

파이썬으로 배우는 데이터 구조



한동대학교
전산전자공학부
김영섭 교수



학습 목표

힙(Heap)과 BST와의 차이점을 이해하고

힙(Heap)의 시간복잡도를 알 수 있다

Data Structures in Python

Chapter 8

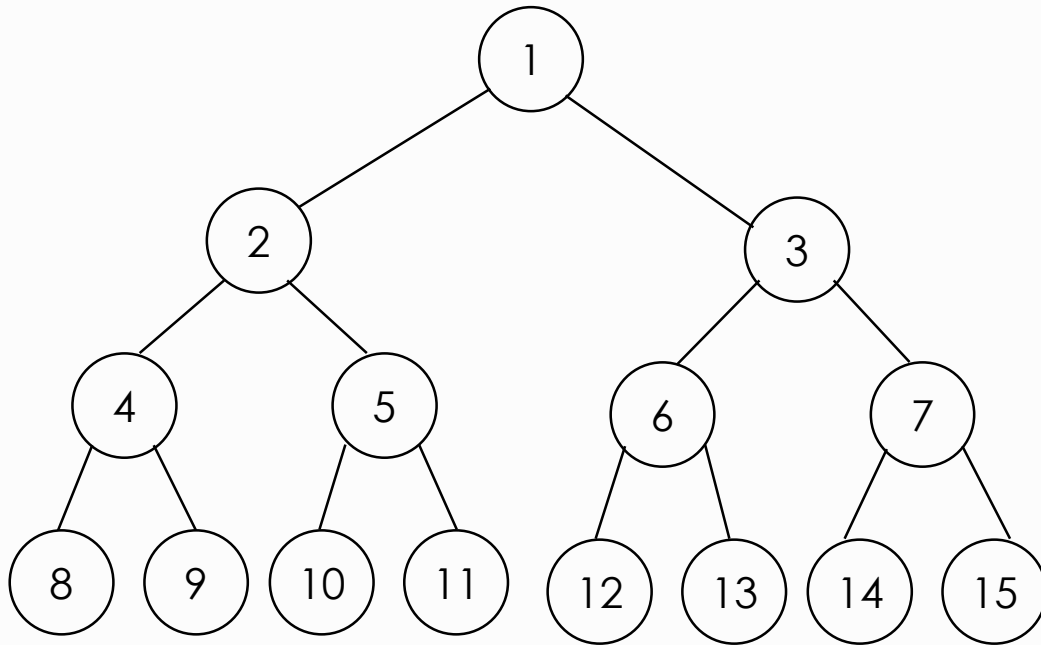
- **Heap and Priority Queue**
- **Heap Coding**
- **Min/MaxHeap and Heap sort**

Agenda & Readings

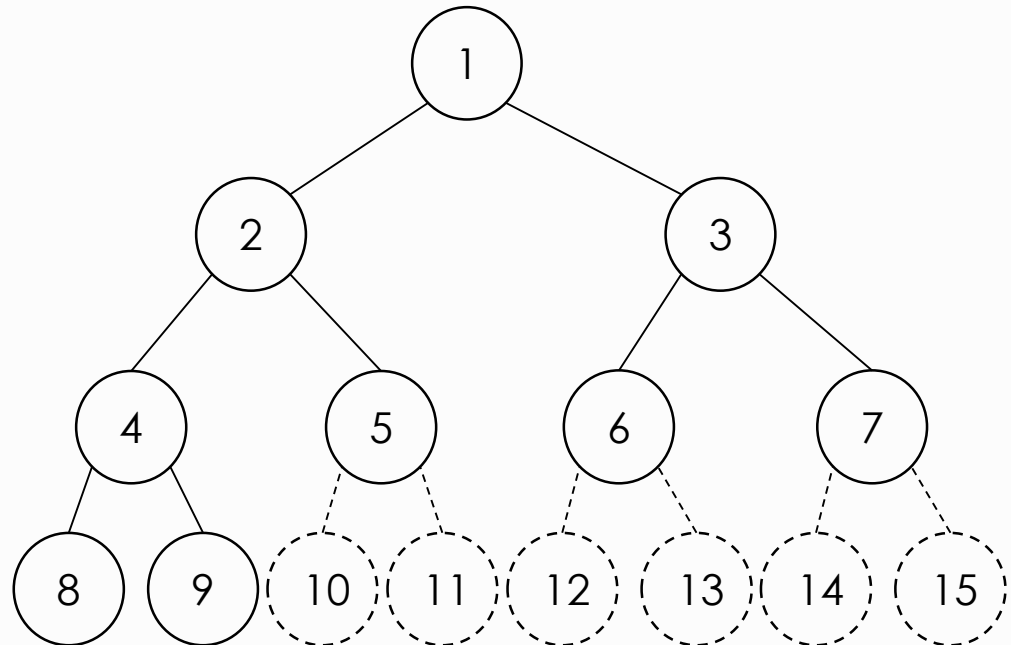
- Heap and Priority Queue
 - Complete Binary Tree (Review)
 - Heap and Priority Queue
 - Heap ADT
 - Time Complexity
- Reference:
 - Problem Solving with Algorithms and Data Structures

Binary trees - Properties

- **Definition:** A **full** binary tree of level k is a binary tree having $2^k - 1$ nodes, $k \geq 0$.
- **Definition:** A binary tree with n nodes and level k is **complete** if and only if its nodes correspond to the nodes numbered from 1 to n in the full binary tree of level k .



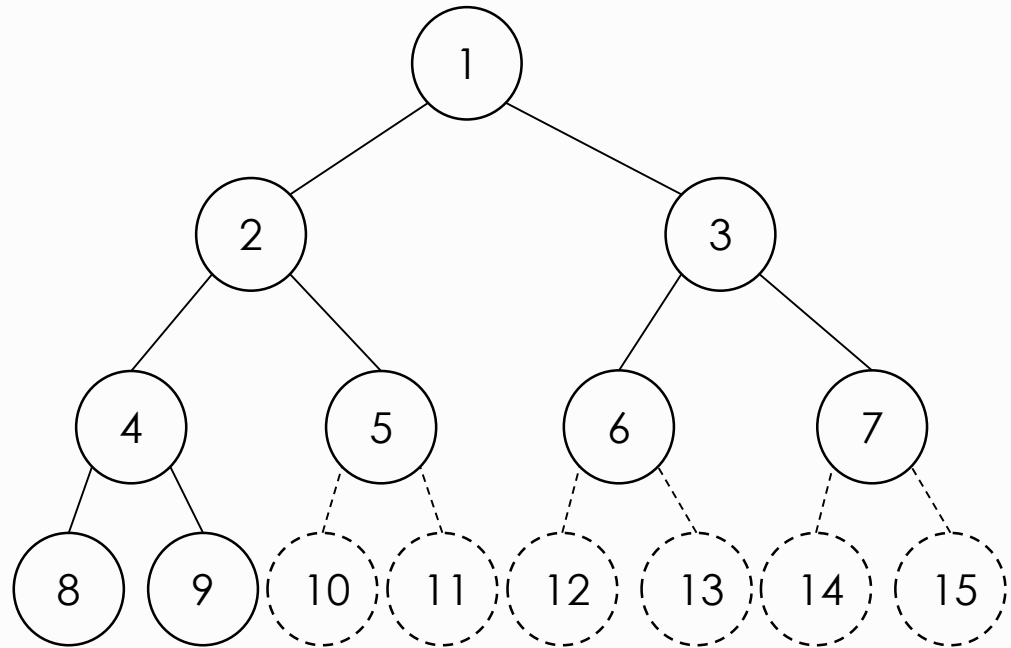
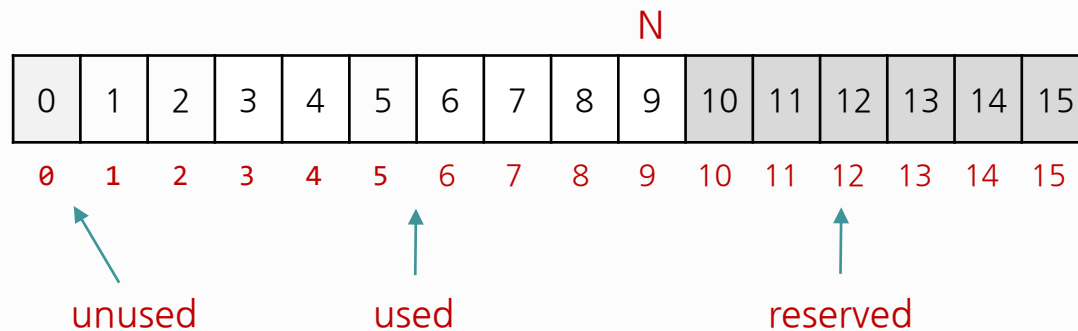
*A **full** binary tree*



*A **complete** binary tree*

Binary trees - Array representation

- A **complete** binary tree with n nodes, any node index i , $1 \leq i \leq n$, we have
 - $\text{parent}(i)$ is at $\lfloor i/2 \rfloor$ If $i = 1$, i is at the root and has no parent
 - $\text{leftChild}(i)$ is at $2i$ if $2i \leq n$. If $2i > n$, then i has no left child.
 - $\text{rightChild}(i)$ is at $2i+1$ if $2i+1 \leq n$. If $2i+1 > n$, then i has no right child.



*A **complete** binary tree*

Heaps & Priority Queues

- **Heaps** are frequently used to implement **priority queues**.
 - Because it provides an efficient implementation for **priority queues**.

Heaps & Priority Queues

- **Heaps** are frequently used to implement **priority queues**.
 - Because it provides an efficient implementation for **priority queues**.
- **Priority queues**.
 - Queues with priorities associated to.
 - **Example:** A line waiting to be served at a bank and served FIFO except if a senior or a disabled person arrives in the line. They are served first. Seniors and disabled persons have higher priority than others.

Heaps & Priority Queues

- **Heaps** are frequently used to implement **priority queues**.
 - Because it provides an efficient implementation for **priority queues**.
- **Priority queues**.
 - Queues with priorities associated to.
 - **Example:** A line waiting to be served at a bank and served FIFO except if a senior or a disabled person arrives in the line. They are served first. Seniors and disabled persons have higher priority than others.
- **A typical ADT for Priority Queue**
 - Get the top priority element (min or max)
 - Insert an element
 - Delete the top priority element
 - Decrease the priority of an element

- $O(1)$
- $O(\log n)$
- $O(\log n)$
- $O(\log n)$

Heaps & Priority Queues

- Challenge: Find the largest M items in a stream of N items.
- Constraints: Not enough memory to store N items.

N huge,
 M large

Order of insert of finding the largest M **in a stream of N items**

implementation	insert	delete	min/max
unordered array	1	N	N
ordered array	N	1	1
goal	$\log N$	$\log N$	$\log N$

N huge

Mission Impossible?

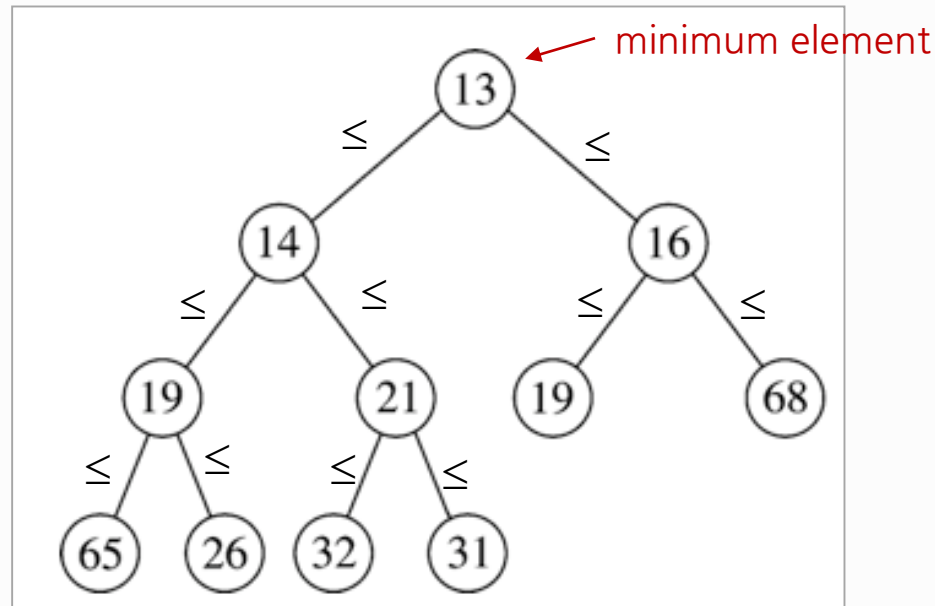
priority queue

Binary heap

- **Binary heap**: array representation of a **heap-ordered** complete binary tree
- **Properties:**
 - **Heap-ordered:**
Parent's key no smaller than children's keys. [max-heap]
 - **Heap-structure:**
A complete binary tree

Binary heap

- **Binary heap**: array representation of a **heap-ordered** complete binary tree
- **Properties:**
 - **Heap-ordered:**
Parent's key **no smaller** than children's keys for max-heap. (**no greater** for min-heap)
 - **Heap-structure:**
A complete binary tree

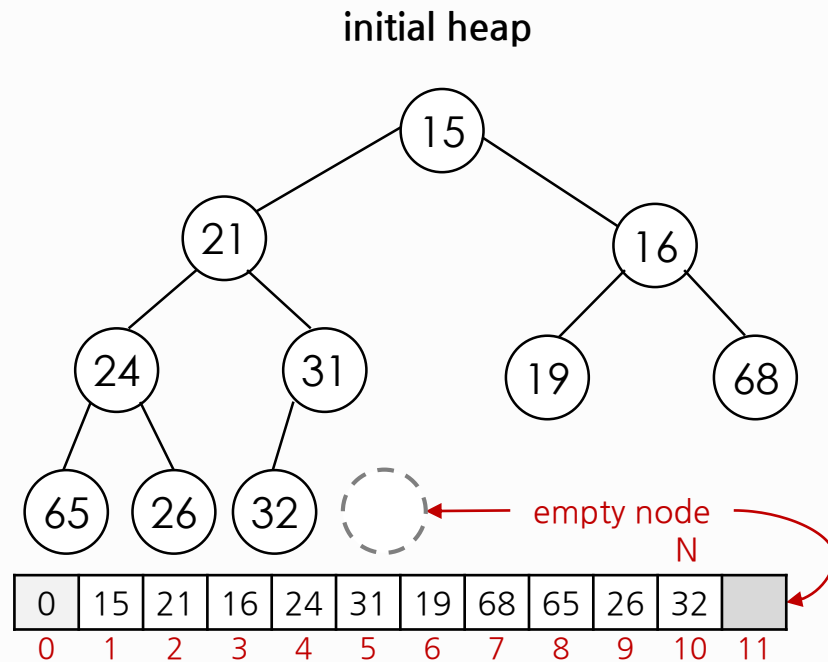


- Duplicates are allowed
- No order implied for elements which do not share ancestor-descendant relationship

min-heap: insert(heap, 14)

Algorithm:

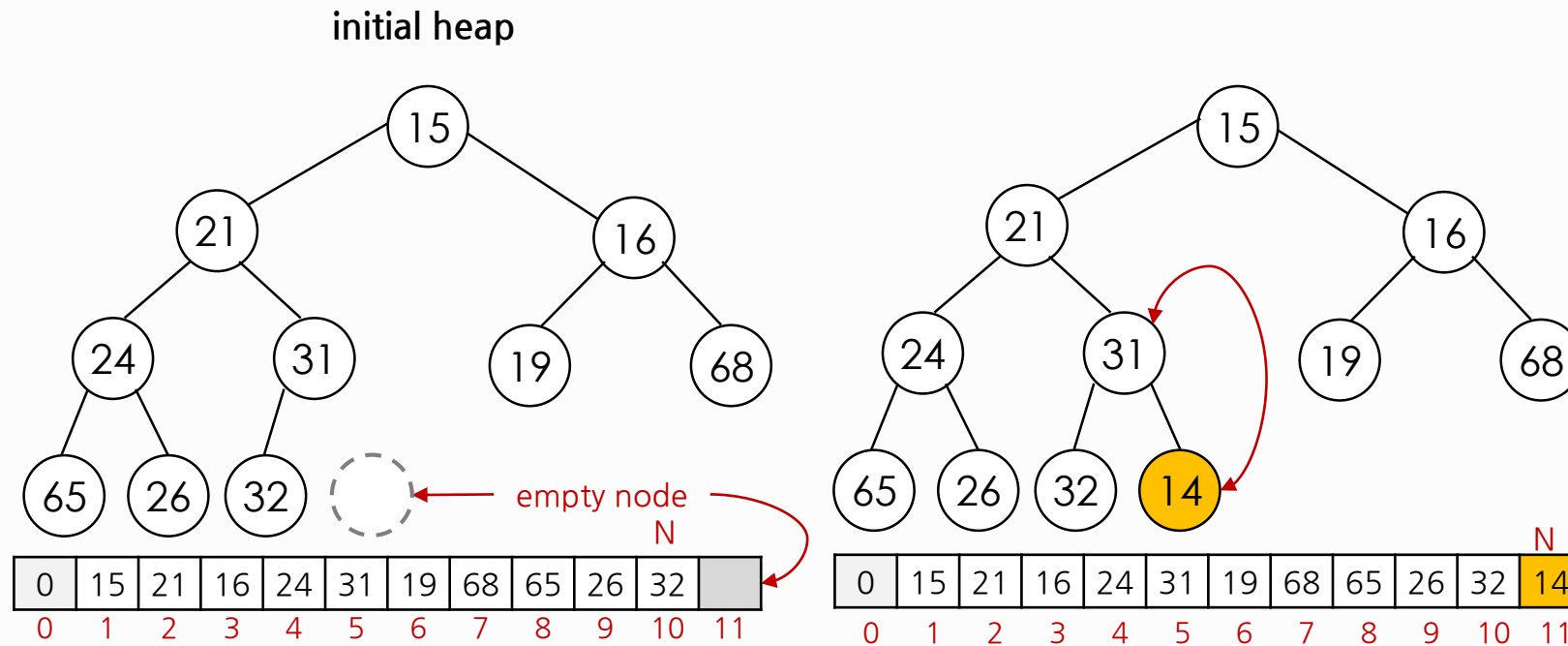
- Insert a new element **while maintaining a heap-structure**
- Move the element up the heap **while not satisfying heap-ordered**



min-heap: insert(heap, 14)

Algorithm:

- Insert a new element **while maintaining a heap-structure**
- Move the element up the heap **while not satisfying heap-ordered**

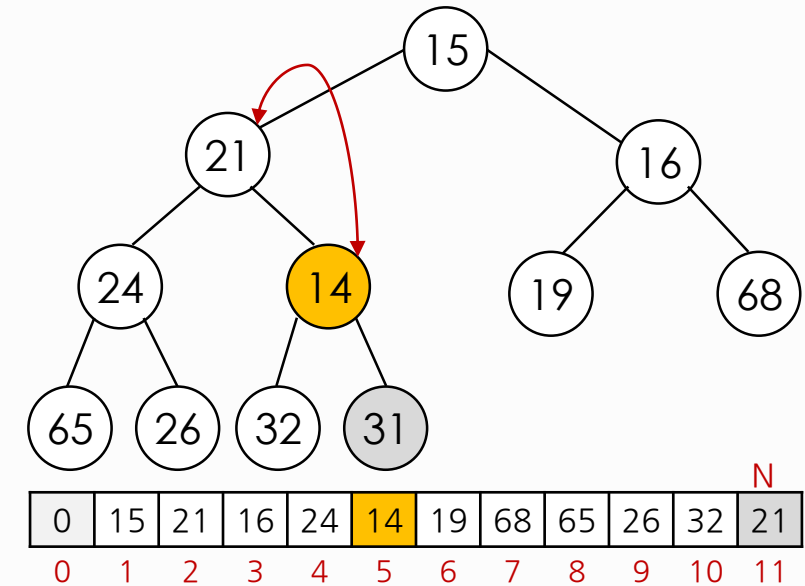
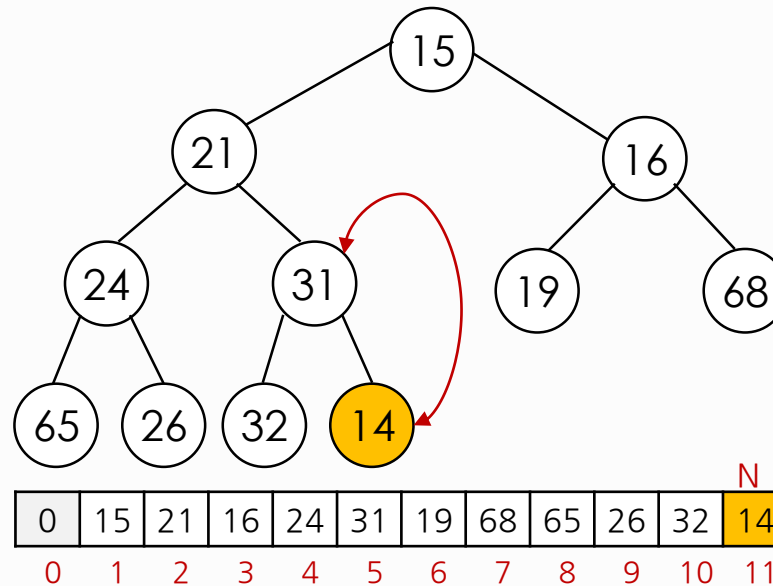
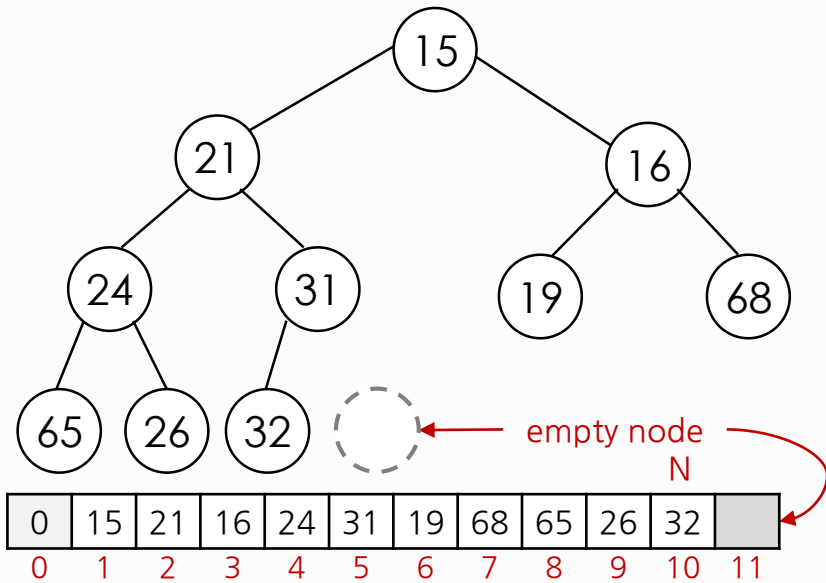


min-heap: insert(heap, 14)

Algorithm:

- Insert a new element **while maintaining a heap-structure**
- Move the element up the heap **while not satisfying heap-ordered**

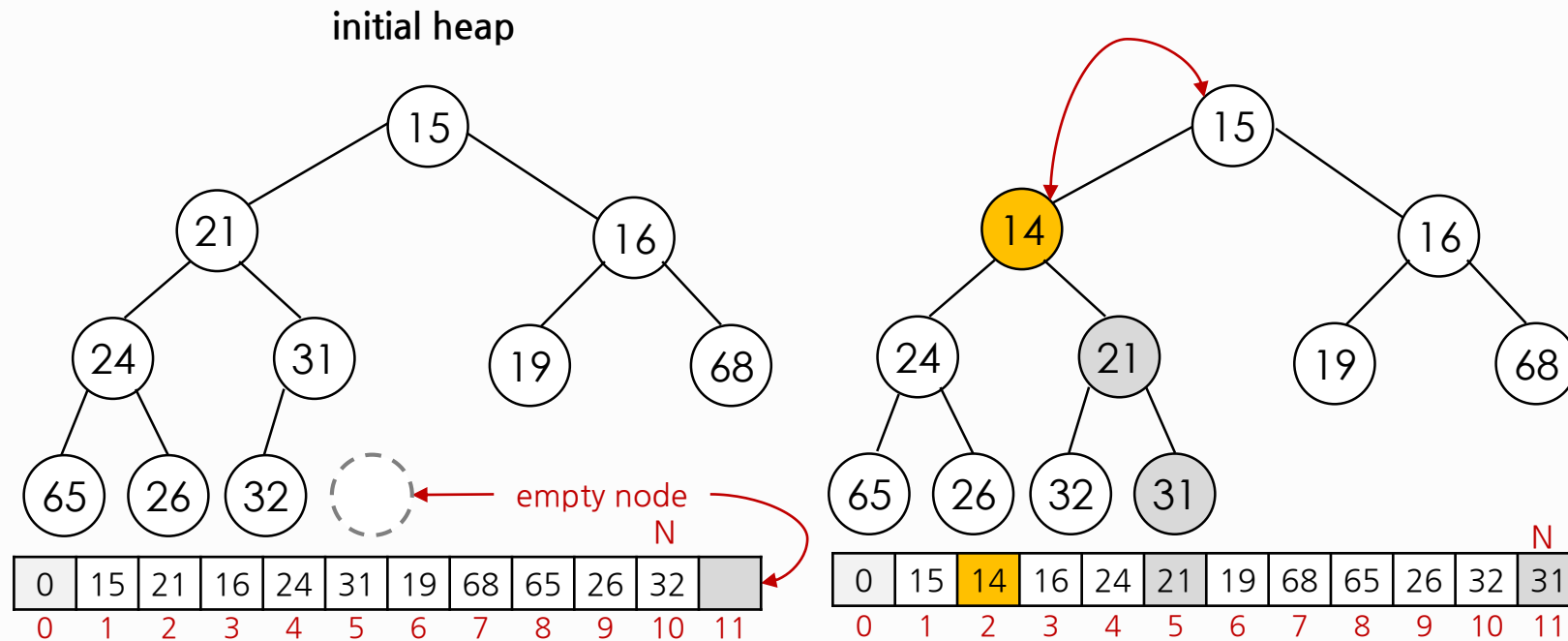
initial heap



min-heap: insert(heap, 14)

