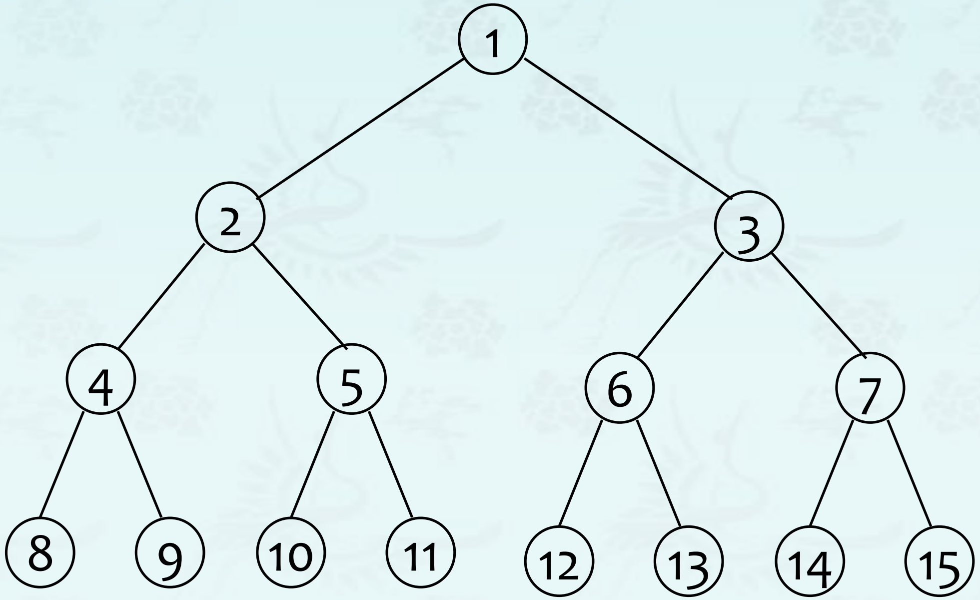
heap

- *complete binary tree (review)*
- *heap and priority queues (Chapter 9)*
- *binary heap and minheap*
- *maxheap demo*
- *maxheap coding*
- *heap sort (Chapter 7)*

Binary Trees – Properties

Definition: *A **full** binary tree* of $\boldsymbol{level\ k}$ is a binary tree having **2^k – 1 nodes**, $k \geq 0$.

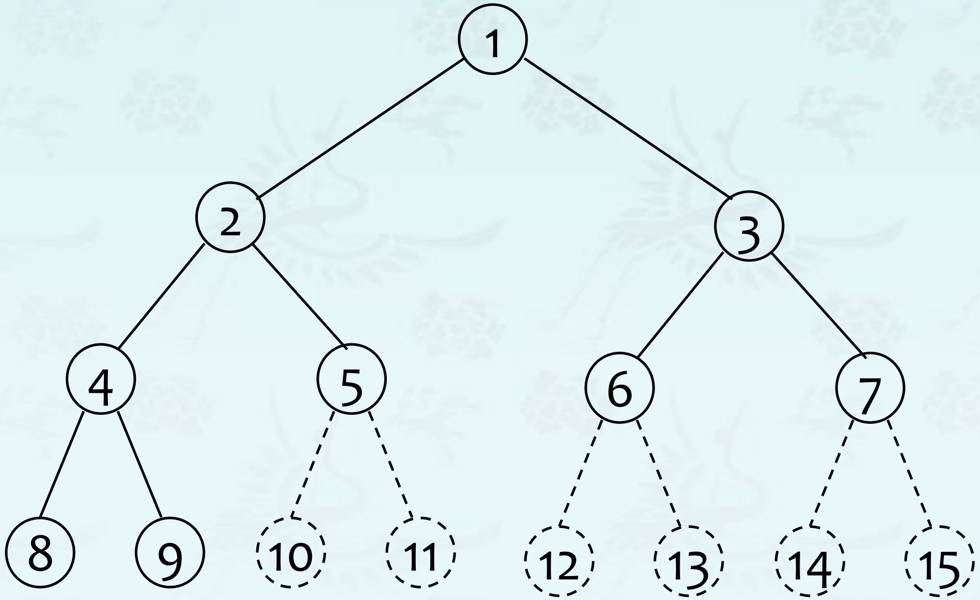


*A **full** binary tree*

Binary Trees – Properties

Definition: A ***full*** *binary tree* of $\boldsymbol{level\ k}$ 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 $level\ k$.

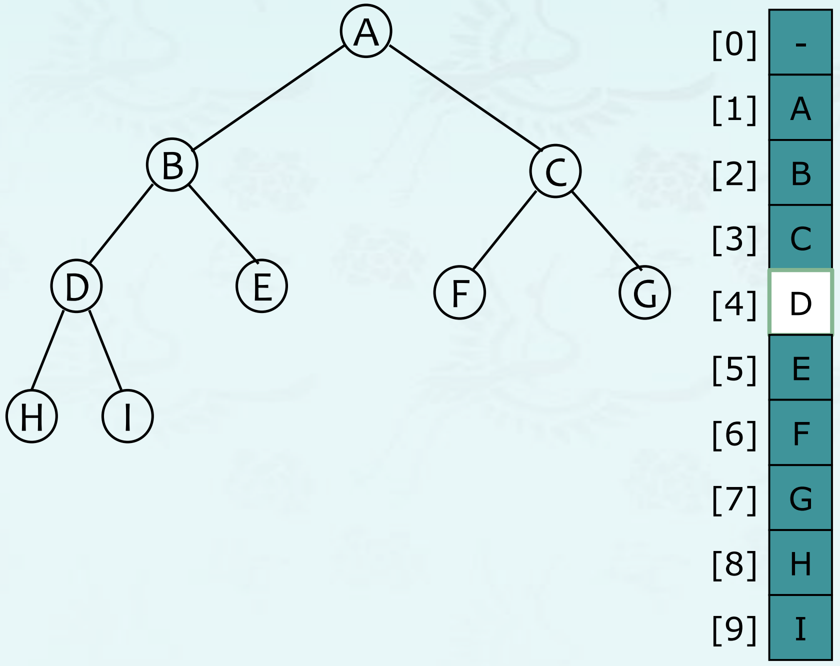


*A **complete** binary tree*

Binary Trees – Array representation

Property: a **complete** binary tree with n nodes, any node index i, $1 \leq \ i \ \leq n$, we have

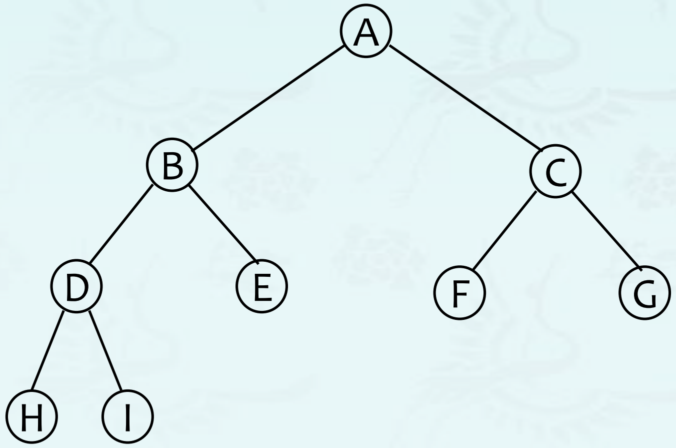
(1) $parent(i)$ is at $\boxed{\lfloor i/2 \rfloor}$ $if\ i\ =\ 1.$ $If\ i\ =\ 1,\ i$ is at the root and has no parent.
(2) $leftChild(i)$ is at $\mathbf{2i}$ $if\ \ 2i\ <=\ n.$ $If\ 2i\ >\ n,$ then i has no left child.
(3) $rightChild(i)$ is at $\mathbf{2i+1}$ $if\ \ 2i\ +\ 1\ \ <=\ n.$ $If\ 2i\ +\ 1\ >\ n,$ then i has no right child.



Binary Trees – Array representation

Property: a **complete** binary tree with n nodes, any node index i, $1 \leq i \leq n$, we have

(1) $parent(i)$ is at $\boxed{\lfloor i/2 \rfloor}$ $if\ i\ =\ 1.$ $If\ i\ =\ 1,\ i$ is at the root and has no parent.
(2) $leftChild(i)$ is at $\mathbf{2i}$ $if\ 2i\ <=\ n.$ $If\ 2i\ >\ n,$ then i has no left child.
(3) $rightChild(i)$ is at $\mathbf{2i + 1}$ $if\ 2i\ +\ 1\ <=\ n.$ $If\ 2i\ +\ 1\ >\ n,$ then i has no right child.



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

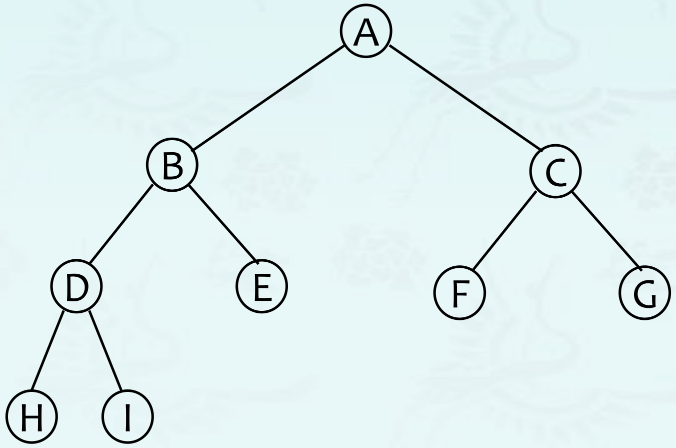
Example:
Find its parent, left child and right child at node D

Solution:

Binary Trees – Array representation

Property: a **complete** binary tree with n nodes, any node index i, $1 \leq i \leq n$, we have

(1) $parent(i)$ is at $\boxed{\lfloor i/2 \rfloor}$ $if\ i = 1.$ $If\ i = 1,$ i is at the root and has no parent.
(2) $leftChild(i)$ is at $\mathbf{2i}$ $if\ 2i <= n.$ $If\ 2i > n,$ then i has no left child.
(3) $rightChild(i)$ is at $\mathbf{2i+1}$ $if\ 2i+1 <= n.$ $If\ 2i+1 > n,$ then i has no right child.



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

Example:
Find its parent, left child and right child at node D

Solution:
$parent(i = 4)$ is at 4/2 = 2
$leftChild(4)$ is at 2x4 = 8
$rightChild(4)$ is at 2x4 + 1 = 9

How do you like this property of the tree?

heap

- ***complete binary tree (review)***
- ***heap and priority queues (Chapter 9)***
- *binary heap and minheap*
- *heap coding*
- *heap sort (Chapter 7)*

Heaps & Priority Queues

Heaps are frequently used to implement **priority queues.**

- Because it provides an efficient implementation for **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 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

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

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

Operation	Complexity
Get the top priority element (min or max)	O(1)
Insert an element	O(log n)
Delete the top priority element	O(log n)
Decrease the priority of an element	O(log n)

Priority queue applications

- Event-driven simulation. [customers in a line, colliding particles]
- Numerical computation. [reducing roundoff 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]

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.

N huge,
M large

Constraints: Not enough memory to store N items.

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.

N huge,
M large

Constraints: Not enough memory to store N items.

%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

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

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

Mission Impossible?

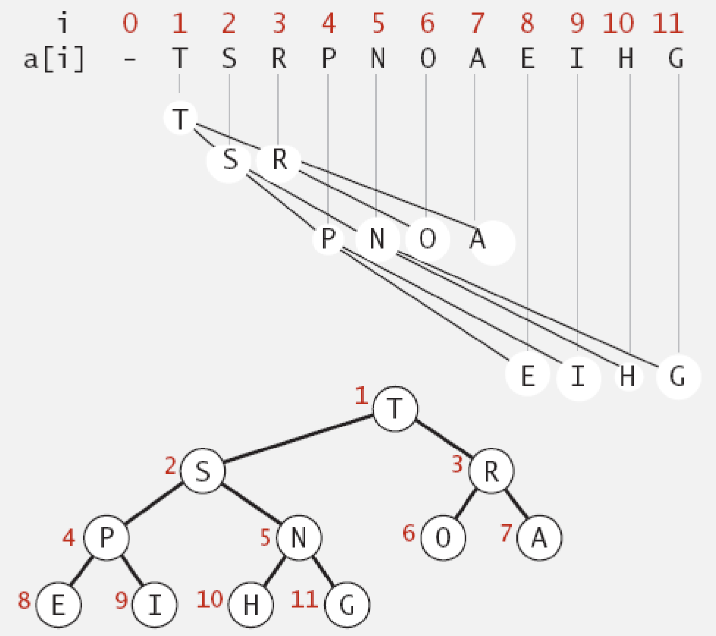
heap

- complete binary tree (review)
- heap and priority queues (Chapter 9)
- binary heap and minheap
- maxheap demo
- maxheap coding
- heap sort (Chapter 7)

Binary heap: array representation of a **heap-ordered** complete binary tree

- **Properties:**

- Array representation

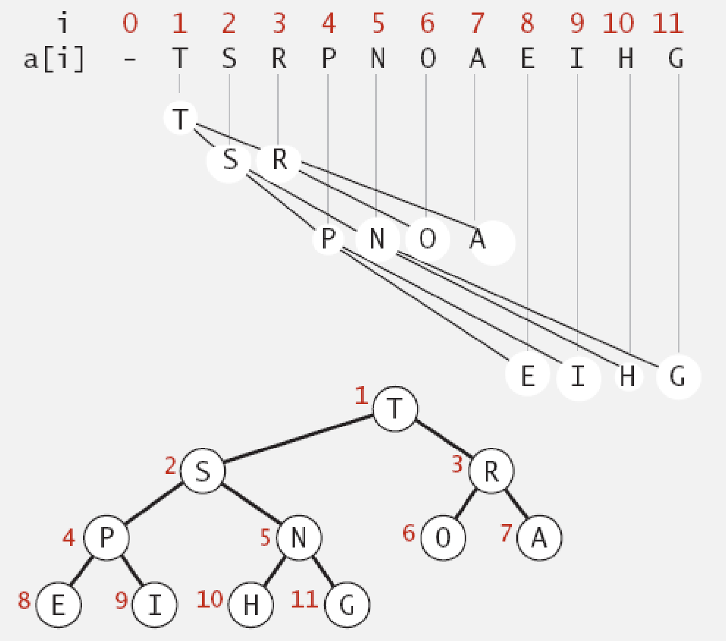


Heap representations

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



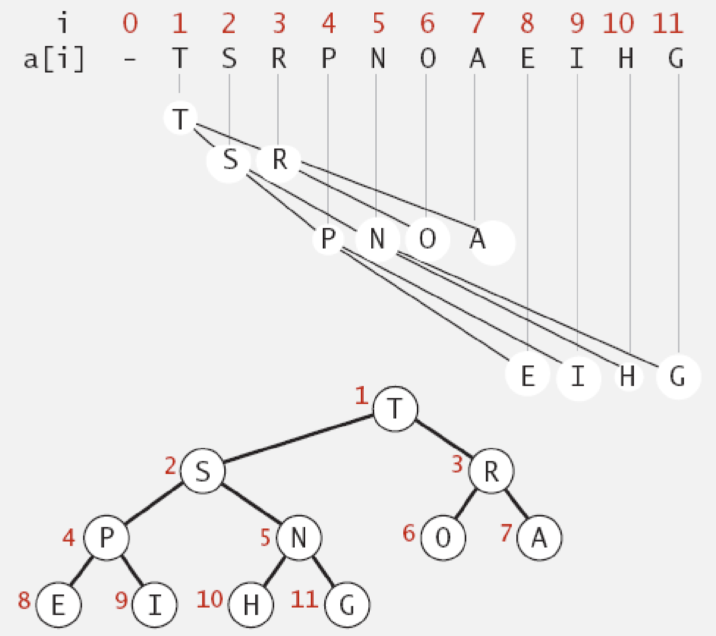
Heap representations

Heaps & Priority Queues

Binary heap: array representation of a **heap-ordered** complete binary tree

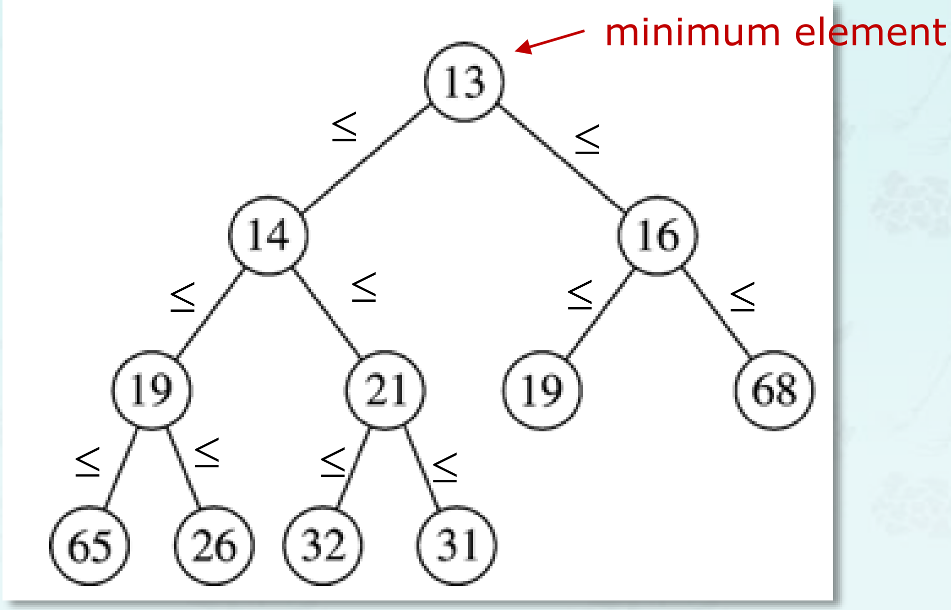
- **Properties:**
  - **Heap-ordered:**
    Parent's key no smaller than children's keys. [maxheap]
  - **Heap-structure**:
    A complete binary tree

- 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!



