***floor(i/2)***

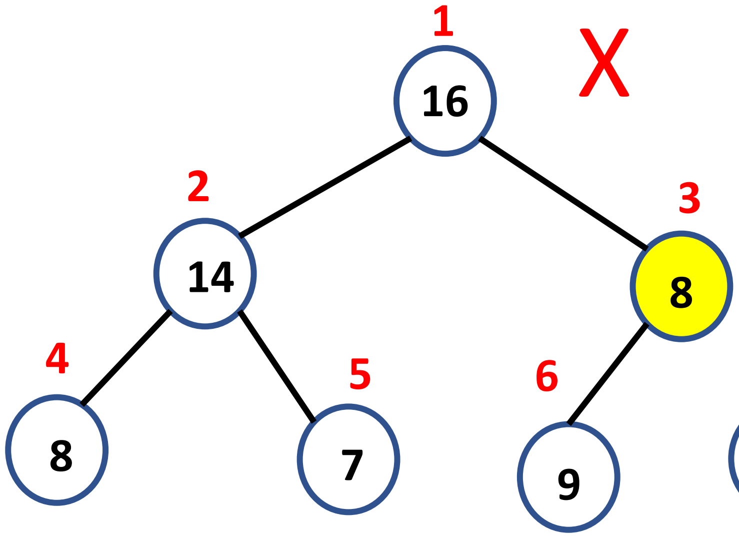
Left child (i) is stored at index ***2*i***

Right child (i) is stored at index ***2 * i + 1***

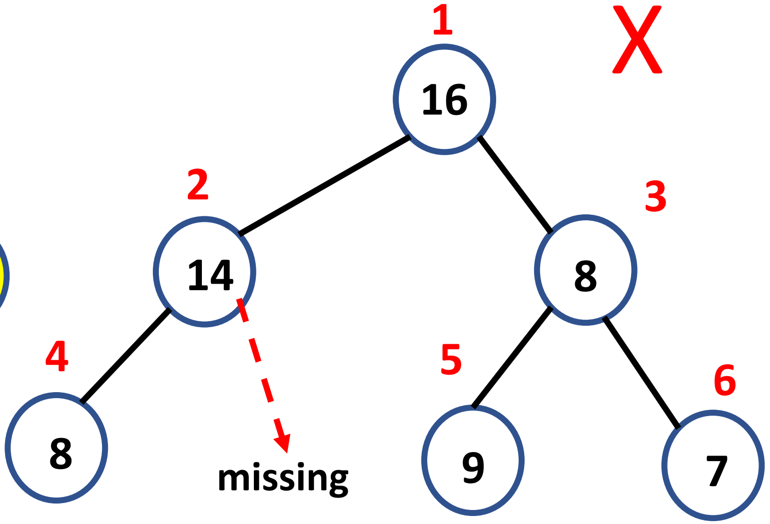
Examples



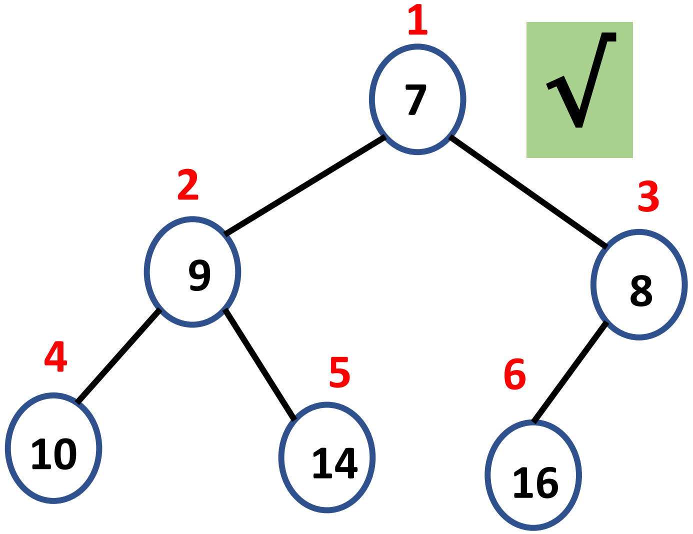
Yes, it is a max-heap



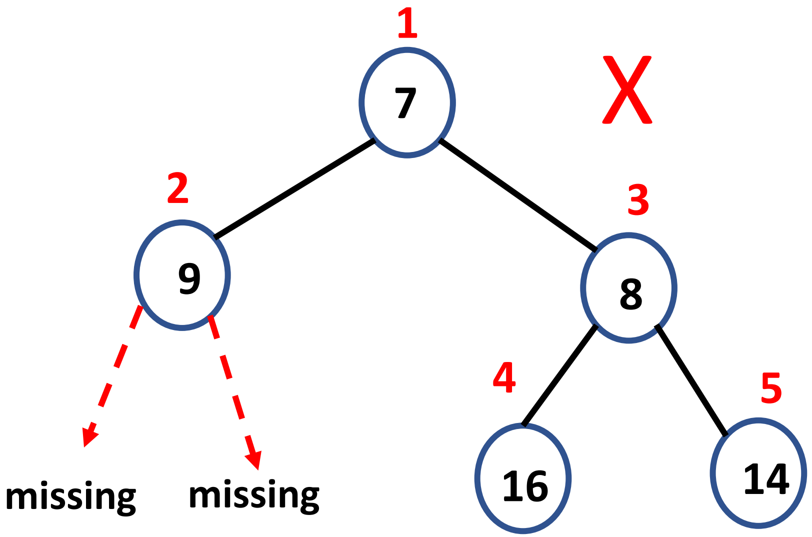
Not a max-heap, because Node 8 violates the heap property



Not a heap, because it is not complete binary tree.