Algorithm:

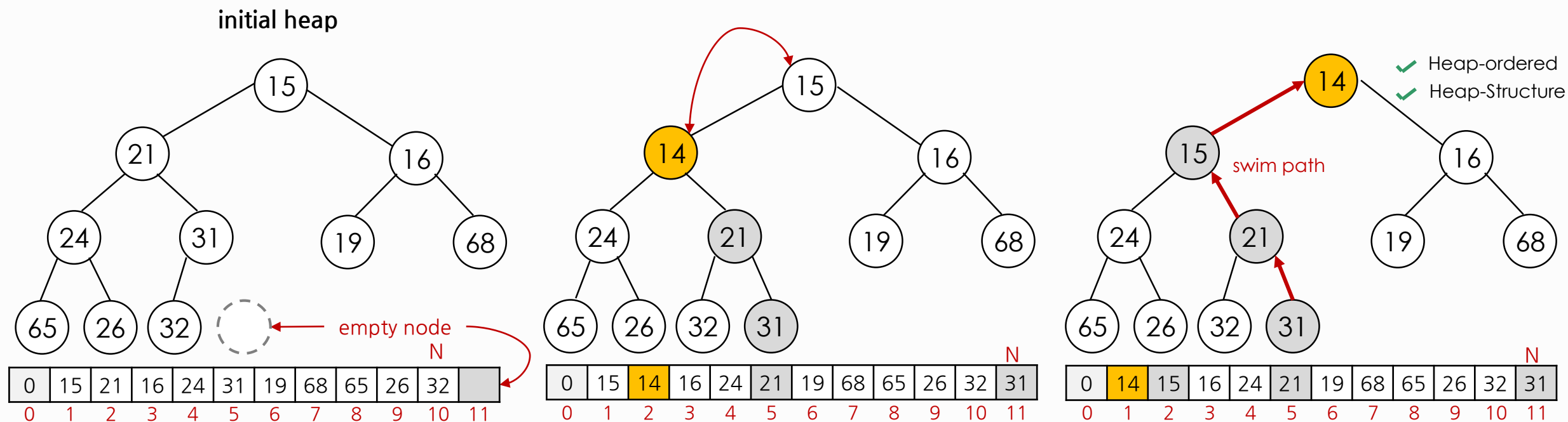
- Insert a new element **while maintaining a heap-structure**
- Move the element up the heap **while not satisfying heap-ordered**



min-heap: insert(heap, 14)

Algorithm:

- Insert a new element **while maintaining a heap-structure**
- Move the element up the heap **while not satisfying heap-ordered**

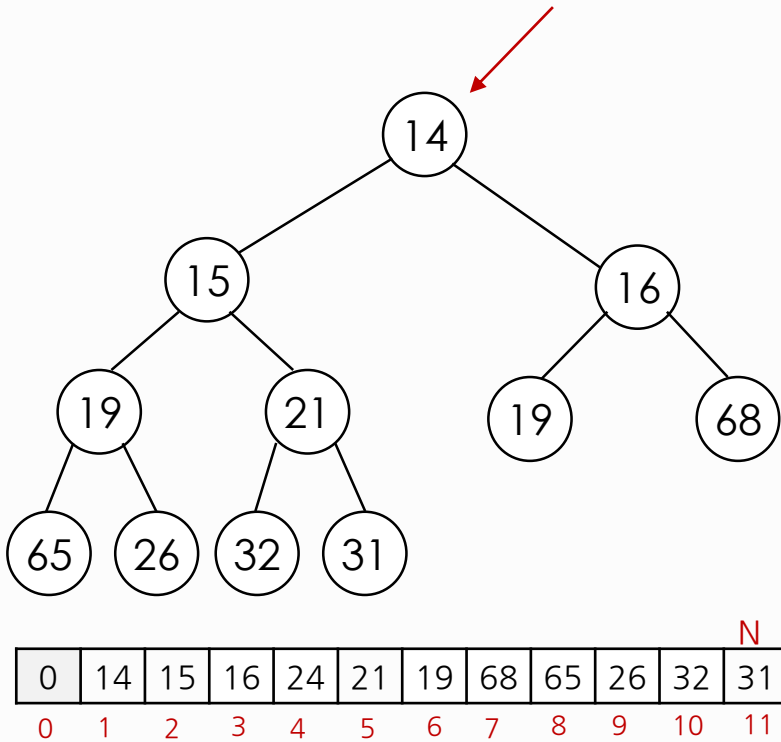


min-heap: delete() or dequeue()

- Swap the root and the last element.
- Heap decreases by one in size.
- **Move down (sink) the root** while not satisfying heap-ordered.
 - Minimum element is **always** at the root (by min-heap definition).

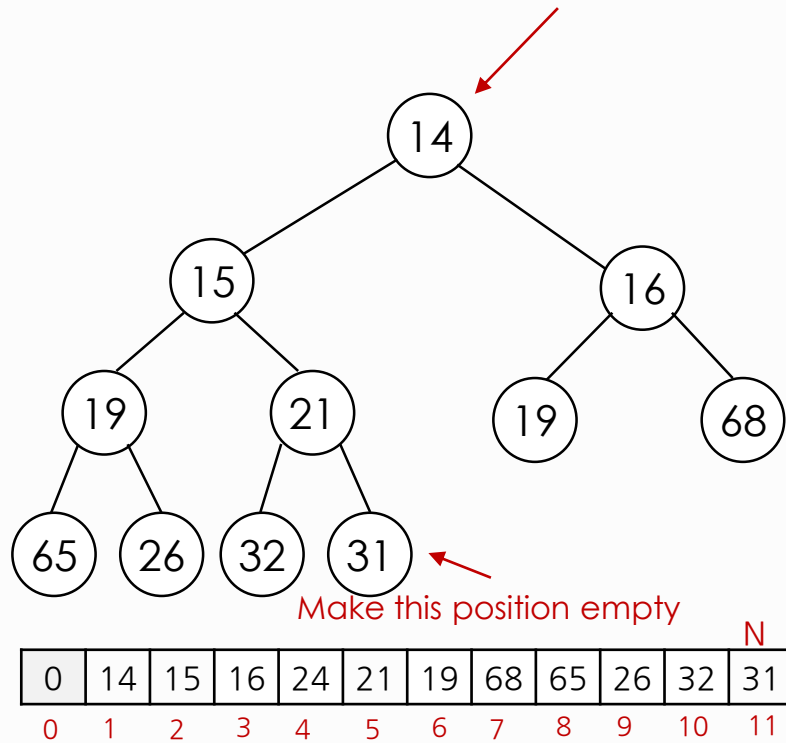
min-heap: delete() or dequeue()

- Which position of the node will be empty?

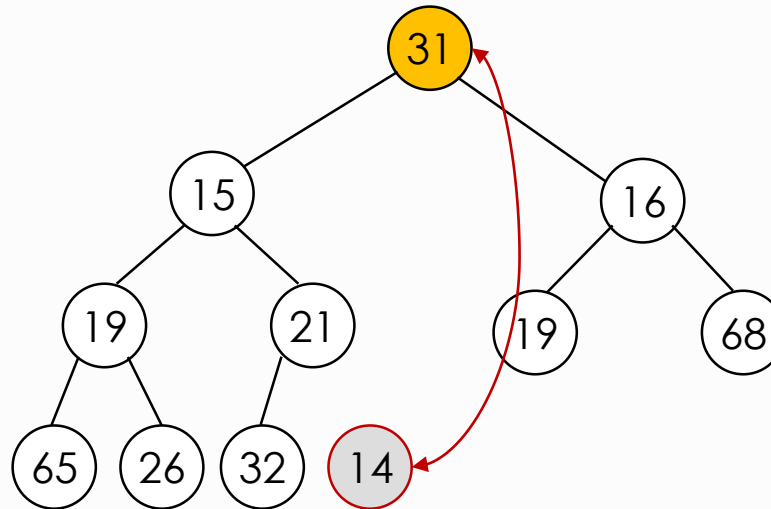


min-heap: delete() or dequeue()

- Which position of the node will be empty?

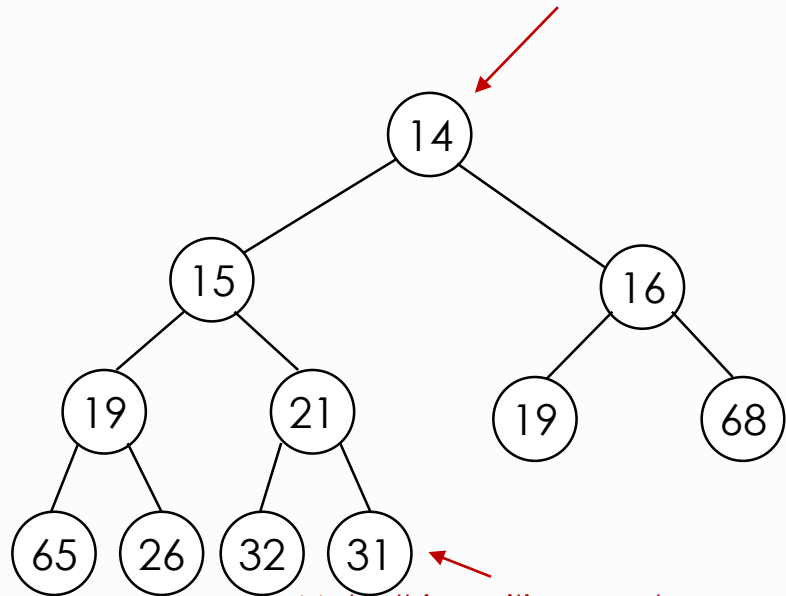


- swap the root & last
- N--**



min-heap: delete() or dequeue()

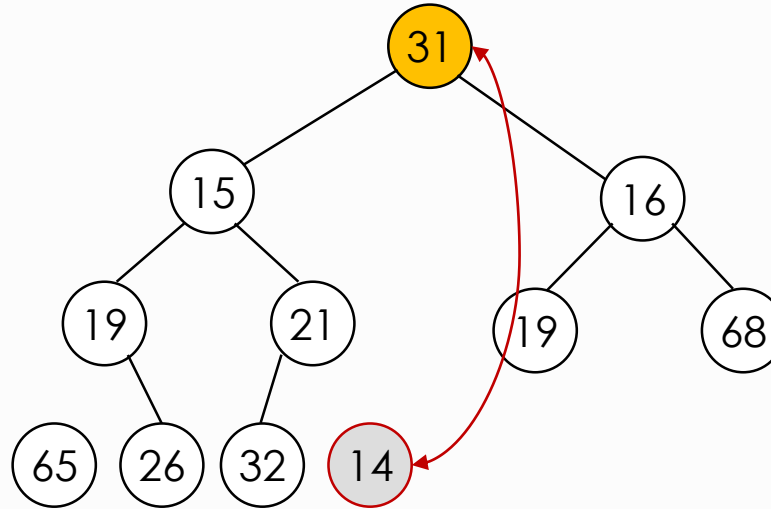
- Which position of the node will be empty?



Make this position empty

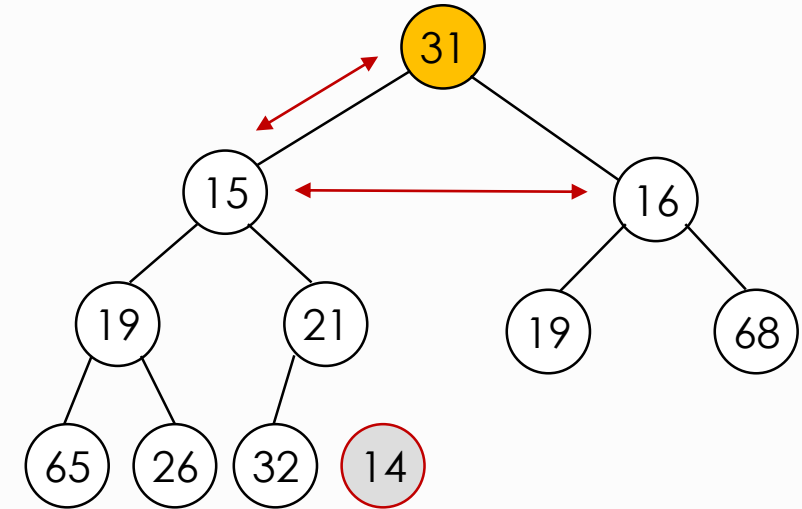
0	14	15	16	24	21	19	68	65	26	32	31
0	1	2	3	4	5	6	7	8	9	10	11

- swap the root & last
- N--



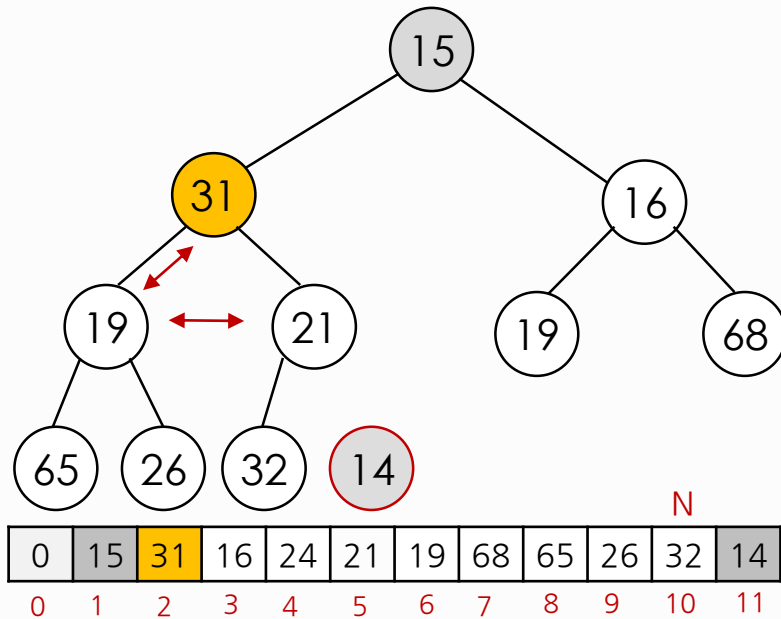
0	31	15	16	24	21	19	68	65	26	32	14
0	1	2	3	4	5	6	7	8	9	10	11

- heap-ordered?
- sink(): select one of two children and compare



min-heap: delete() or dequeue()

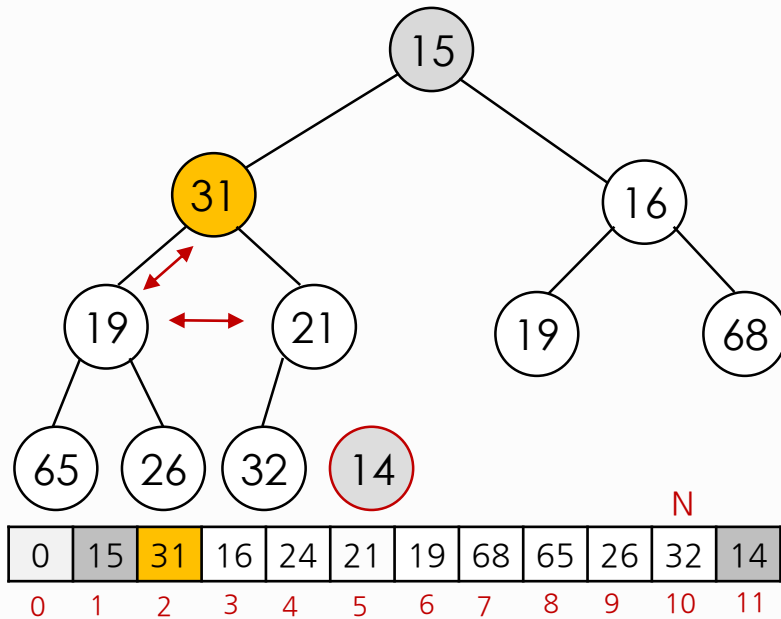
- heap-ordered?
- **sink()**: select one of two children and compare



- Is $31 > \min(14, 16)$?
- Yes - swap 31 with $\min(14, 16)$

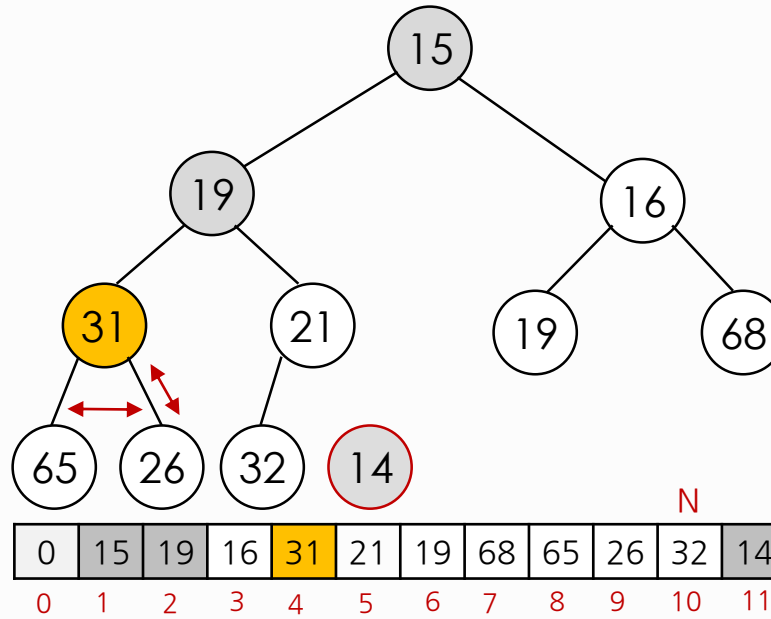
min-heap: delete() or dequeue()

- heap-ordered?
- **sink()**: select one of two children and compare



- Is $31 > \min(14, 16)$?
- Yes - swap 31 with $\min(14, 16)$

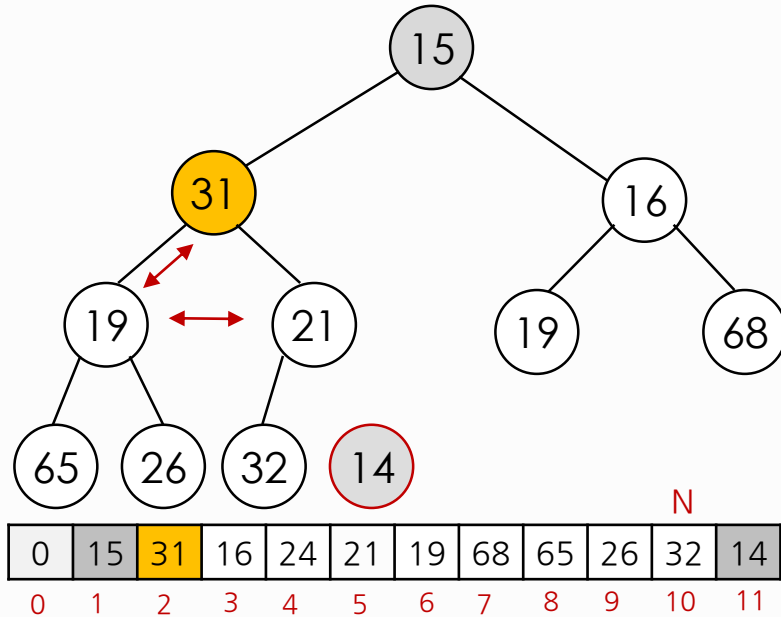
- heap-ordered?
- **sink()**: select one of two children and compare



- Is $31 > \min(19, 21)$?
- Yes - swap 31 with $\min(19, 21)$

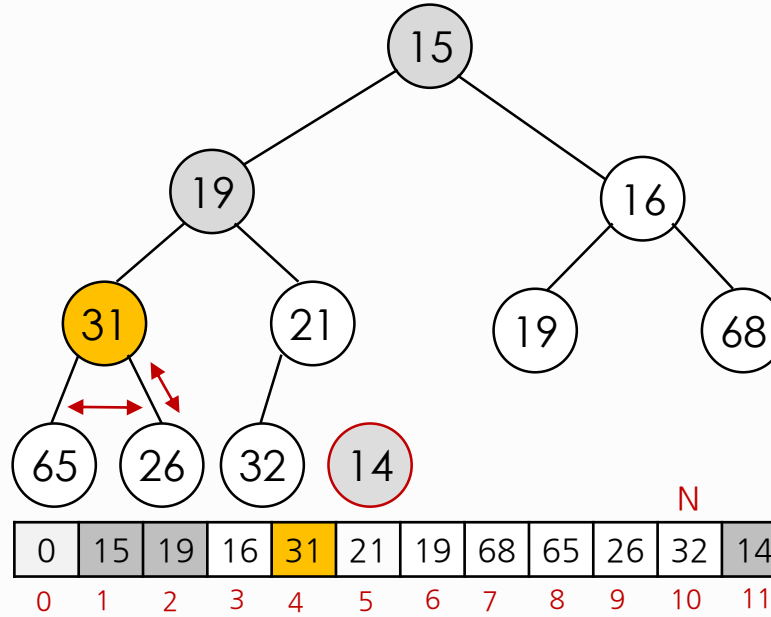
min-heap: delete() or dequeue()

- heap-ordered?
- **sink()**: select one of two children and compare



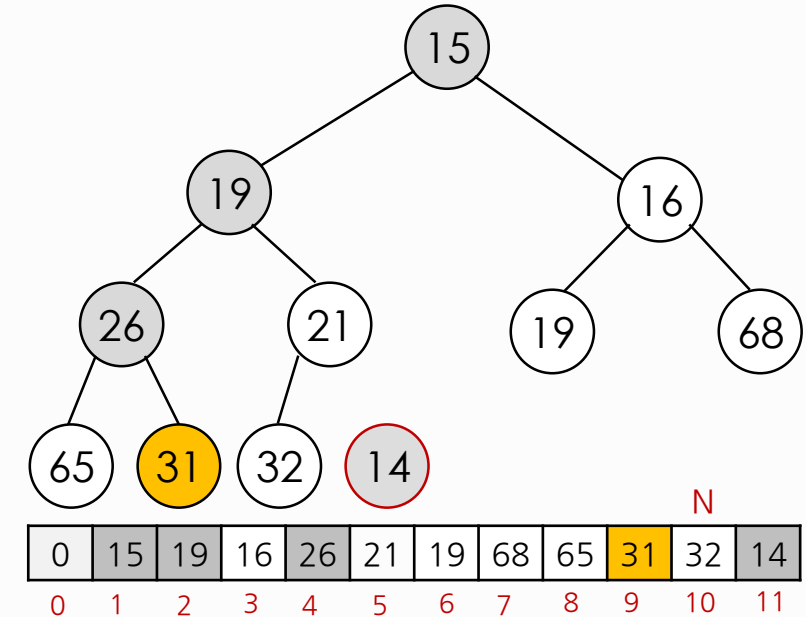
- Is $31 > \min(14, 16)$?
- Yes - swap 31 with $\min(14, 16)$

- heap-ordered?
- **sink()**: select one of two children and compare



- Is $31 > \min(19, 21)$?
- Yes - swap 31 with $\min(19, 21)$

- heap-ordered?
- **sink()**: select one of two children and compare



- Is $31 > \min(65, 26)$?
- Yes - swap 31 with $\min(65, 26)$

✓ Heap-ordered
✓ Heap-Structure

Binary heap: Time complexity:

- Level of heap is $\lfloor \log_2 N \rfloor$
- insert: $O(\log N)$ for each insert
 - In practice, expect less
- delete: $O(\log N)$ // deleting root node or any node
- increase/decrease key: $O(\log N)$

Implementation	Insert	Delete	max
Unordered array	1	N	N
Ordered array	N	1	1
Binary heap	log N	log N	1

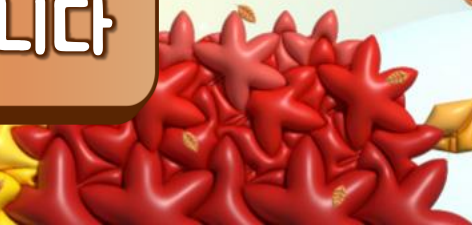
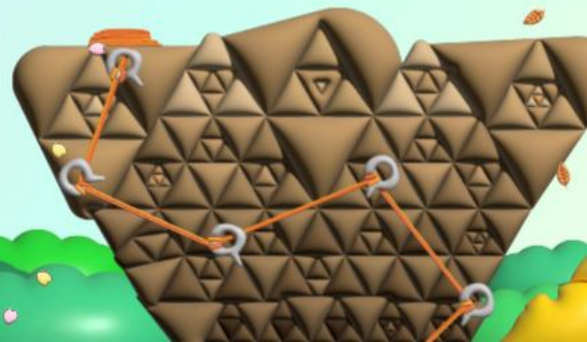
Mission Completed

학습 정리

- 1) 힙(Heap)은 우선순위 큐(Priority queue)에 주로 사용된다
- 2) 힙(Heap)에서의 삽입과 삭제 작업은 최대 $O(\log n)$ 시간복잡도로 수행할 수 있다

파이썬으로 배우는 데이터 구조

수고했습니다
곧 다음 시간에
다시 뵙겠습니다



Data Structures in Python

- Heap and Priority Queue
- **Heap Coding**
- Min/MaxHeap and Heap sort

Proof:

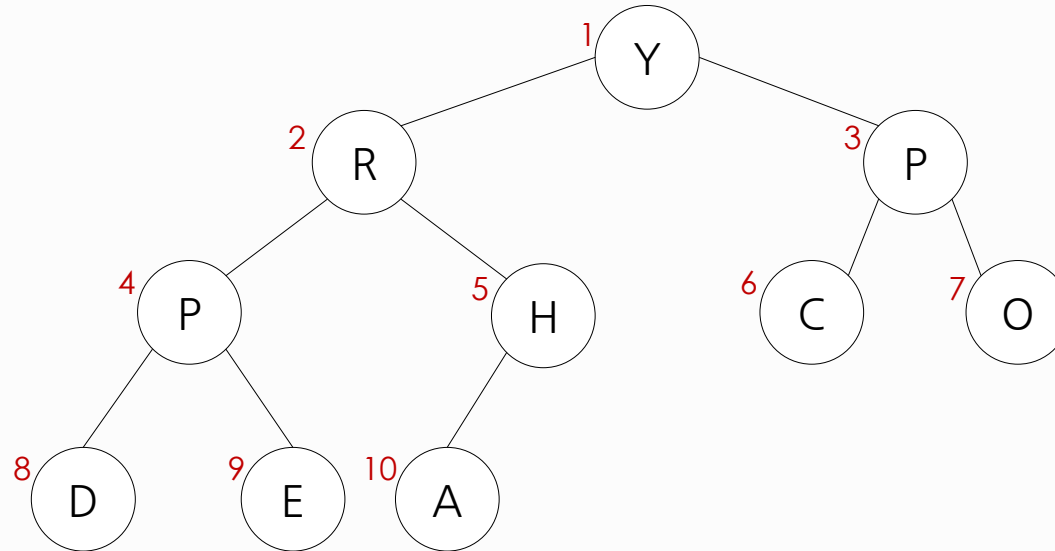
<https://stackoverflow.com/questions/9755721/how-can-building-a-heap-be-on-time-complexity>
<https://www.insertingwiththeweb.com/data-structures/binary-heap/build-heap-proof/>
<https://www.quora.com/How-is-the-time-complexity-of-building-a-heap-is-o-n>

References in Korean:

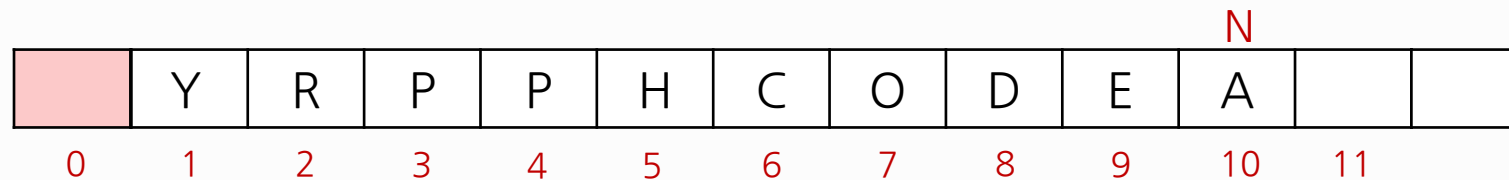
<https://ratsgo.github.io/data%20structure&algorithm/2017/09/27/heapsort/>
<https://zeddios.tistory.com/56>

max-heap example

- **Insert:** Add node at end, then swim it up.
- **Remove:** Swap root with last node, then sink down.

