Data Structure and Algorithms

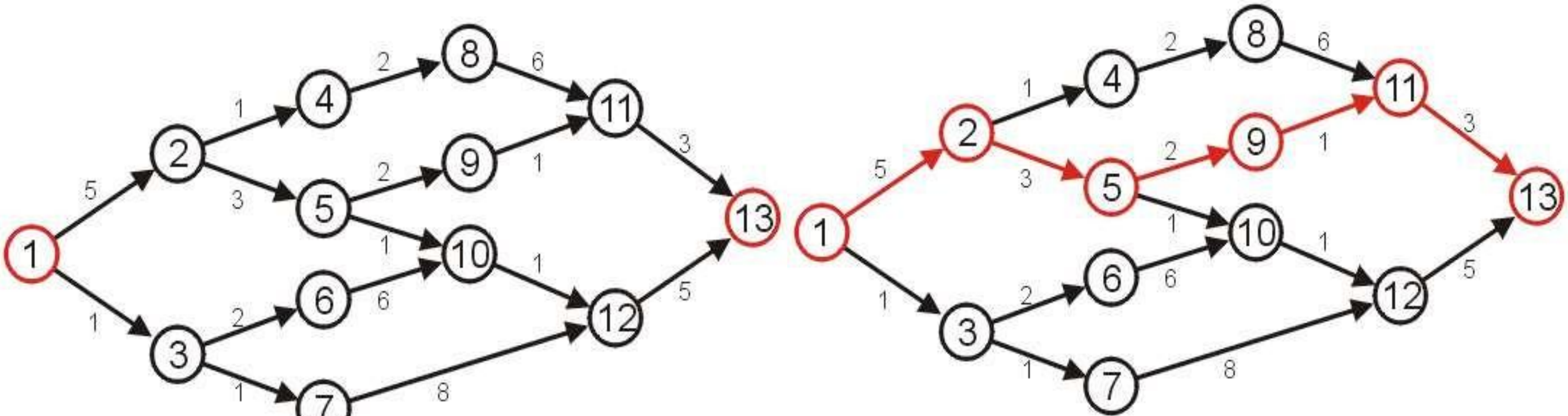
Affefah Qureshi

Department of Computer Science

Iqra University, Islamabad Campus.

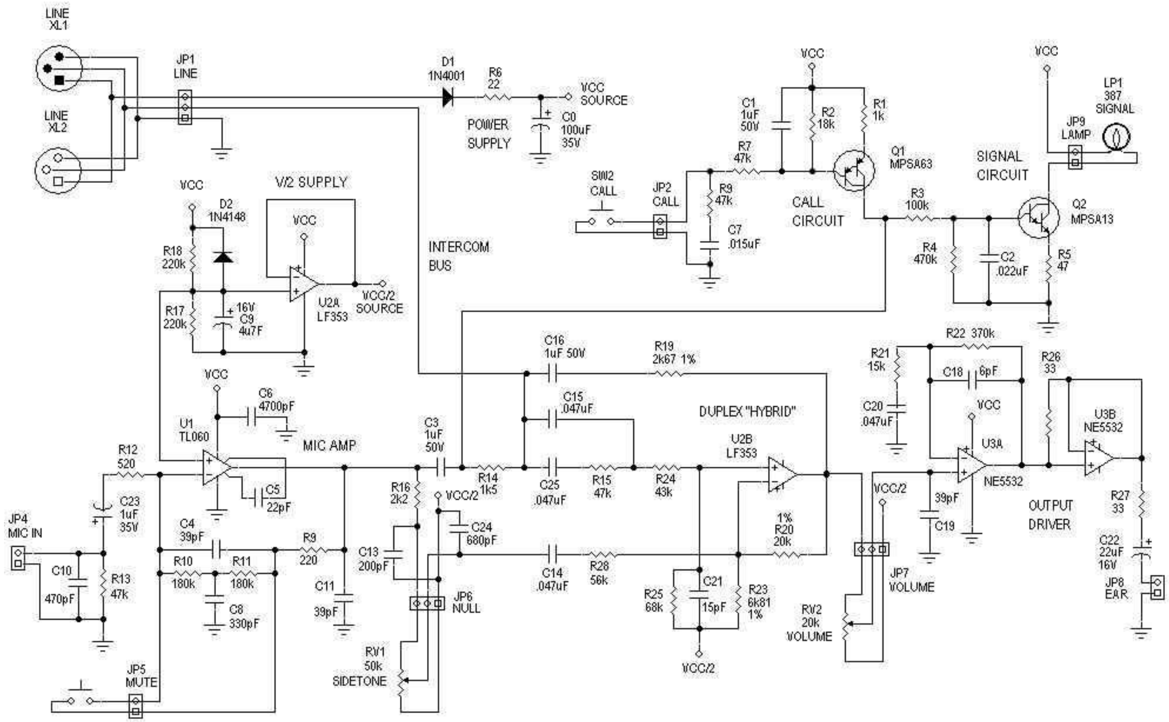
Shortest Path

- Given a weighted graph
  - Problem is to find the shortest path between two given vertices
- Length of a path in a weighted graph
  - Sum of the weights of each of the edges in that path
- Example: Shortest path from vertex 1 to vertex 13
  - Other paths exists but they are longer



Application – Circuit Design

- The time it takes for a change in input to affect an output depends on the shortest path



Application – Computer Networks

- The Internet is a collection of interconnected computer networks
  - Information is passed through packets

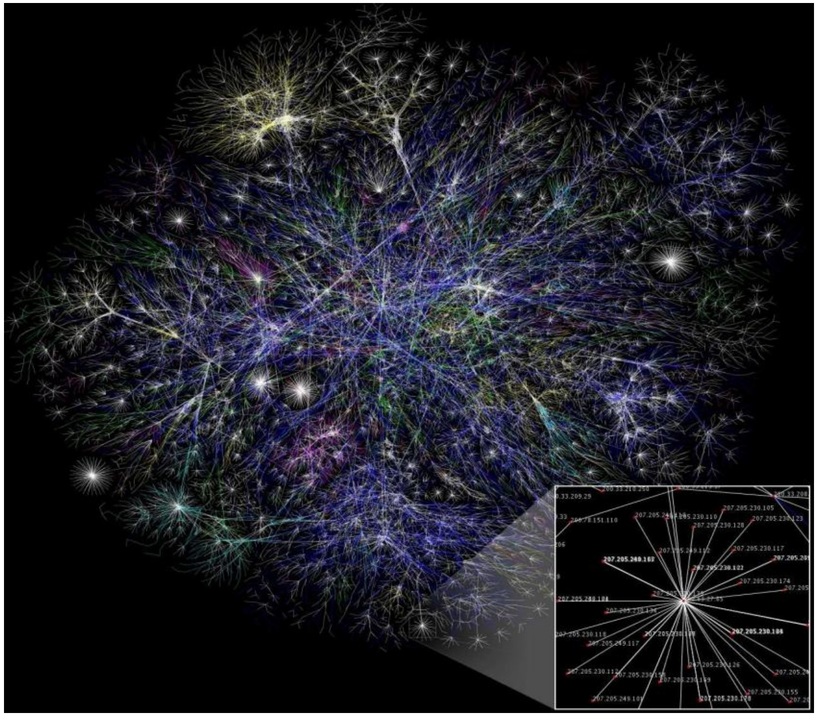
- Packets are passed from the source, through routers, to their destination

- Routers are connected to either:
  - Individual computers, or
  - Other routers

- These may be represented as graphs

Application – Computer Networks

- A visualization of the graph of the routers and their various connections through a portion of the Internet



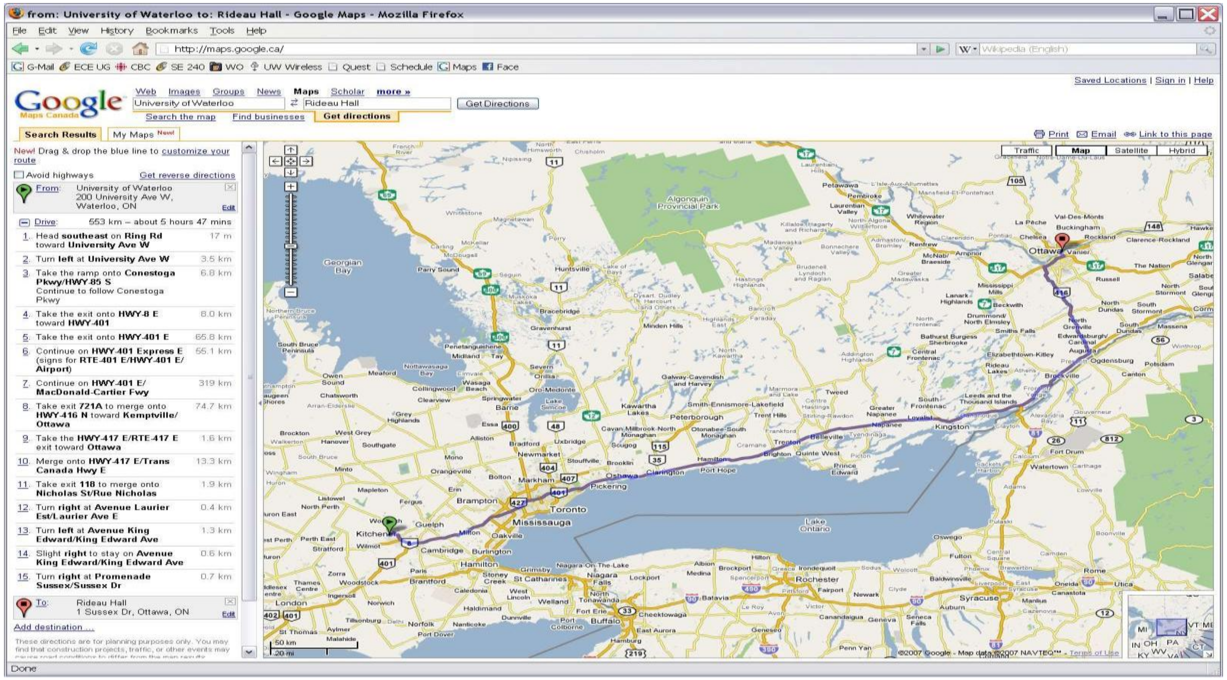
Application – Computer Networks

- The path a packet takes depends on the IP address

- Metrics for measuring the shortest path may include
  - Low latency (minimize time)
  - Minimum hop count (all edges have weight 1)

Application – Traffic

- Find the shortest route between to points on a map
  - Shortest path, however, need not refer to distance…



