힙 구조

강지훈
jhkang@cnu.ac.kr
충남대학교 컴퓨터공학과

Double-ended Priority Queues

Double-Ended Priority Queue

- It supports the following operations:
  1. Insert an element with an arbitrary key.
  2. Delete an element with the largest key.
  3. Delete an element with the smallest key.

Class "DoubleEndedPriorityQ"

Public Functions

- public DoubleEndedPriorityQ () {...}
- public DoubleEndedPriorityQ (int givenMaxSize) {...}
- public boolean isEmpty () {...}
- public boolean isFull () {..}
- public int size () {...}
- public boolean add (Element anElement) {...}
- public int min () {...}
- public int max () {...}

Implementation

- Using Min-Max Heaps or Deaps

- Private Instance Variables

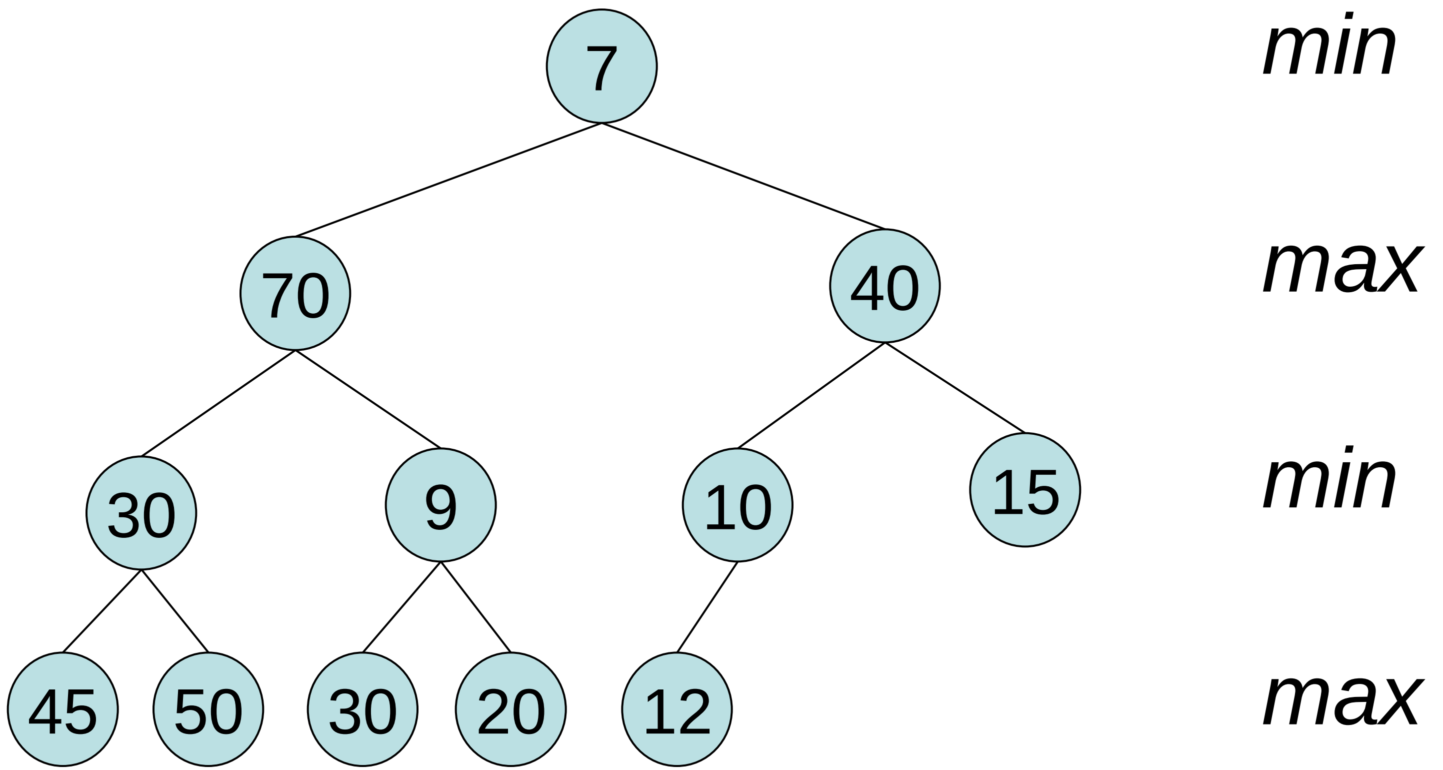
```
public class DoubleEndedPriorityQ {
    private int     _maxSize ;
    private int     _size ;
    private int[]   _heap ;
    ……
}
```

