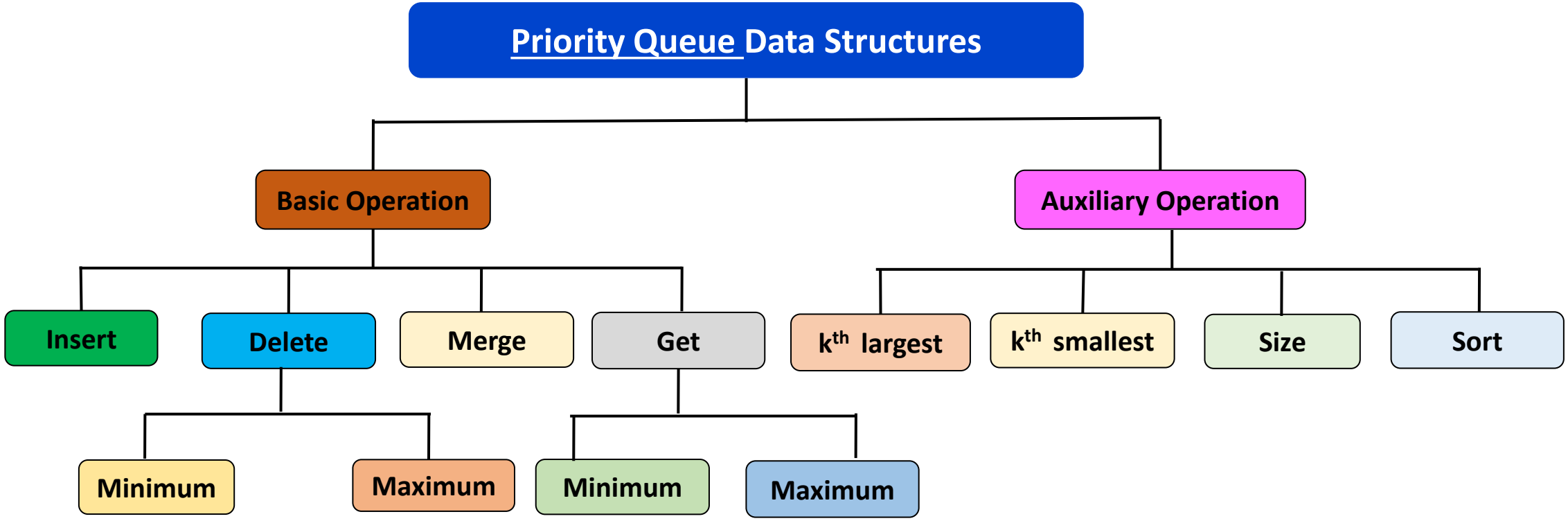
CS2x1:Data Structures and Algorithms

Koteswararao Kondepu

k.kondepu@iitdh.ac.in

Recap: Priority Queue (1)

Define: A *Priority Queue* (like a Queue) has a restriction in accessing elements, but each element has an implicit "*priority*", such that the element with *the highest ("max") priority* is always *DeQueued*, regardless of the order in which it was *EnQueued*.

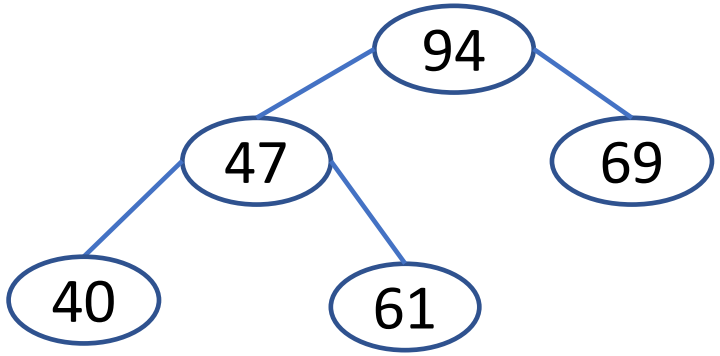


Heap: Merging (1)

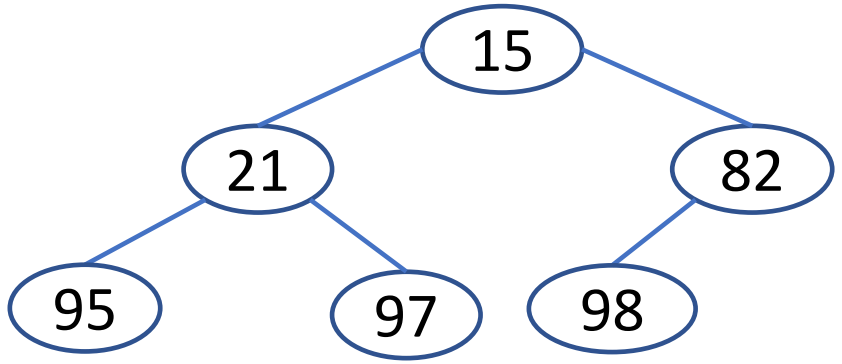
(i) Merging two heap tree properties:

- ✓ Lets consider H1 and H2 are two heap trees. The H2 heap tree is merging with H1 heap tree (i.e., include all the nodes from H2 → H1) .
- ✓ H2 can be a min-heap or max-heap, the resulting tree will be based on the H1 heap tree. If H1 is max-heap, then the resulting tree will be max-heap, otherwise, it will be min-heap.
- ✓ Merging Steps:
  - Step#1: delete the root node from H2
  - Step#2: insert the node into H1 by satisfying the properties of H1.
  - Step#3: repeat the Step#1 and Step#2 till the H2 is not empty