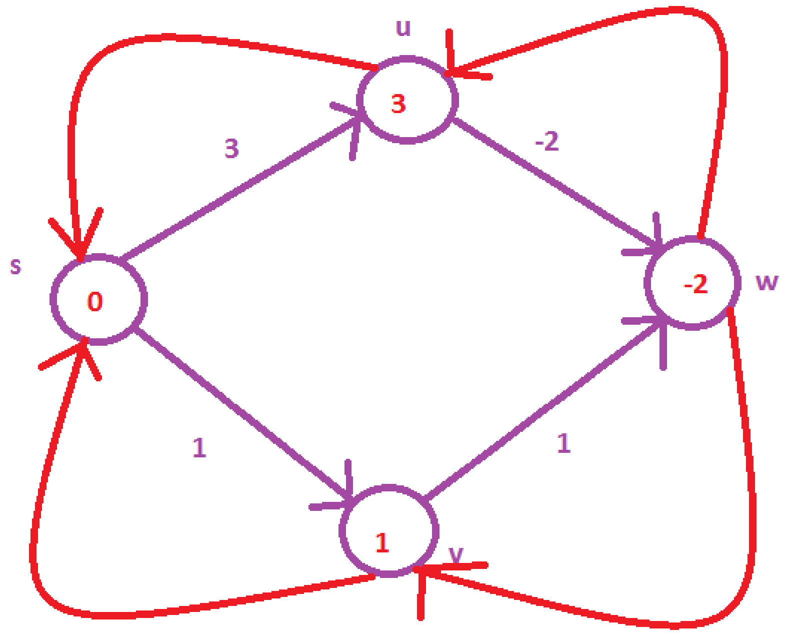
Variants of Shortest Path

Given a graph G = (V, E)

- Single-source shortest paths
  - Find shortest path from a given source vertex s to each vertex v ϵ V
- Single-destination shortest paths
  - Find shortest path to a given destination vertex t from each vertex v
- Single-pair shortest path
  - Find shortest path from u to v for given vertices u and v
- All-pairs shortest-paths
  - Find shortest path from u to v for every pair of vertices u and v

Single Source Shortest Path

Dijkstra's Algorithm



Dijkstra's Algorithm

- Problem: From a given source vertex $s \in V$, find the shortest-paths and their weights $w(s,v)$ for all $v \in V$

- Idea of the Algorithm
  - Maintain a set S of vertices whose shortest-path distances from s are known
  - At each step add to S the vertex $v \in V-S$ whose distance estimate from s is minimal
  - Update the distance estimates of vertices adjacent to v

Dijkstra's Algorithm - Pseudocode

i. Mark the source node with a current distance of 0 and the rest with infinity.

ii. Set the non-visited node with the smallest current distance as the current node.

iii. For each neighbor, N of the current node adds the current distance of the adjacent node with the weight of the edge connecting 0->1. If it is smaller than the current distance of Node, set it as the new current distance of N.

iv. Mark the current node 1 as visited.

v. Go to step 2 if there are any nodes are unvisited.

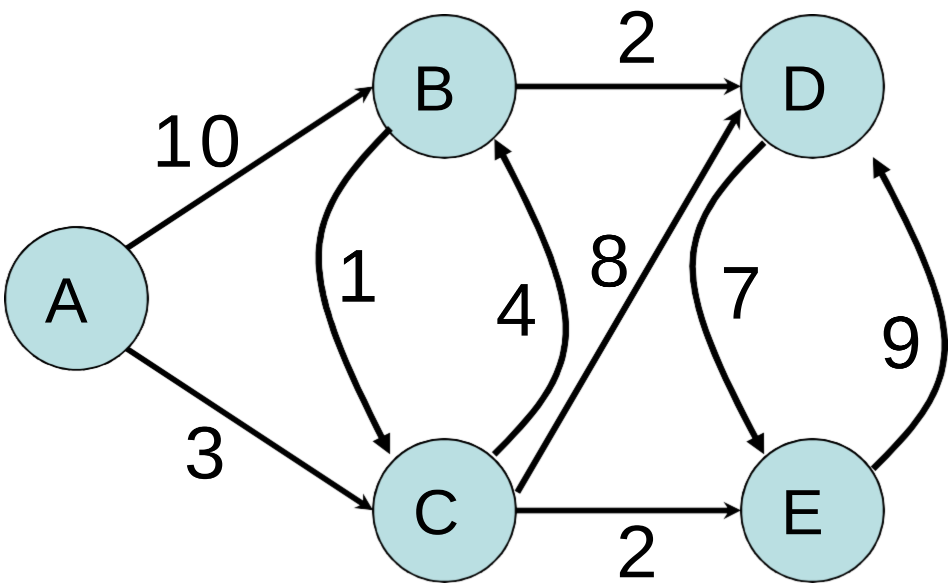
Comments on Dijkstra's Algorithm

- If at some point, all unvisited vertices have a distance ∞?
  - This means that the graph is unconnected
  - We have found the shortest paths to all vertices in the connected subgraph containing the source vertex

- To find the shortest path between vertices v_j and v_k?
  - Apply the same algorithm, but stop when visiting vertex v_k

- Does the algorithm change if graph is directed?
  - No

Dijkstra's Algorithm – Example



A	B	C	D	E
A,C	10	3		
A,C,E	7	3	11	5
A,C,E,B	7	3	11	5
A,C,E,B,D	7	3	9	5

S: {A, C, E, B, D}

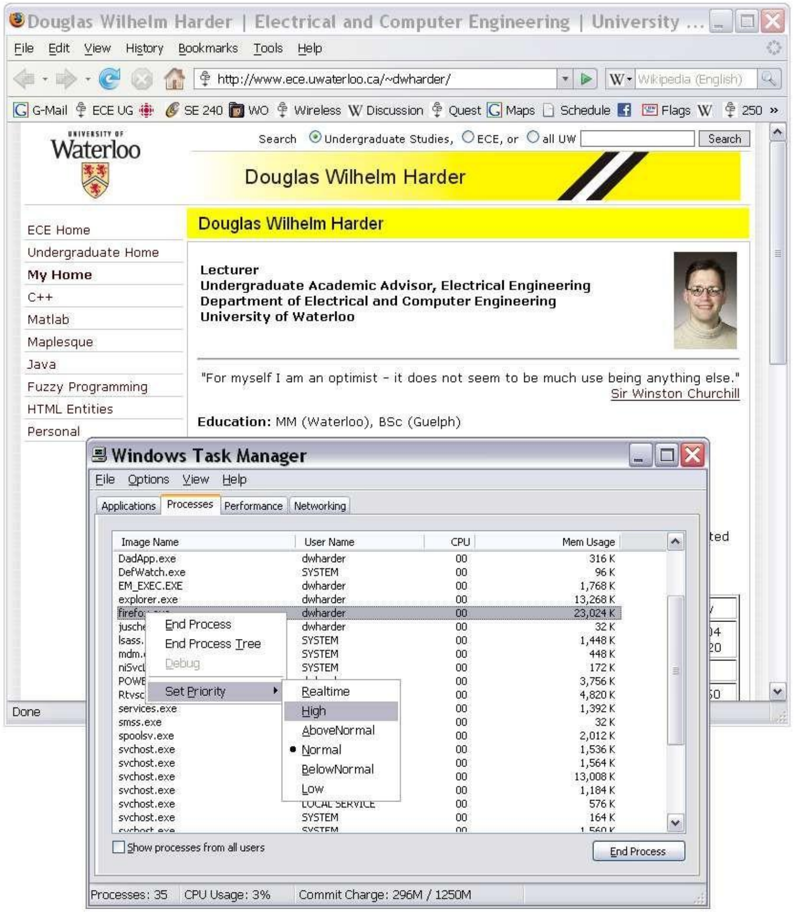
Priority Queue/ Heap

Motivation

- With queues the order may be summarized by first in, first out
- Some tasks may be more important or timely than others
  - Higher priority
- Priority queues
  - Enqueue objects using a partial ordering based on priority
  - Dequeue that object which has highest priority

Applications Of Priority Queue

- Hold jobs for a printer in order of length
- Store packets on network routers in order of urgency
- Ordering CPU jobs
- Emergency room admission processing



The priority of processes in Windows may be set in the Windows Task Manager

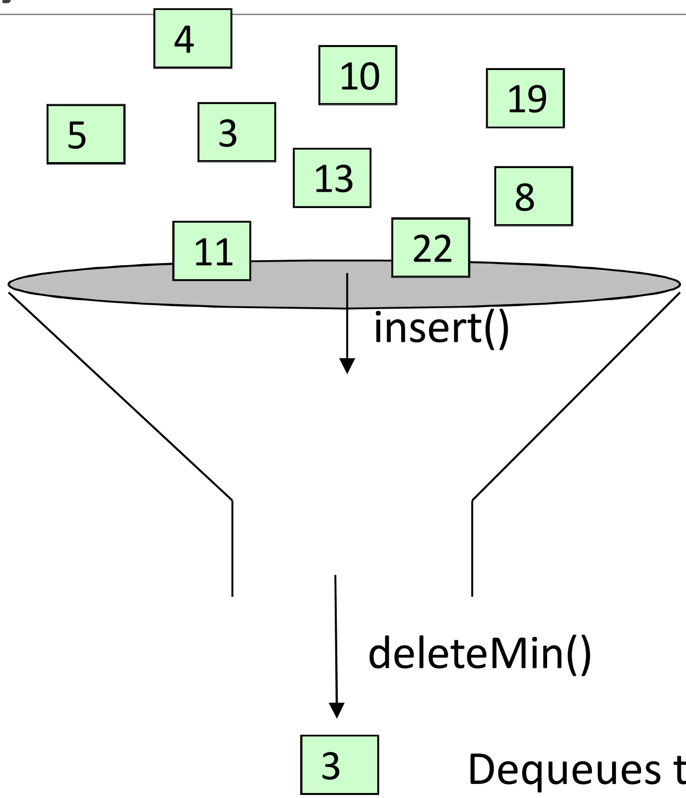
Priority Queue – ADT

- insert(i.e., enqueue)
  - Dynamic insert
  - Specification of a priority level (0-high, 1,2.. Low)

- deleteMin(i.e., dequeue)
  - Returns the current “highest priority” element in the queue
    - Element with the minimum priority level
  - Deletes that element from the queue

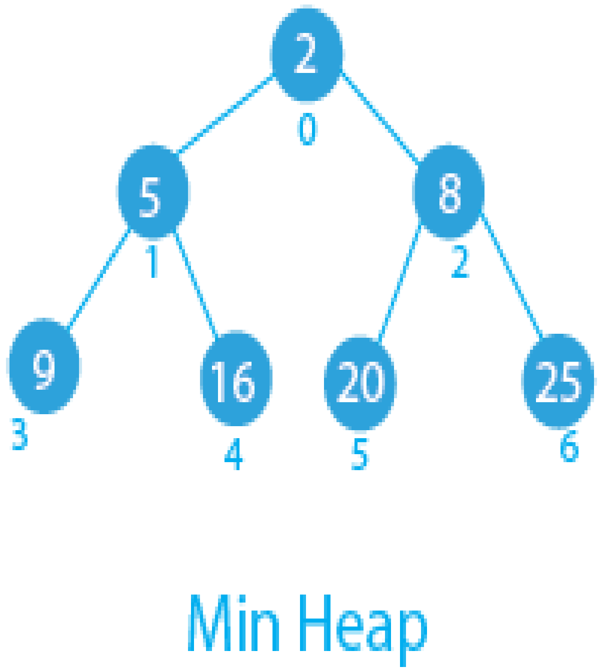
- Performance goal is to make the run time of each operation as close to O(1) as possible