Heap representations

minheap example



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

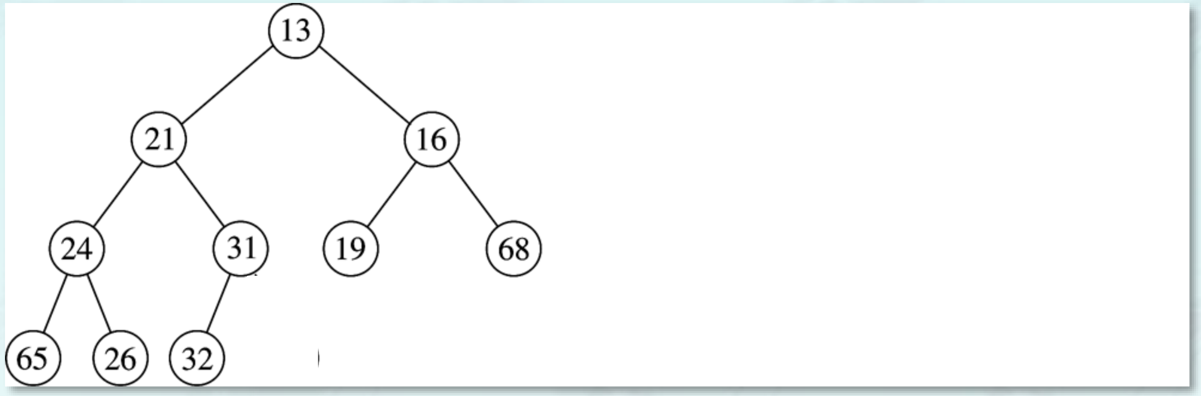
minheap example

insertion:

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

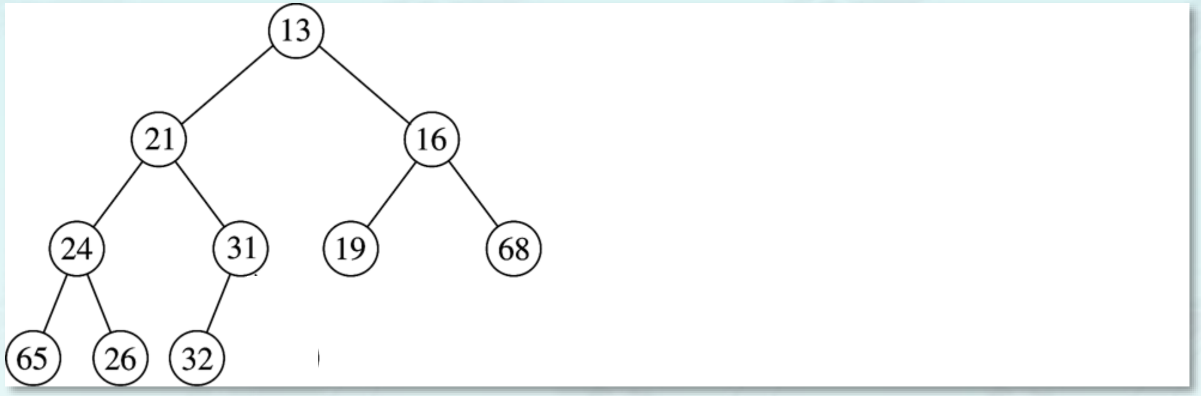
minheap example

insertion: Insert a node 14



minheap example

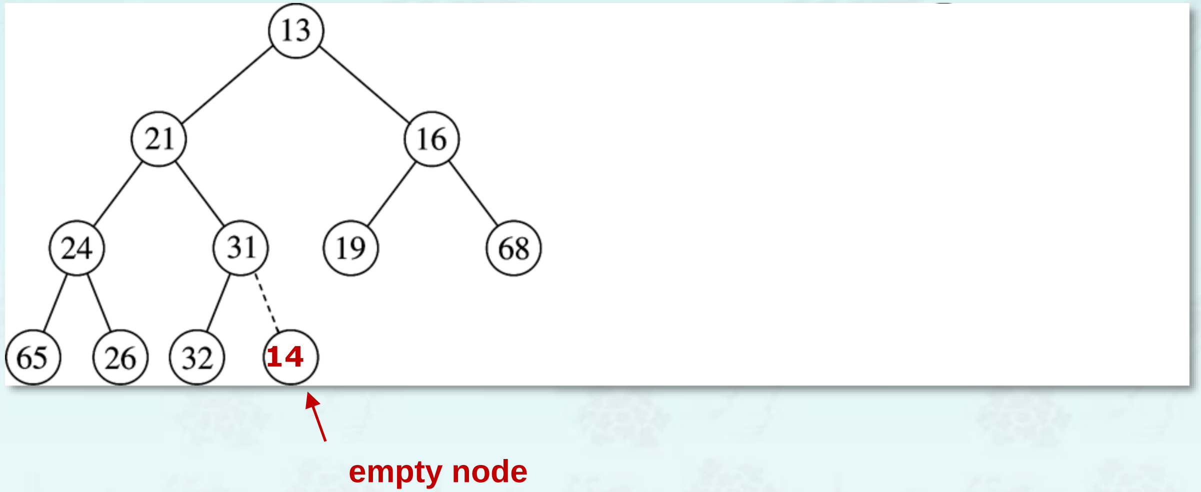
insertion: Insert a node 14   Where is an empty node to start?



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

minheap example

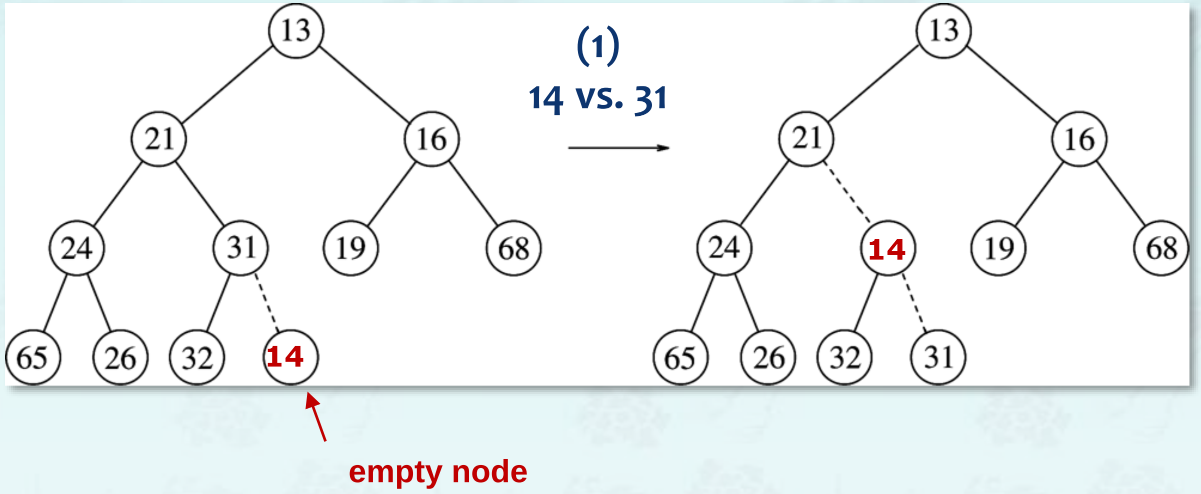
insertion: Insert a node 14



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

minheap example

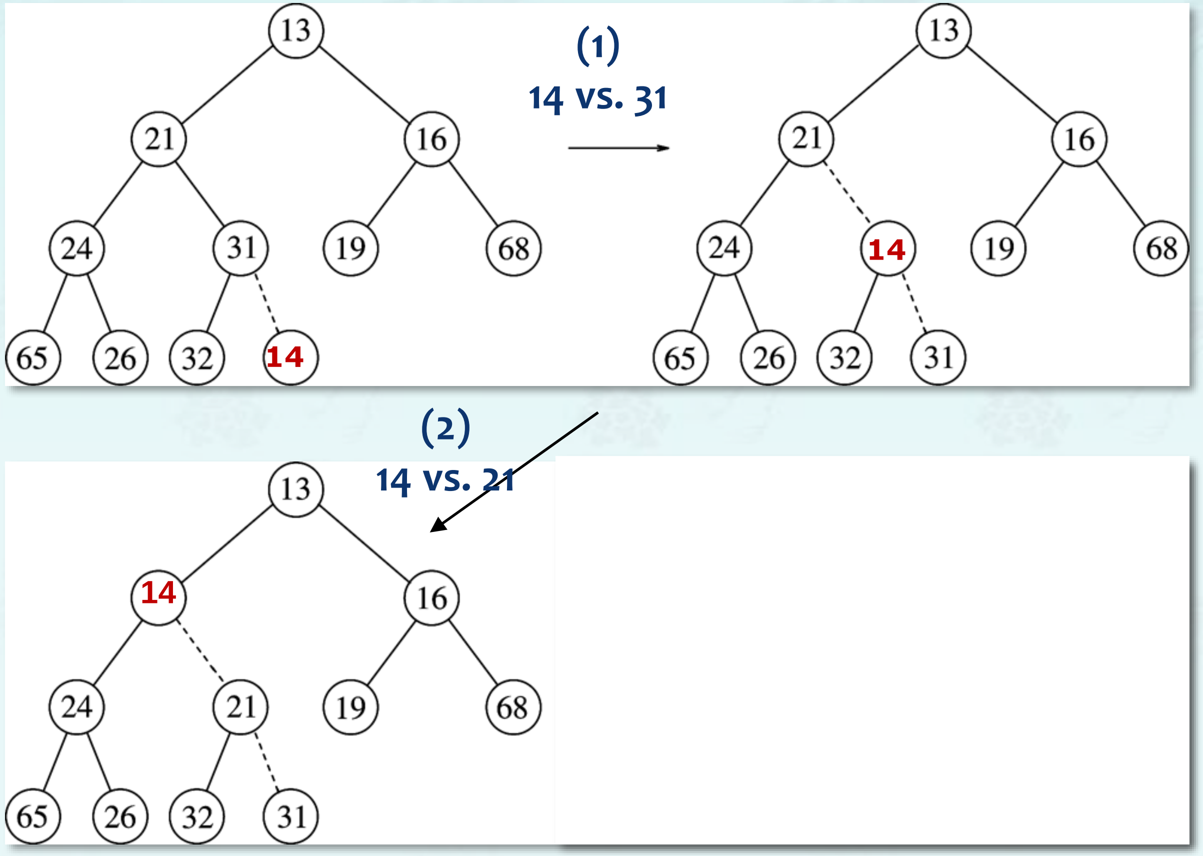
insertion: Insert a node 14



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

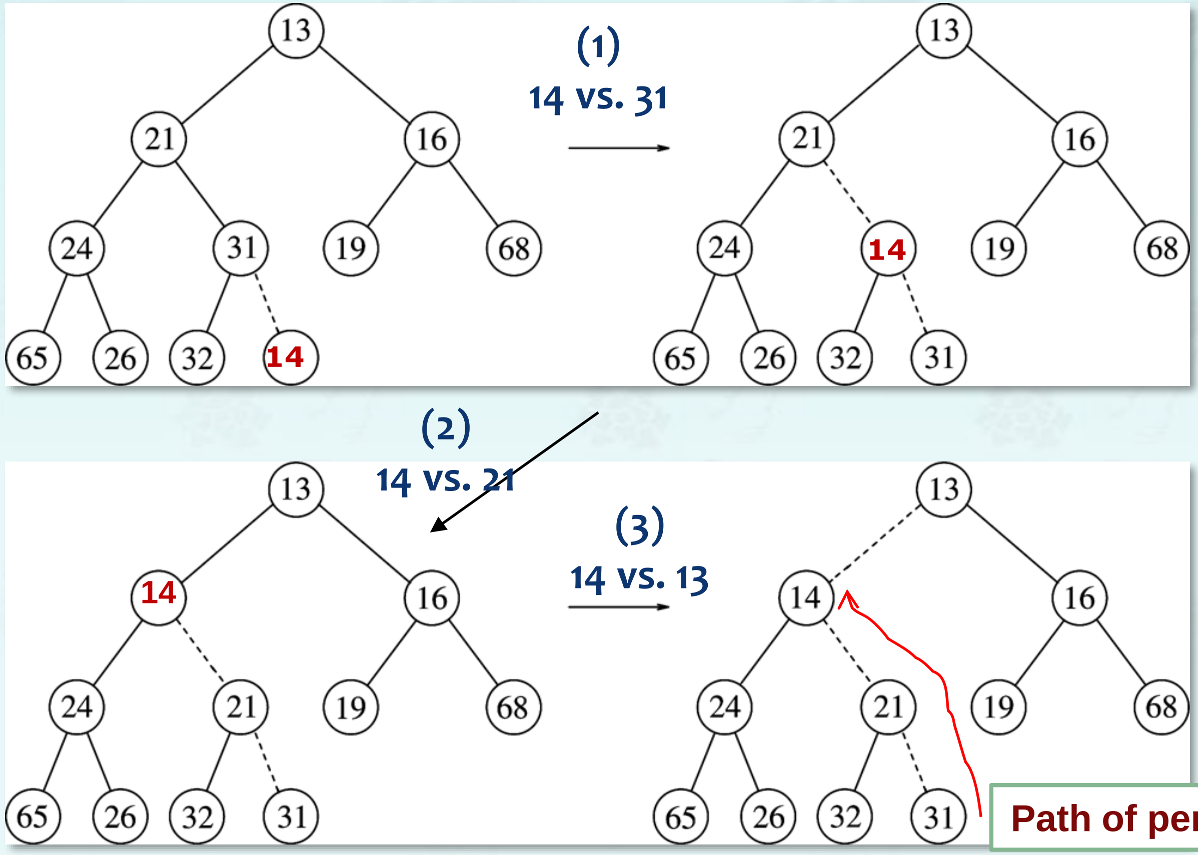
minheap example

insertion: Insert a node 14



minheap example

insertion: Insert a node 14



Heap-ordered

Heap-Structure

Path of percolation (swim) up

minheap example

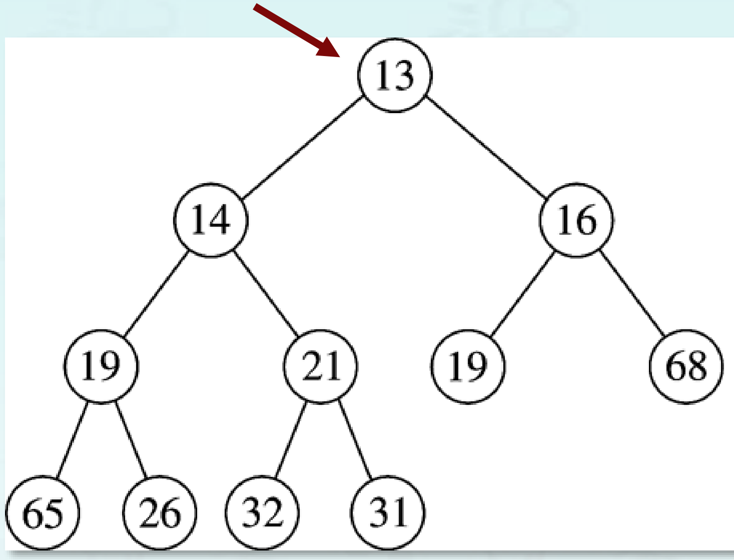
deletion: dequeue – delete the root

- Swap the root and the 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 minheap definition).

minheap example

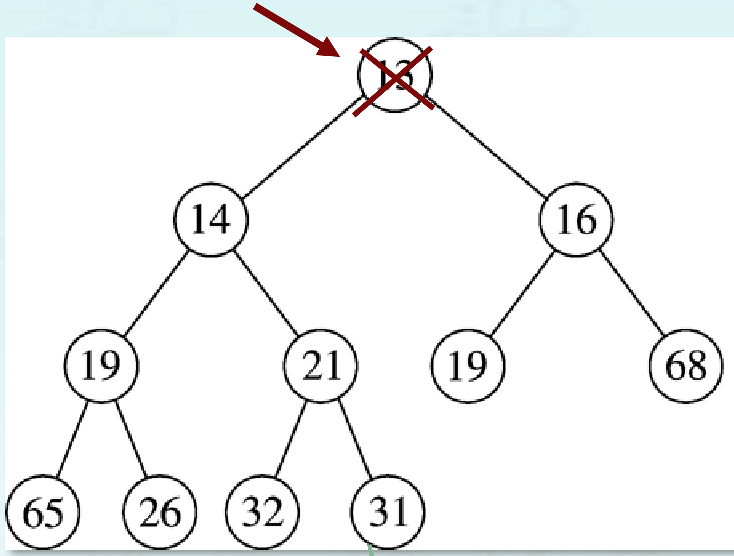
deletion: dequeue – delete the root

Which position of the node will be empty?



