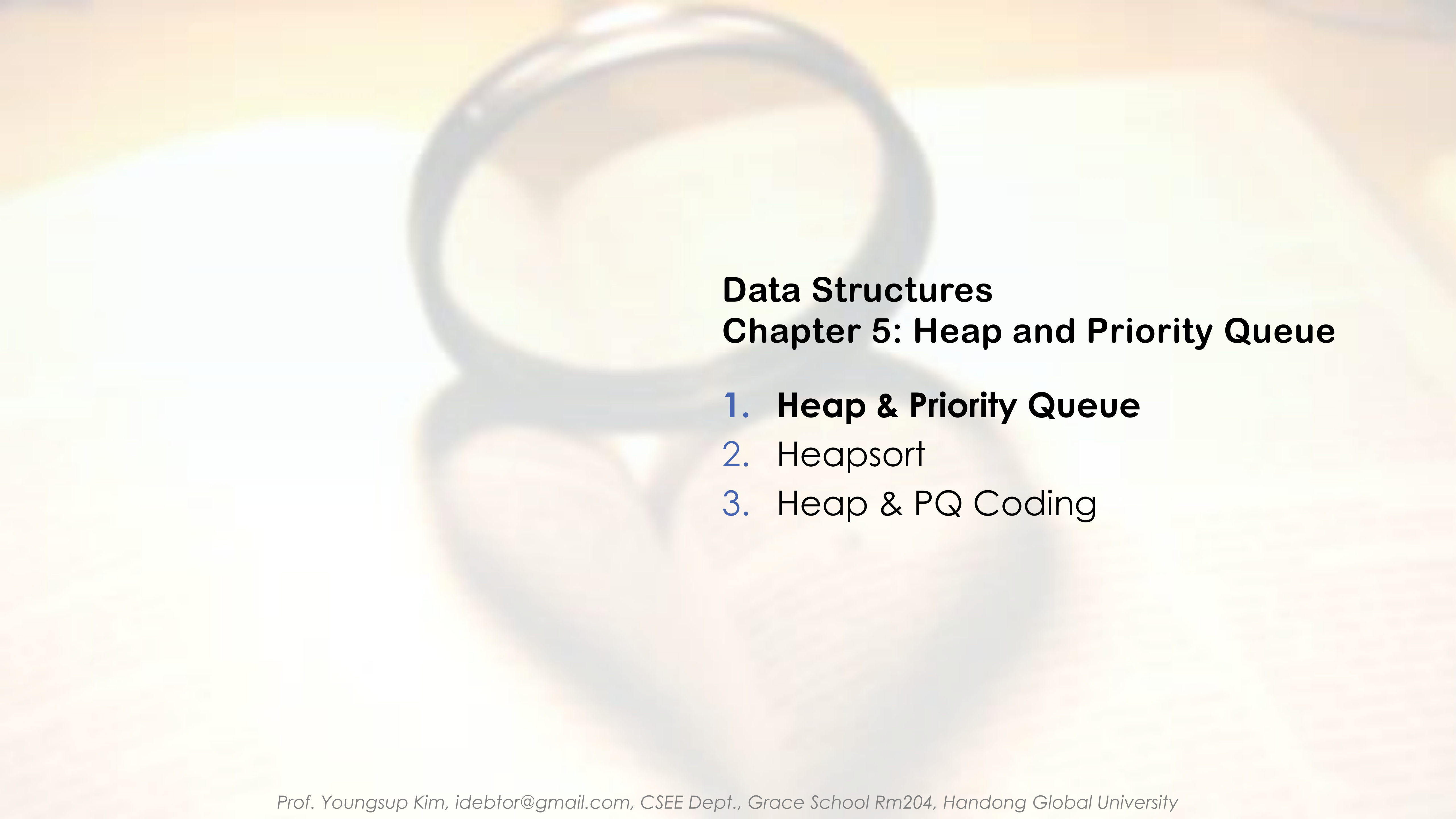
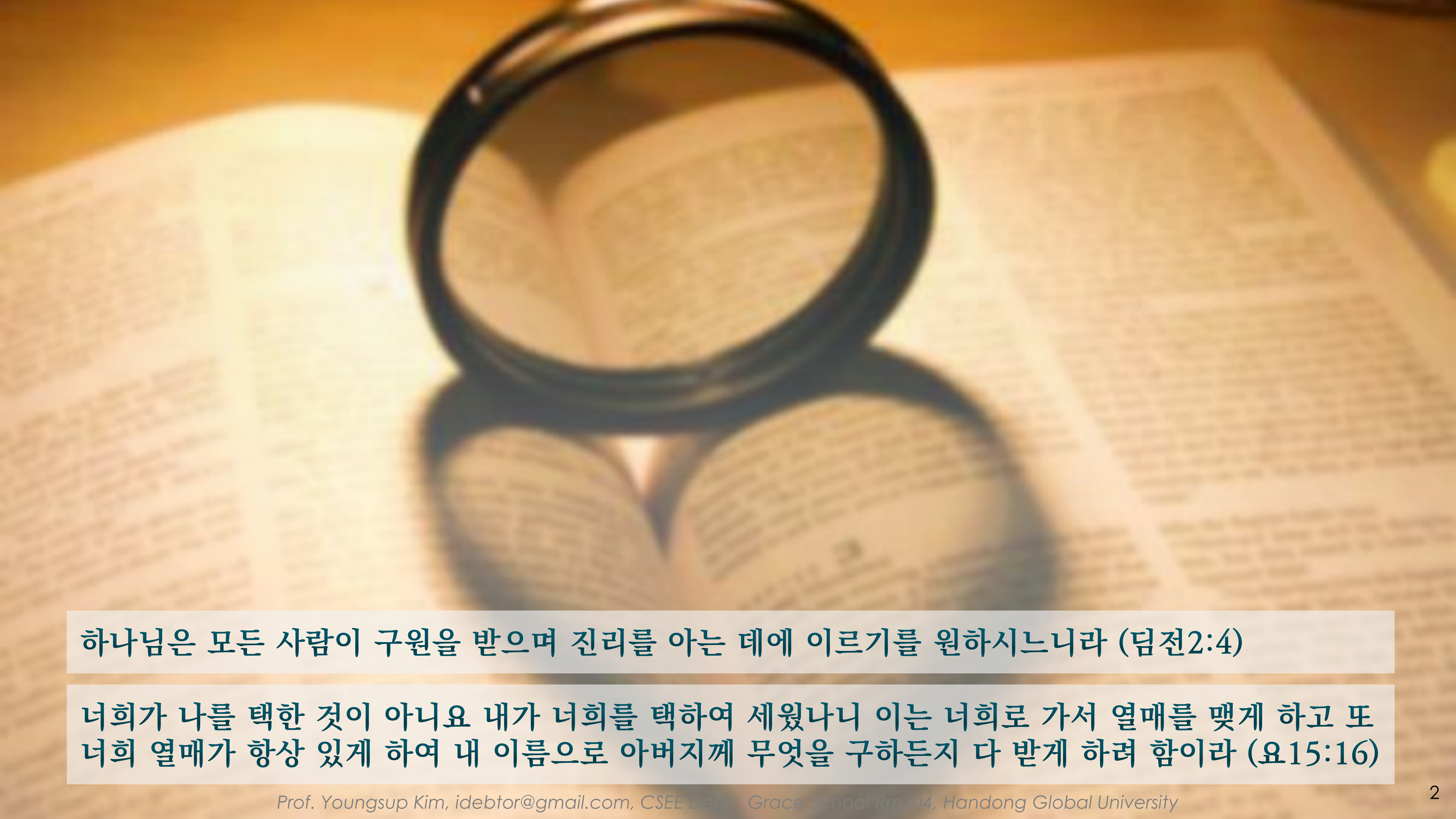


Data Structures
Chapter 5: Heap and Priority Queue

1. **Heap & Priority Queue**
2. Heapsort
3. Heap & PQ Coding

Prof. Youngsup Kim, idebtor@gmail.com, CSEE Dept., Grace School Rm204, Handong Global University



하나님은 모든 사람이 구원을 받으며 진리를 아는 데에 이르기를 원하시느니라 (딤전2:4)

너희가 나를 택한 것이 아니요 내가 너희를 택하여 세웠나니 이는 너희로 가서 열매를 맺게 하고 또 너희 열매가 항상 있게 하여 내 이름으로 아버지께 무엇을 구하든지 다 받게 하려 함이라 (요15:16)

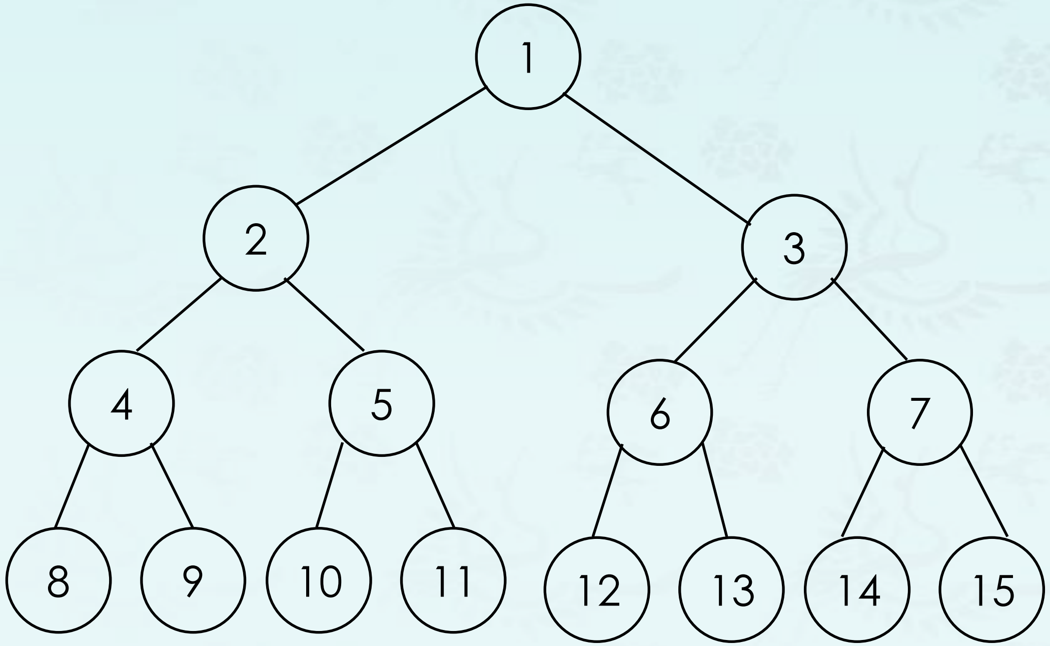
Data Structures
Chapter 5: Heap and Priority Queue

1. **Heap & Priority Queue**
   - **Complete Binary Tee (Review)**
   - **Binary Heap and Priority Queue**
   - Minheap and Maxheap
2. Heapsort
3. Heap & PQ Coding

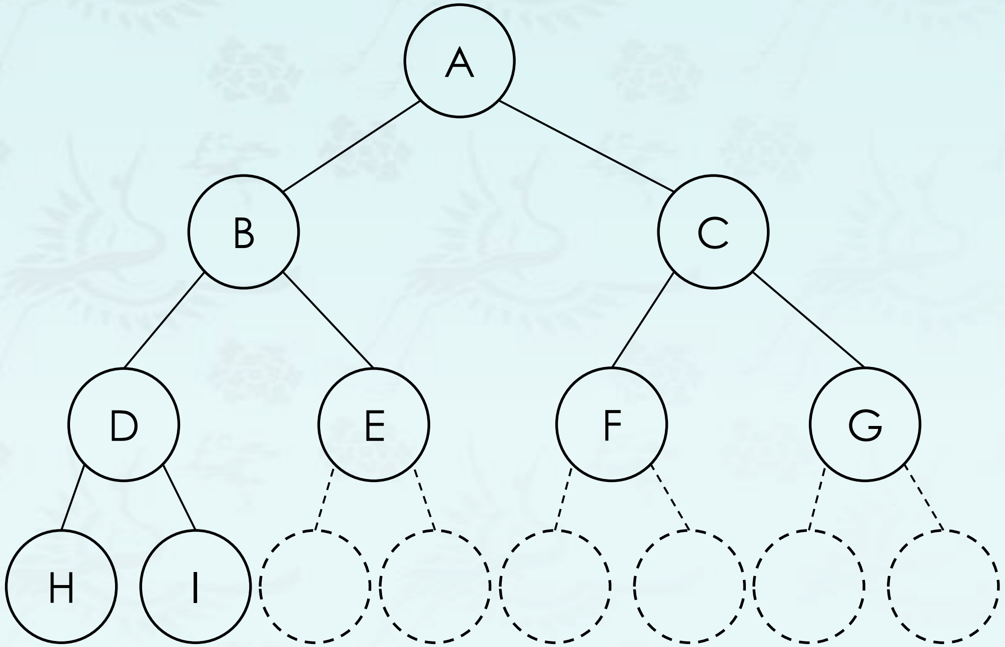
Prof. Youngsup Kim, idebtor@gmail.com, CSEE Dept., Grace School Rm204, Handong Global University

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** iff its nodes correspond to the nodes numbered from 1 to n in the full binary tree of level k .



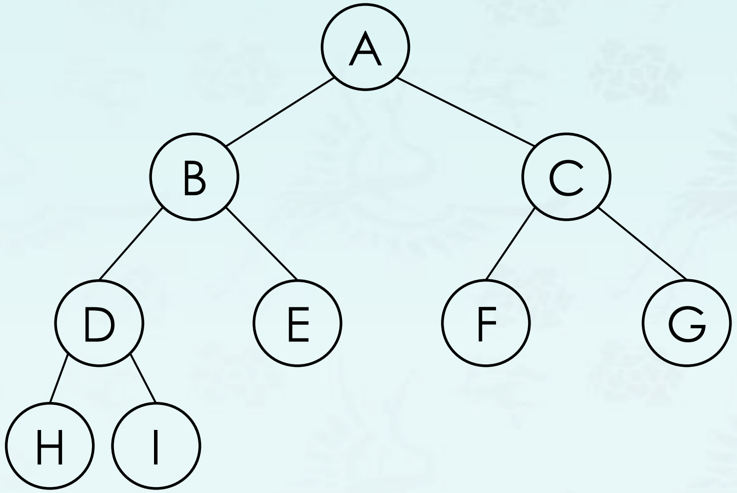
*A **full** binary tree*



*A **complete** binary tree*

Binary trees - Array representation

- **Q:** Let’s suppose that you have a **complete binary tree** in an array. Find its parent, left child and right child at node D.



[0]	-
[1]	A
[2]	B
[3]	C
[4]	D
[5]	E
[6]	F
[7]	G
[8]	H
[9]	I

Solution:

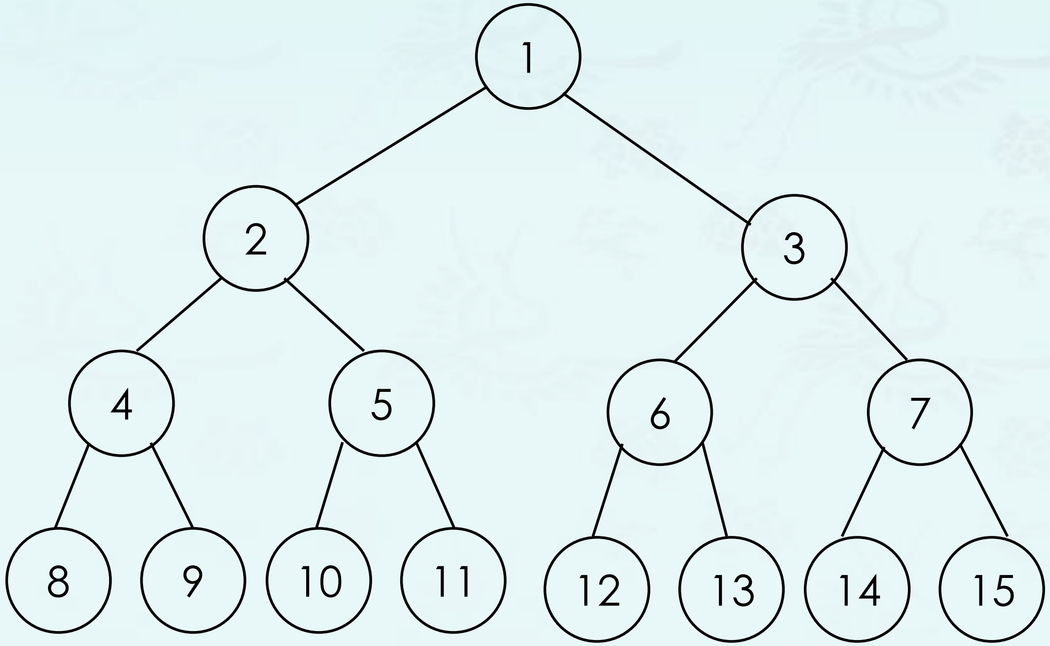
$parent(x = 4)$ is at 4/2 = 2

$leftChild(4)$ is at 2x4 = 8

$rightChild(4)$ is at 2x4 + 1 = 9

Binary trees - Array representation

- **Q:** Let's suppose that you have a **complete binary tree** in an array, how can we locate node x's parent or child?
- A **complete** binary tree with n nodes, any node index i, $1 \le i \le n$, we have
  - `parent(i) is at` $\lfloor i/2 \rfloor$ `If i = 1, i is at the root and has no parent`
  - `leftChild(i) is at 2i  if 2i <= n. If 2i > n, then i has no left child.`
  - `rightChild(i) is at 2i+1 if 2i+1 <= n. If 2i+1 > n,  then i has no right child.`



Wow! Can we use this to all binary trees?
Why not?