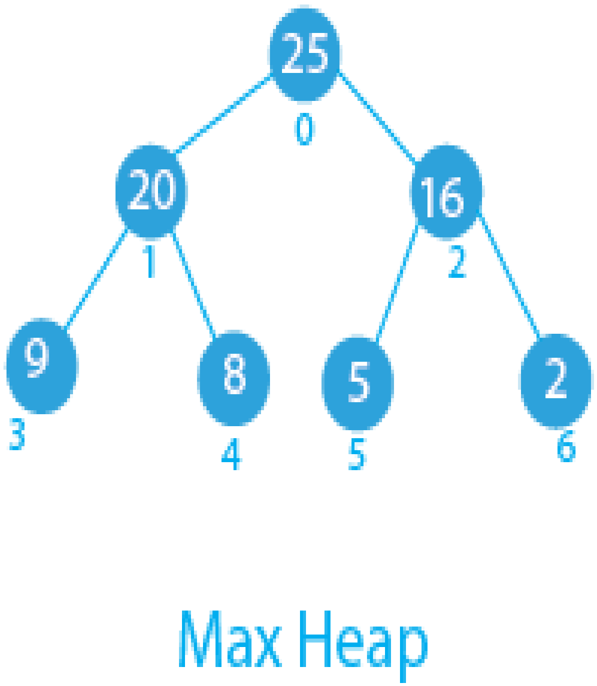
Priority Queue – ADT



Dequeues the next element with the highest priority

Binary Heap





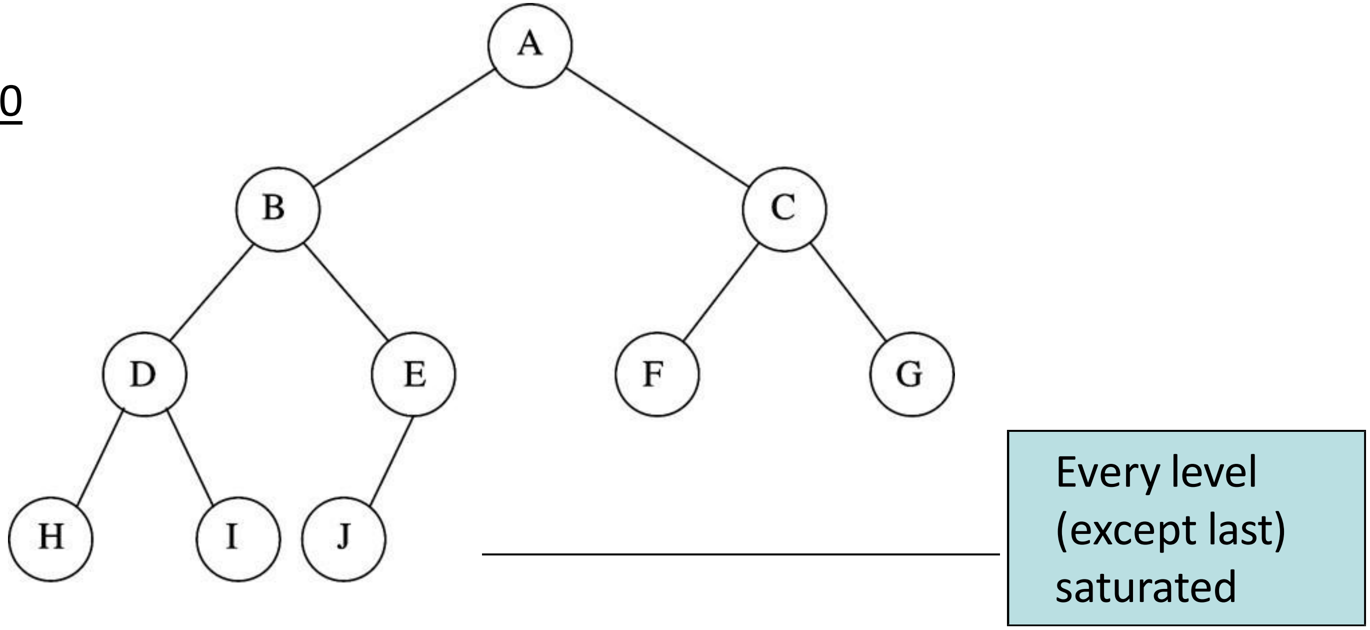
Binary Heap

- A binary heap is a binary tree with two properties
  - Structure property
  - Heap-order property

Binary Heap – Structure Property

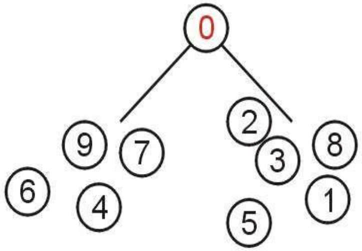
- A binary heap is (almost) complete binary tree
  - Each level (except possibly the bottom most level) is completely filled
  - The bottom most level may be partially filled (from left to right)

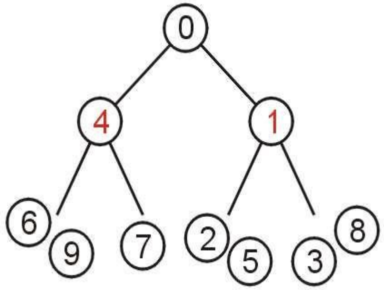
N=10



Binary Heap – Heap-Order Property

- Min-Heap property
  - Key associated with the root is less than or equal to the keys associated with either of the sub-trees (if any)
  - Both of the sub-trees (if any) are also binary min-heaps





- Properties of min-heap
  - A single node is a min-heap
  - Minimum key always at root
  - For every node X, key(parent(X)) ≤ key(X)
  - No relationship between nodes with similar key

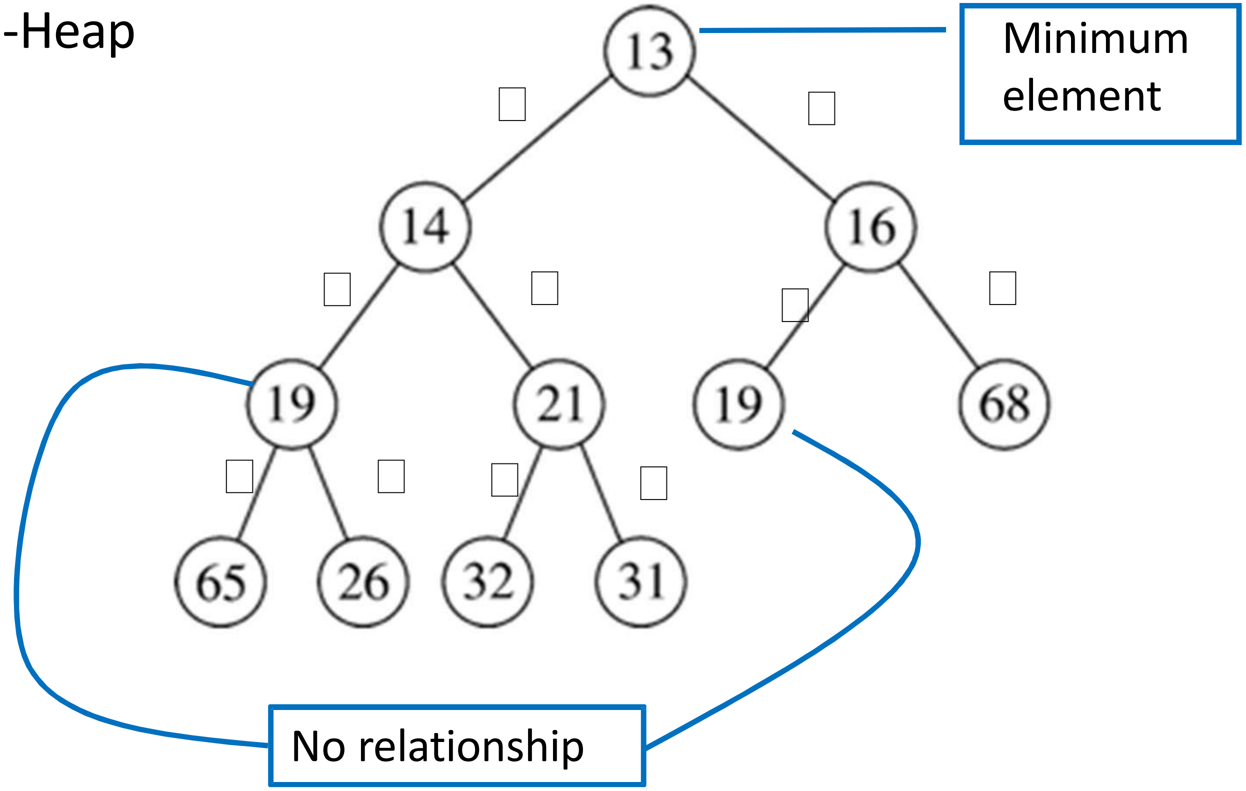
Binary Heap – Heap-Order Property

- Max-Heap property
  - Maximum key at the root
  - For every node X, key(parent(X)) ≥ key(X)