Example: *H1: 94, 47, 69, 40, 61;*



H2: 15, 21, 82, 95, 97, 98

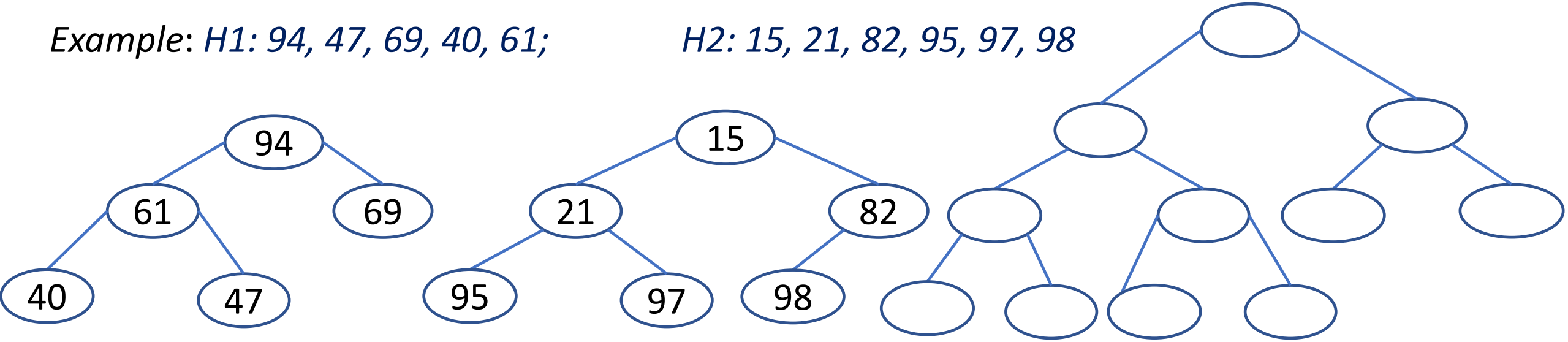


Heap: Merging (2)

(i) Merging two heap tree properties:

- ✓ Lets consider H1 and H2 are two heap trees. The H2 heap tree is merging with H1 heap tree (i.e., include all the nodes from H2 → H1) .
- ✓ H2 can be a min-heap or max-heap, the resulting tree will be based on the H1 heap tree. If H1 is max-tree, then the resulting tree will be max-heap, otherwise, it will be min-heap.
- ✓ Merging Steps:
  - Step#1: delete the right-most in the last level of H2
  - Step#2: insert the node in to H1 by satisfying the properties of H1.
  - Step#3: repeat the Step#1 and Step#2 till the H2 is not empty

Example: *H1: 94, 47, 69, 40, 61;* *H2: 15, 21, 82, 95, 97, 98*



Heap: Applications

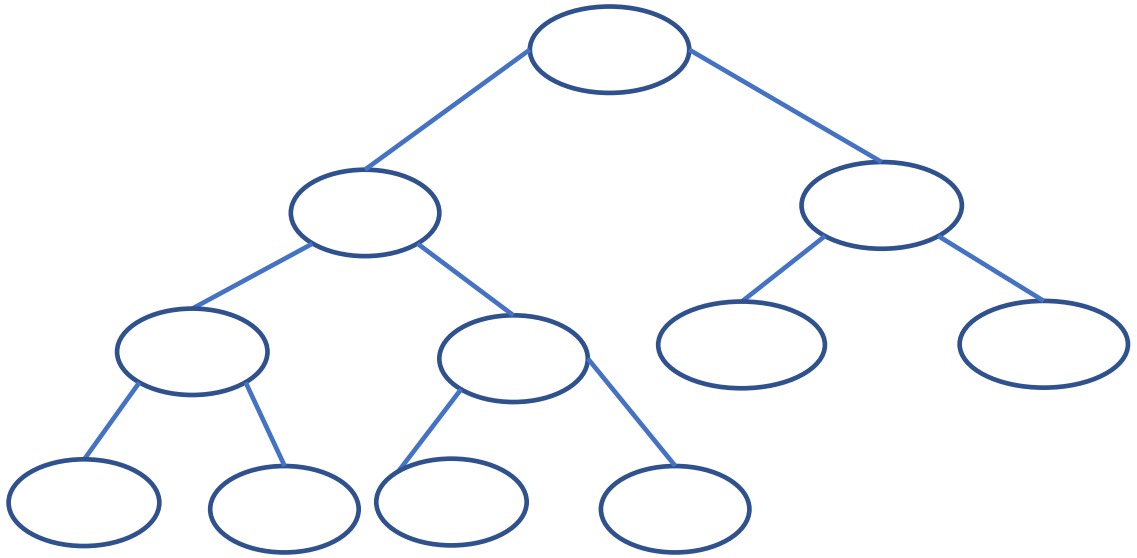
(i) Sorting; (ii) Priority Queue

(i) Sorting method based on the heap tree → heap sort

- ✓ Sorting Steps:
  - Step#1: Build the heap tree with the given set of data
  - Step#2: (i) Delete the root node from the heap
    (ii) Rebuild the heap after the deletion
    (iii) Print the deleted node in the output
  - Step#3: Repeat Step#2 until the heap tree is empty

Example: *97, 96, 67, 63, 83, 31, 57, 3, 45, 12, 74*

Step#1

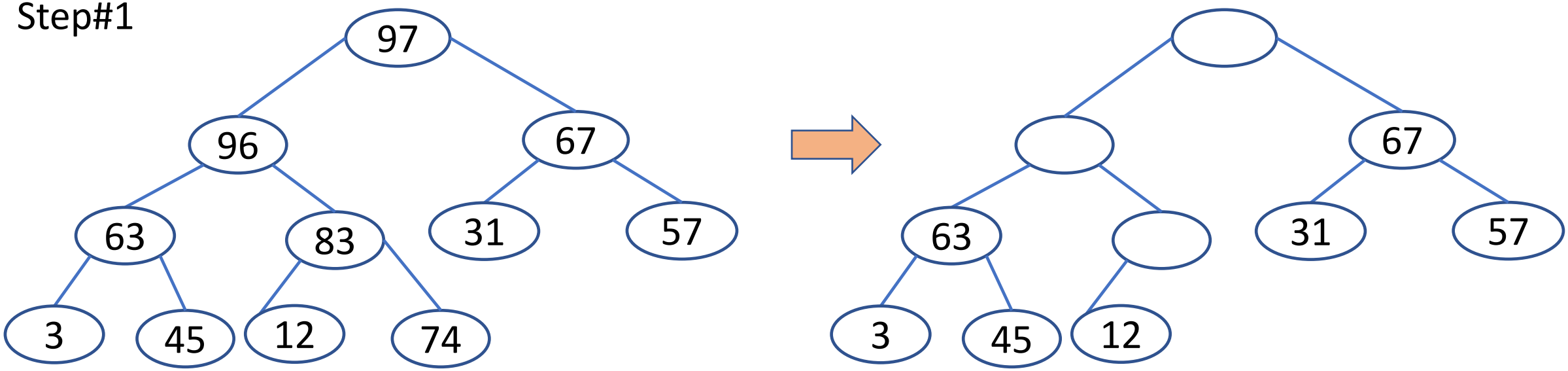


Heap: Sorting Applications (1)