Problem remains:
The problem with storing an arbitrary binary tree using an array is the inefficiency *in memory usage*.

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

Priority queue applications

Event-driven simulation.	[customers in a line, colliding particles]
Numerical computation.	[reducing round-off error]
Data compression.	[Huffman codes]
Graph searching.	[Dijkstra's algorithm, Prim's algorithm]
Number theory.	[sum of powers]
Artificial intelligence.	[A* search]
Statistics.	[maintain largest M values in a sequence]
Operating systems.	[load balancing, interrupt handling]
Discrete optimization.	[bin packing, scheduling]
Spam filtering.	[Bayesian spam filter]

Heaps & Priority Queues

- Challenge: Find the largest **M** items in a stream of **N** items.
  - Fraud detection: isolate $$ transactions.
  - Hacking: KT's customer DB access by their sales agents
  - File maintenance: find biggest files, directories, or emails.
- Constraints: Not enough memory to store N items.

N huge, M large
N >> M

%more trans.txt

```
Turing       6/17/1990     644.08
vonNeumann  3/26/2002     4121.85
Dijkstra     8/22/2007     2678.40
vonNeumann  1/11/1999     4409.74
Dijkstra     11/18/1995    837.42
Hoare        5/10/1993     3229.27
vonNeumann  2/12/1994     4732.35
Hoare        8/18/1992     4381.21
Turing       1/11/2002     66.10
Thompson     2/27/2000     4747.08
Turing       2/11/1991     2156.86
Hoare        8/12/2003     1025.70
vonNeumann  10/13/1993    2520.97
Dijkstra     9/10/2000     708.95
Turing       10/12/1993    3532.36
Hoare        2/10/2005     4050.20
```

%java TopM 5 < trans.txt

```
Thompson    2/27/2000    4747.08
vonNeumann 2/12/1994    4732.35
vonNeumann 1/11/1999    4409.74
Hoare        8/18/1992    4381.21
vonNeumann 3/26/2002    4121.85
```

Sort key

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
N >> M

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

implementation	time	space
sort	N log N	N
binary heap	N log M	M
best in theory	N	M

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 growth of finding the largest M **in a stream of N items**

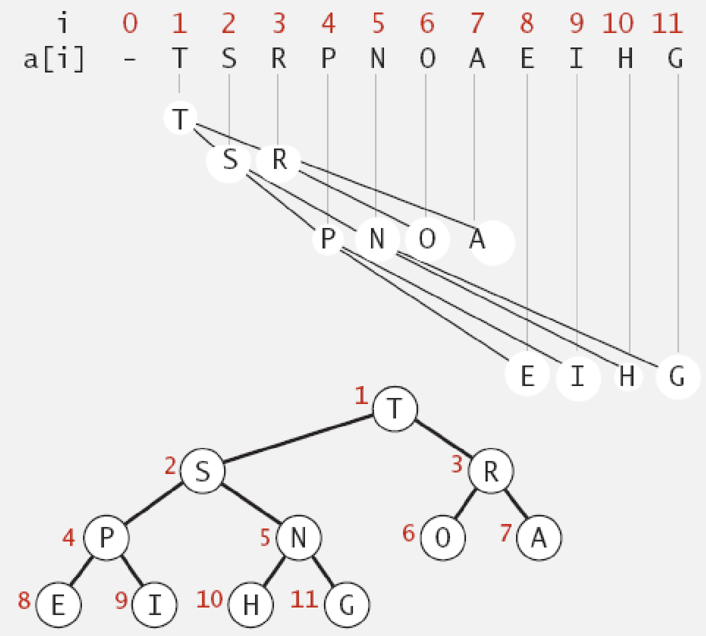
implementation	insert	delete	min/max
unordered array	1	N	N
ordered array	N	1	1
goal	log N	log N	log N

N huge

Mission Impossible?

Binary heap

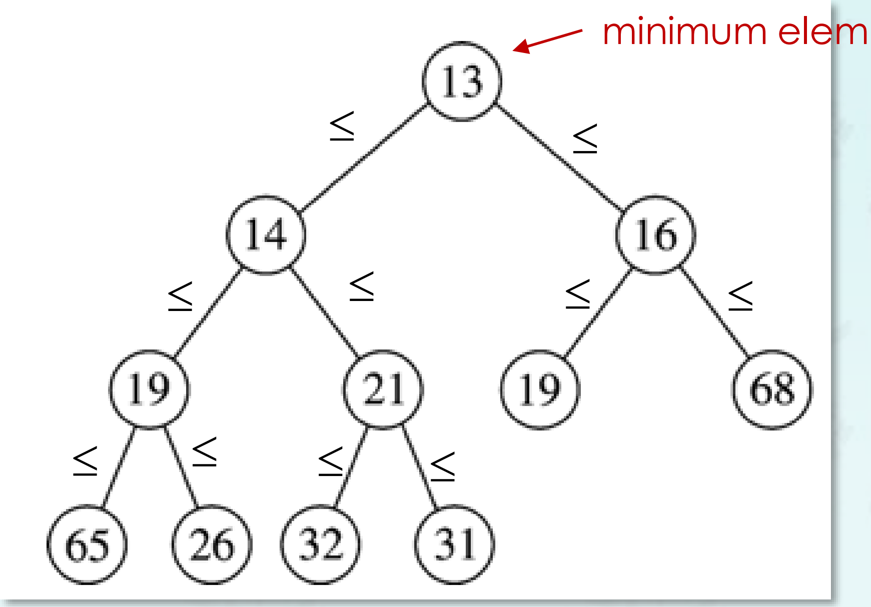
- **Binary heap**: array representation of a **heap-ordered** complete binary tree
- **Properties:**
  - **Heap-ordered:**
    Parent's key no smaller than children's keys. [maxheap]
  - **Heap-structure**:
    A complete binary tree
- Array representation
  - Indices start at 1.
  - Take nodes in level order.
    - Parent at k is at k/2.
    - Children at k are at 2k and 2k+1.
  - No explicit links needed!



i	0	1	2	3	4	5	6	7	8	9	10	11
a[i]	-	T	S	R	P	N	O	A	E	I	H	G

Heap representations

minheap example

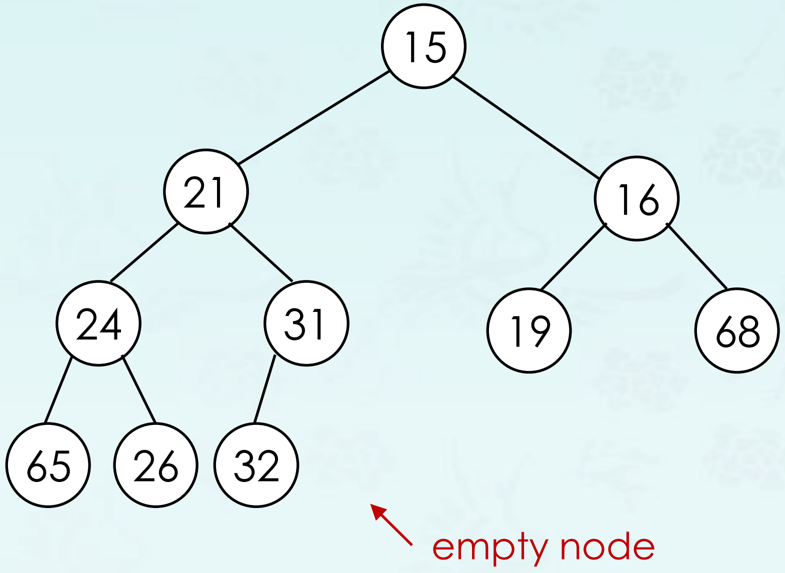


minimum element

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

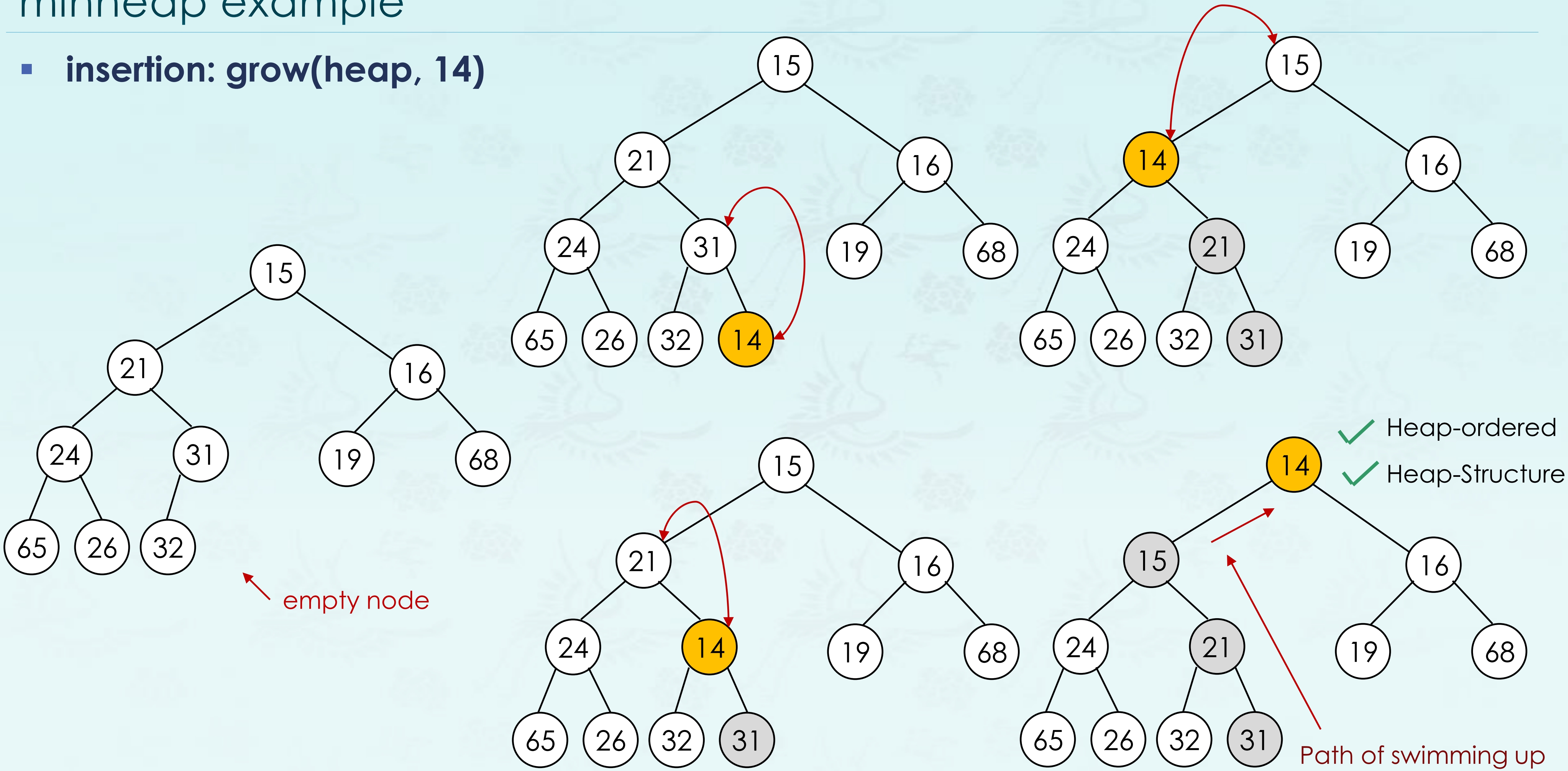
minheap example

- **insertion: grow(heap, 14)**
  - Insert a new element **while maintaining a heap-structure**
  - Move the element up the heap **while not satisfying heap-ordered**
  - Where is an empty node to start?



minheap example

- **insertion: grow(heap, 14)**



empty node

✓ Heap-ordered

✓ Heap-Structure

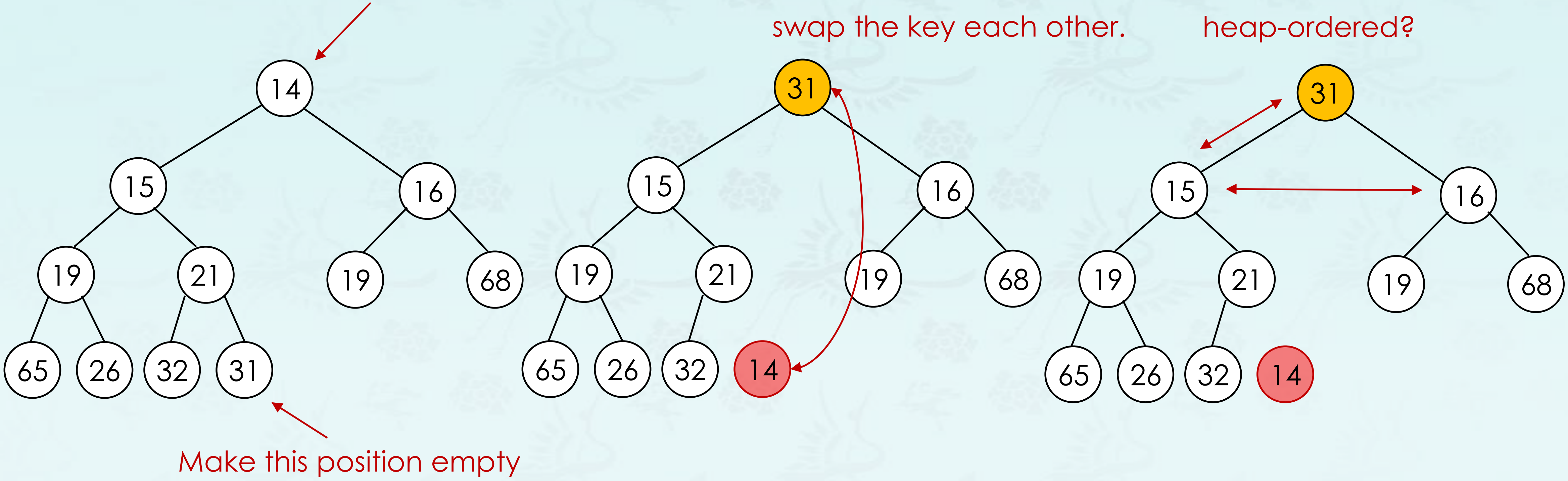
Path of swimming up

minheap example

- **deletion: dequeue – delete the root**
  - Swap the root and thelast 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 minheap definition).

minheap example

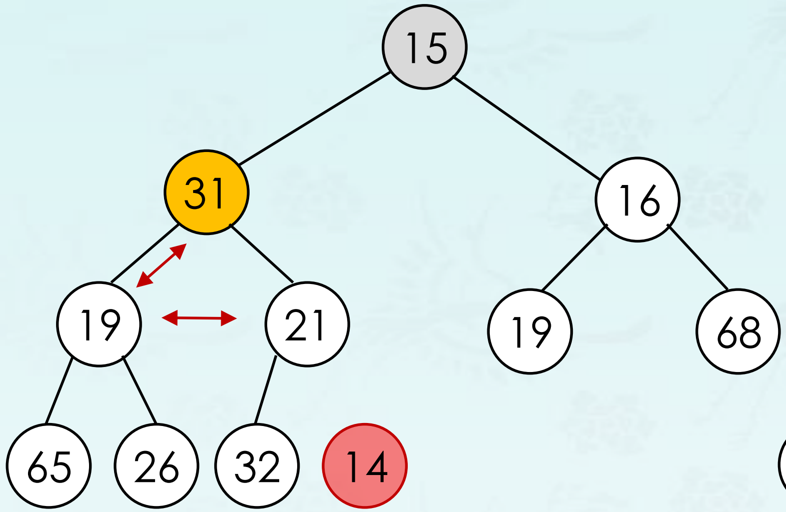
- **deletion: trim() or dequeue() -** Which position of the node will be empty?



minheap example

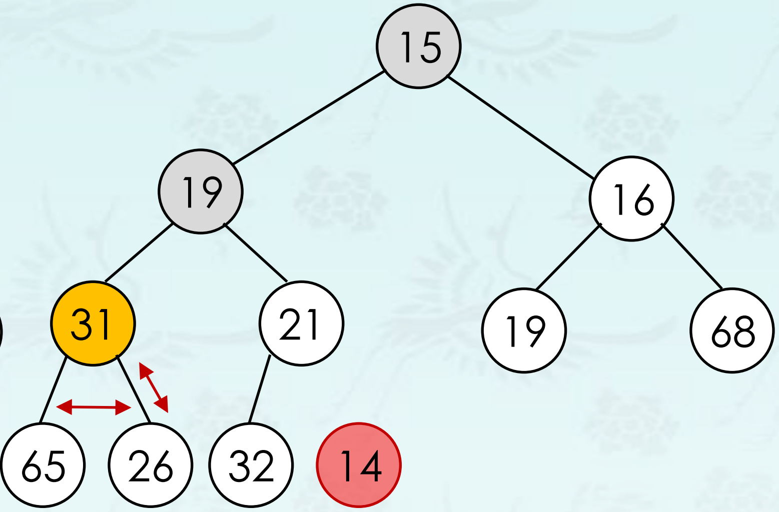
- **deletion: trim() or dequeue() -** Which position of the node will be empty?

heap-ordered?