Min-Max Heaps

Min-Max Heaps

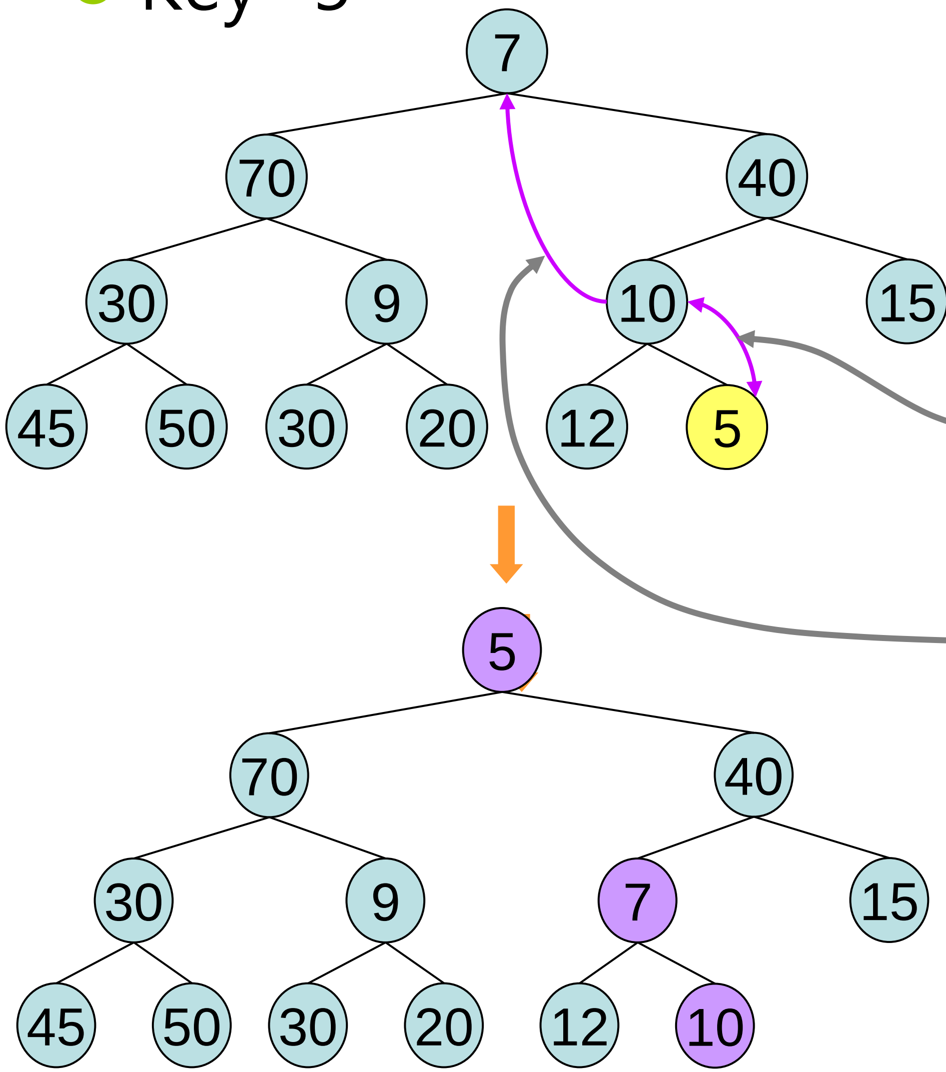
- A complete binary tree such that if it is not empty, each element has a field, called *key*.
- Alternating levels of this tree are min levels and max levels, respectively.
- The root is on a min level.
- Let x be a node in a min-max heap.
  - If x is on a min level, then the element in x has the minimum key from among all elements in the subtree with root x.
    - We call this node a min node.
  - Similarly, if x is on a max level, then the element in x has the maximum key from among all elements in the subtree with root x.
    - We call this node a max node.

An Example of Min-Max Heap



Insertion

- Key=5

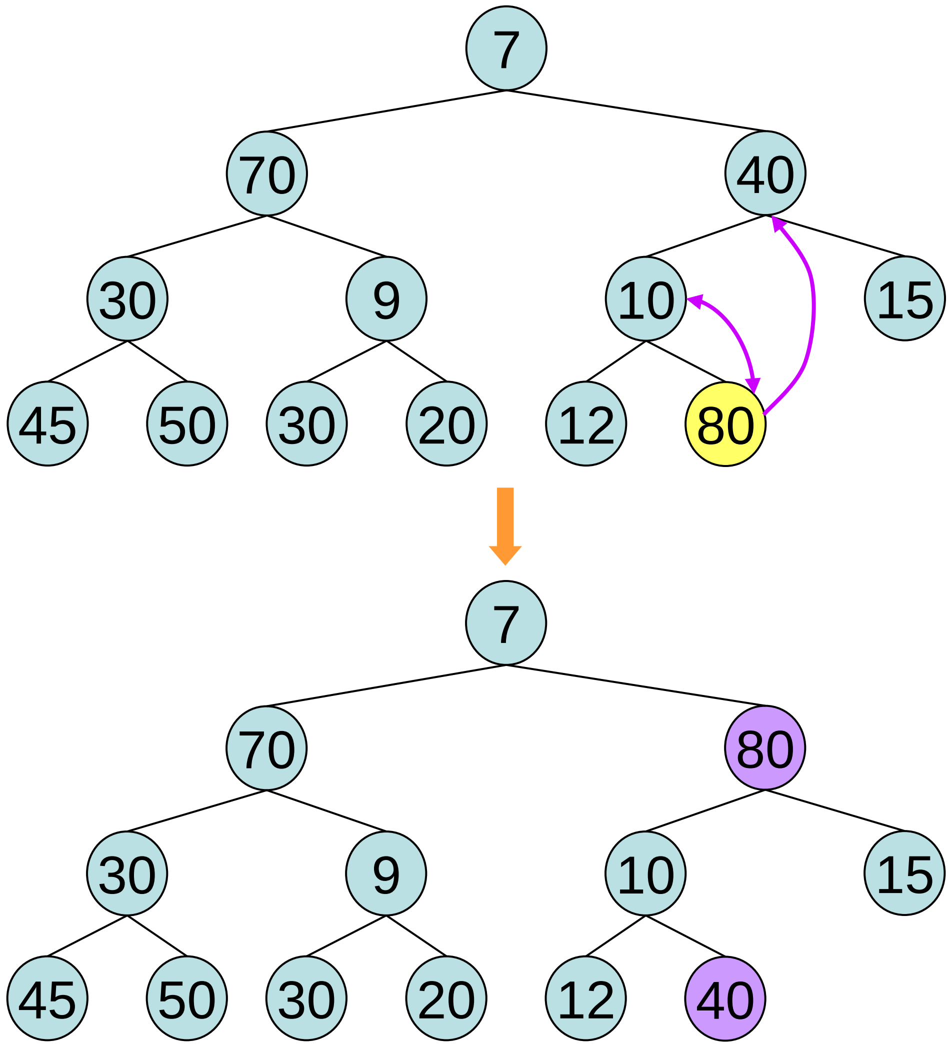


- Determine whether the inserted node will be located on the min level or on the max level by comparing its parent node.
- In this case, '5' will be put to the min level.

- Then, '5' is exchanged with '10'.
- '5' will go up only through the min levels.