- Insert and deleteMax must maintain heap-order property

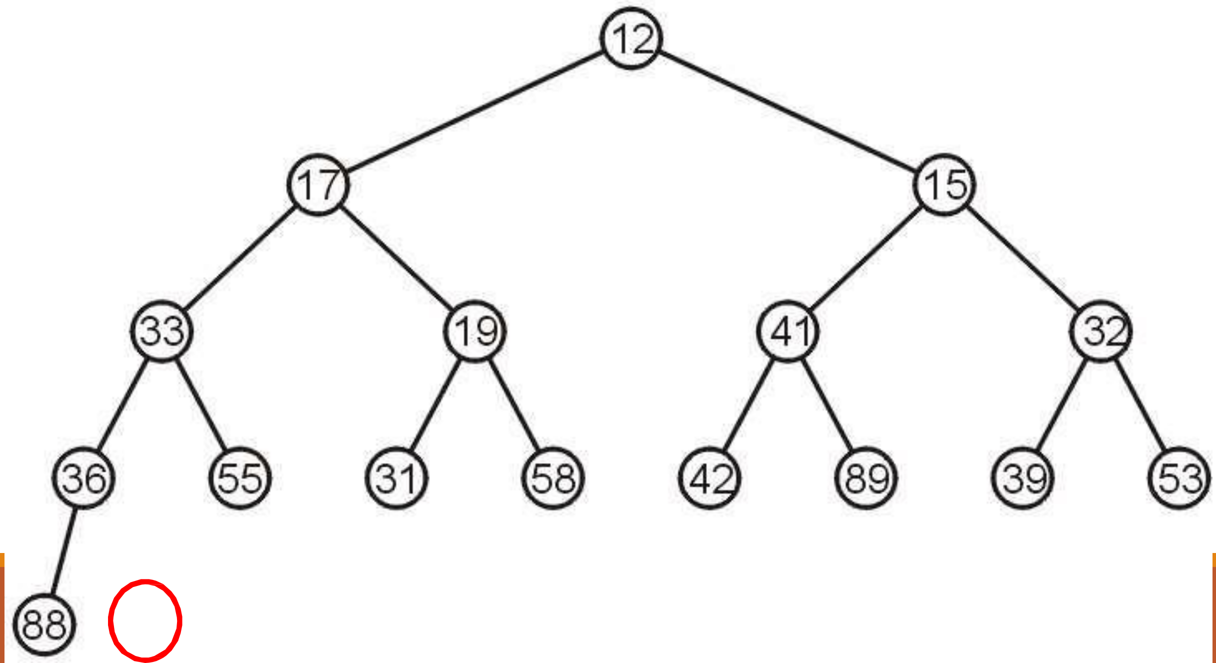
Heap-Order Property – Example

- Min-Heap



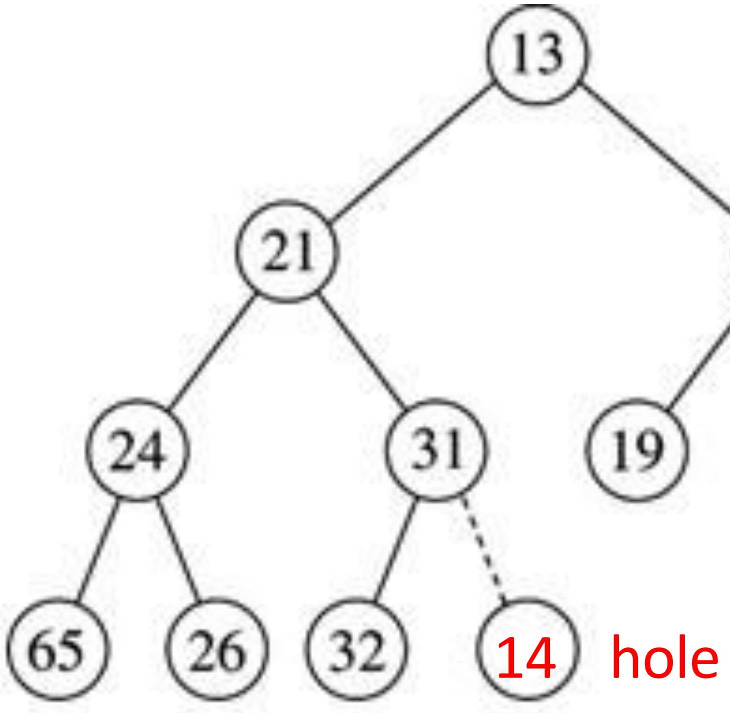
Heap Operations – **insert**

- Insert new element into the heap at the next available slot ("hole")
  - Maintaining (almost) complete binary tree
- Percolate the element up the heap while heap-order property not satisfied