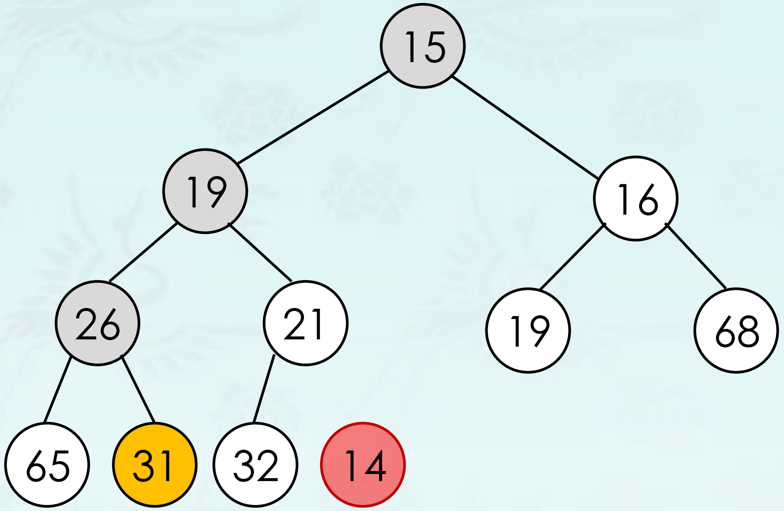
- Is 31 > min(14,16)?
- Yes - swap 31 with min(14,16)

heap-ordered?



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

heap-ordered?



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

✓ Heap-ordered

✓ Heap-Structure

Binary heap operations 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 in min/max heap
- decreaseKey: O(log N)
- increaseKey: O(log N)
- remove: O(log N) // removing a node in any location

Binary heap operations time complexity with N items:

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

Mission Completed

Data Structures
Chapter 5: Heap and Priority Queue

1. **introduction**
   - Complete Binary Tee (Review)
   - Heap and Priority Queue
2. Binary Heap
   - Min heap, **Max heap**
   - Priority Queue
3. Heapsort
4. Heap & PQ Coding

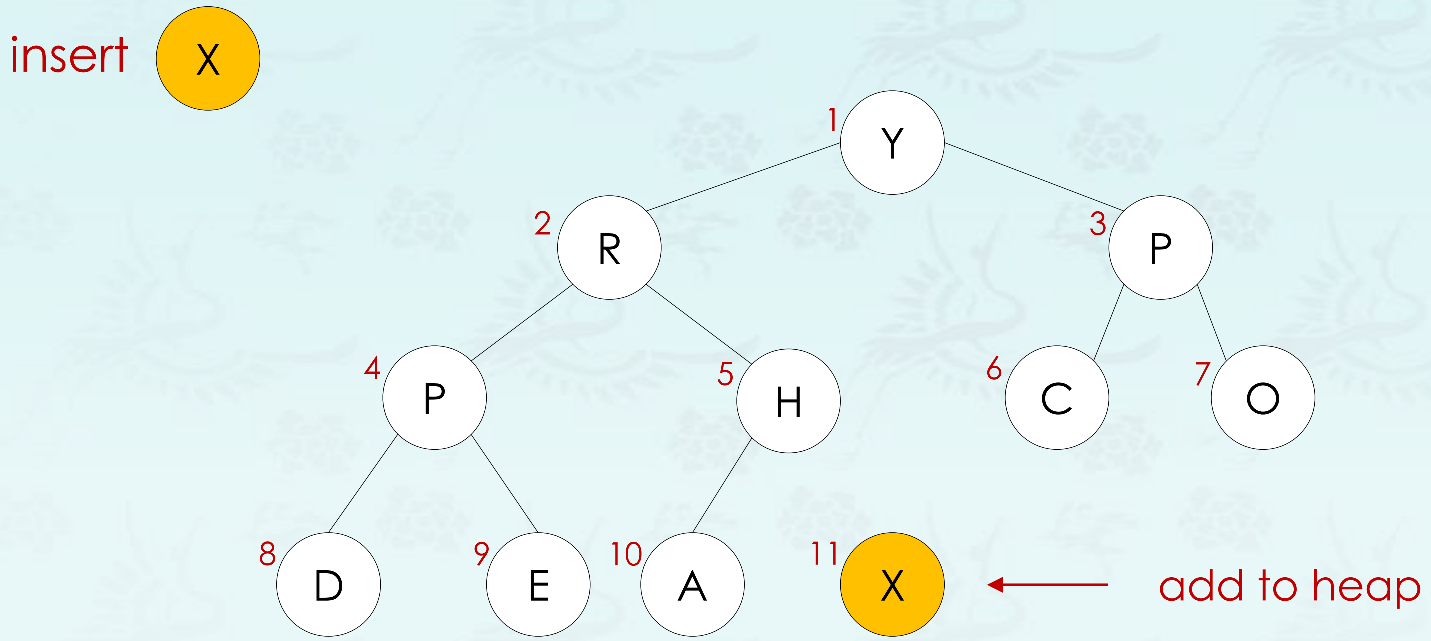
Heap ADT - heap.h

- Heap ADT: A **one - based** and **one dimensional array** is used to simplify parent and child calculations.
- heap.h

```
struct Heap {
  int *nodes;         // an array of nodes
  int capacity;       // array size of node or key, item
  int N;              // the number of nodes in the heap
  bool (*comp)(Heap*, int, int);
  Heap(int capa = 2) {
    capacity = capa;
    nodes = new int[capacity];
    N = 0;
    comp = nullptr;
  };
  ~Heap() {};
};
using heap = Heap*;
```

maxheap example

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

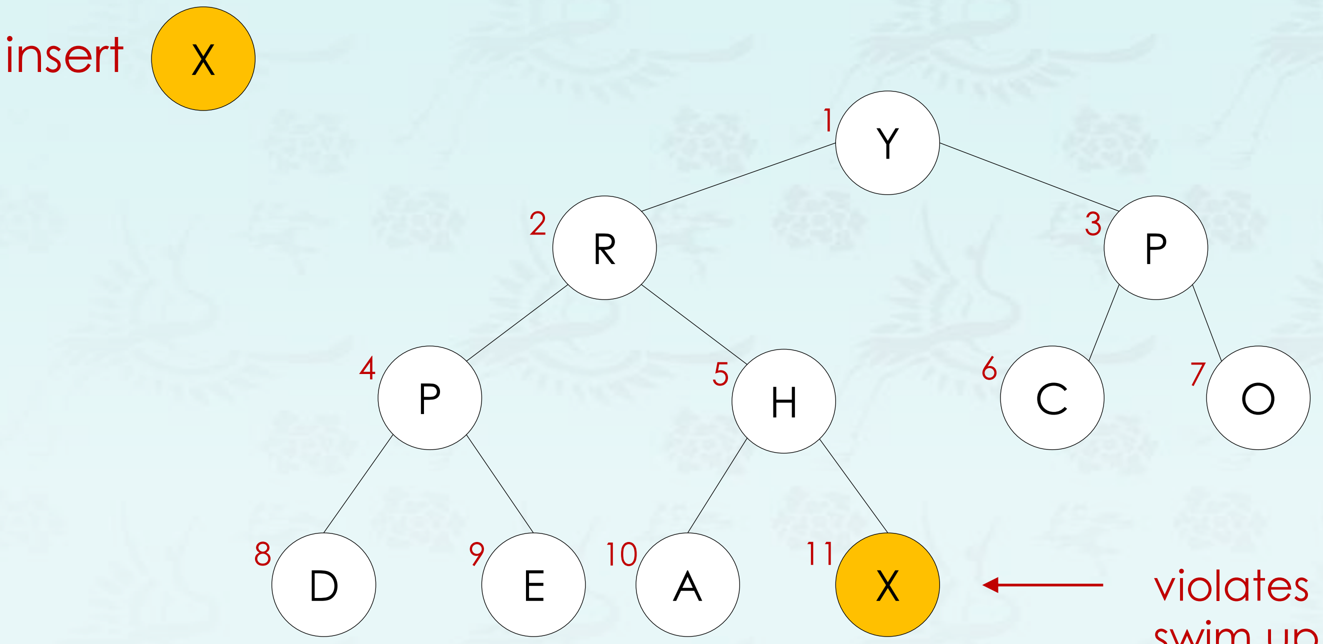


`h->N` `h->nodes[++h->N] = key;`

heap-ordered		Y	R	P	P	H	C	O	D	E	A		
	0	1	2	3	4	5	6	7	8	9	10	11	

maxheap example

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

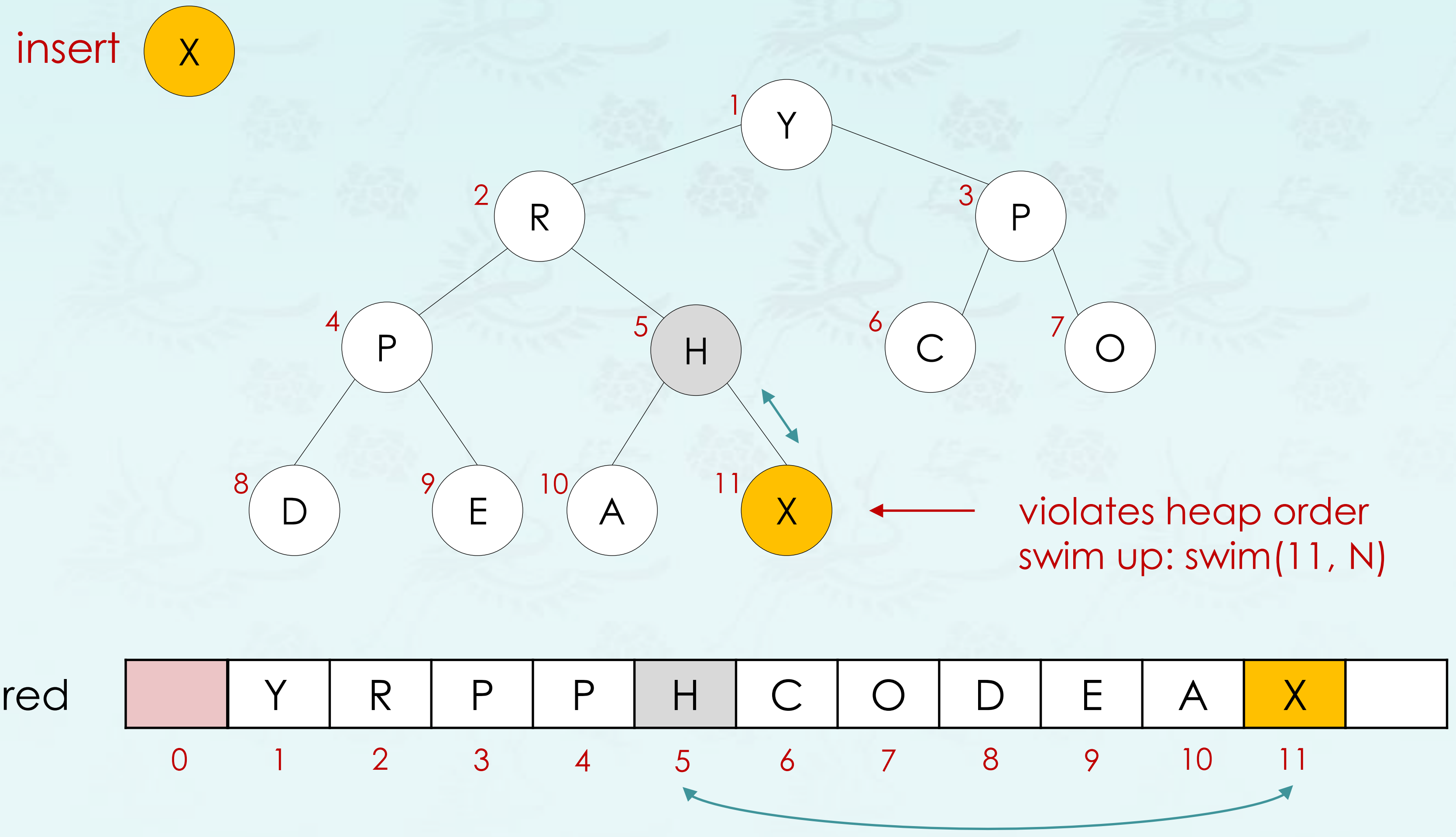


violates heap order
swim up: swim(11, N)

heap-ordered		Y	R	P	P	H	C	O	D	E	A	X	
	0	1	2	3	4	5	6	7	8	9	10	11	
												h->N	

maxheap example

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

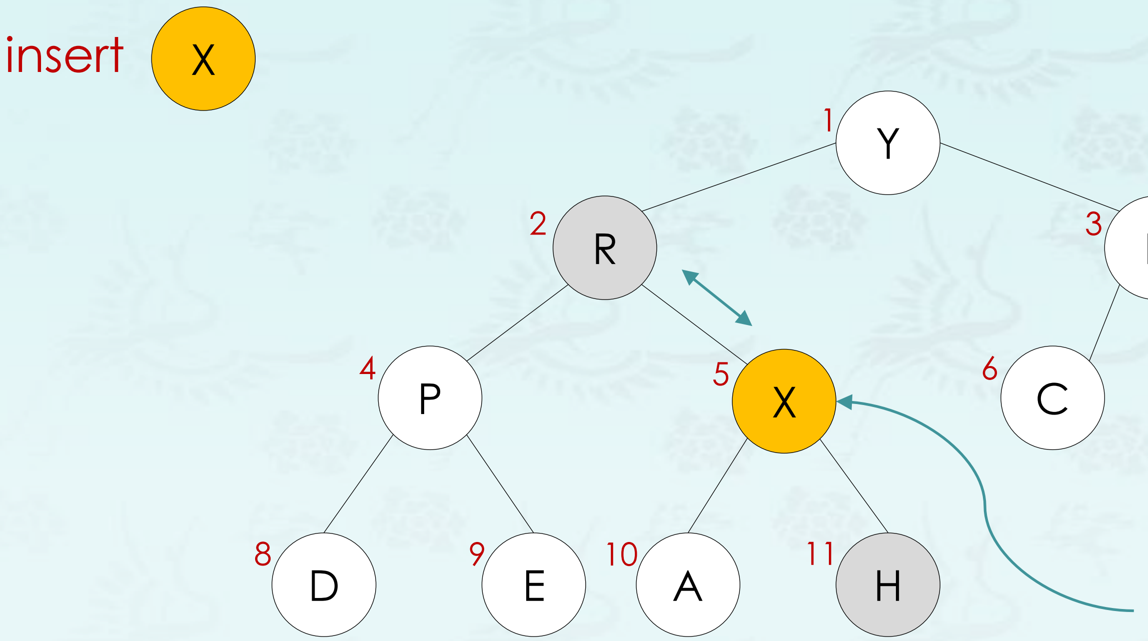


violates heap order
swim up: swim(11, N)