Insertion

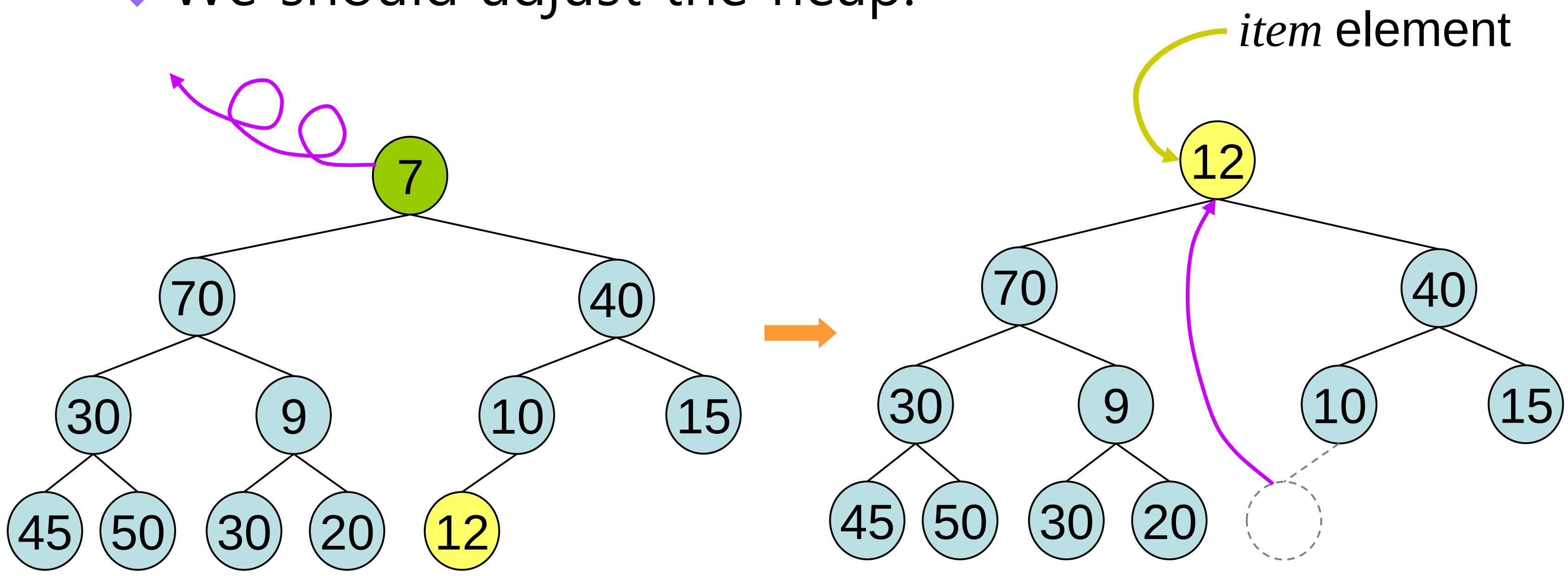
- Key=80



Analysis of Insert : $O(\log n)$

Deletion of MIN

- The root has the smallest key.
  - So, the root is deleted as the min element.
- The last element of the heap is deleted and reinserted into the root.
  - We should adjust the heap.



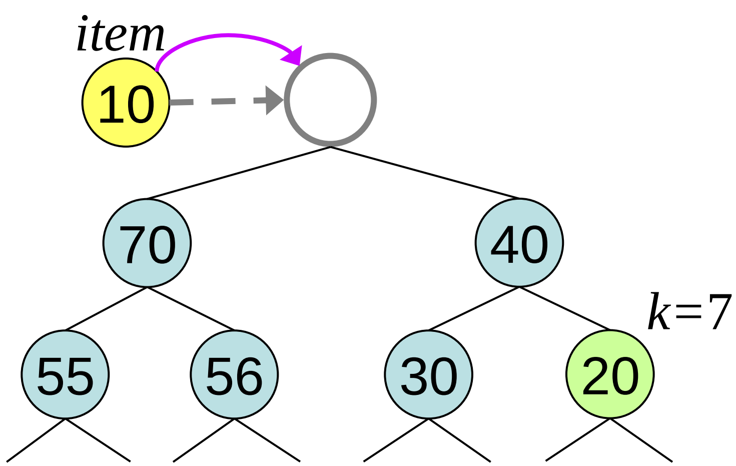
Adjusting the heap after deleting MIN.

- The root has no children.
  - The tree becomes empty after deletion.
- The root has at least one child.
  - The smallest key is in one of the children or grand-children of the root. Let this be node k.