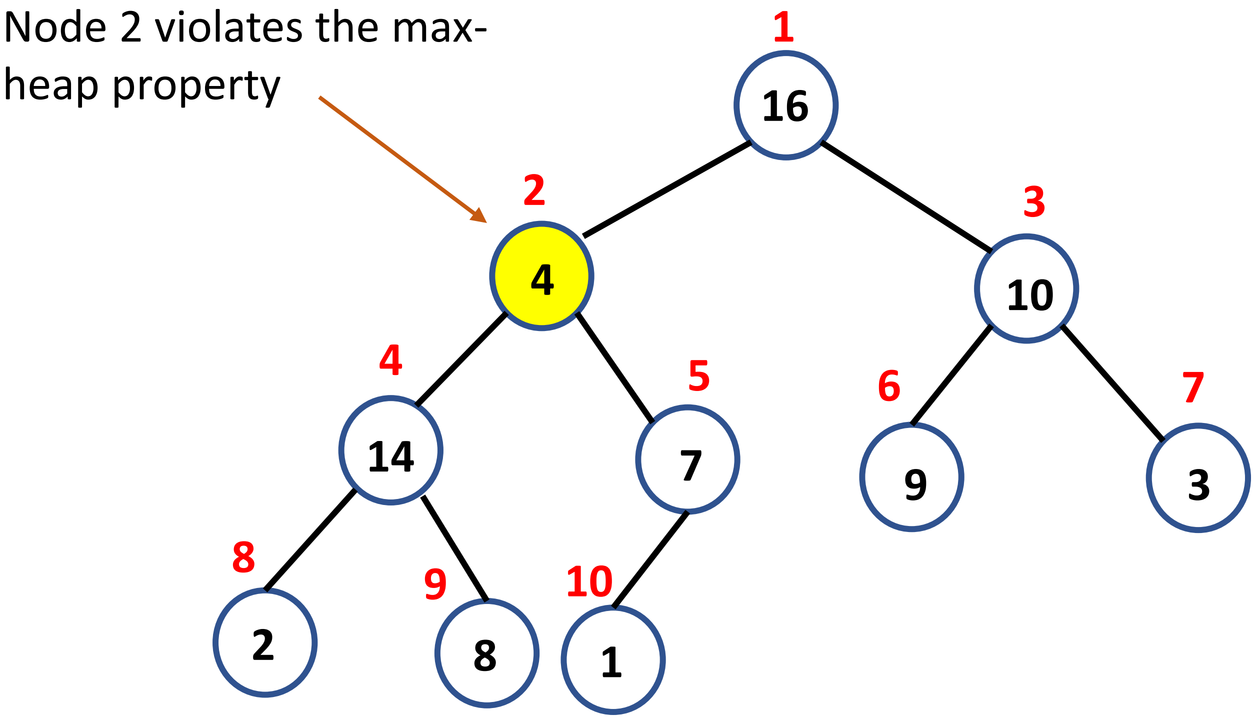


Yes, it is a min-heap

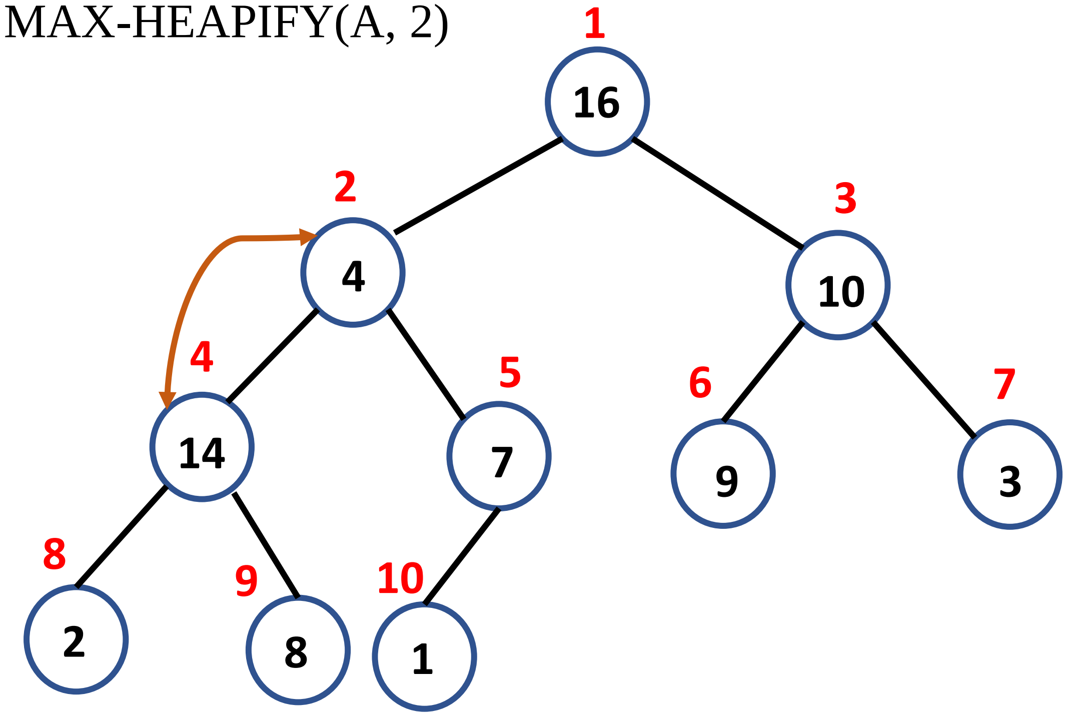


No, it is not a min-heap, because it not a complete binary tree

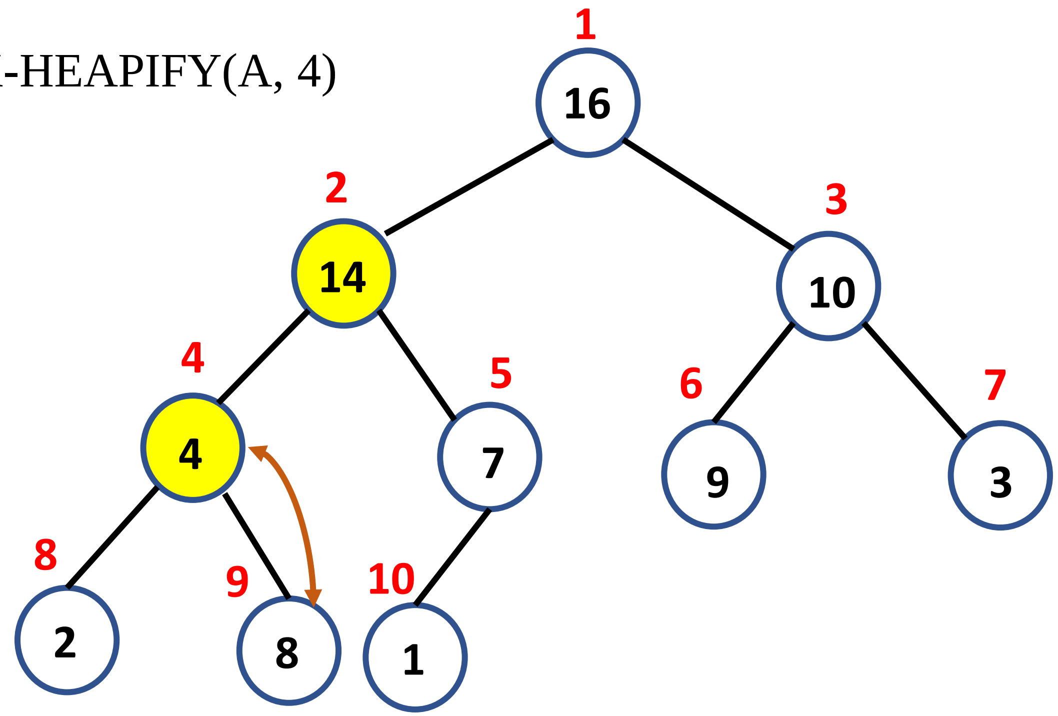
```
MAX-HEAPIFY(A,i)
    l = LEFT(i)
    r = RIGHT(i)
    if l ≤ A.heap-size and A[l] > A[i]
            largest = l
    else  largest = i
    if r ≤ A.heap-size and A[r] > A[largest]
            largest = r
    if largest ≠ i
            exchange A[i] with A[largest]
            MAX-HEAPIFY(A , largest)
```



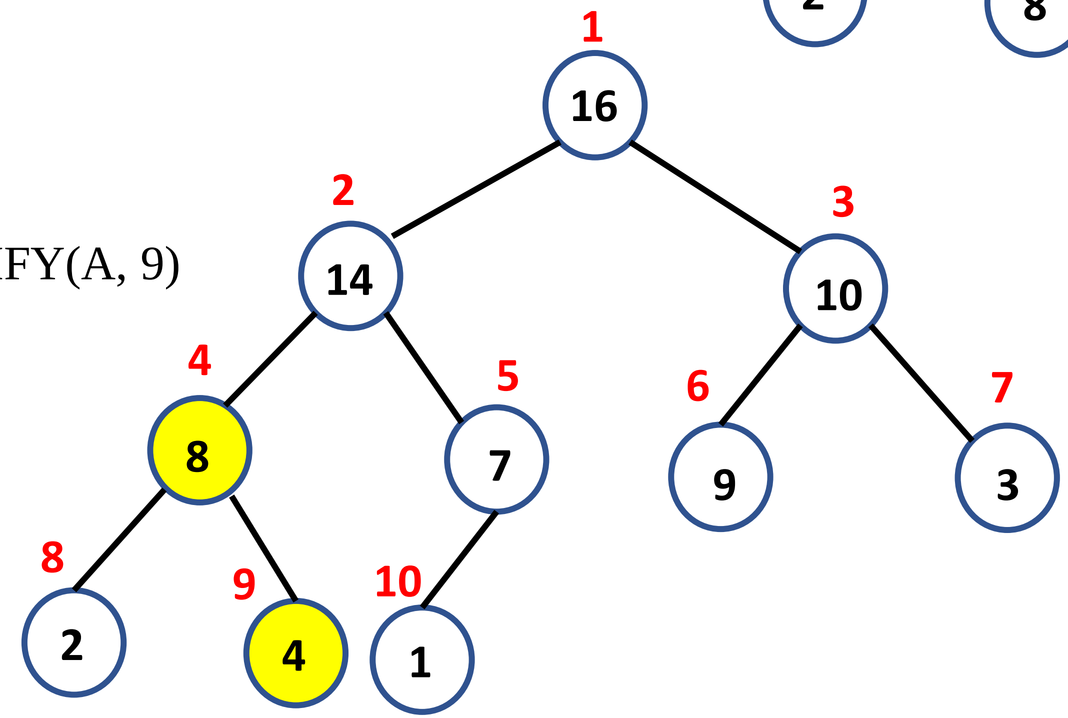
MAX-HEAPIFY(A, 2)



MAX-HEAPIFY(A, 4)



MAX-HEAPIFY(A, 9)