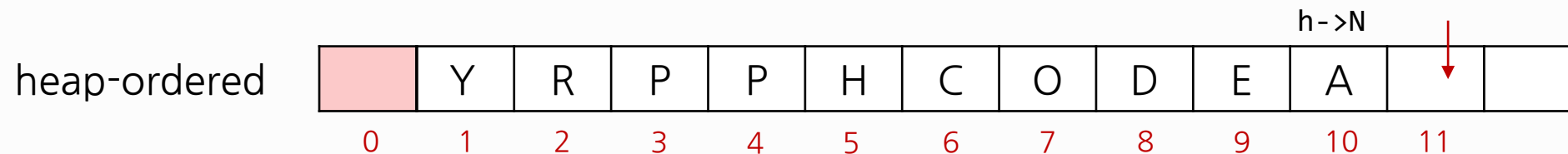
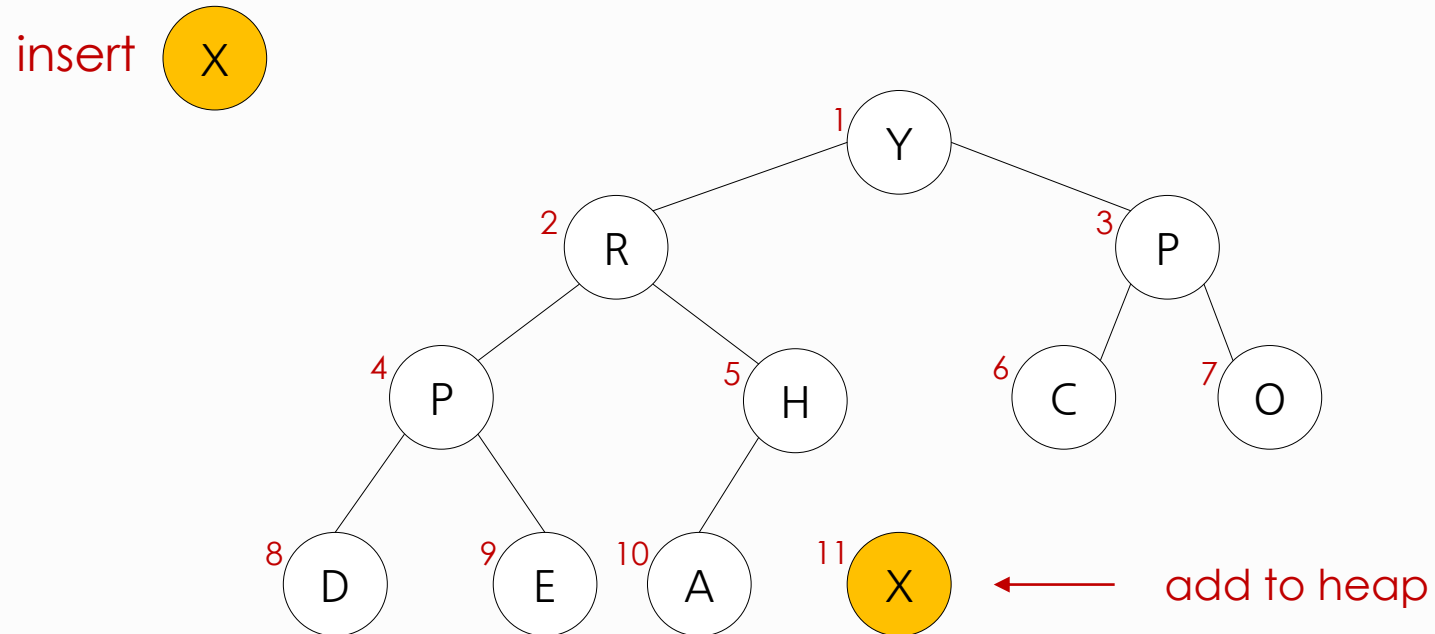


heap-ordered



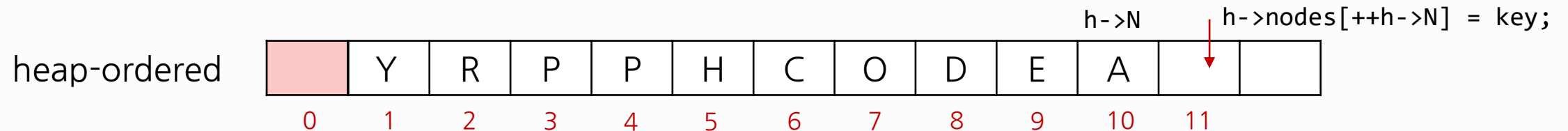
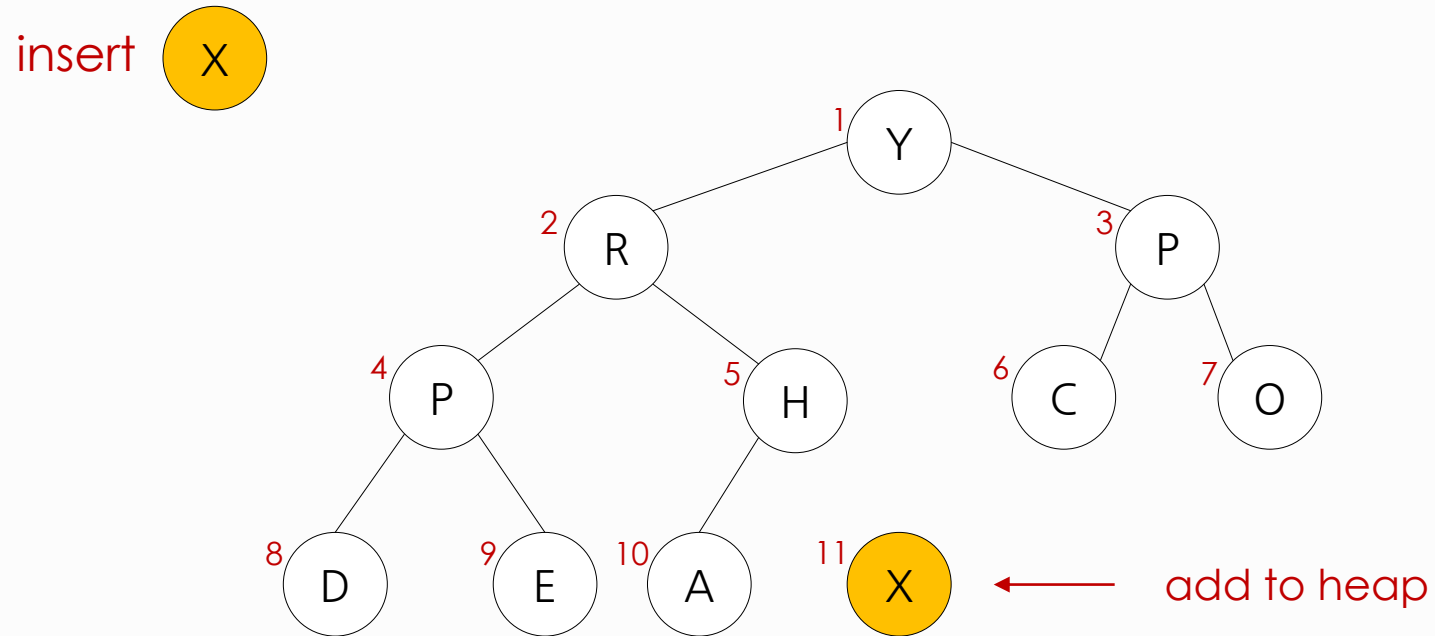
max-heap example

- **Insert:** Add node at end, then swim it up.
- **Remove:** Swap root with last node, then sink down.



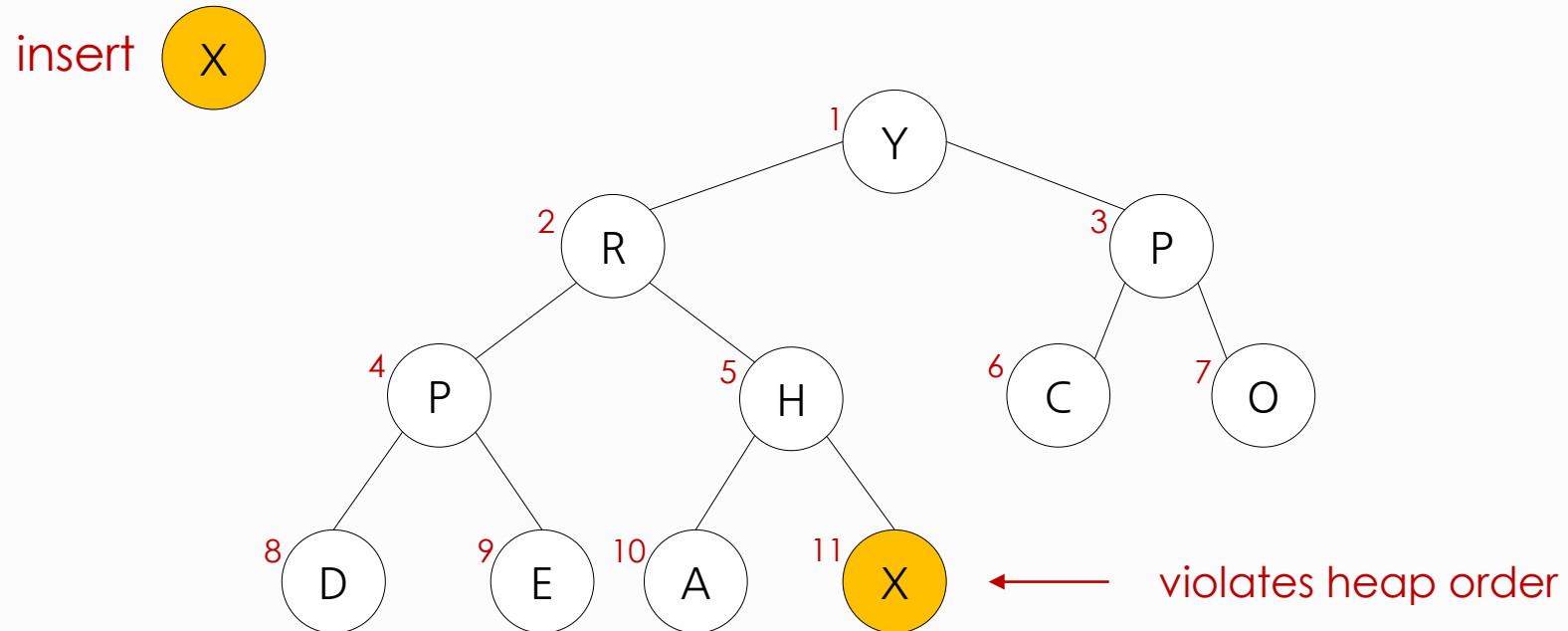
max-heap example

- **Insert:** Add node at end, then swim it up.
- **Remove:** Swap root with last node, then sink down.

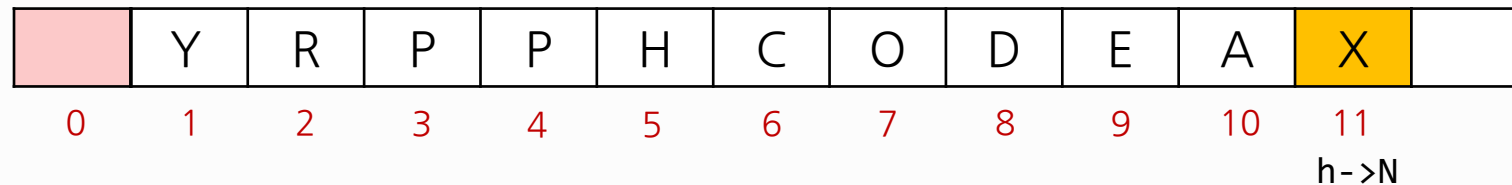


max-heap example

- **Insert:** Add node at end, then swim it up.
- **Remove:** Swap root with last node, then sink down.

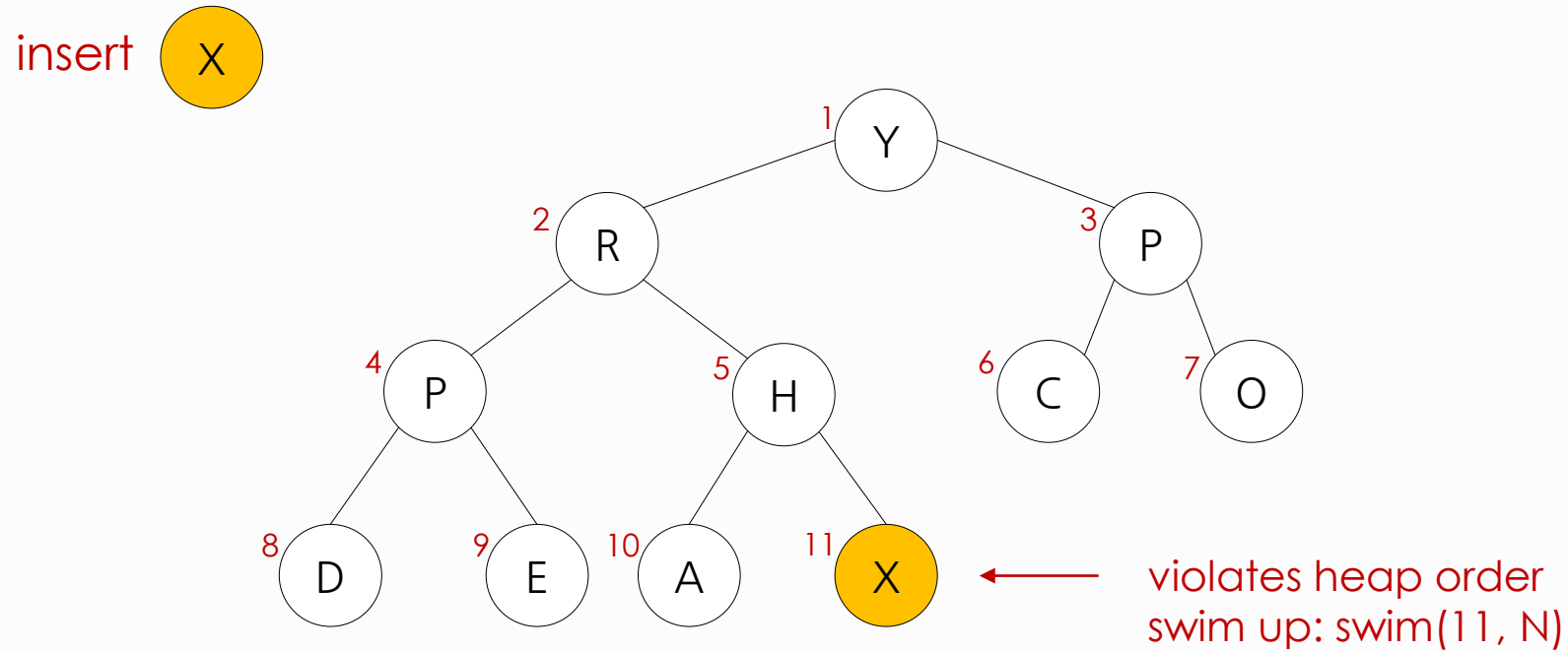


heap-ordered

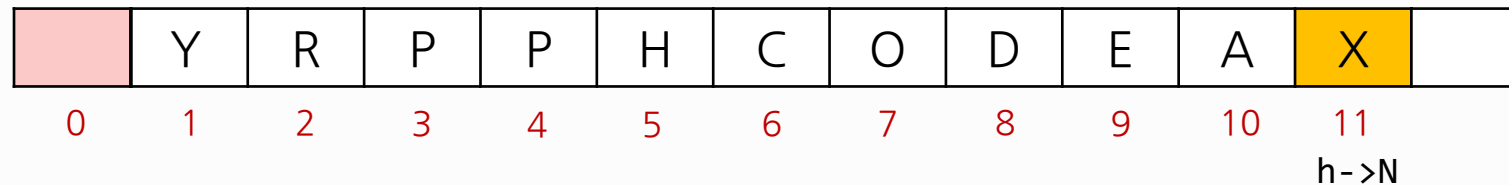


max-heap example

- **Insert:** Add node at end, then swim it up.
- **Remove:** Swap root with last node, then sink down.

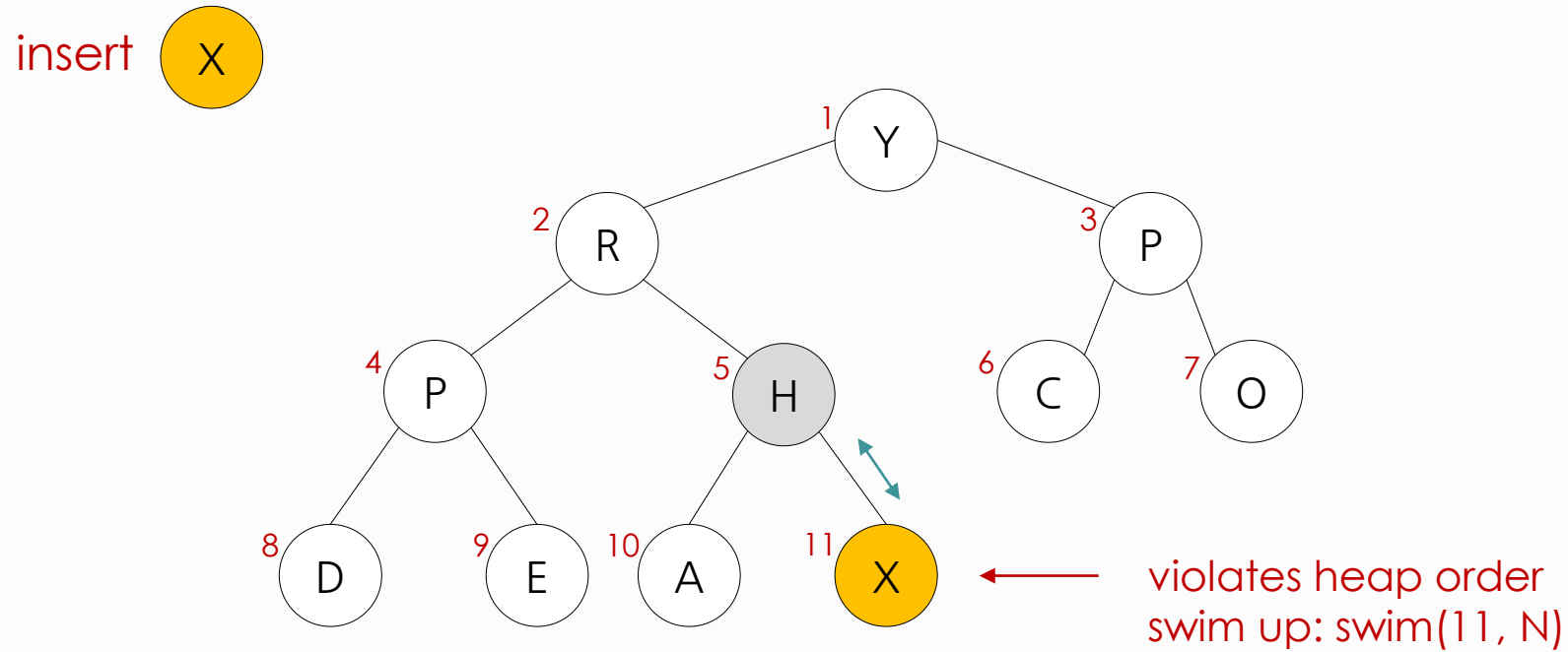


heap-ordered

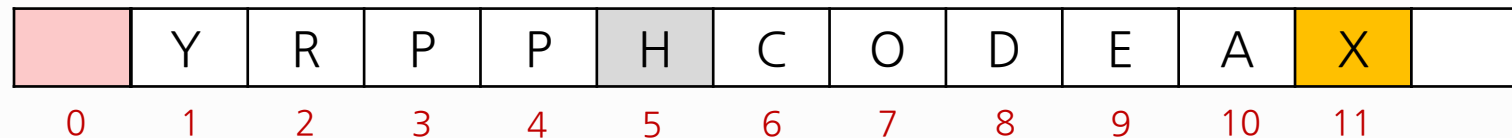


max-heap example

- **Insert:** Add node at end, then swim it up.
- **Remove:** Swap root with last node, then sink down.

