ALGORITHMS

(CSCI 2200)

Week 4

Heaps and Heapsort

Laura Toma
Bowdoin College

Week 4 Announcements

- Assignment 1 was due last night by 11pm, on Gradescope
- Assignment 2 is due on 9/26 by 11pm, on Gradescope
- You need to check regularly class website

Week 4 Overview

- The priority queue data structure
- The heap
    - Definition, min-heaps and max-heaps
    - Operations: Insert, Delete-Min, Heapify, Buildheap
    - Heapsort
- Quicksort
    - Partition
- Randomized quicksort

The Priority Queue

- A container of objects that have keys (or: priorities)
- Supported operations on a Min-pqueue
  - **Insert**: insert a new object to the queue
  - **Delete-Min:** delete the object with a minimum key value
- Max-pqueues are symmetrical

PQueue Applications

- Sorting
    - Insert the objects into a priority queue; then call Delete-Min to put the elements in order
    - Run time: n x Insert + n x DeleteMin
- Event managers
    - objects = the events
    - key = time the events is scheduled to occur
    - DeleteMin: gives the next scheduled event
- Process scheduling
    - objects = processes waiting to be scheduled on the processor
    - key = priority of each event
    - DeleteMax: gives the next process to be scheduled

The binary heap

The heap

- The (binary) heap is standard implementation of a PQ

Min-heaps	Max-heaps
Operations:	Operations:
• Insert(A, element e)	• Insert(A, element e)
• DeleteMin(A)	• DeleteMax(A)
• Heapify(A, i)	• Heapify(A, i)
• Buildheap(A)	• Buildheap(A)

Run time: O(lg n) — Insert, DeleteMax, Heapify

O(n) — Buildheap

symmetrical

The min-heap

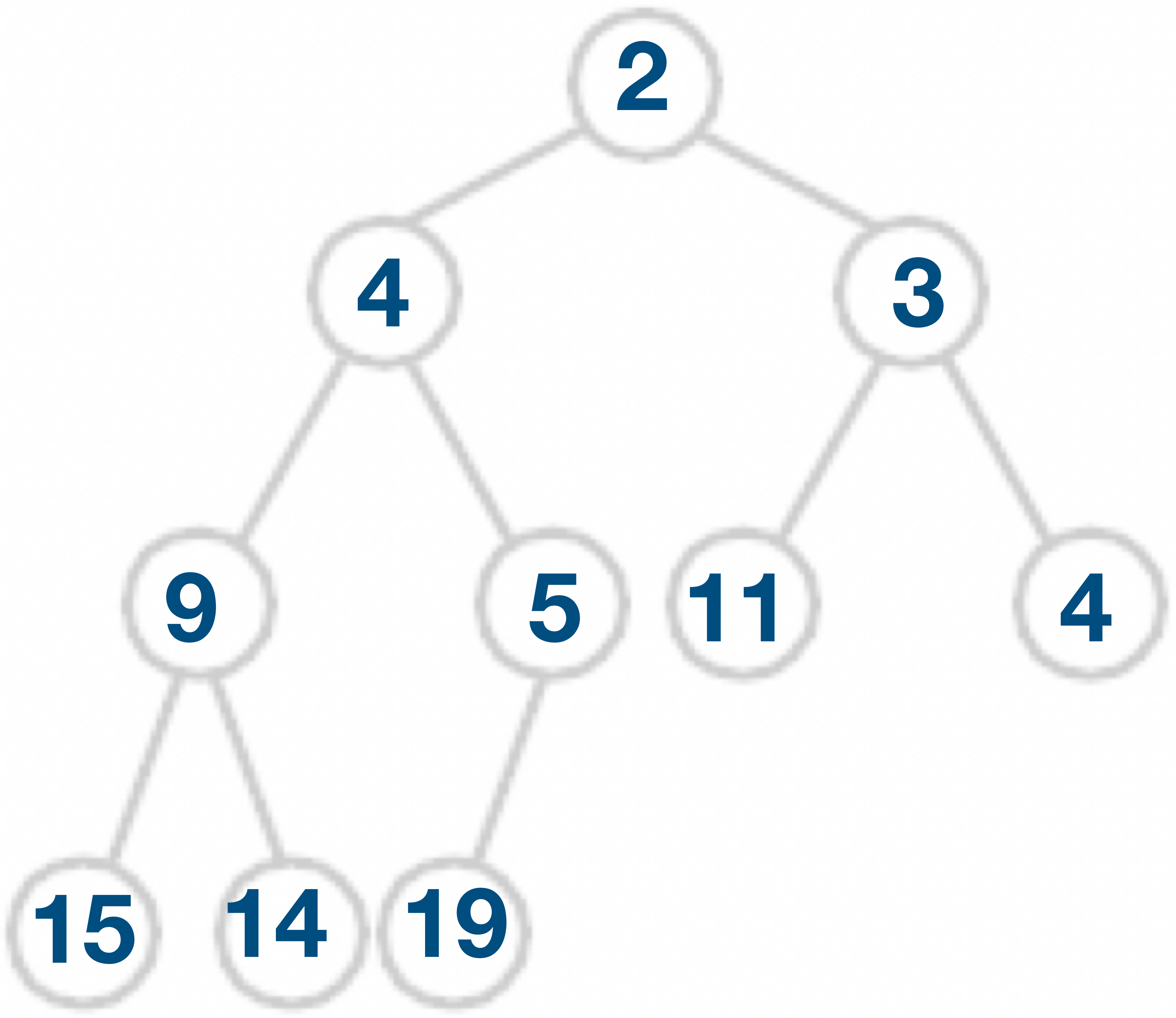
An array: viewed as corresponding to a complete binary tree (except last level, which is filled from left to right)

Heap property: for all nodes v, priority(v) <= priority of children(v)

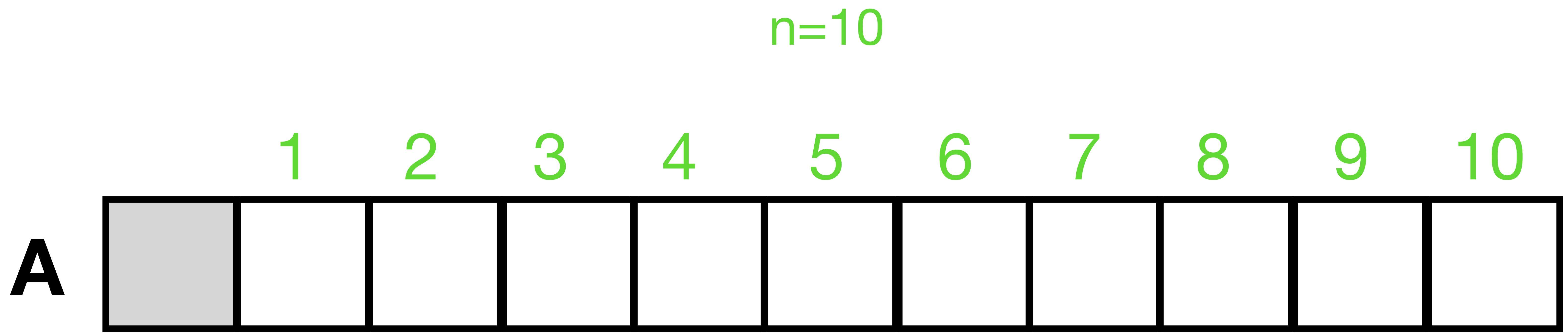
A

2	4	3	9	5	11	4	15	14	19

n=10



Properties



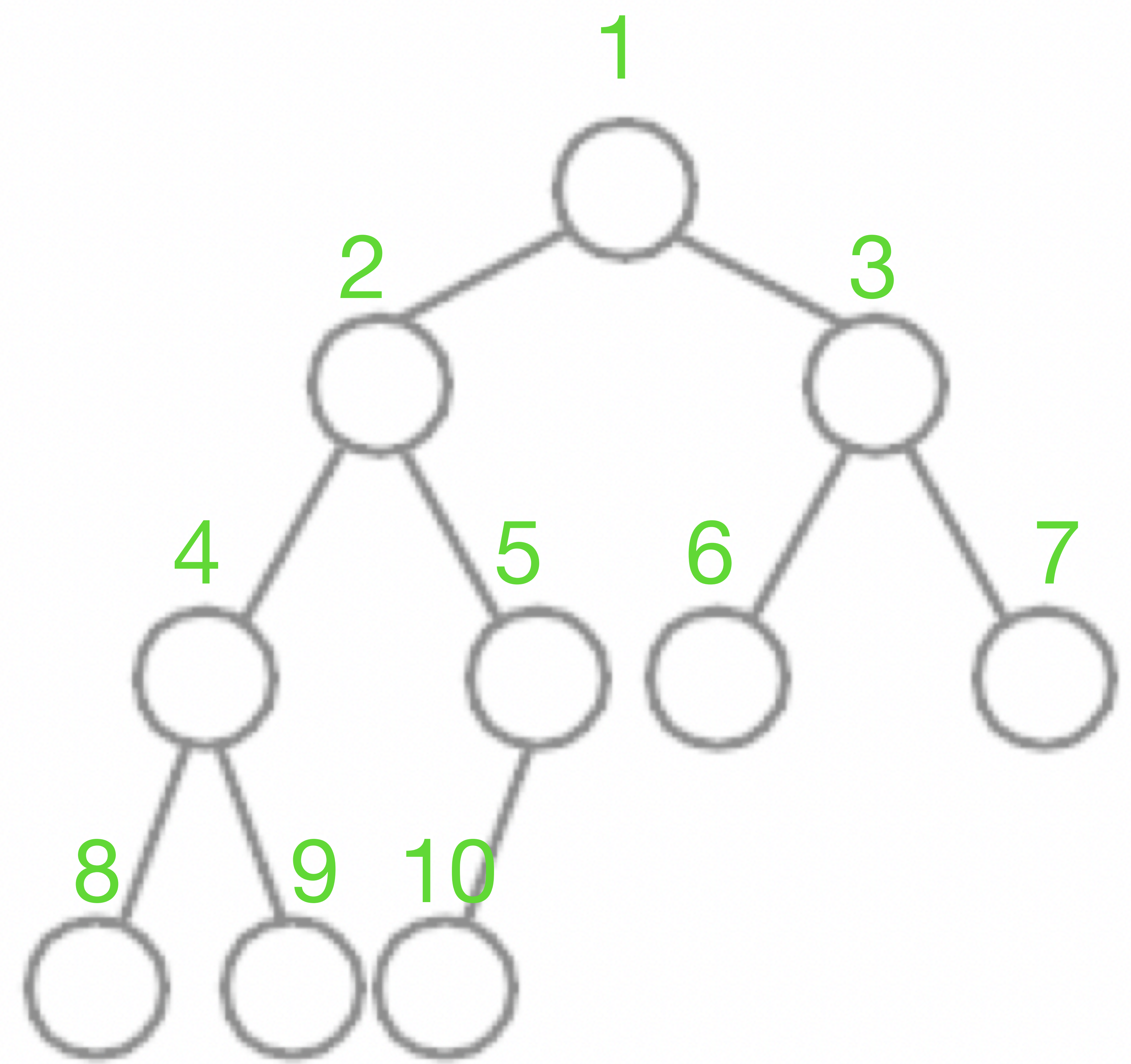
1. The smallest element is in the root

2. The height of a heap of n elements is $\Theta(\lg n)$

3. The indices of the children and parent of a node can be inferred (without storing pointers)

For node at index i:

- left(i) = 2 i
- right(i) = 2i+1
- parent(i) = i/2



Operations supported by a min-heap

- Peak(A):
    - A is a heap; returns the min element in A
- Insert(A, e)
    - A is a heap; Insert element e and maintain A as a heap.
- Delete-Min()
    - A is a heap; delete the min element in A and return it. Maintain A as a heap.
- Heapify(A, i)
    - left(i) is a heap and right(i) is a heap. Make a heap under i
- Buildheap(A)
    - A is an array. Shuffle elements around so that A becomes a heap.
- Heapsort (A)
    - sort A in place

