Data Structures and Algorithms Design (ZG519)

BITS Pilani

SESSION 4 -PLAN

Contact Sessions(#)	List of Topic Title	Text/Ref Book/external resource
4	Heaps - Definition and Properties, Representations (Vector Based and Linked), Insertion and deletion of elements	T1: 2.4

The Heap Data Structure

- The heap is a binary tree T that stores a collection of keys at its nodes and that satisfies two additional properties: a relational property defined in terms of the way keys are stored in T and a structural property defined in terms of T itself.
- We assume that a **total order relation** on the keys is given, for example, by a comparator.

Total order relation

- Comparison rule is defined for every pair of keys and it must satisfy the following properties:

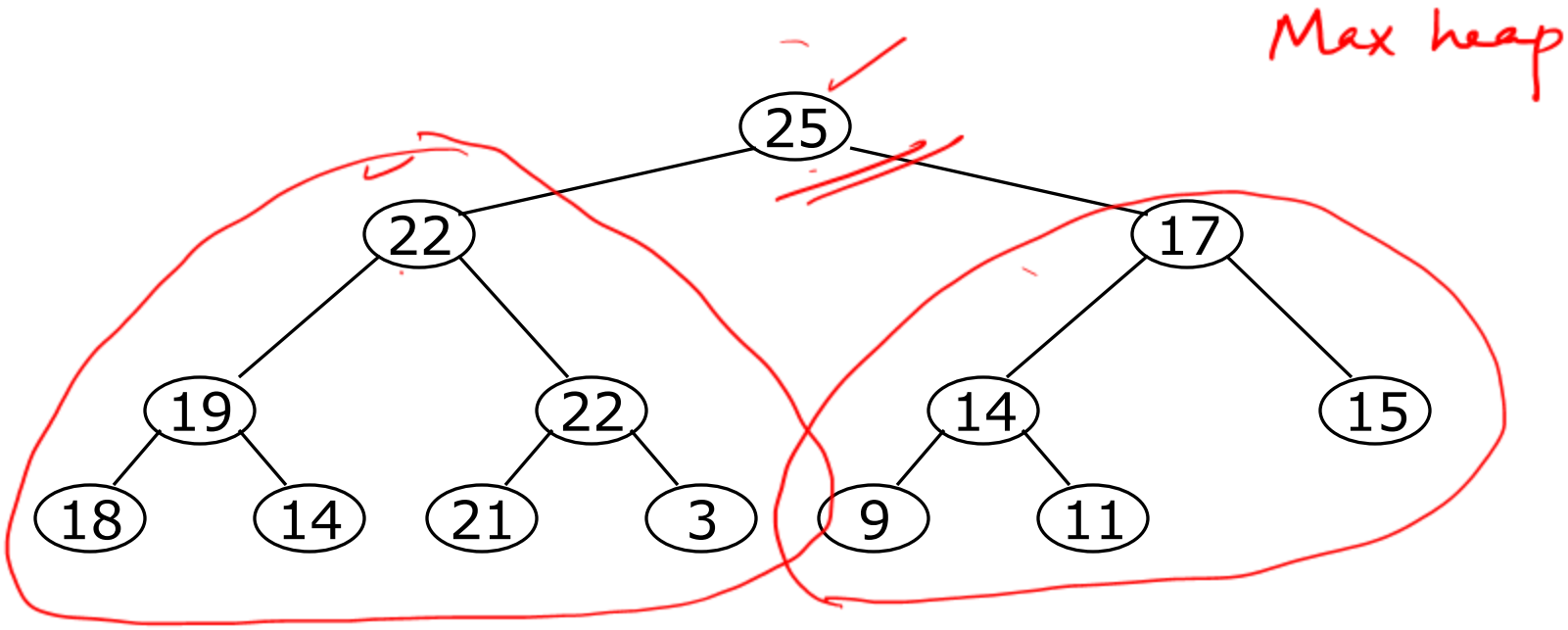
  - **Reflexive property**: $k \leq k$.
  - **Antisymmetric property**: if $k_1 \leq k_2$ and $k_2 \leq k_1$, then $k_1 = k_2$.
  - **Transitive property**: if $k_1 \leq k_2$ and $k_2 \leq k_3$, then $k_1 \leq k_3$.