(i) Sorting;

- Step#2: (i) Delete the root node from the heap
  (ii) Rebuild the heap after the deletion
  (iii) Print the deleted node in the output
- Step#3: Repeat Step#2 until the heap tree is empty

Step#1



After deleting node 97

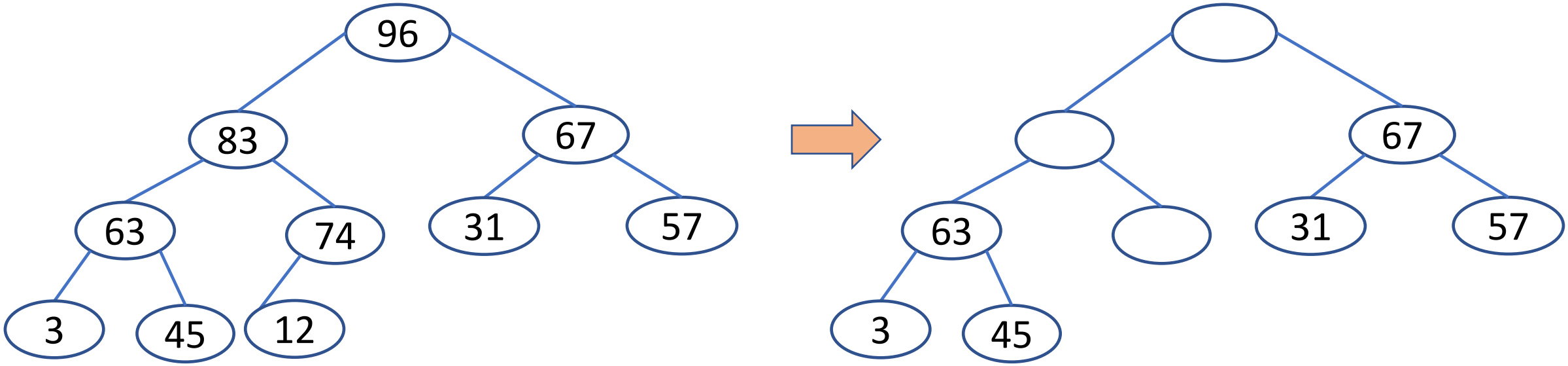
Output: ***97***

Heap: Sorting Applications (2)

(i) Sorting;

- Step#2: (i) Delete the root node from the heap
  (ii) Rebuild the heap after the deletion
  (iii) Print the deleted node in the output
- Step#3: Repeat Step#2 until the heap tree is empty

Step#2



After deleting node 96

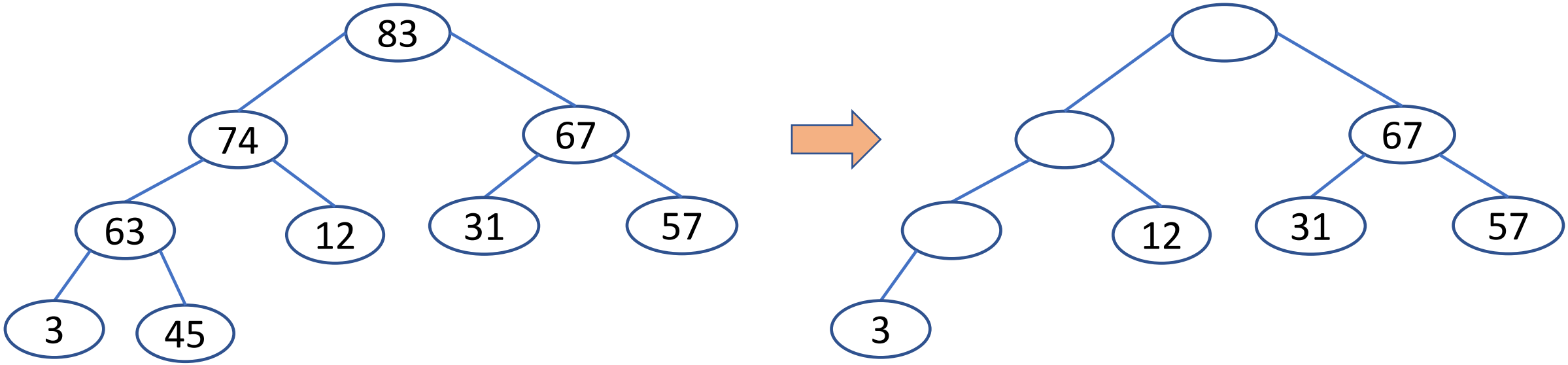
Output: ***97, 96***

Heap: Sorting Applications (3)

(i) Sorting;

- Step#2: (i) Delete the root node from the heap
  (ii) Rebuild the heap after the deletion
  (iii) Print the deleted node in the output
- Step#3: Repeat Step#2 until the heap tree is empty

Step#2



After deleting node 83

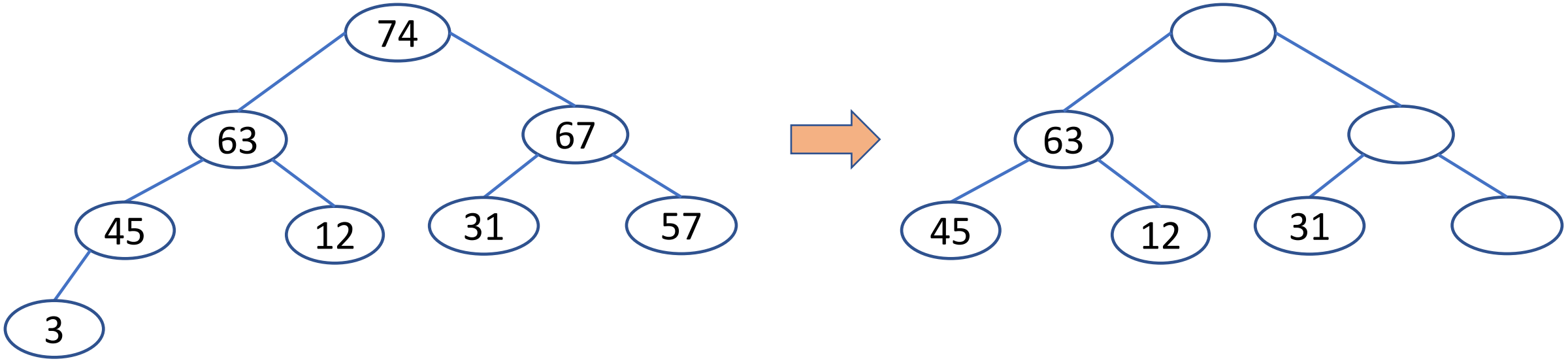
Output: ***97, 96, 83***

Heap: Sorting Applications (4)