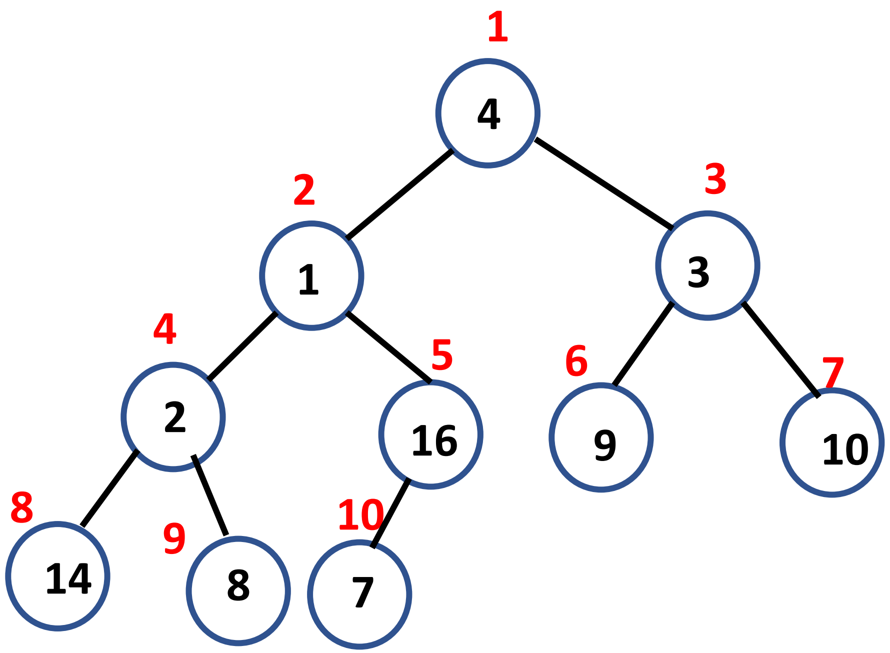


```
BUILD-MAX-HEAP(A)
A.heapsize = A.length
for i = floor( A.length/2 ) downto 1
      MAX-HEAPIFY(A,i)
```

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

i = 10/2 = 5

MAX-HEAPIFY(A, 5) Heap property is satisfied



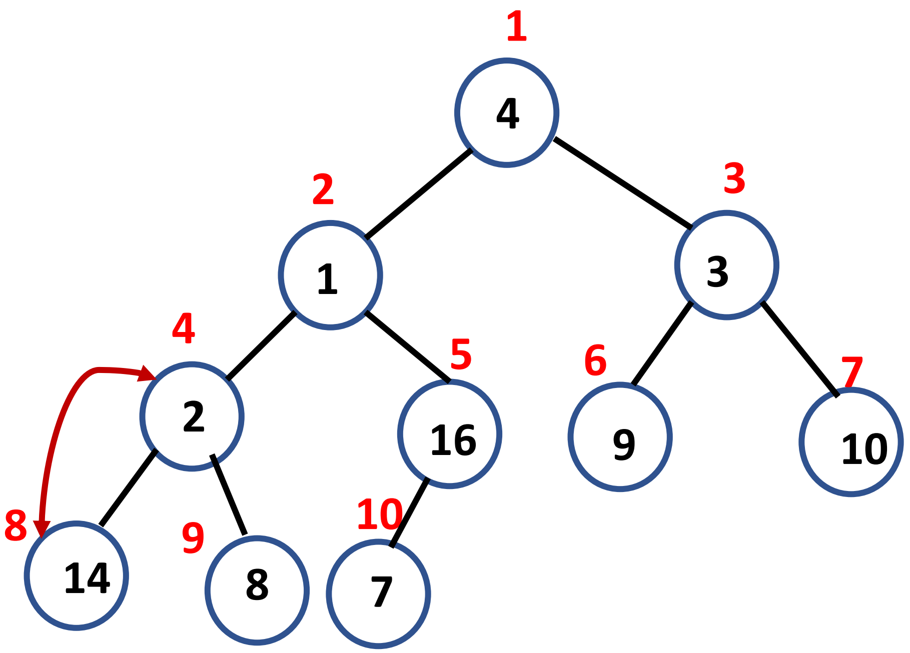
i = 10/2 = 5

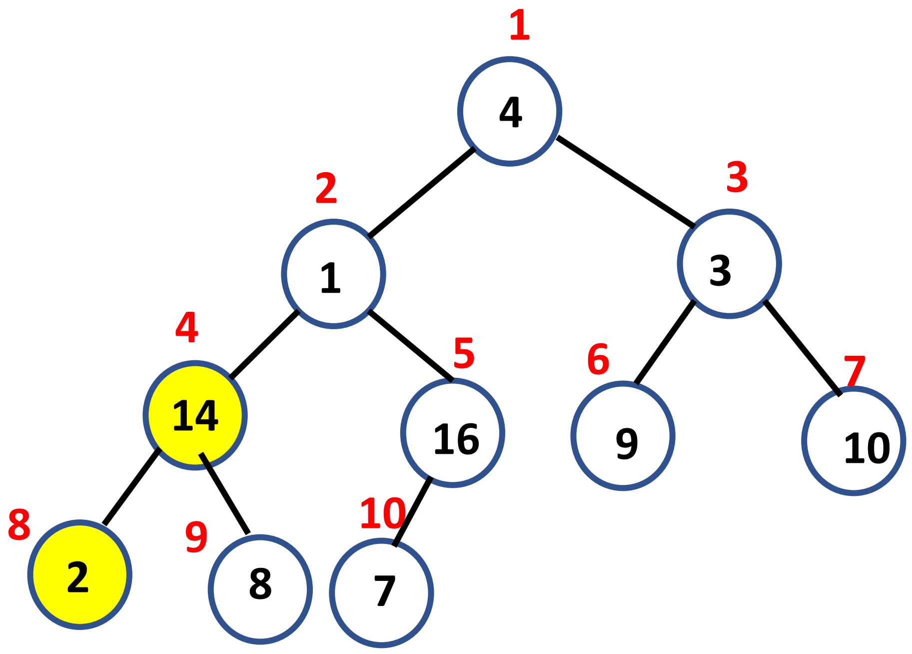
MAX-HEAPIFY(A, 5)

i = i – 1 = 4

MAX-HEAPIFY(A, 4)

MAX-HEAPIFY(A, 8)





i = 10/2 = 5

MAX-HEAPIFY(A, 5)

i = i – 1 = 4

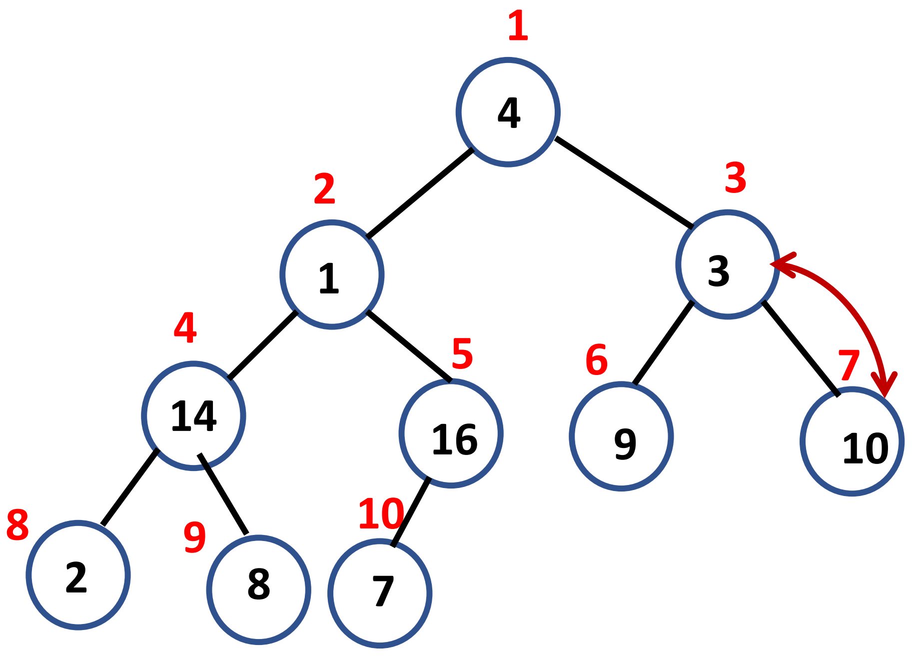
MAX-HEAPIFY(A, 4)

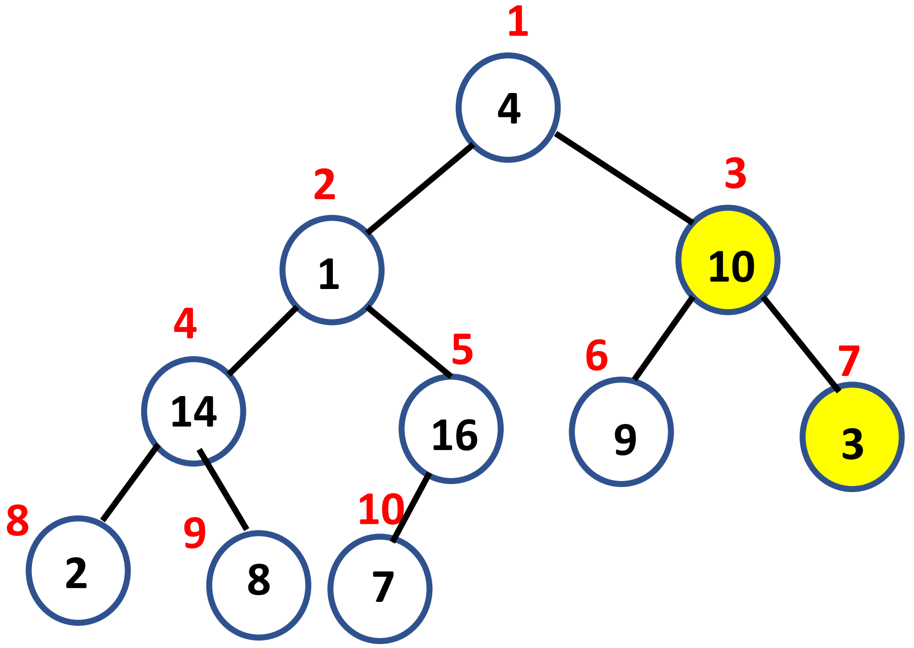
MAX-HEAPIFY(A, 8)

i = i – 1 = 3

MAX-HEAPIFY(A, 3)