- Any comparison rule <= that satisfies these three properties will never lead to a comparison contradiction.

The Heap Data Structure:

- A heap is a binary tree storing elements at its nodes and satisfying the following properties:
  - **Heap-Order property (Relational property):**
    - for every node v other than the root, ***key***(***v***) ≥ ***key***(***parent***(***v***)) –Min heap
    - ***key***(***v***)<= ***key***(***parent***(***v***))-Max heap
- The keys encountered on a path from the root to an external node of T are in non decreasing /non increasing order.
- A minimum/max key is always stored at the root of T.

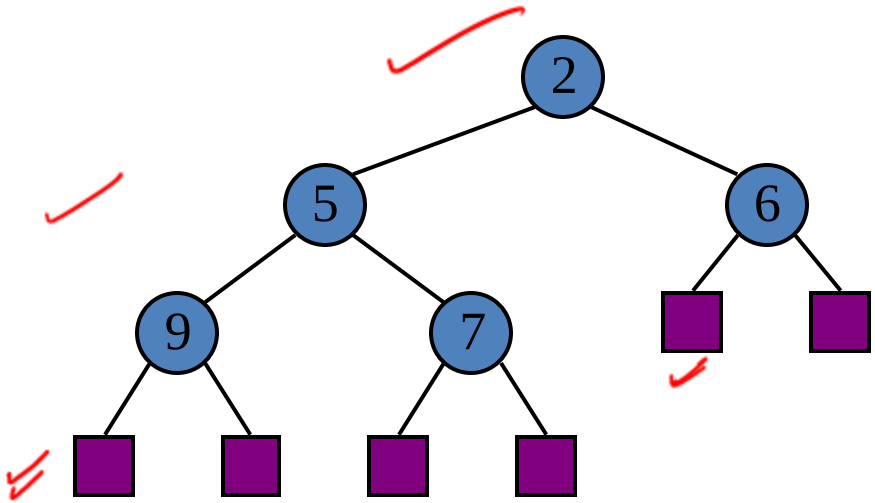
The Heap Data Structure

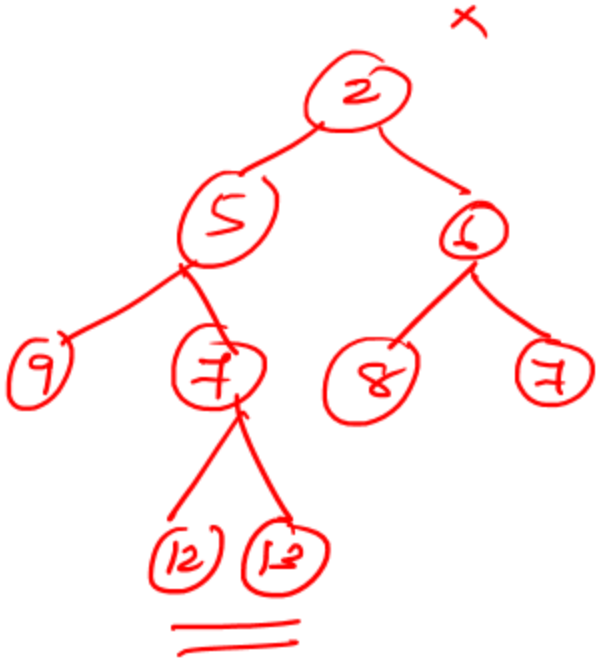


The Heap Data Structure

Structural property:

- Heap data structure is always a complete binary tree

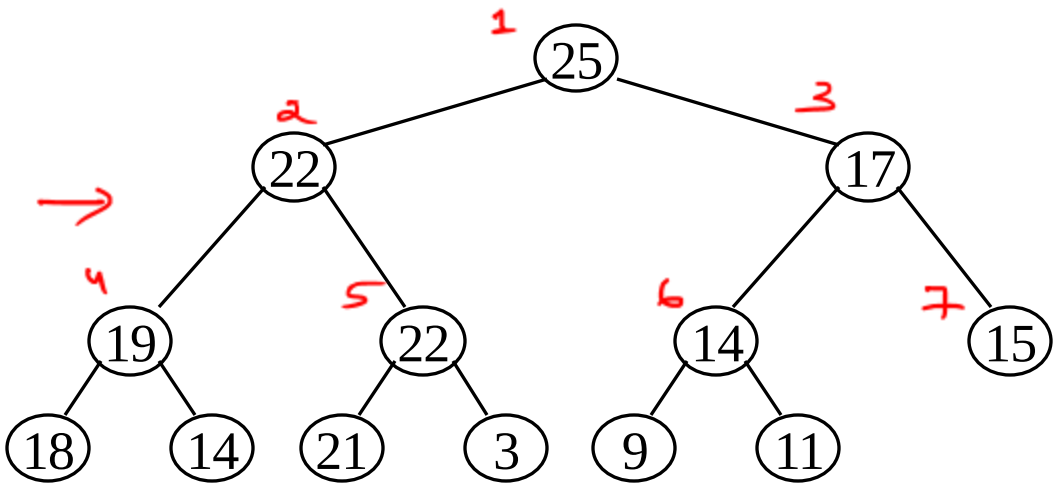




Vector-based Heap Implementation

- We can represent a heap with n keys by means of a vector of length $n + 1$
- For the node at rank i
  - the left child is at rank $2i$
  - the right child is at rank $2i + 1$
- Links between nodes are not explicitly stored
- In other words the parent of a node stored in i^{th} location is at floor$[i/2]$
- The cell at rank 0 is not used

Vector-based Heap Implementation

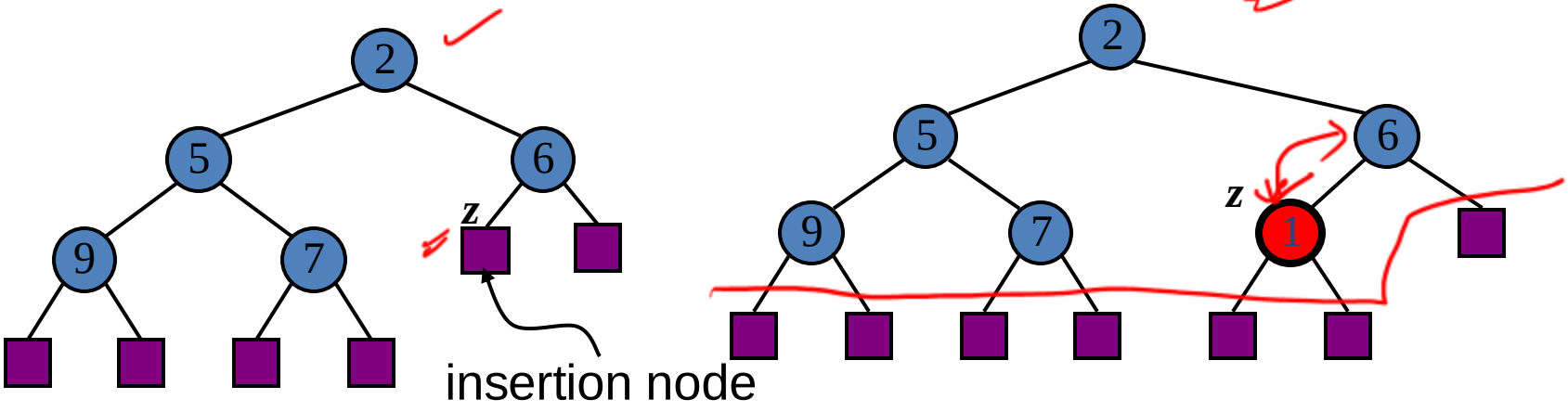


0	1	2	3	4	5	6	7	8	9	10	11	12	13
	25	22	17	19	22	14	15	18	14	21	3	9	11

- the left child is at rank 2i
- the right child is at rank 2i + 1
  - Example: the children of node 3 (17) are 6 (14) and 7 (15)
  - Note that when the heap T is implemented with a vector, the index of the last node is always equal to n

Insertion into a Heap

- Method insertItem corresponds to the insertion of a key k to the heap
- The insertion algorithm consists of three steps
  - Find the insertion node z (the new last node)
  - Store k at z and expand z into an internal node
  - Restore the heap-order property (discussed next)