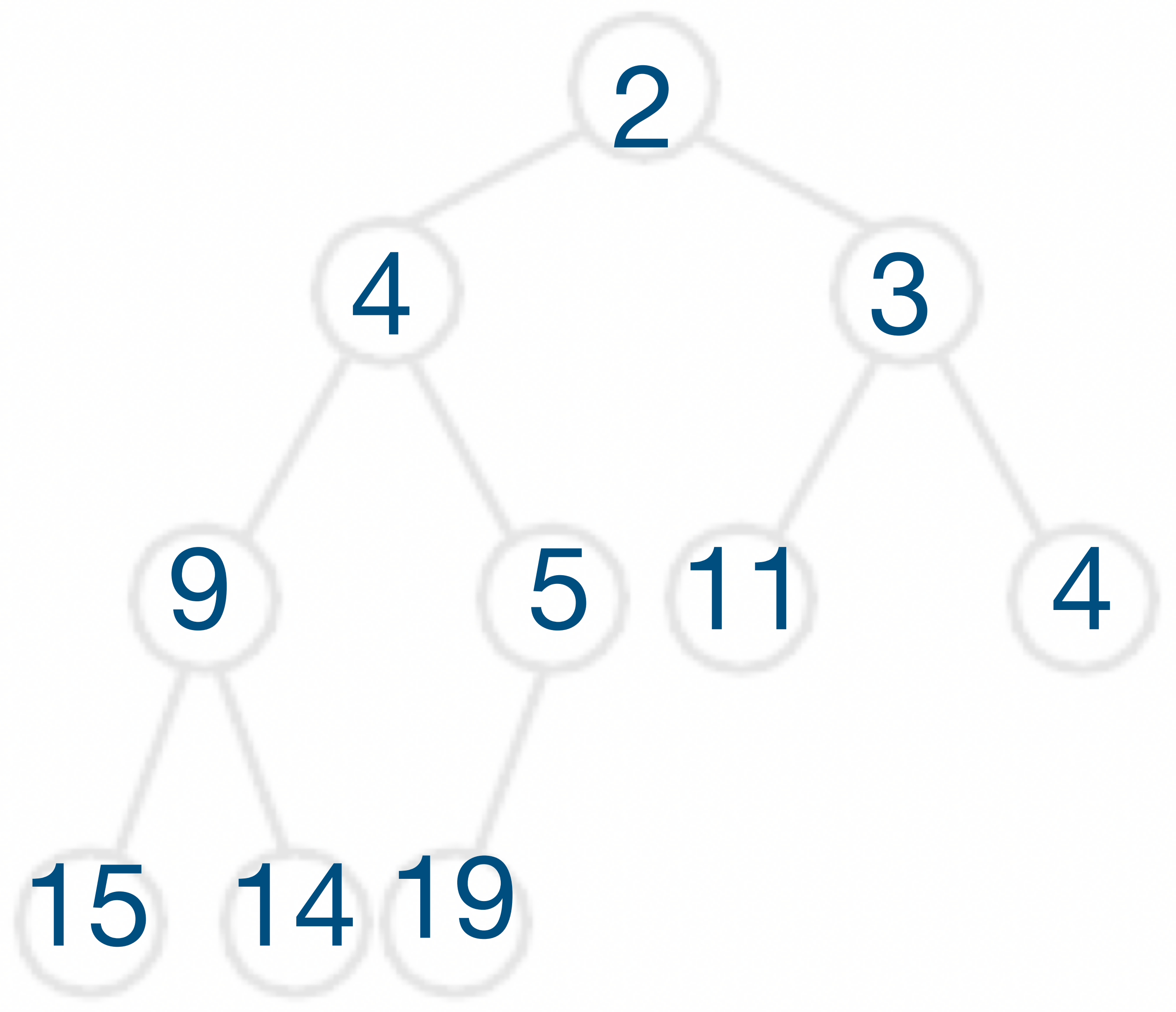
Inserting in a heap

n=10

A

	2	4	3	9	5	11	4	15	14	19

Insert(A, 3)



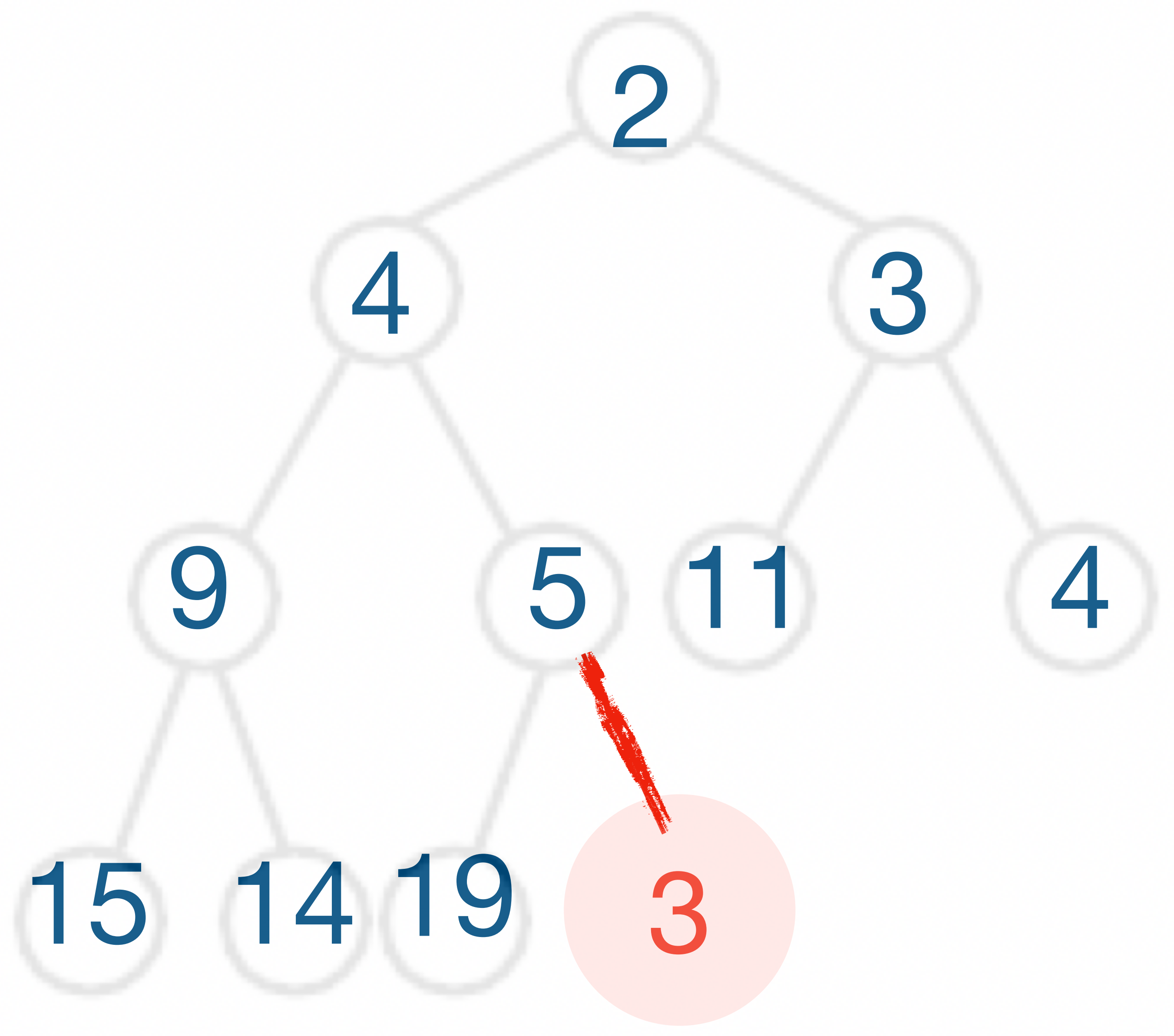
Inserting in a heap

n=11
~~n=10~~

A

	2	4	3	9	5	11	4	15	14	19	3

Insert(A, 3)