MAX-HEAPIFY(A, 7)





i = 10/2 = 5

MAX-HEAPIFY(A, 5)

i = i – 1 = 4

MAX-HEAPIFY(A, 4)

MAX-HEAPIFY(A, 8)

i = i – 1 = 3

MAX-HEAPIFY(A, 3)

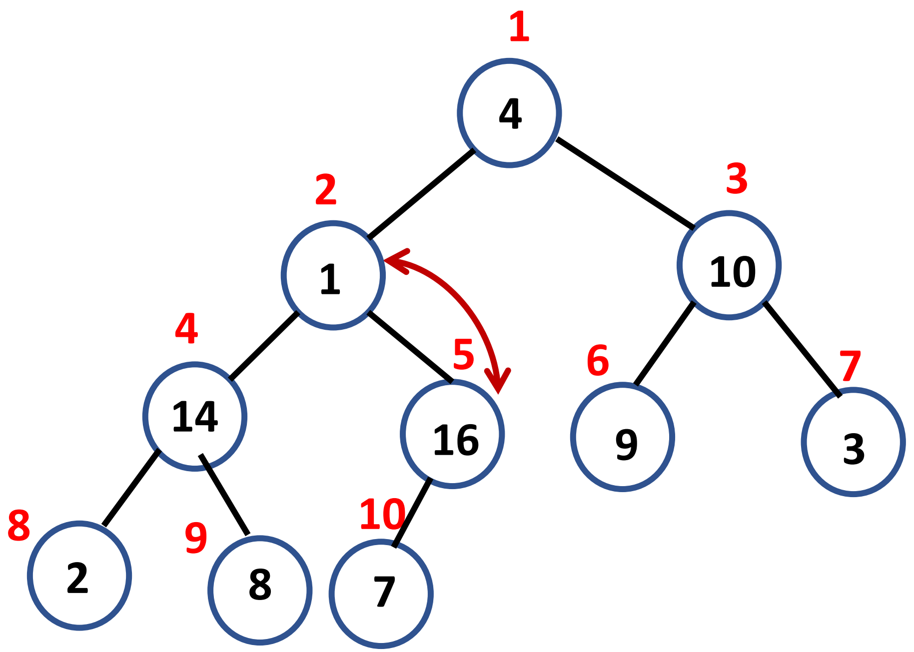
MAX-HEAPIFY(A, 7)

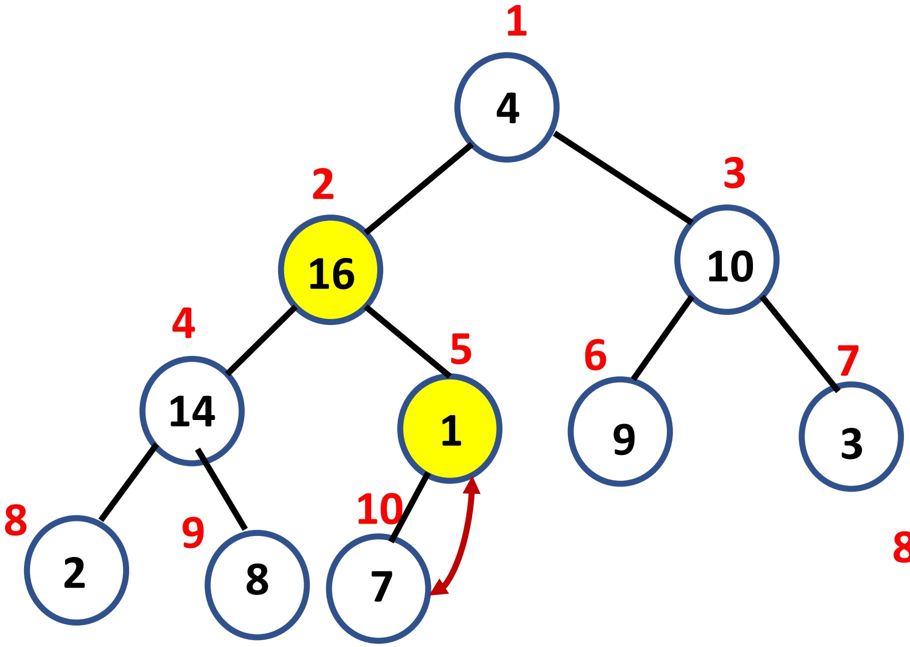
i = i – 1 = 2

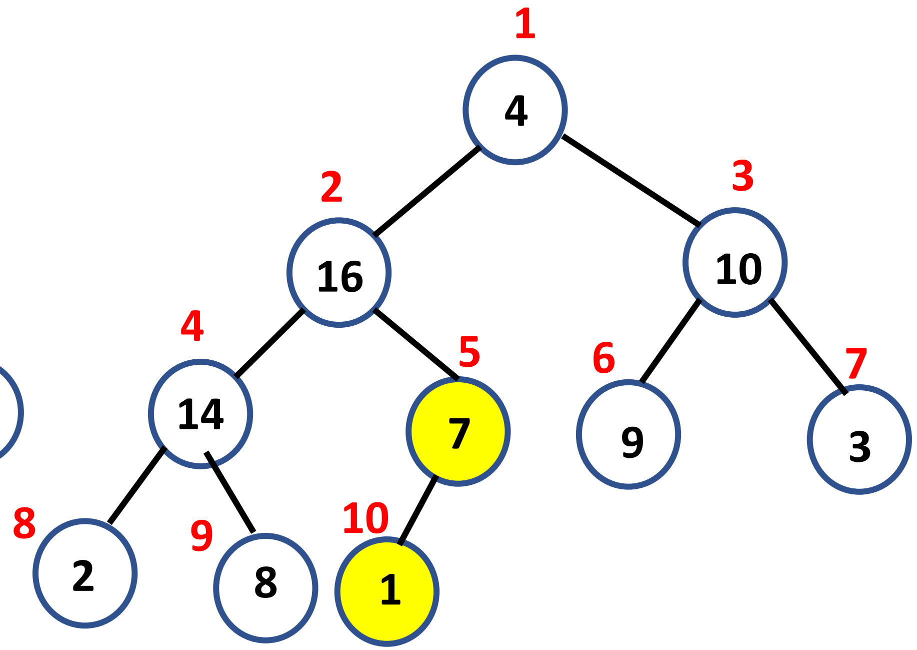
MAX-HEAPIFY(A, 2)

MAX-HEAPIFY(A, 5)

MAX-HEAPIFY(A, 10)







i = 10/2 = 5

MAX-HEAPIFY(A, 5)

i = i – 1 = 4

MAX-HEAPIFY(A, 4)

MAX-HEAPIFY(A, 8)

i = i – 1 = 3

MAX-HEAPIFY(A, 3)

MAX-HEAPIFY(A, 7)

i = i – 1 = 2

MAX-HEAPIFY(A, 2)

MAX-HEAPIFY(A, 5)

MAX-HEAPIFY(A, 10)