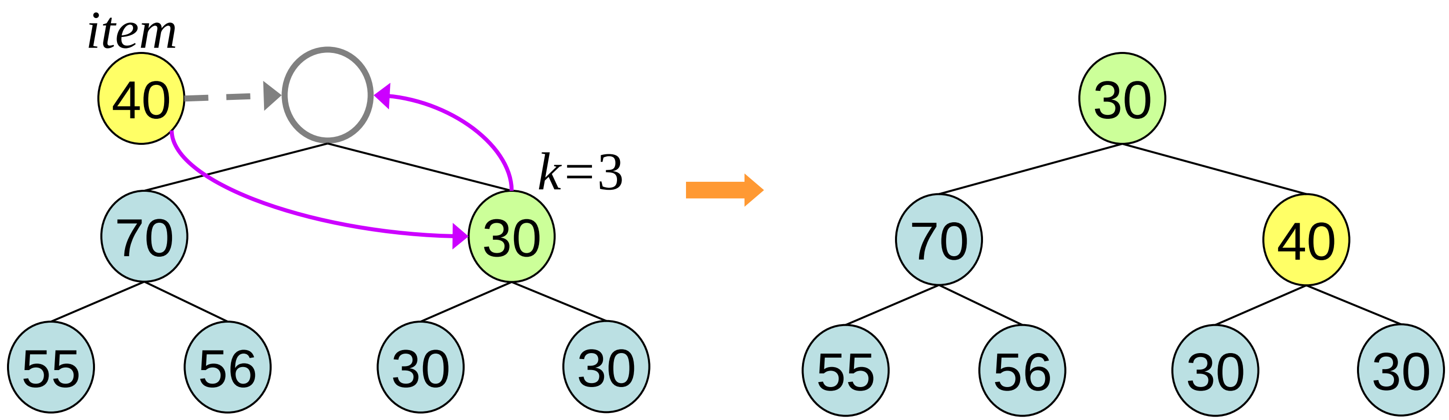
(a) $item.key \le heap[k].key$

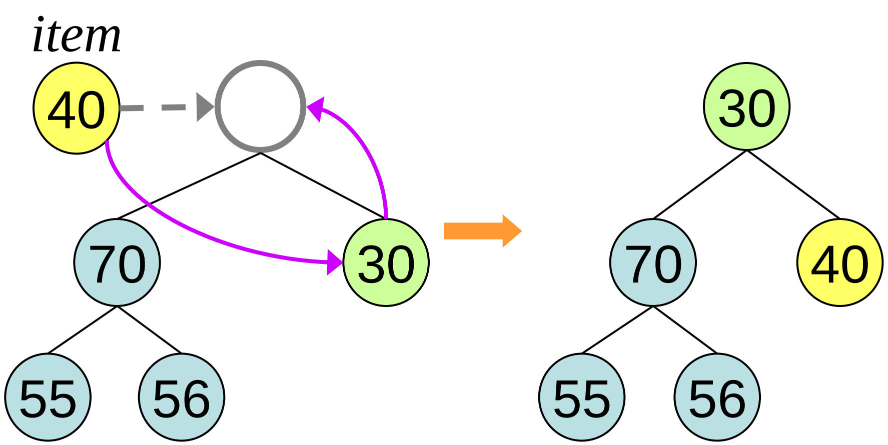
In this case, there is no element in the heap with key smaller than $item.key$.

So, $item$ may be inserted into the root.



(b) *item.key* > *heap*[*k*].*key* and node *k* is a child of the root.
Then *k* is a max node. Hence, node *k* has no descendants with key larger than *heap*[*k*].*key*.
So, the element *heap*[*k*] may be moved to the root and item inserted into node *k*.





Note: Actually if node *k* has descendents, then they have the same key value as the *heap*[*k*].*key*.

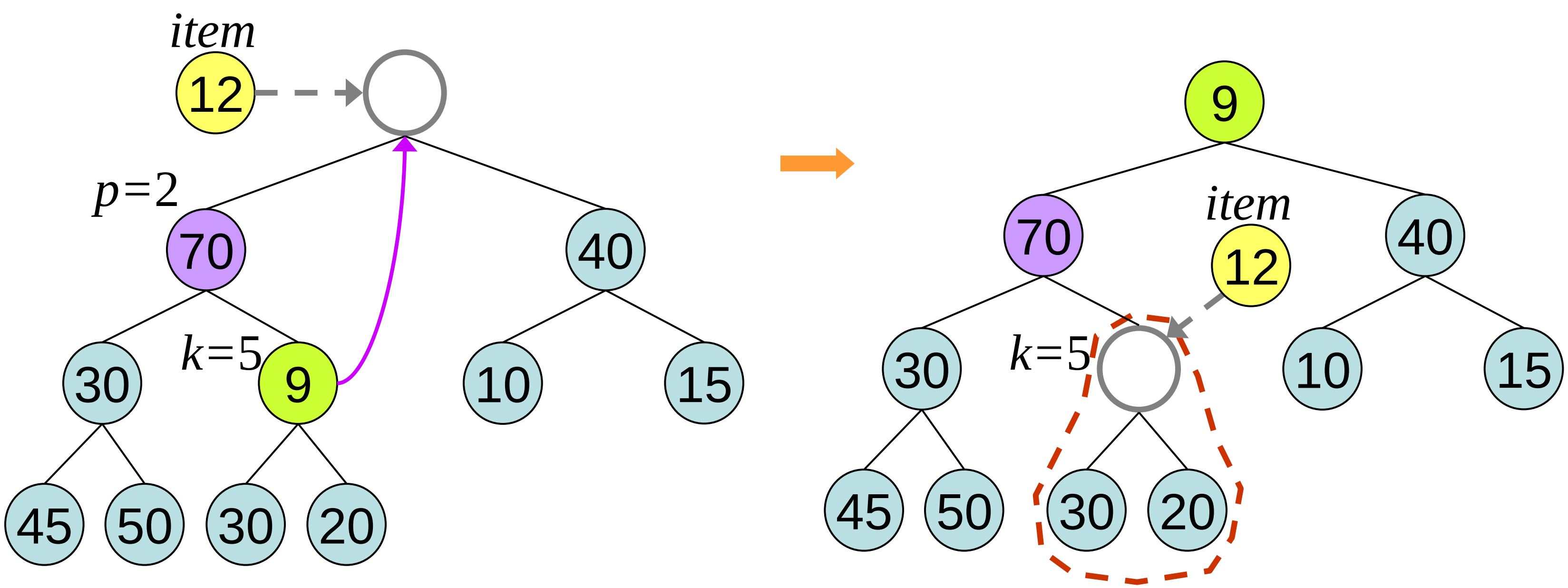
(c) $item.key > heap[k].key$ and k is a grandchild of the root.

Let p be the parent of k. (i.e., $p = \lfloor k/2 \rfloor$)

$heap[k]$ may be moved to the root.

① $item.key \leq heap[p].key$

Repeat the above adjusting process for the sub min-max heap with root k.