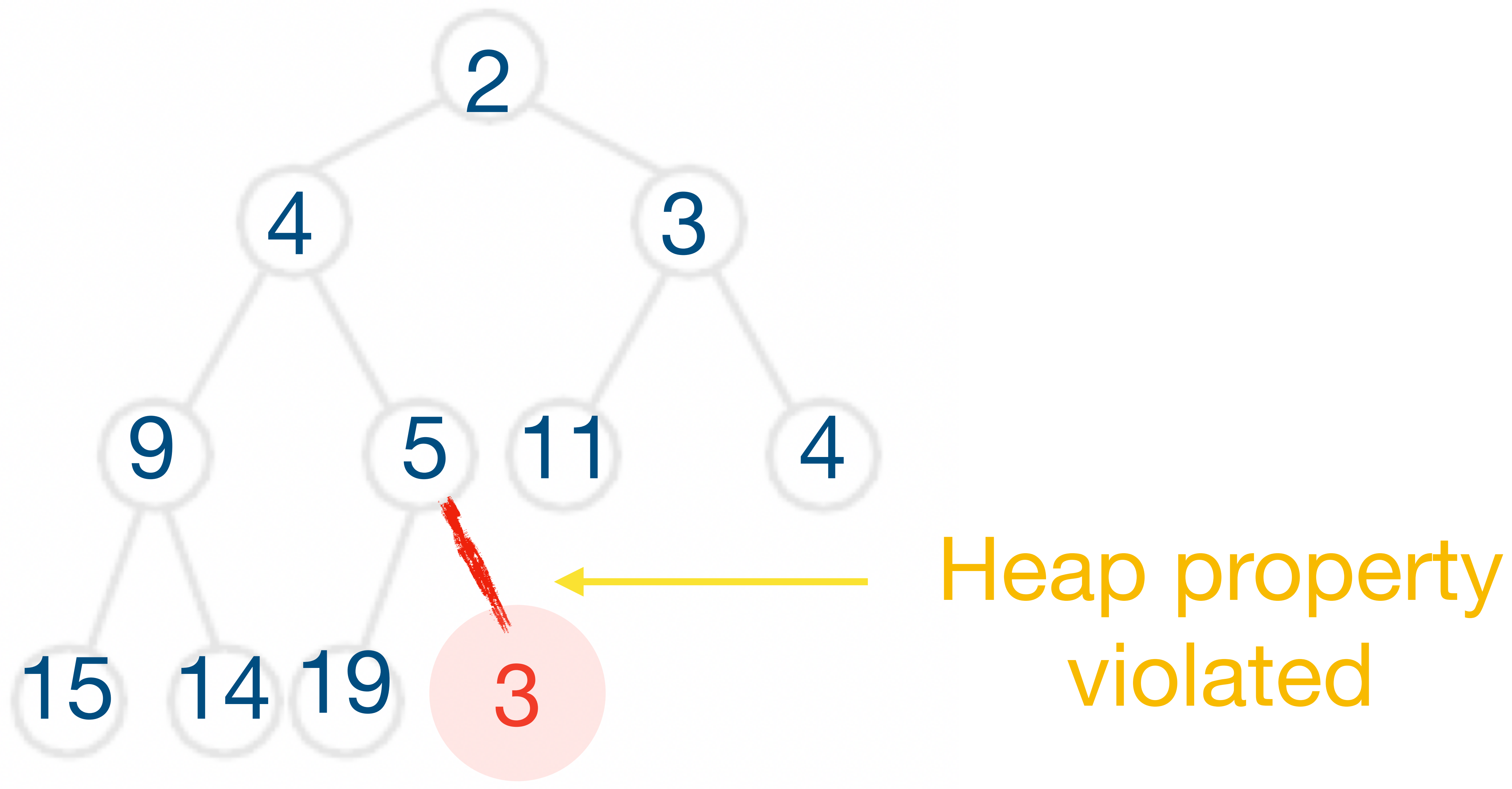


Inserting in a heap

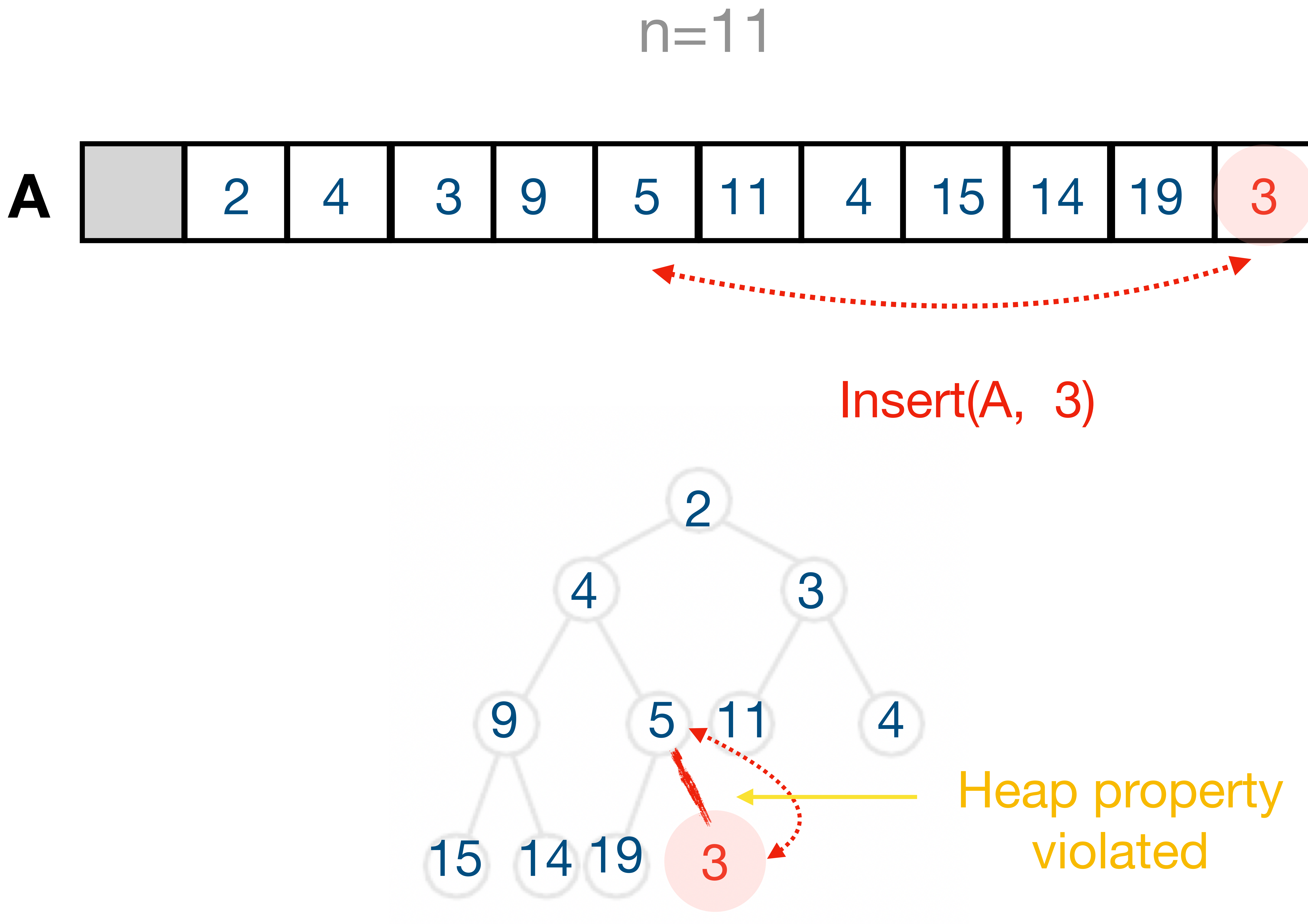
n=11

A		2	4	3	9	5	11	4	15	14	19	3

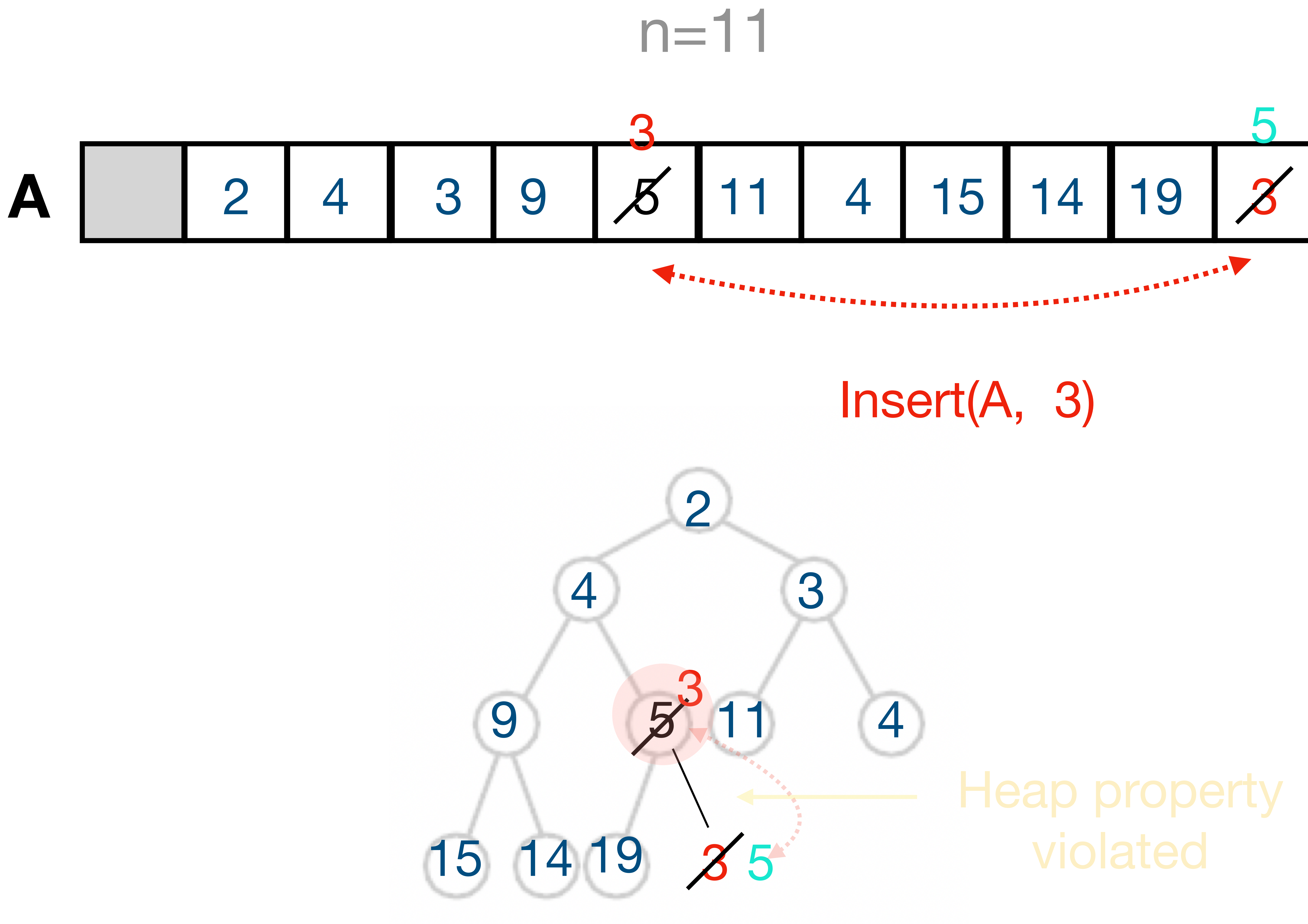
Insert(A, 3)



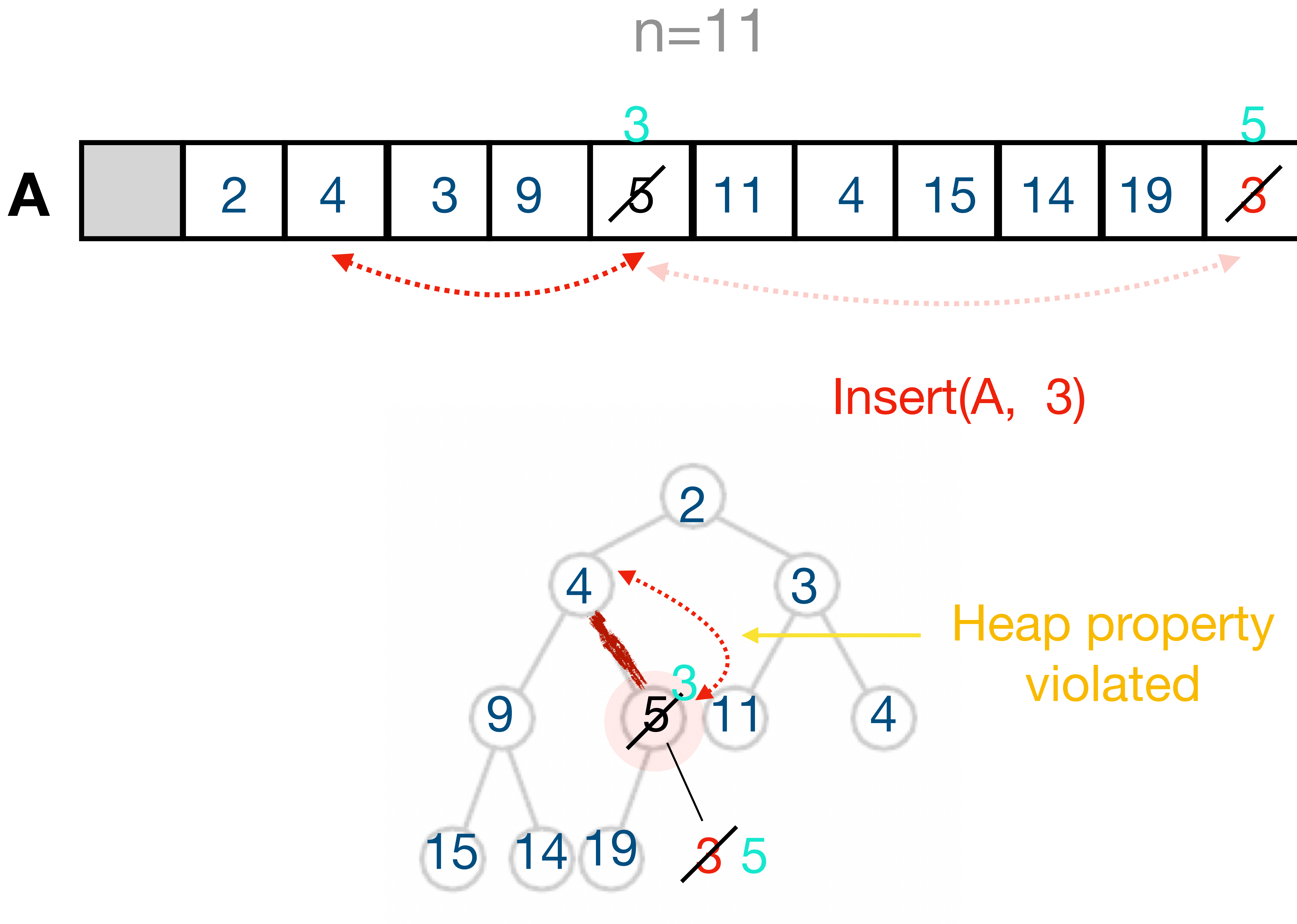
Inserting in a heap



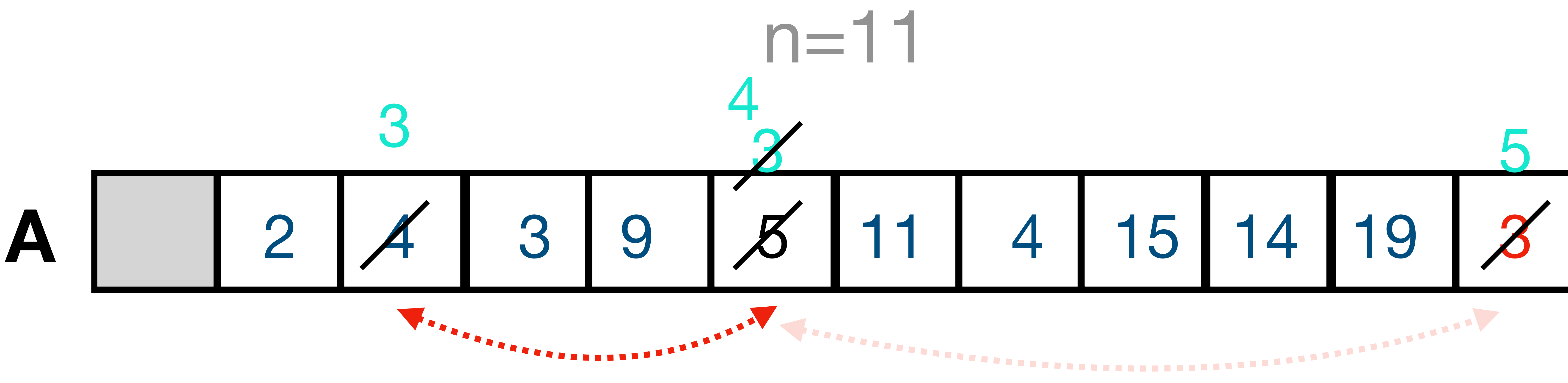
Inserting in a heap



Inserting in a heap



Inserting in a heap

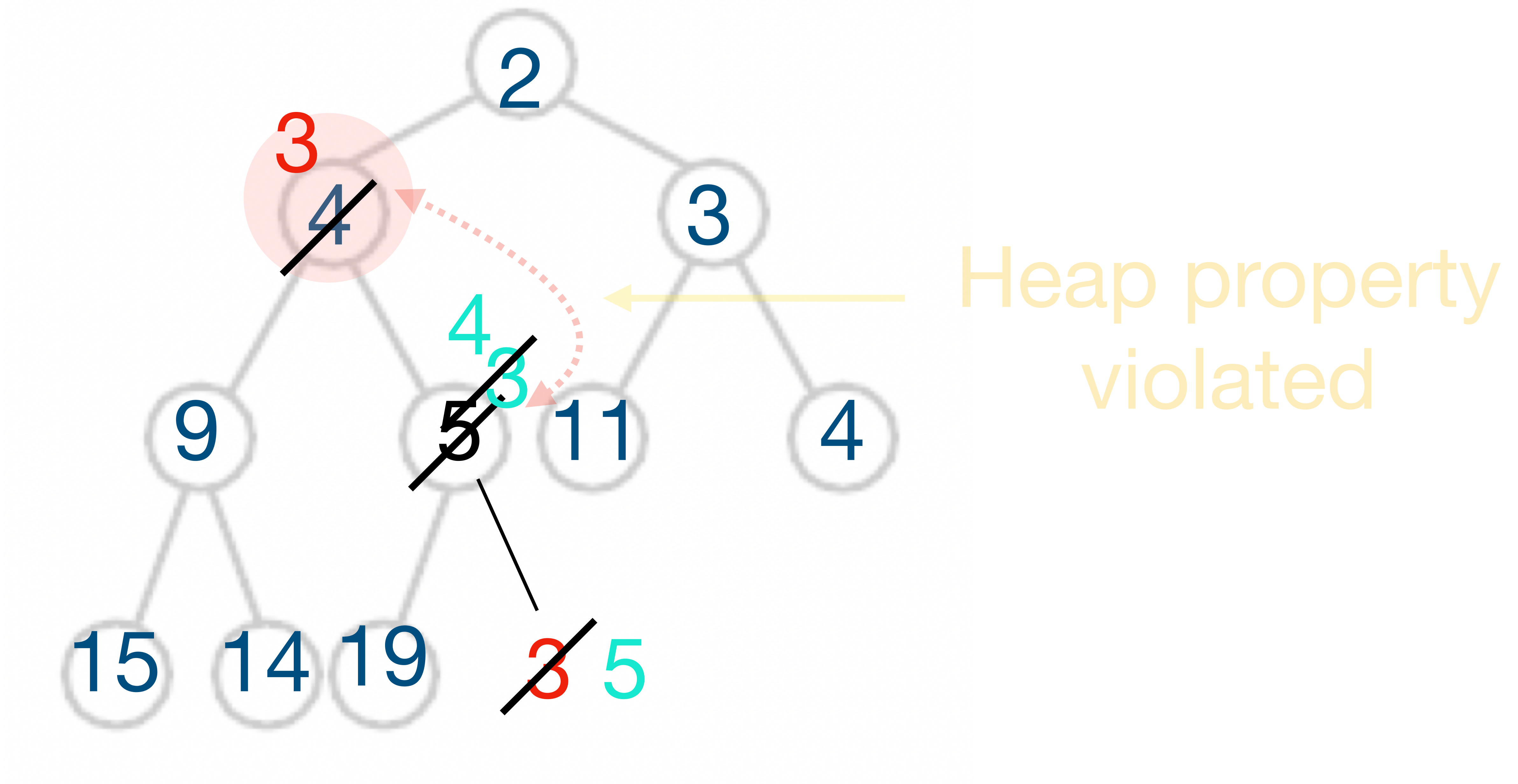


Insert(A, e)