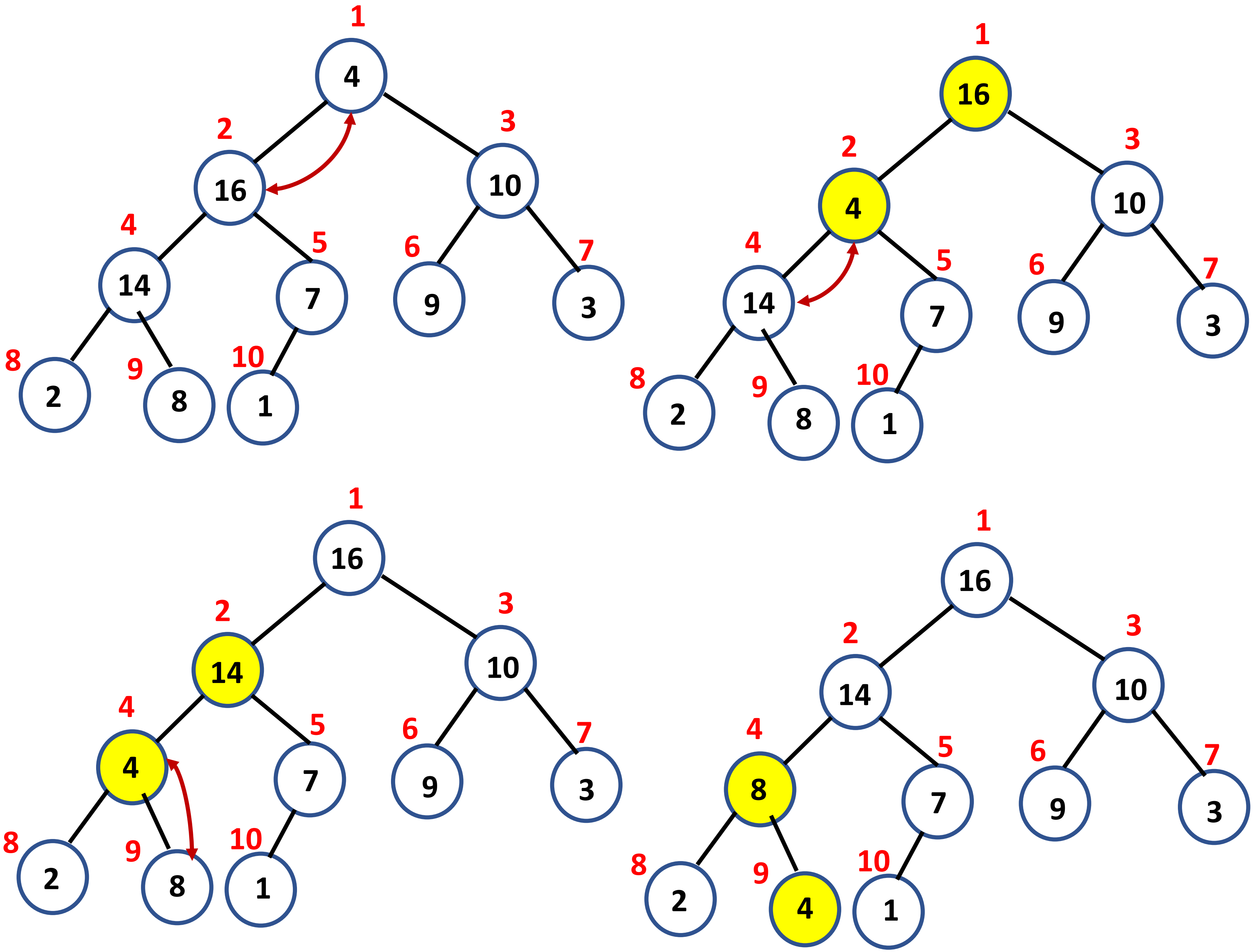
i = i – 1 = 1

MAX-HEAPIFY(A, 1)

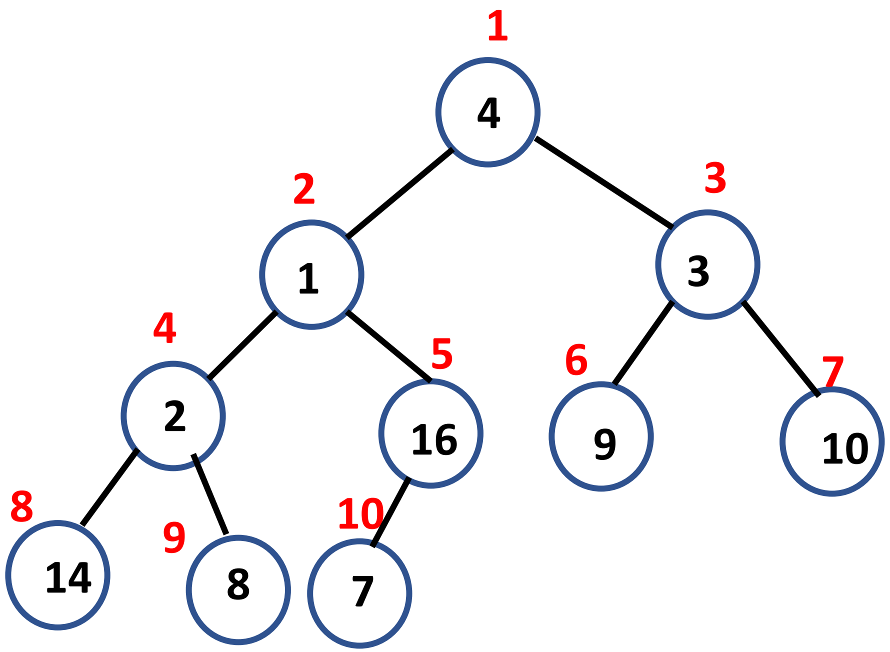
MAX-HEAPIFY(A, 2)

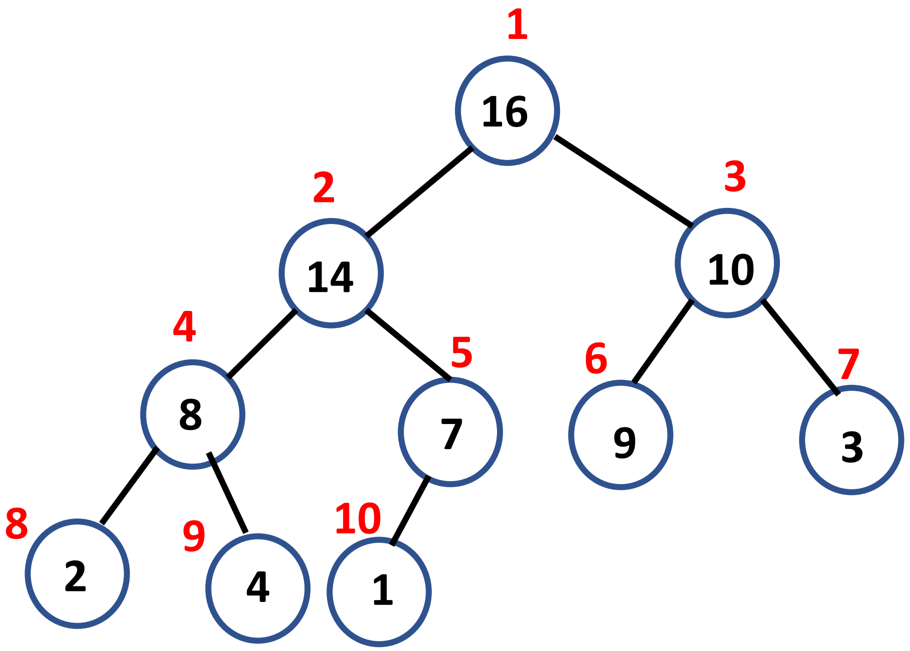
MAX-HEAPIFY(A, 4)

MAX-HEAPIFY(A, 9)



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





Max- heap

Example : Max-Heap

- Illustrate the operation of Build-Max-Heap on the array A ={ 5, 3, 17, 10, 84, 19, 6, 22, 9}

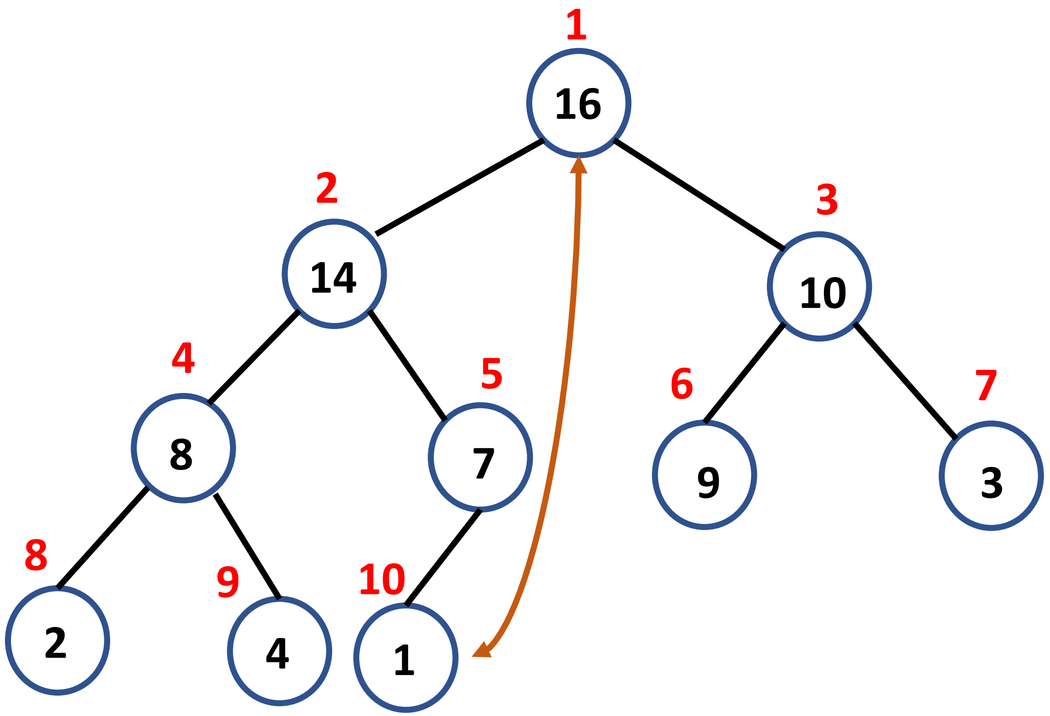
HEAP operations

Heap Operation	Description	Time complexity
MAX-HEAPIFY(A)	used to maintain the max-heap property.	O(log n)
MAX-HEAP-INSERT(A,k)	used to insert an element in to heap	O(log n)
HEAP-EXTRACT-MAX(A)	removes and returns the largest key.	O(log n)
HEAP-INCREASE-KEY(A, x, k)	Increases the value at index x to k. if k>A[x]	O(log n)
HEAP-MAXIMUM(A)	returns the largest key in heap	O(1)
BUILD-MAX-HEAP(A,n)	Used to construct max heap	O(n)