minheap example

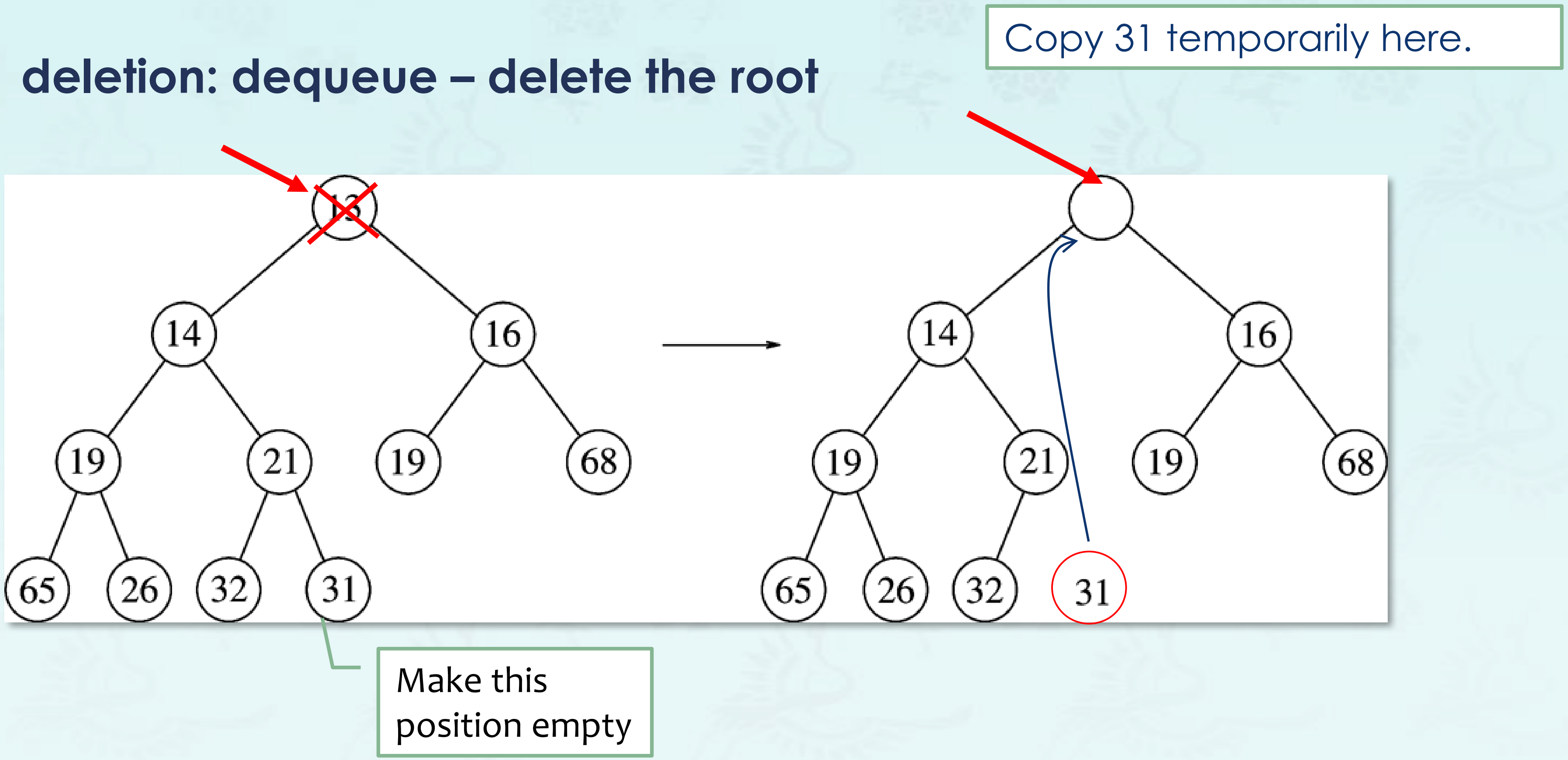
deletion: dequeue – delete the root



Make this position empty

minheap example

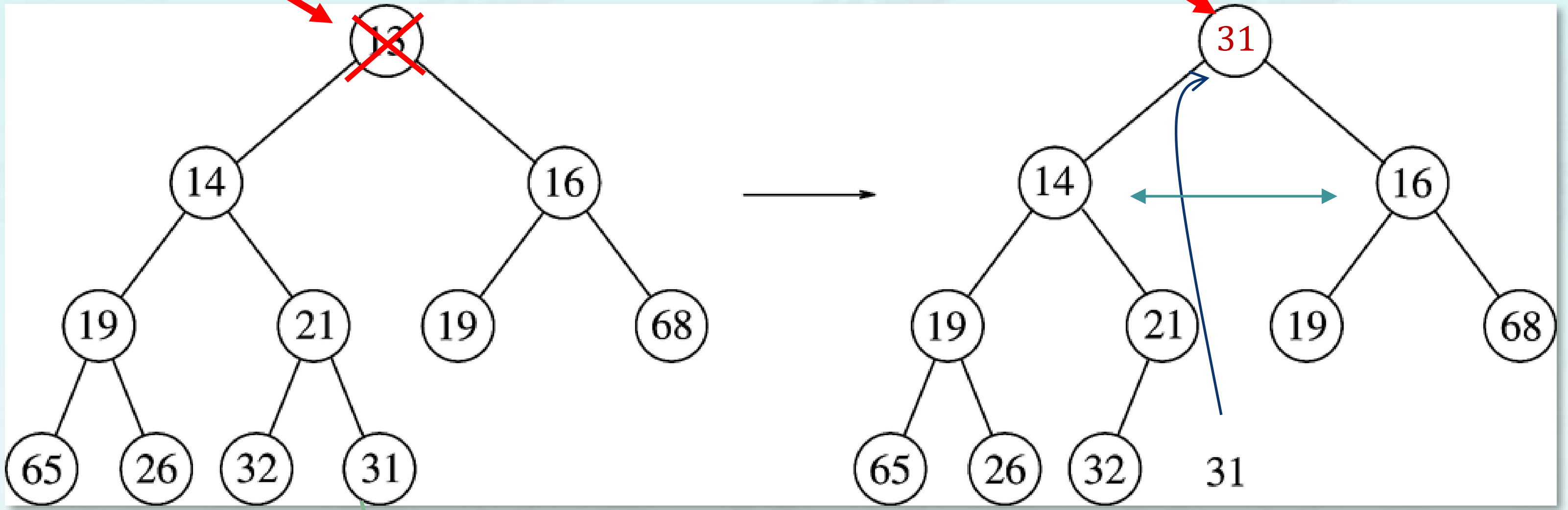
deletion: dequeue – delete the root



minheap example

deletion: dequeue – delete the root

Copy 31 temporarily here and ask **heap-ordered?**



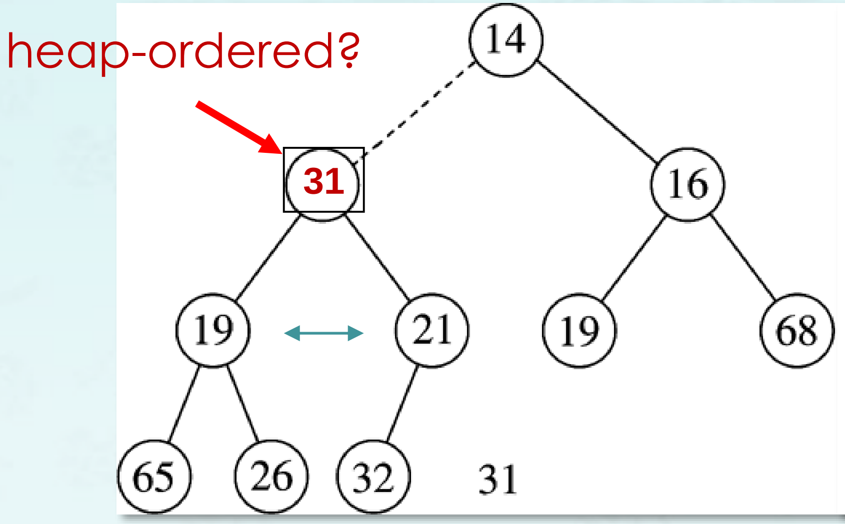
Make this position empty

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

sink…

minheap example

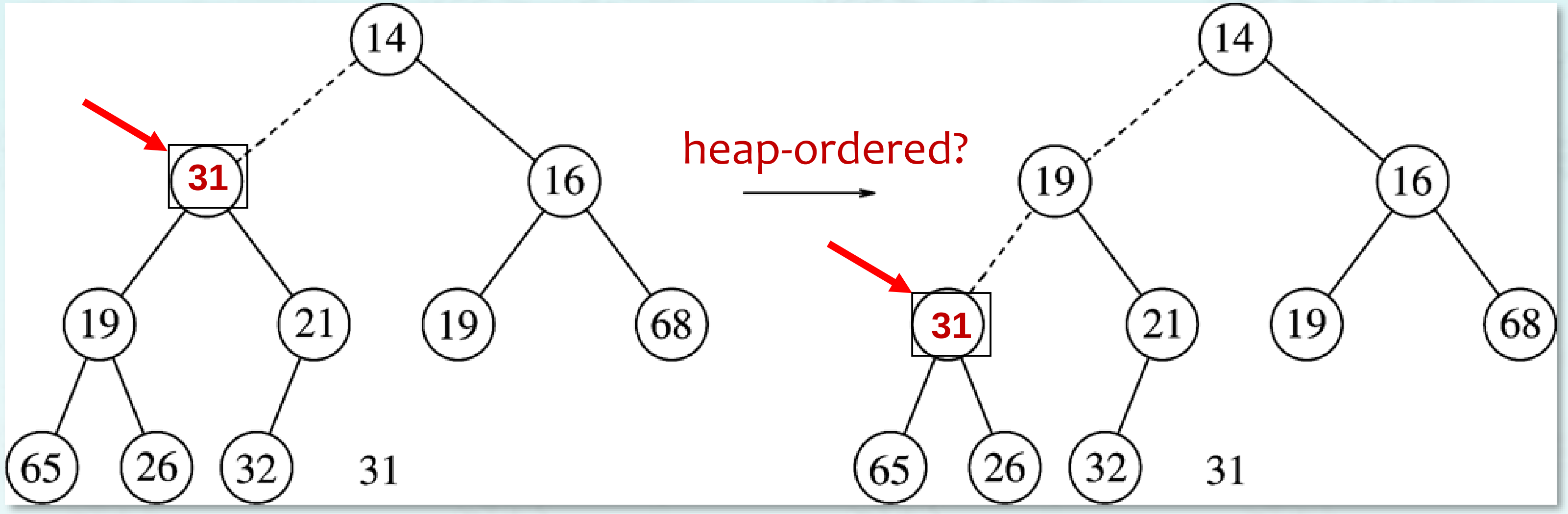
deletion: dequeue – delete the root



Is 31 > min(19,21)?

- Yes - swap 31 with min(19,21)

deletion: dequeue – delete the root

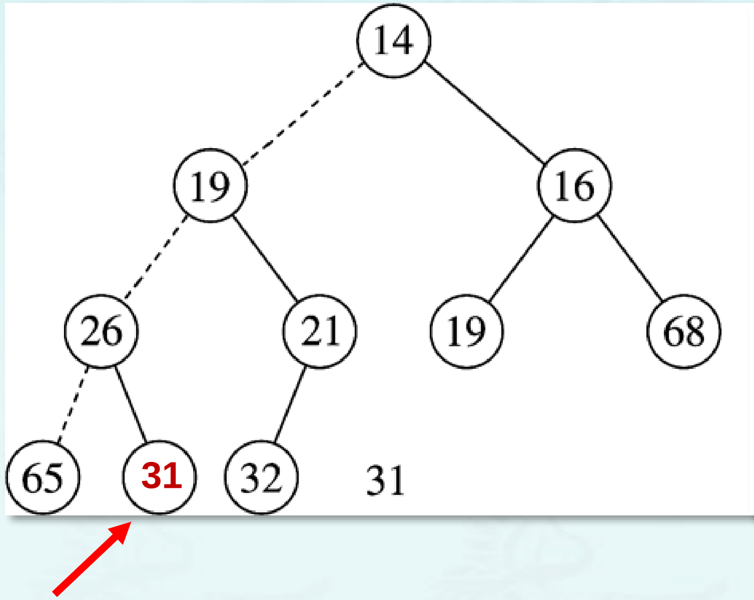


Is 31 > min(19,21)?
- Yes - swap 31 with min(19,21)

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

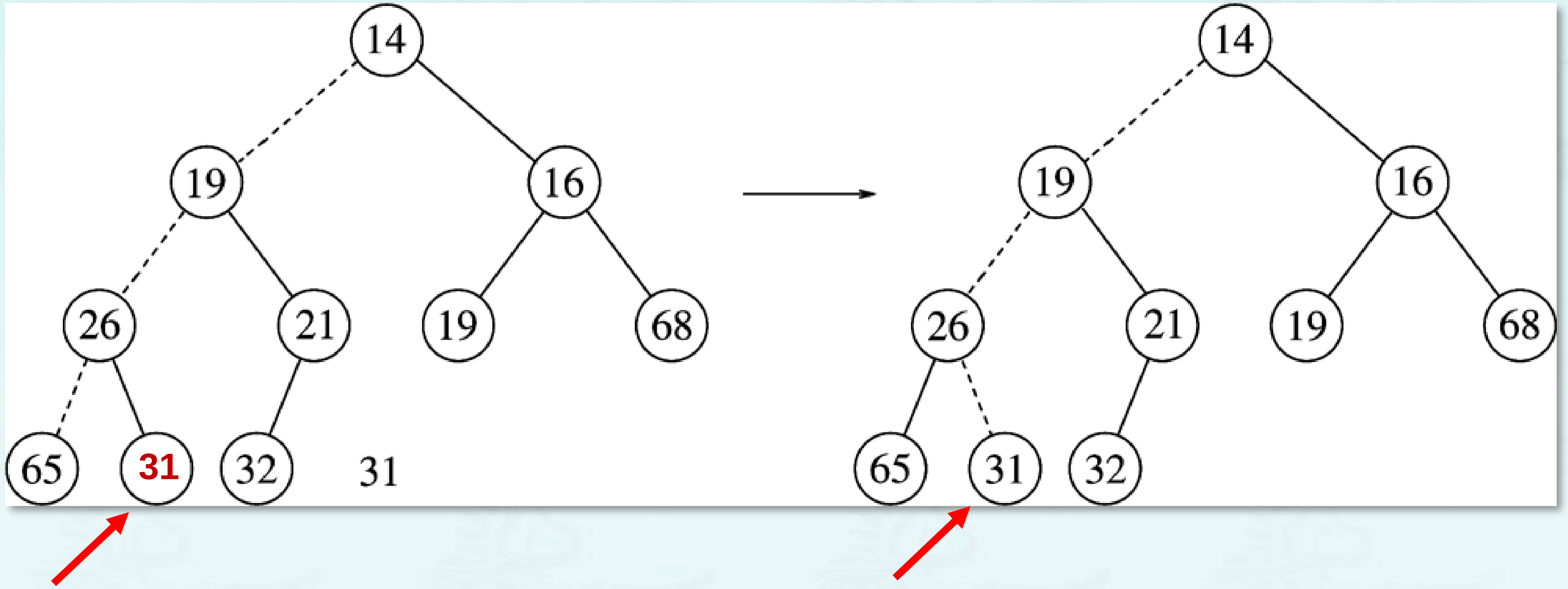
minheap example

deletion: dequeue – delete the root



minheap example

deletion: dequeue – delete the root



✓ 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

heap

- complete binary tree (review)
- heap and priority queues (Chapter 9)
- binary heap and minheap
- maxheap demo
- maxheap coding
- heap sort (Chapter 7)

maxheap example

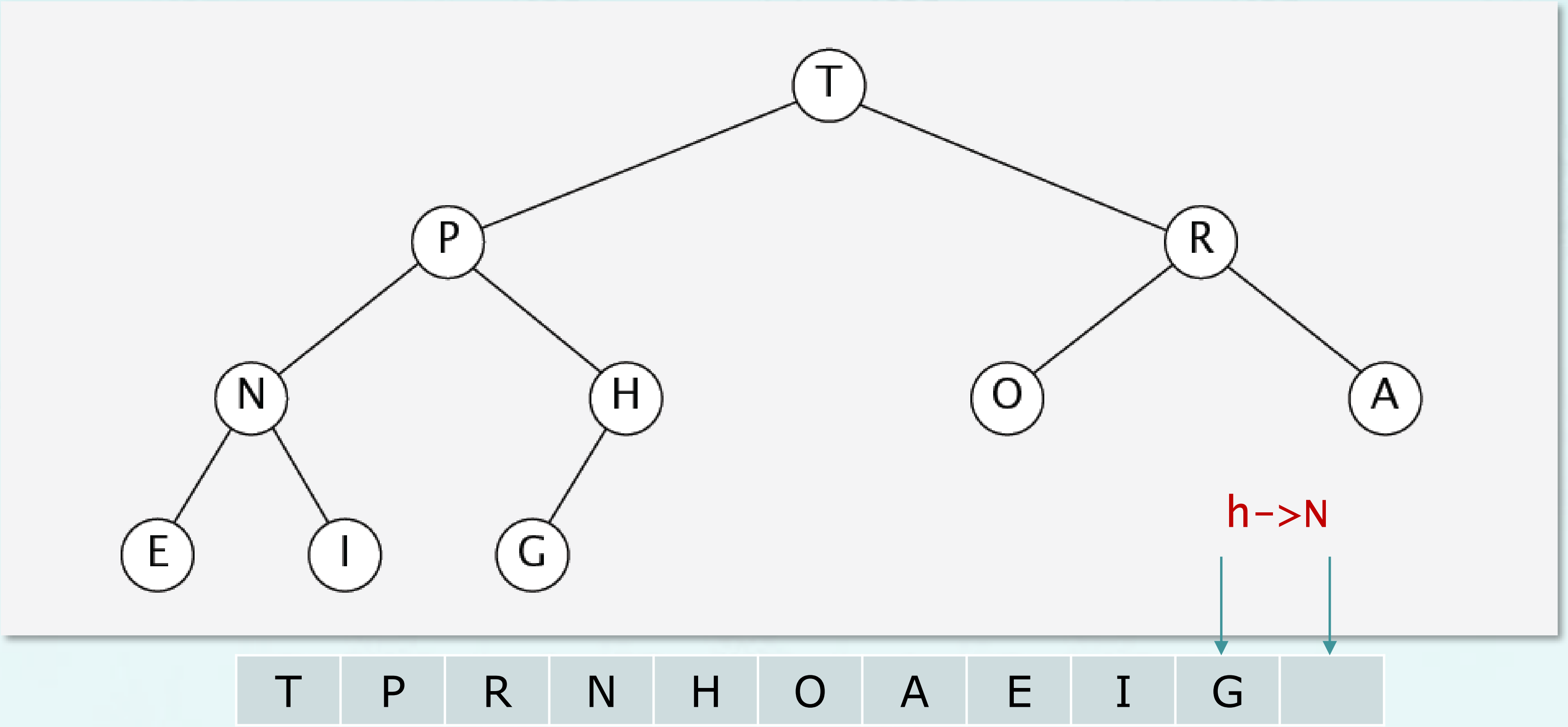
- **Insert**: Add node at end, then swim it up.