1. Add e at the end of the heap

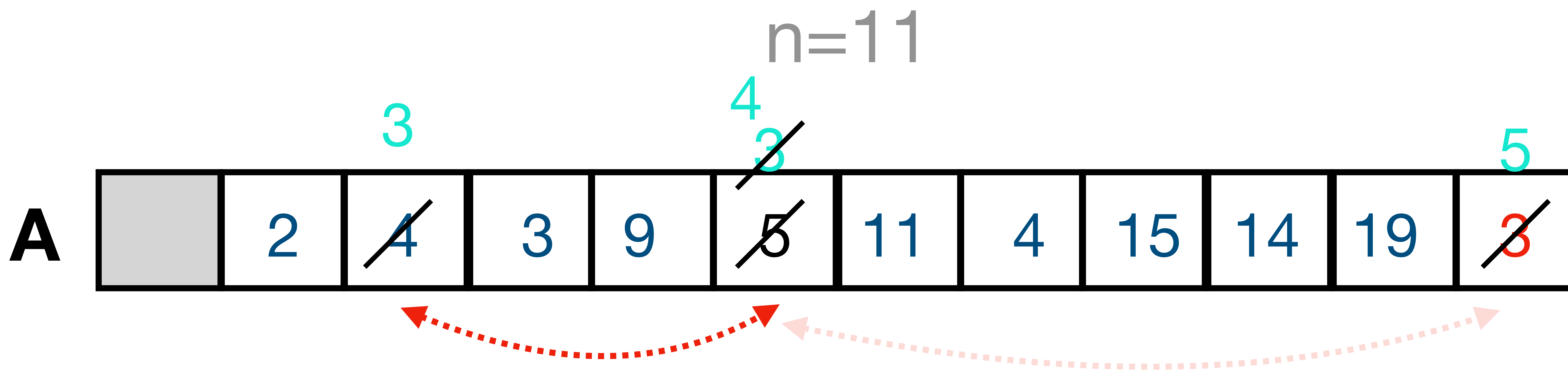
2. “Bubble-up” to restore heap property: swap e with its parent, and repeat

Insert(A, 3)



Why is this correct?

Inserting in a heap

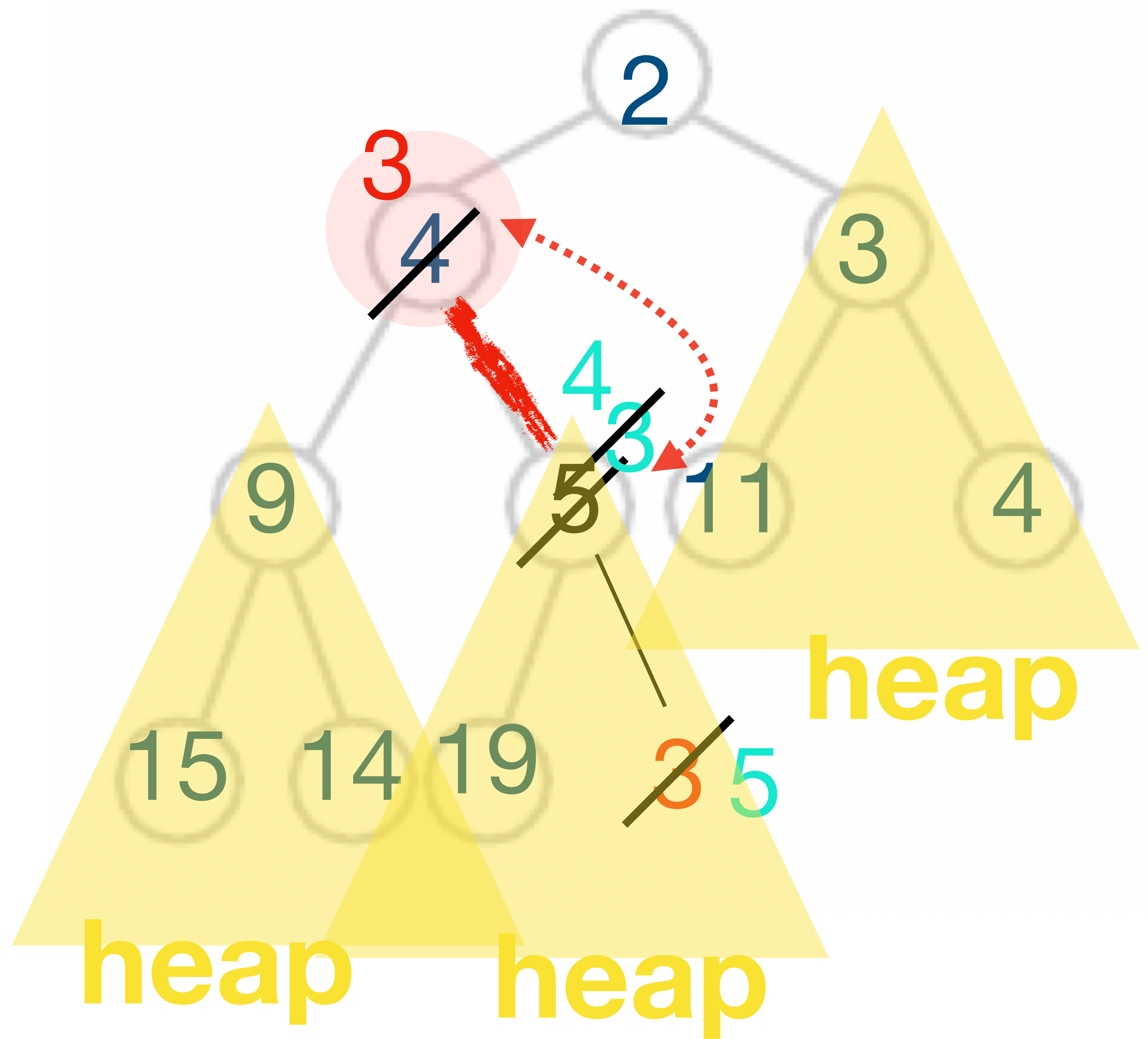


Insert(A, e)

1. Add e at the end of the heap
2. "Bubble-up" to restore heap property: swap e with its parent, and repeat

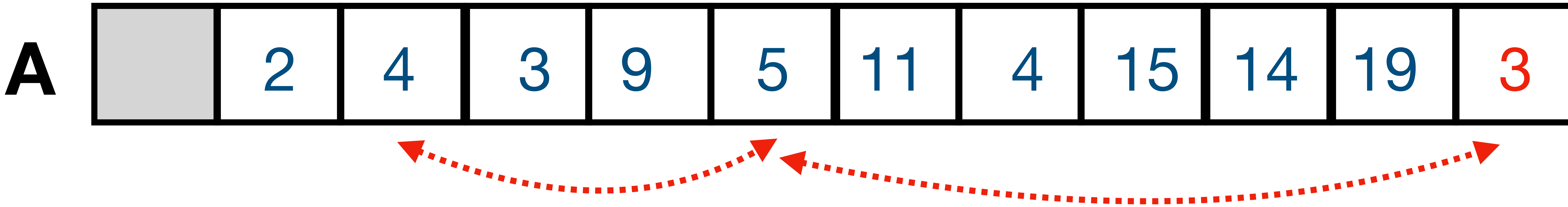
Why is this correct?

Insert(A, 3)



Inserting in a heap

n=11

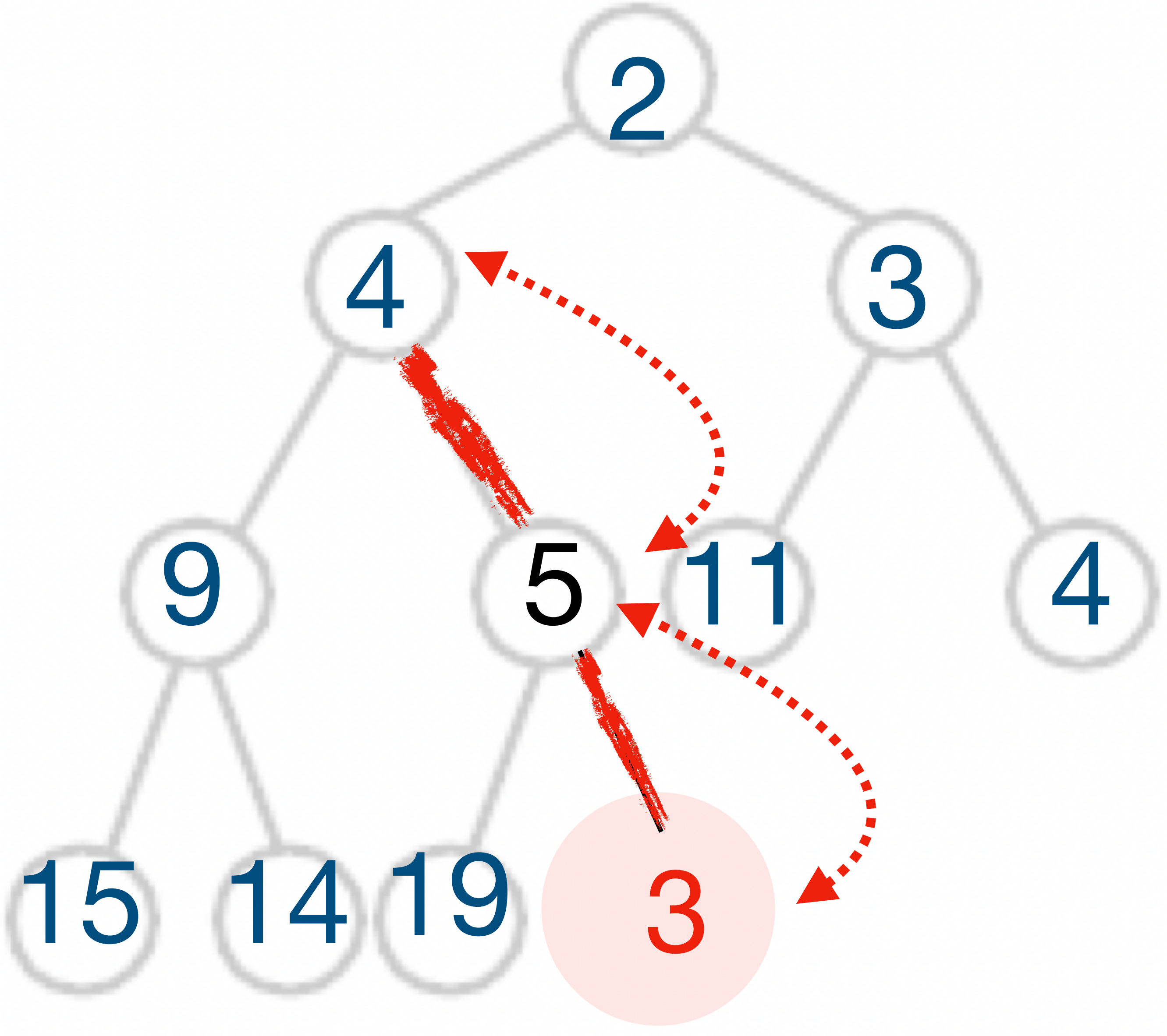


Insert(A, e)

1. Add e at the end of the heap

2. "Bubble-up" to restore heap property: swap e with its parent, and repeat

Insert(A, 3)