0	1	2	3	4	5	6	7	8	9	10	11	
	Y	R	P	P	H	C	O	D	E	A	X	

maxheap example

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

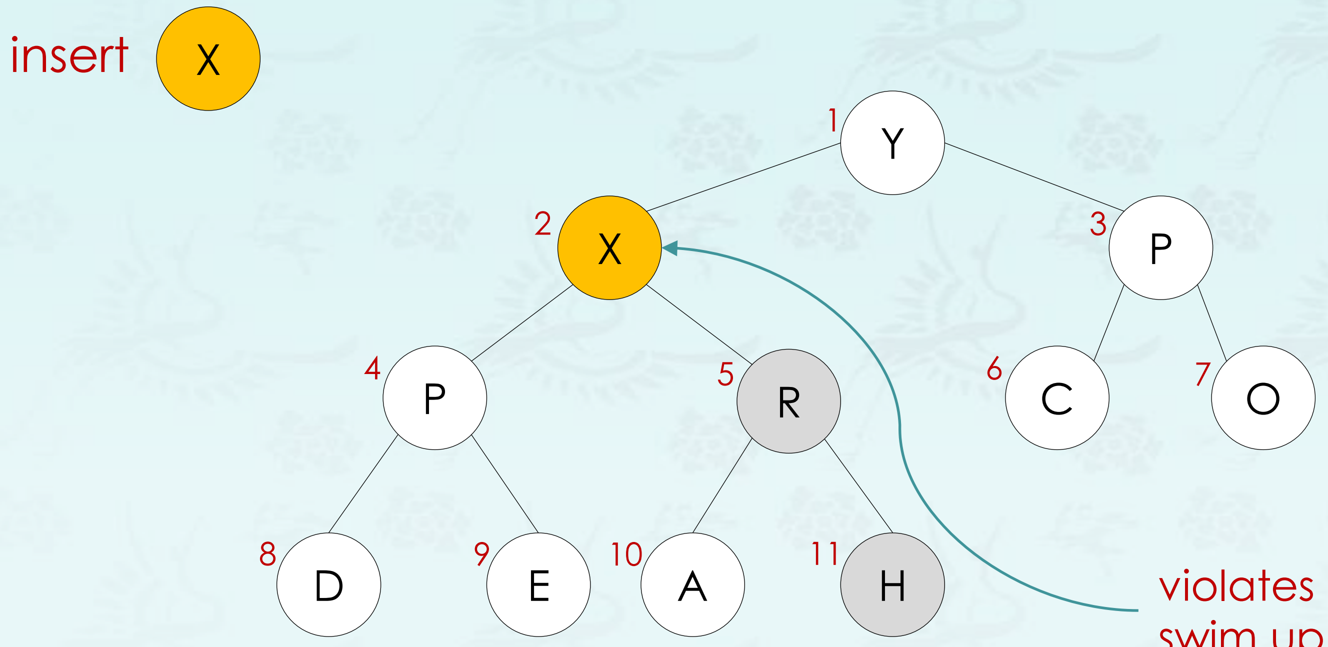


violates heap order
swim up: swim(11, N)

heap-ordered		Y	R	P	P	X	C	O	D	E	A	H	
	0	1	2	3	4	5	6	7	8	9	10	11	

maxheap example

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

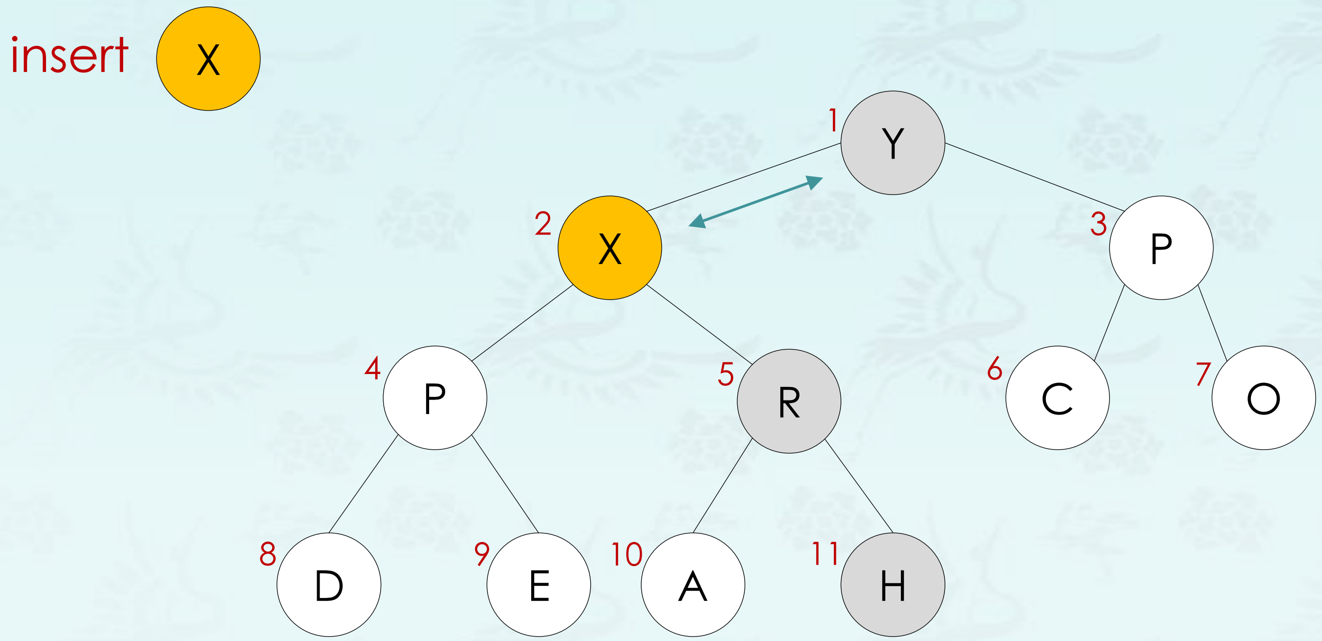


violates heap order
swim up: swim(11, N)

heap-ordered		Y	X	P	P	R	C	O	D	E	A	H	
	0	1	2	3	4	5	6	7	8	9	10	11	

maxheap example

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



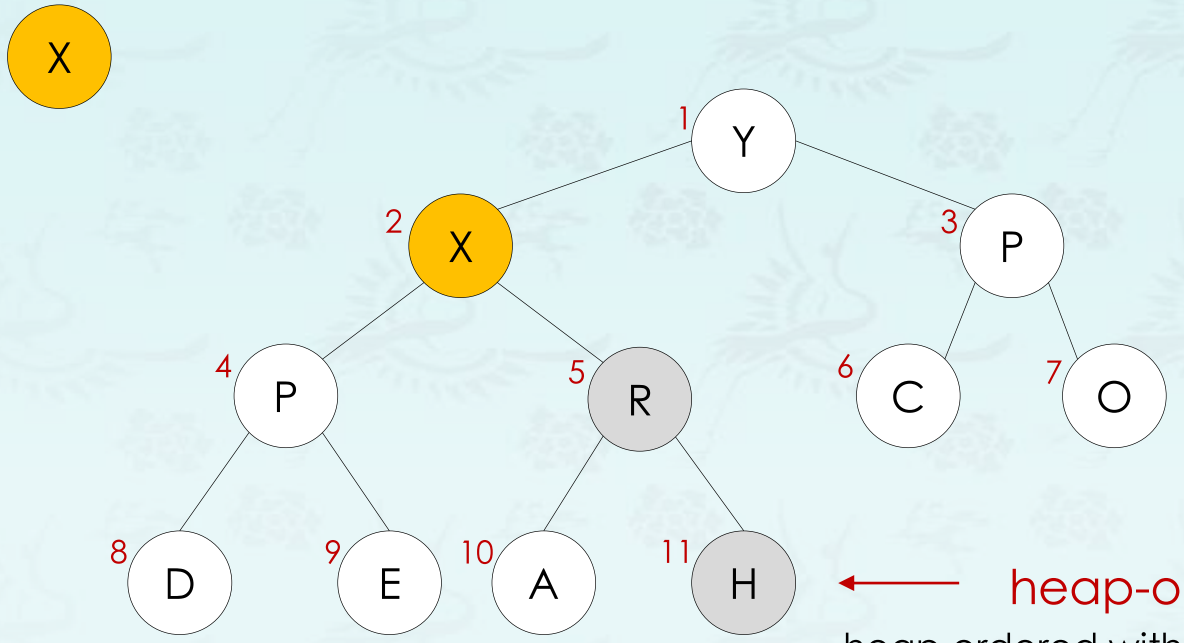
heap-ordered

	Y	X	P	P	R	C	O	D	E	A	H	
0	1	2	3	4	5	6	7	8	9	10	11	

maxheap example

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

insert



heap-ordered

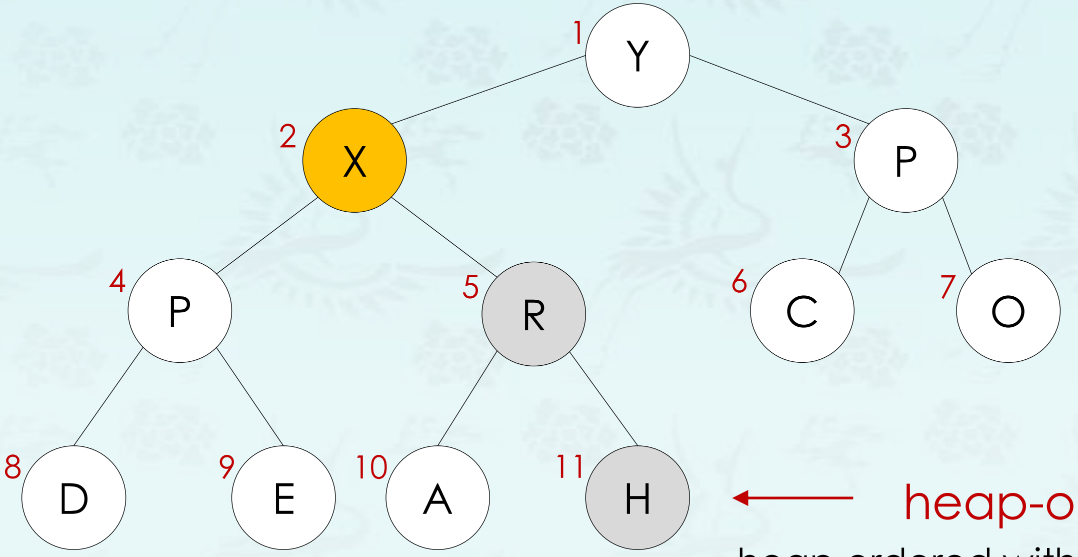
heap ordered with two swaps only!

heap-ordered		Y	X	P	P	R	C	O	D	E	A	H	
	0	1	2	3	4	5	6	7	8	9	10	11	

maxheap 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

heap ordered with two swaps only!

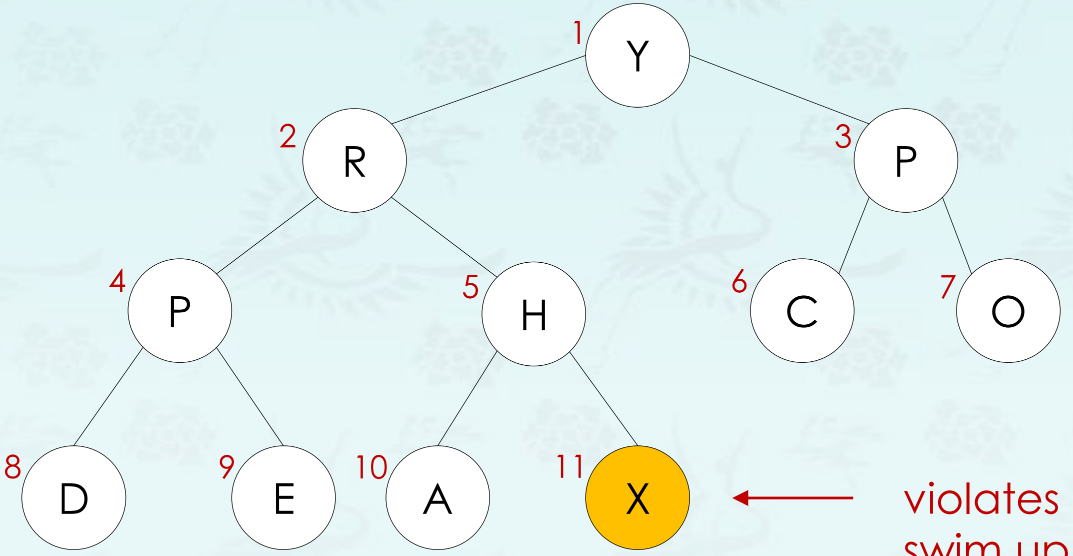
heap-ordered		Y	X	P	P	R	C	O	D	E	A	H	
	0	1	2	3	4	5	6	7	8	9	10	11	

maxheap example

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

```
void swim(heap h, int k) {
  while (k > 1 && less(h, k / 2, k)) {
    swap(h, k / 2, k);
    k = k / 2;
  }
}
```

How to code insert X



violates heap order
swim up: swim(11, N)

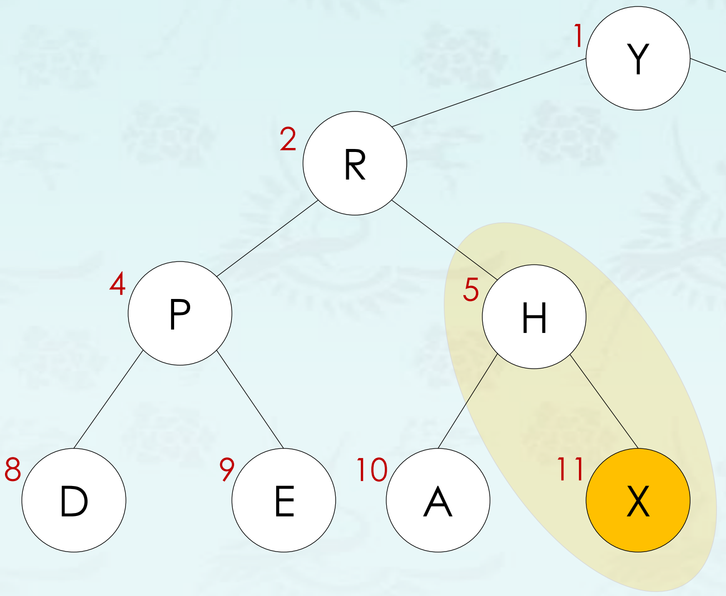
heap-ordered		Y	R	P	P	H	C	O	D	E	A	X
	0	1	2	3	4	5	6	7	8	9	10	11

maxheap example

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

```
void swim(heap h, int k) {          k=11
  while (k > 1 && less(h, k / 2, k)) {
    swap(h, k / 2, k);          k/2=5, k=11
    k = k / 2;                  k=5
  }
}
```

How to code insert X



violates heap order
swim up: swim(11, N)

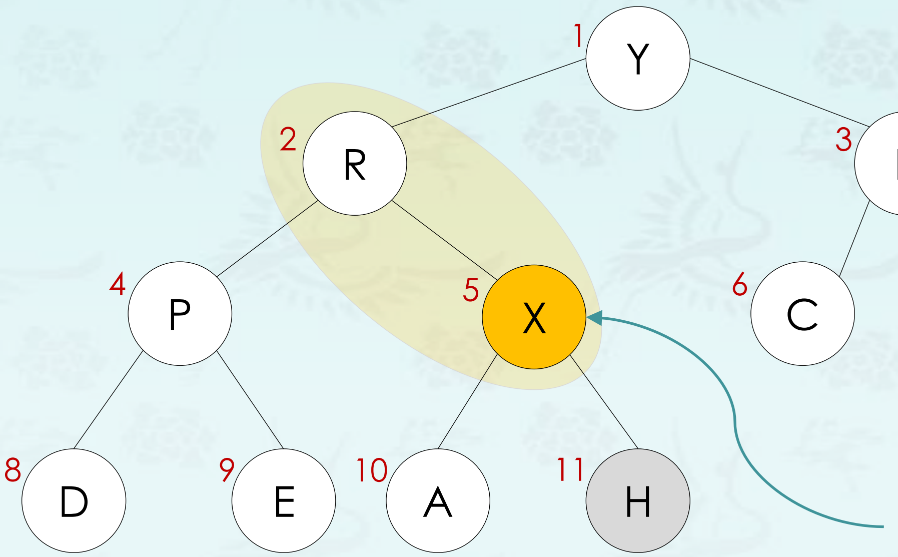
heap-ordered		Y	R	P	P	H	C	O	D	E	A	X	
	0	1	2	3	4	5	6	7	8	9	10	11

maxheap example

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

```
void swim(heap h, int k) {          k=5
  while (k > 1 && less(h, k / 2, k)) {
    swap(h, k / 2, k);       k/2=2, k=5
    k = k / 2;               k=2
  }
}
```