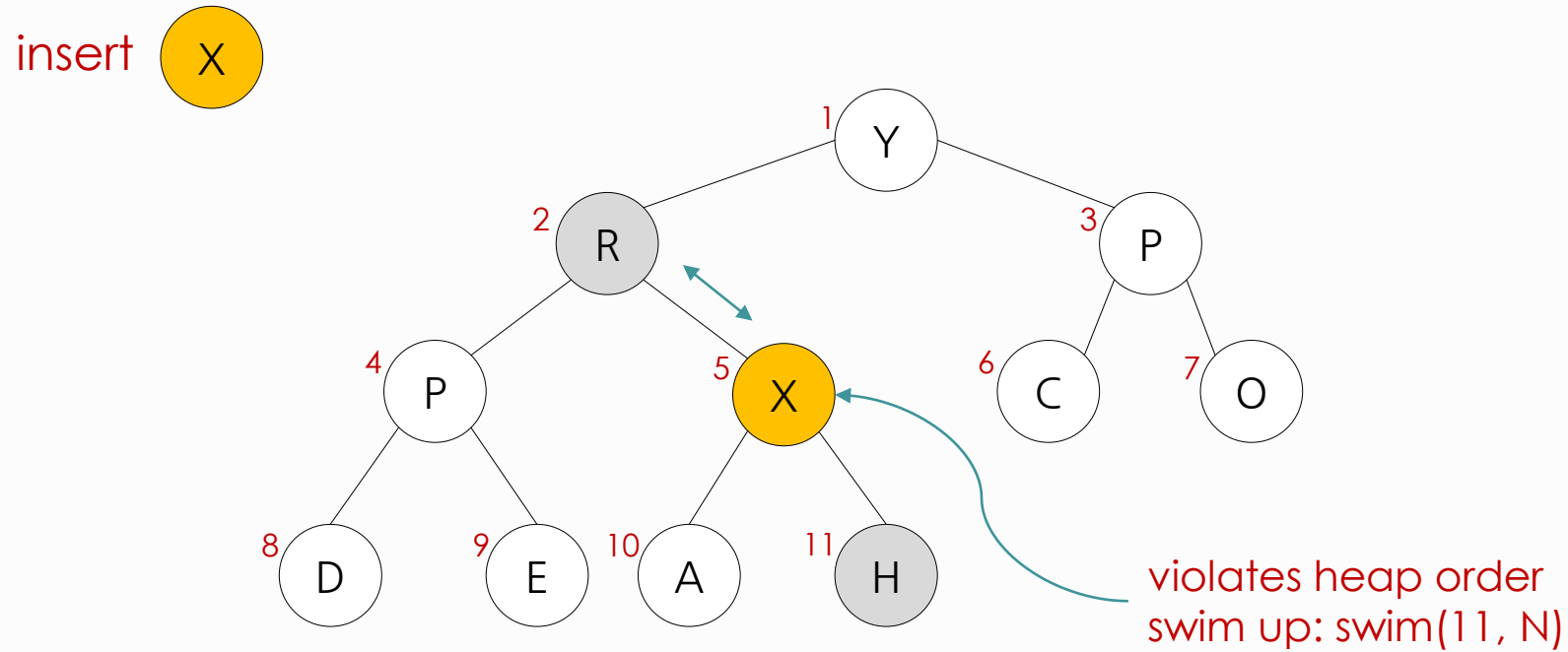


heap-ordered

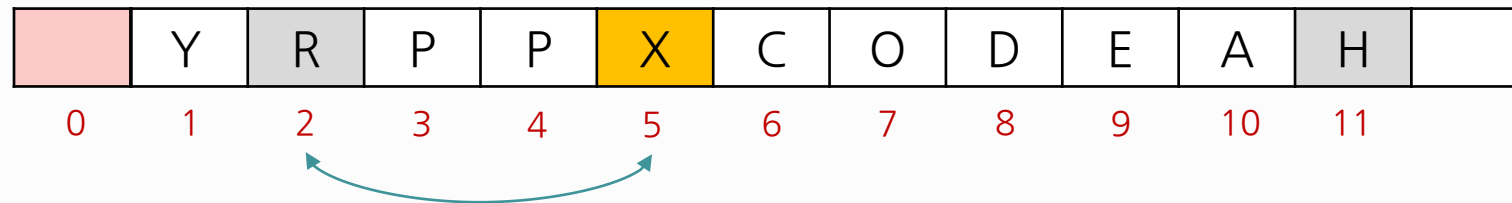


max-heap example

- **Insert:** Add node at end, then swim it up.
- **Remove:** Swap root with last node, then sink down.

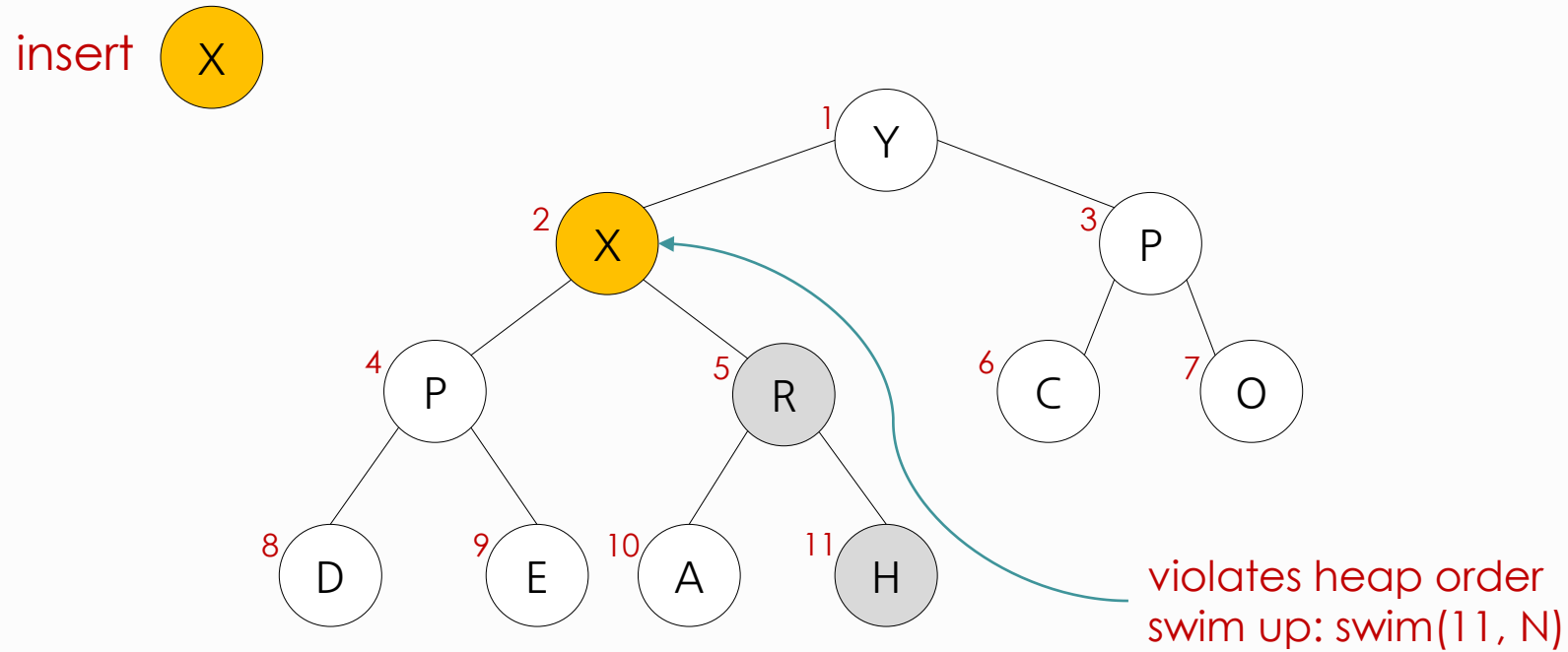


heap-ordered

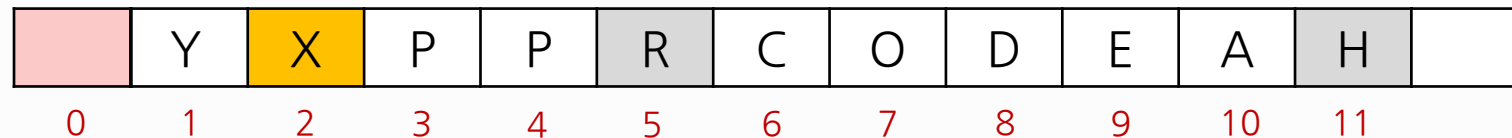


max-heap example

- **Insert:** Add node at end, then swim it up.
- **Remove:** Swap root with last node, then sink down.

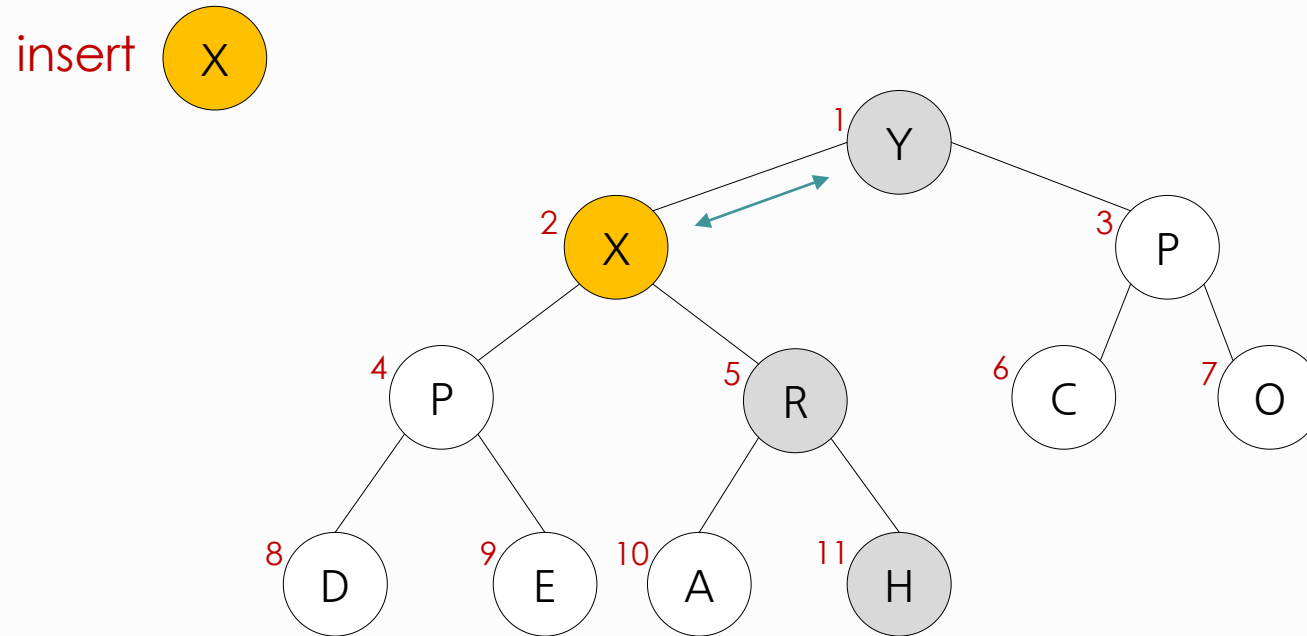


heap-ordered

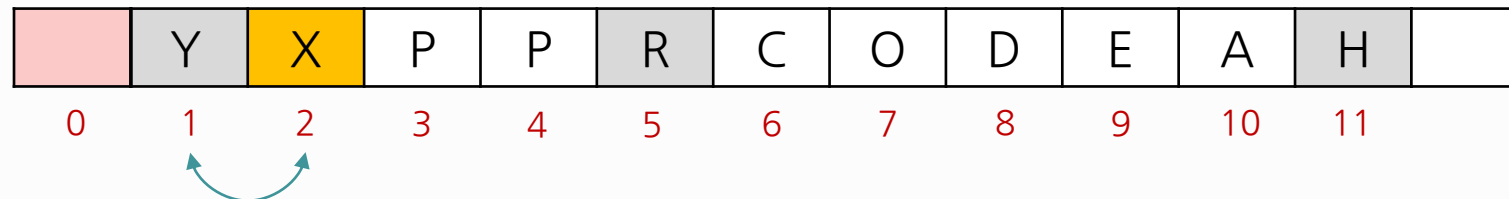


max-heap example

- **Insert:** Add node at end, then swim it up.
- **Remove:** Swap root with last node, then sink down.

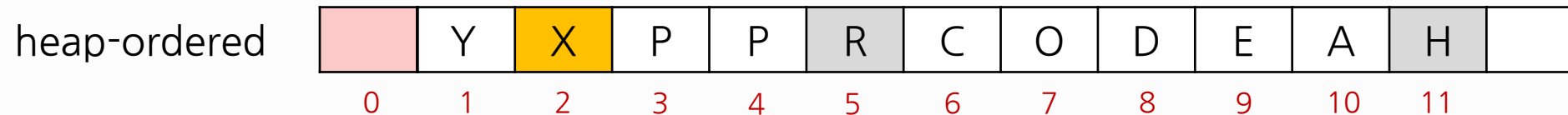
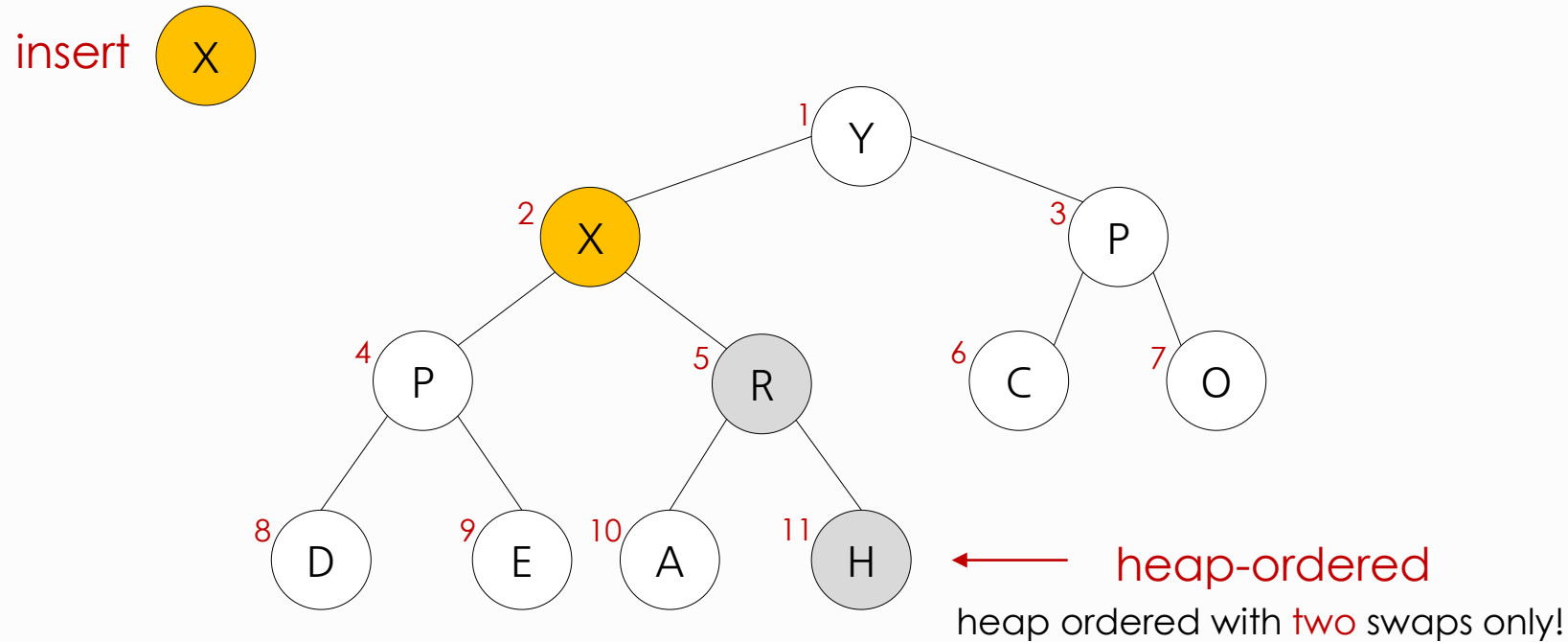


heap-ordered



max-heap example

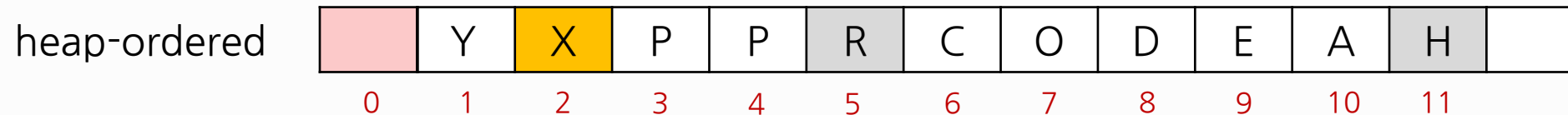
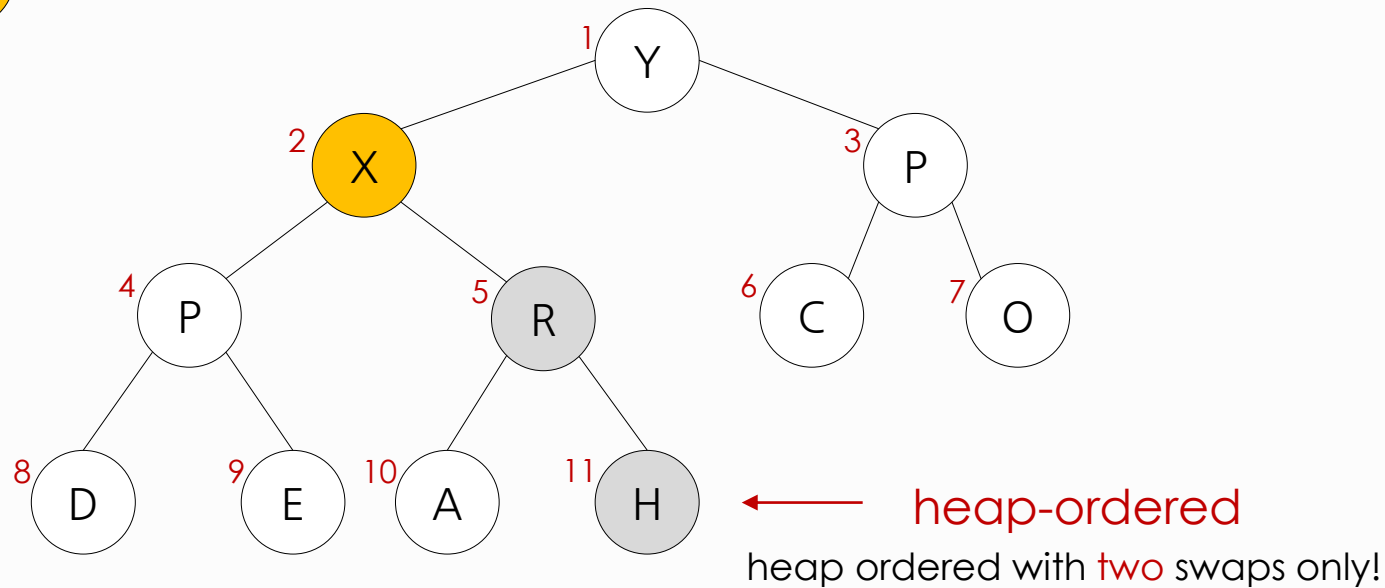
- **Insert:** Add node at end, then swim it up.
- **Remove:** Swap root with last node, then sink down.



max-heap example

- **Insert:** Add node at end, then swim it up.
- **Remove:** Swap root with last node, then sink down.

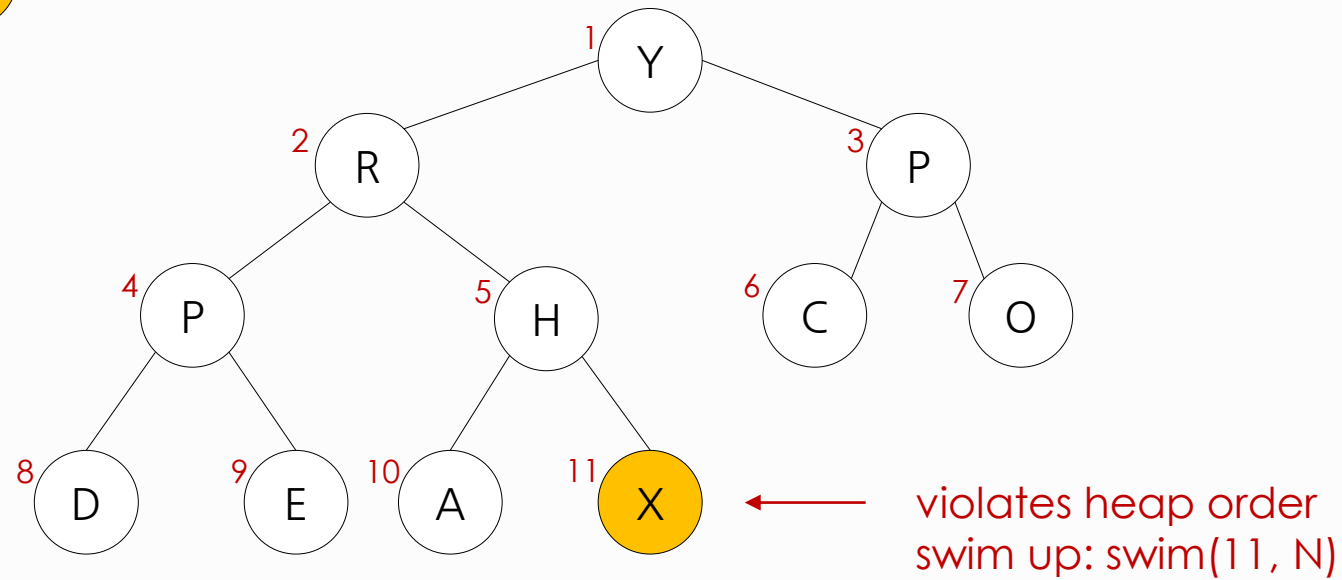
How to code insert 



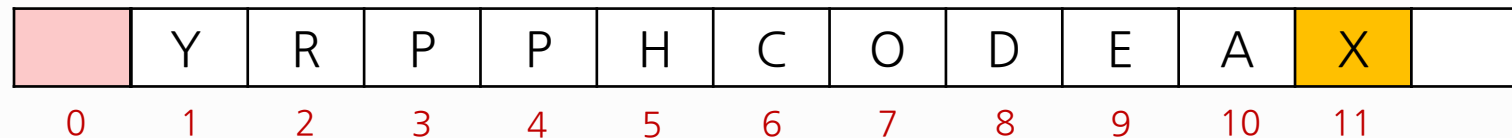
max-heap example

- **Insert:** Add node at end, then swim it up.
- **Remove:** Swap root with last node, then sink down.

How to code insert 



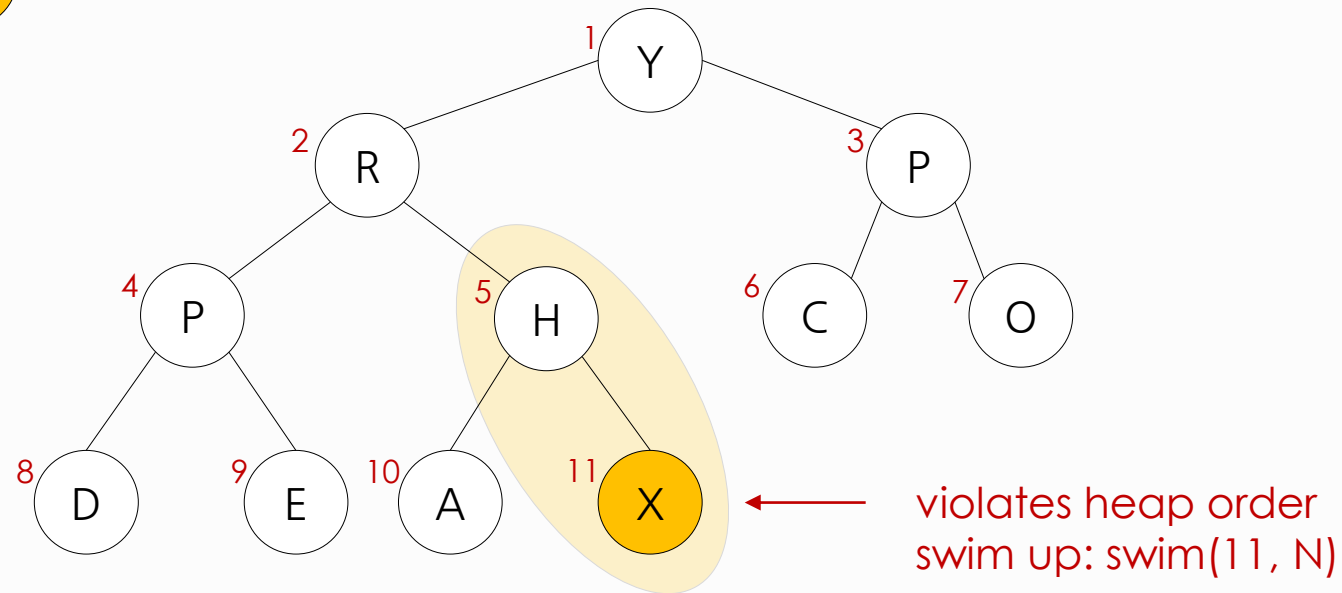
heap-ordered



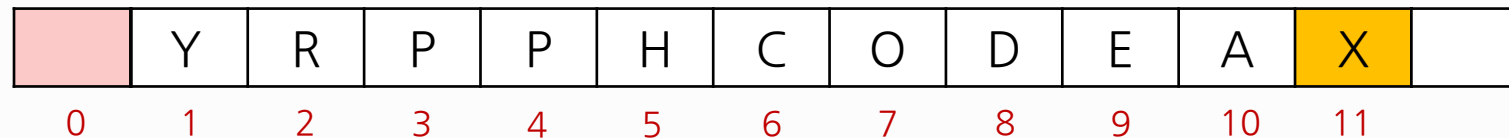
max-heap example

- **Insert:** Add node at end, then swim it up.
- **Remove:** Swap root with last node, then sink down.