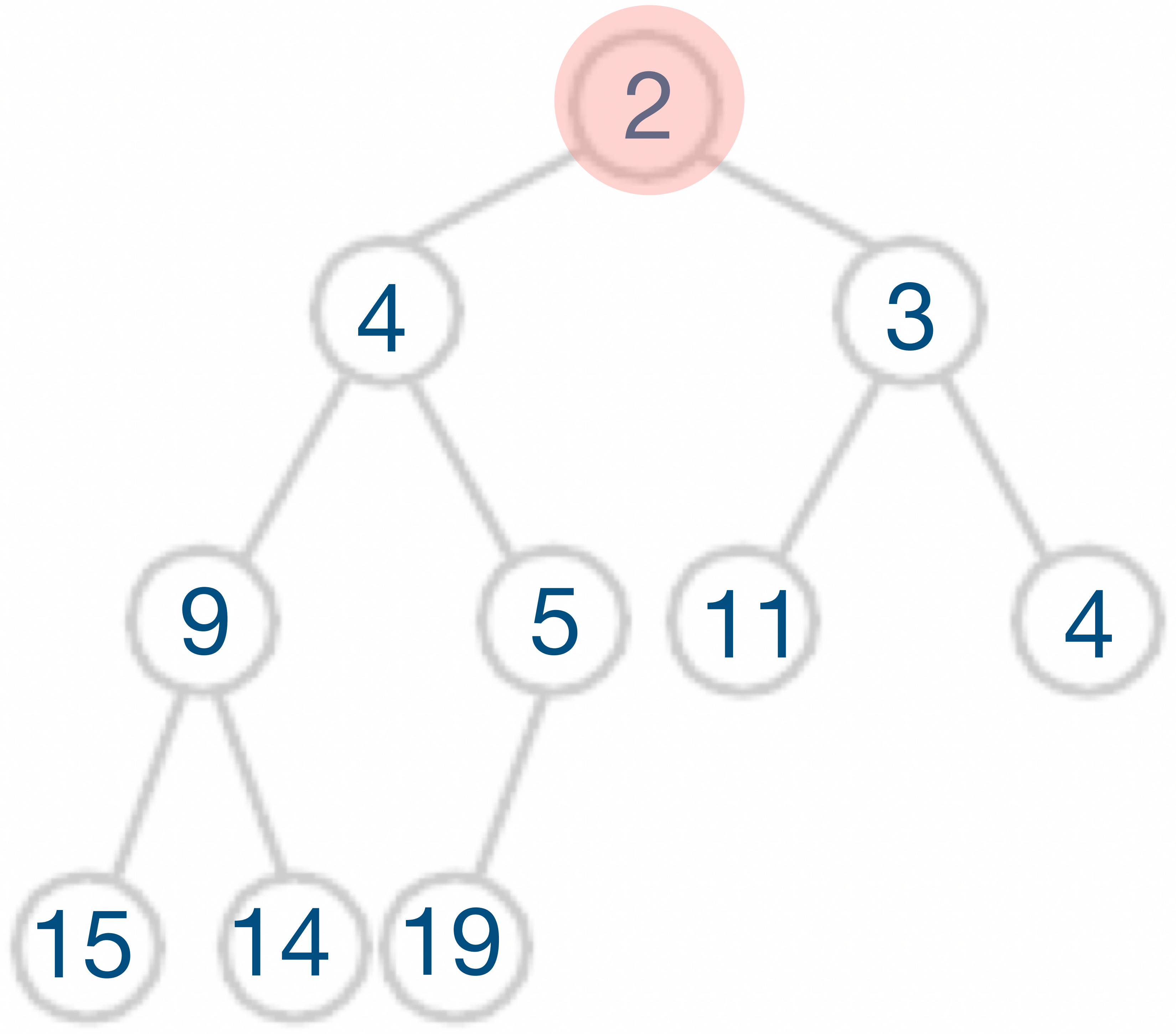
Run time: O(lg n)

DeleteMin in a heap

A		2	4	3	9	5	11	4	15	14	19

n=10

DeleteMin(A)

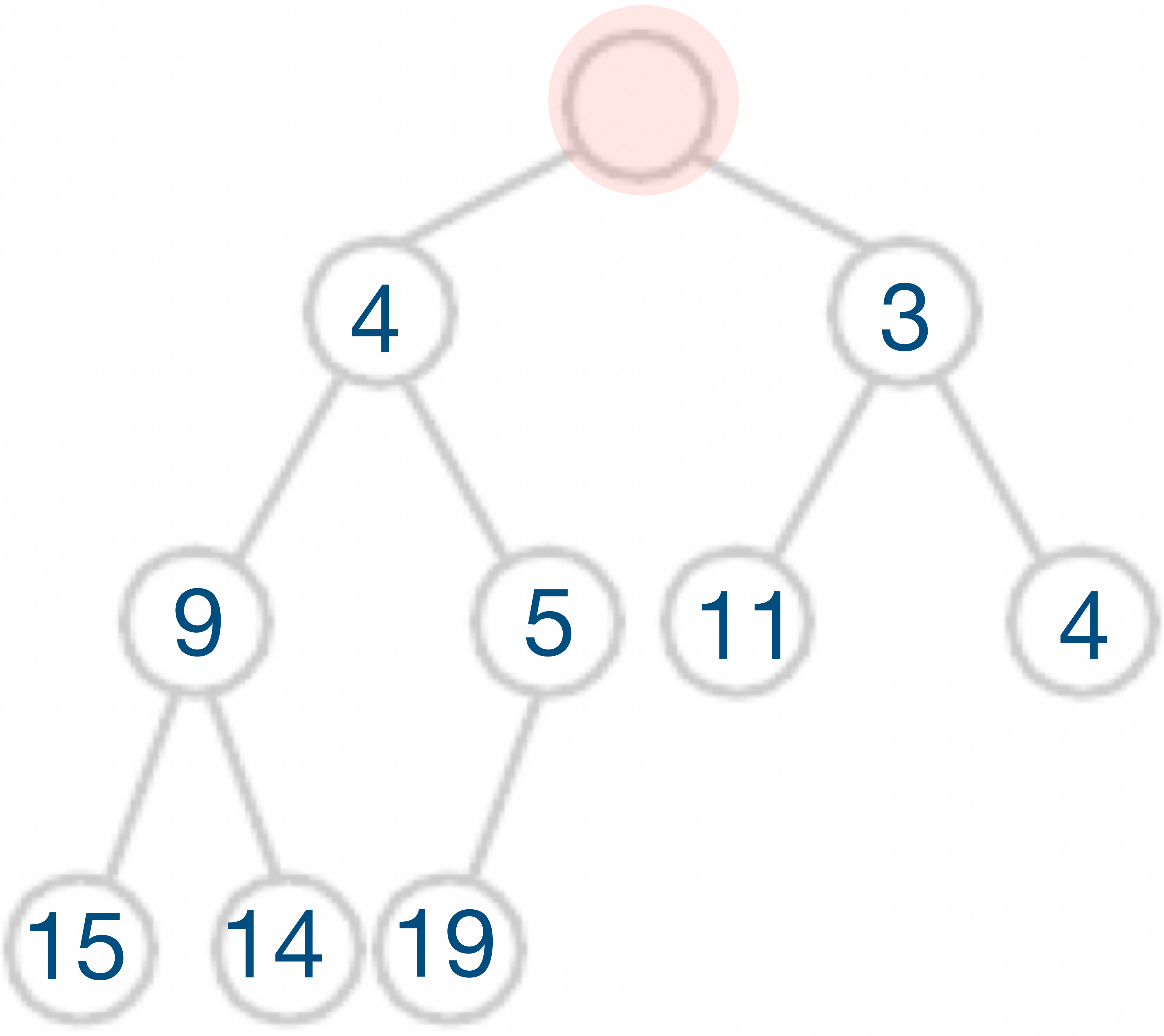


DeleteMin in a heap

A			4	3	9	5	11	4	15	14	19

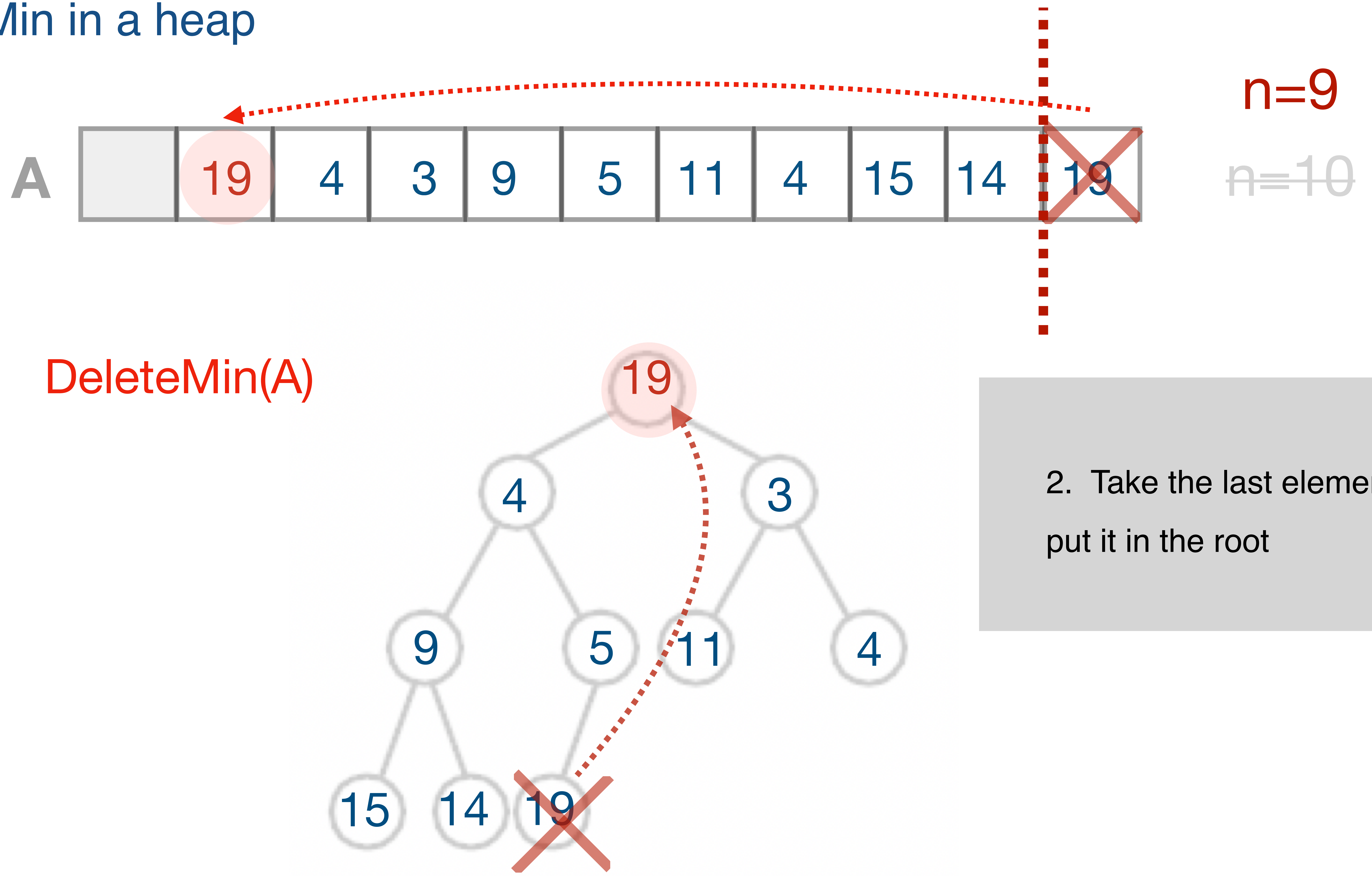
n=10

DeleteMin(A)