T	P	R	N	H	O	A	E	I	G	

maxheap example

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

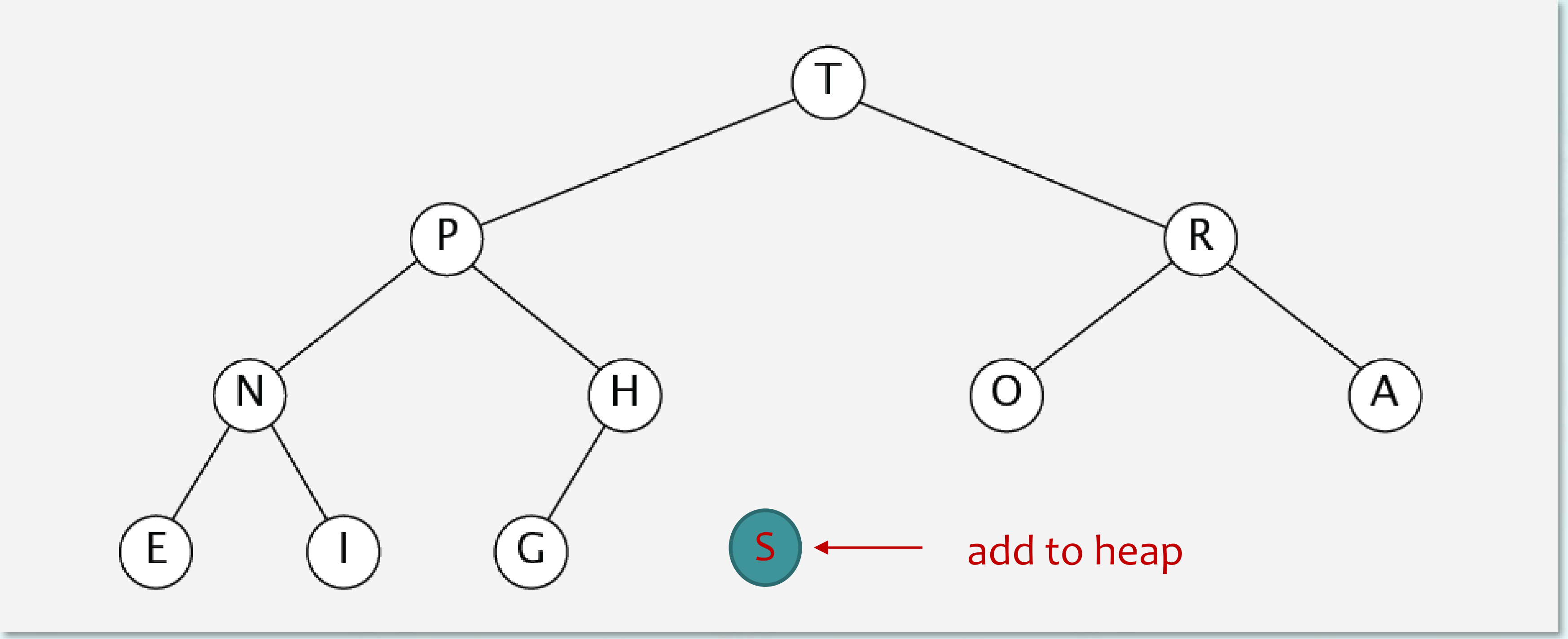
Heap ordered



maxheap example

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

insert S



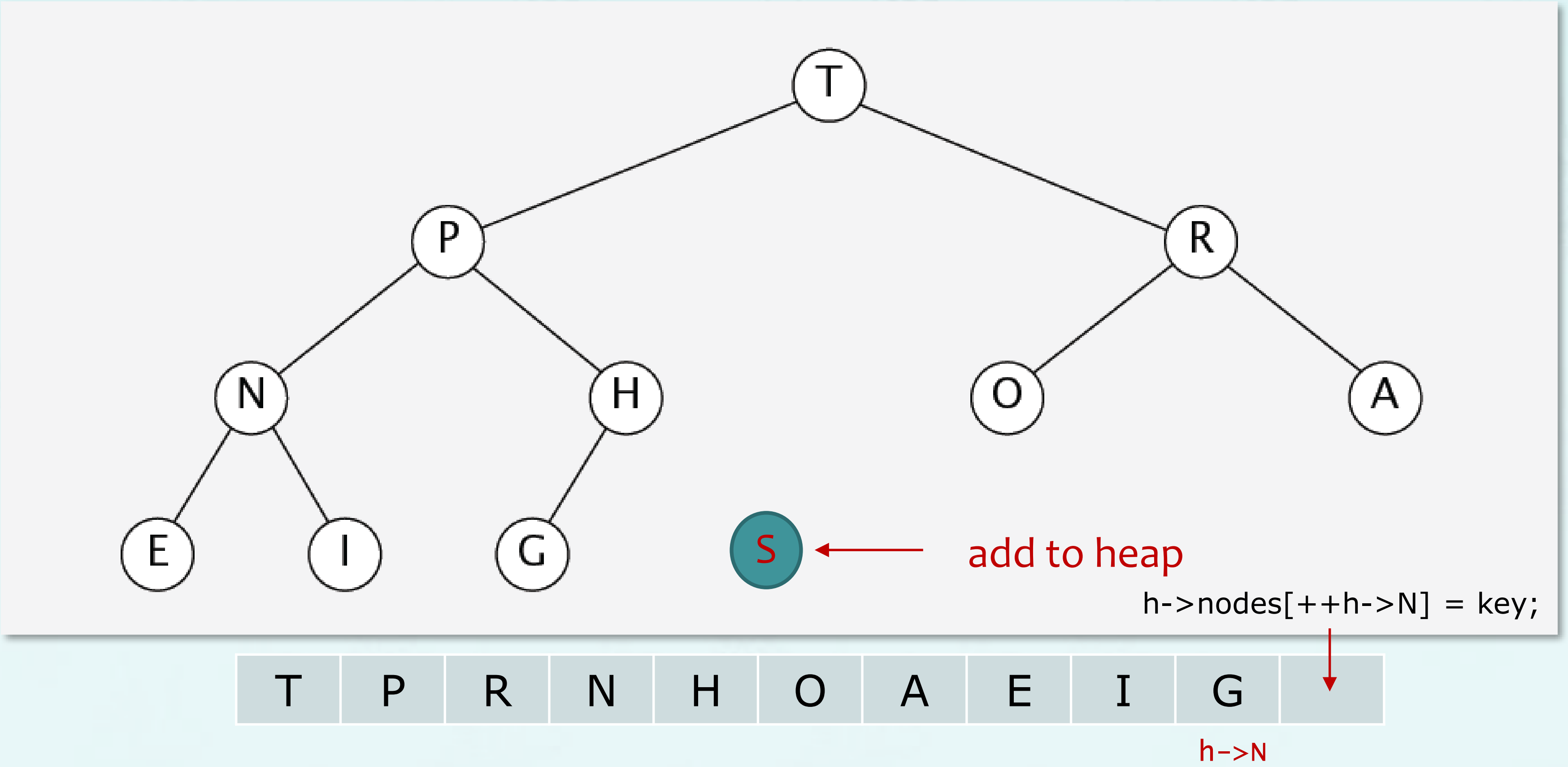
T	P	R	N	H	O	A	E	I	G	

h->N

maxheap example

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

insert S

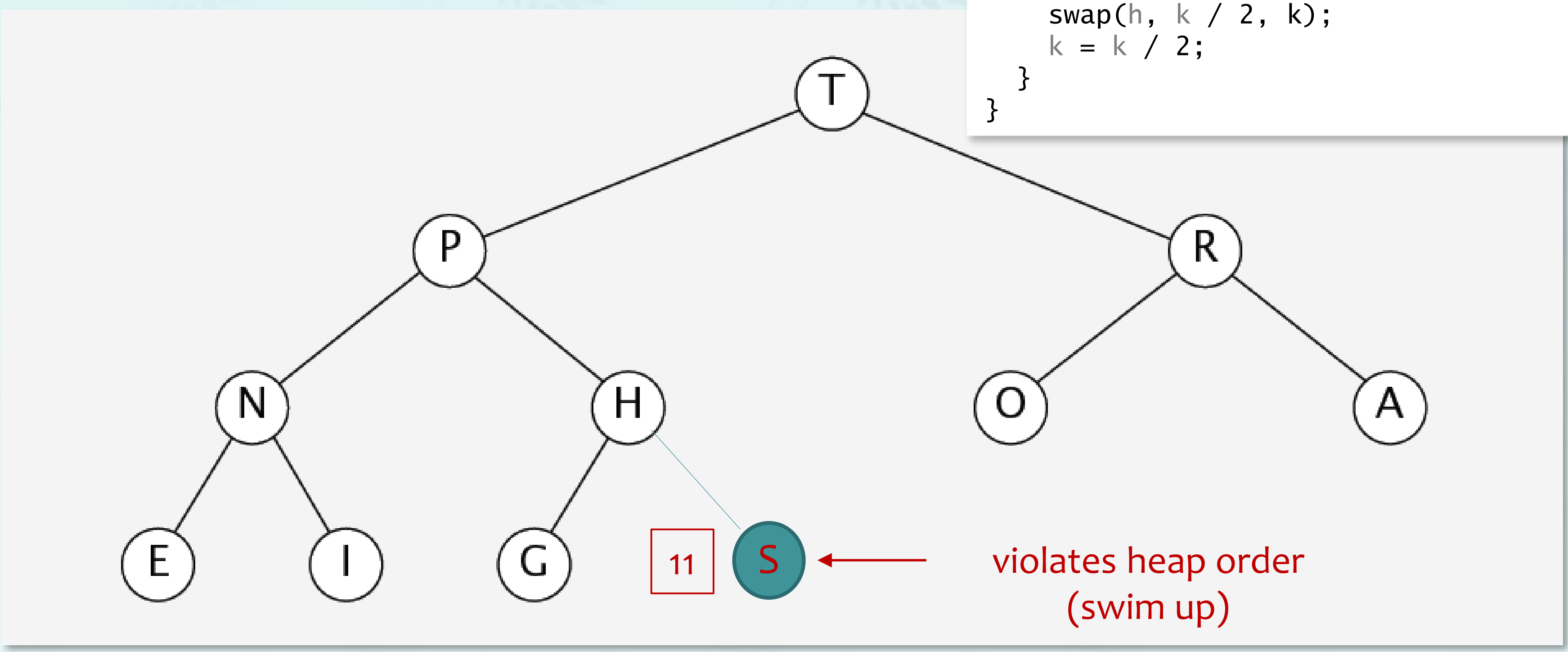


maxheap example

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

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

insert S



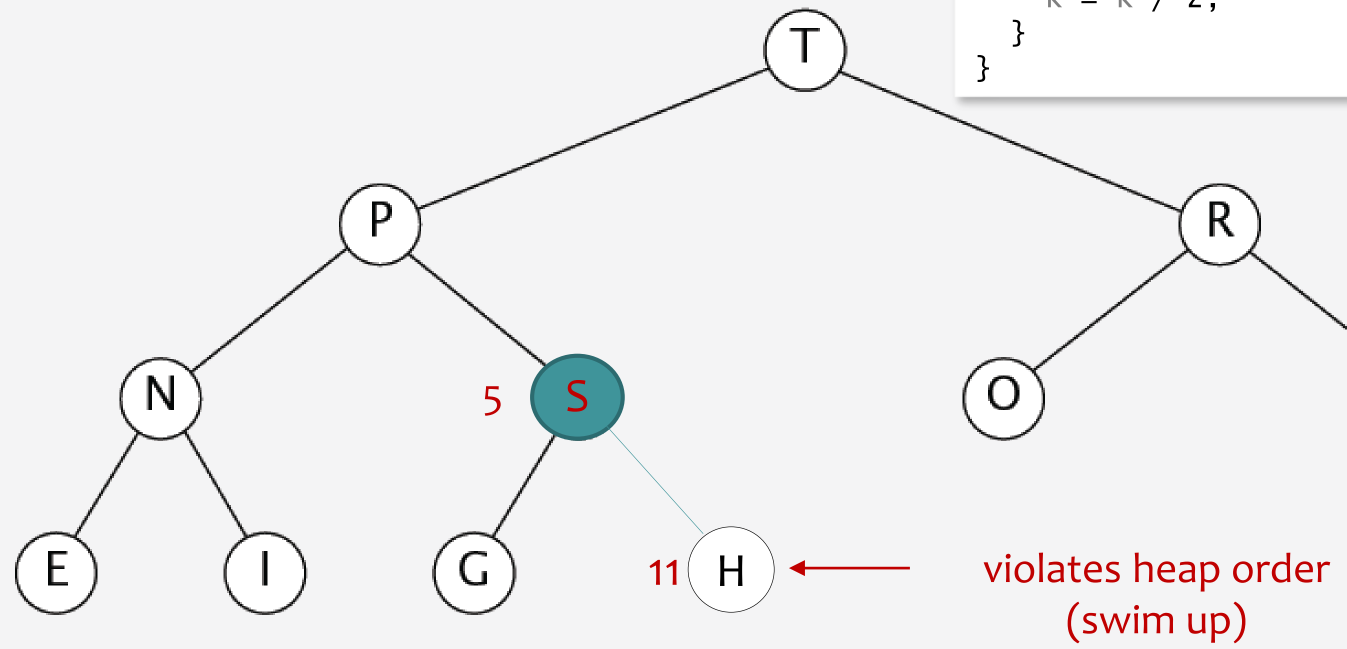
T	P	R	N	H	O	A	E	I	G	S
										11

maxheap example

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

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

insert S



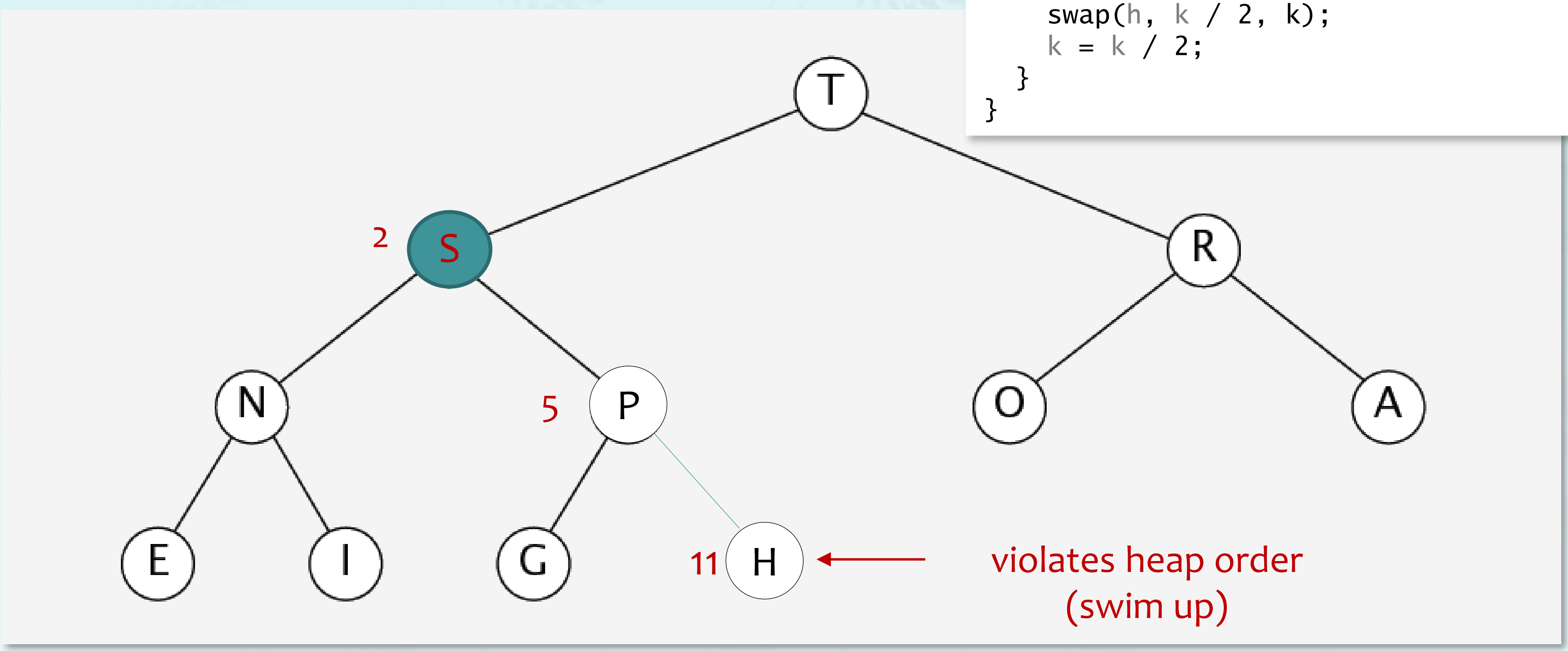
T	P	R	N	S	O	A	E	I	G	H
				5						11

maxheap example

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

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

insert S

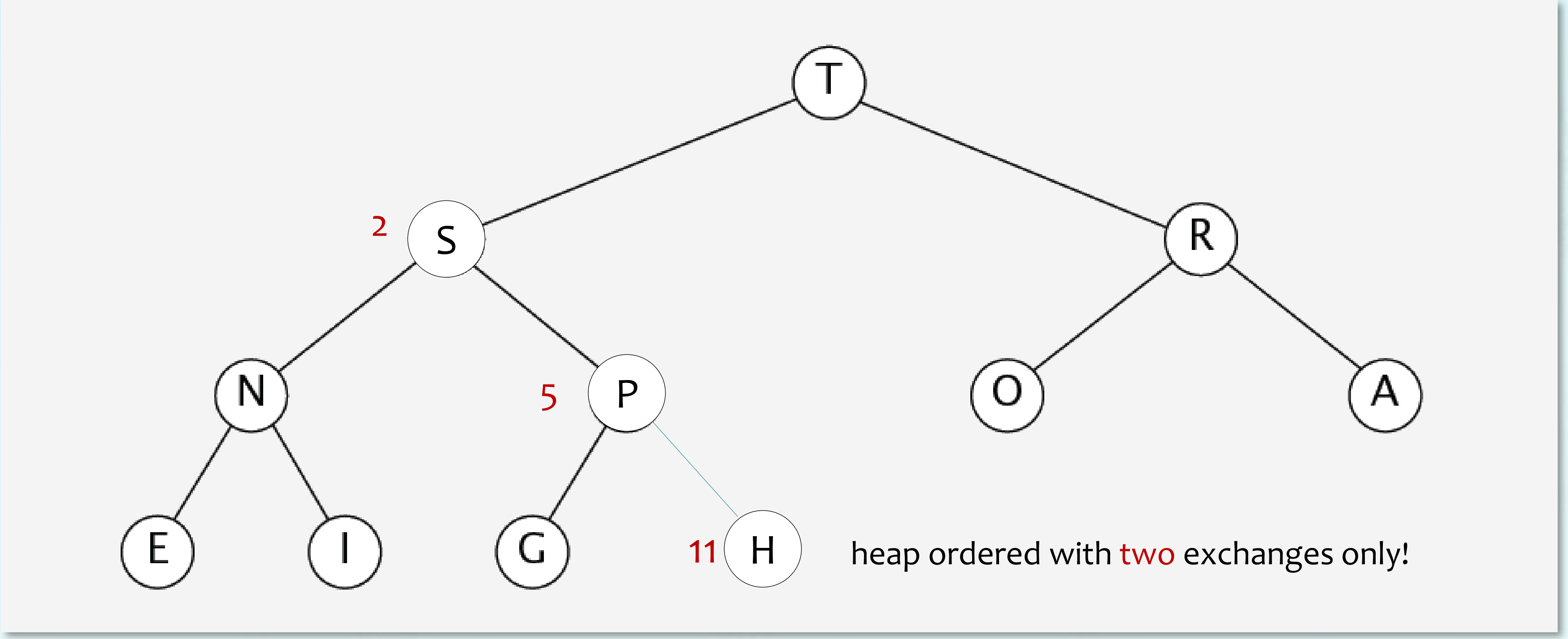


T	S	R	N	P	O	A	E	I	G	H
	2			5						11

maxheap example

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

heap ordered



T	S	R	N	P	O	A	E	I	G	H
	2			5						11

maxheap example

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

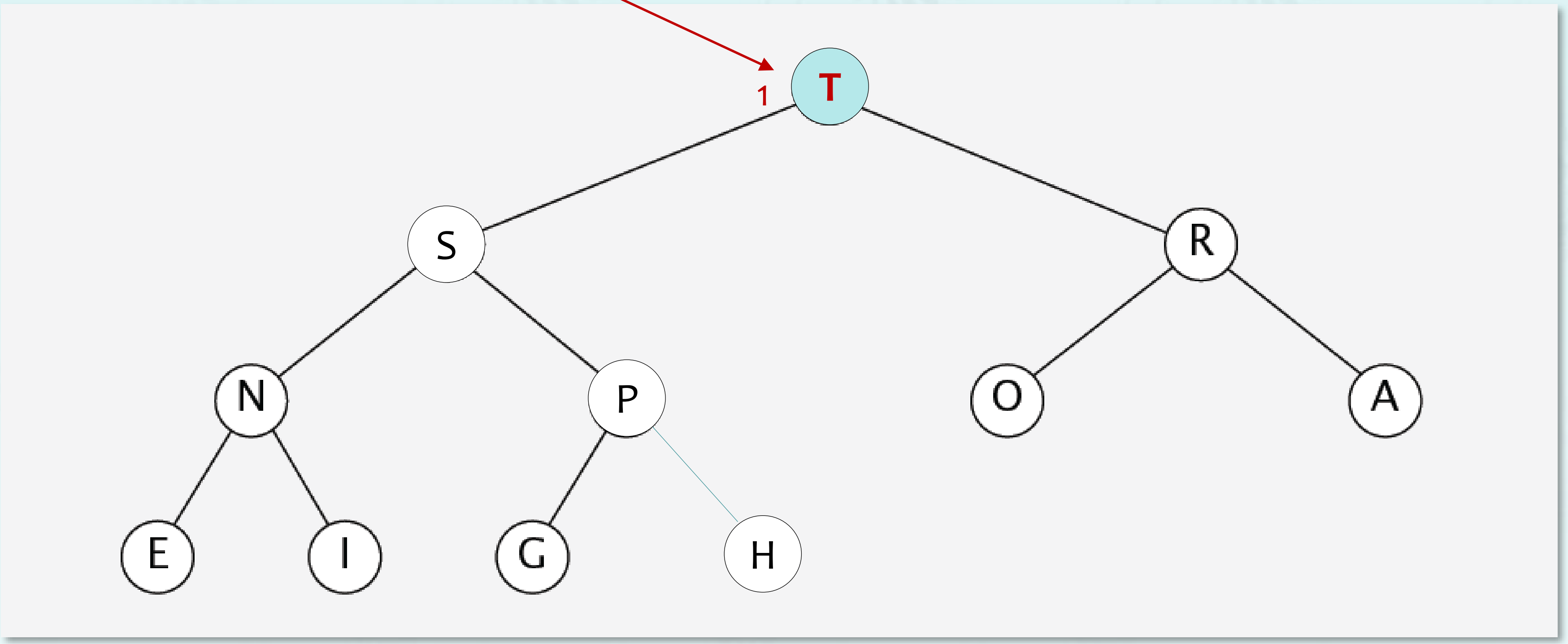
remove the maximum(root)