The above procedures run in ***O(log n)*** time, so heap data structure is used to implement priority queue.

HEAP-EXTRACT-MAX(A)

```
if A.heapsize < 1
    error "heap underflow"
max = A[1]
A[1] = A[A.heapsize]
A.heapsize = A.heapsize – 1
MAX-HEAPIFY(A , 1)
return max
```

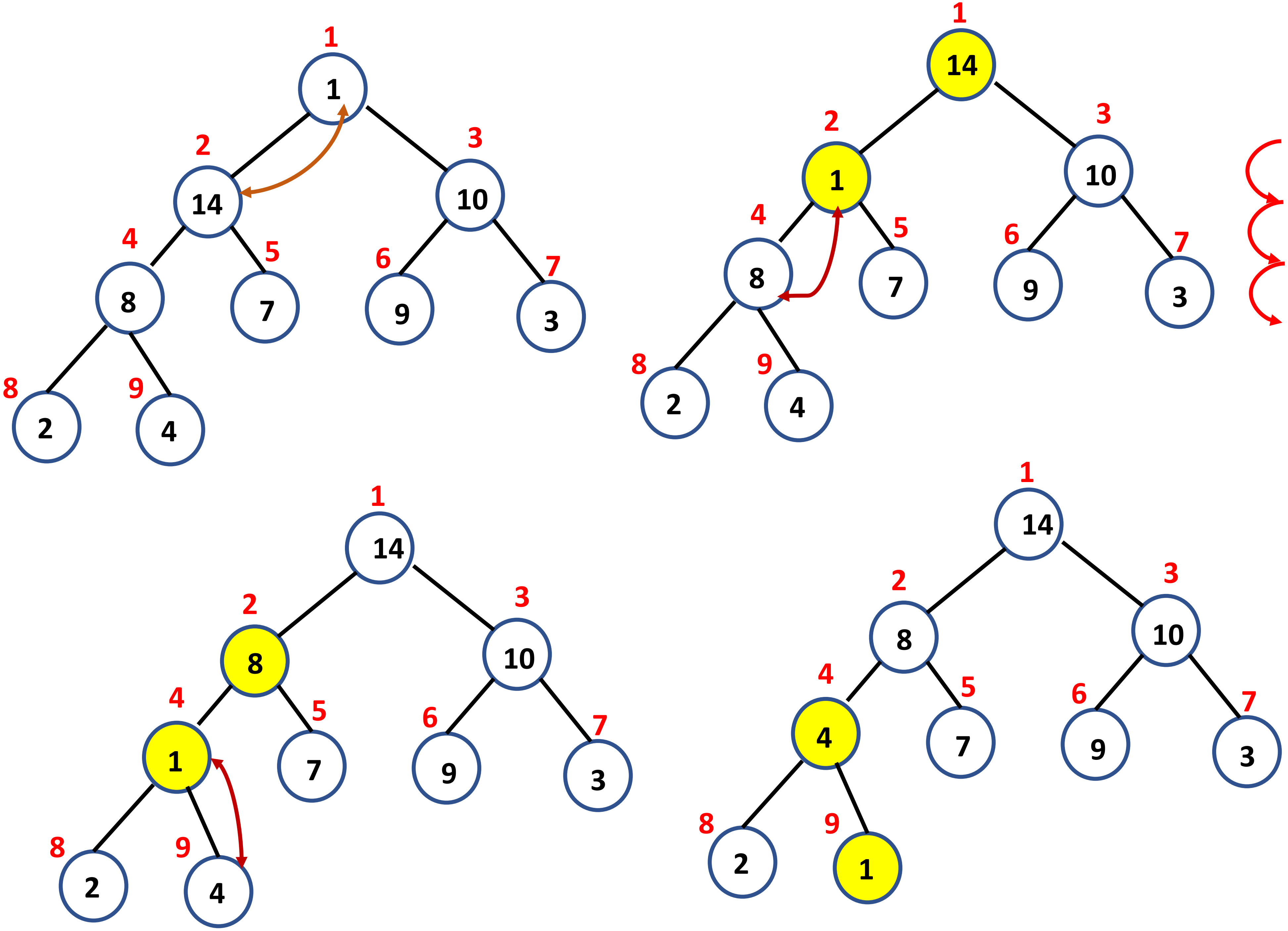


max = 16

Assign A[1] with A[10]

reduce A.heapsize by 1

MAX-HEAPIFY(A, 1)



max = 16

Exchange A[1] with A[10]

reduce A.heapsize by 1

MAX-HEAPIFY(A, 1)

MAX-HEAPIFY(A, 2)

MAX-HEAPIFY(A, 4)

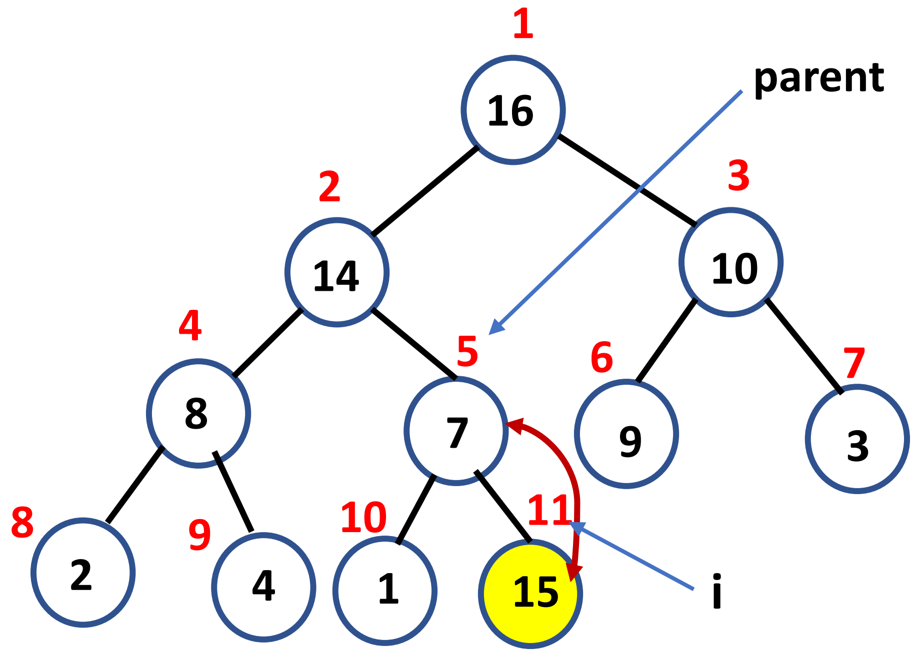
MAX-HEAPIFY(A, 9)