1. Save the element in the root (will return it)

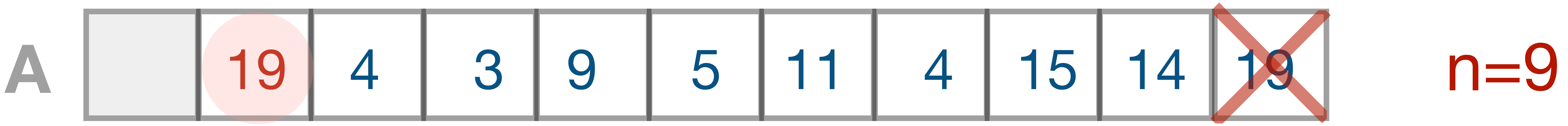
DeleteMin in a heap

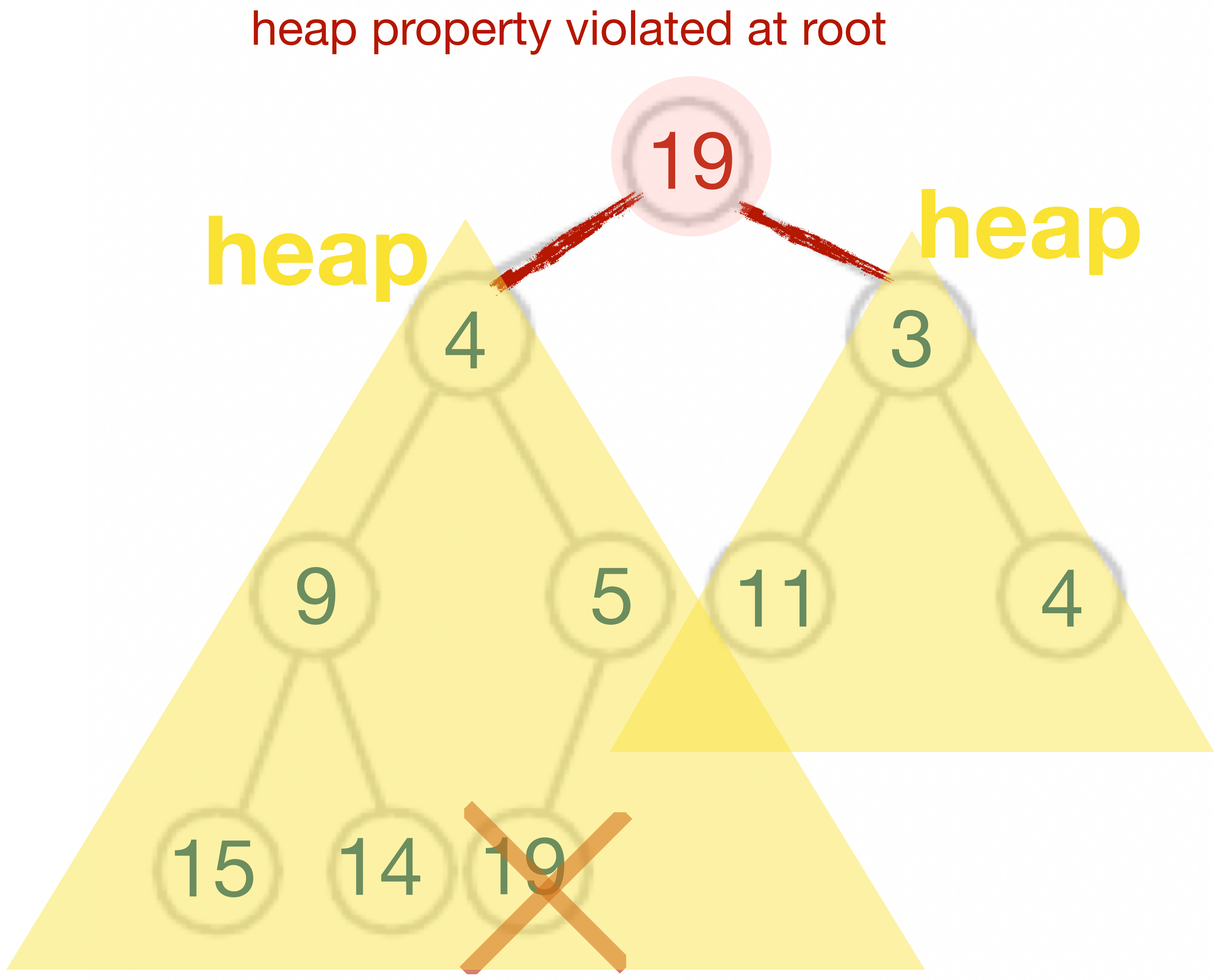


DeleteMin(A)

2. Take the last element and put it in the root

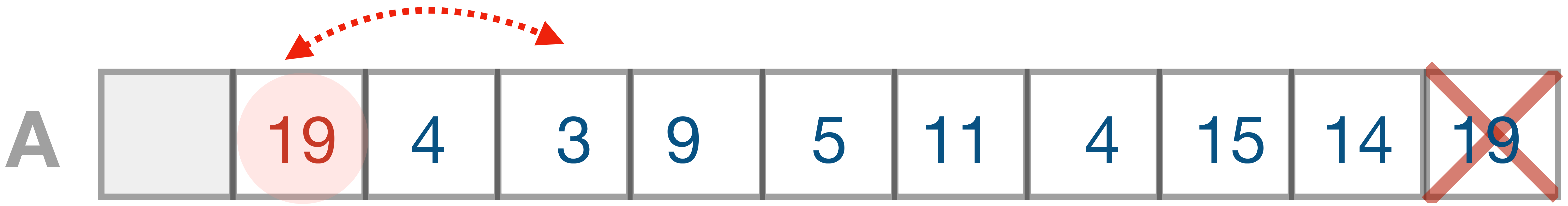
DeleteMin in a heap



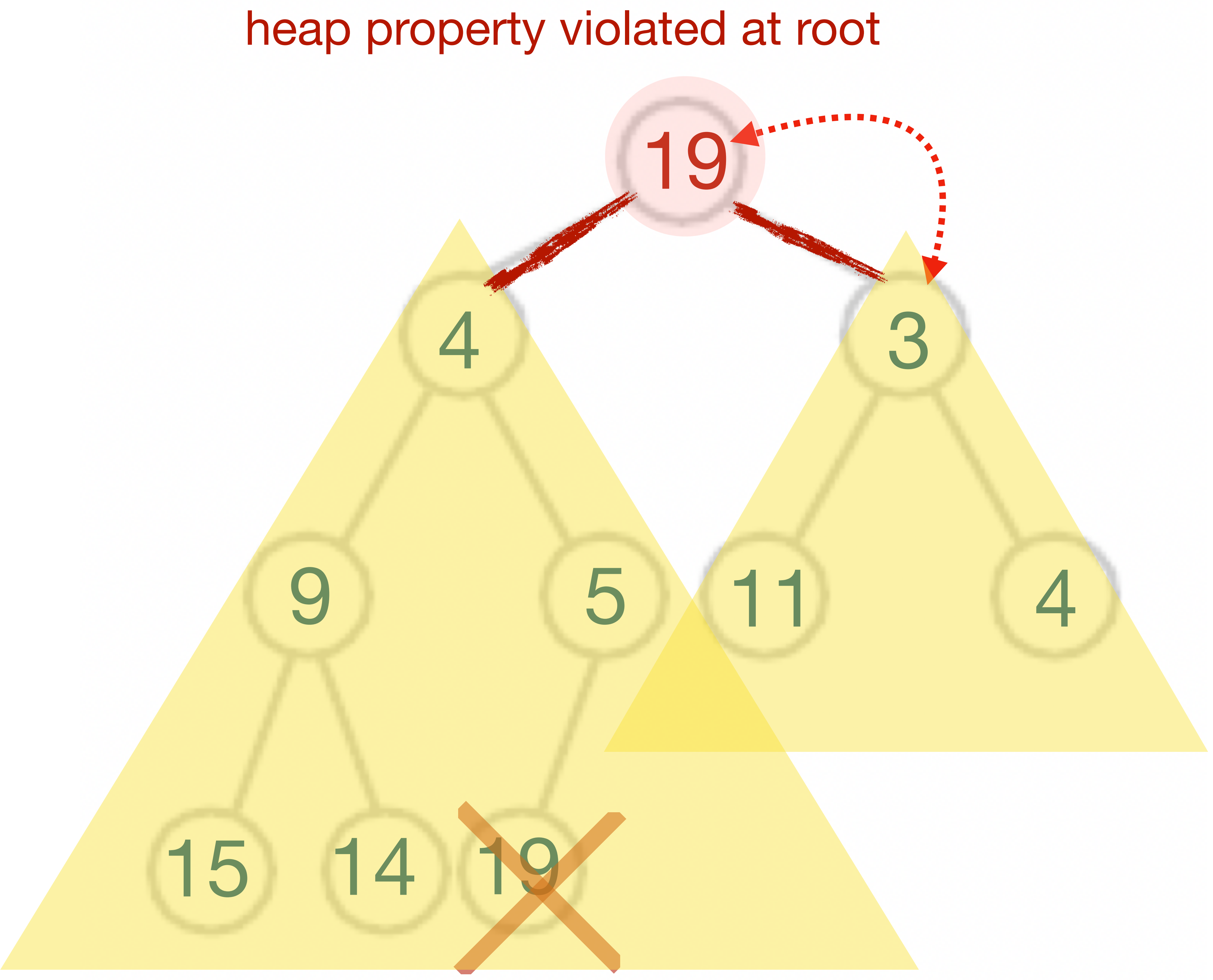


3. “Bubble-down” to restore heap property: swap root with its largest child, and repeat

DeleteMin in a heap

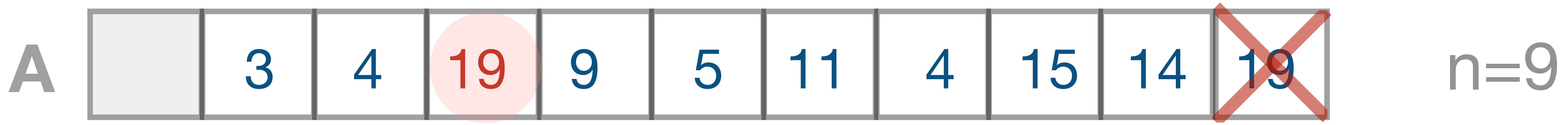


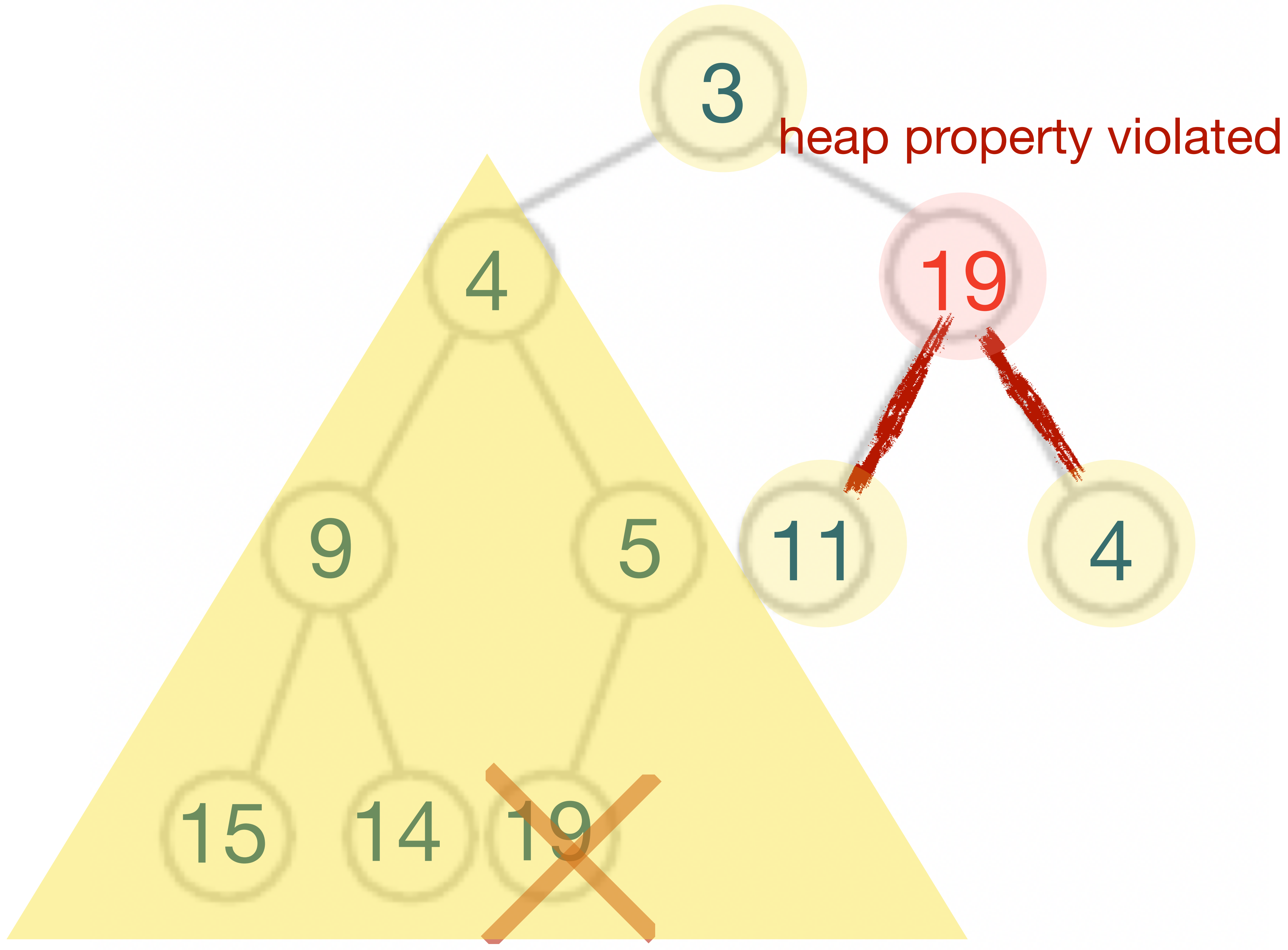
n=9



3. “Bubble-down” to restore heap property: swap root with its largest child, and repeat