T	S	R	N	P	O	A	E	I	G	H
1										

maxheap example

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

remove the maximum(root)



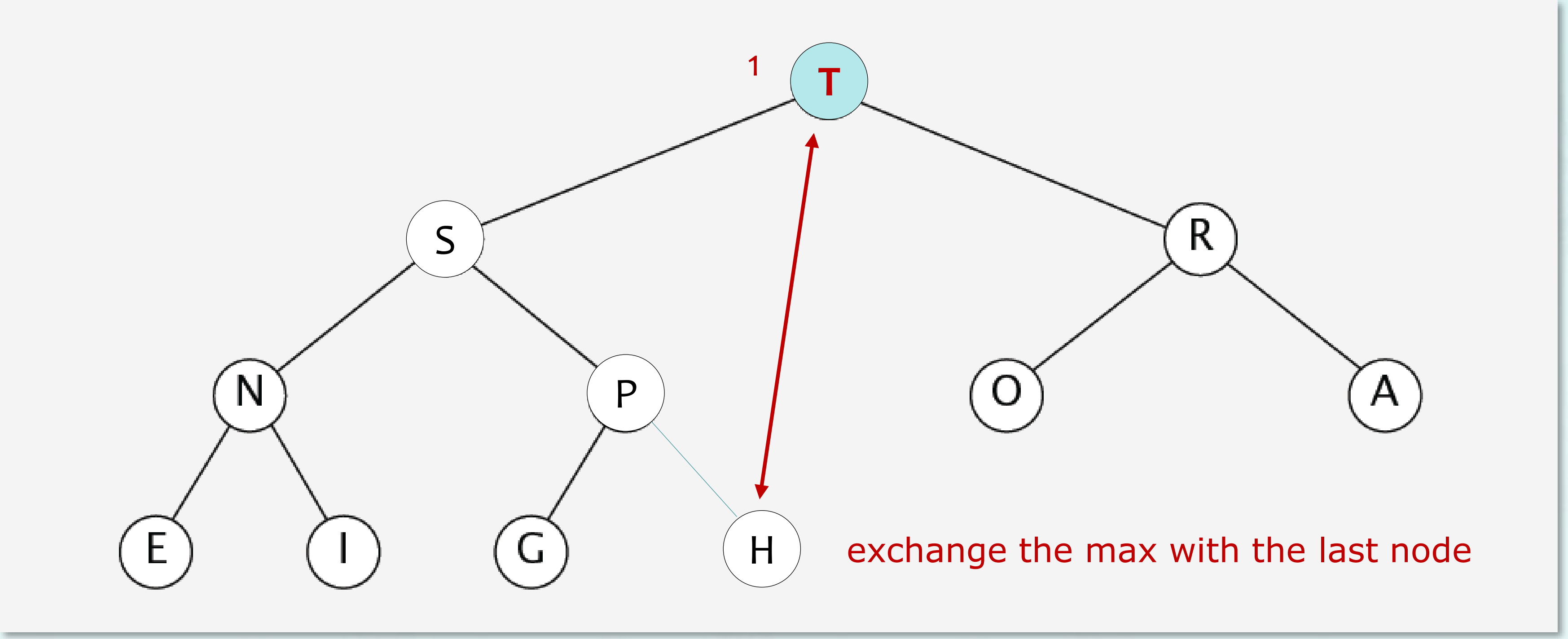
T	S	R	N	P	O	A	E	I	G	H

1

maxheap example

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

remove the maximum(root)



T	S	R	N	P	O	A	E	I	G	H

1

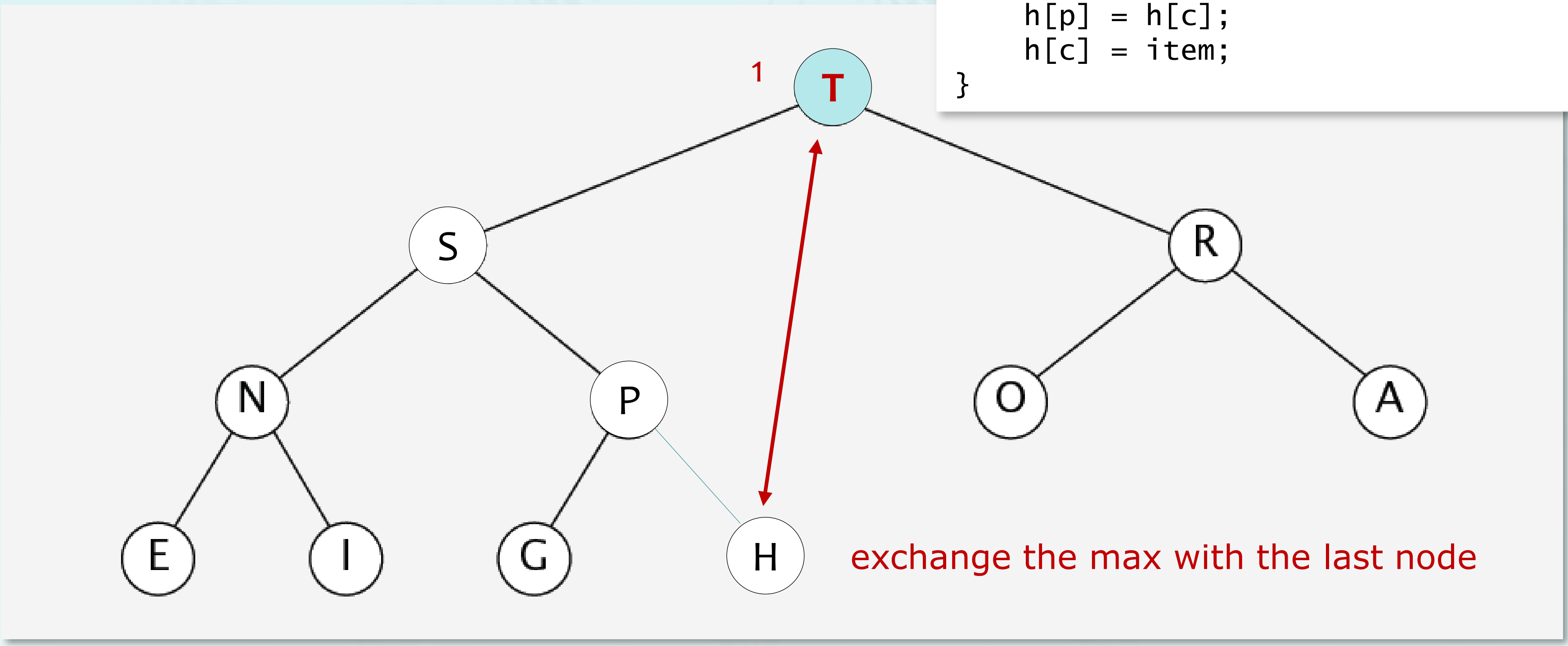
h->N

maxheap example

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

```
void swap(heap h, int p, int c) {
    Key item = h[p];
    h[p] = h[c];
    h[c] = item;
}
```

remove the maximum(root)

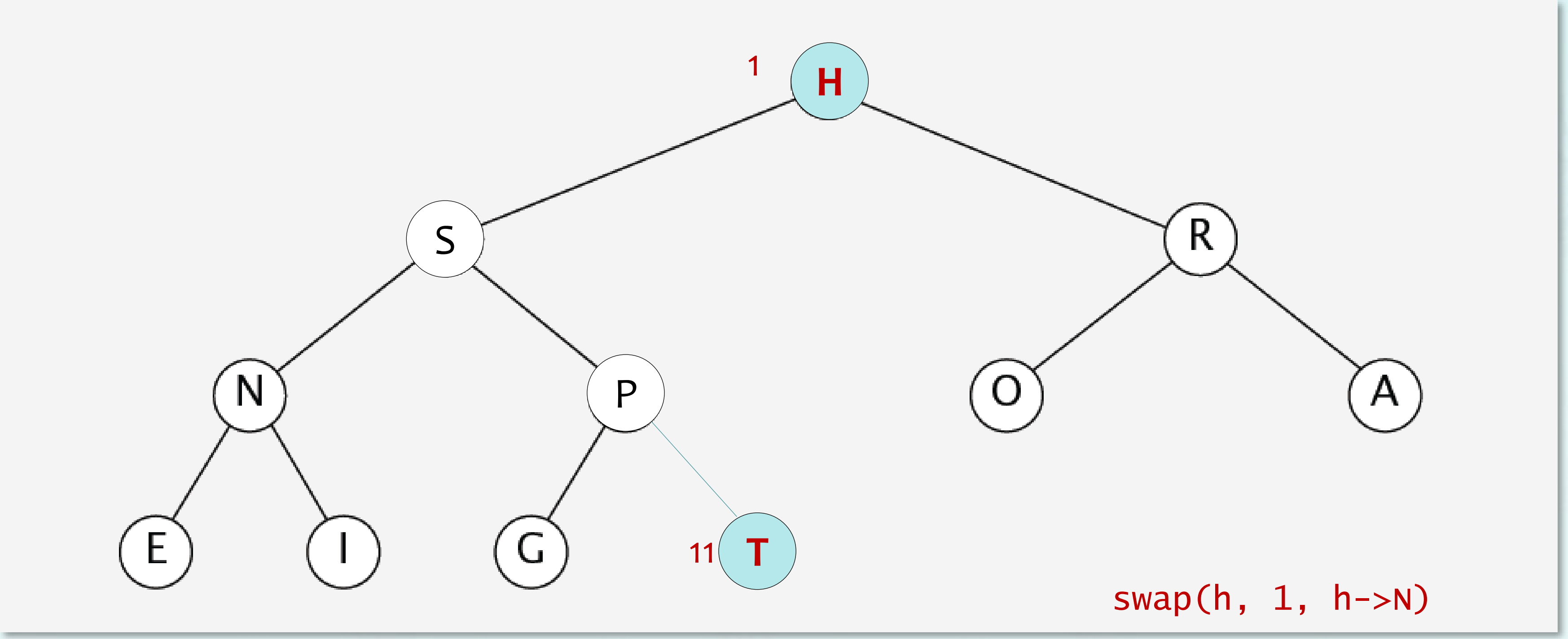


T	S	R	N	P	O	A	E	I	G	H
1										h->N

maxheap example

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

remove the maximum(root)

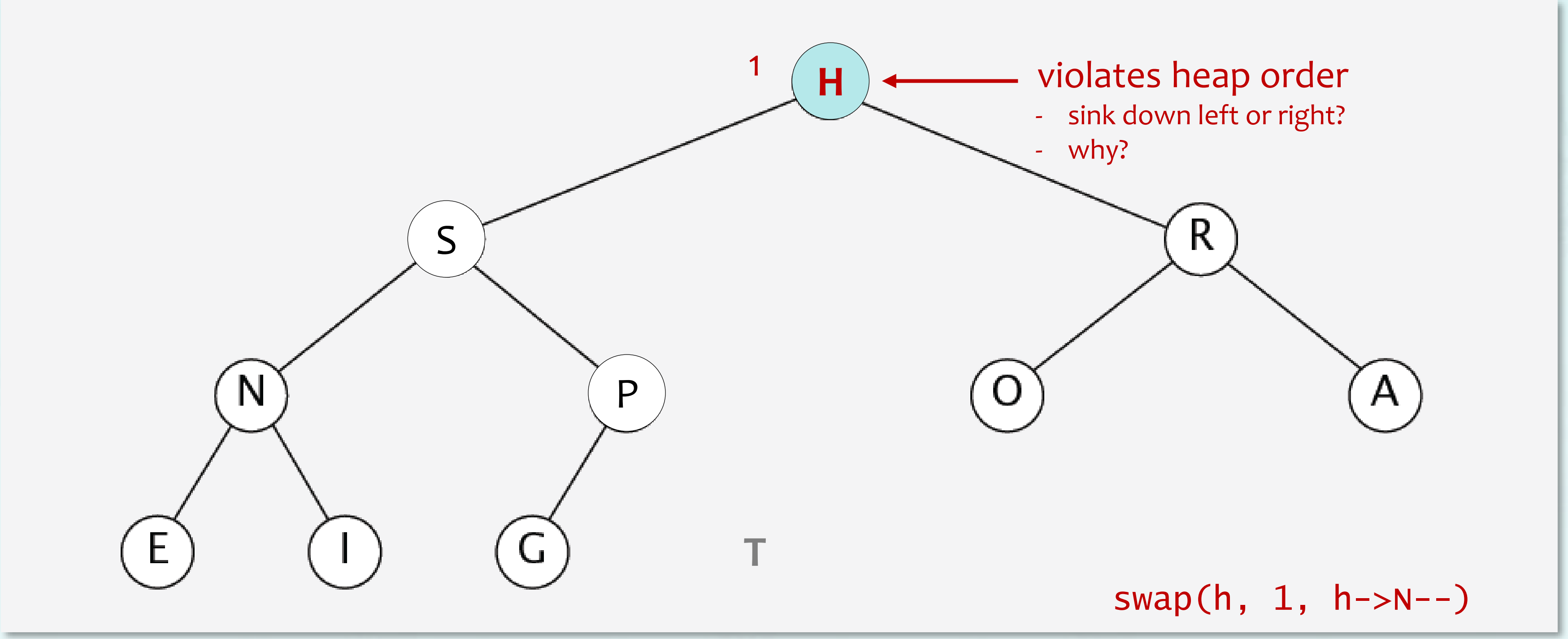


H	S	R	N	P	O	A	E	I	G	T
1										11

maxheap example

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

remove the maximum(root)



H	S	R	N	P	O	A	E	I	G	T

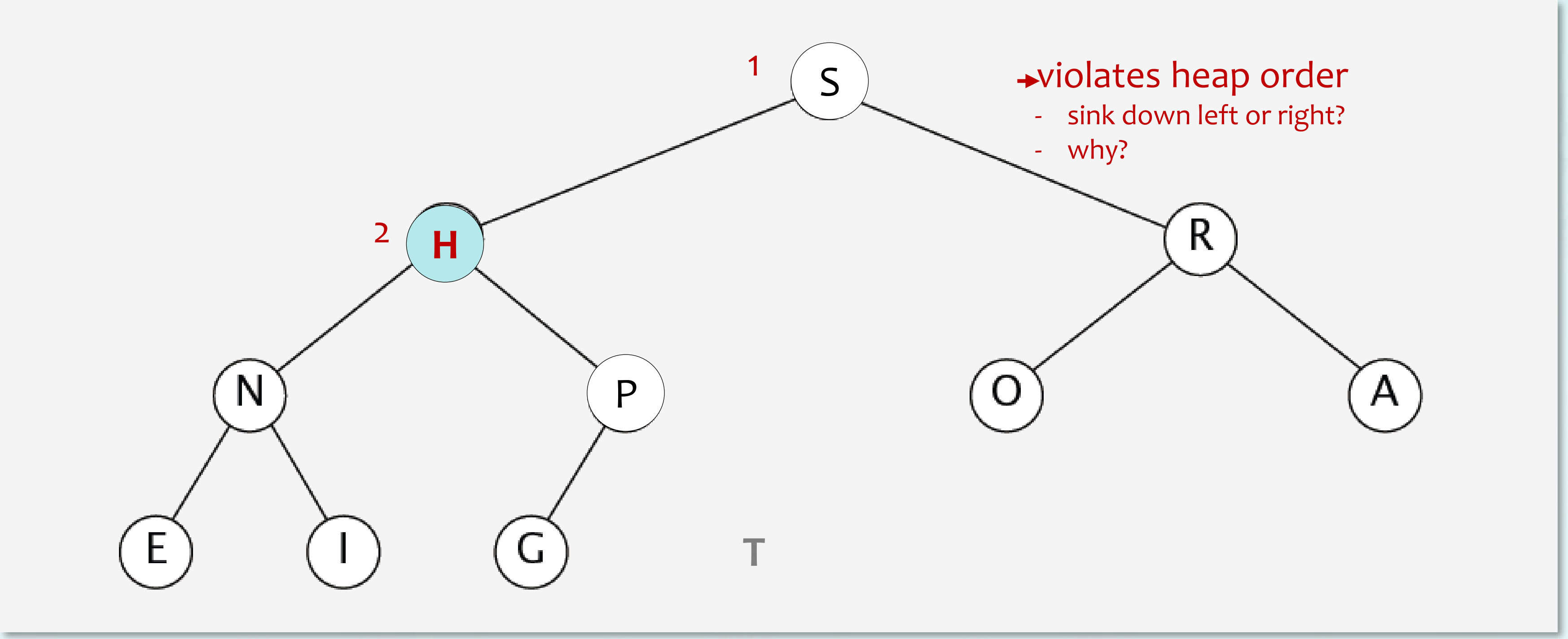
1

heap size decreased by one

maxheap example

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

remove the maximum(root)