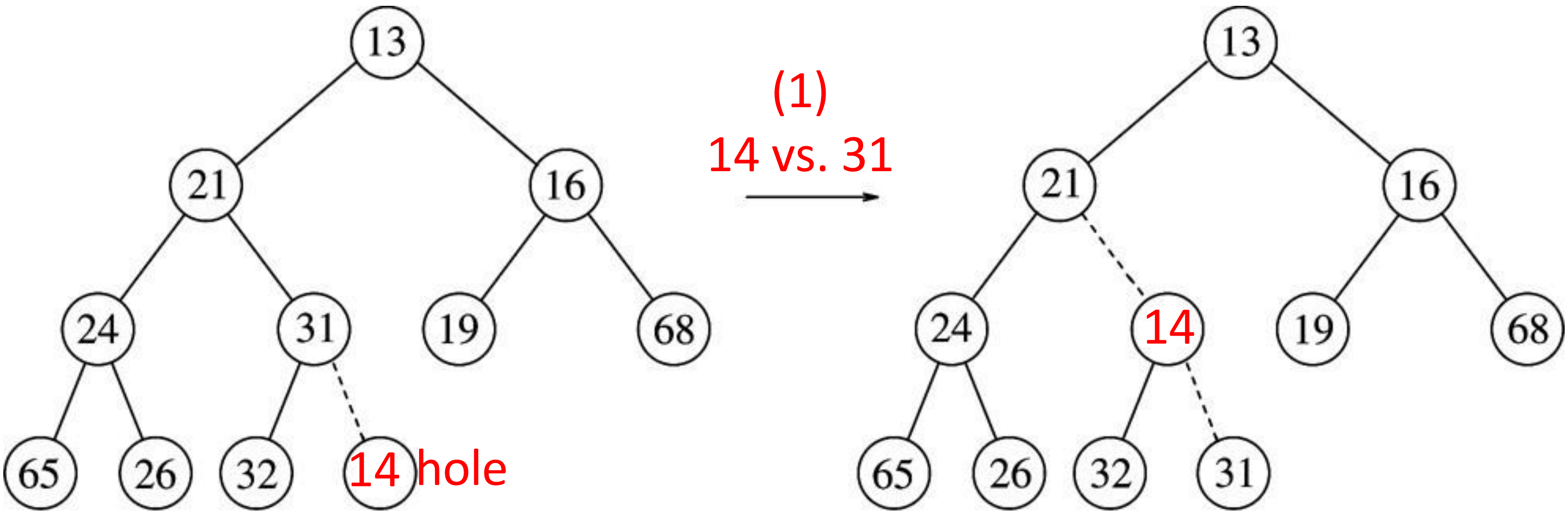


Heap Insert – Example

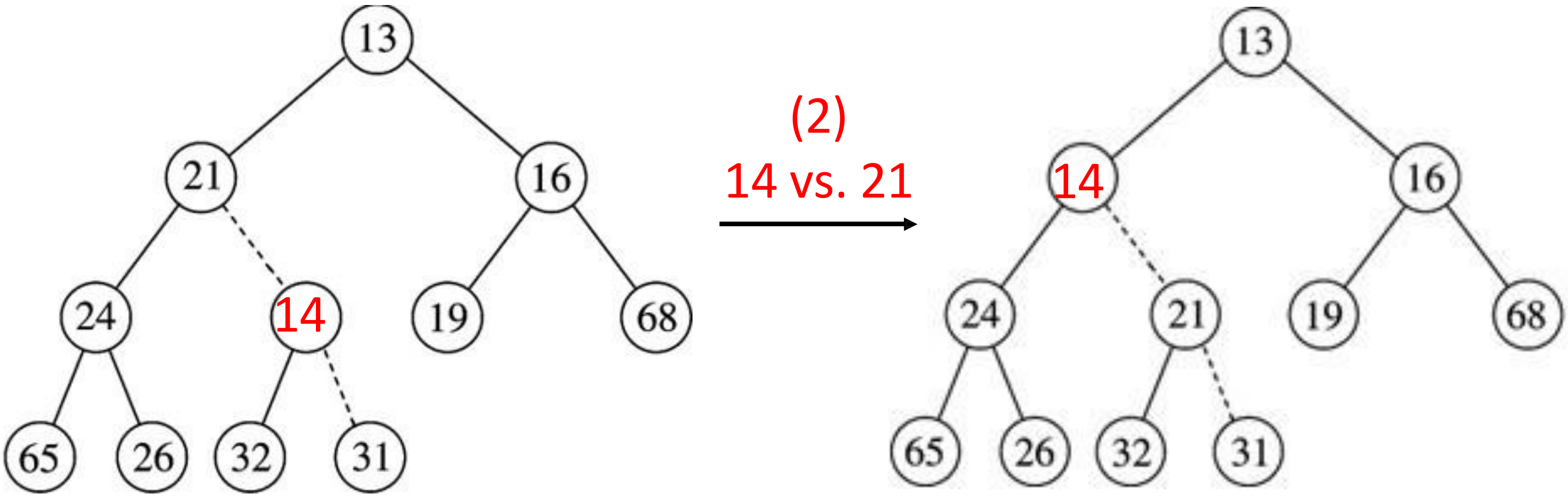
- Insert 14



Heap Insert – Example

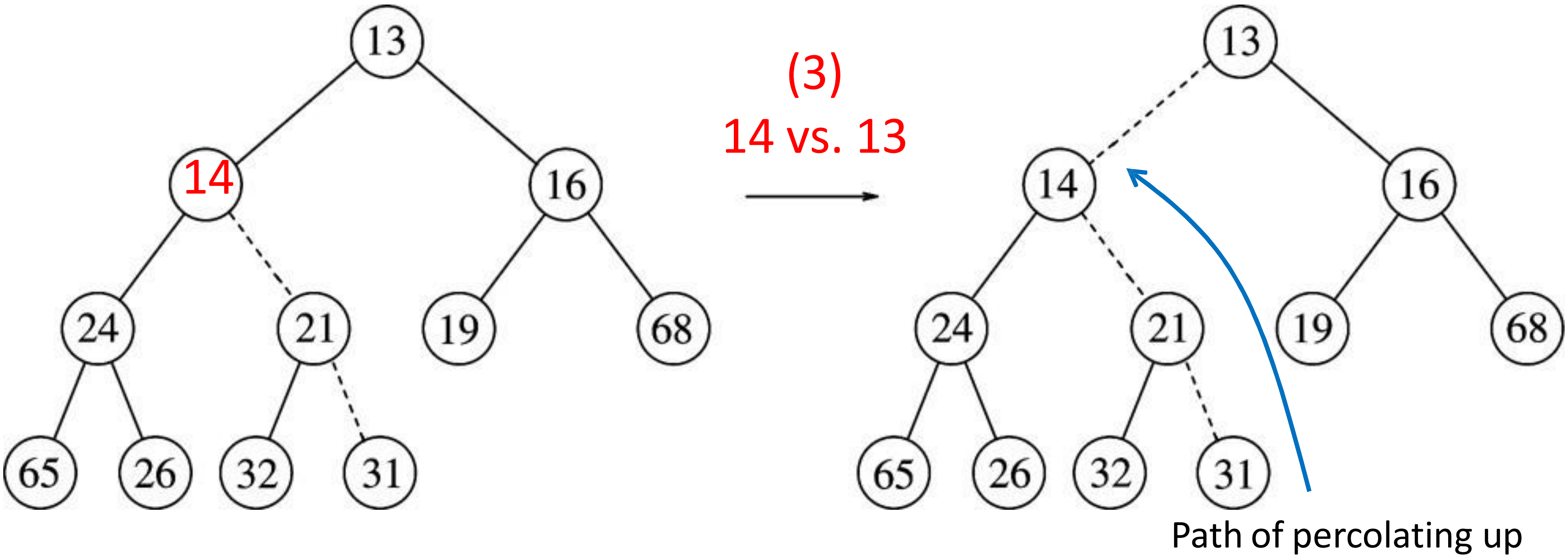


Heap Insert – Example



Heap Insert – Example

- Insert 14

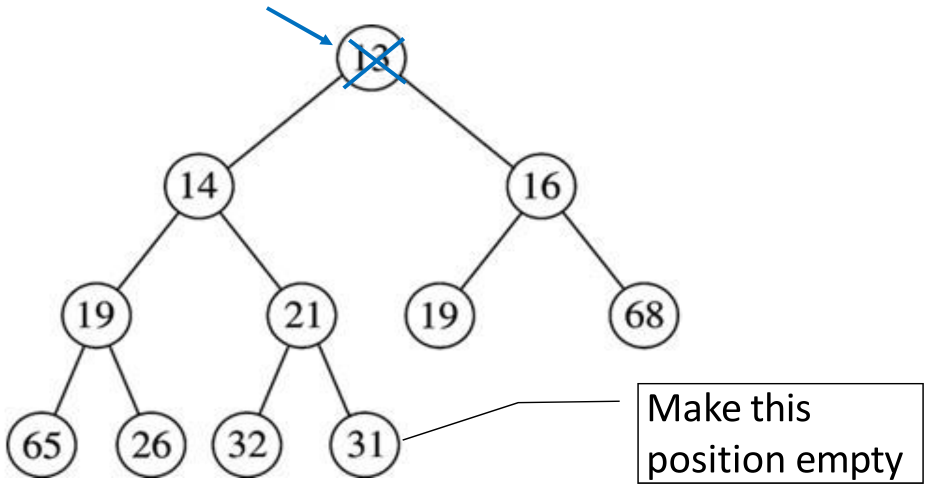


Heap order property

Structure property

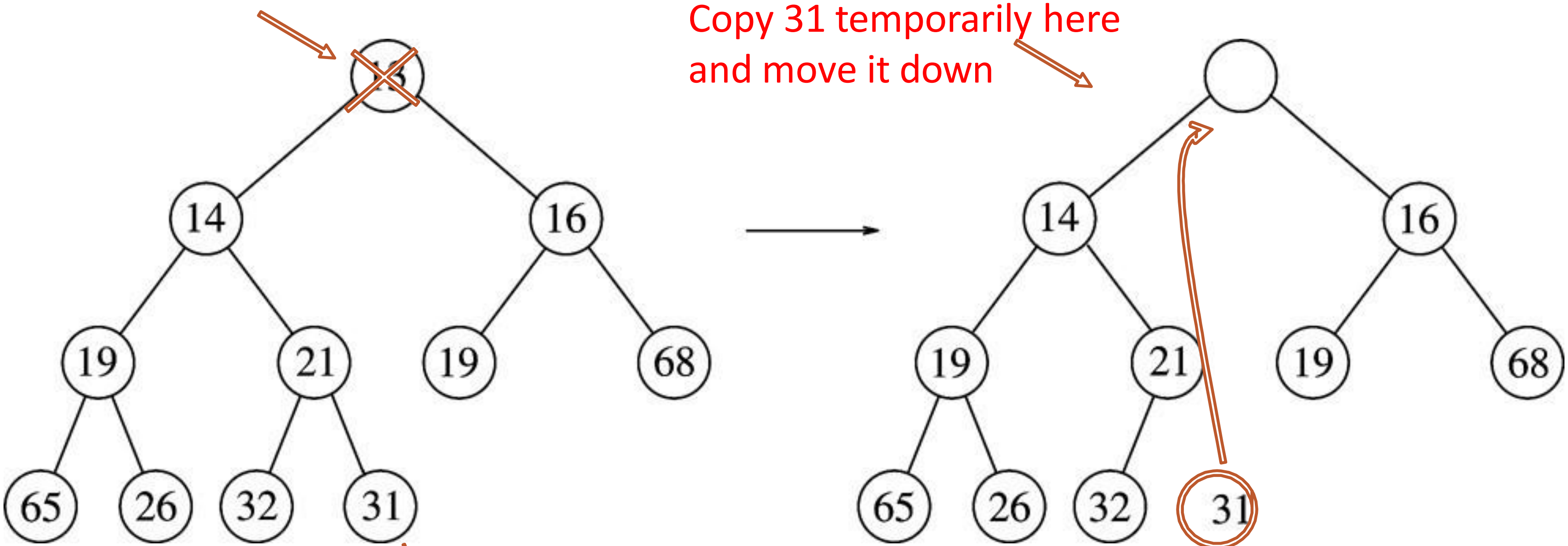
Heap Operation – deleteMin

- Minimum element is always at the root
  - Return the element at the root and delete it
- Heap decreases by one in size
- Move last element of the tree into hole at root
- Percolate down while heap-order property not satisfied



deleteMin – Example

Copy 31 temporarily here and move it down

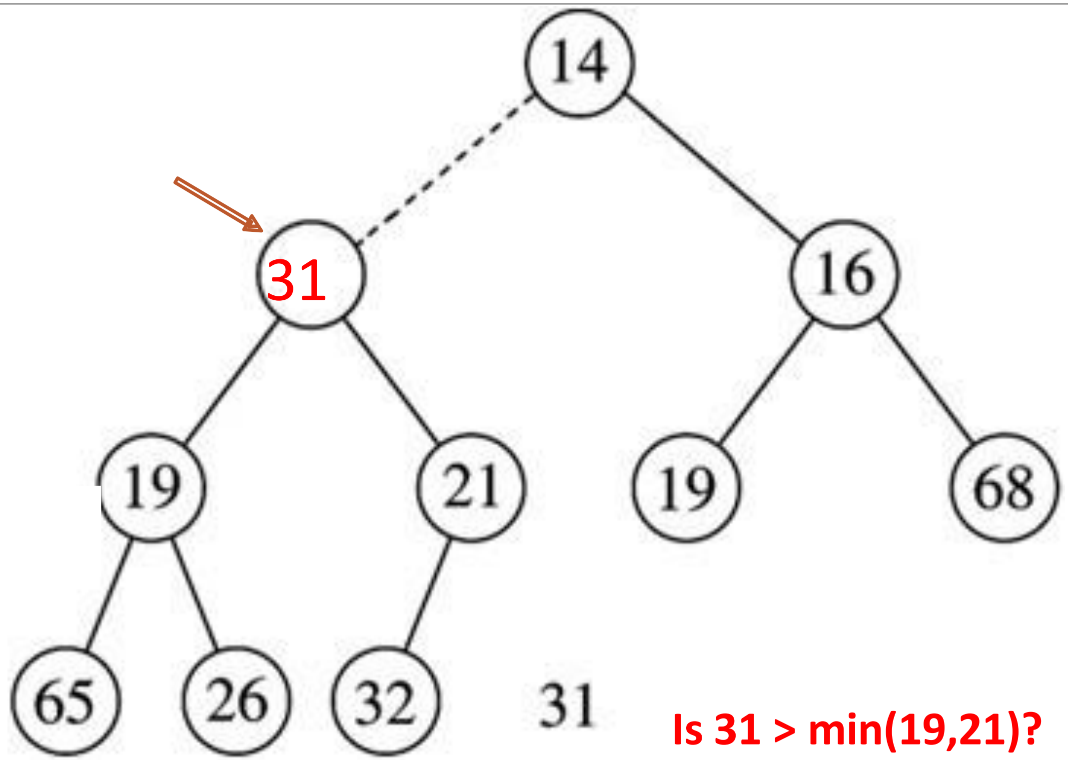


Make this position empty

Is 31 > min(14,16)?

- Yes - swap 31 with min(14,16)

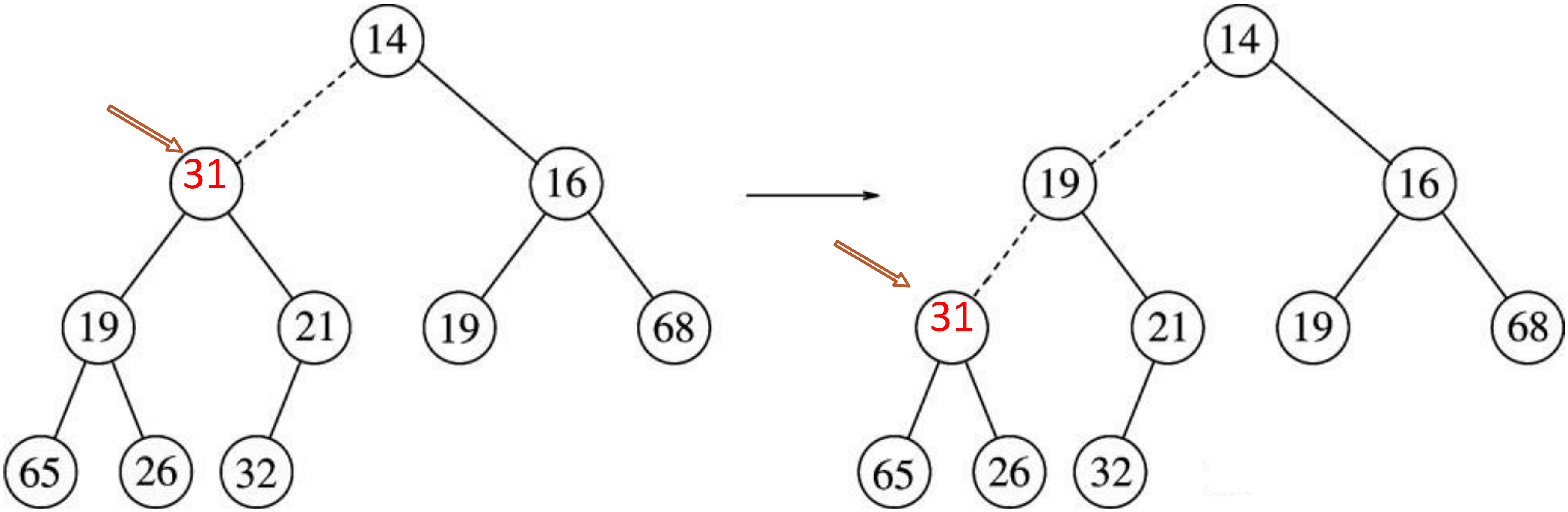
deleteMin – Example



Is 31 > min(19,21)?