How to code insert X



violates heap order
swim up: swim(11, N)

heap-ordered

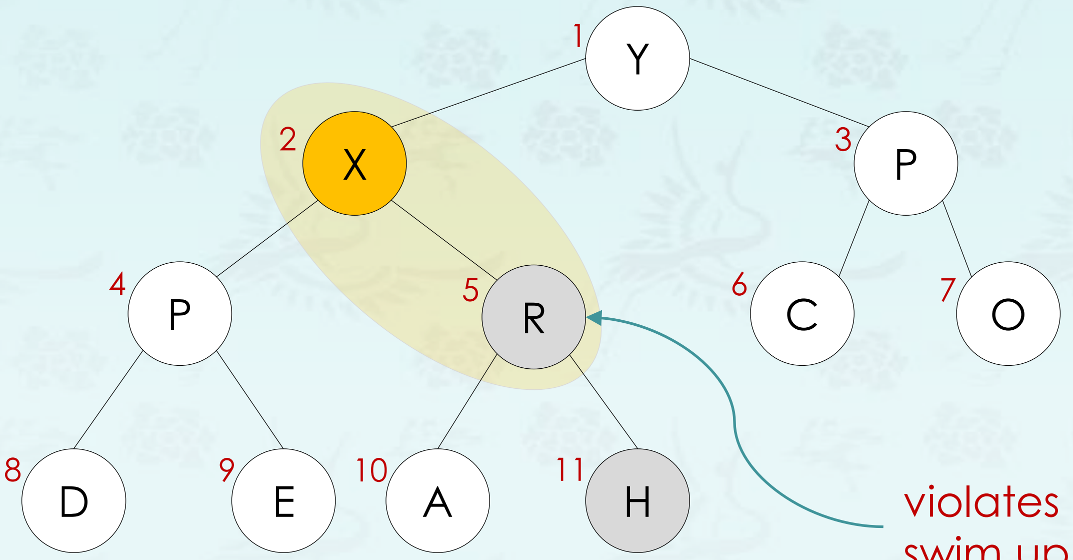
	Y	R	P	P	X	C	O	D	E	A	H	
0	1	2	3	4	5	6	7	8	9	10	11	

maxheap example

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

```
void swim(heap h, int k) {
  while (k > 1 && less(h, k / 2, k)) {
    swap(h, k / 2, k);     k/2=2, k=5
    k = k / 2;             k=2
  }
}
```

How to code insert X



violates heap order
swim up: swim(11, N)

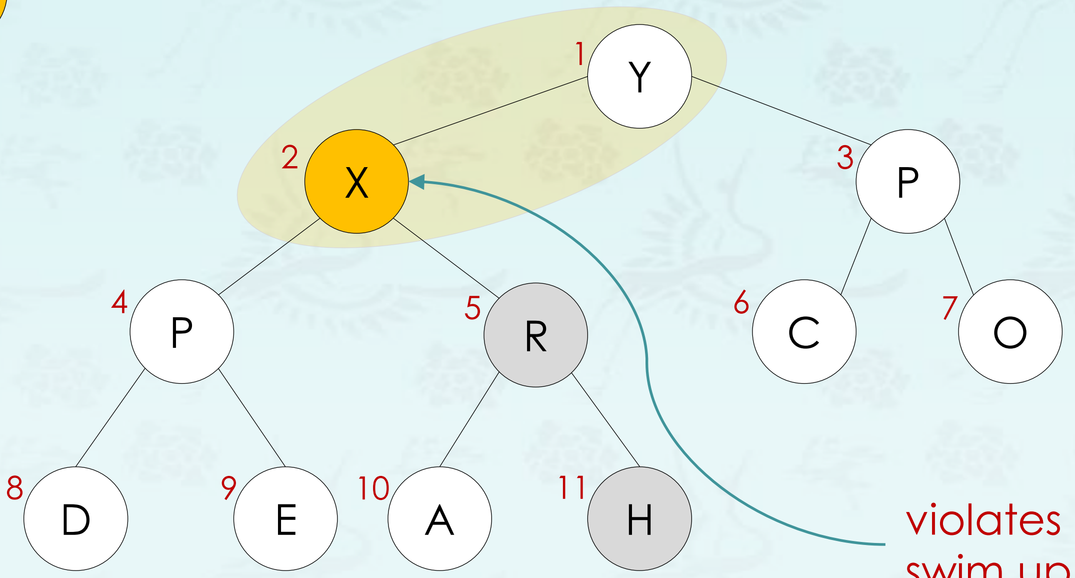
heap-ordered		Y	X	P	P	R	C	O	D	E	A	H	
	0	1	2	3	4	5	6	7	8	9	10	11	

maxheap example

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

```
void swim(heap h, int k) {          k=2
  while (k > 1 && less(h, k / 2, k)) {
    swap(h, k / 2, k);          k/2=1, k=2
    k = k / 2;                  exit while()
  }
}
```

How to code insert X



violates heap order
swim up: swim(11, N)

heap-ordered

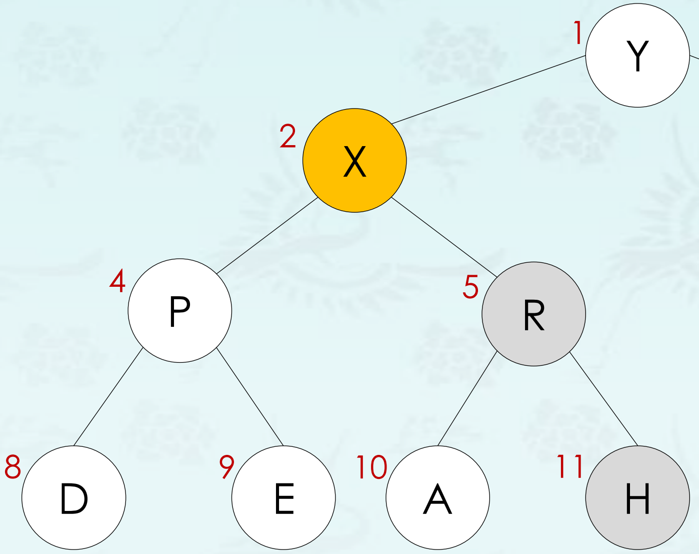
0	1	2	3	4	5	6	7	8	9	10	11	
	Y	X	P	P	R	C	O	D	E	A	H	

Prof. Youngsup Kim, idebtor@gmail.com, CSEE Dept., Grace School Rm204, Handong Global University

maxheap example

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

How to code insert X



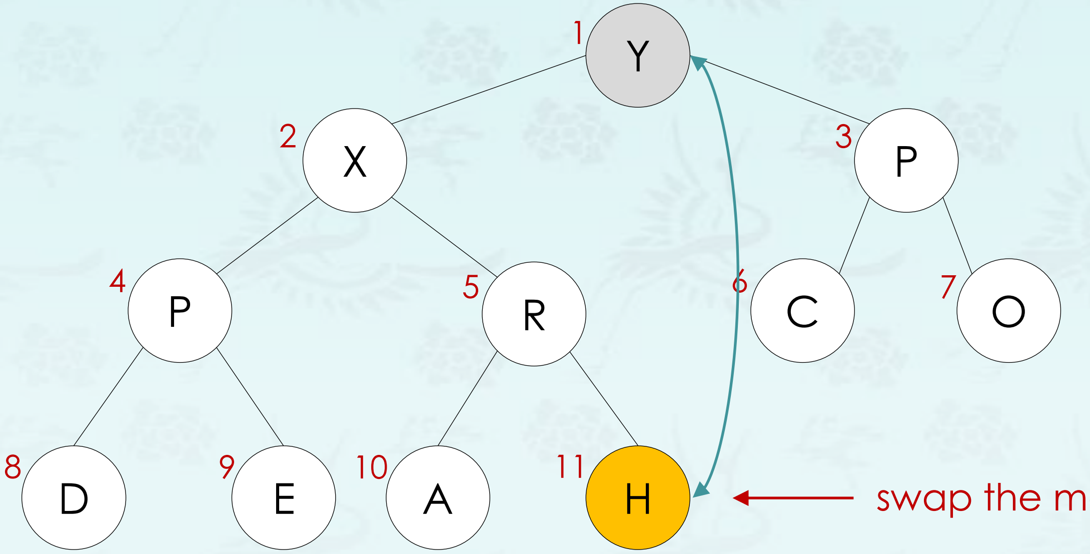
heap-ordered
swim up: swim(11, N)

heap-ordered		Y	X	P	P	R	C	O	D	E	A	H	
	0	1	2	3	4	5	6	7	8	9	10	11	

maxheap example

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

remove the max (root)



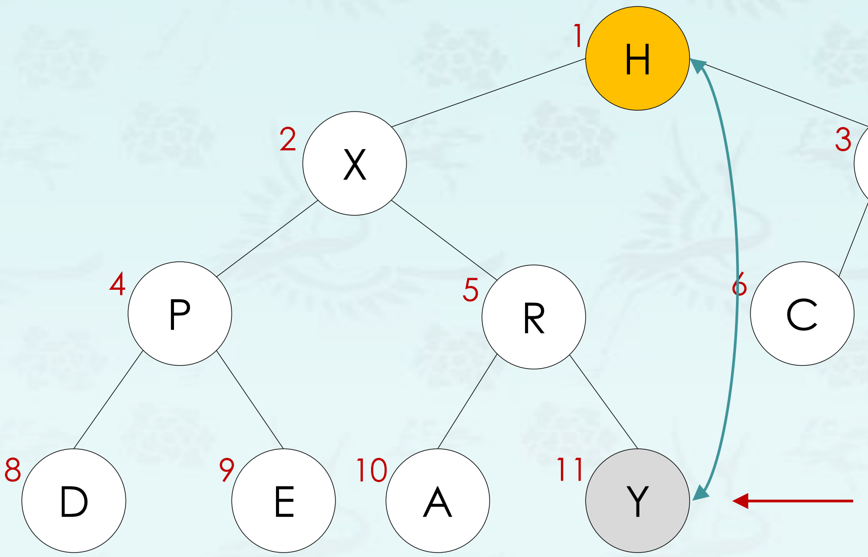
swap the max with the last node

heap-ordered		Y	X	P	P	R	C	O	D	E	A	H	
	0	1	2	3	4	5	6	7	8	9	10	11	
												h->N	

maxheap example

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

remove the max (root)



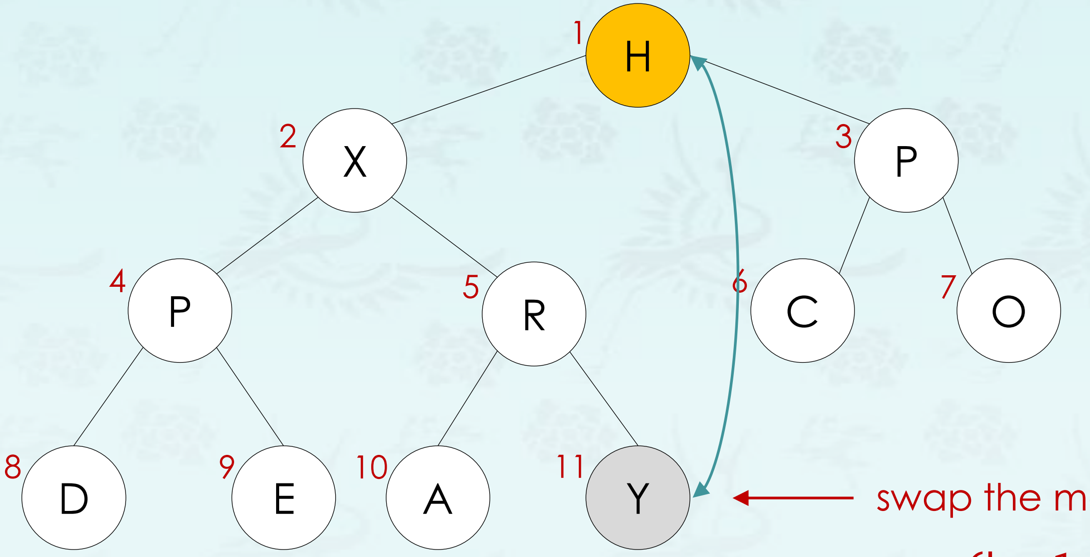
swap the max with the last node

heap-ordered		H	X	P	P	R	C	O	D	E	A	Y	
	0	1	2	3	4	5	6	7	8	9	10	11	
												h->N	

maxheap example

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

remove the max (root)



swap the max with the last node

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

heap-ordered

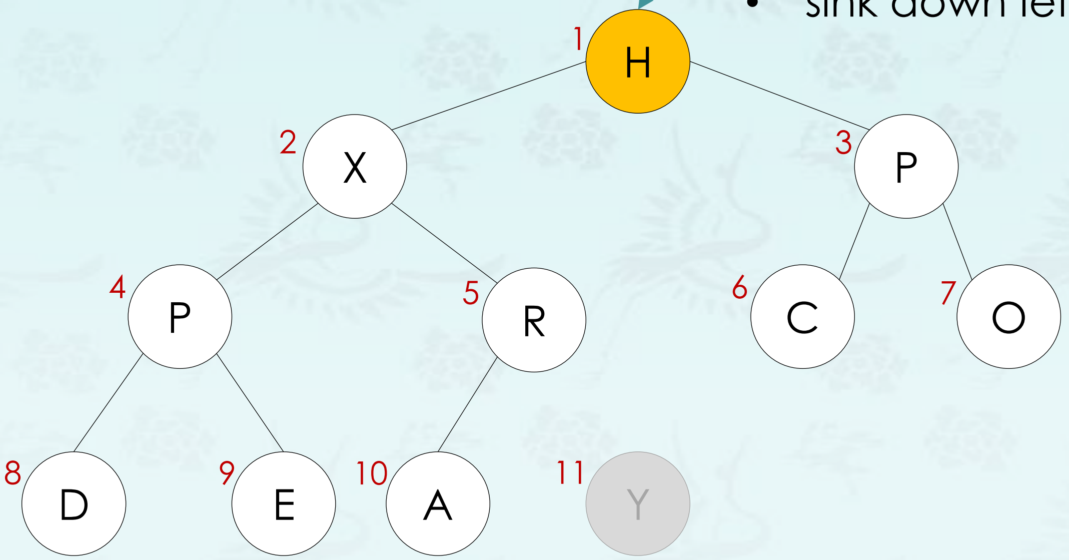
	H	X	P	P	R	C	O	D	E	A	Y	
0	1	2	3	4	5	6	7	8	9	10	11	
											h->N	

maxheap example

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

remove the max (root)

- violates heap order: sink(1, N)
- sink down left or right?



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

heap-ordered

	H	X	P	P	R	C	O	D	E	A	Y	
0	1	2	3	4	5	6	7	8	9	10	11	

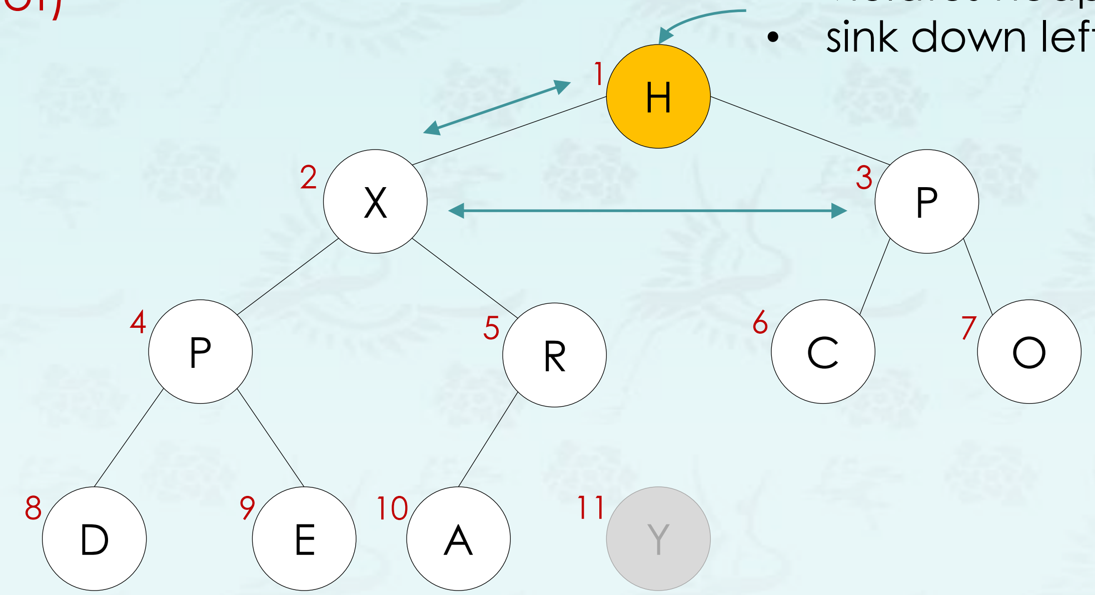
`h->N` heap size decreased by one

maxheap example

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

remove the max (root)

- violates heap order: sink(1, N)
- sink down left or right?