(i) Sorting;

- Step#2: (i) Delete the root node from the heap
  (ii) Rebuild the heap after the deletion
  (iii) Print the deleted node in the output
- Step#3: Repeat Step#2 until the heap tree is empty

Step#2



After deleting node 74

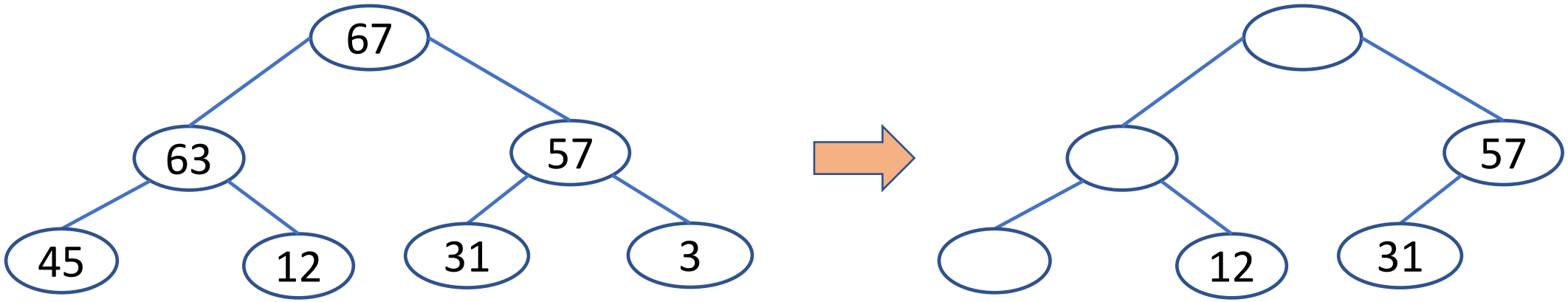
Output: ***97, 96, 83, 74***

Heap: Sorting Applications (5)

(i) Sorting;

- Step#2: (i) Delete the root node from the heap
  (ii) Rebuild the heap after the deletion
  (iii) Print the deleted node in the output
- Step#3: Repeat Step#2 until the heap tree is empty

Step#2



After deleting node 67

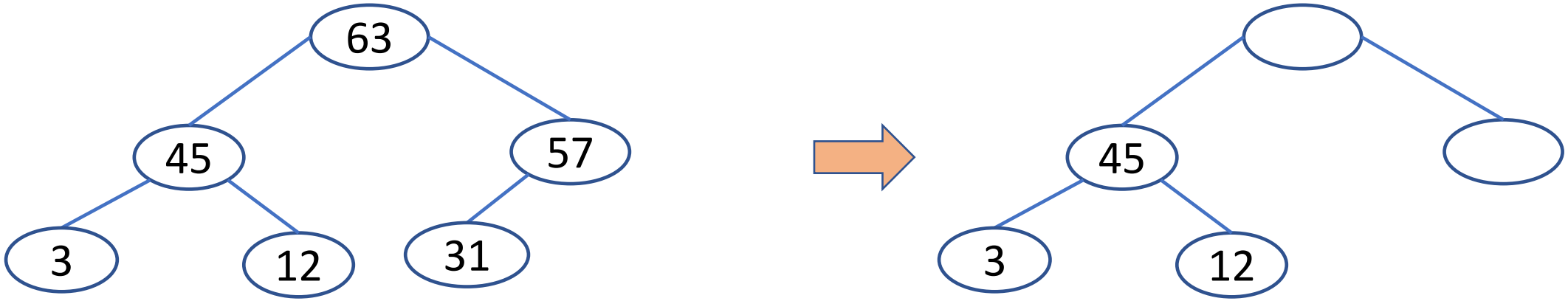
Output: ***97, 96, 83, 74, 67***

Heap: Sorting Applications (6)

(i) Sorting;

- Step#2: (i) Delete the root node from the heap
  (ii) Rebuild the heap after the deletion
  (iii) Print the deleted node in the output
- Step#3: Repeat Step#2 until the heap tree is empty

Step#2



After deleting node 63

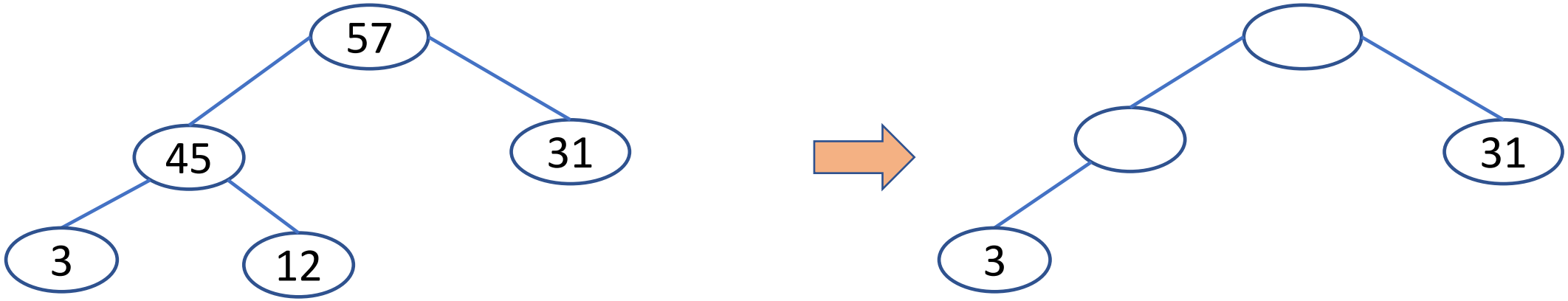
Output: ***97, 96, 83, 74, 67, 63***

Heap: Sorting Applications (7)

(i) Sorting;