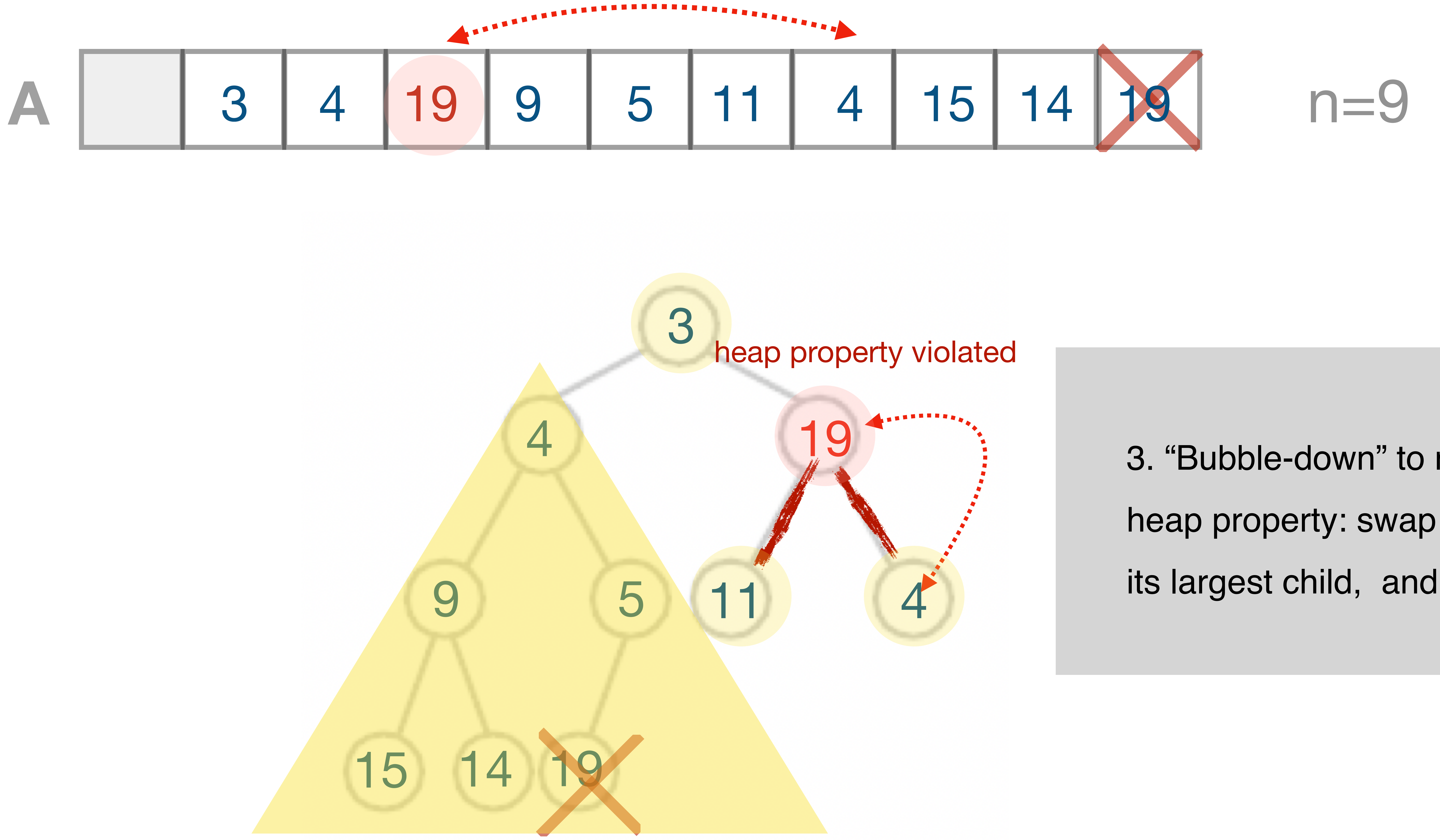
DeleteMin in a heap





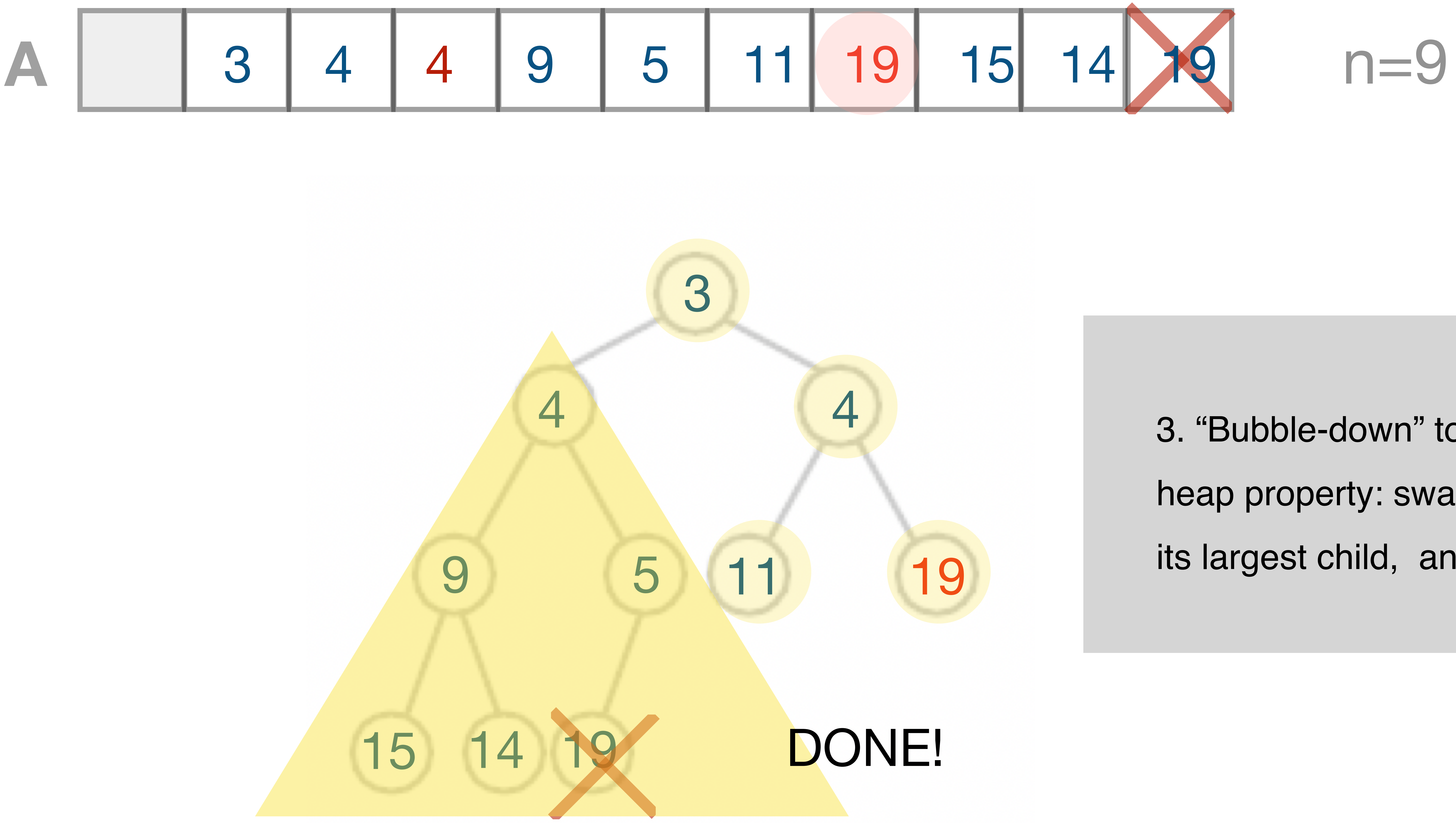
3. "Bubble-down" to restore heap property: swap root with its largest child, and repeat

DeleteMin in a heap



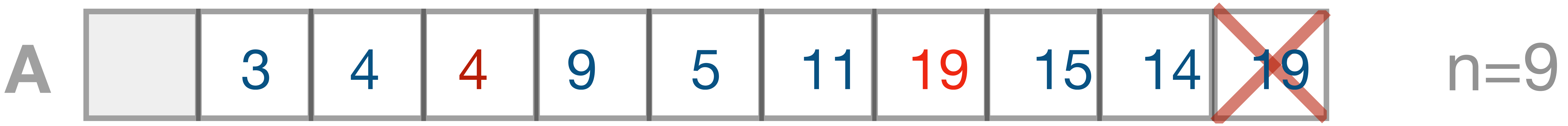
3. "Bubble-down" to restore heap property: swap root with its largest child, and repeat

DeleteMin in a heap

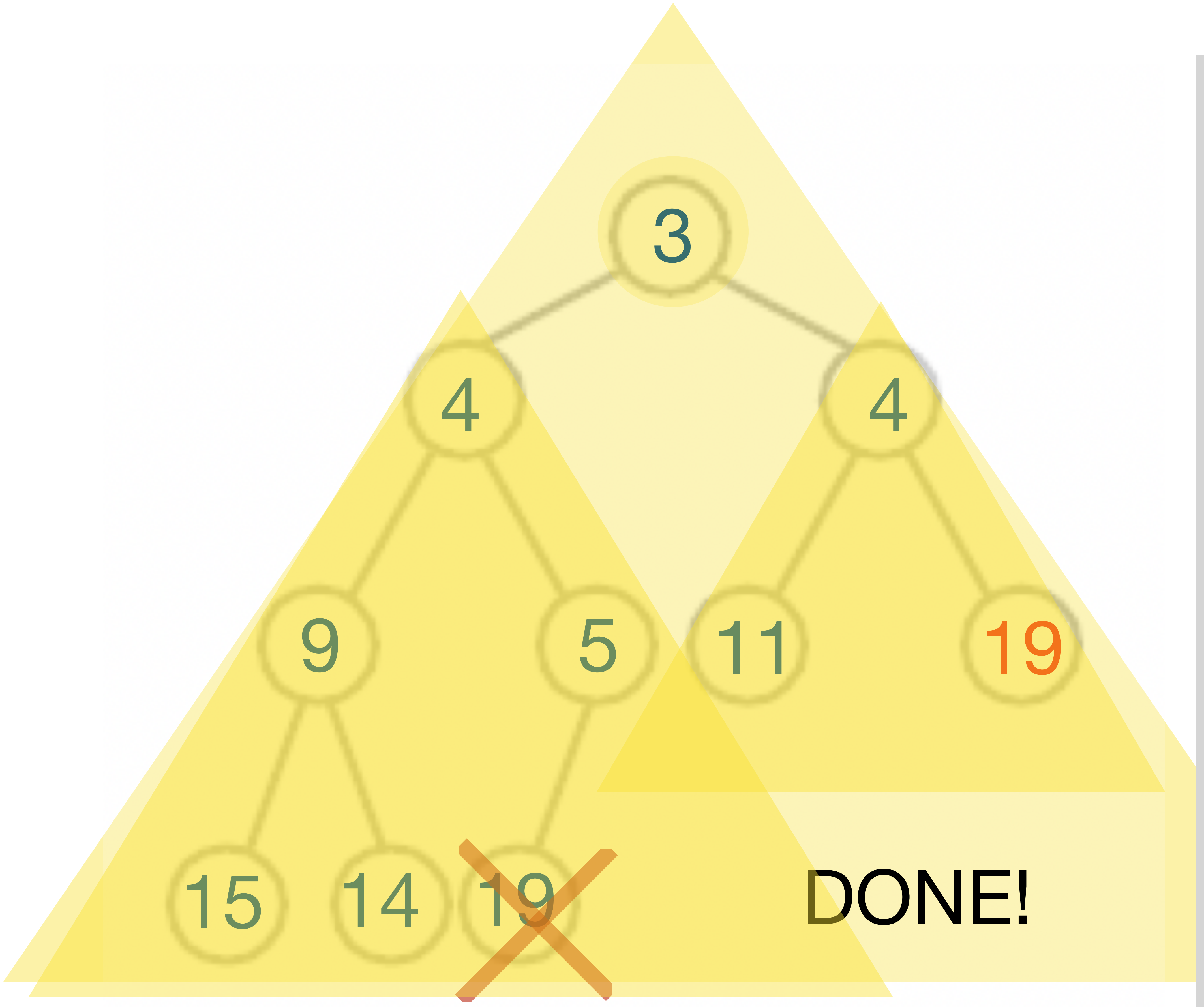


3. “Bubble-down” to restore heap property: swap root with its largest child, and repeat