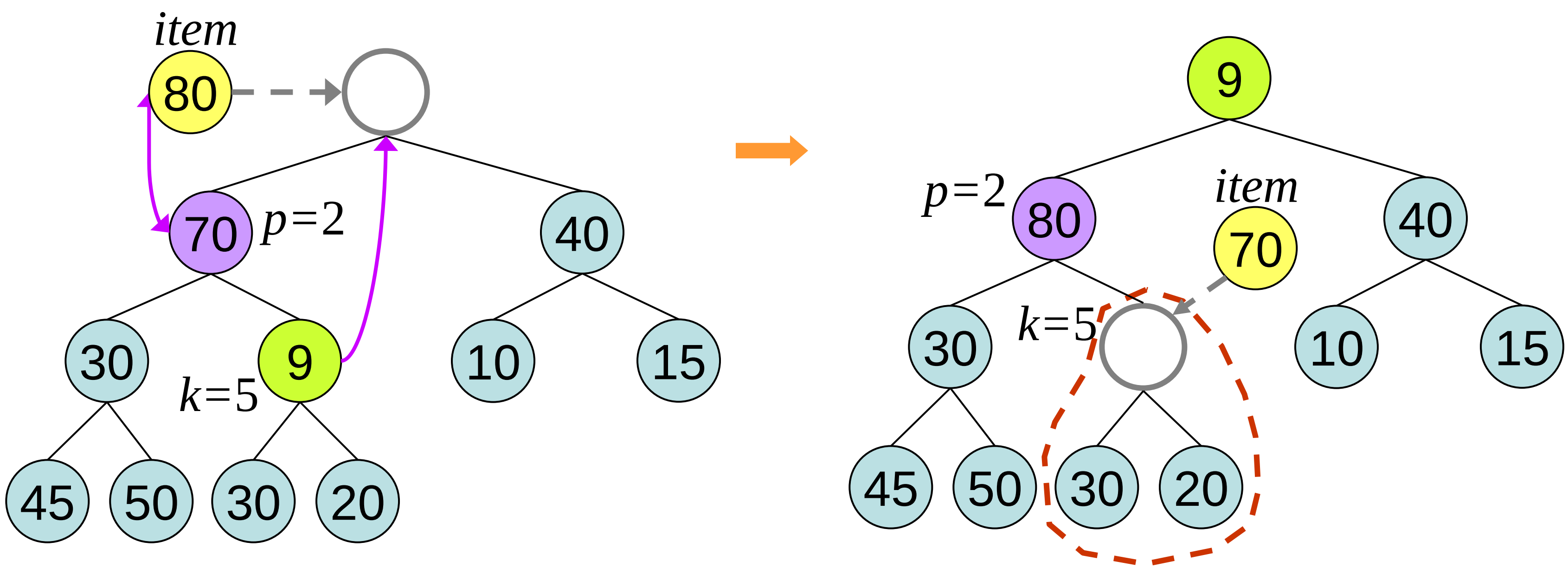


② $item.key > heap[p].key$

item and *heap*[*p*] are interchanged.

The max node *p* contains the largest key in the sub-heap with the root *p*.

Repeat the above adjusting process for the sub min-max heap with root *k*.



- Analysis of Delete MIN : O(log *n*).

Deaps

Deaps

- Double-ended heap
  - supports the double ended priority queue operations.
    - insert
    - delete min
    - delete max

- $O(\log n)$ for each operation
  - n is the size of a deap.

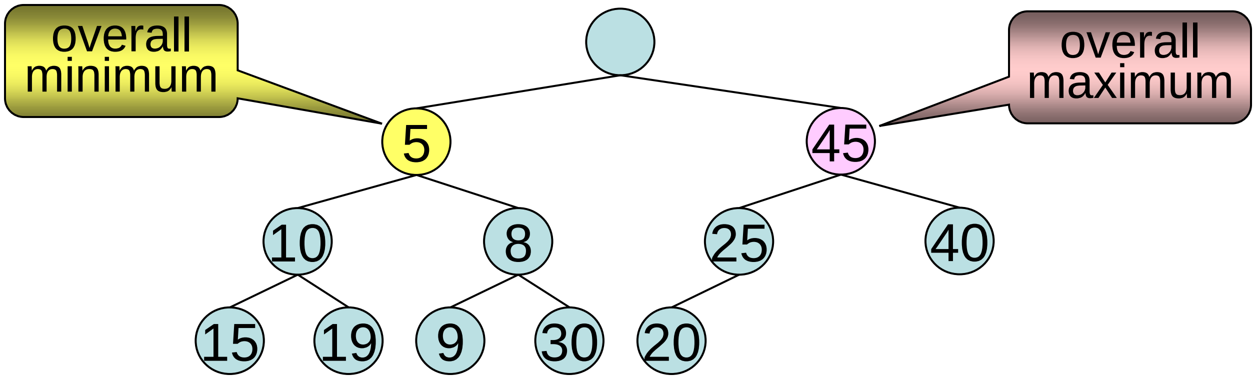
Deaps

- A deap is a complete binary tree that is either empty or satisfies the following properties:
  1. The root contains no element.
  2. The left subtree is a min-heap.
  3. The right subtree is a max-heap.
  4. If the right subtree is not empty, then let i be any node in the left subtree.

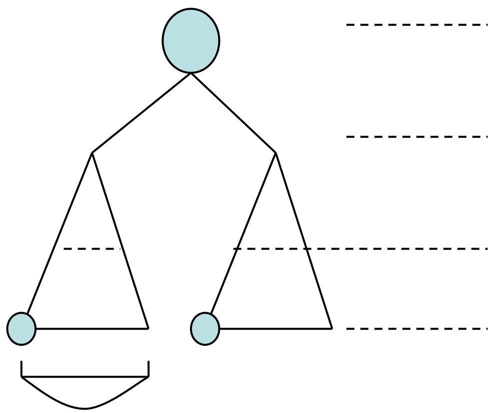
     Let j be the corresponding node in the right subtree.

     If such a j does not exist, then let j be the node in the right subtree that corresponds to the parent of i.

     The key in node i is less than or equal to the key in j.