- Step#2: (i) Delete the root node from the heap
  (ii) Rebuild the heap after the deletion
  (iii) Print the deleted node in the output
- Step#3: Repeat Step#2 until the heap tree is empty

Step#2



After deleting node 57

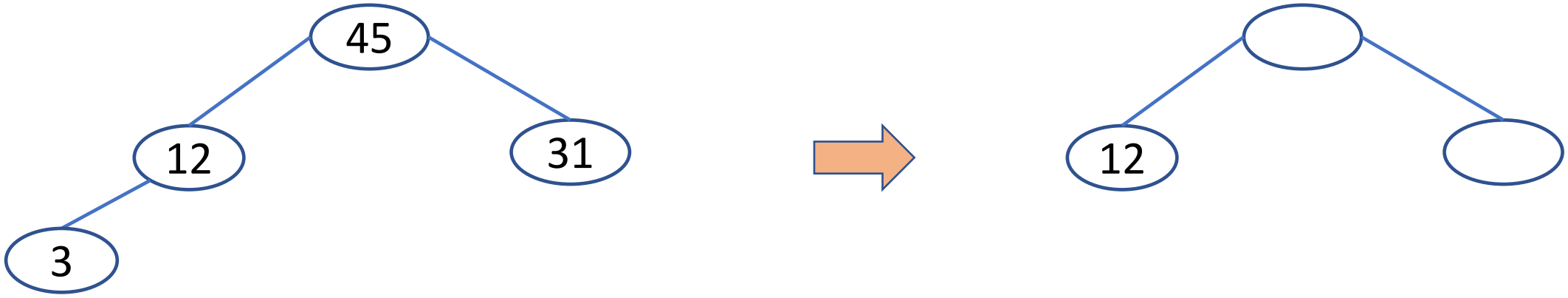
Output: ***97, 96, 83, 74, 67, 63, 57***

Heap: Sorting Applications (8)

(i) Sorting;

- Step#2: (i) Delete the root node from the heap
  (ii) Rebuild the heap after the deletion
  (iii) Print the deleted node in the output
- Step#3: Repeat Step#2 until the heap tree is empty

Step#2



After deleting node 45

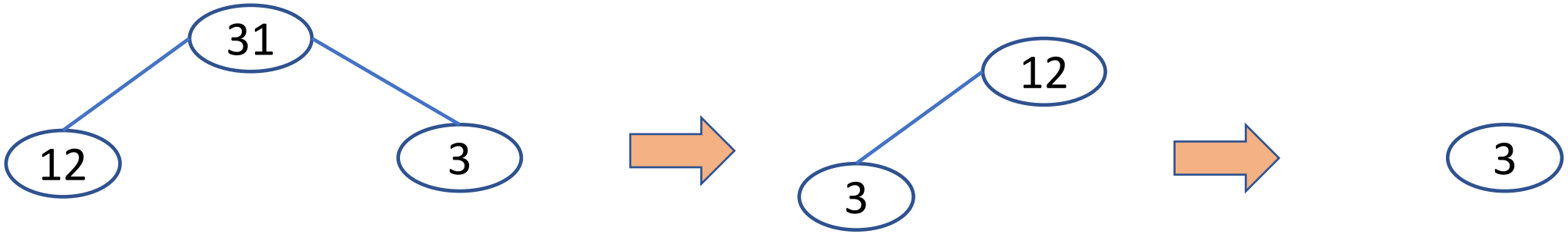
Output: ***97, 96, 83, 74, 67, 63, 57, 45***

Heap: Sorting Applications (9)

(i) Sorting;

- Step#2: (i) Delete the root node from the heap
  (ii) Rebuild the heap after the deletion
  (iii) Print the deleted node in the output
- Step#3: Repeat Step#2 until the heap tree is empty

Step#2



After deleting node 31 and heapifying

After deleting node 12

Final Output: ***97, 96, 83, 74, 67, 63, 57, 45, 31, 12, 3***

Finding the k^{th} largest element using a max heap

Steps:

Step#1: Build a max heap from the given set of elements.

Step#2: Perform k-1 *DeleteMax* operations on the max heap. This will remove the k-1 largest elements from the max heap.

Step#3: The k^{th} largest element will now be at the root of the max heap; *return* the k^{th} largest element as the result.

Input: 94, 61, 69, 40, 47; find the 4th largest element from the given input set?

Finding the k^{th} smallest element using a min heap

Steps:

Step#1: Build a *min heap* from the given set of elements.

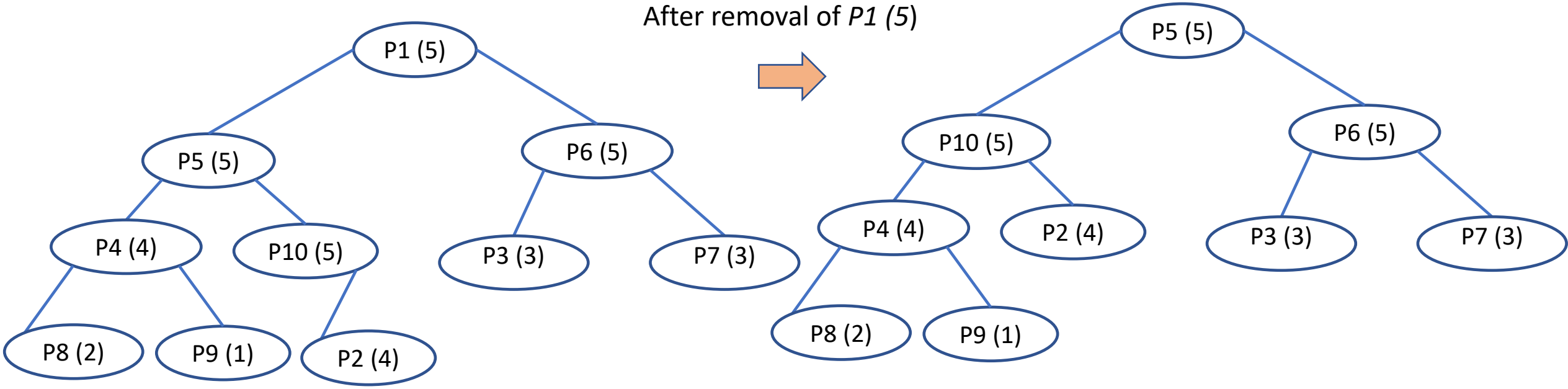
Step#2: Perform k-1 *DeleteMin* operations on the *min heap*. This will remove the k-1 smallest elements from the min heap.