- Yes - swap 31 with min(19,21)

deleteMin – Example



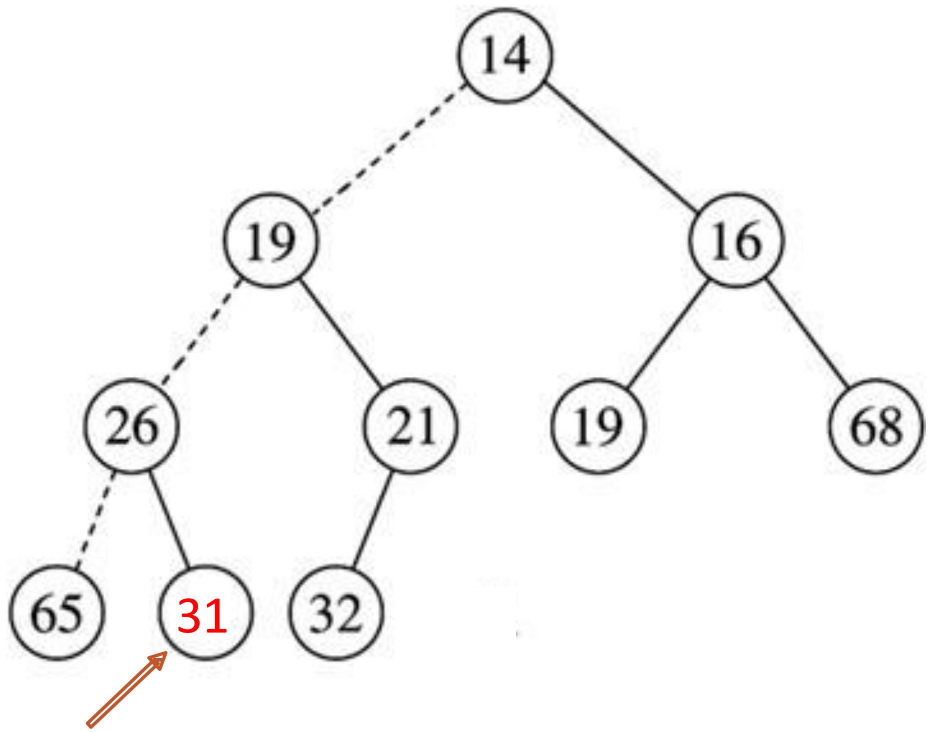
Is 31 > min(19,21)?

- Yes - swap 31 with min(19,21)

Is 31 > min(65,26)?

- Yes - swap 31 with min(65,26)

deleteMin – Example



Array-Based Implementation Of Binary Tree

Left child(i) = at position 2i

Right child(i) = at position 2i + 1

Parent(i) = at position $\lfloor i/2 \rfloor$

N=10

Array representation:

i

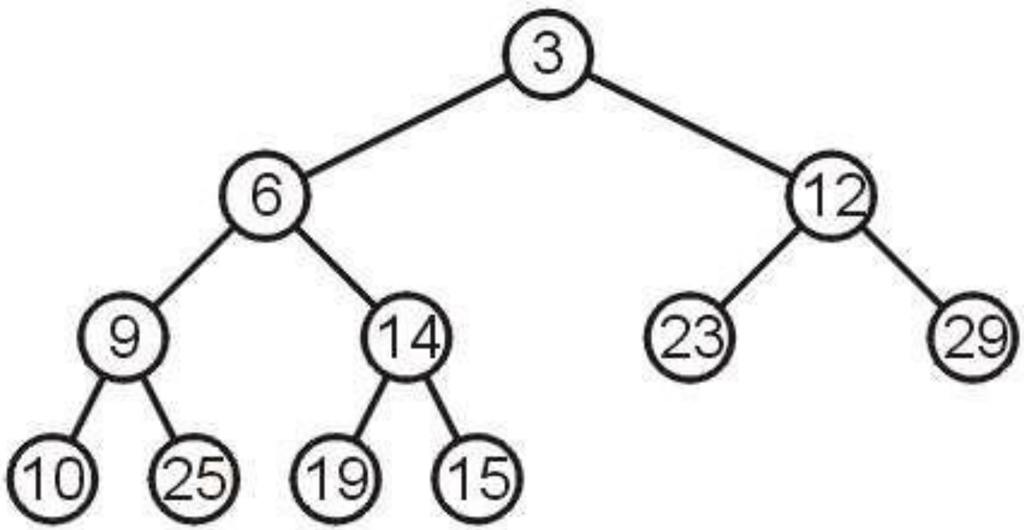
2i + 1

$\lfloor i/2 \rfloor$

2i

Array-Based Implementation Of Binary Heap

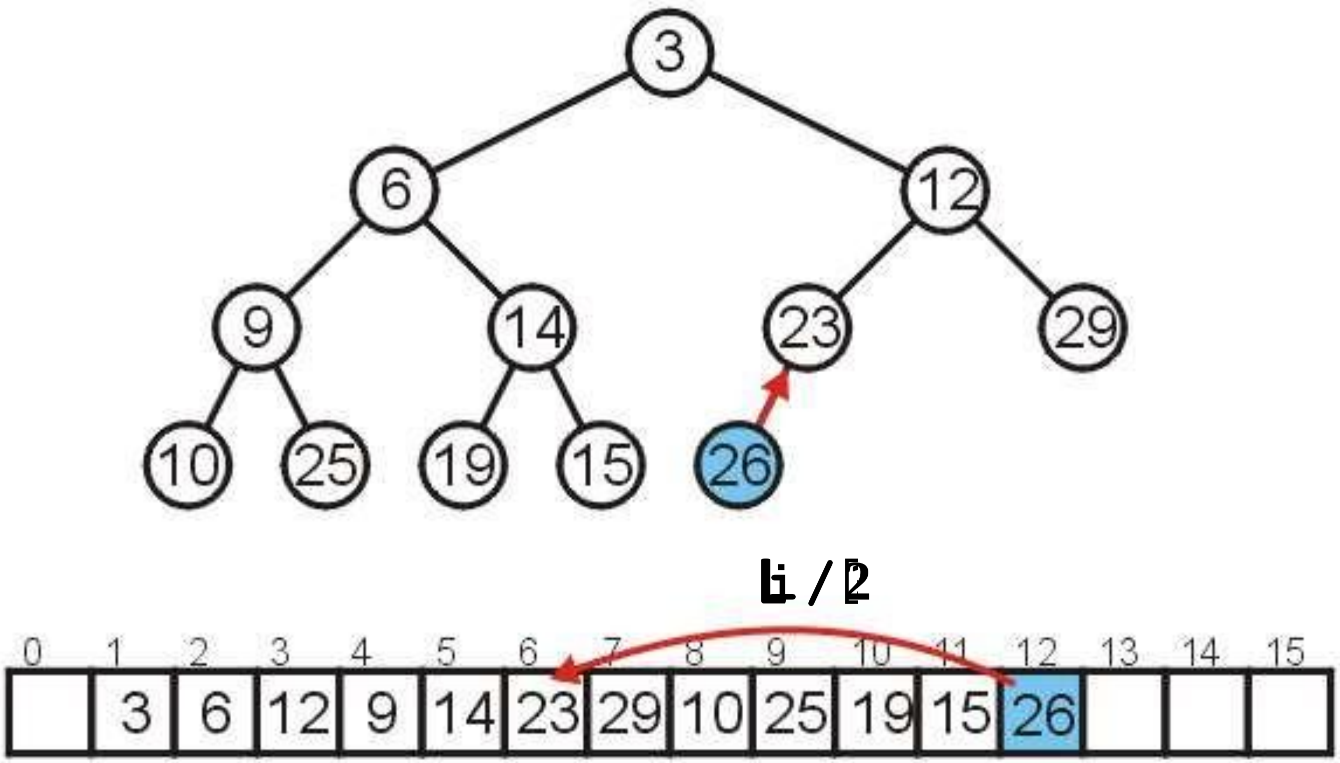
- Consider the following heap, both as a tree and in its array representation



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	3	6	12	9	14	23	29	10	25	19	15				