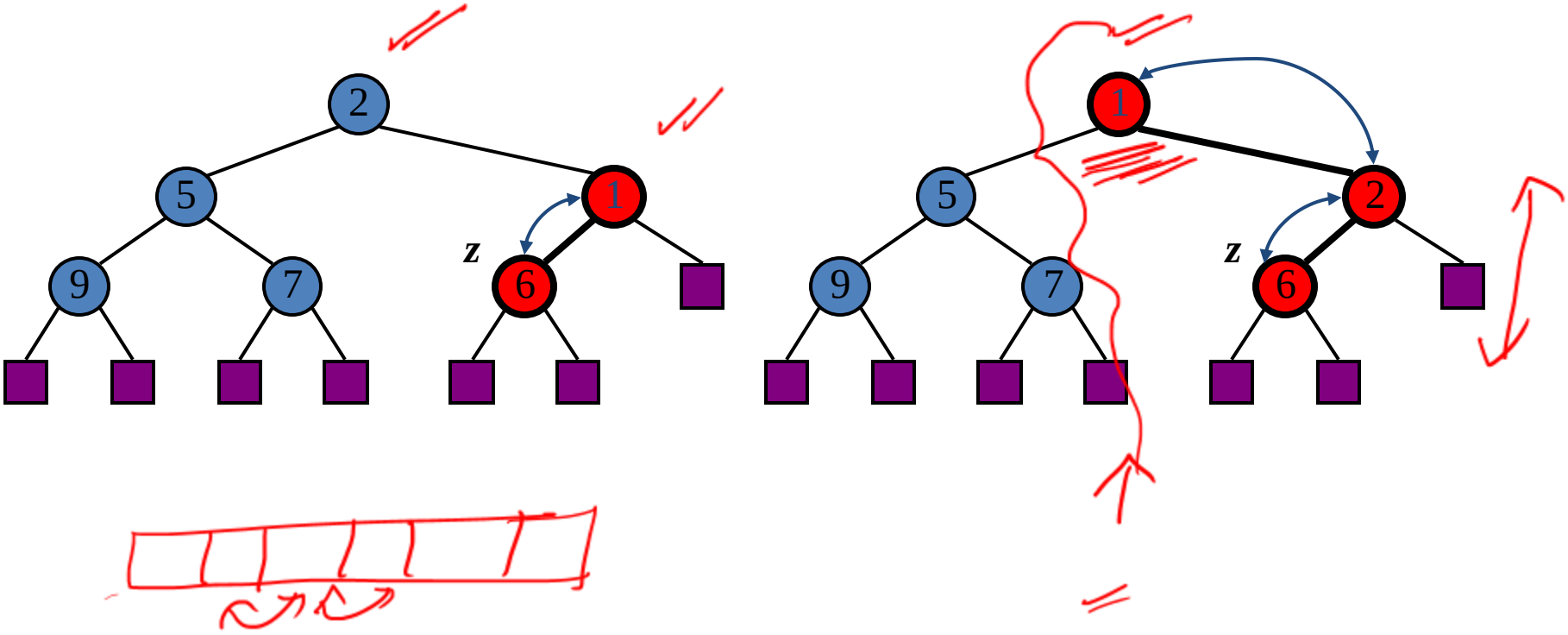


  - External nodes are represented a

Upheap

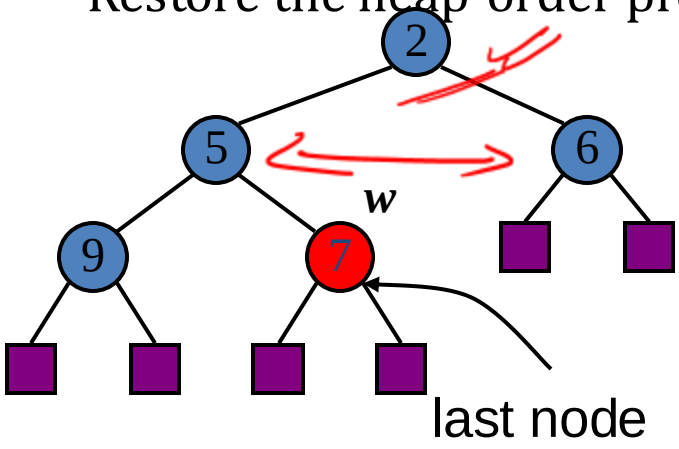
- After the insertion of a new key k, the heap-order property may be violated
- Algorithm upheap restores the heap-order property by swapping k along an upward path from the insertion node
- Upheap terminates when the key k reaches the root or a node whose parent has a key smaller than or equal to k
- Since a heap has height $O(\log n)$, upheap runs in $O(\log n)$ time