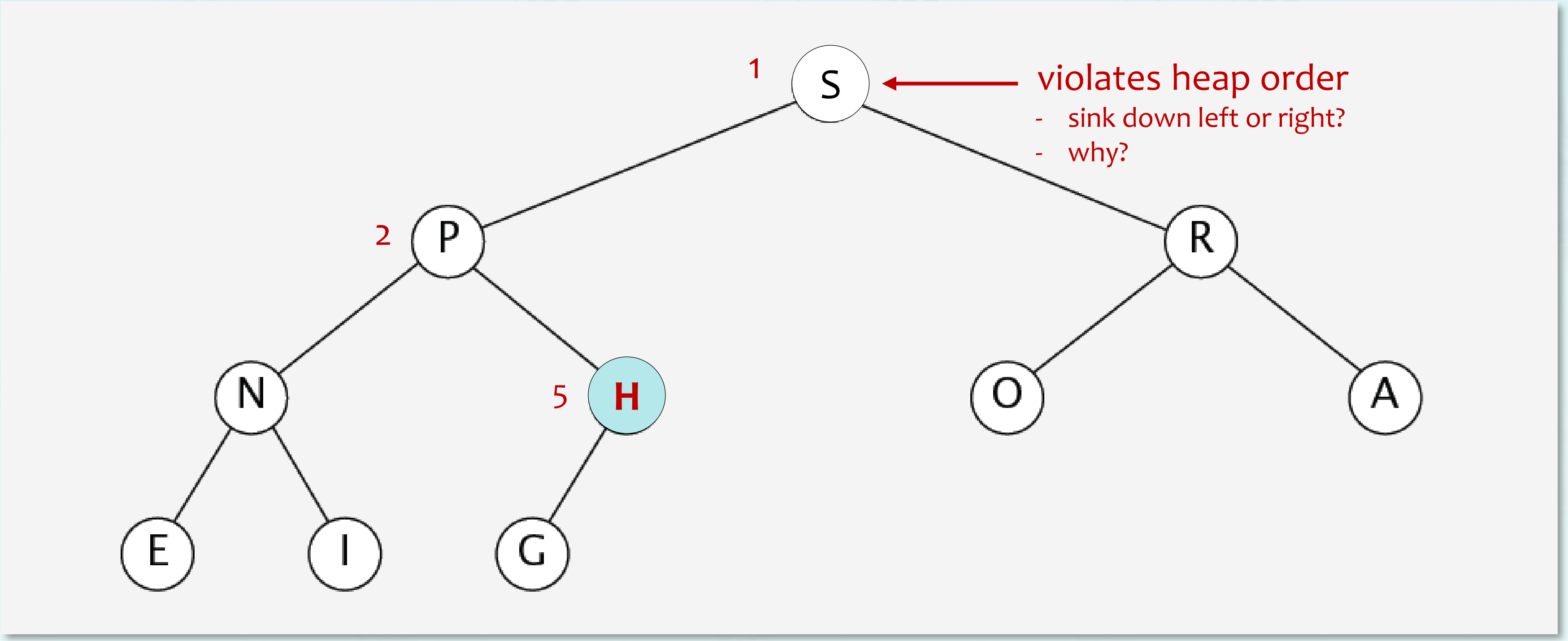


S	H	R	N	P	O	A	E	I	G	T
1	2									

maxheap example

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

remove the maximum(root)

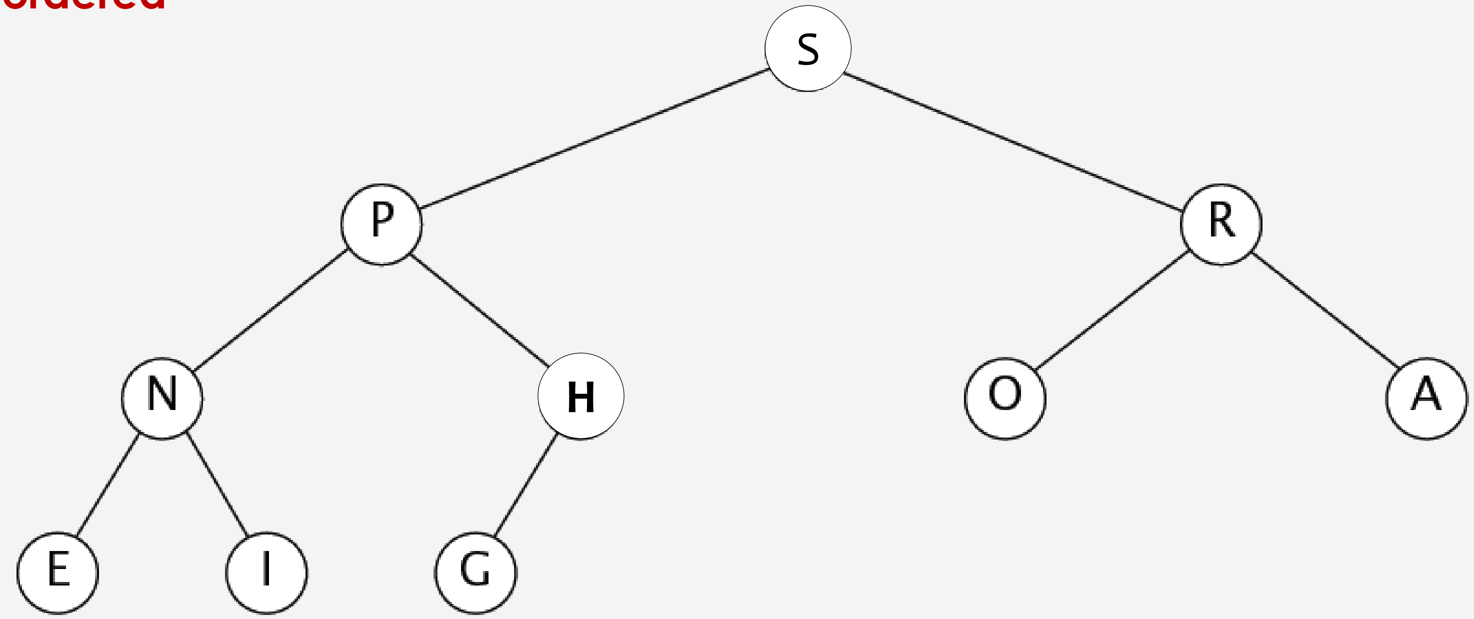


S	P	R	N	H	O	A	E	I	G	
1	2			5						

maxheap example

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

heap ordered

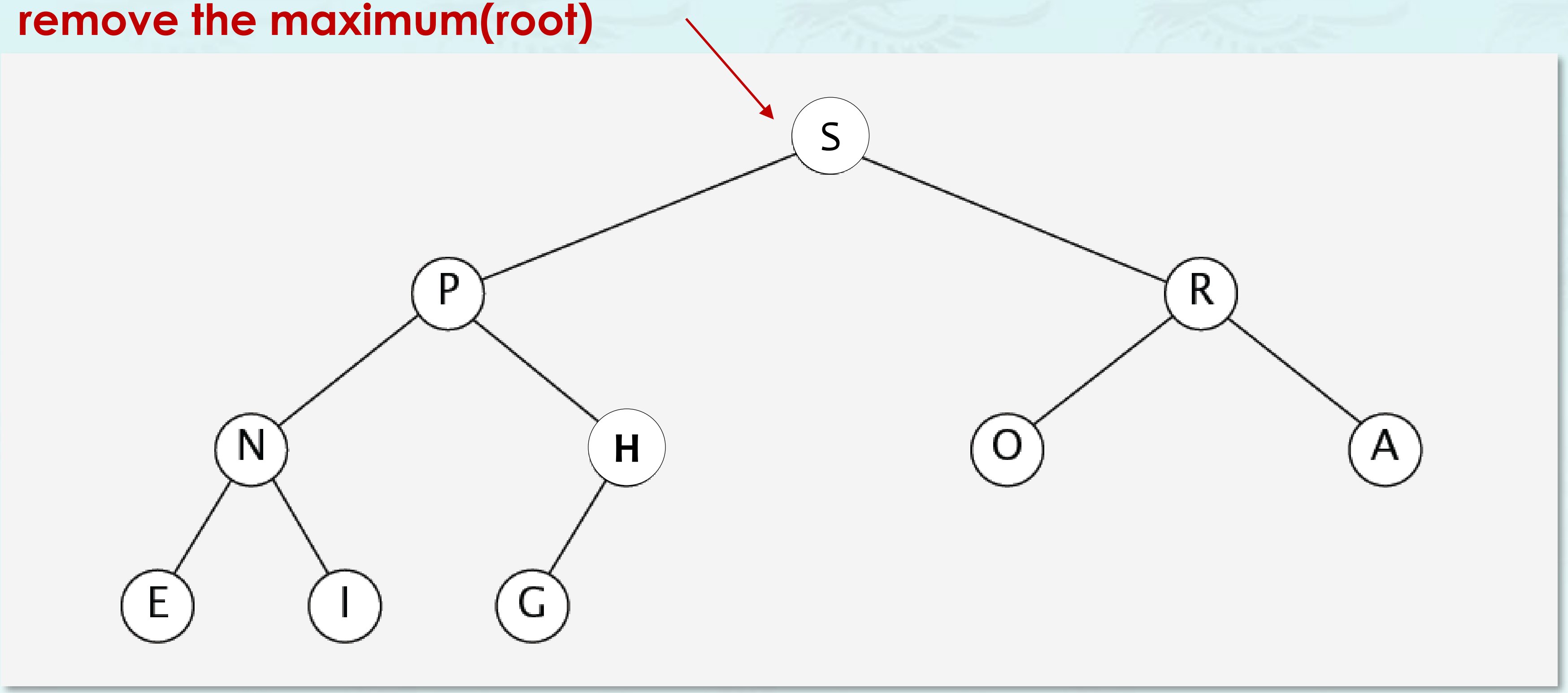


S	P	R	N	H	O	A	E	I	G	

maxheap example

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

remove the maximum(root)



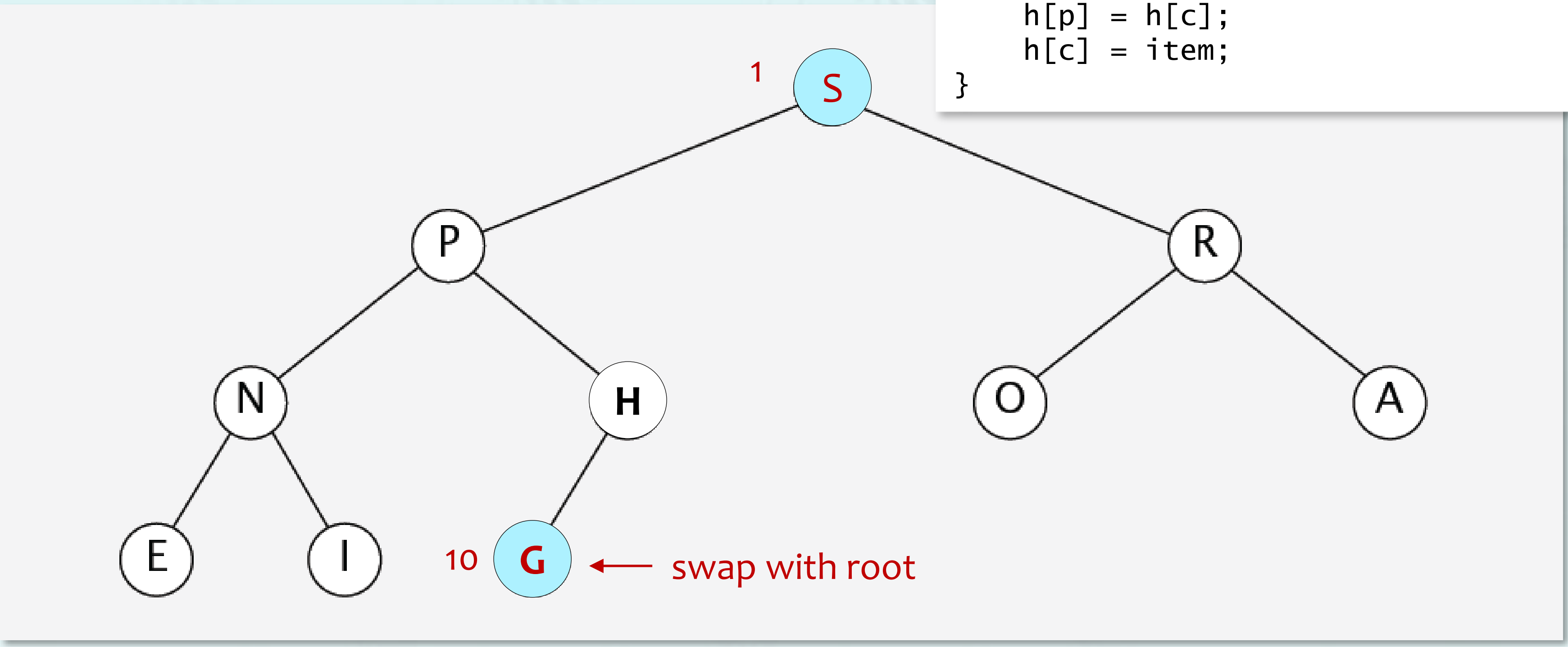
S	P	R	N	H	O	A	E	I	G	

maxheap example

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

```
void swap(heap h, int p, int c) {
    Key item = h[p];
    h[p] = h[c];
    h[c] = item;
}
```

remove the maximum(root)



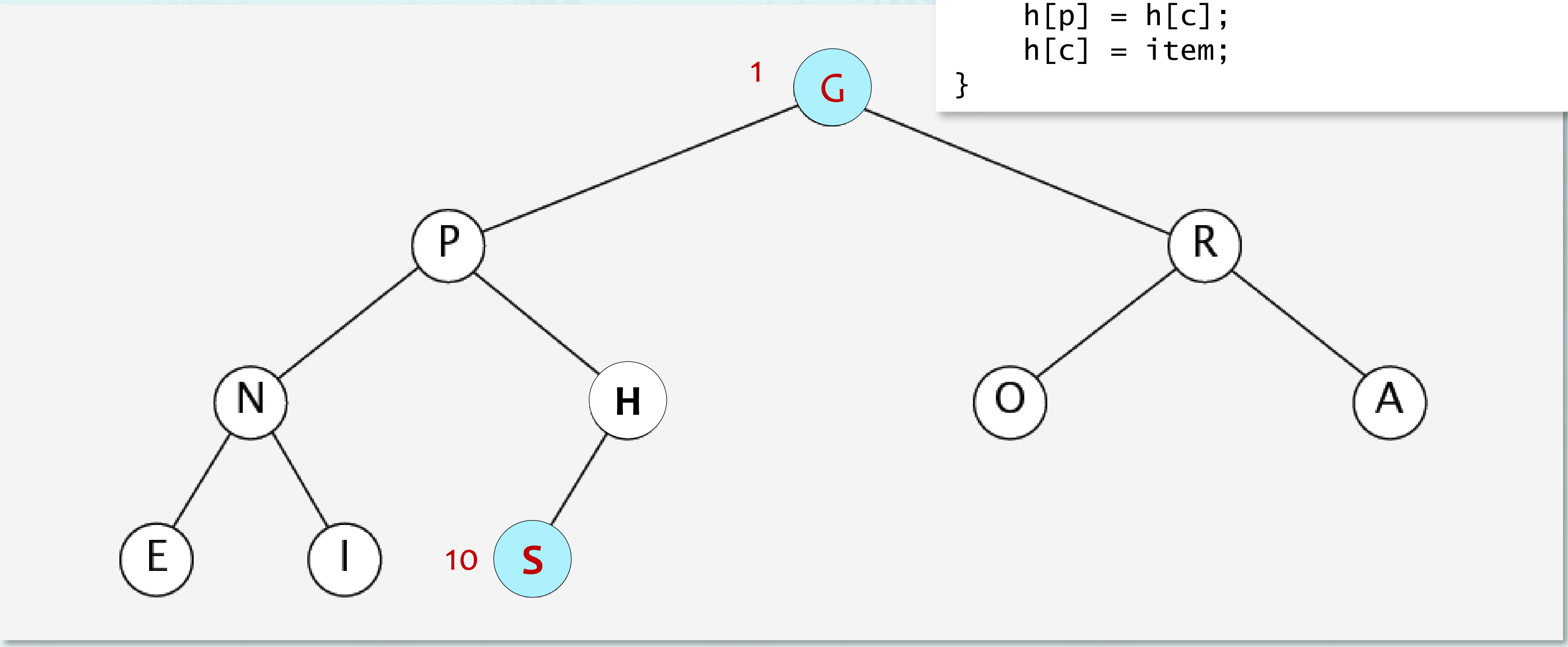
S	P	R	N	H	O	A	E	I	G	
1									10 h->N	

maxheap example

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

```
void swap(heap h, int p, int c) {
    Key item = h[p];
    h[p] = h[c];
    h[c] = item;
}
```

remove the maximum(root)

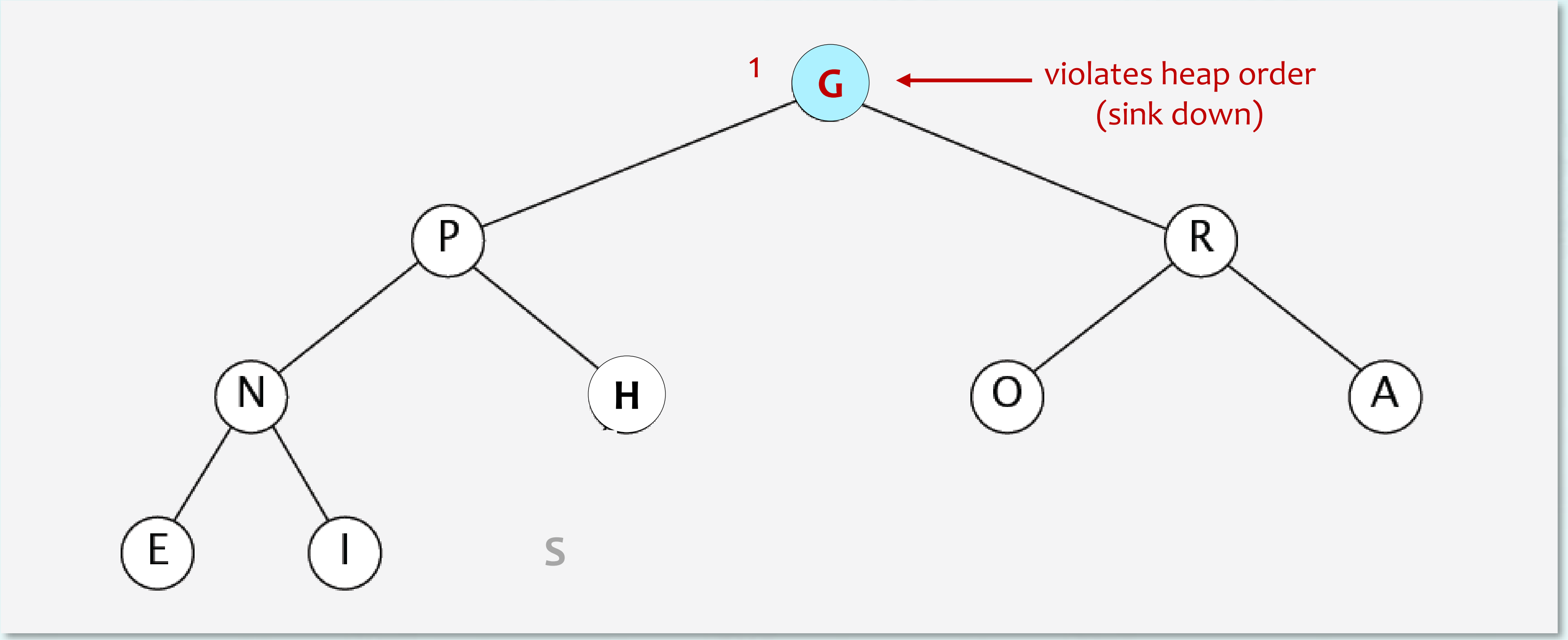


G	P	R	N	H	O	A	E	I	S	
1									10	

maxheap example

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

remove the maximum(root)