Step#3: The k^{th} smallest element will now be at the root of the min heap; *return* the k^{th} smallest element as the result.

Input: 94, 61, 69, 40, 47; find the 3rd smallest element from the given input set?

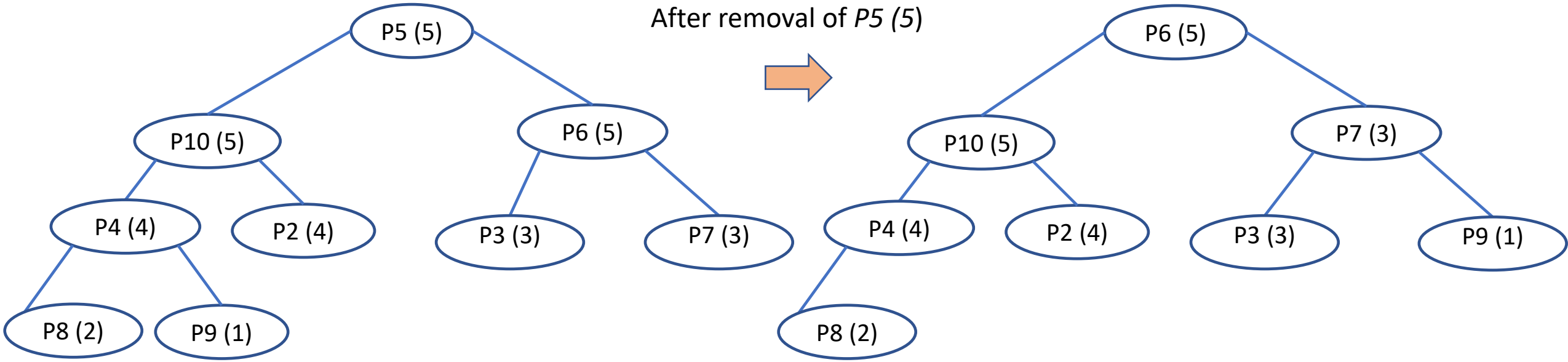
Heap: Priority Queue Applications (1)

Process:	*P1*	*P2*	*P3*	*P4*	*P5*	*P6*	*P7*	*P8*	*P9*	*P10*
Priority:	5	4	3	4	5	5	3	2	1	5



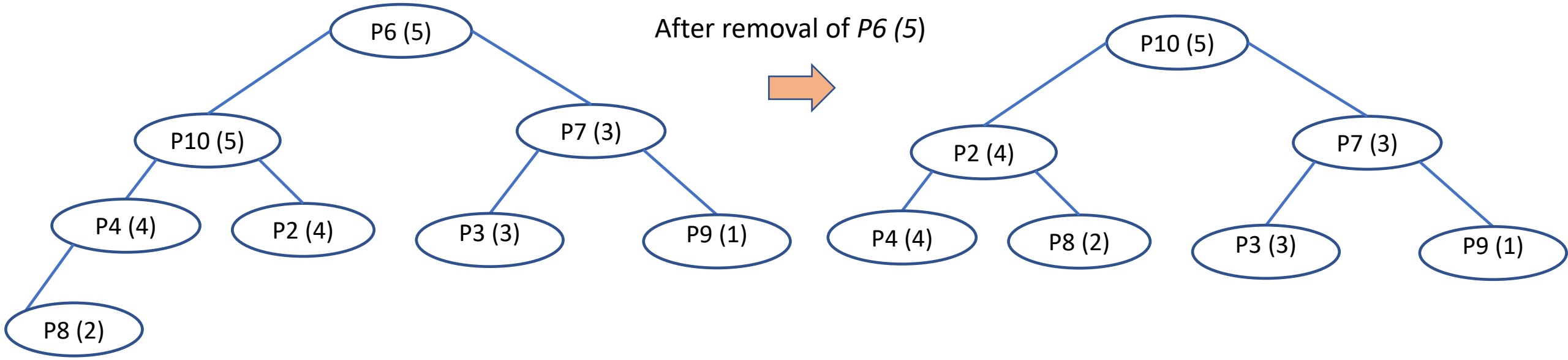
Heap: Priority Queue Applications (2)

Process:	*P1*	*P2*	*P3*	*P4*	*P5*	*P6*	*P7*	*P8*	*P9*	*P10*
Priority:	5	4	3	4	5	5	3	2	1	5



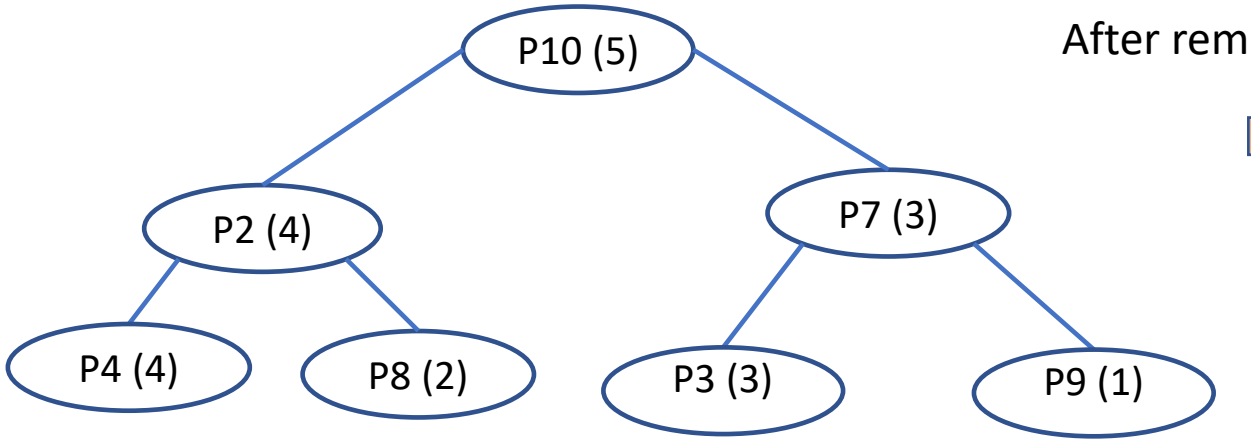
Heap: Priority Queue Applications (3)