How to compute the value of j:



2^{k-2} nodes

Level	leftmost node #	# of nodes at the level
1	2^0	2^0
2	2^1	2^1
k-1	2^{k-2}	2^{k-2}
k	2^{k-1}	2^{k-1}

$$j = i + 2^{k-2}$$

$$2^{k-1} \le i < 2^k$$

$$k - 1 \le \log_2 i < k$$

$$k - 1 = \lfloor \log_2 i \rfloor$$

$$k = \lfloor \log_2 i \rfloor + 1$$

$$j = i + 2^{(\lfloor \log_2 i \rfloor + 1) - 2}$$

$$= i + 2^{\lfloor \log_2 i \rfloor - 1}$$

Consequently,

$$j = i + 2^{\lfloor \log_2 i \rfloor - 1};$$

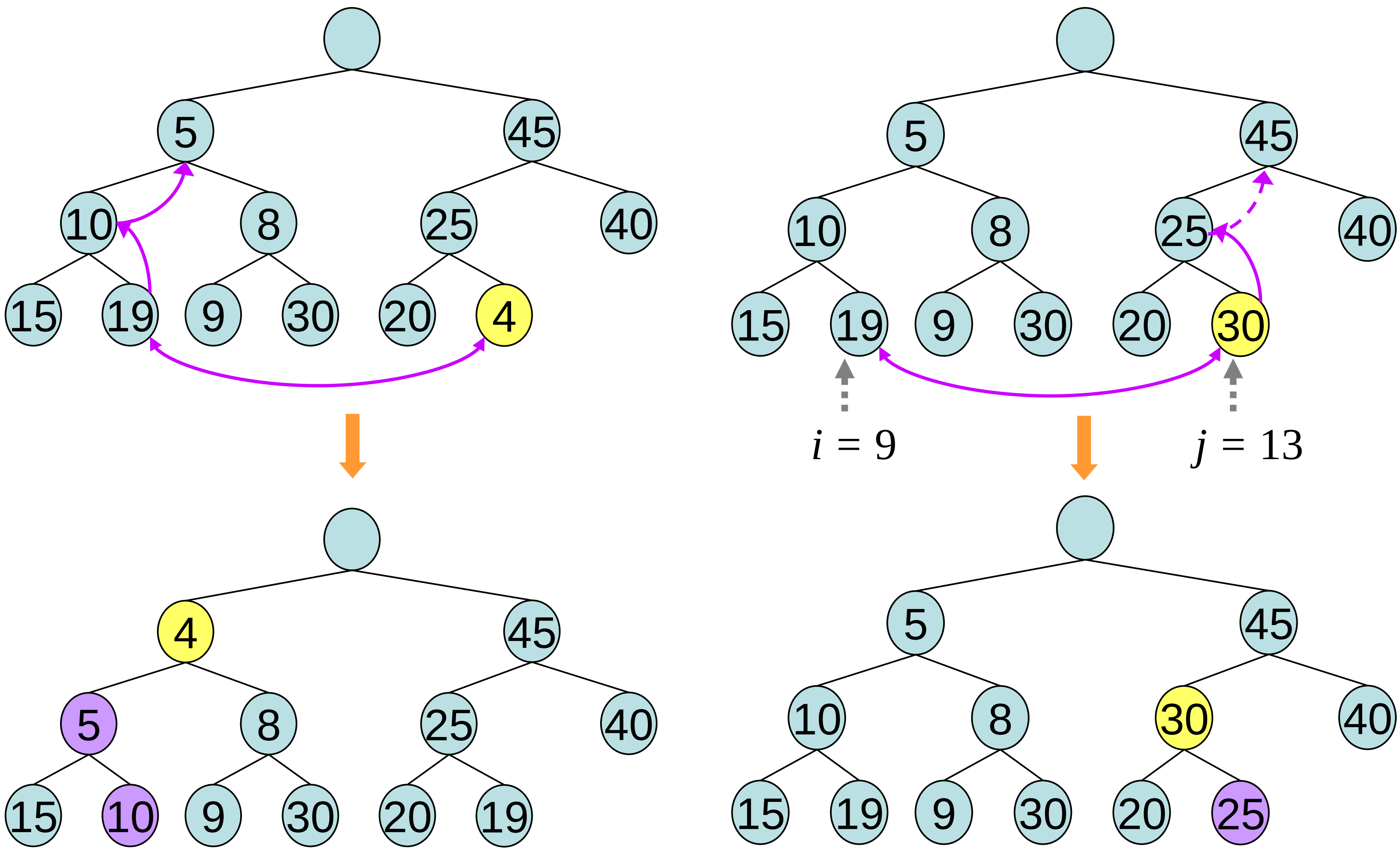
if ($j > n$) j /= 2;

Functions for Deap Insertion

- *max_heap(n)*
  - Returns TRUE iff n is a position in the max-heap of the deap.
- *min_partner(n)*
  - Computes the min-heap node that corresponds to the max-heap position n. The value is $(n-2^{\lfloor \log_2 n \rfloor -1})$.
- *max_partner(n)*
  - Computes the max-heap node that corresponds to the min-heap position n. The value is: if $n <= N$ then $(n+2^{\lfloor \log_2 n \rfloor -1})$, else $(n+2^{\lfloor \log_2 n \rfloor -1})/2$.
- *min_insert* and *max_insert*