G	P	R	N	H	O	A	E	I	S	

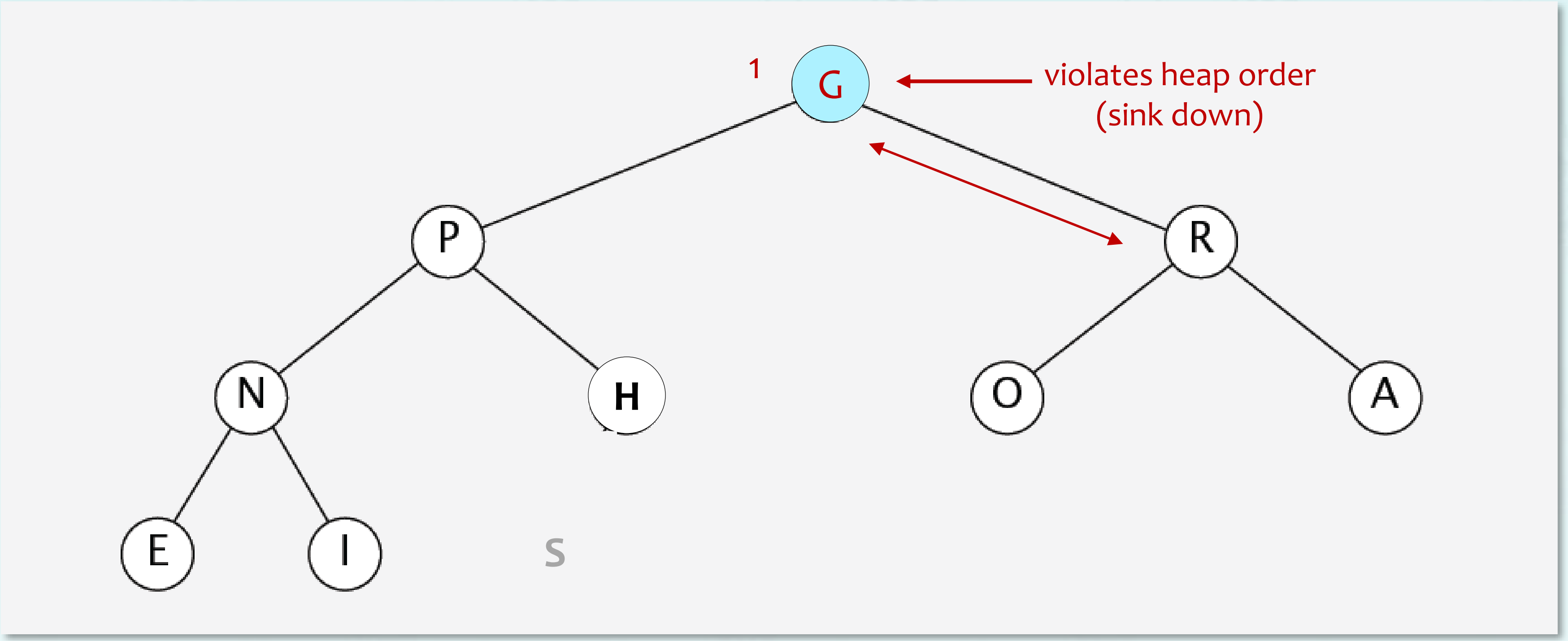
1

heap size decreased by one

maxheap example

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

remove the maximum(root)



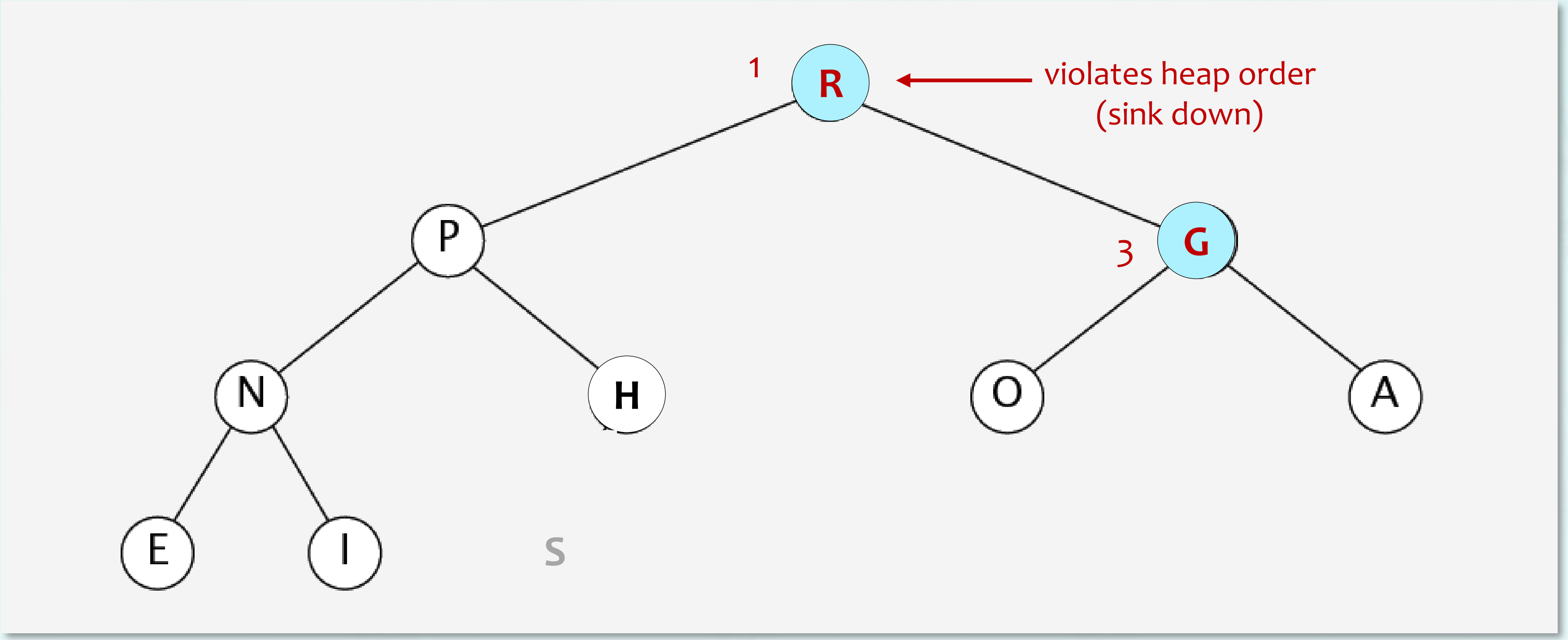
G	P	R	N	H	O	A	E	I	S	

1 N=9

maxheap example

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

remove the maximum(root)

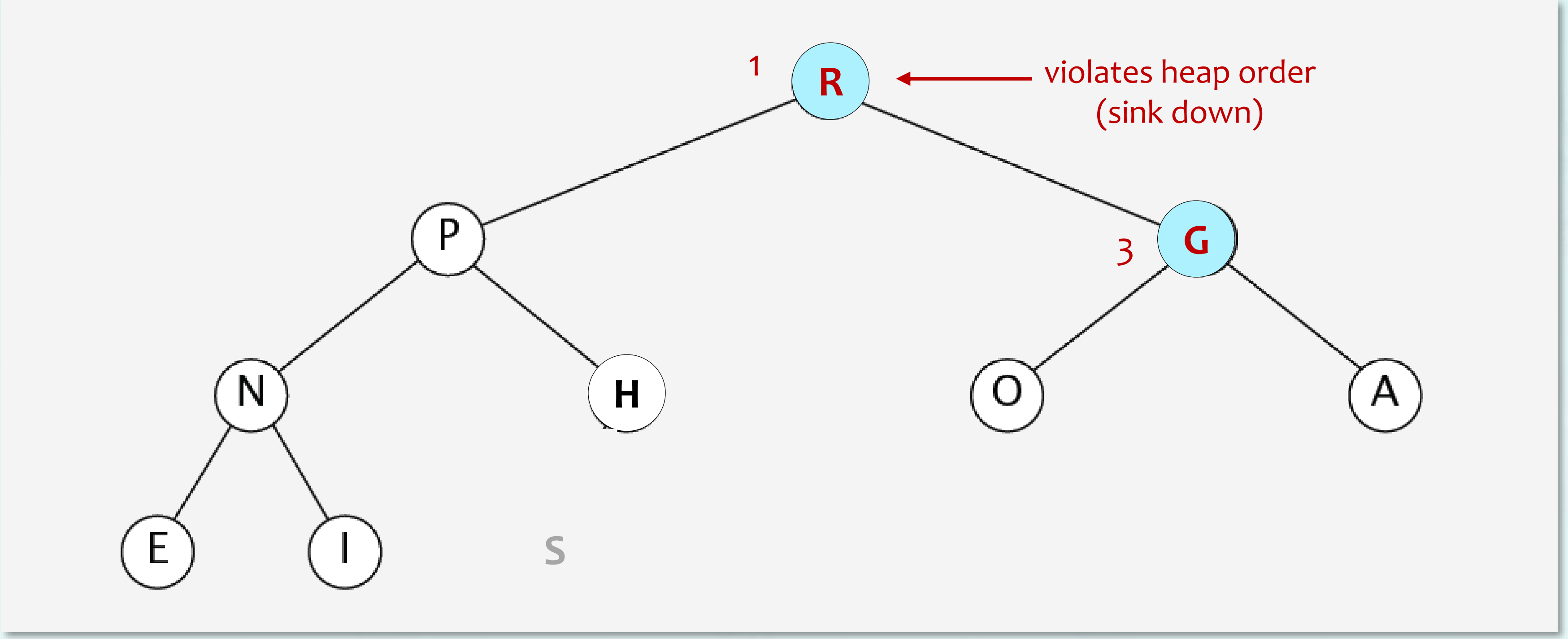


R	P	G	N	H	O	A	E	I	S	
1		3						N=9		

maxheap example

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

remove the maximum(root)

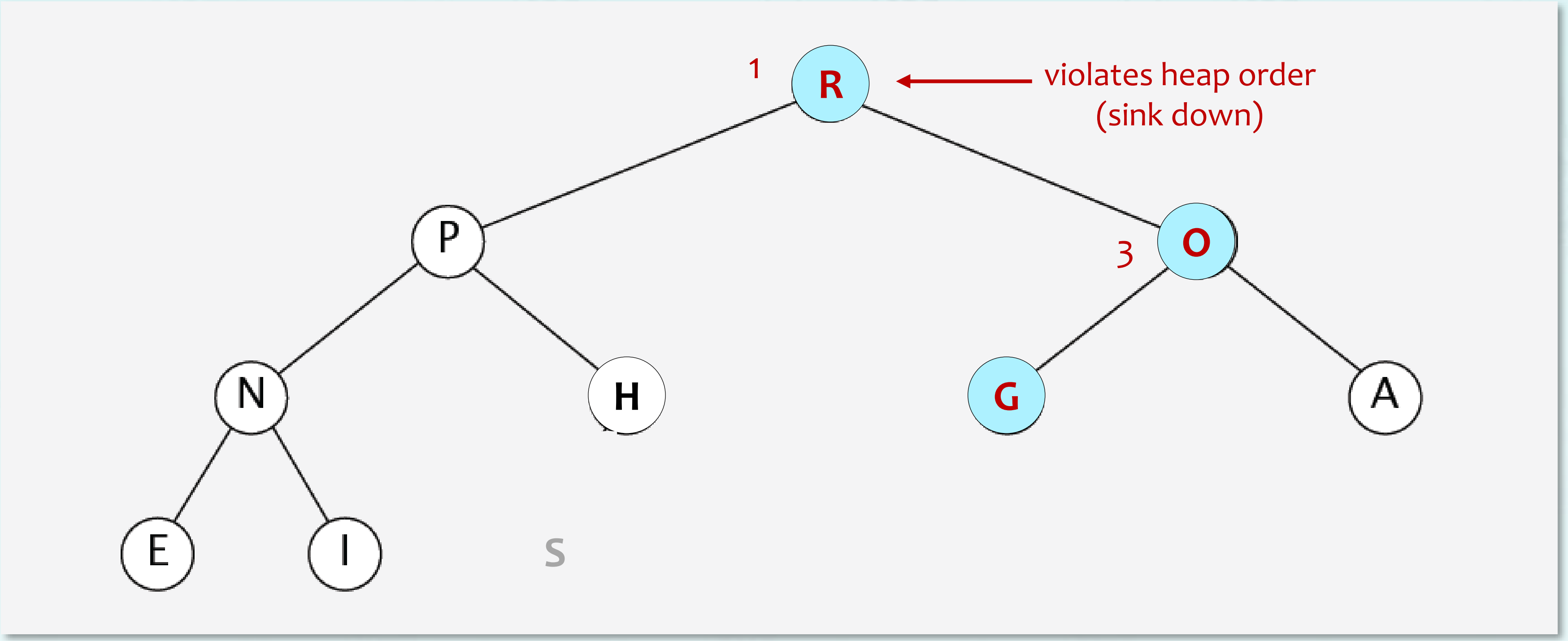


R	P	G	N	H	O	A	E	I	S	
1		3						N=9		

maxheap example

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

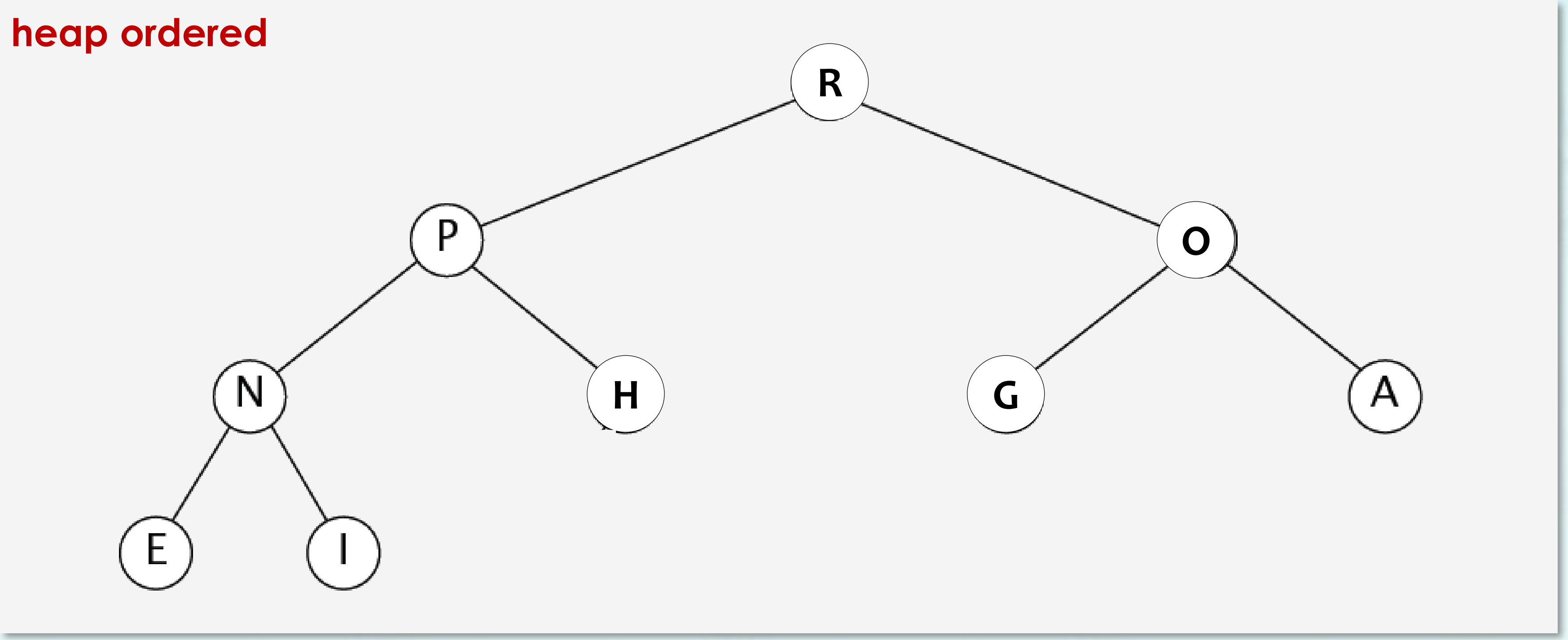
remove the maximum(root)