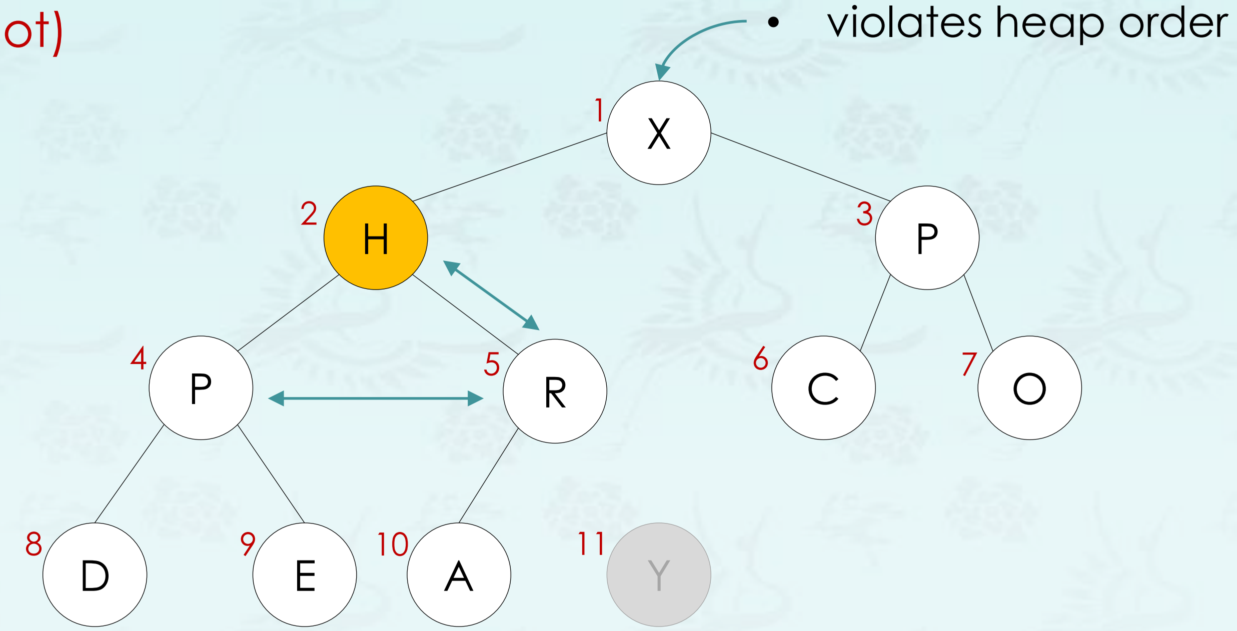


heap-ordered		X	H	P	P	R	C	O	D	E	A	Y	
	0	1	2	3	4	5	6	7	8	9	10	11	
											h->N		

maxheap 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

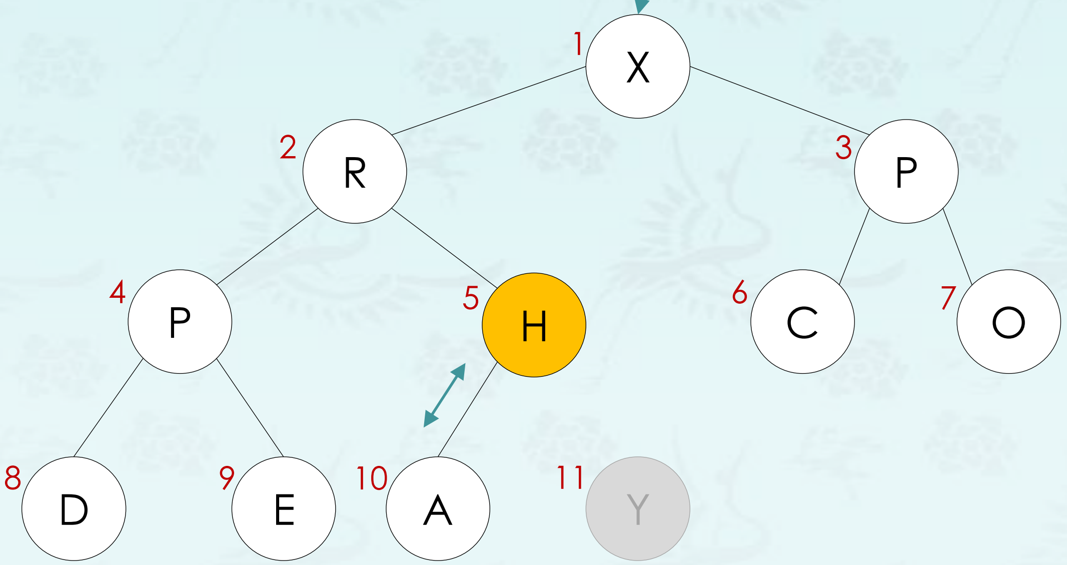
	X	H	P	P	R	C	O	D	E	A	Y	
0	1	2	3	4	5	6	7	8	9	10	11	
										h->N		

maxheap example

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

remove the max (root)

- violates heap order



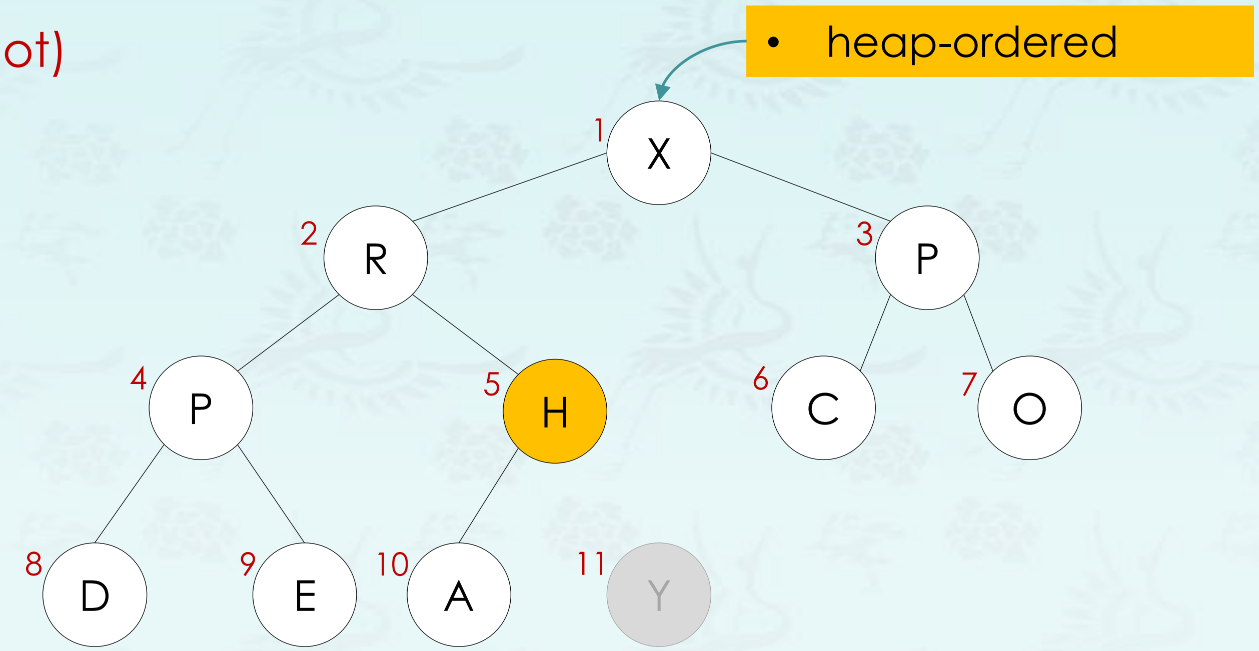
heap-ordered

	X	R	P	P	H	C	O	D	E	A	Y	
0	1	2	3	4	5	6	7	8	9	10	11	

maxheap example

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

remove the max (root)



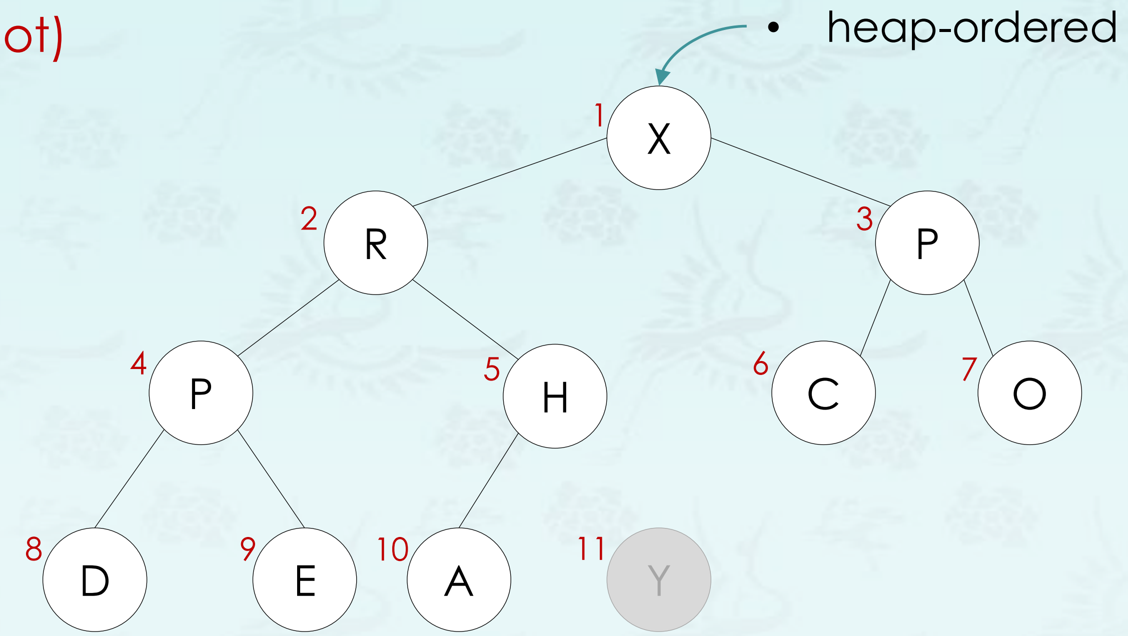
- heap-ordered

heap-ordered		X	R	P	P	H	C	O	D	E	A	Y	
	0	1	2	3	4	5	6	7	8	9	10	11	
											h->N		

maxheap 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

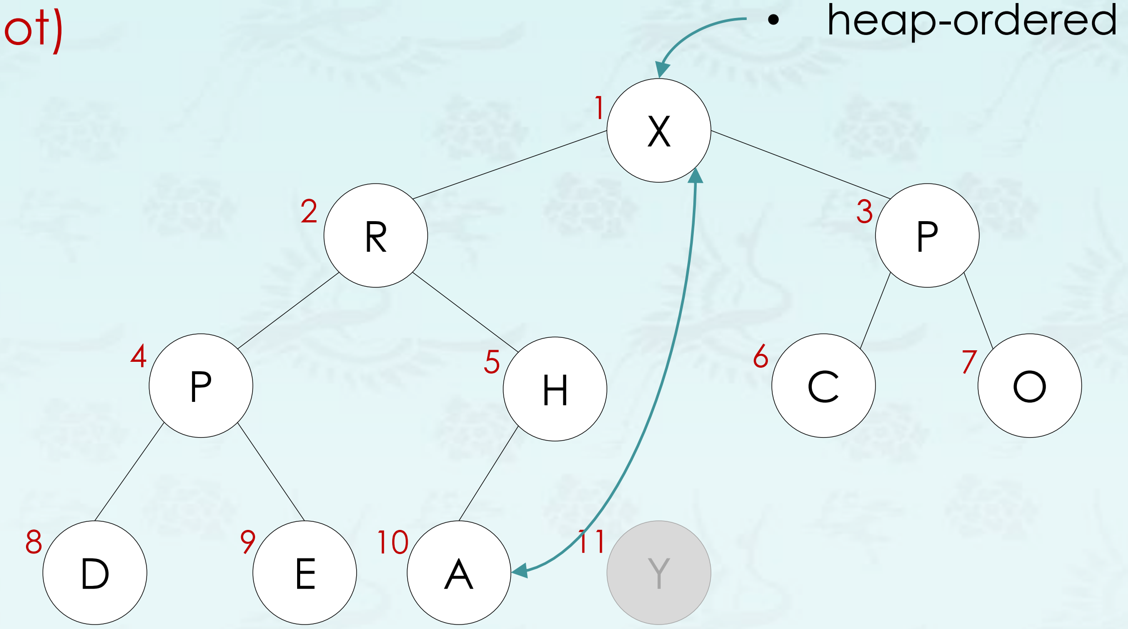
heap-ordered		X	R	P	P	H	C	O	D	E	A	Y	
	0	1	2	3	4	5	6	7	8	9	10	11	
											h->N		

maxheap 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



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

heap-ordered		X	R	P	P	H	C	O	D	E	A	Y	
	0	1	2	3	4	5	6	7	8	9	10	11	

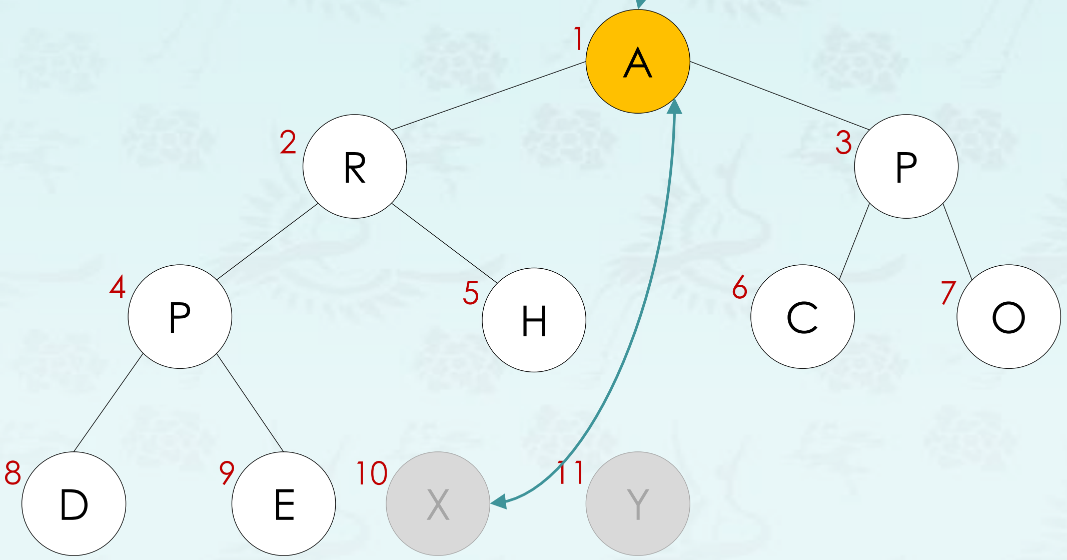
h->N heap size decreased by one

maxheap example

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

remove the max (root)

- violates heap order: sink(1, N)