How to code insert 



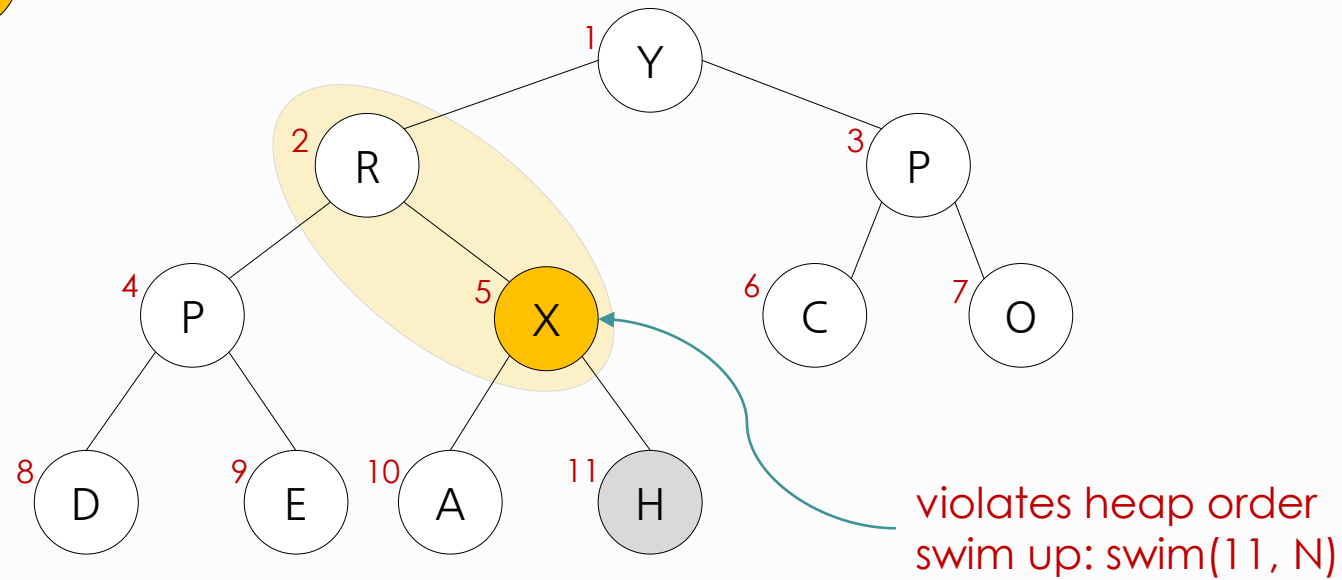
heap-ordered



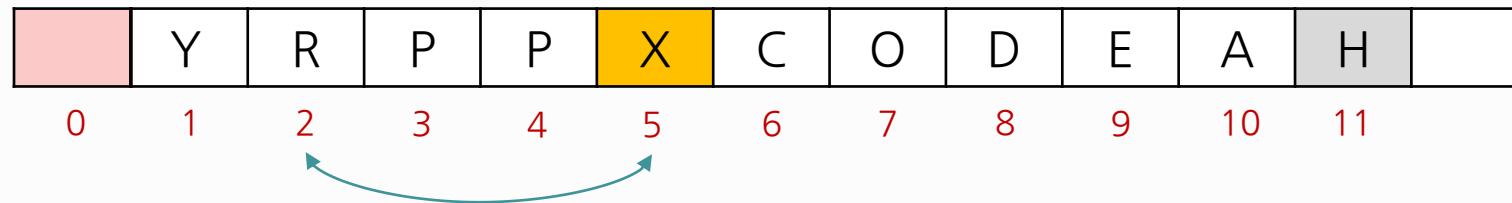
max-heap example

- **Insert:** Add node at end, then swim it up.
- **Remove:** Swap root with last node, then sink down.

How to code insert 



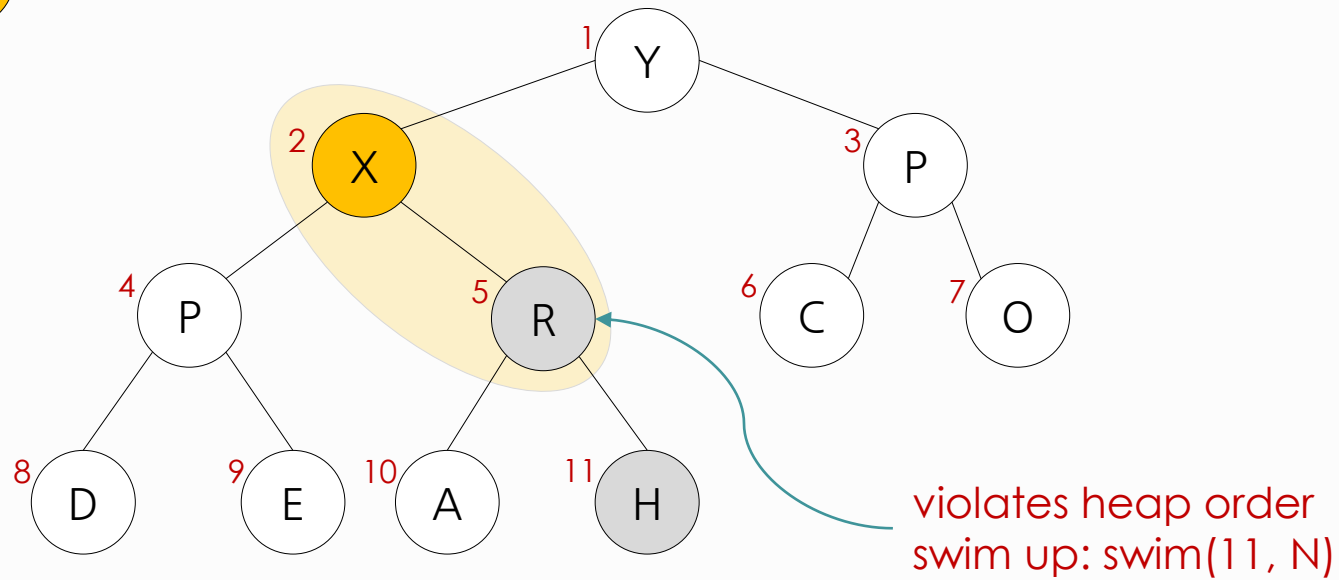
heap-ordered



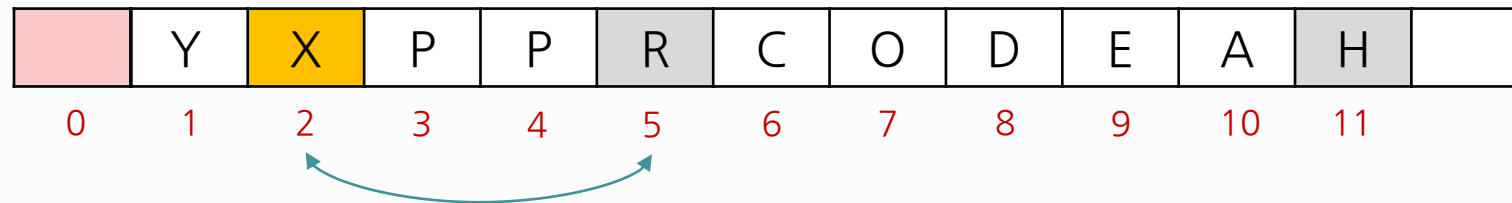
max-heap example

- **Insert:** Add node at end, then swim it up.
- **Remove:** Swap root with last node, then sink down.

How to code insert 



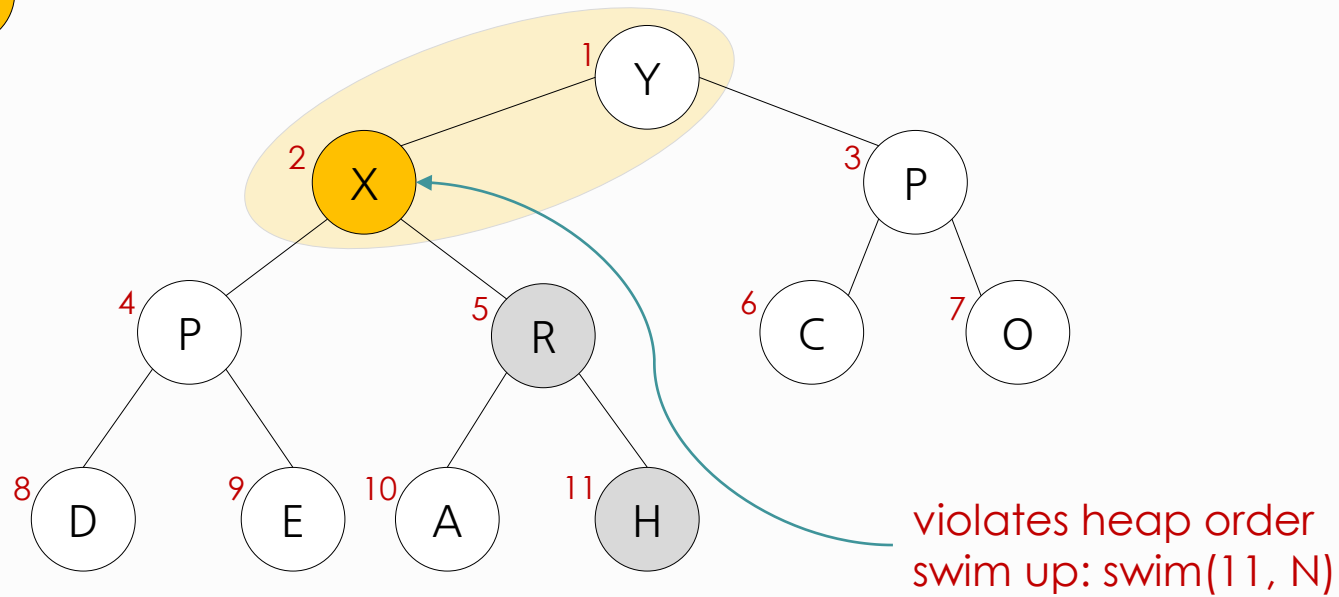
heap-ordered



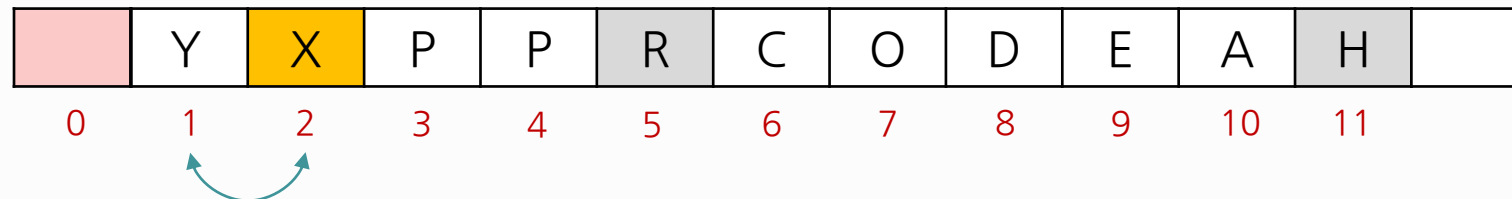
max-heap example

- **Insert:** Add node at end, then swim it up.
- **Remove:** Swap root with last node, then sink down.

How to code insert X



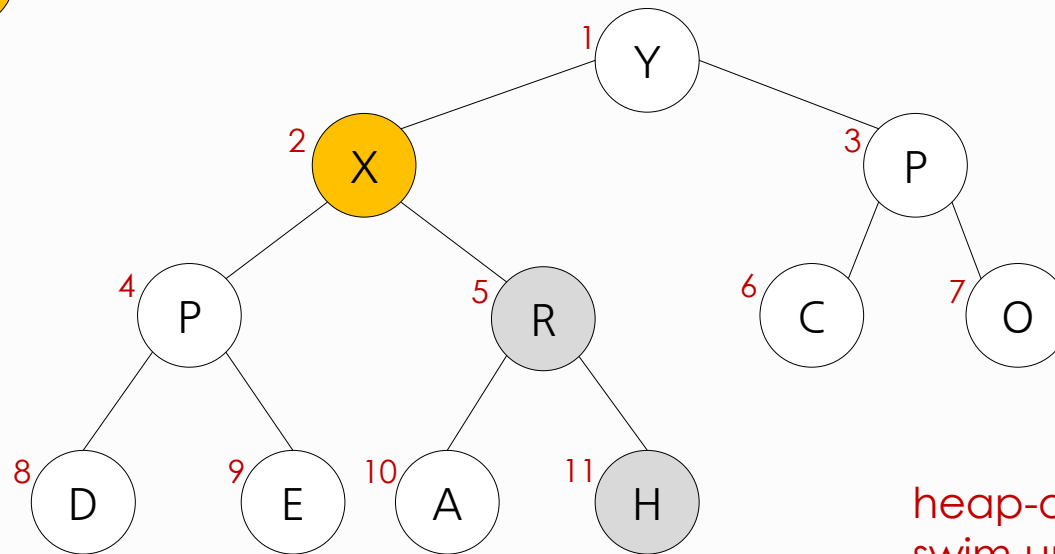
heap-ordered



max-heap example

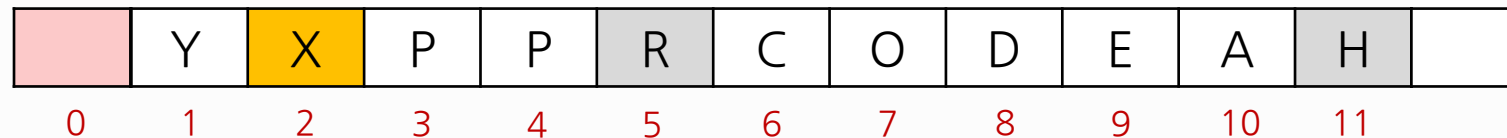
- **Insert:** Add node at end, then swim it up.
- **Remove:** Swap root with last node, then sink down.

How to code insert 



heap-ordered
swim up: swim(11, N)

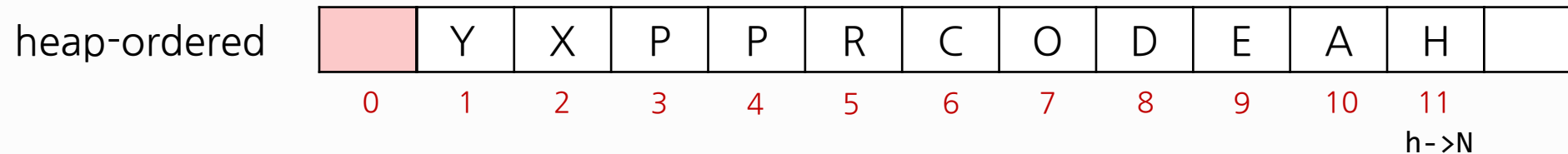
heap-ordered



max-heap example

- **Insert:** Add node at end, then swim it up.
- **Remove:** Swap root with last node, then sink down.

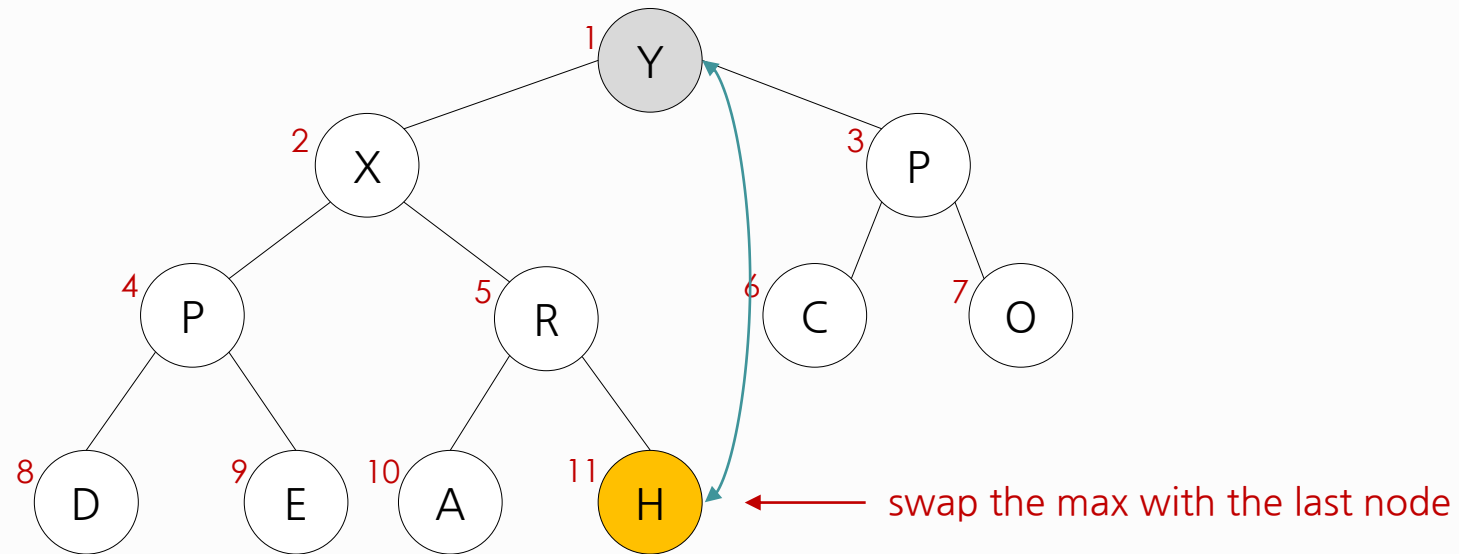
remove the max (root) – pop in priority queue



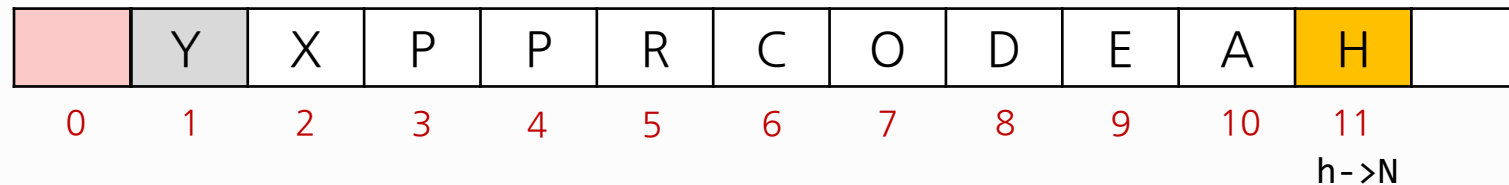
max-heap example

- **Insert:** Add node at end, then swim it up.
- **Remove:** Swap root with last node, then sink down.

remove the max (root)



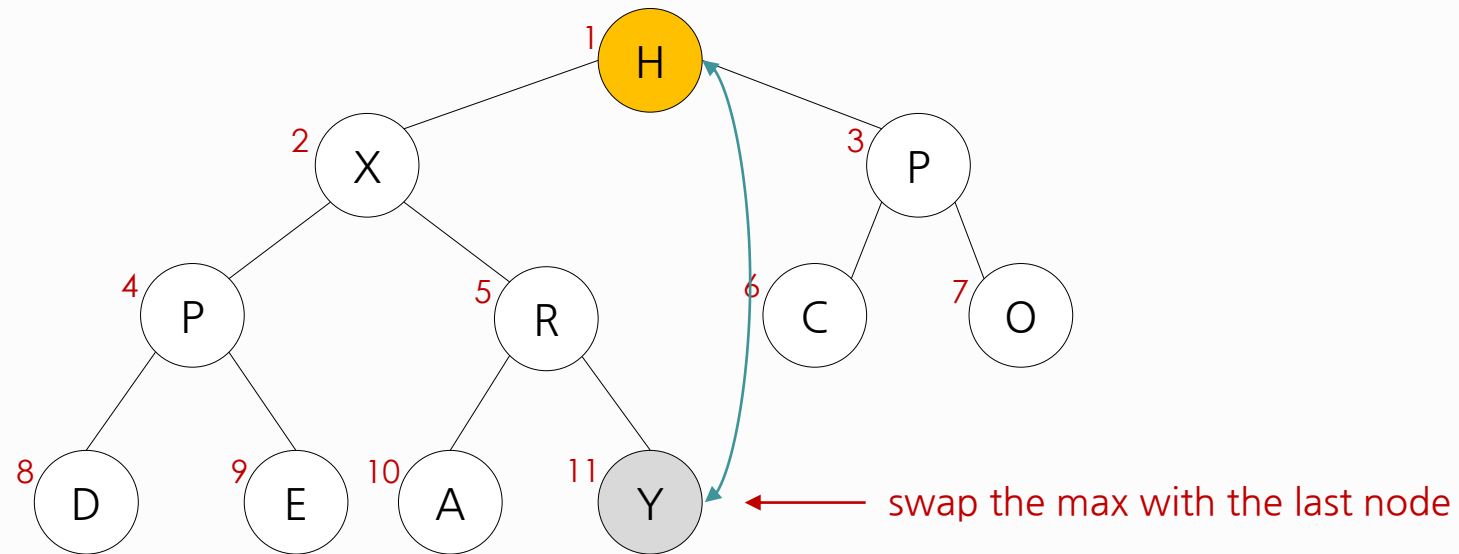
heap-ordered



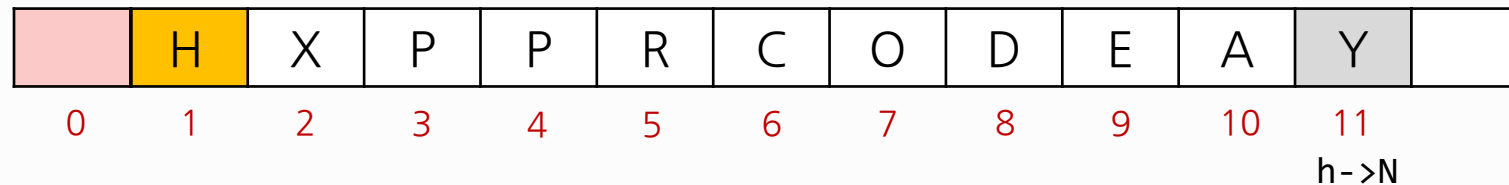
max-heap example

- **Insert:** Add node at end, then swim it up.
- **Remove:** Swap root with last node, then sink down.

remove the max (root)



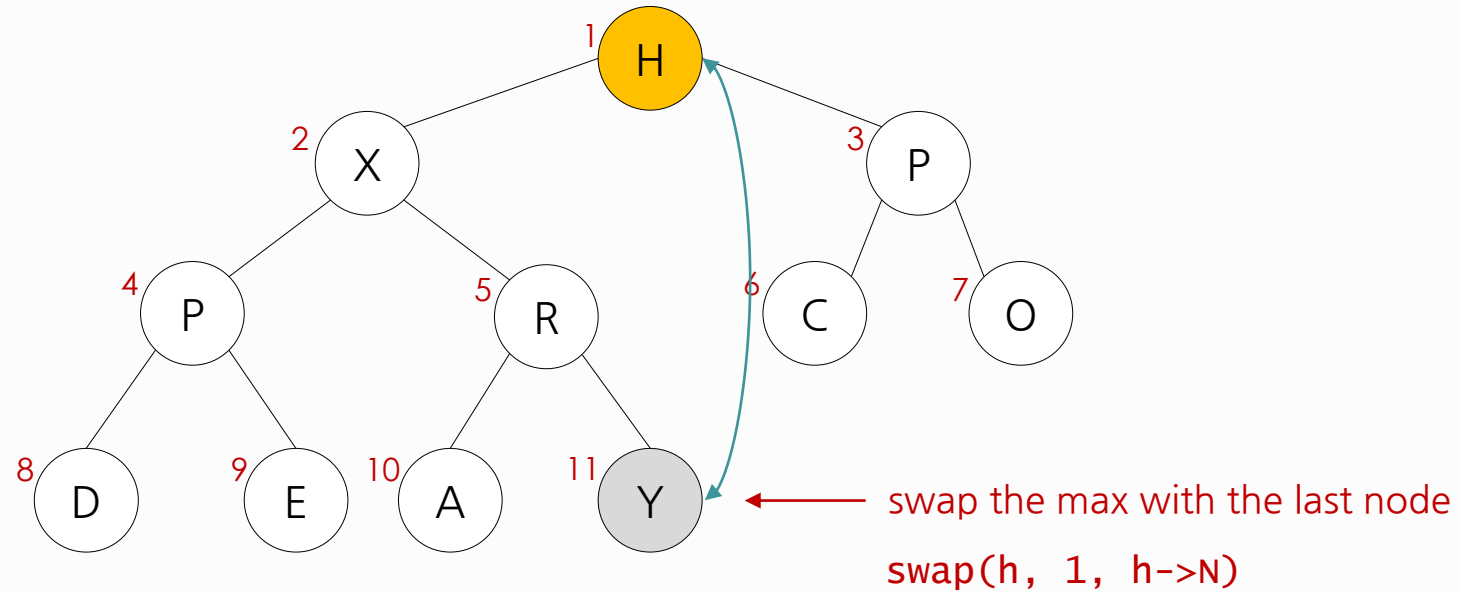
heap-ordered



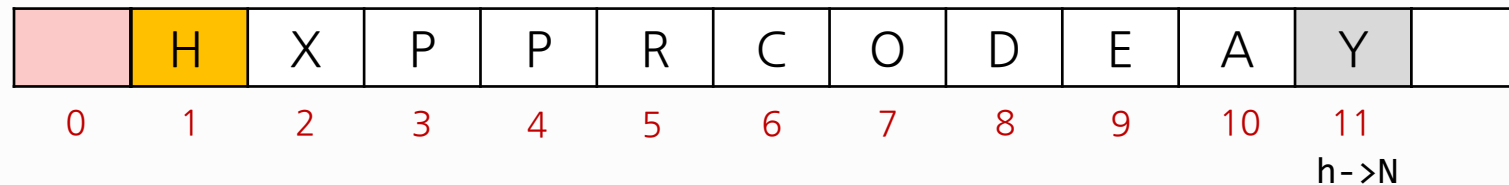
max-heap example

- **Insert:** Add node at end, then swim it up.
- **Remove:** Swap root with last node, then sink down.

remove the max (root)