R	P	O	N	H	G	A	E	I	S	
1		3			6			N=9		

maxheap example

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



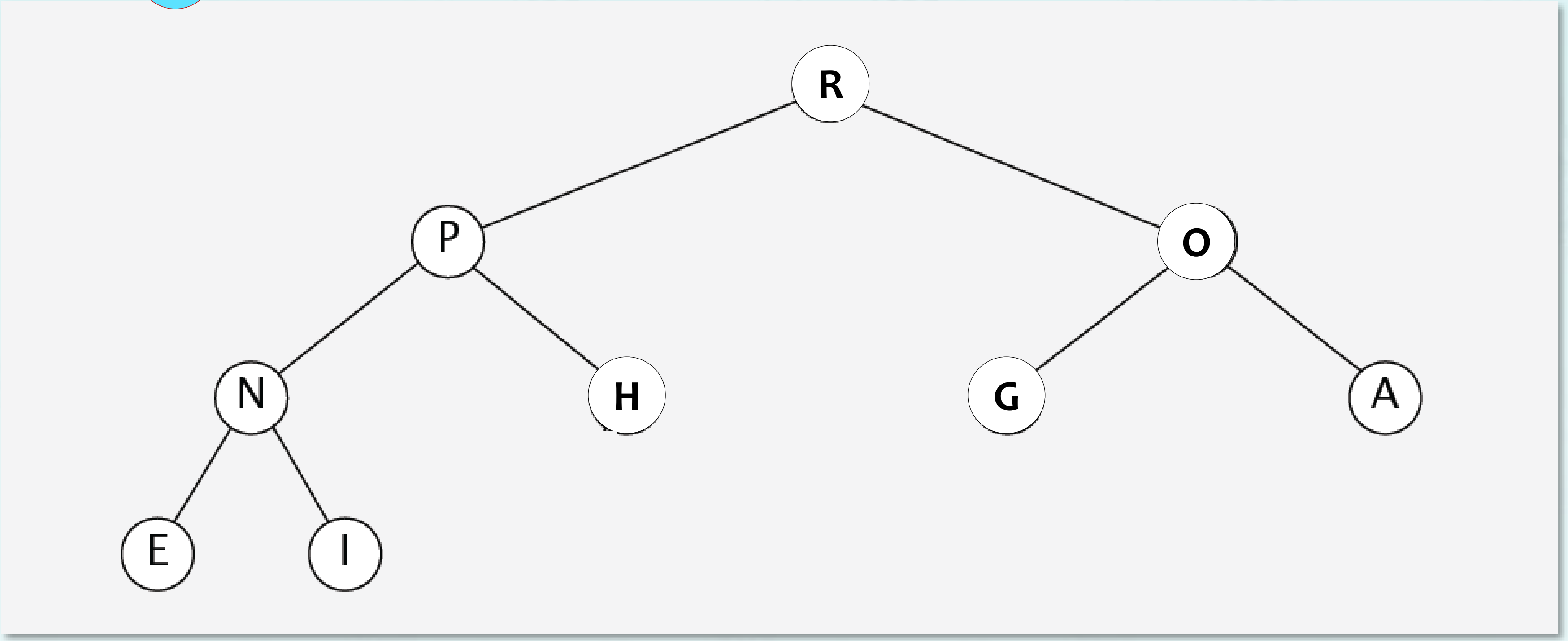
R	P	O	N	H	G	A	E	I		

N=9

maxheap example

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

insert S

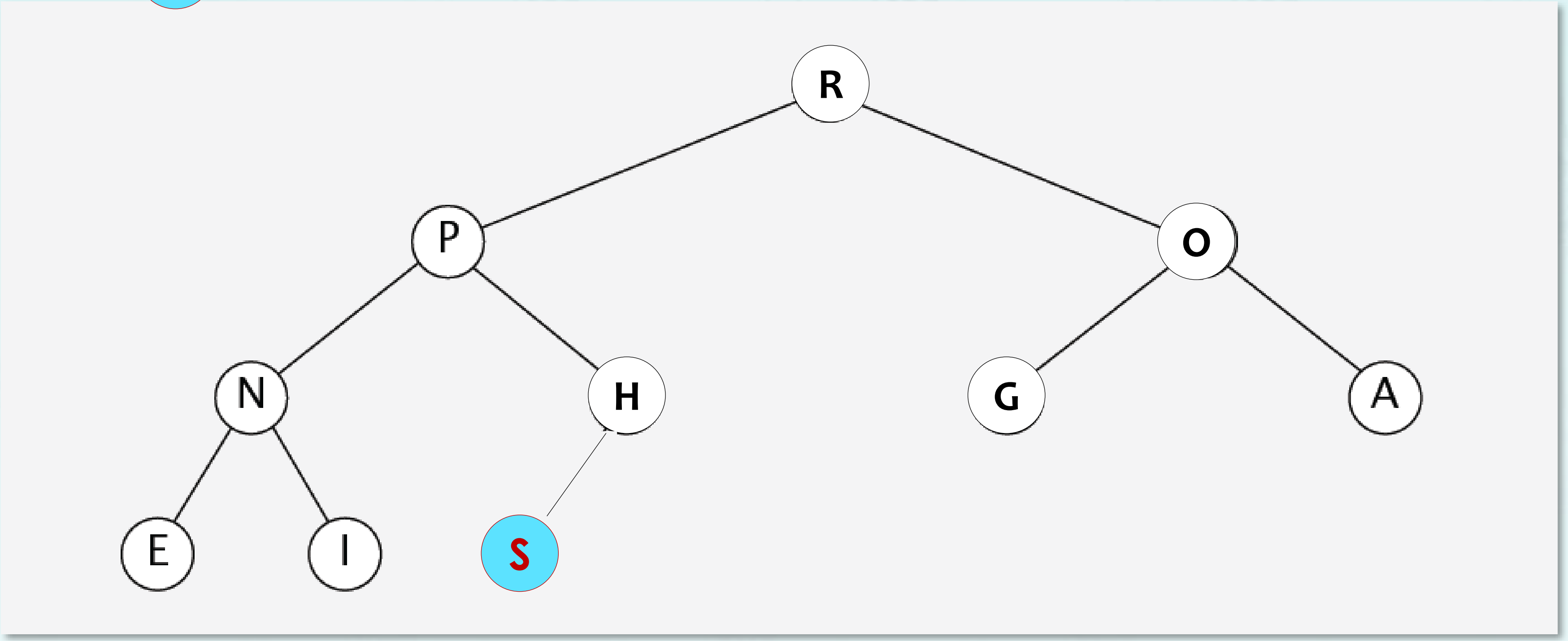


R	P	O	N	H	G	A	E	I		

maxheap example

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

insert S

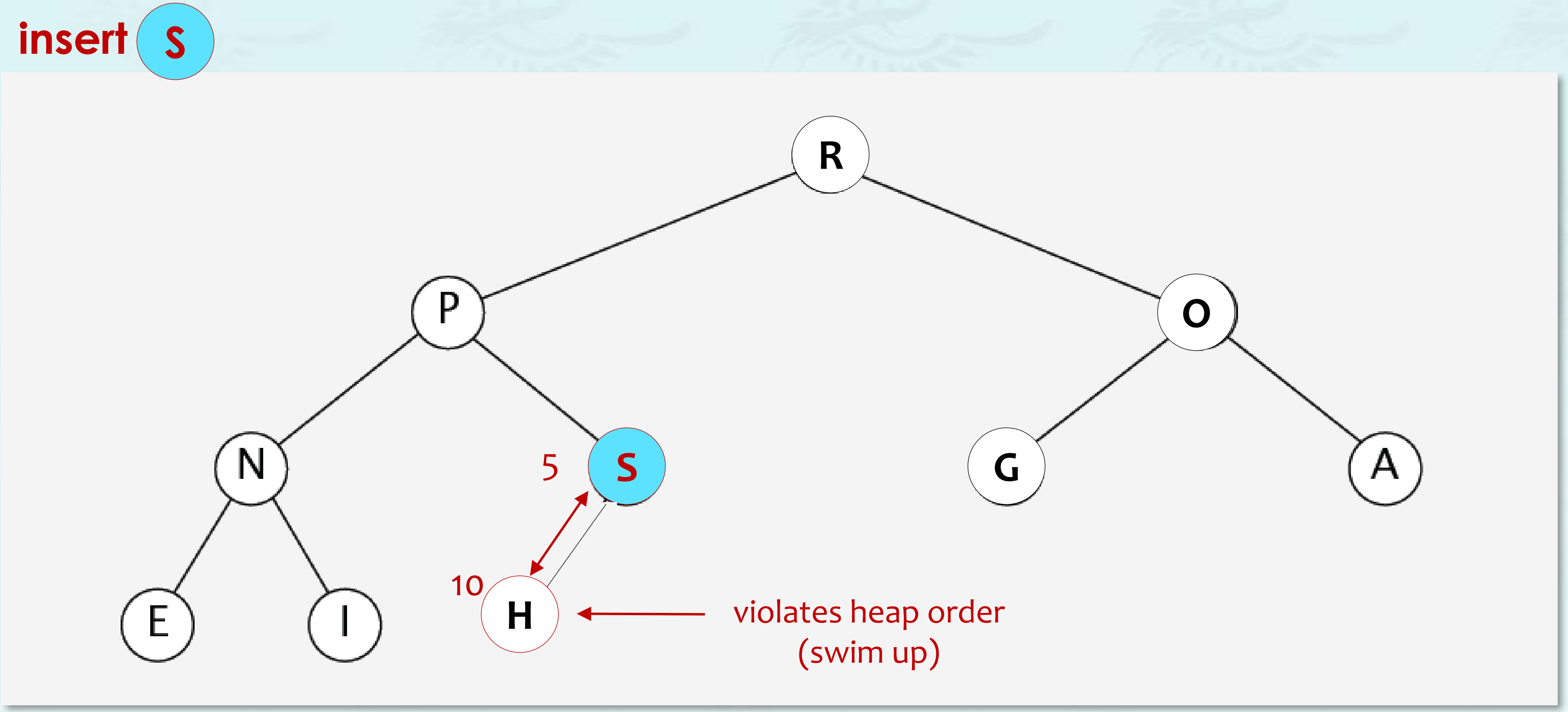


R	P	O	N	H	G	A	E	I	S	

N=10

maxheap example

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



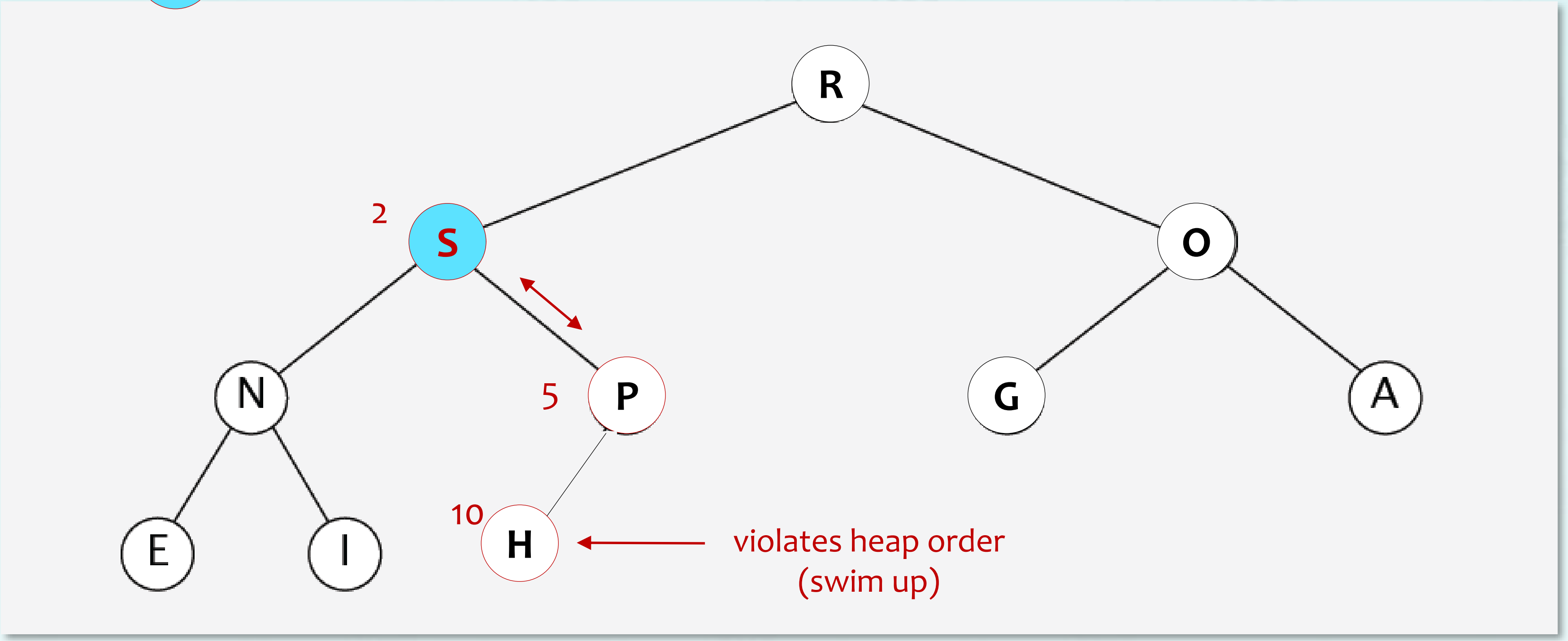
R	P	O	N	S	G	A	E	I	H	

5 N=10

maxheap example

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

insert S

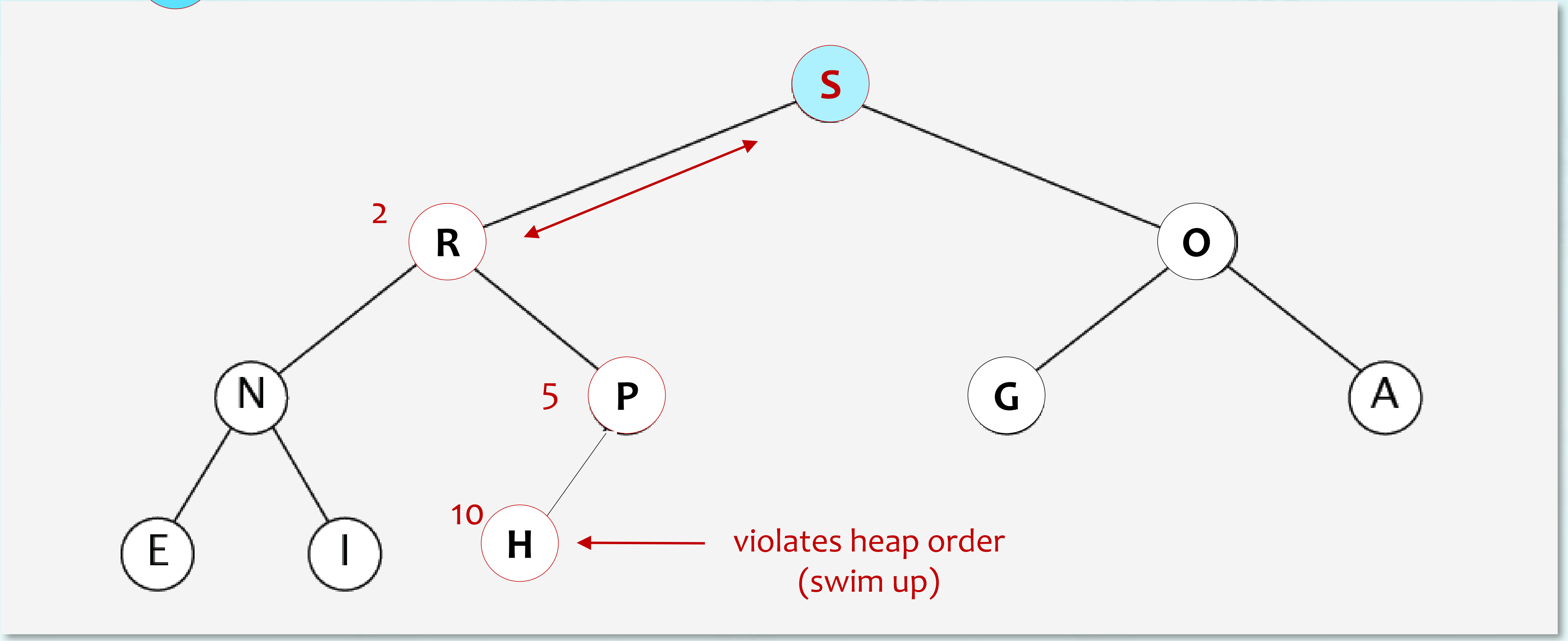


R	S	O	N	P	G	A	E	I	H	
	2			5					N=10	

maxheap example

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

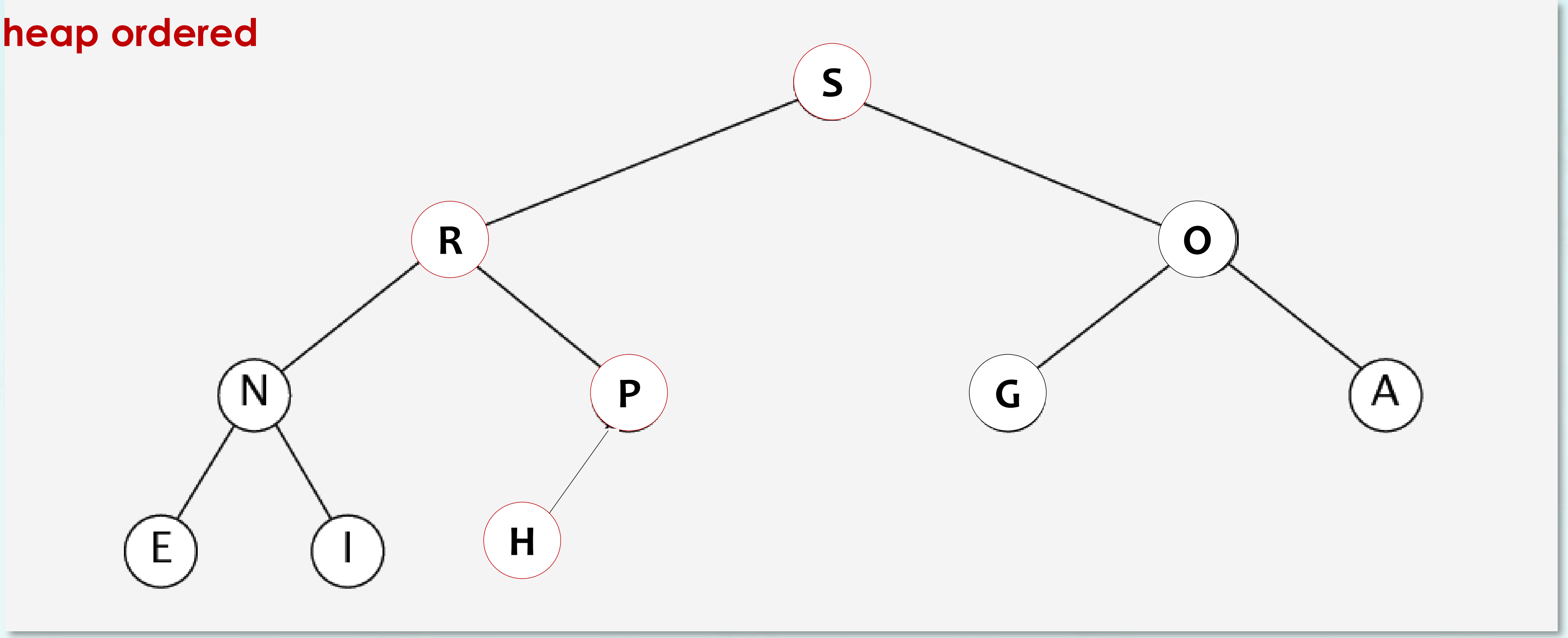
insert S



S	R	O	N	P	G	A	E	I	H	
1	2			5					N=10	

maxheap example

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



S	R	O	N	P	G	A	E	I	H	

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)
https://stackoverflow.com/questions/9755721/how-can-building-a-heap-be-on-time-complexity

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

heap

- *complete binary tree (review)*
- *heap and priority queues (Chapter 9)*
- *binary heap and minheap*
- *maxheap demo*
- *maxheap coding*
- *heap sort (Chapter 7)*

Chapter 7

this node is deleted if trim selected

new node is added if grow selected

index	0	1	2	3	4	5	6	7
nodes[ ]		7	3	6	8			

unused

N=4

capacity=8