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

heap-ordered		A	R	P	P	H	C	O	D	E	X	Y	
	0	1	2	3	4	5	6	7	8	9	10	11	

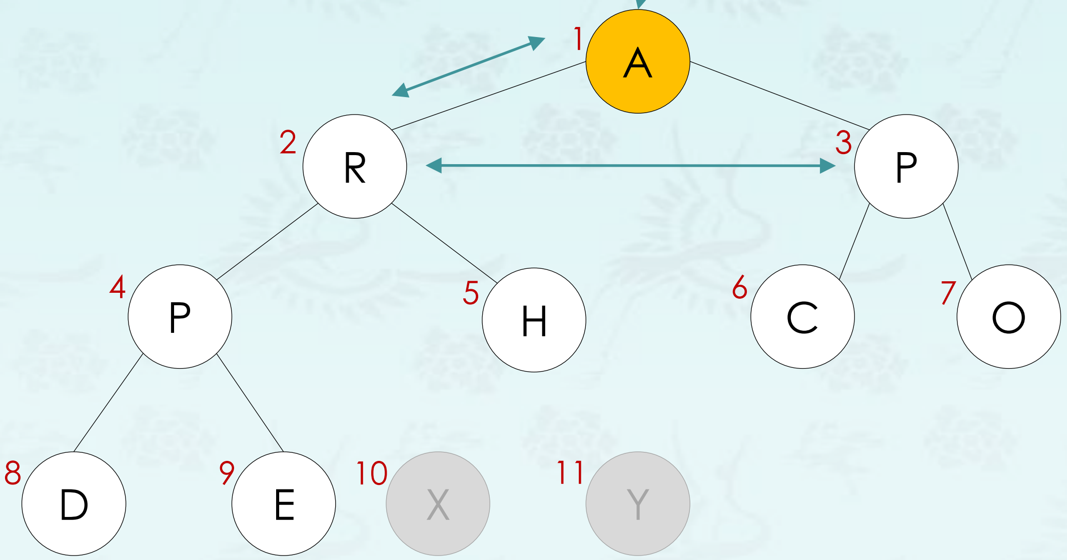
`h->N` heap size decreased by one

maxheap example

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

remove the max (root)

- violates heap order: sink(1, N)



heap-ordered

	R	A	P	P	H	C	O	D	E	X	Y	
0	1	2	3	4	5	6	7	8	9	10	11	

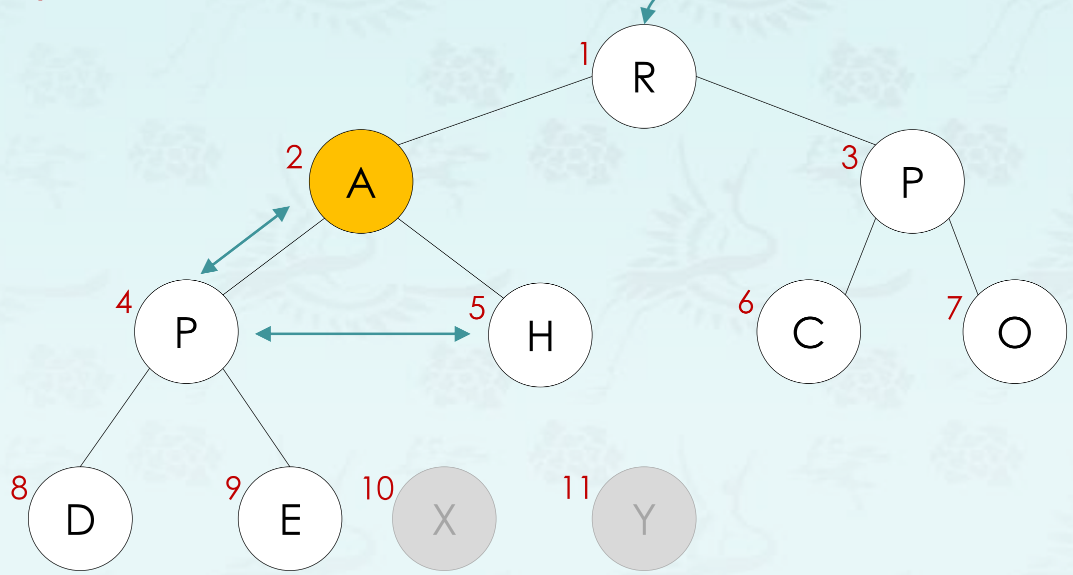
h->N heap size decreased by one

maxheap example

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

remove the max (root)

- violates heap order: sink(1, N)



heap-ordered

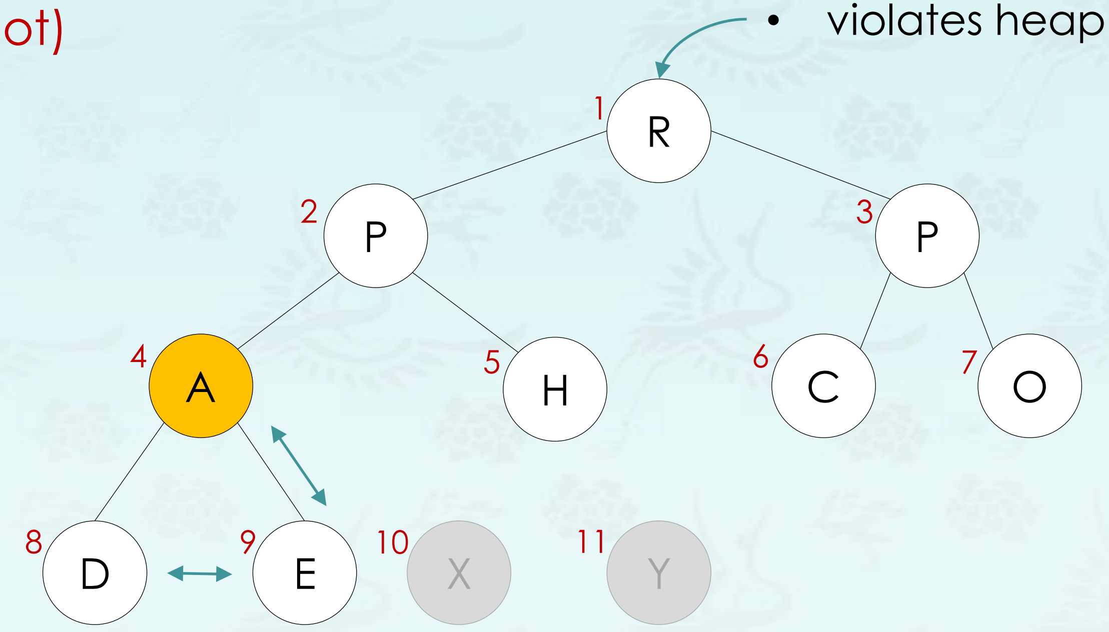
	R	A	P	P	H	C	O	D	E	X	Y	
0	1	2	3	4	5	6	7	8	9	10	11	

maxheap example

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

remove the max (root)

- violates heap order: sink(1, N)



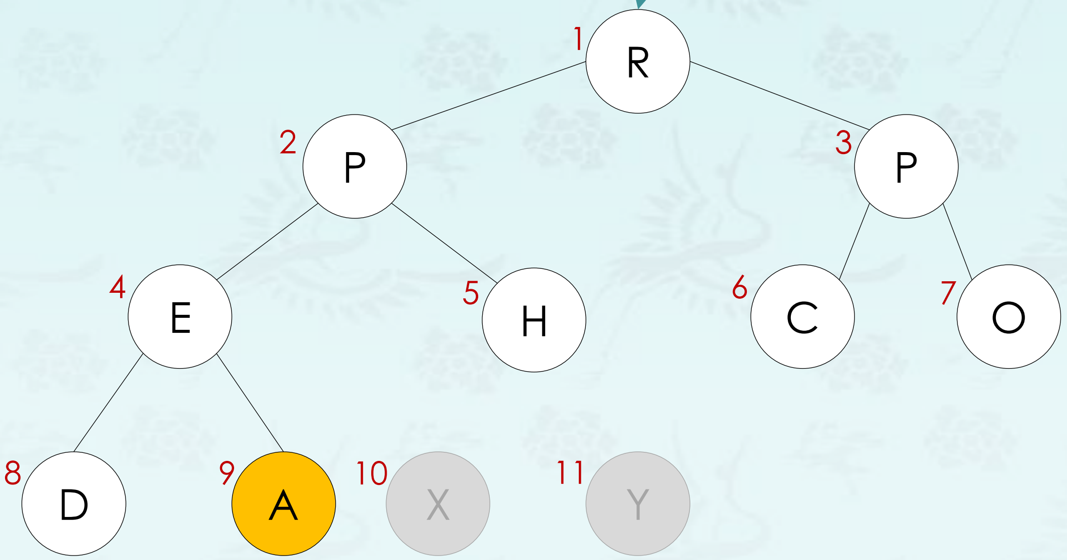
heap-ordered		R	P	P	A	H	C	O	D	E	X	Y	
	0	1	2	3	4	5	6	7	8	9	10	11	

maxheap example

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

remove the max (root)

- violates heap order: sink(1, N)

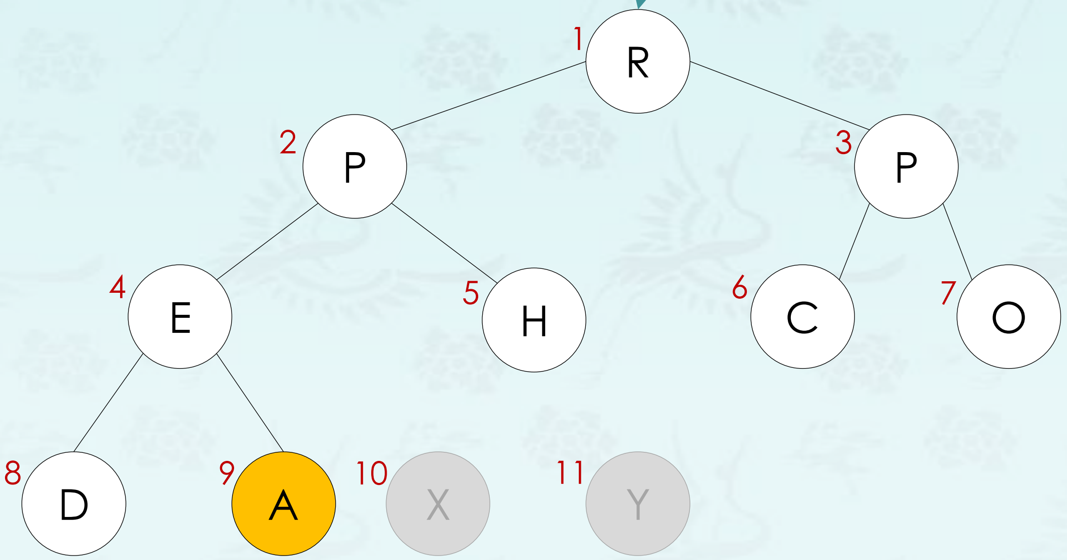


heap-ordered		R	P	P	E	H	C	O	D	A	X	Y	
	0	1	2	3	4	5	6	7	8	9	10	11	

maxheap example

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

remove the max (root)