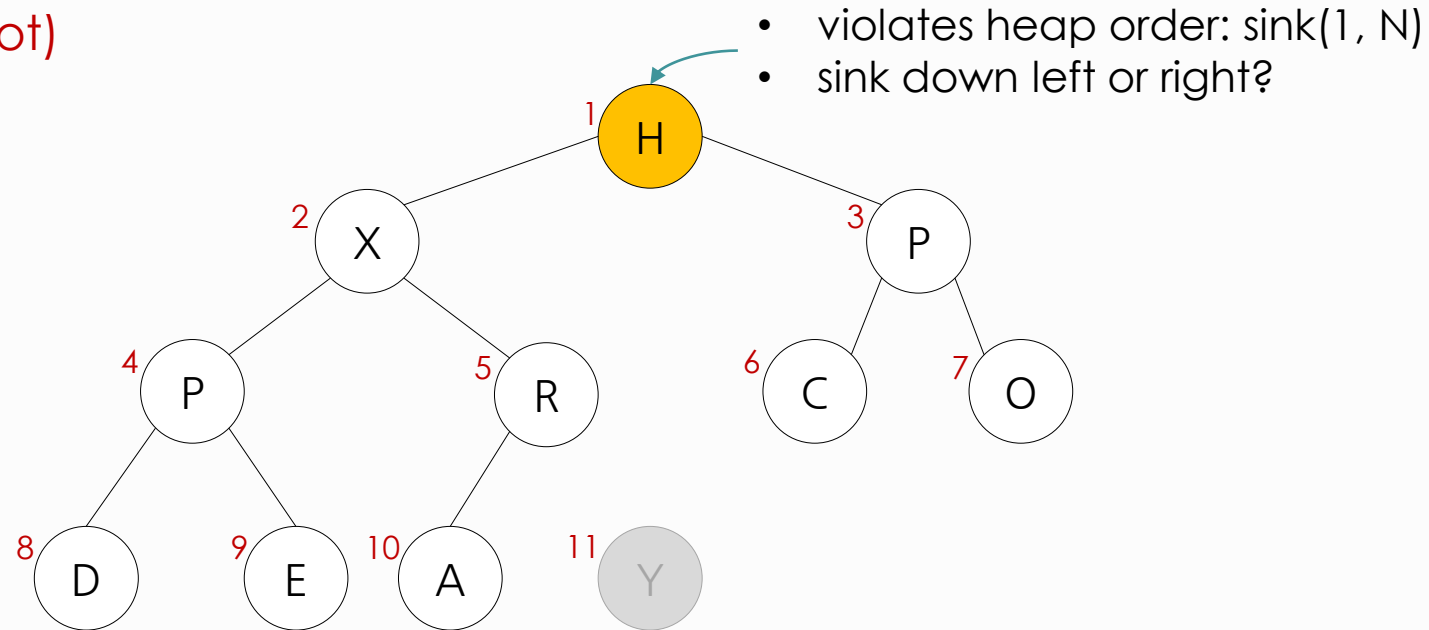
heap-ordered



max-heap example

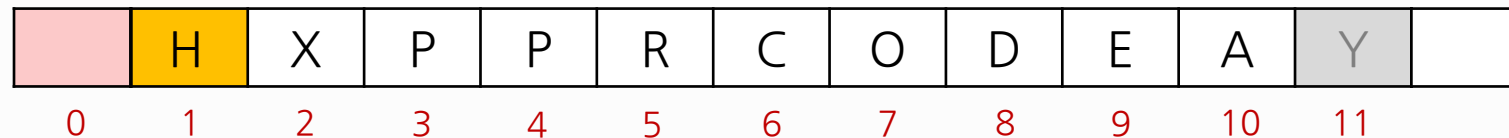
- **Insert:** Add node at end, then swim it up.
- **Remove:** Swap root with last node, then sink down.

remove the max (root)



swap(h, 1, h->N--)

heap-ordered

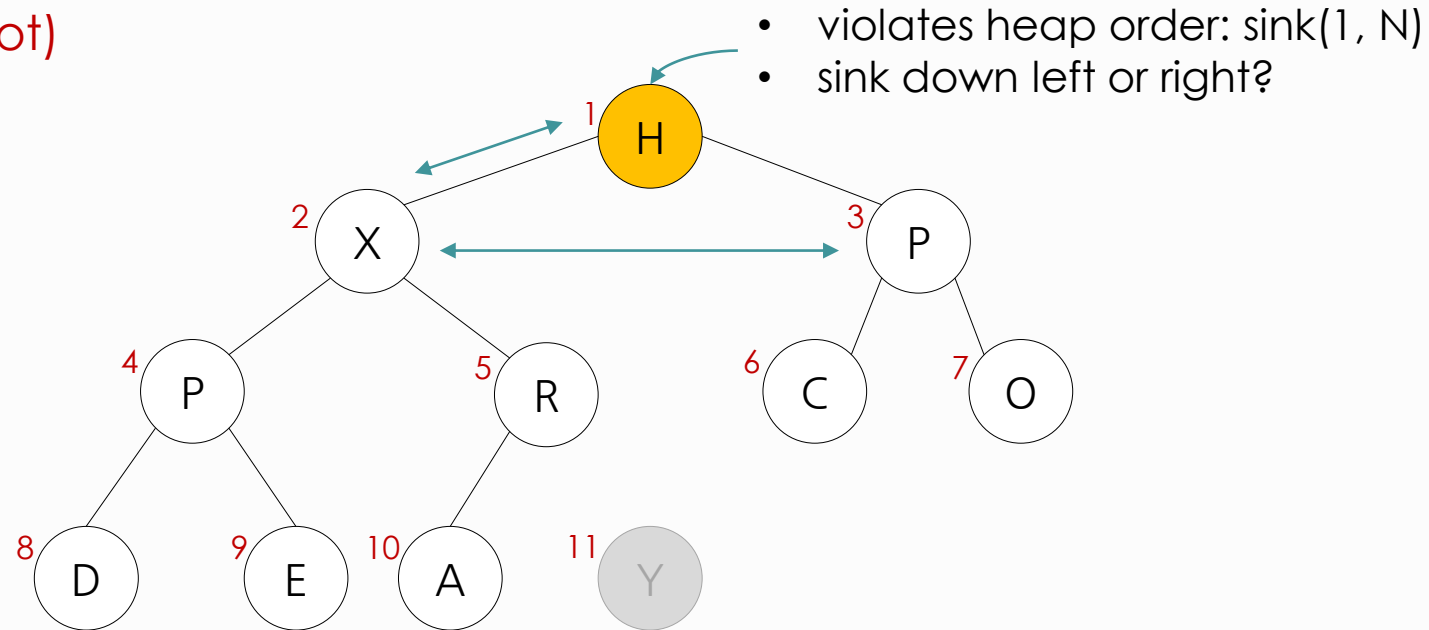


h->N heap size decreased by one

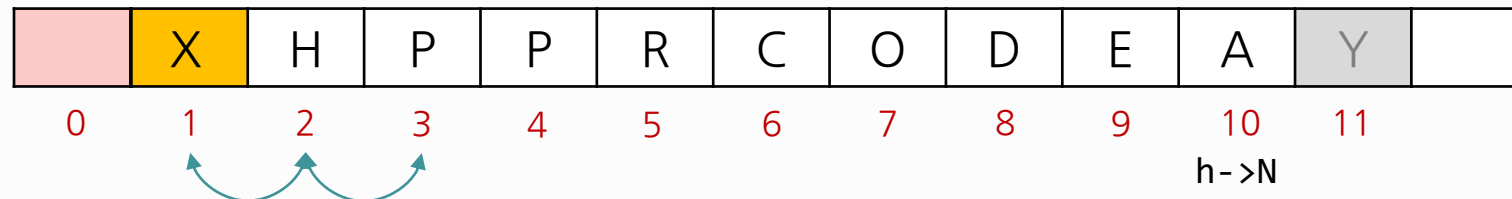
max-heap example

- **Insert:** Add node at end, then swim it up.
- **Remove:** Swap root with last node, then sink down.

remove the max (root)



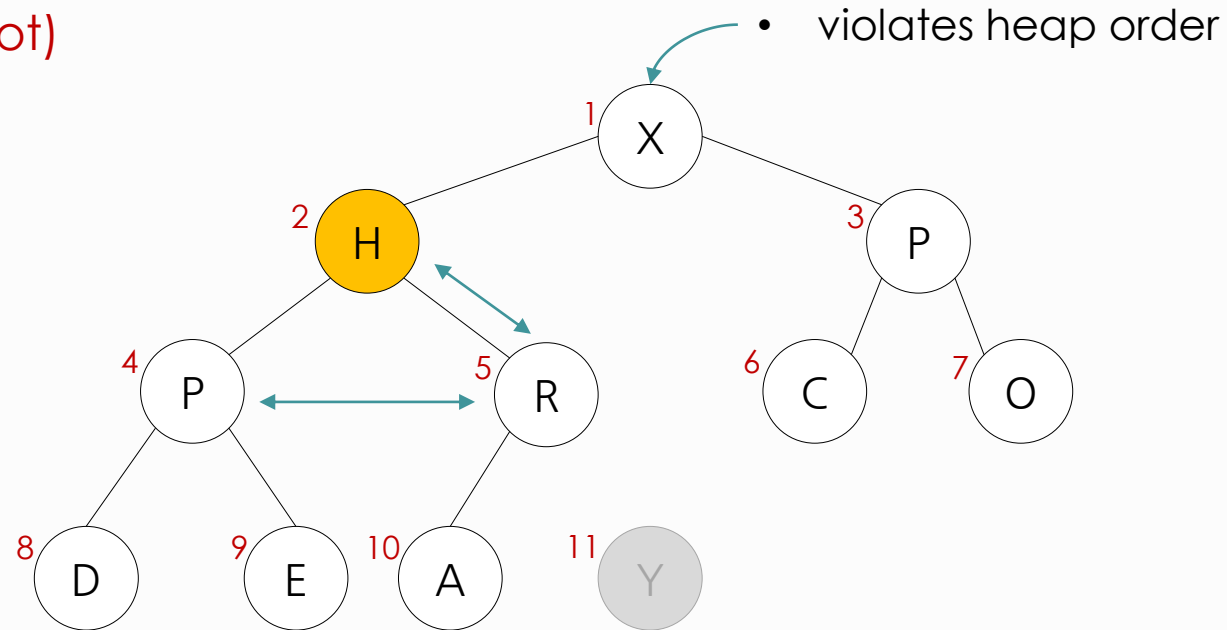
heap-ordered



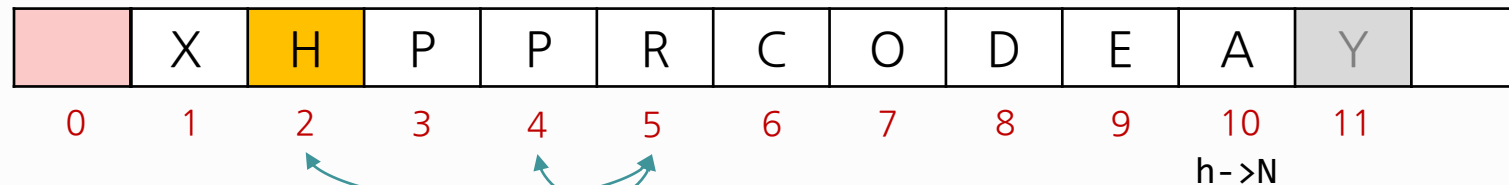
max-heap example

- **Insert:** Add node at end, then swim it up.
- **Remove:** Swap root with last node, then sink down.

remove the max (root)



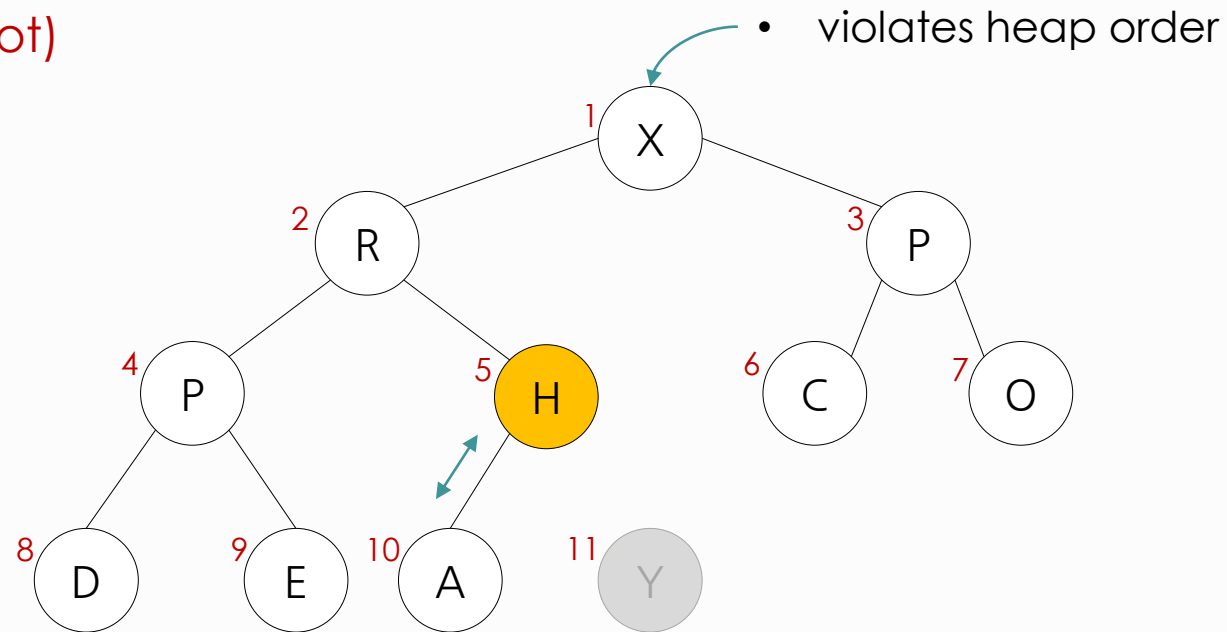
heap-ordered



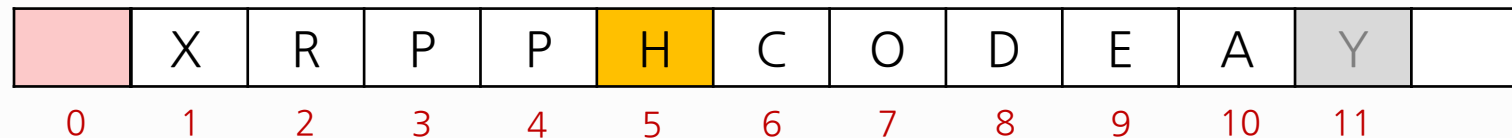
max-heap example

- **Insert:** Add node at end, then swim it up.
- **Remove:** Swap root with last node, then sink down.

remove the max (root)



heap-ordered

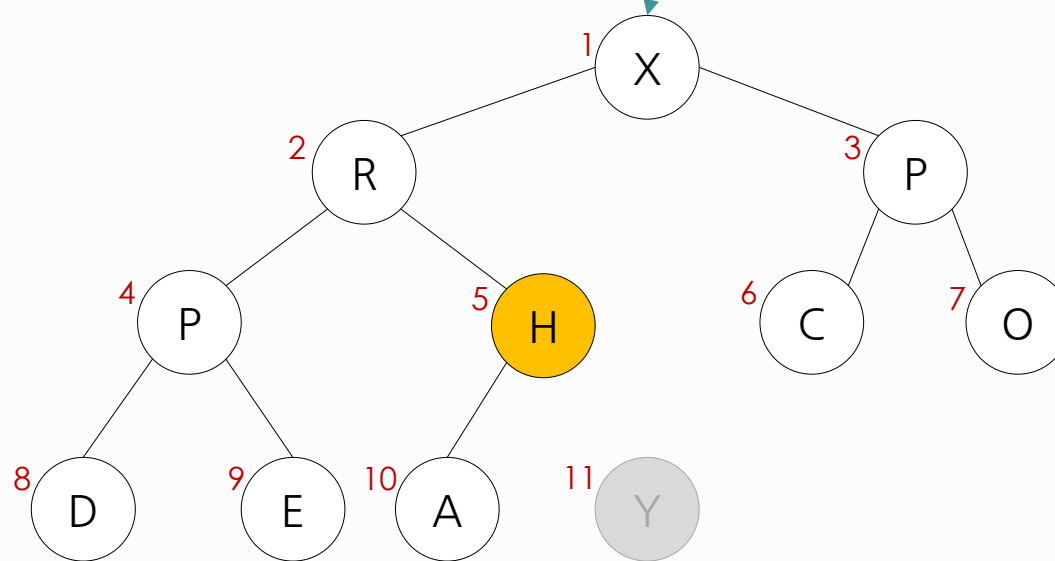


max-heap example

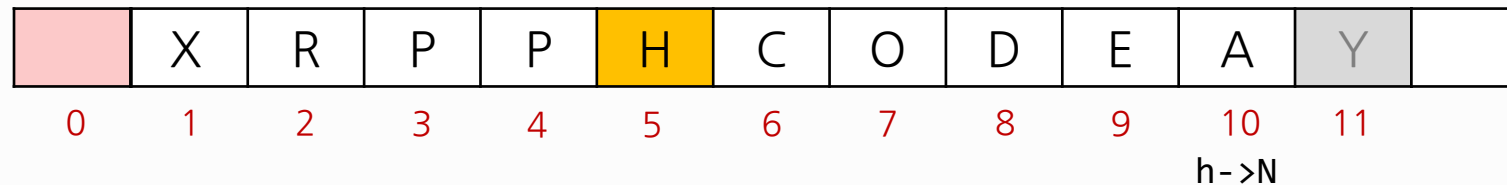
- **Insert:** Add node at end, then swim it up.
- **Remove:** Swap root with last node, then sink down.

remove the max (root)

• heap-ordered



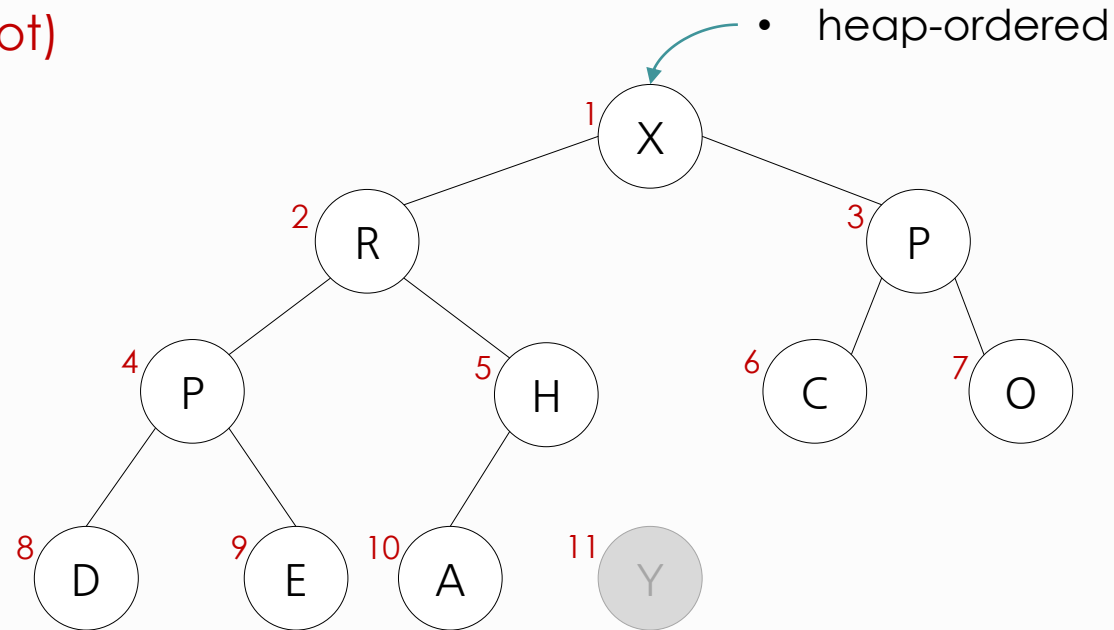
heap-ordered



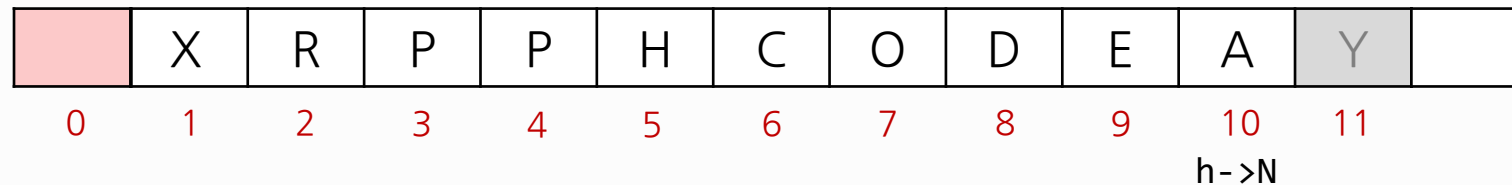
max-heap example

- **Insert:** Add node at end, then swim it up.
- **Remove:** Swap root with last node, then sink down.

remove the max (root)



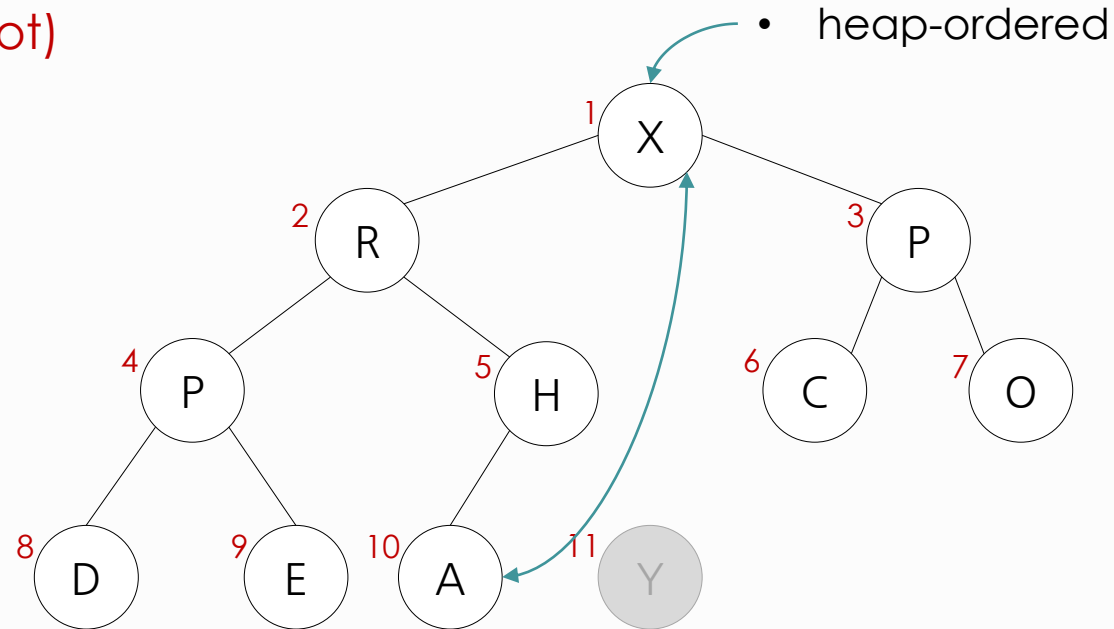
heap-ordered



max-heap example

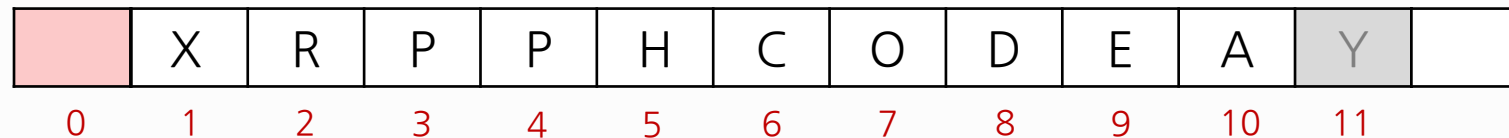
- **Insert:** Add node at end, then swim it up.
- **Remove:** Swap root with last node, then sink down.