Process:	*P1*	*P2*	*P3*	*P4*	*P5*	*P6*	*P7*	*P8*	*P9*	*P10*
Priority:	5	4	3	4	5	5	3	2	1	5

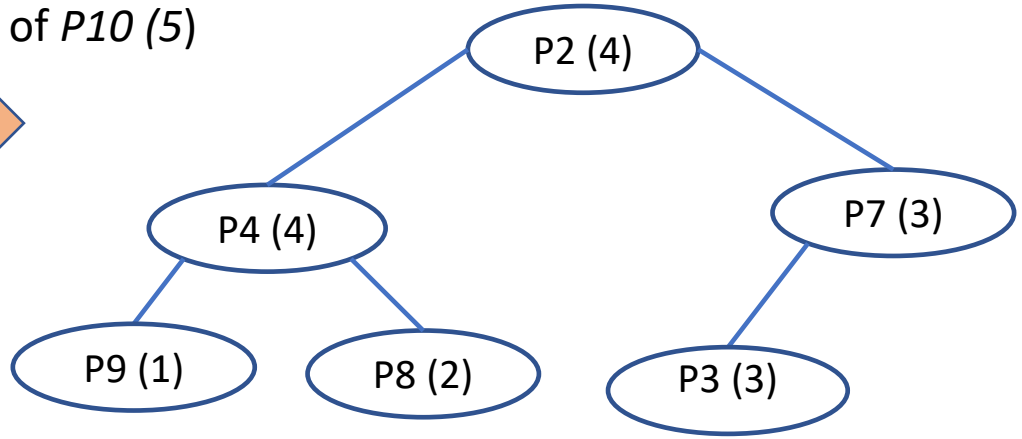


Heap: Priority Queue Applications (4)

Process:	*P1*	*P2*	*P3*	*P4*	*P5*	*P6*	*P7*	*P8*	*P9*	*P10*
Priority:	5	4	3	4	5	5	3	2	1	5

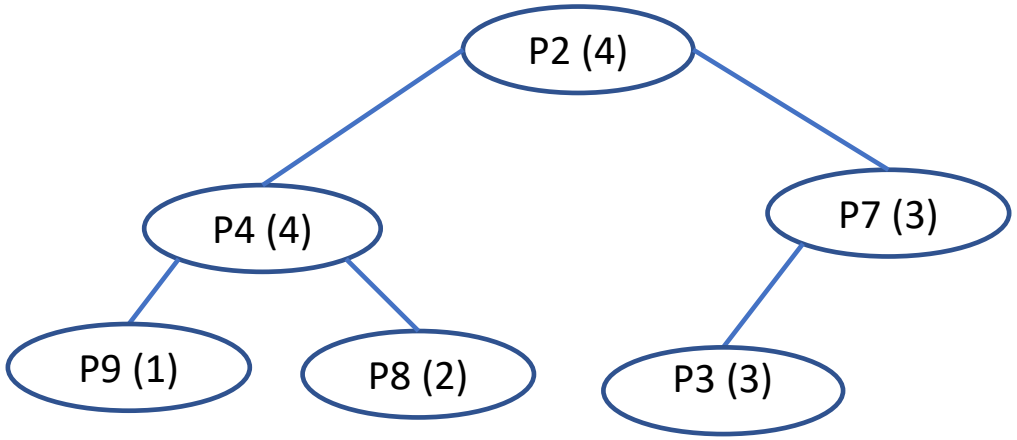


After removal of *P10 (5)*

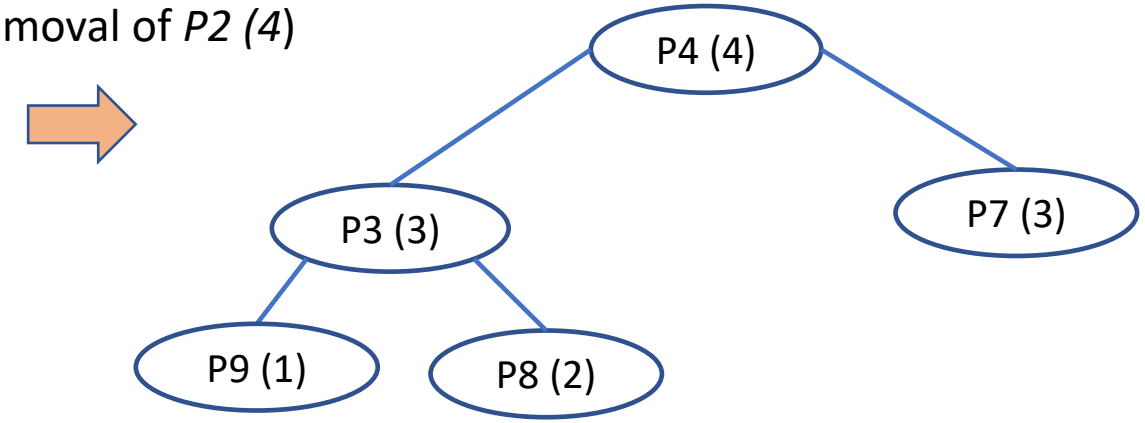


Heap: Priority Queue Applications (5)

Process:	*P1*	*P2*	*P3*	*P4*	*P5*	*P6*	*P7*	*P8*	*P9*	*P10*
Priority:	5	4	3	4	5	5	3	2	1	5