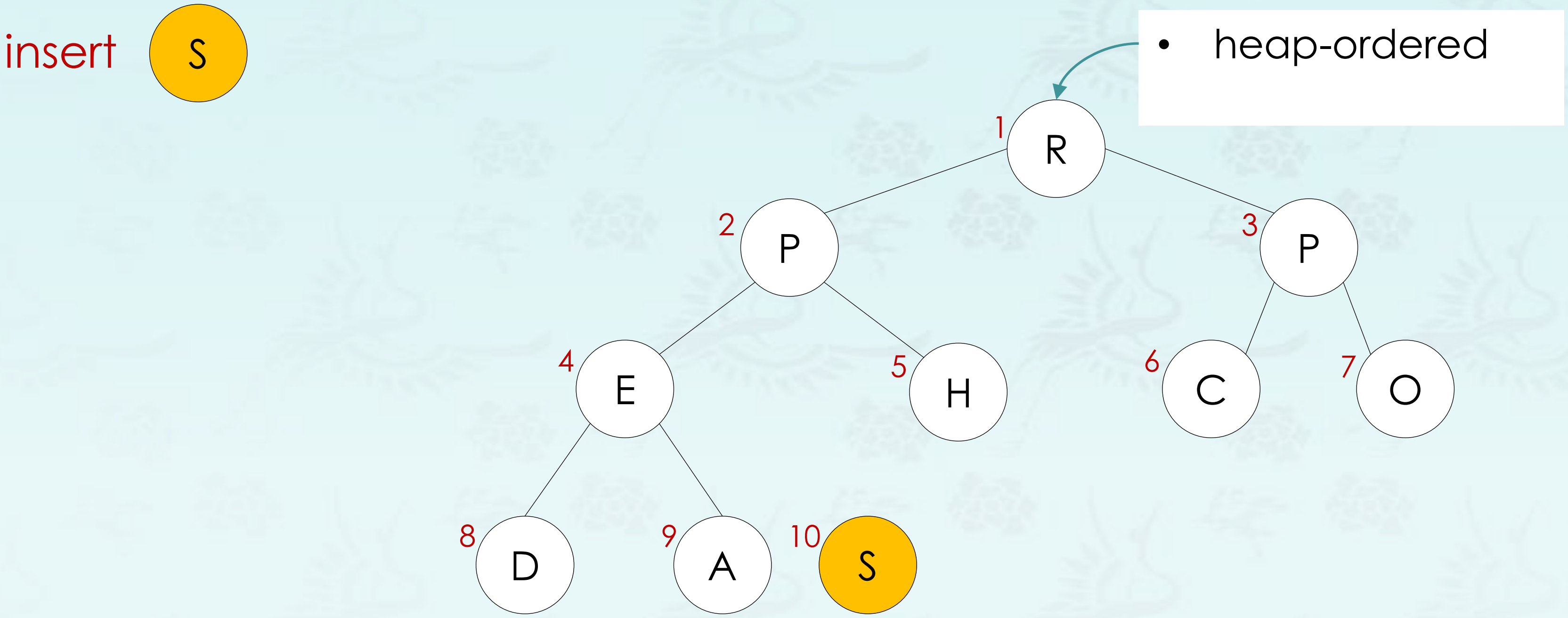
- heap-ordered

heap-ordered

0	1	2	3	4	5	6	7	8	9	10	11	
	R	P	P	E	H	C	O	D	A	X	Y	

maxheap example

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



heap-ordered

h->N

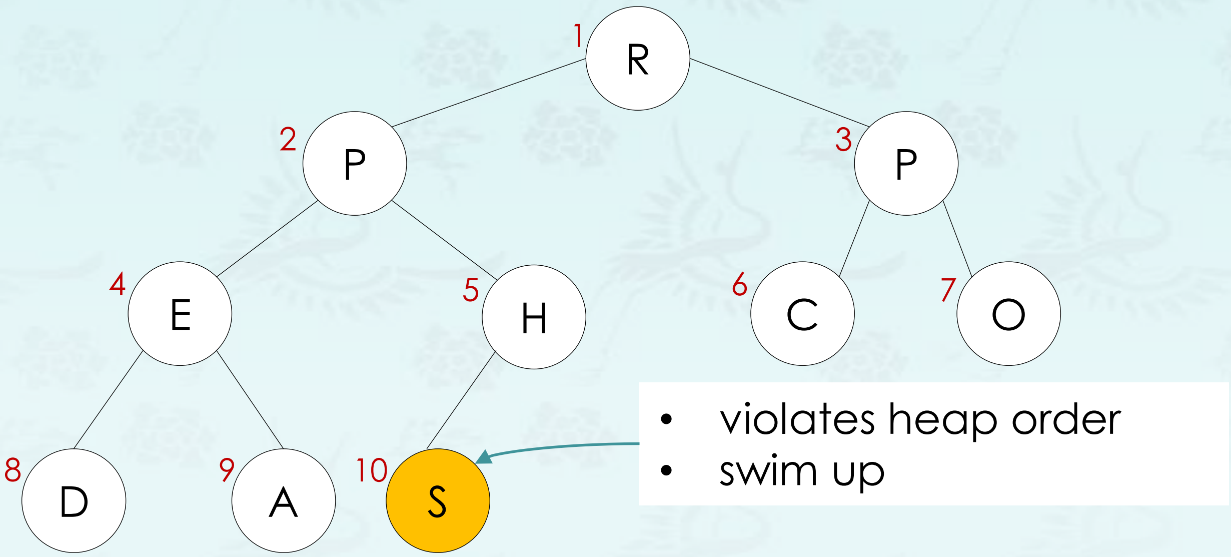
```
h->nodes[++h->N] = key;
```

	R	P	P	E	H	C	O	D	A	S	
0	1	2	3	4	5	6	7	8	9	10	11

maxheap example

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

insert S



- violates heap order
- swim up

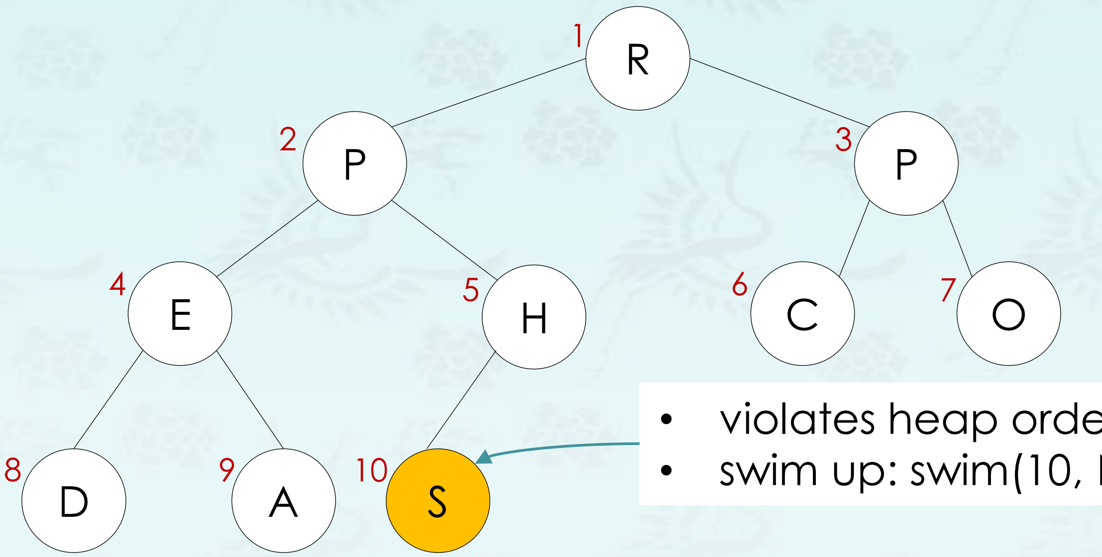
h->N

										h->N		
heap-ordered		R	P	P	E	H	C	O	D	A	S	
	0	1	2	3	4	5	6	7	8	9	10	11

maxheap example

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

insert S



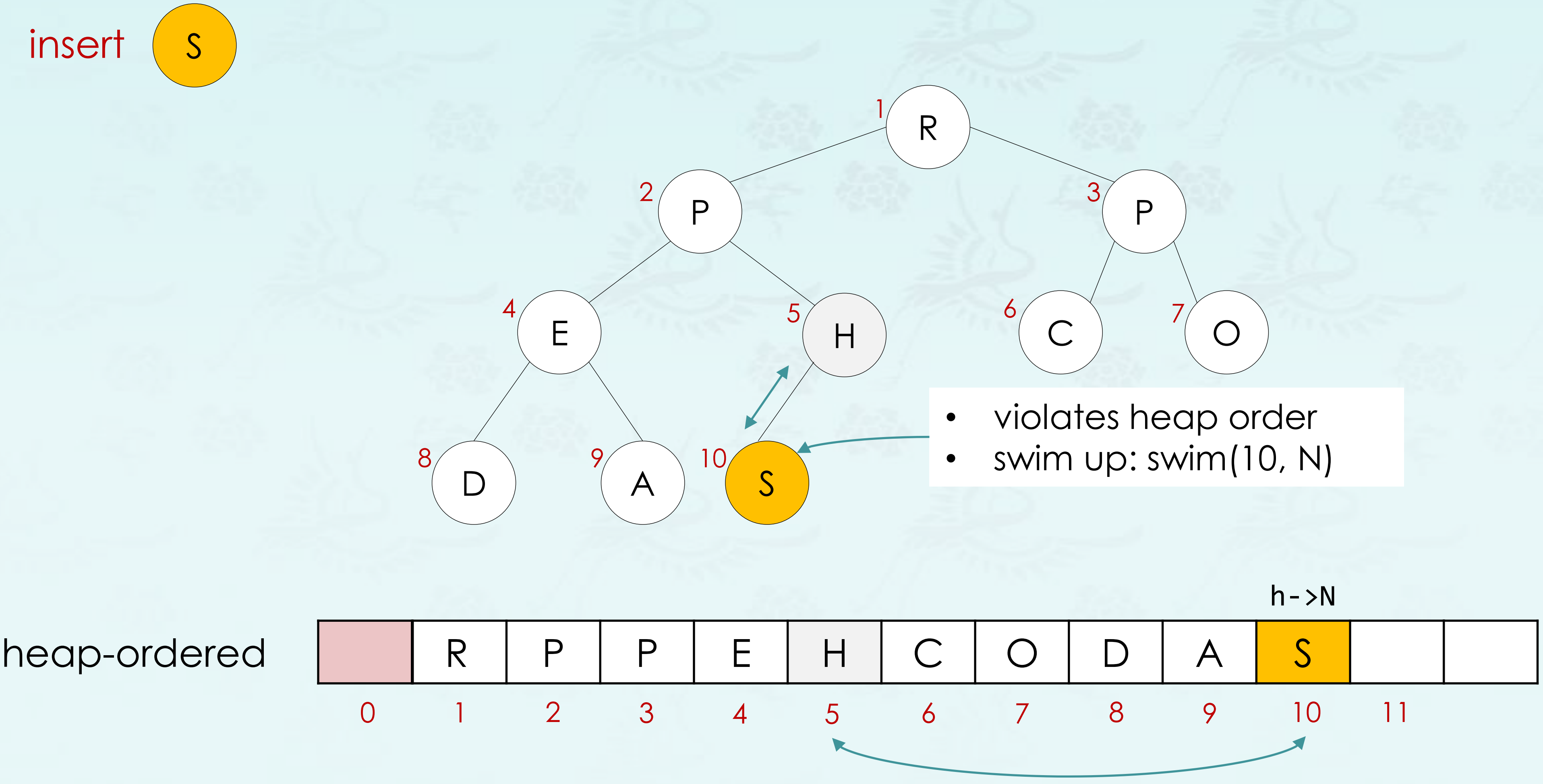
- violates heap order
- swim up: swim(10, N)

heap-ordered

										h->N	
	R	P	P	E	H	C	O	D	A	S	
0	1	2	3	4	5	6	7	8	9	10	11

maxheap example

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



insert S

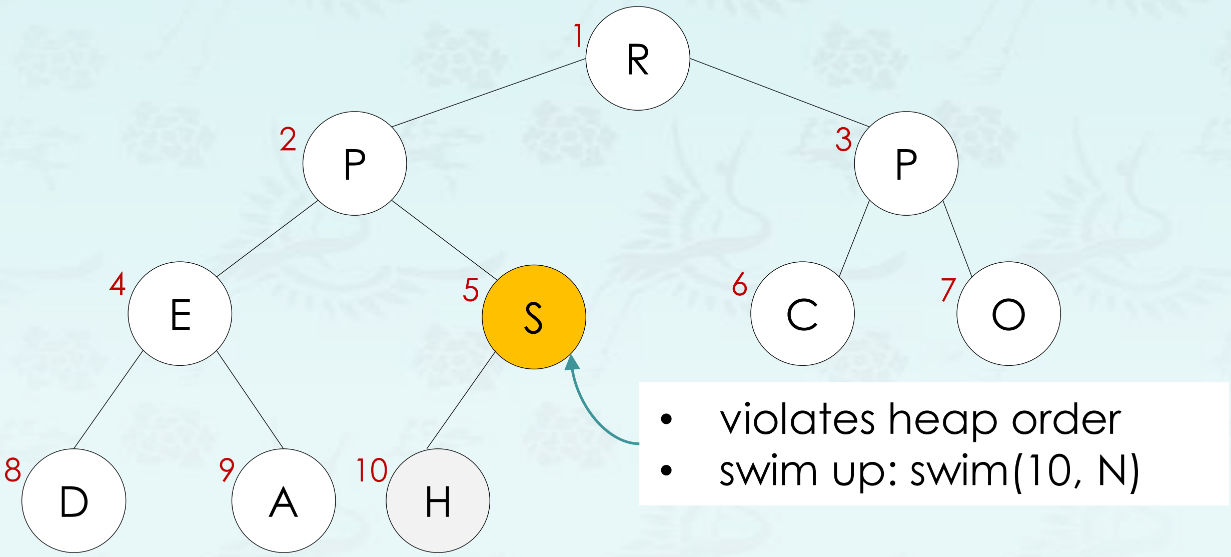
- violates heap order
- swim up: swim(10, N)

											h->N		
heap-ordered		R	P	P	E	H	C	O	D	A	S		
	0	1	2	3	4	5	6	7	8	9	10	11	

maxheap example

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

insert S



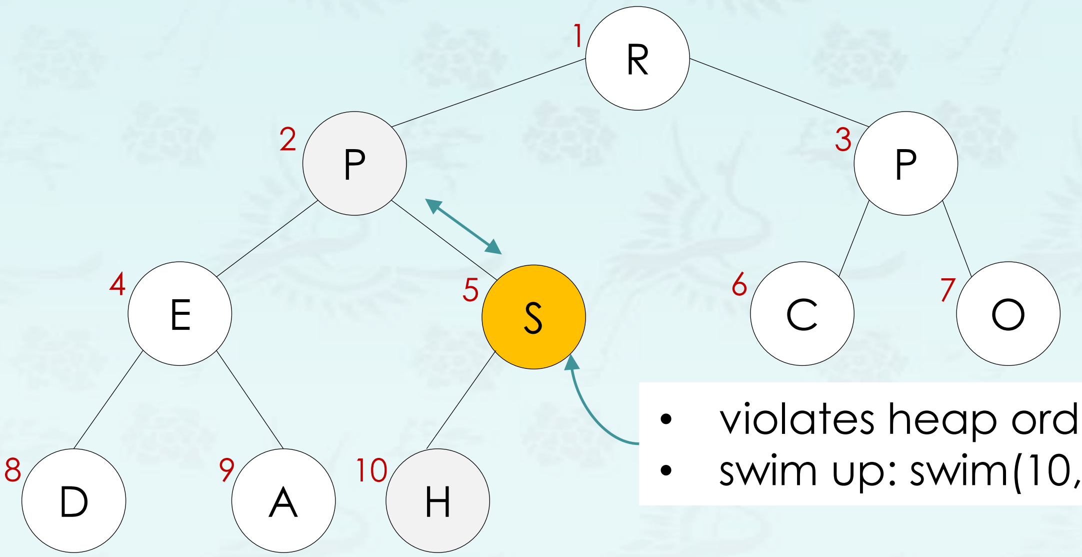
- violates heap order
- swim up: swim(10, N)

											h->N		
heap-ordered		R	P	P	E	S	C	O	D	A	H		
	0	1	2	3	4	5	6	7	8	9	10	11	

maxheap example

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

insert S



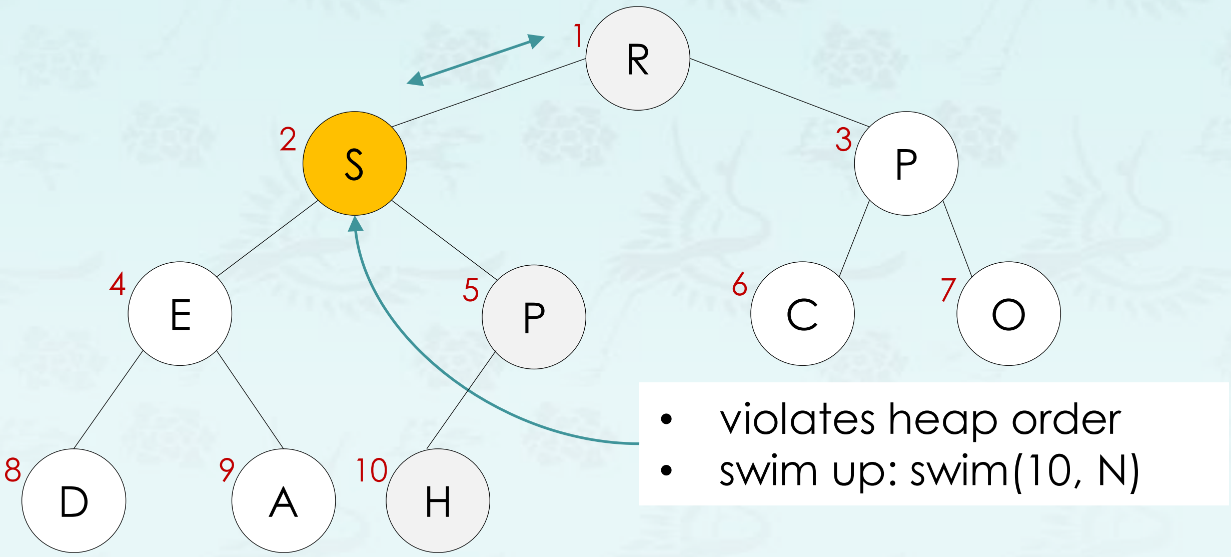
- violates heap order
- swim up: swim(10, N)

											h->N		
heap-ordered		R	P	P	E	S	C	O	D	A	H		
	0	1	2	3	4	5	6	7	8	9	10	11	

maxheap example

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

insert S



- violates heap order
- swim up: swim(10, N)

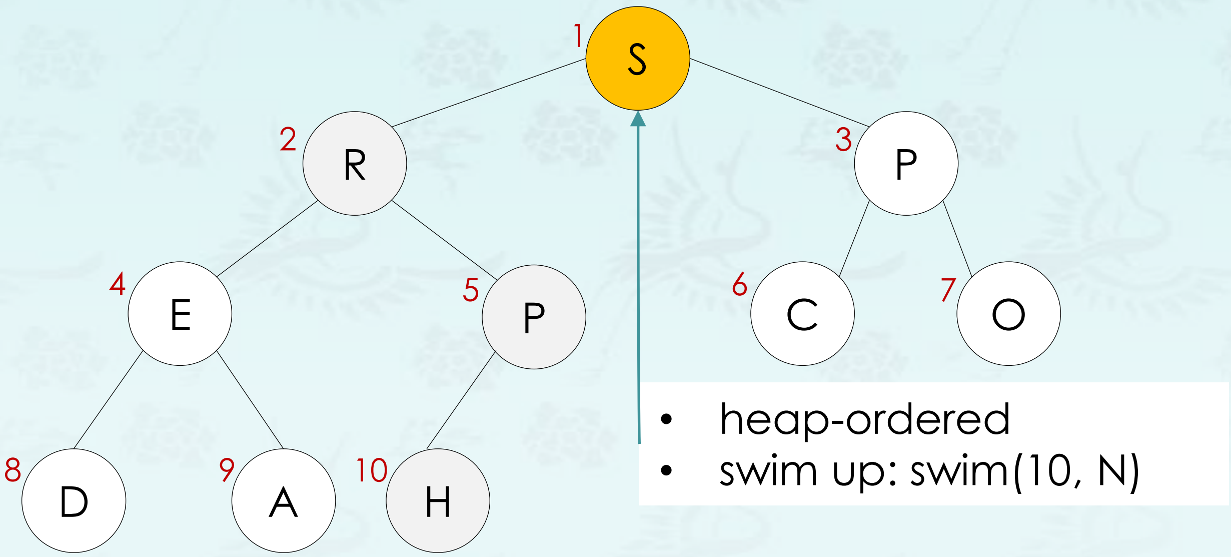
heap-ordered

									h->N		
	R	S	P	E	P	C	O	D	A	H	
0	1	2	3	4	5	6	7	8	9	10	11

maxheap example

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

insert S



- heap-ordered
- swim up: swim(10, N)

heap-ordered

										h->N	
	S	R	P	E	P	C	O	D	A	H	
0	1	2	3	4	5	6	7	8	9	10	11

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 in min/max heap
- decreaseKey: O(log N)
- increaseKey: O(log N)
- remove: O(log N)   // removing a node in any location

- **Heapify():** O(N)
- **Heapsort():** O(n log n)
-     Because O(N) heapify + O(n log n) remove nodes = O(n log n)
- **Proof:**
  - https://stackoverflow.com/questions/9755721/how-can-building-a-heap-be-on-time-complexity
  - https://www.growingwiththeweb.com/data-structures/binary-heap/build-heap-proof/
  - https://www.quora.com/How-is-the-time-complexity-of-building-a-heap-is-o-n
  - http://www.cs.umd.edu/~meesh/351/mount/lectures/lect14-heapsort-analysis-part.pdf

Binary heap operations time complexity with N items:

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

Mission Completed

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

Summary &

quaestio quaestio qo< ? ? ? ?

Data Structures
Chapter 5: Heap and Priority Queue

1. **introduction**
   - Complete Binary Tee (Review)
   - Heap and Priority Queue
2. Binary Heap
   - Min heap, Max heap
   - Priority Queue
3. Heapsort
4. Heap & PQ Coding