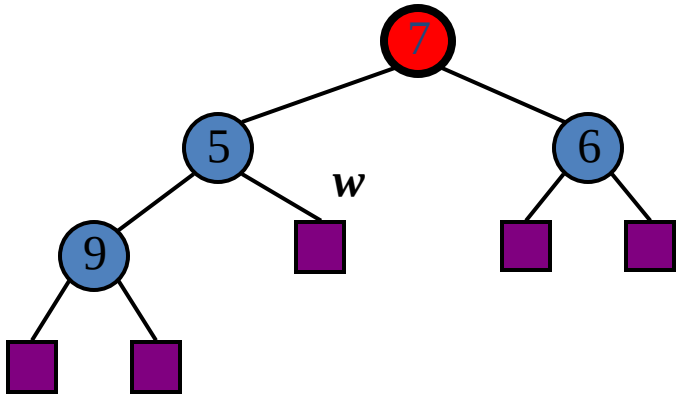
Upheap



Removal from a Heap

- Method removeMin corresponds to the removal of the root key from the (min) heap
- The removal algorithm consists of three steps
  - Replace the root key with the key of the last node ***w***
  - Compress ***w*** and its children into a leaf
  - Restore the heap-order property (discussed next)

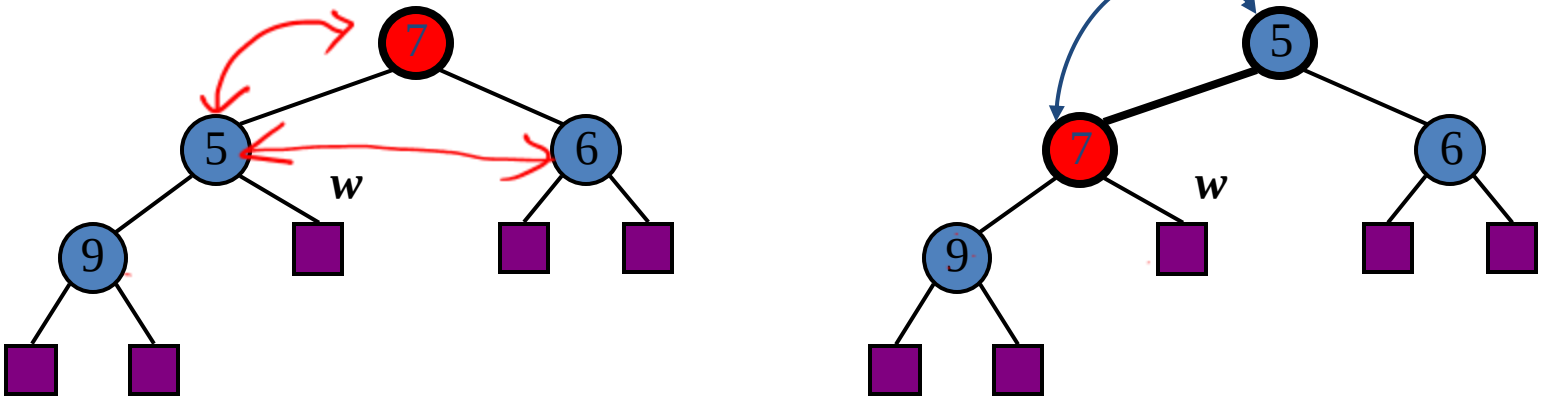




Downheap

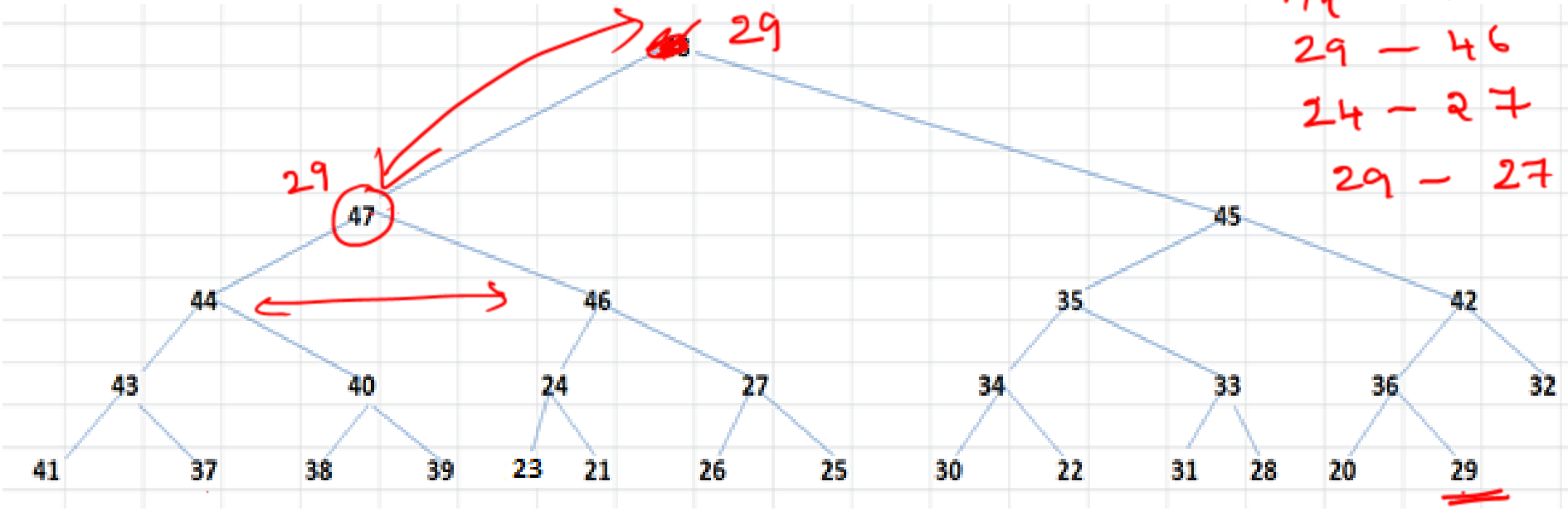
- After replacing the root key with the key k of the last node, the heap-order property may be violated
- Algorithm downheap restores the heap-order property by swapping key k along a downward path from the root
- Downheap terminates when key k reaches a leaf or a node whose children have keys greater than or equal to k
- Since a heap has height $O(\log n)$, downheap runs in $O(\log n)$ time

Downheap



QN-1

- Consider the following binary heap.



45–47
29–47
46–46
29–46
24–27
29–27

29

29, 36, 42, 45

48

QN-1 Contd.

- Suppose that the last operation performed in the binary heap above was inserting the key x. Find all possible values of x. Give suitable explanations

- Suppose that you delete the maximum key from the binary heap above. Find all keys that are involved in one (or more) comparisons. Show the comparisons.

Solution:

- 29 36 42 45 .To insert a node in a binary heap, we place it in the next available leaf node and swim it up. Thus, 29 36 42 45, and 48 are the only keys that might move. But, the last inserted key could not have been 48, because, then, 45 would have been the old root (which would violate heap order because the left child of the root is 47).
- 24 ,27,29,44,45,46,47 The compares are 45-47,29-47, 44-46, 29-46 ,24-27, 29-27

QN-2

- Write an algorithm to find the **kth largest** element of an array using a
  - **Min** Heap.
  - **Max** Heap
- What is the time complexity in both cases if the array is **unsorted**?

QN-2-Solution

- **For Min Heap**: Create a min heap of size k by taking k elements of the array. Now we compare the root with the remaining elements of the array. If the element is greater than the root, then replace root with this element and min_heapify .Iterate this for all elements of the array. The root of the heap will be the kth largest element.

- Time Complexity: O(k + (n-k)Logk)

QN-3

- **For Max Heap**: Create a max heap of size n by taking all elements of the array. Now remove the root k-1 times and re-heapify. The root of the heap now will be the kth largest element.

- Time Complexity: for building a Max Heap – O(n), for deleting the root k-1 times – O((k-1)log n), so total is O(n + (k-1)logn)