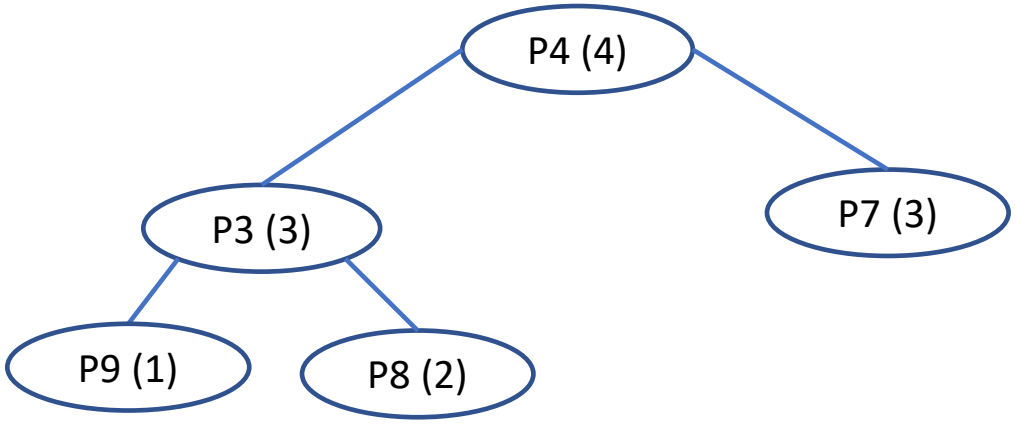


After removal of *P2 (4)*

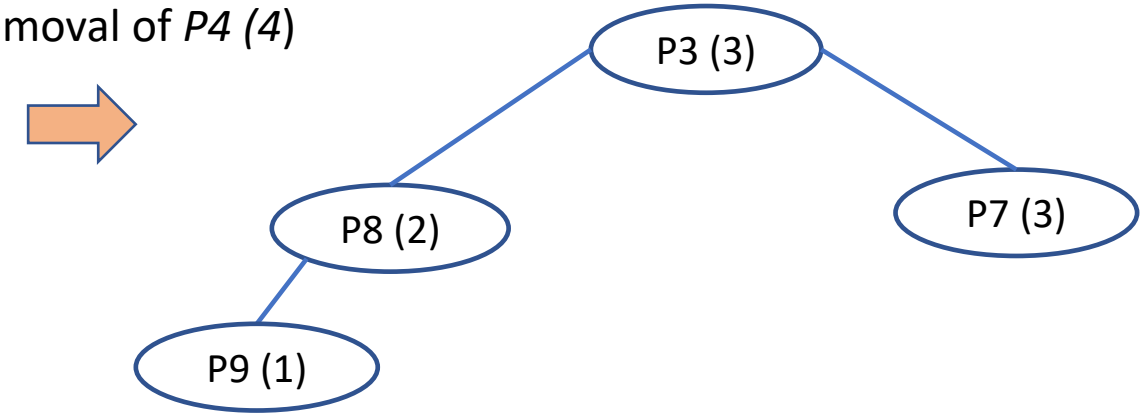


Heap: Priority Queue Applications (6)

Process:	*P1*	*P2*	*P3*	*P4*	*P5*	*P6*	*P7*	*P8*	*P9*	*P10*
Priority:	5	4	3	4	5	5	3	2	1	5

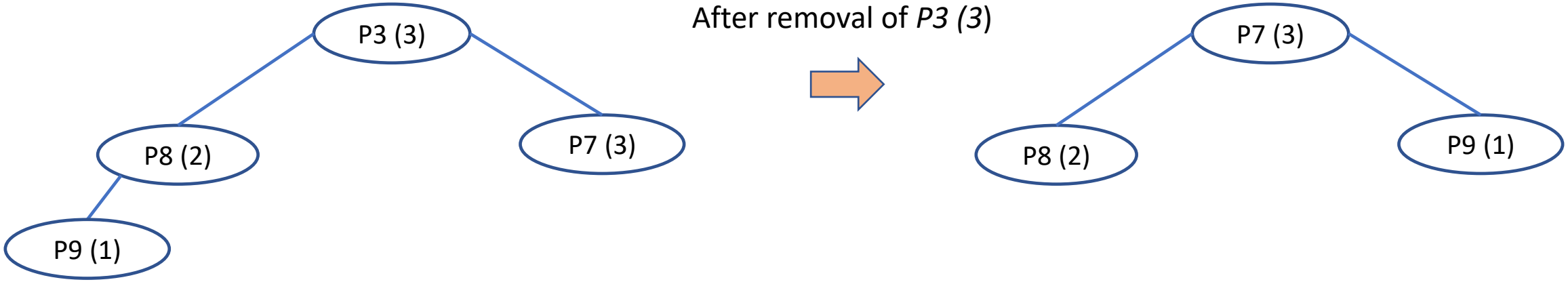


After removal of *P4 (4)*



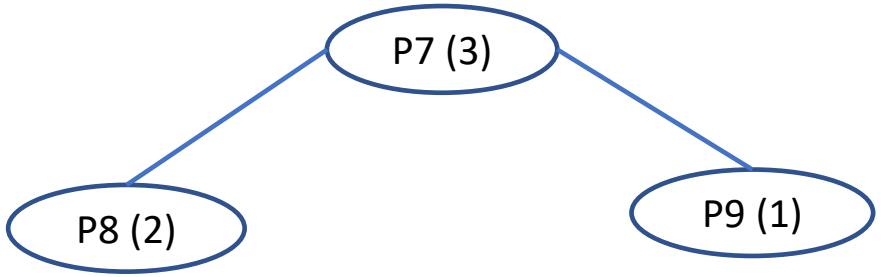
Heap: Priority Queue Applications (7)

Process:	*P1*	*P2*	*P3*	*P4*	*P5*	*P6*	*P7*	*P8*	*P9*	*P10*
Priority:	5	4	3	4	5	5	3	2	1	5

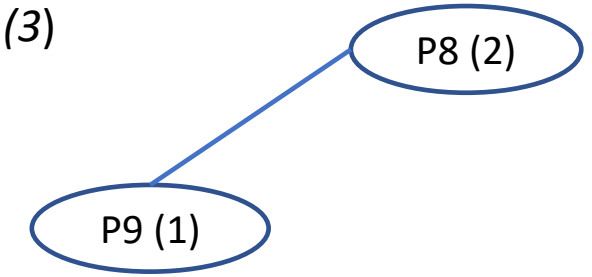


Heap: Priority Queue Applications (7)

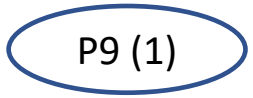
Process:	*P1*	*P2*	*P3*	*P4*	*P5*	*P6*	*P7*	*P8*	*P9*	*P10*
Priority:	5	4	3	4	5	5	3	2	1	5



After removal of *P7 (3)*



After removal of *P8 (2)*



Finally *P9 (1)* is served

Heap: Insertion

Which of the following sequences of array elements forms a heap?

A. {23, 17, 14, 6, 13, 10, 1, 12, 7, 5}
B. {23, 17, 14, 6, 13, 10, 1, 5, 7, 12}
C. {23, 17, 14, 7, 13, 10, 1, 5, 6, 12}
D. {23, 17, 14, 7, 13, 10, 1, 12, 5, 7}

thank you!

email:
k.kondepu@iitdh.ac.in