DeleteMin in a heap



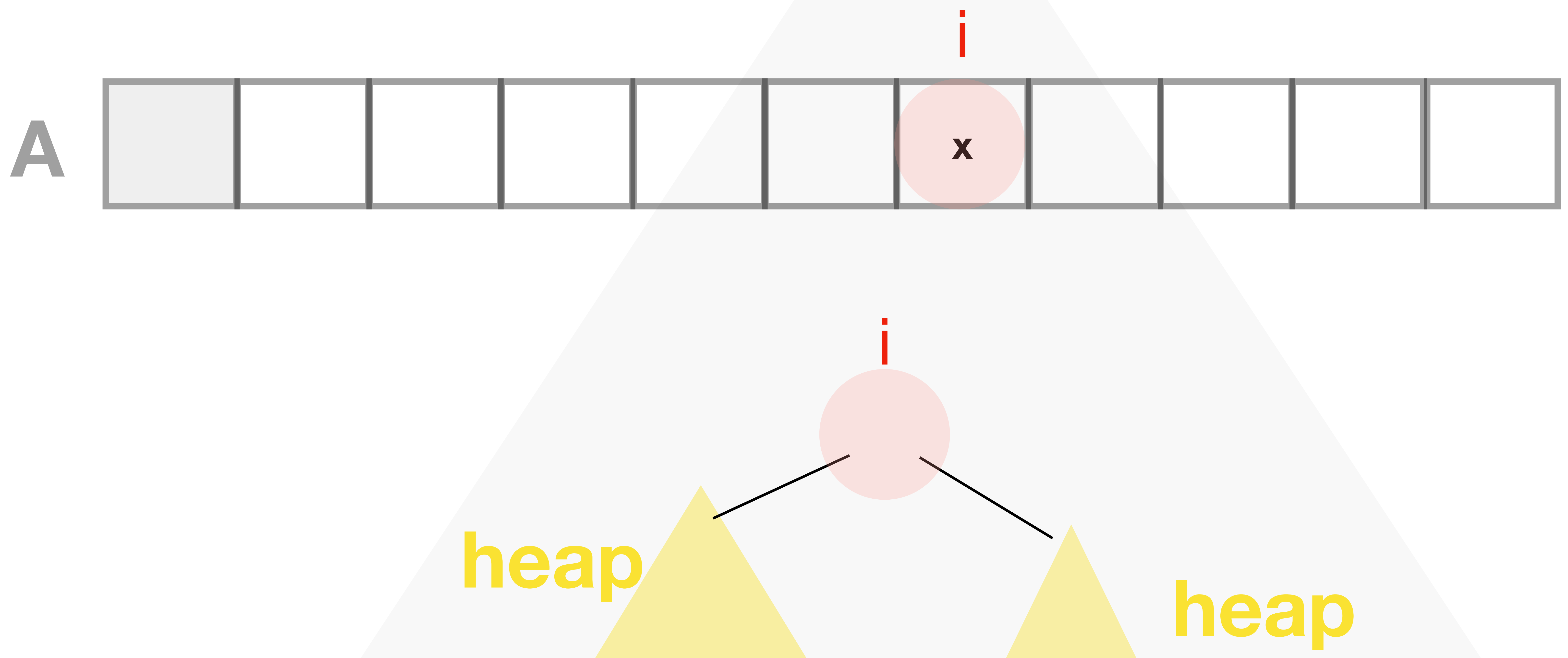
Run time: O(lg n)



DeleteMin(A)

1. Save the element in the root (will return it)
2. Take the last element and put it in the root
3. “Bubble-down” to restore heap property: swap root with its largest child, and repeat

Heapify(A, i):   makes a heap under i



- i is an index, $1 \leq i \leq n$
- Before calling Heapify)i): left(i) is a heap, right(i) is a heap, but heap property is violated at node i
- After calling Heapify (i): the subtree rooted at i is a heap

Heapify(A, i):   makes a heap under i

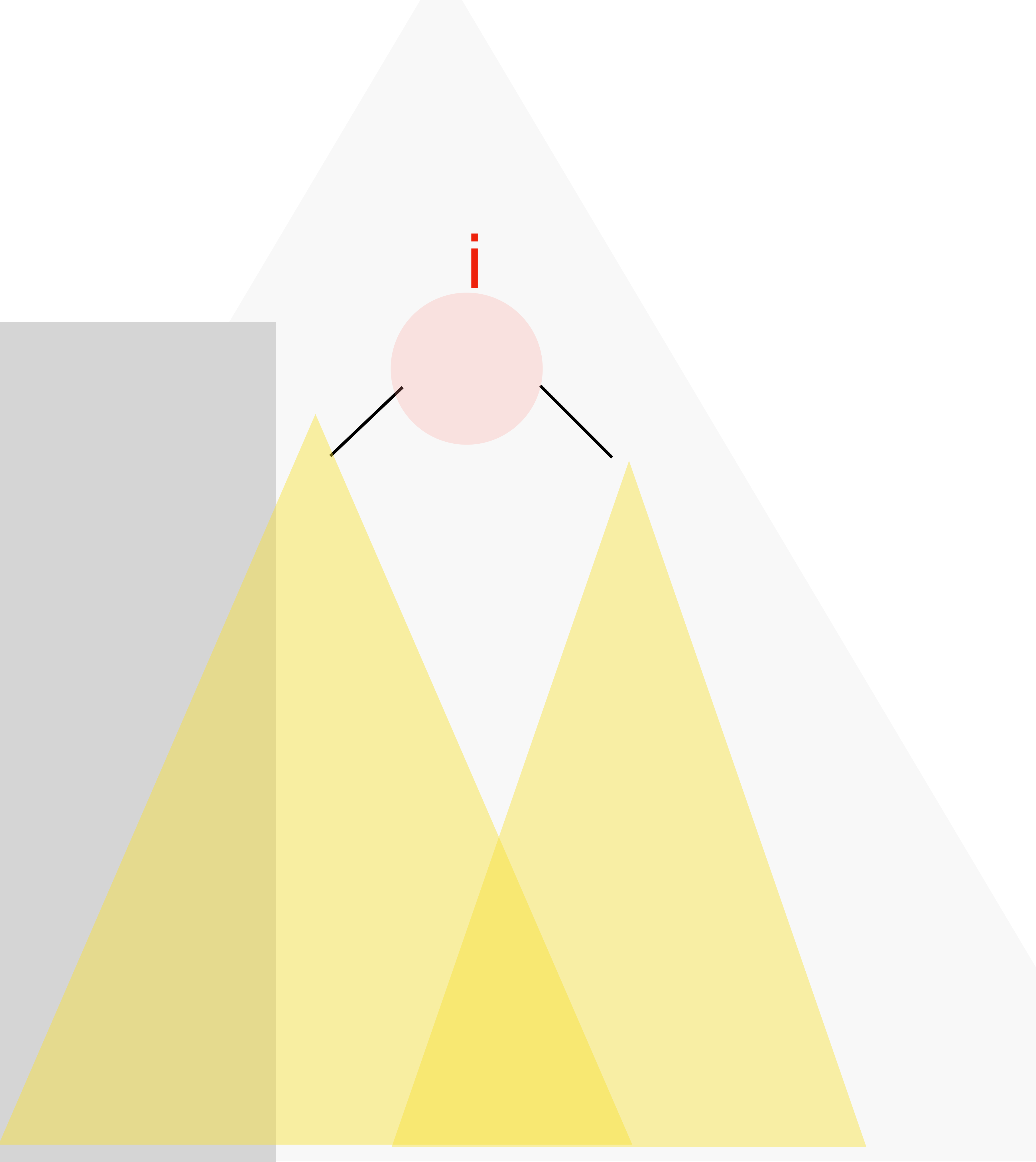
Heapify(A, i)

//find smallest of its children

- l = left(i),   r = right(i)
- if l <= heapsize(A) and A[l] < A[i]: smallest = l, else smallest = i
- if (r <= heapsize(A) and A[r] < A[smallest] : smallest = r

//swap and recurse

- if smallest ! = i:
    - exchange A[i] with A[smallest]
    - Heapify(A, smallest)



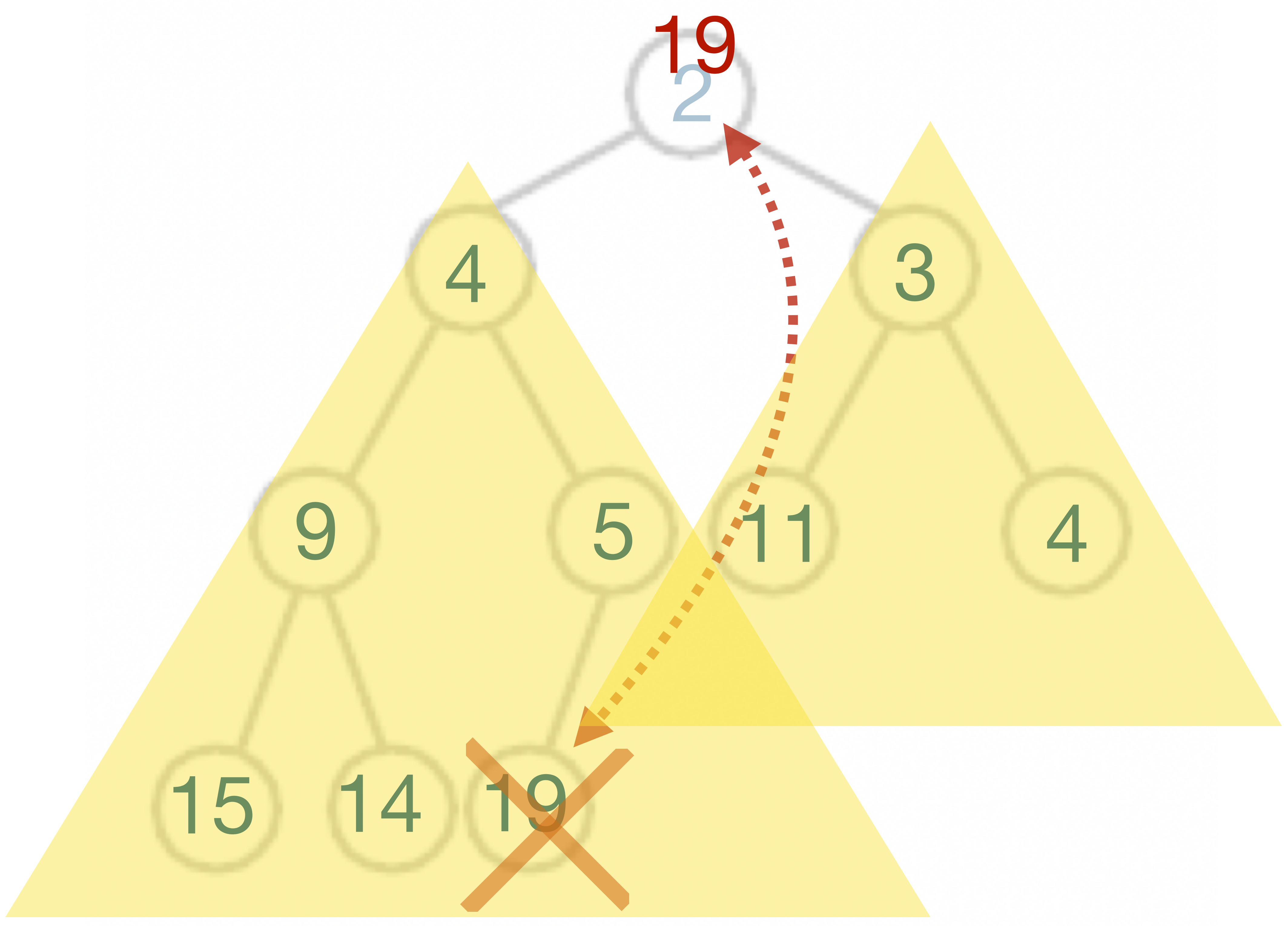
A

	2	4	3	9	5	11	4	15	14	19

n=10

Heap-Delete-Min(A)

- if heapsize(A) < 1: error “heap underflow”
- min = A[1]
- A[1] = A[heapsize(A)]
- heapsize(A) - -
- Heapify(A, 1)
- return min



BuildHeap(A)

- A is an array
- Buildheap makes A into a heap, in place.

- Not in place: Can we do it? How?
- In place: the idea is to call heapify to gradually make A into a heap.

```
BUILDHEAP-smart (A)
  – For i = n/2 down to 1: HEAPIFY-DOWN(i)
```

- Why is this correct?
- Run time: O(n)

Heapsort(A)

- **The problem**: A is an array. Sort A with a heap.

- Not in place:
    - Can we do it? How?

Heapsort(A)

- **The problem**: A is an array. Sort A with a heap.

- Not in place:
    - Can we do it? How?
    - Regular sort using a PQ: insert all elements into a PQ, then deleteMin one at a time.
    - Run time: O(n x insert + n x delete-min) = O(n lg n)

Can we do this (sort with a heap) in place?

Heapsort(A)

- **The problem**: A is an array. Sort A with a heap in place.

- In place:

```
Heapsort(A)
    Convert A into a max-heap
    //Repeatedly Delete-Max and put it at the end of the array
    for i=0 to n-1: A[n-i] = DELETE-MAX(A)
```

Heapsort(A)

- **The problem**: A is an array. Sort A with a heap in place.

- In place:

```
Heapsort(A)
    Convert A into a max-heap
    //Repeatedly Delete-Max and put it at the end of the array
    for i=0 to n-1: A[n-i] = DELETE-MAX(A)
```

Run time: Buildheap + n x Delete-Max ==> O(n lg n)

Heaps: summary

Heaps are arrays + heap property

- Insert(A, e)
- Delete-Min()
- Heapify(A, i)
- Buildheap(A)
- Heapsort (A)

O(lg n)

O(n)

O(n lg n), in place

- Cannot **Search** efficiently in a heap
- Generalize to 3-heaps, …. d-heaps