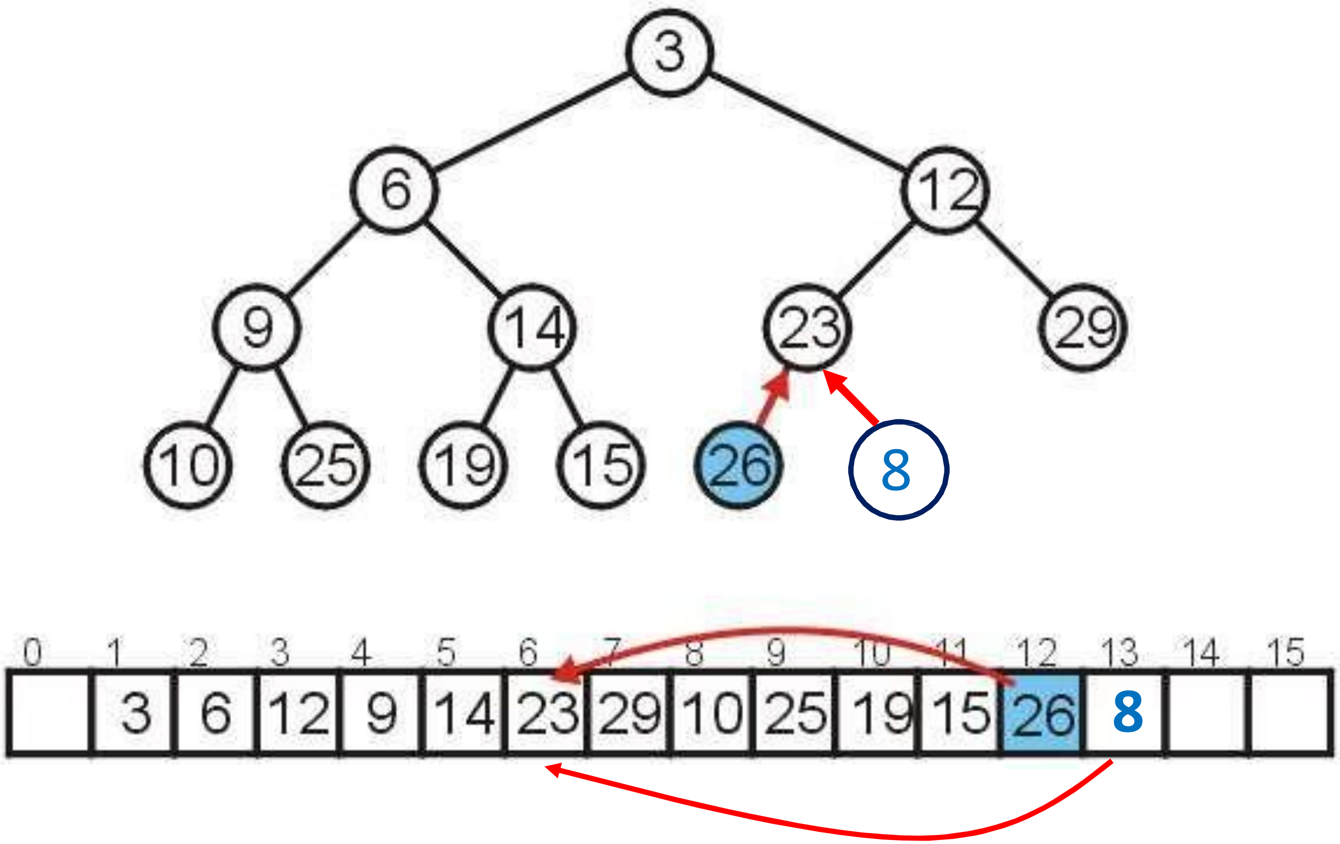
Array-Based Implementation – insert

- Inserting 26 requires no changes



Array-Based Implementation – insert

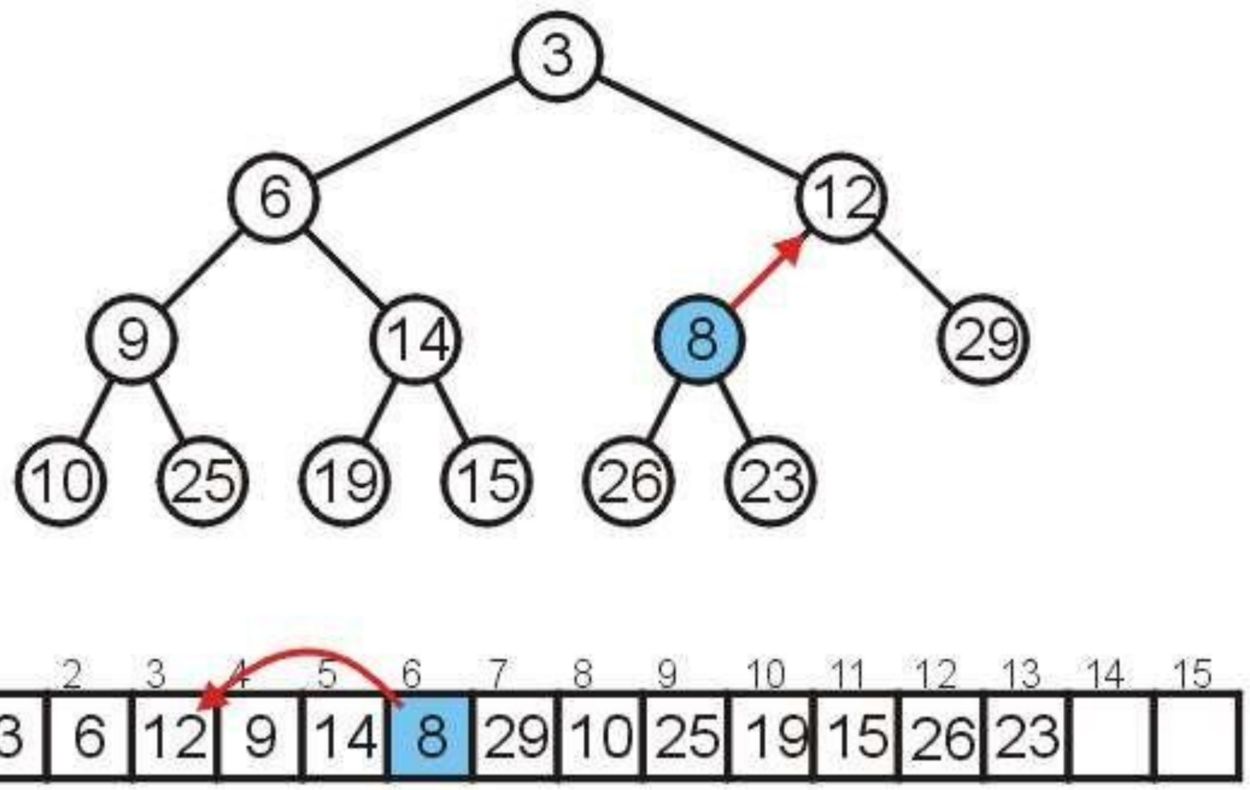
- Inserting 8 requires a few percolations
  - Swap 8 and 23