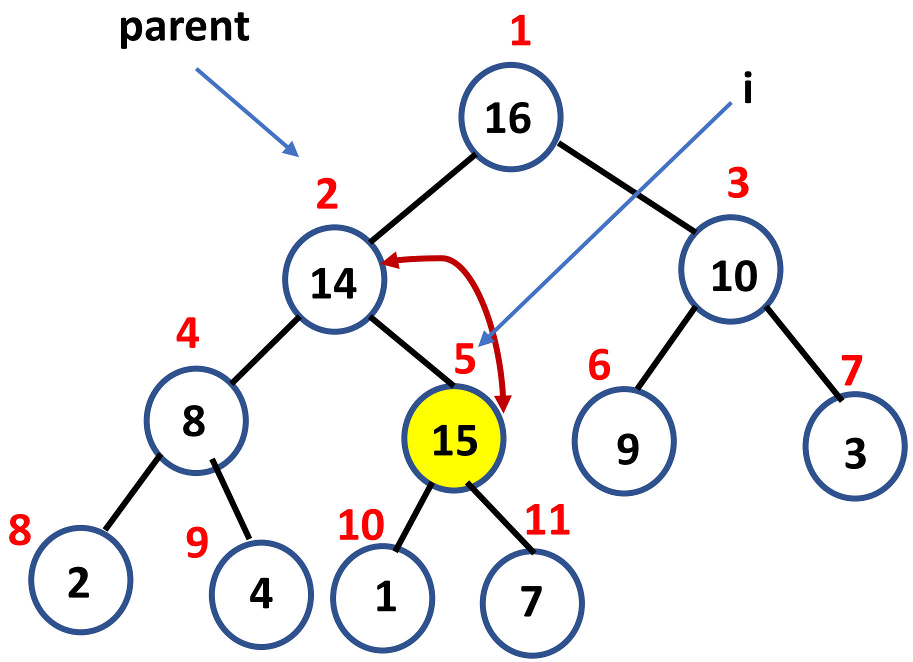
return max

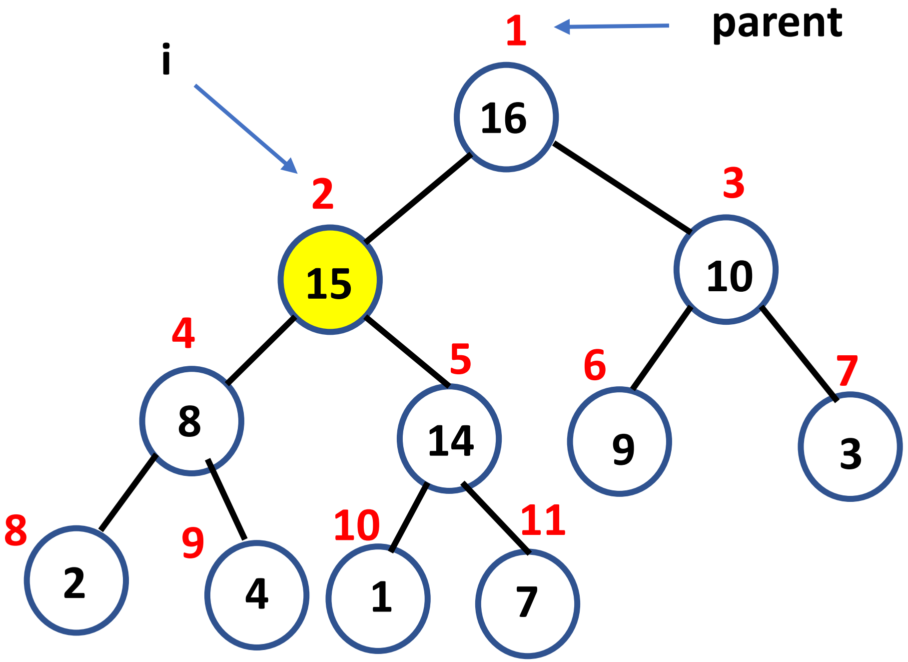
Max-Heap Insert

Max-Heap-Insert(A, key)

1. A.heapSize = A.heapsize + 1
2. i = A.heapSize
3. A[i] = key
4. while i >1 and A[Parent(i)] < A[i]
5. exchange A[i] and A[Parent(i)]
6. i ←Parent(i)





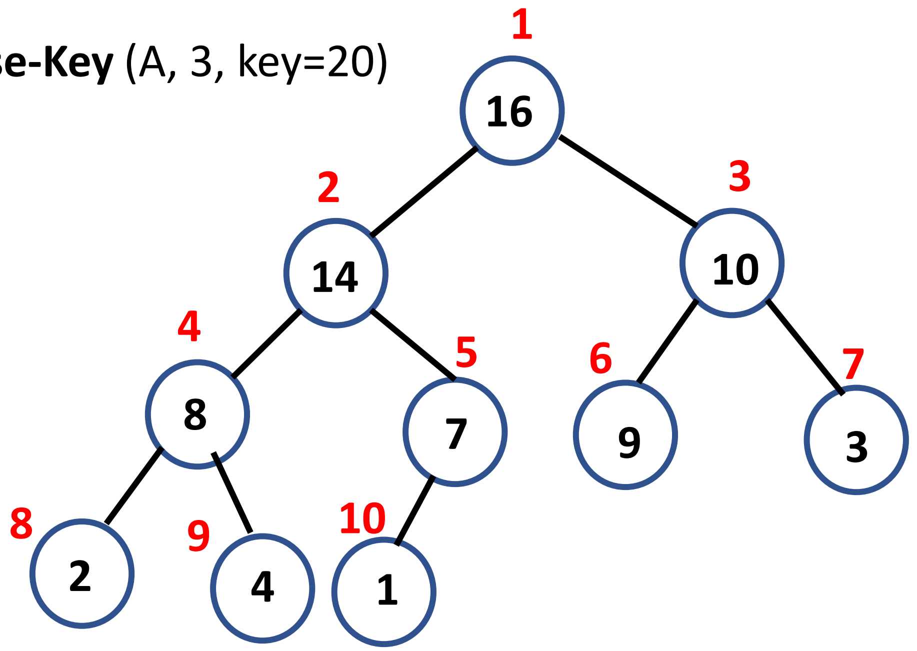


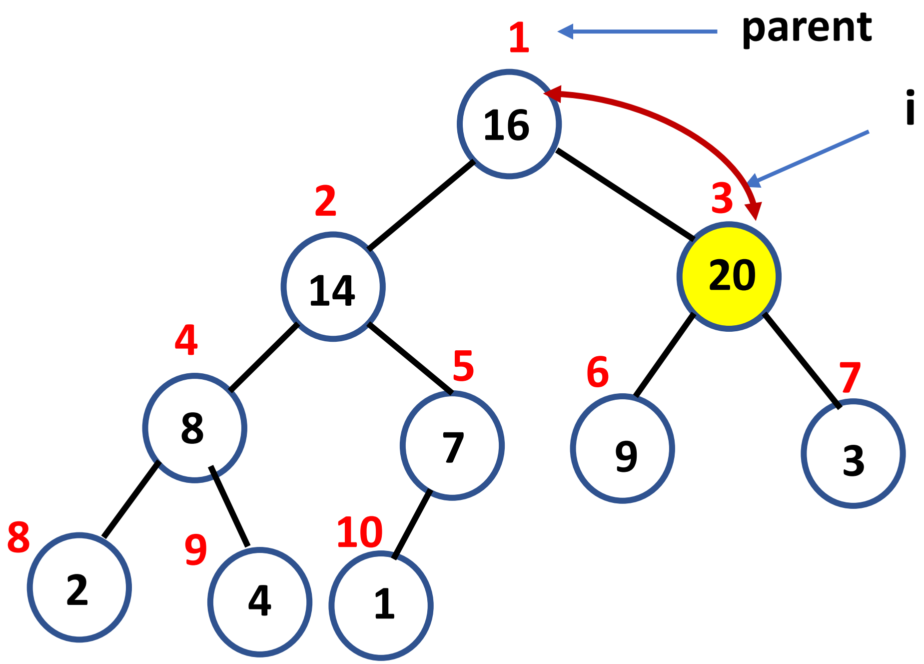
Heap-Increase-Key

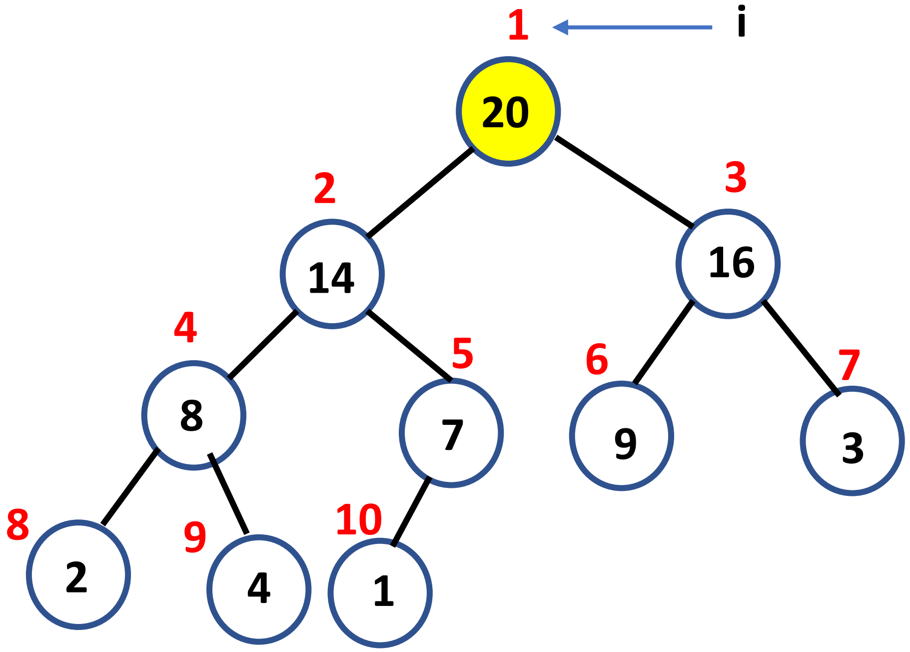
Heap-Increase-Key(A,i,key)

```
if key <A[i]
        error("New key must be larger than current key")
A[i] ←key
while i >1 and A[Parent(i)] < A[i]
        exchange A[i] and A[Parent(i)]
        i ←Parent(i)
```