remove the max (root)



swap(h, 1, h->N--)

heap-ordered

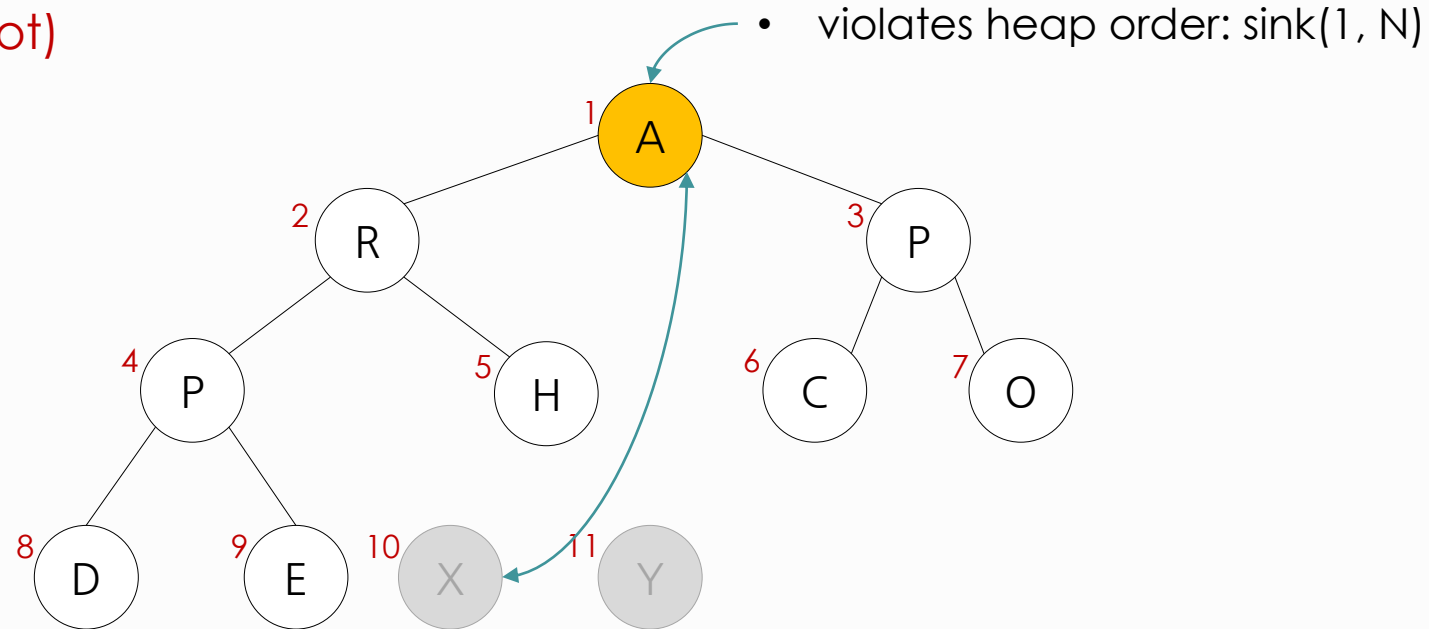


h->N heap size decreased by one

max-heap example

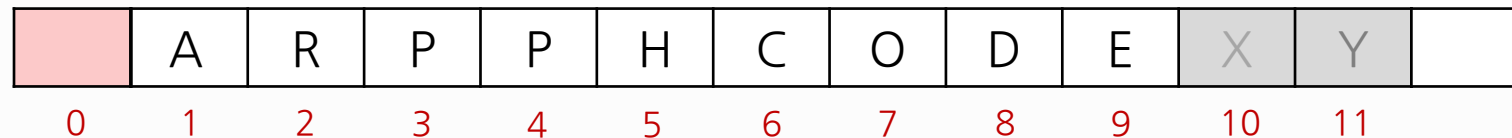
- **Insert:** Add node at end, then swim it up.
- **Remove:** Swap root with last node, then sink down.

remove the max (root)



swap(h, 1, h->N--)

heap-ordered

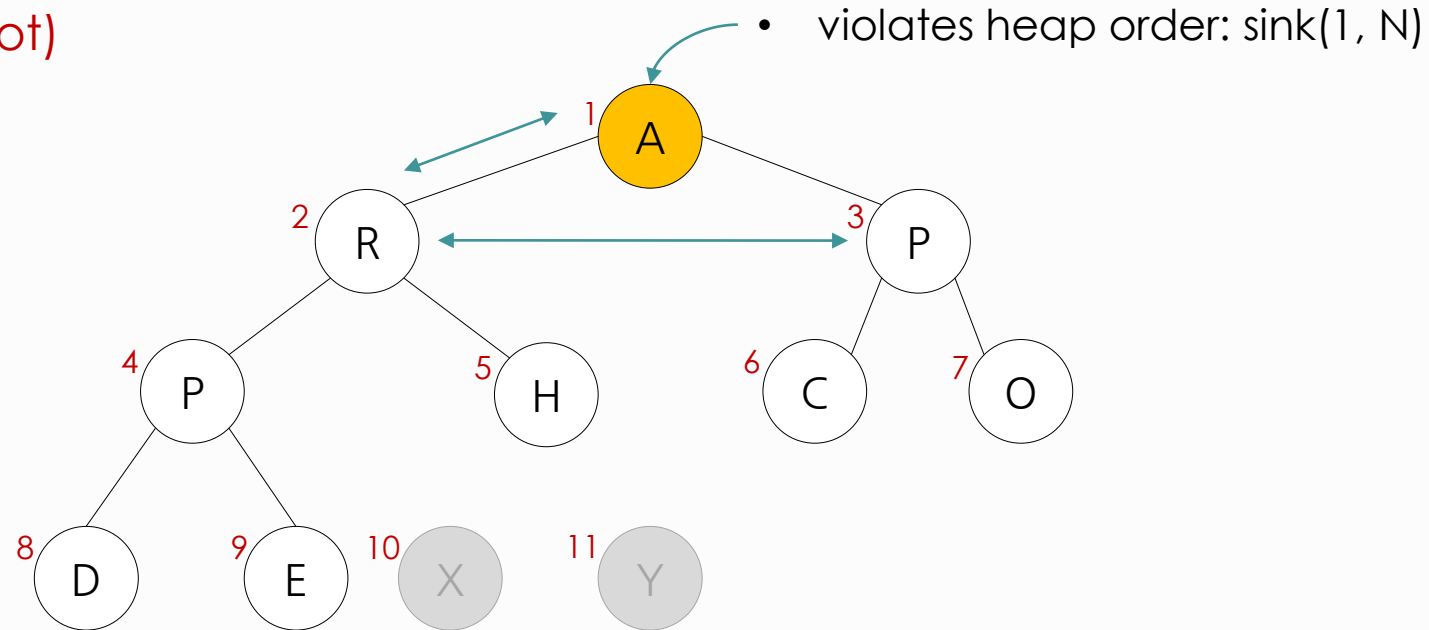


h->N heap size decreased by one

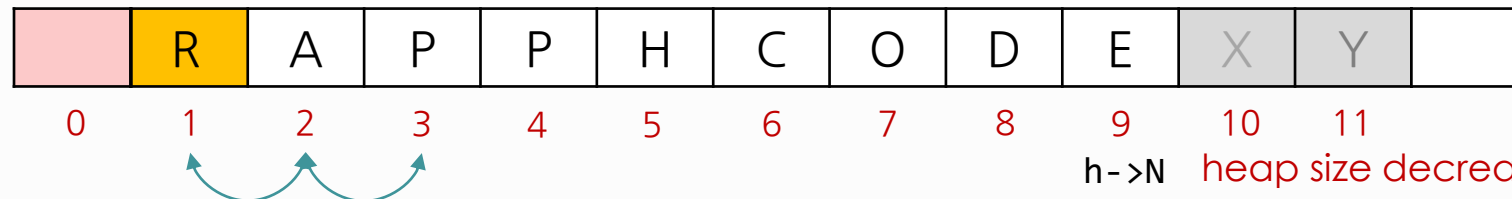
max-heap example

- **Insert:** Add node at end, then swim it up.
- **Remove:** Swap root with last node, then sink down.

remove the max (root)



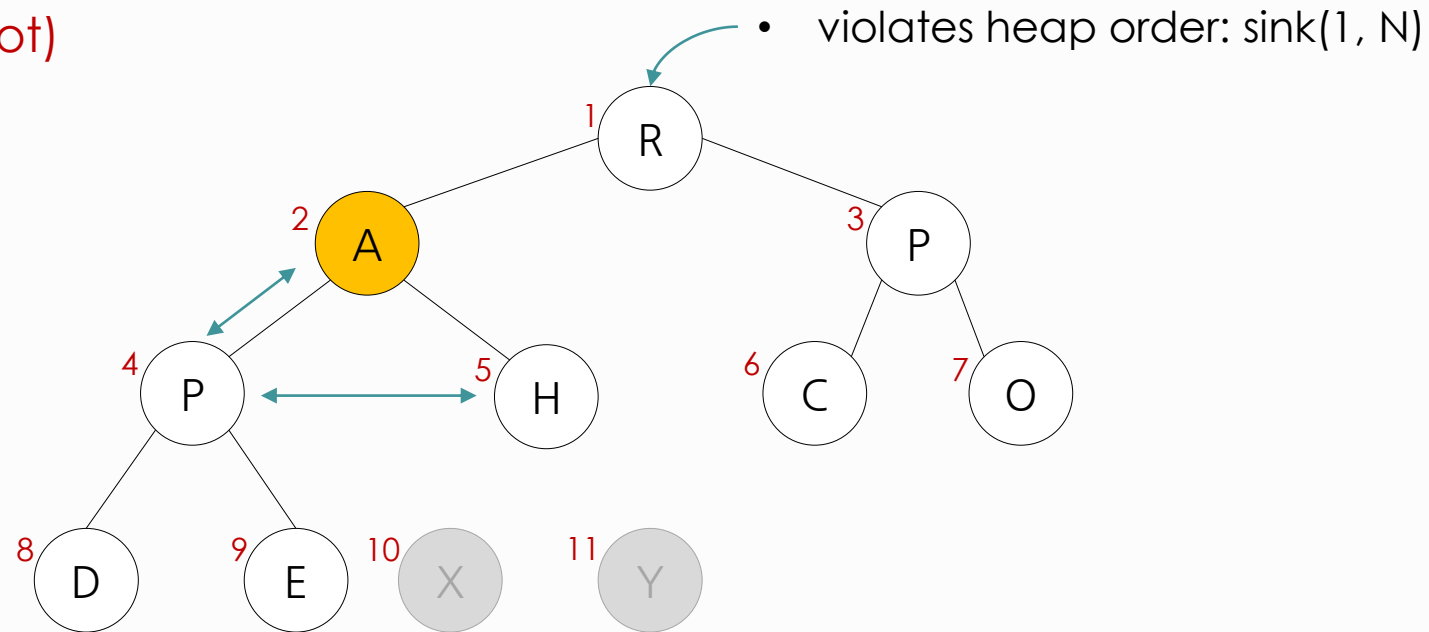
heap-ordered



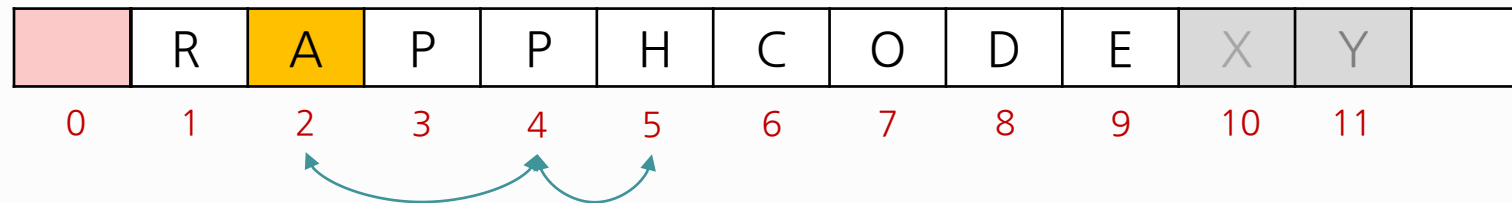
max-heap example

- **Insert:** Add node at end, then swim it up.
- **Remove:** Swap root with last node, then sink down.

remove the max (root)



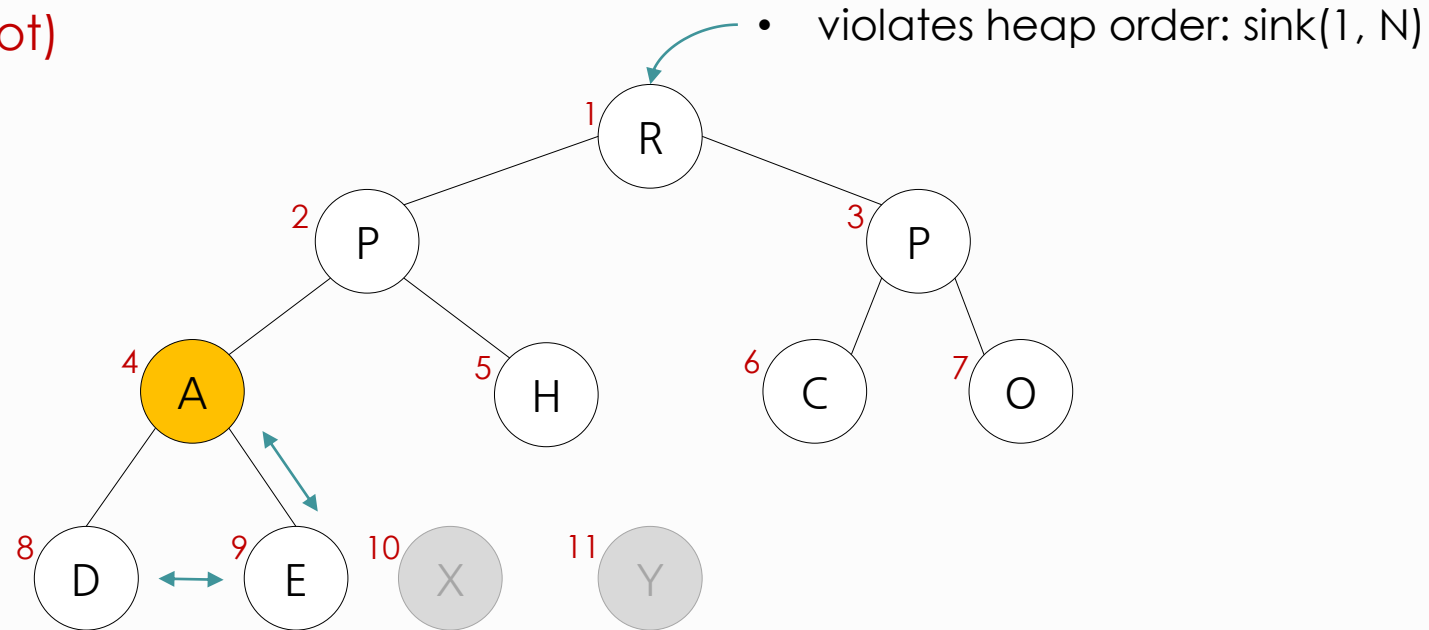
heap-ordered



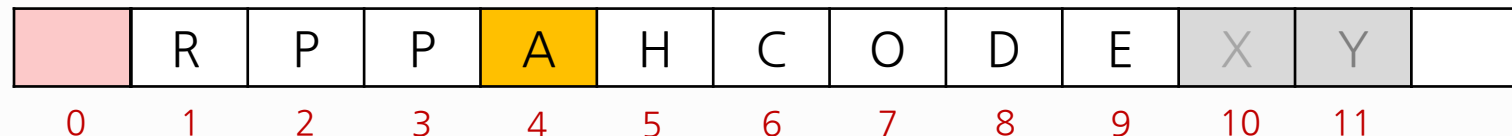
max-heap example

- **Insert:** Add node at end, then swim it up.
- **Remove:** Swap root with last node, then sink down.

remove the max (root)



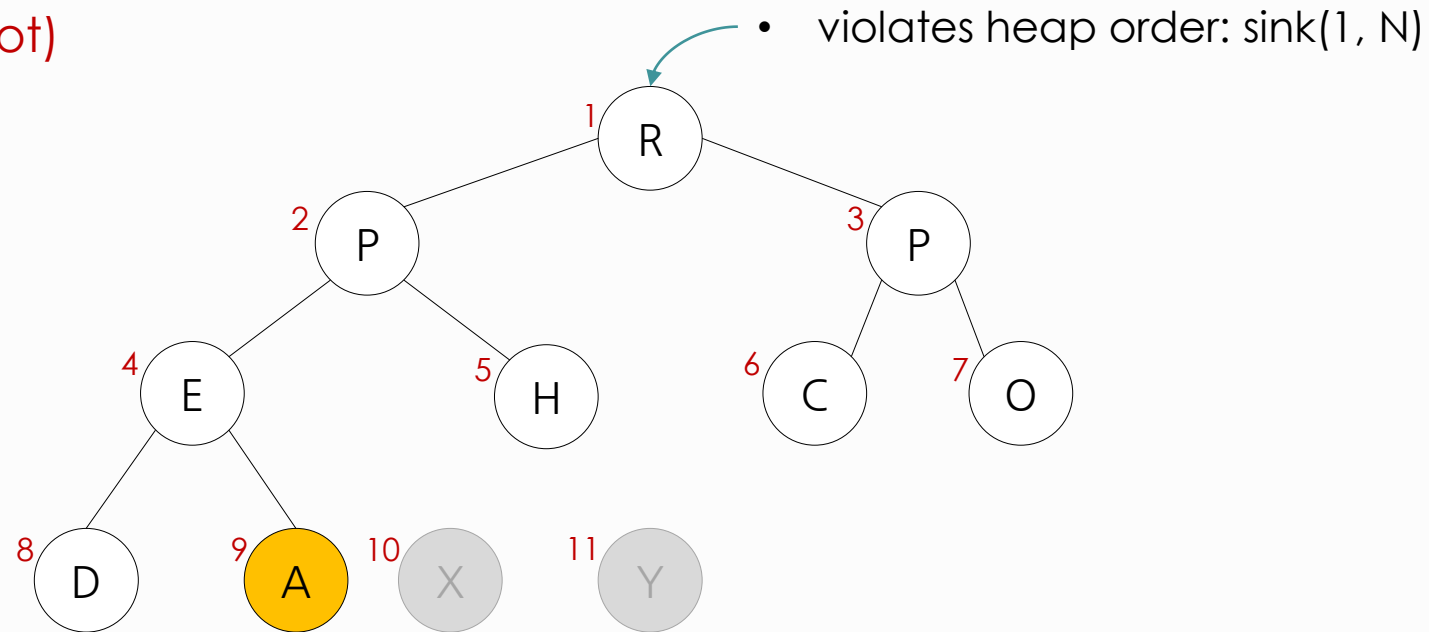
heap-ordered



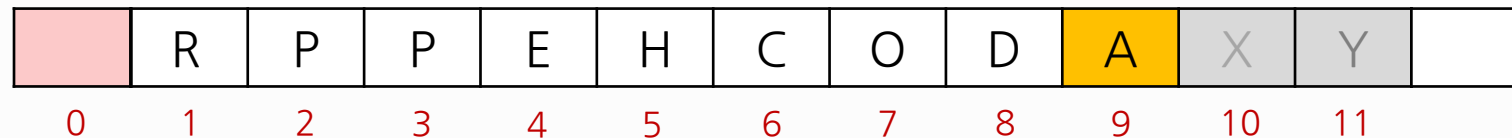
max-heap example

- **Insert:** Add node at end, then swim it up.
- **Remove:** Swap root with last node, then sink down.

remove the max (root)



heap-ordered

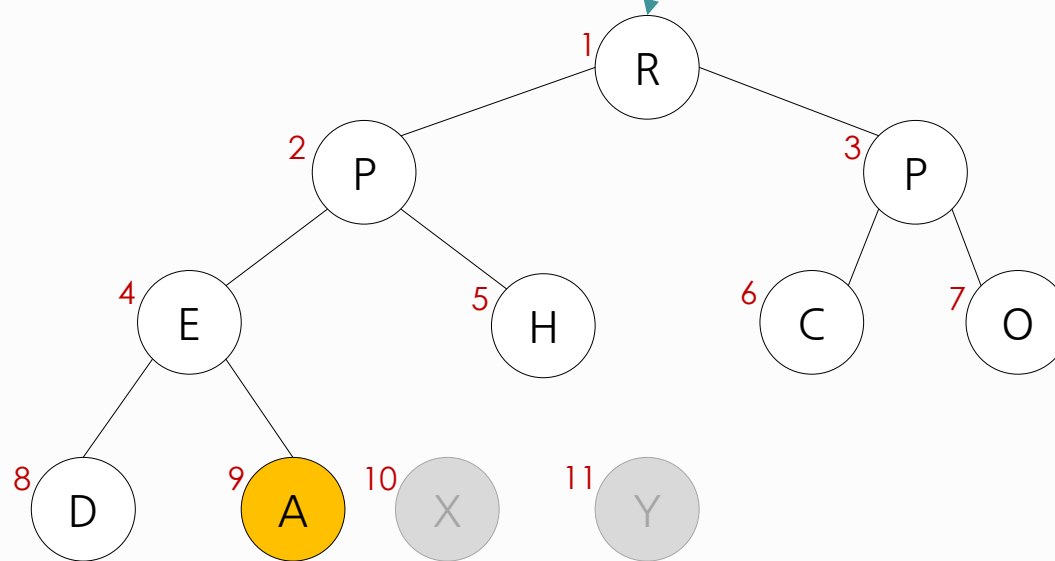


max-heap example

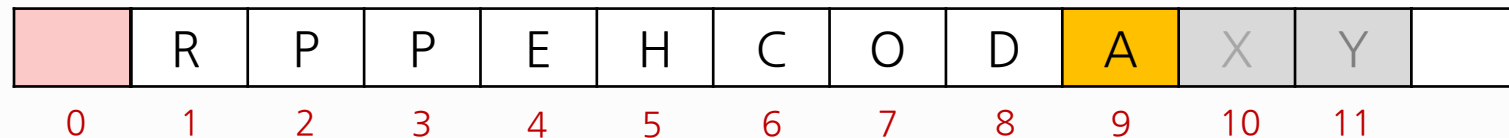
- **Insert:** Add node at end, then swim it up.
- **Remove:** Swap root with last node, then sink down.

remove the max (root)

• heap-ordered



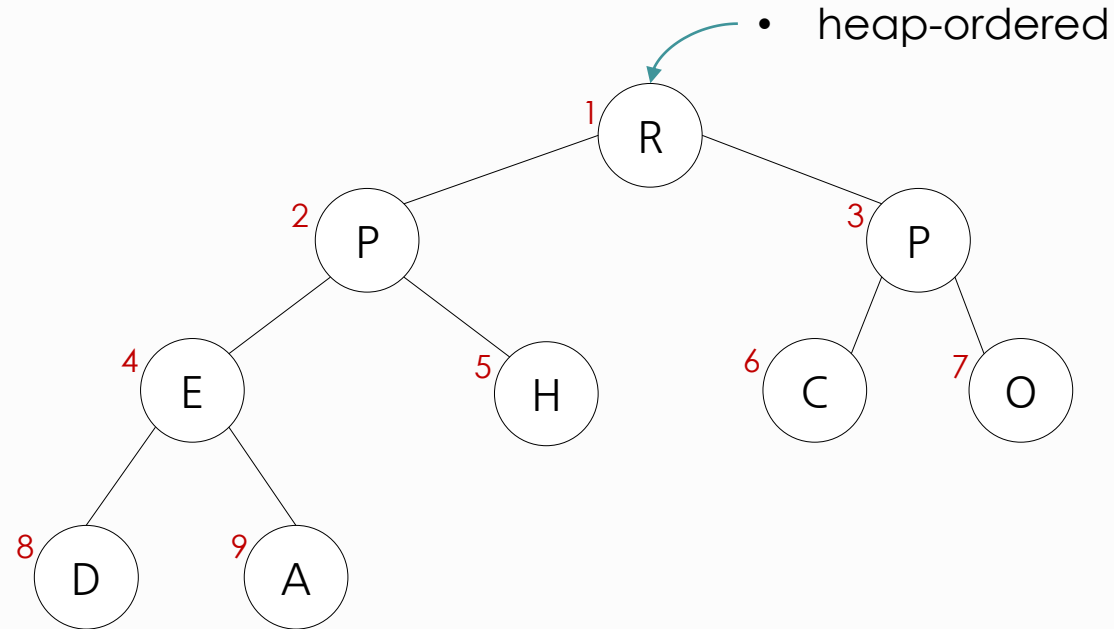
heap-ordered



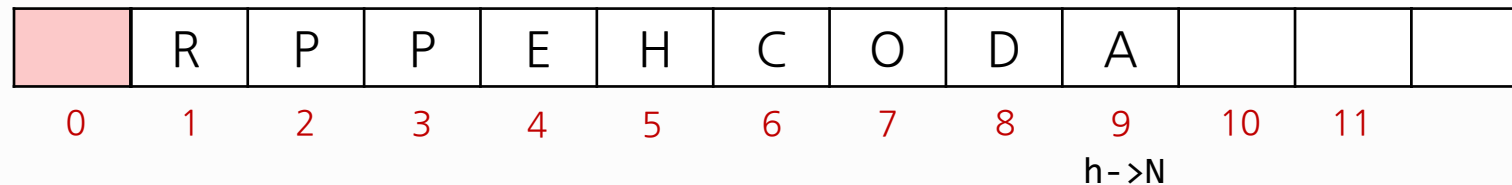
max-heap example

- **Insert:** Add node at end, then swim it up.
- **Remove:** Swap root with last node, then sink down.