$\lfloor i / 2 \rfloor$

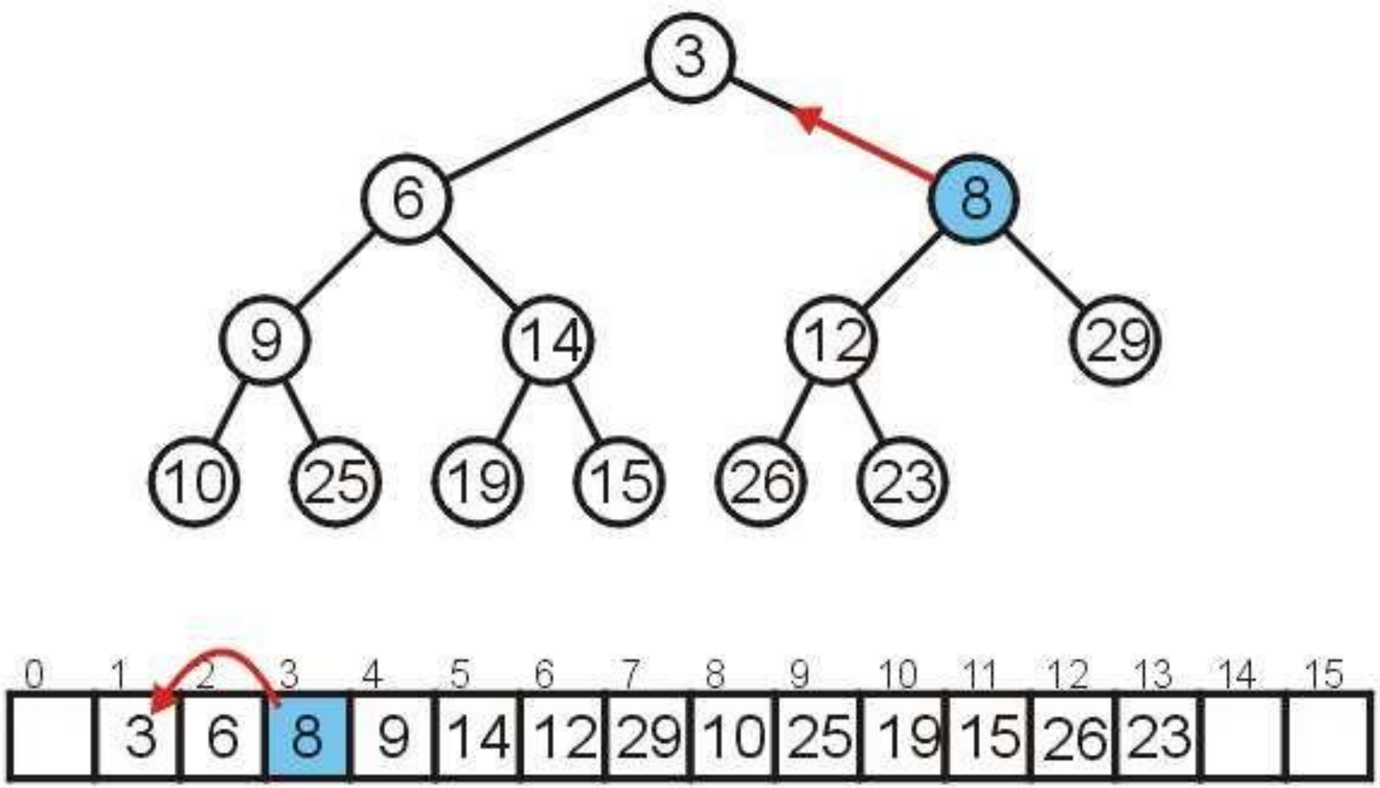
Array-Based Implementation – insert

- Swap 8 and 12



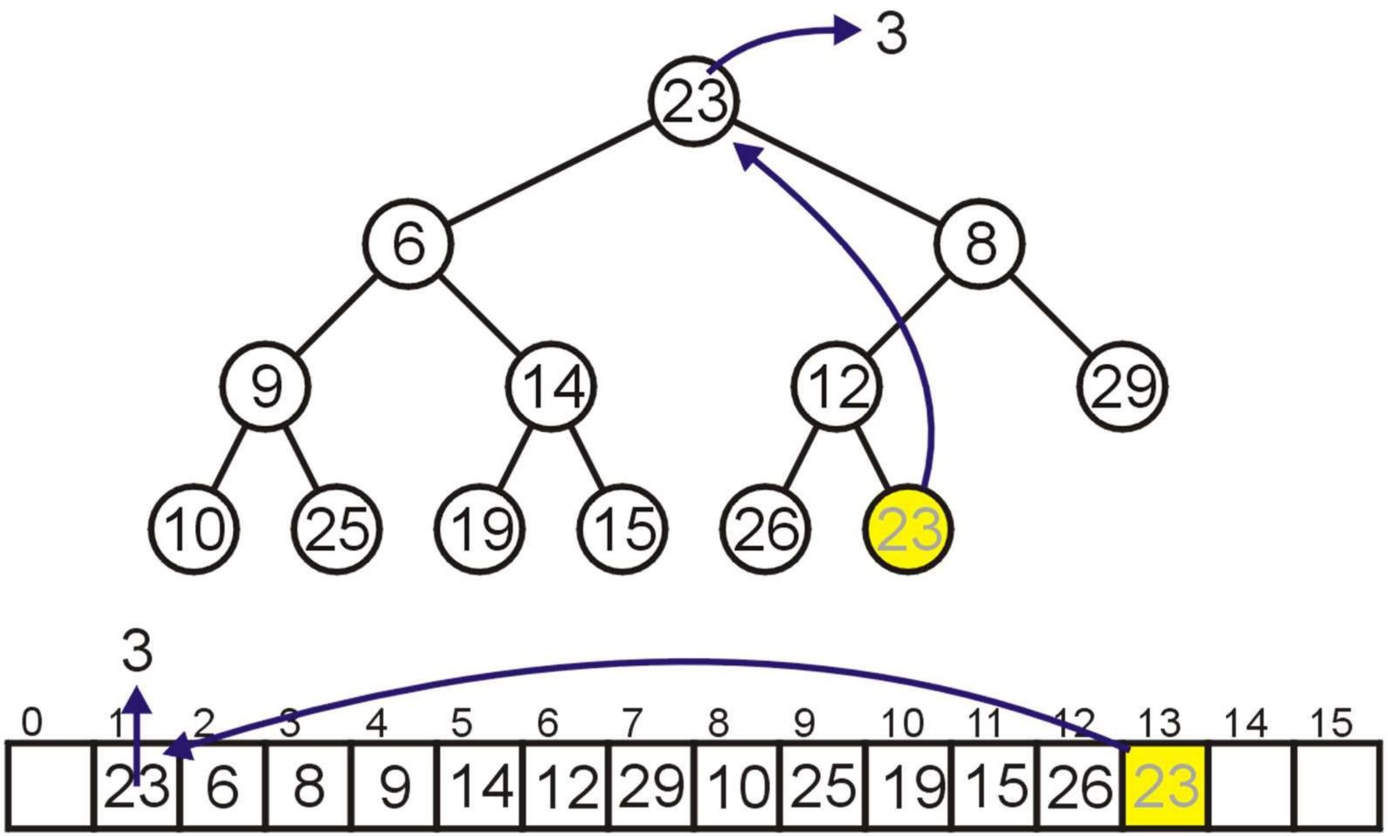
Array-Based Implementation – insert

- At this point, 8 is greater than its parent, so we are finished



Array-Based Implementation – deleteMin

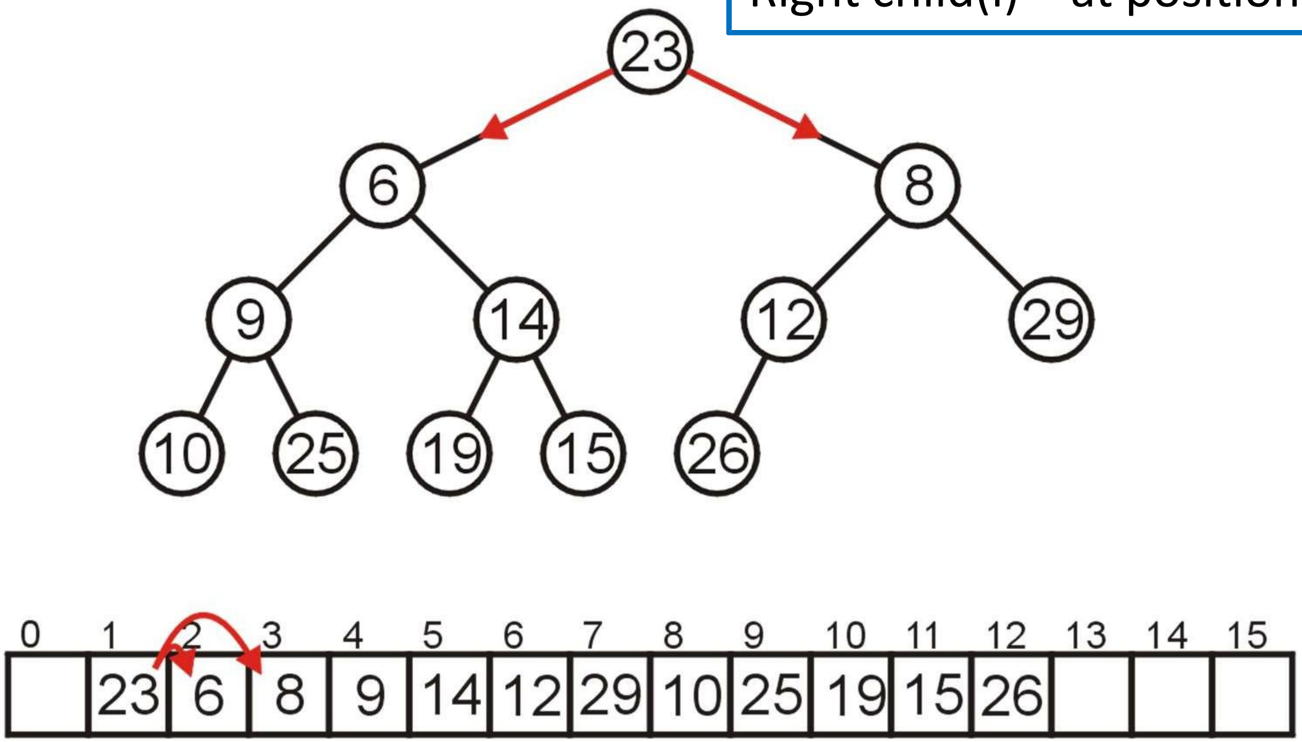
- Removing the top require copy of the last element to the top



Array-Based Implementation – deleteMin

- Percolate down
  - Compare Node 1 with its children: Nodes 2 and 3
  - Swap 23 and 6

Left child(i) = at position 2i
Right child(i) = at position 2i + 1

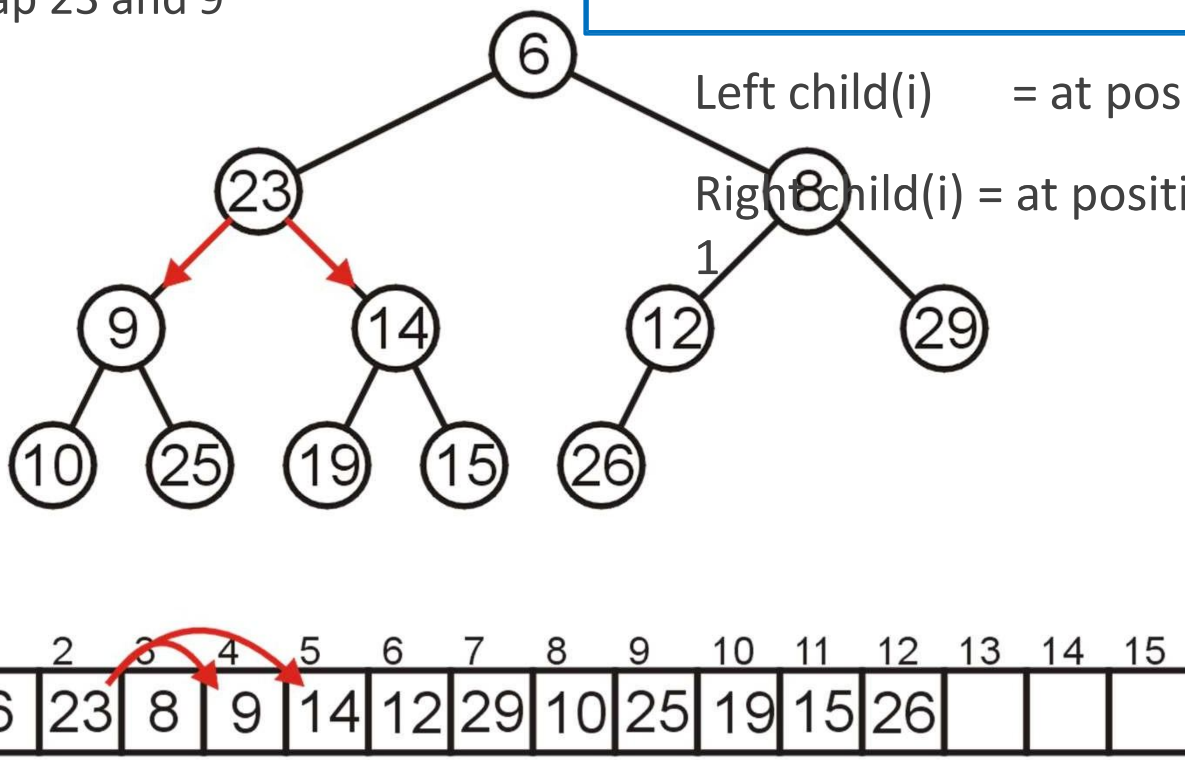


Array-Based Implementation – deleteMin

- Compare Node 2 with its children: Nodes 4 and 5
  - Swap 23 and 9

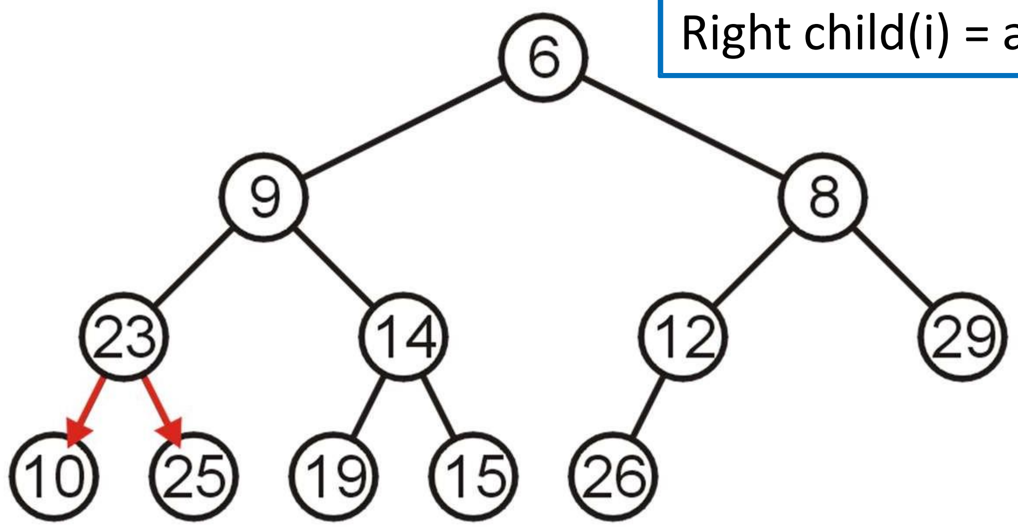
Left child(i) = at position 2i

Right child(i) = at position 2i + 1

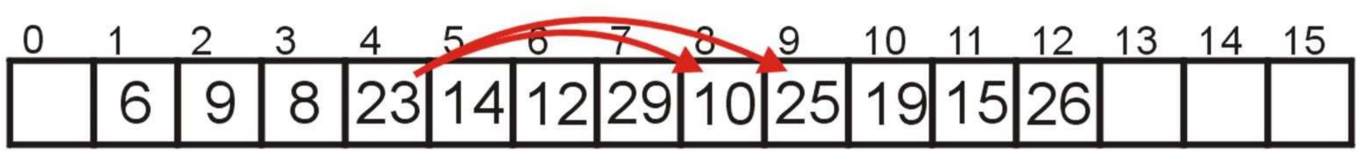


Array-Based Implementation – deleteMin

- Compare Node 4 with its children: Nodes 8 and 9
  - Swap 23 and 10



Left child(i) = at position 2i
Right child(i) = at position 2i + 1

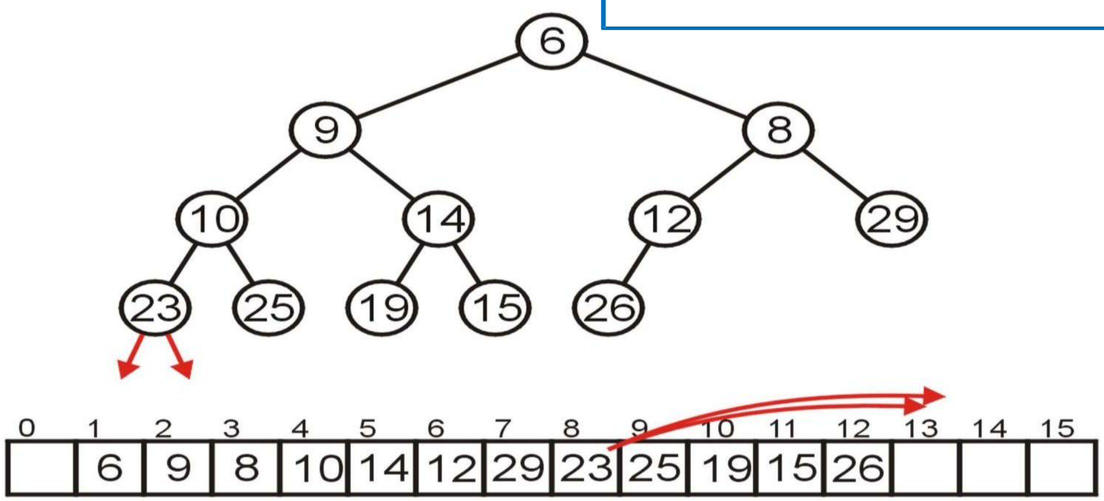


Array-Based Implementation – deleteMin

– Stop

Left child(i) = at position 2i
Right child(i) = at position 2i + 1

- The children of Node 8 are beyond the end of the array.



Any Question So Far?