Heap-Increase-Key (A, 3, key=20)







Priority Queue: Definition

- A **Priority Queue** is a data structure for maintaining a set S of elements, each with an associated value called a key. They can be in two forms: max-priority queues and min-priority queues.
- A max-priority queue supports the following operations:
  - **Insert (S,x)** inserts the element x into the set S, which is equivalent to the operation $S = S \cup \{x\}$.
  - **Maximum(S)** returns the maximum element of S.
  - **Extract-Max(S)** removes and returns the maximum element of S.
  - **Increase-Key(S, x, k)** increases the value of element x's key to the new value k, which is assumed to be at least as large as x's current key value.
- A min-priority queue supports Insert(S,x), Minimum(S), Extract-Min(S), Decrease-Key(S,x,k).

Applications of Priority Queues

Max-Priority Queue Applications:

Used to schedule jobs on a shared computer. The max-priority queue keeps track of the jobs to be performed and their relative priorities. When a job is finished or interrupted the scheduler selects the highest priority job from among the pending jobs by calling **Extract-Max**. The scheduler can add a new job to the queue at any time by calling **Insert**.

Min-Priority Queue Application:

Used in a event-driven simulator. The items in the queue are events to be simulated with an associated time of occurrence that can be used as key. The simulation program calls **Extract-Min** at each step to choose the next event to simulate. The simulator program calls **Insert** method when a new events are produced.

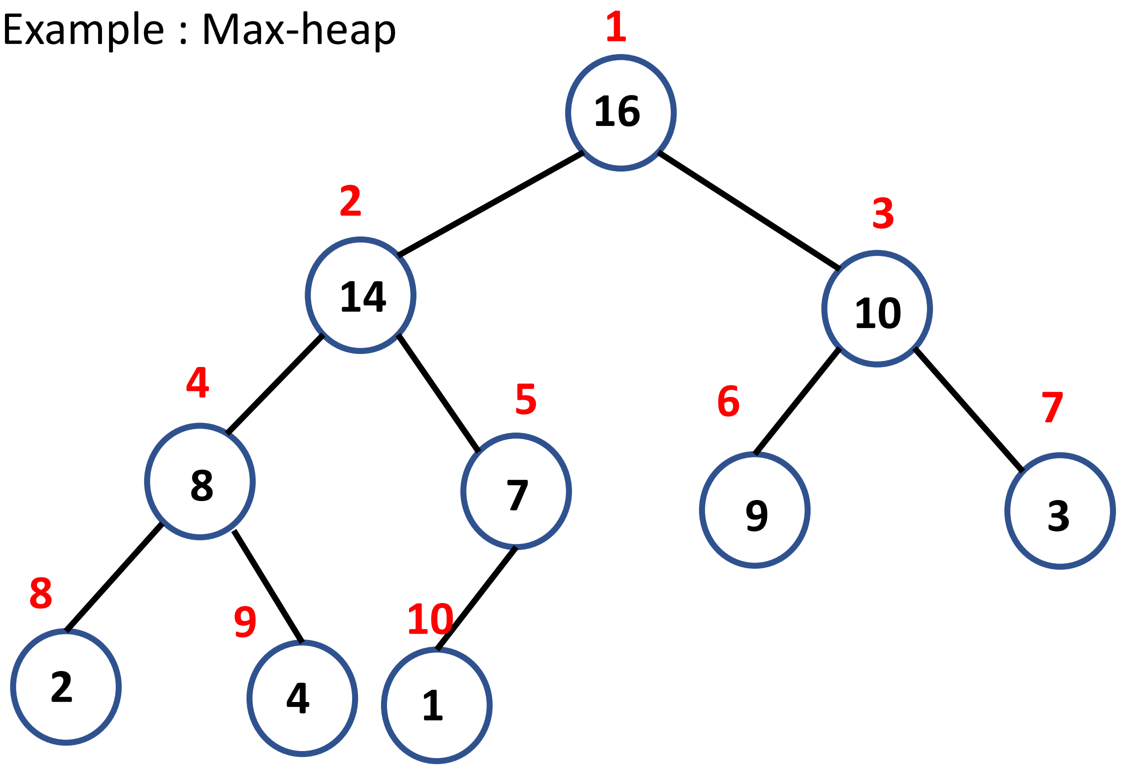
STL – Priority Queue

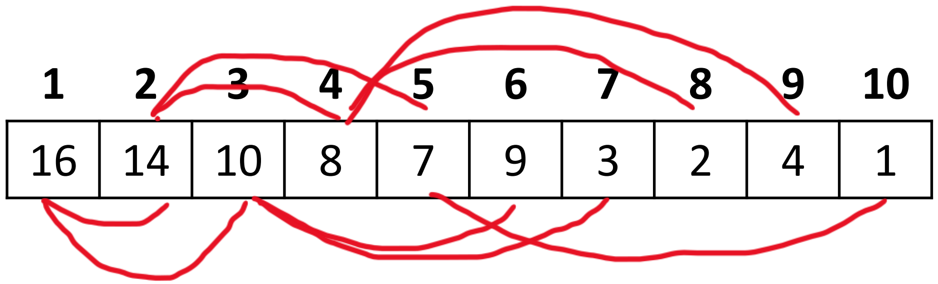
	Without STL	Priority Queue
Package	User-defined class	Available #include<queue>
Creation	PriorityQueue pq;	PriorityQueue<int> pq;
Insert	pq.insert(x)	pq.push(x)
Delete(Extract Maximum)	pq.extractMaximum()	pq.pop()
Increase key	pq.increasekey(x,k)	No operation
Maximum element	pq.maximum()	pq.top()
size	pq.size()	pq.size()
Checking empty	pq. isEmpty()	pq.empty()

Heap

- A heap(binary) is a nearly complete binary tree.
- An Array A that represents a heap is an object with two attributes
  - A. length : gives the number of element in the array.
  - A.heap-size : the number elements that can be stored within array A.
  - $0 \le$ A.heapsize $\le$ A.length.

Example : Max-heap





Parent (i)
    return floor(i/2)

left (i)
    return 2*i

right (i)
    return 2 * i + 1

There are two kinds of heaps

- Max-heaps
- Min-heaps

- Depends on the type heap-property , a heap is categorized as max-heap or min-heap.

- A heap is said to be **max-heap**, if every node i other than root A[parent(i)] ≥ A[i] i.e., the value of a node is at most the value of its parent. i.e., the largest element in a max-heap is stored at the root , and the subtree rooted at a node contains values no larger than that contained at the node itself.

- A heap is said to be **min-heap**, if every node i other than root A[parent(i)] ≤ A[i] i.e., the smallest element is at the root.

HEAP operations

Max-Heap Operations

- ❖MAX-HEAPIFY - used to maintain the max-heap property.
- ❖MAX-HEAP-INSERT - used to insert an element in to heap.
- ❖HEAP-EXTRACT-MAX - removes and returns the largest key.
- ❖HEAP-INCREASE-KEY - Increases the value to k.
- ❖HEAP-MAXIMUM - returns the largest key in heap (O(1))

- The above procedures run in ***O(log n)*** time, so heap data structure is used to implement priority queue.
- ***BUILD-MAX-HEAP*** runs in linear time O(n), produces a max-heap from an unordered input array of size n.

Heap-Increase-Key

Heap-Increase-Key(A,i,key)

```
// Input: A: an array representing a heap, i: an array index, key: a new key greater than A[i]
// Output: A still representing a heap where the key of A[i] was increased to key
// Running Time: O(logn) where n =heap-size[A]
if key <A[i]
        error(“New key must be larger than current key”)
A[i] ←key
while i >1 and A[Parent(i)] < A[i]
      exchange A[i] and A[Parent(i)]
      i ←Parent(i)
```

Max-Heap-Insert(A, key)

```
// Input: A: an array representing a heap, key: a key to insert
// Output: A modified to include key
// Running Time: O(logn) where n =heap-size[A]
A.heapsize ← A.heapsize + 1
A[A.heapsize] ← −∞
Heap-Increase-Key(A,A[A.heapsize],key)
```