insert 



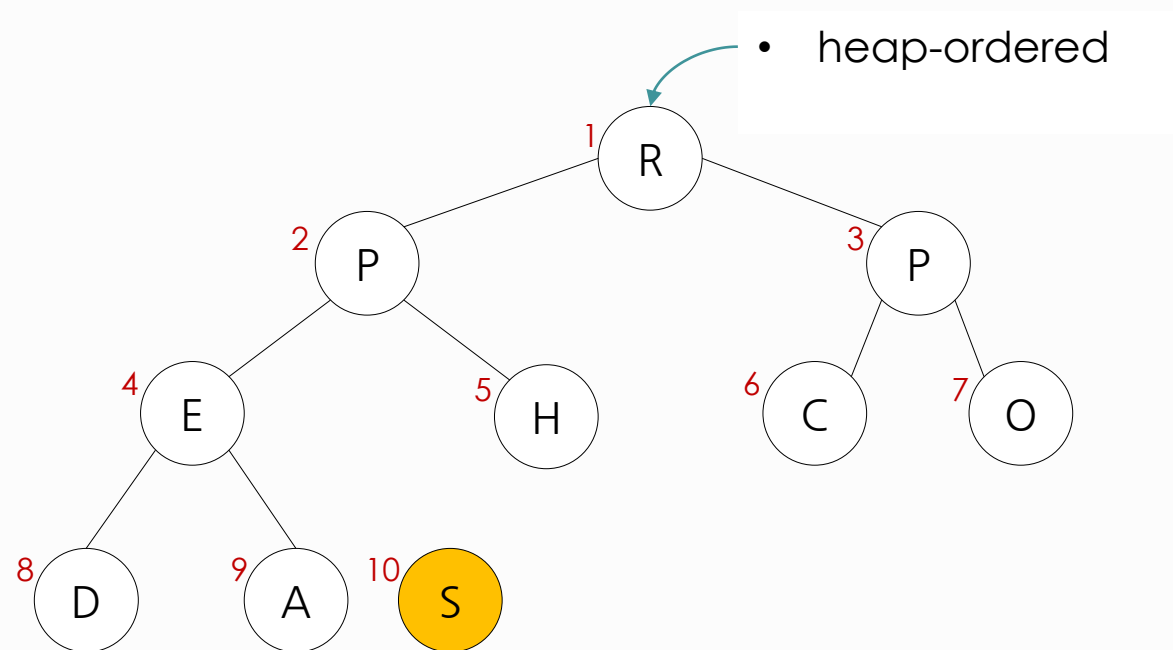
heap-ordered



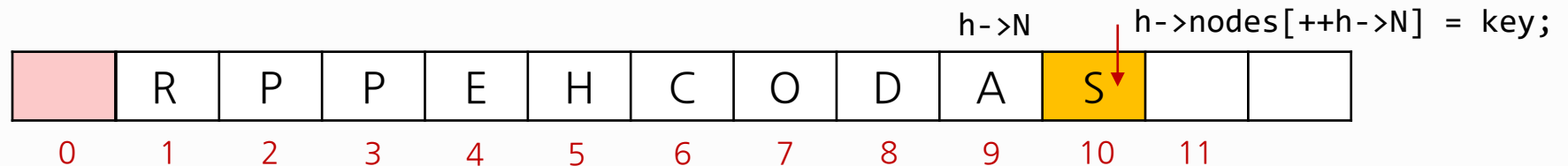
max-heap example

- **Insert:** Add node at end, then swim it up.
- **Remove:** Swap root with last node, then sink down.

insert 



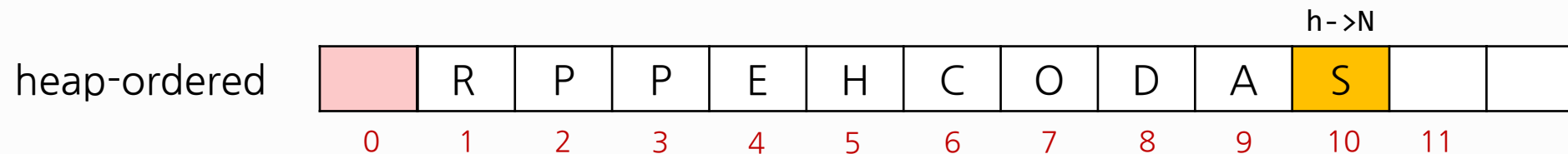
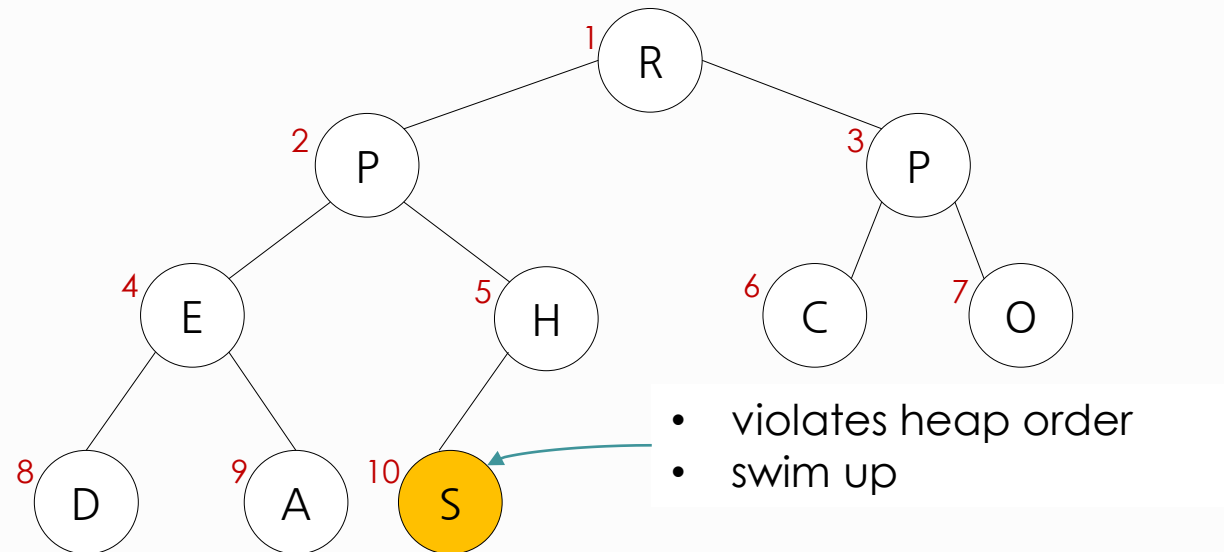
heap-ordered



max-heap example

- **Insert:** Add node at end, then swim it up.
- **Remove:** Swap root with last node, then sink down.

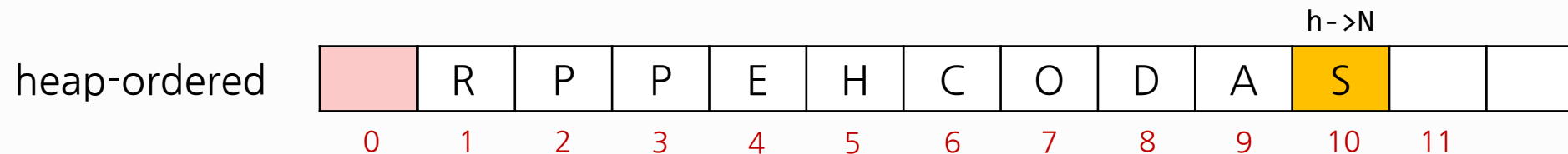
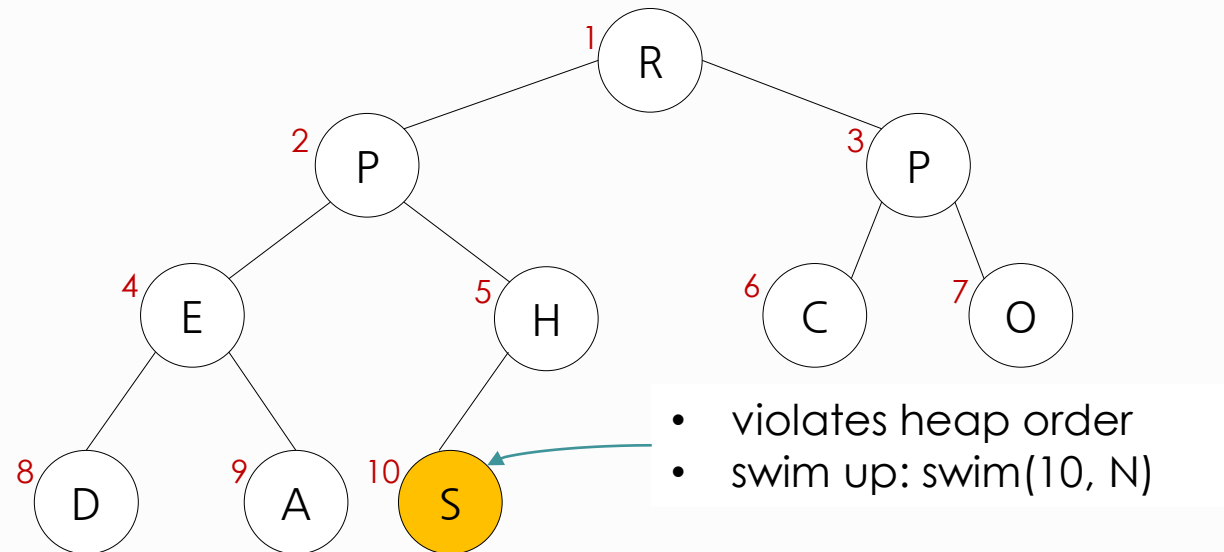
insert 



max-heap example

- **Insert:** Add node at end, then swim it up.
- **Remove:** Swap root with last node, then sink down.

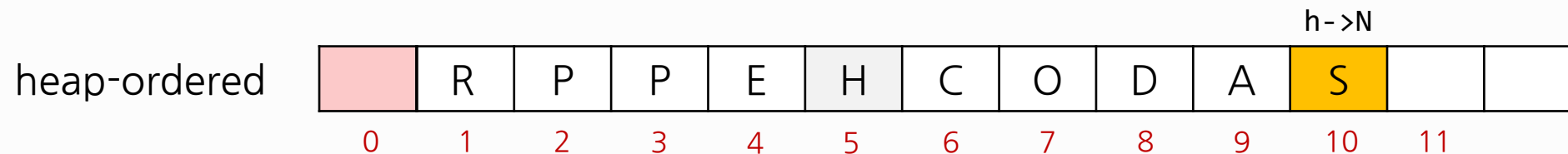
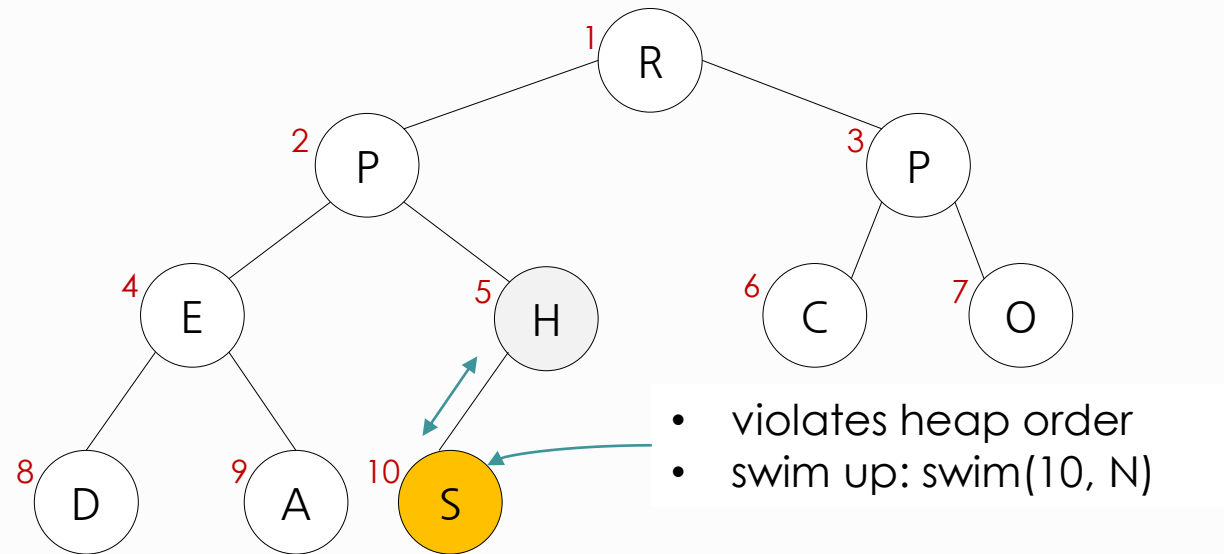
insert 



max-heap example

- **Insert:** Add node at end, then swim it up.
- **Remove:** Swap root with last node, then sink down.

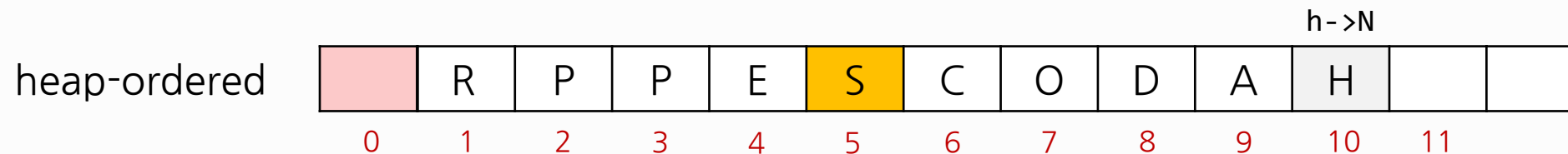
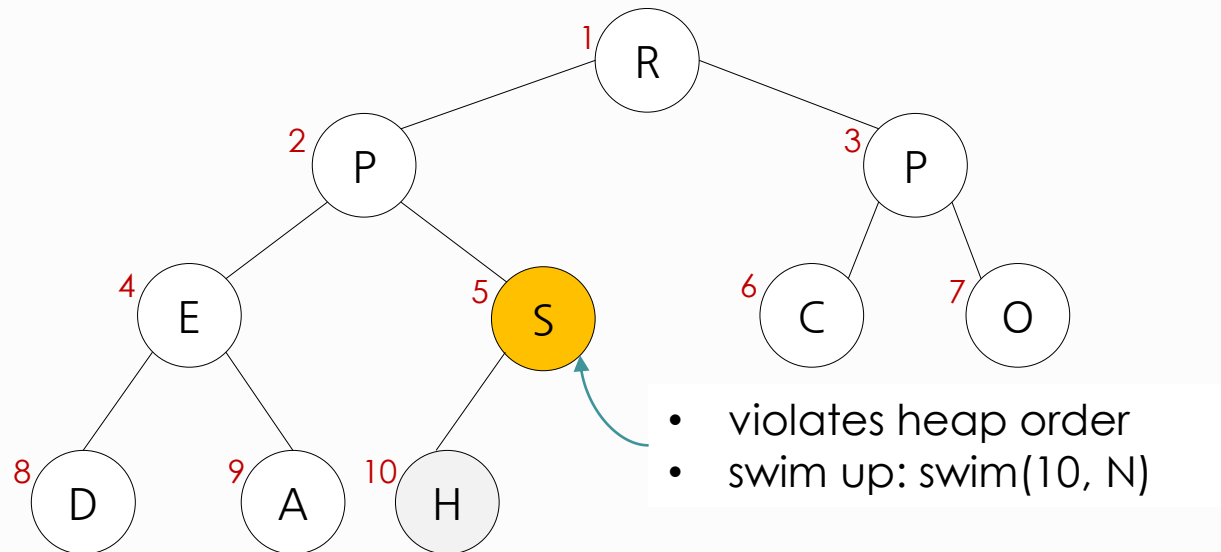
insert 



max-heap example

- **Insert:** Add node at end, then swim it up.
- **Remove:** Swap root with last node, then sink down.

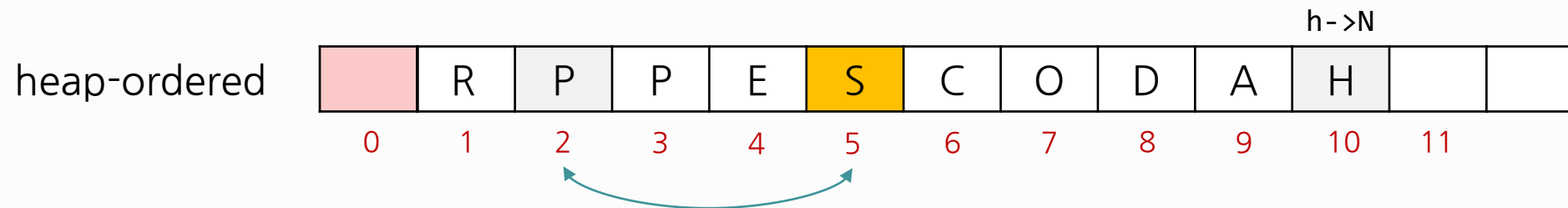
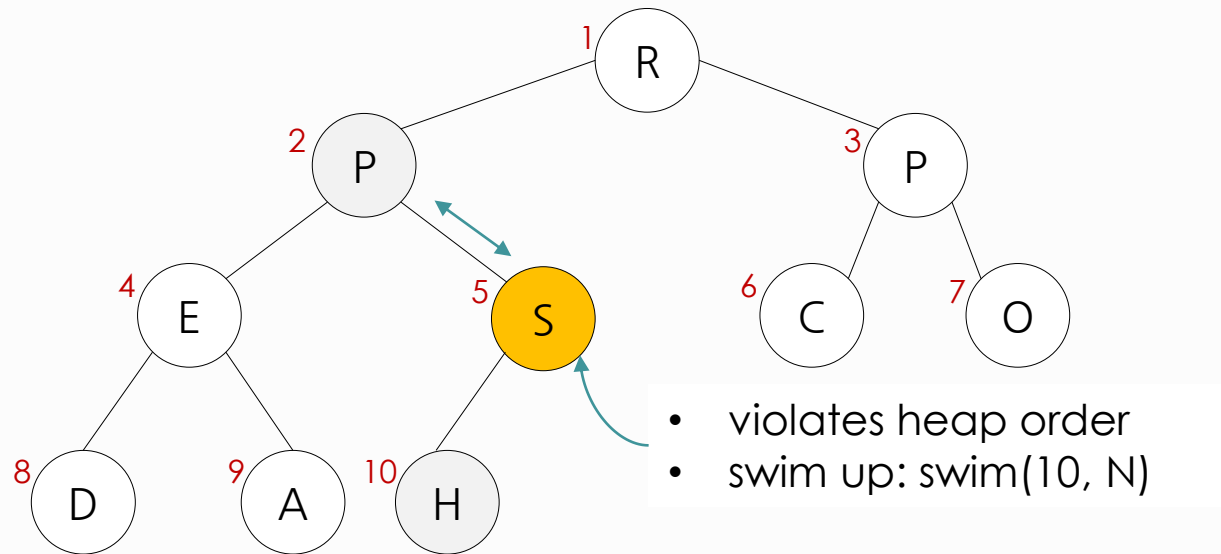
insert 



max-heap example

- **Insert:** Add node at end, then swim it up.
- **Remove:** Swap root with last node, then sink down.

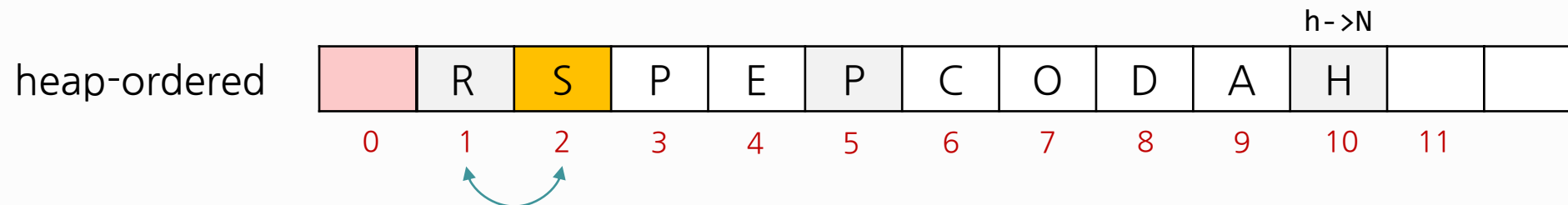
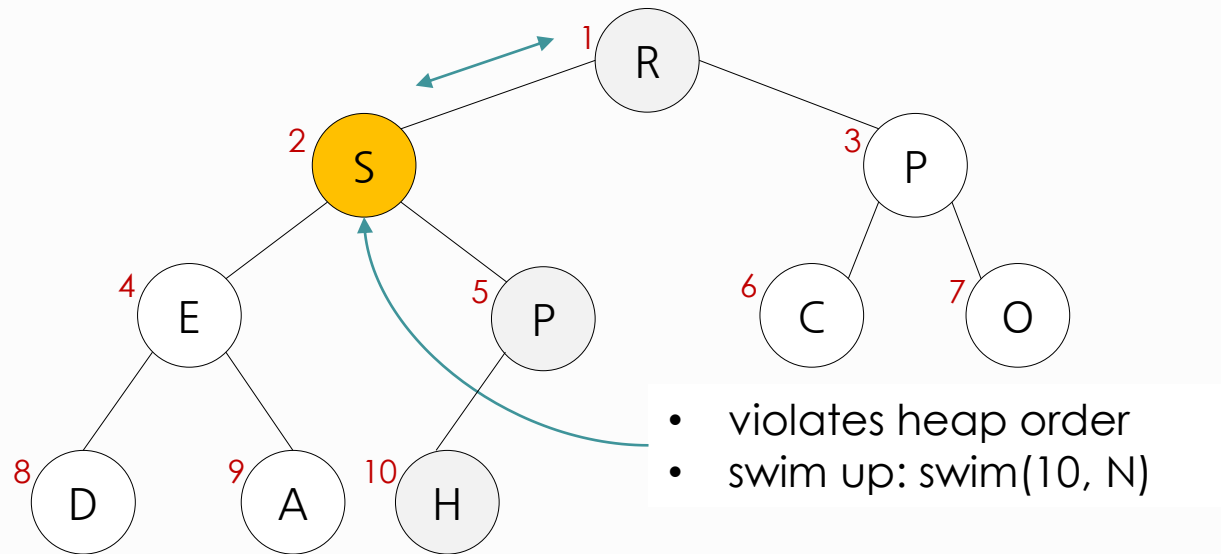
insert 



max-heap example

- **Insert:** Add node at end, then swim it up.
- **Remove:** Swap root with last node, then sink down.

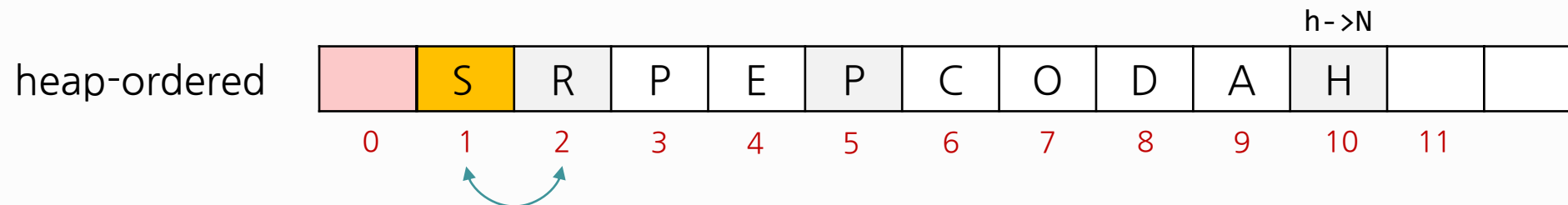
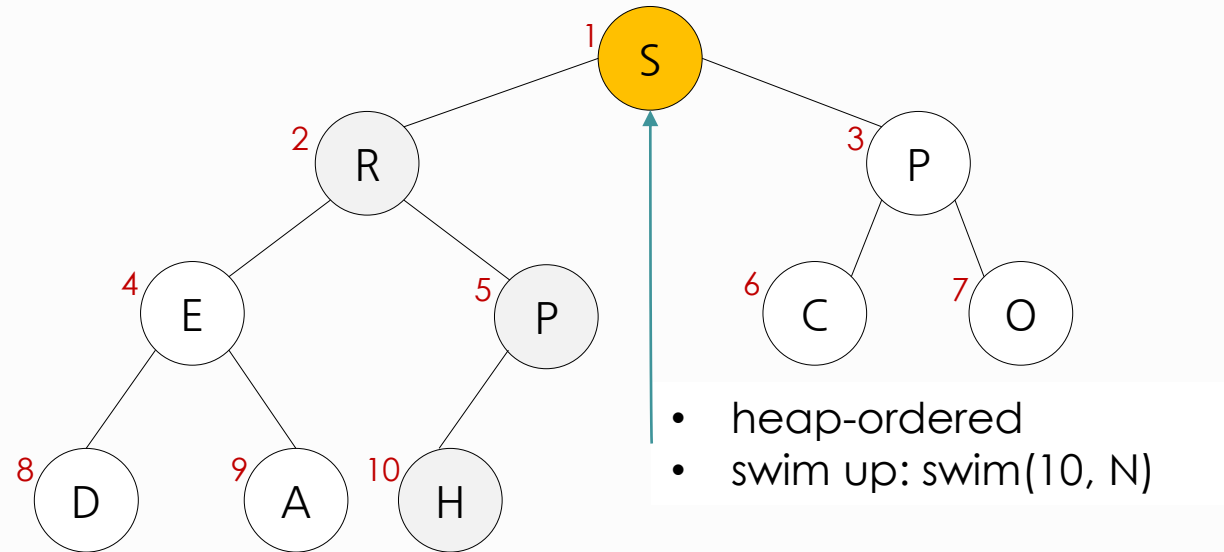
insert 



max-heap example

- **Insert:** Add node at end, then swim it up.
- **Remove:** Swap root with last node, then sink down.

insert 



Binary heap operations time complexity with N items:

- Level of heap is $\lfloor \log_2 N \rfloor$
- insert: $O(\log N)$ for each insert
 - In practice, expect less
- delete: $O(\log N)$ // deleting root node or any node
- increase/decrease key: $O(\log N)$
- **Heapify():** $O(N)$
- **Heapsort():** $O(N \log N)$
- Because $O(N)$ heapify + $O(\log N)$ delete = $O(N \log N)$
- **Proof:**
 - <https://stackoverflow.com/questions/9755721/how-can-building-a-heap-be-on-time-complexity>
 - <https://www.insertingwiththeweb.com/data-structures/binary-heap/build-heap-proof/>
 - <https://www.quora.com/How-is-the-time-complexity-of-building-a-heap-is-o-n>
- **References in Korean:**
 - <https://ratsgo.github.io/data%20structure&algorithm/2017/09/27/heapsort/>
 - <https://zeddios.tistory.com/56>