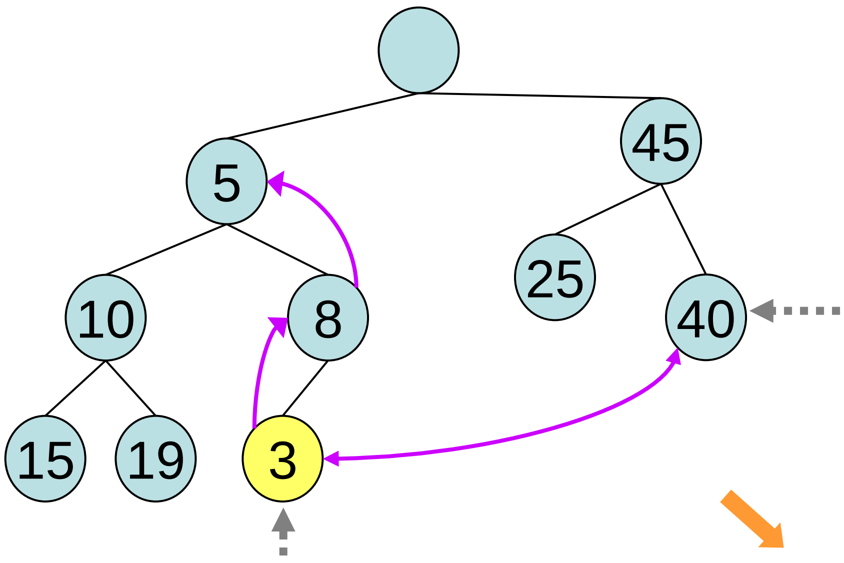
❑ Insertion to a Deap [1]

■ On a MAX side:



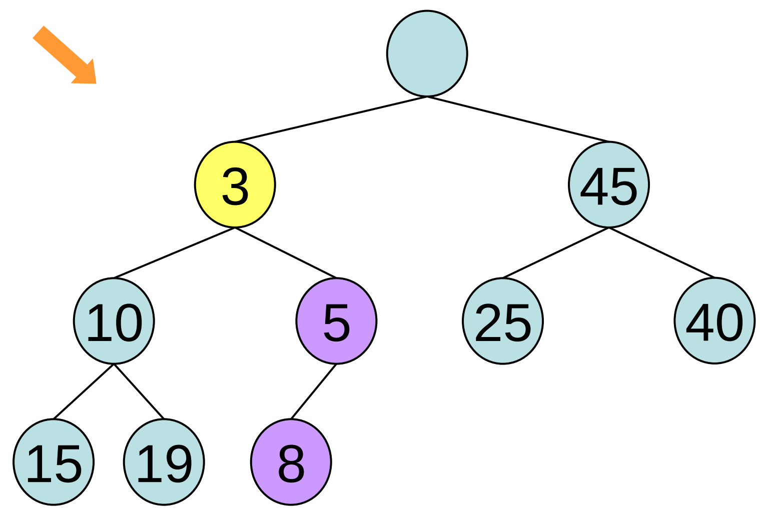
Insertion to a Deap [2]

On a MIN side:



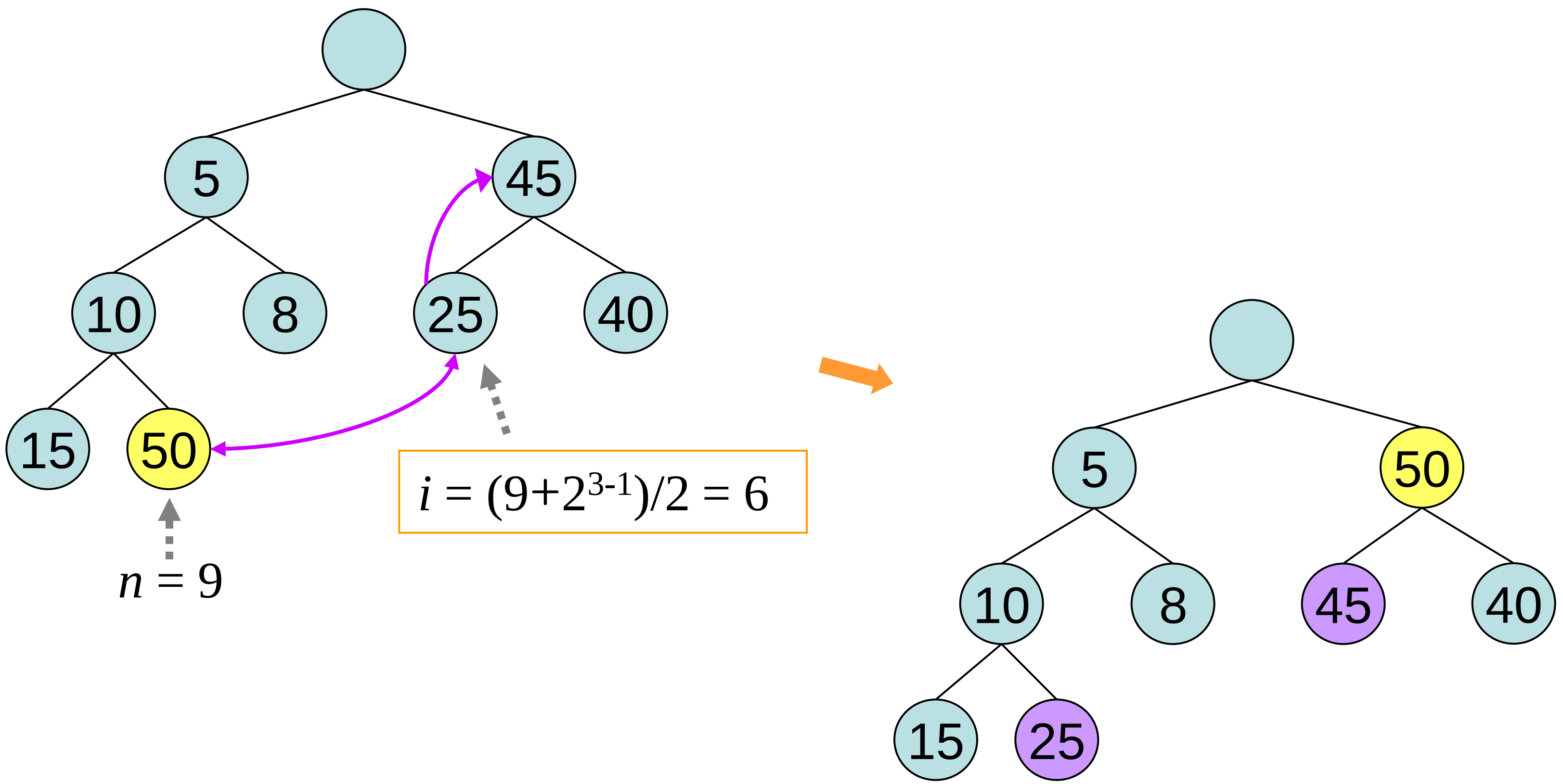
$n = 10$

$$i = (n + 2^{[\log_2 n] - 1}) / 2$$
$$= (10 + 2^{3-1}) / 2$$
$$= 7$$



Insertion to a Deap

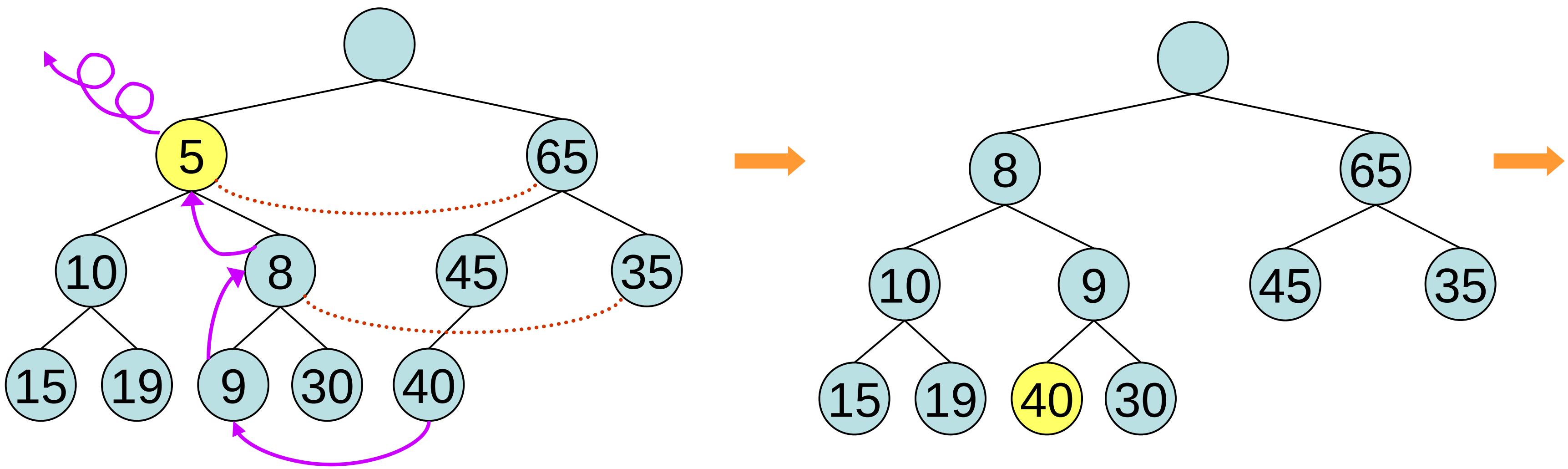
- On a MIN side:



- Analysis of Insert : O(log n)

Deletion of Min/Max [1]

Deletion of Min



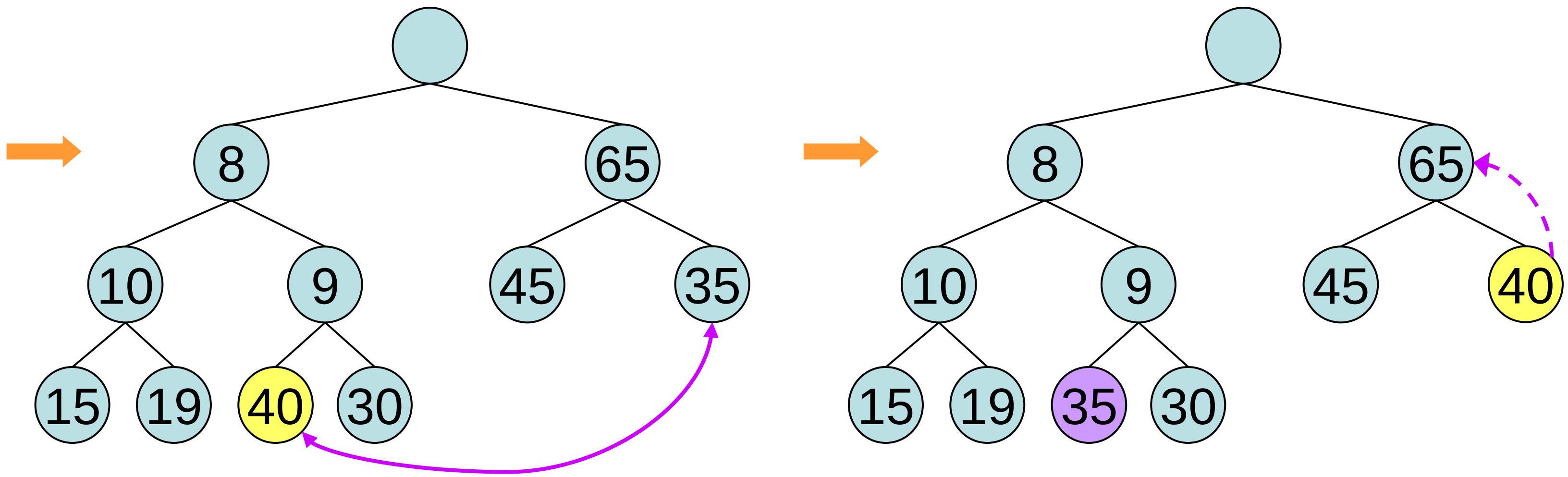
- Node 8 can be moved up since 8 ≤ 35 ≤ 65.
- Node 9 can be moved up since 9 ≤ 35.

At this time, we should insert 40 on the MIN side.

Deletion of Min/Max [2]

Deletion of Min (Cont'd)

- Modified Insertion



- Analysis of Delete : O(log *n*)

Delete of Max

- It is performed in a similar manner

End
of
Heap Structures