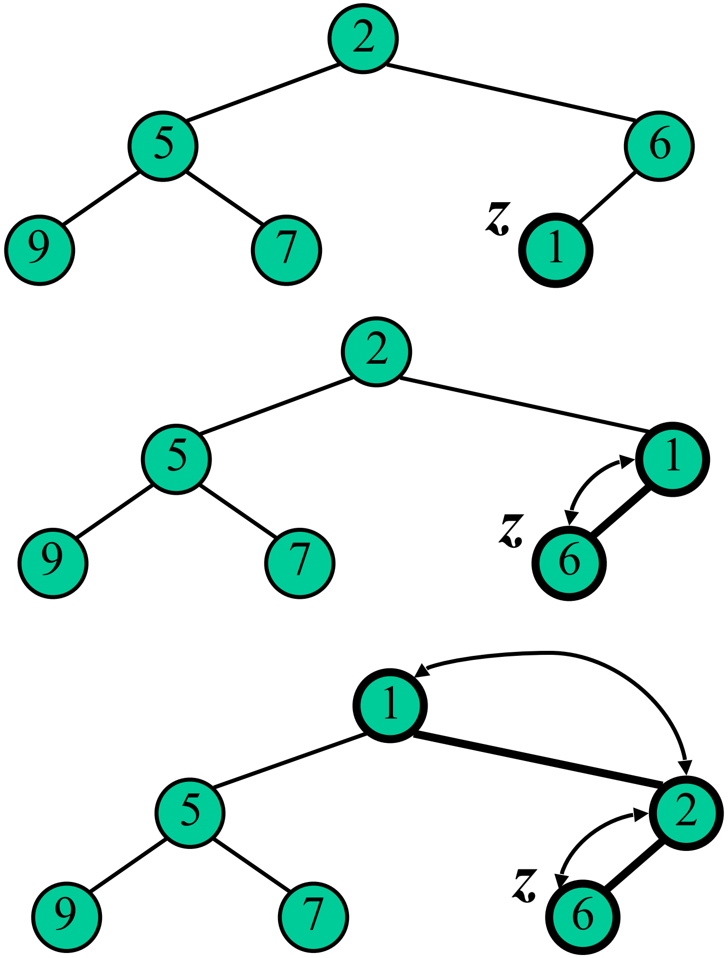
CS206

Array-based Heaps

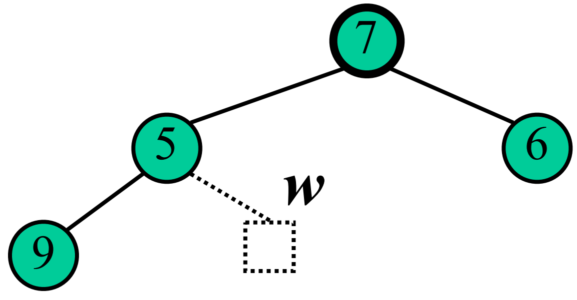
Upheap

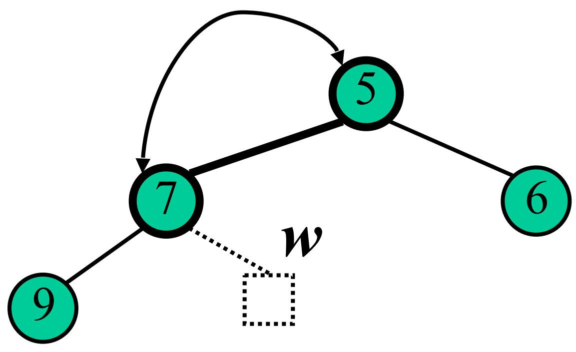
- Restore heap order
  - swap upwards
  - stop when finding a smaller parent
  - or reach root
- $O(\log n)$



Downheap

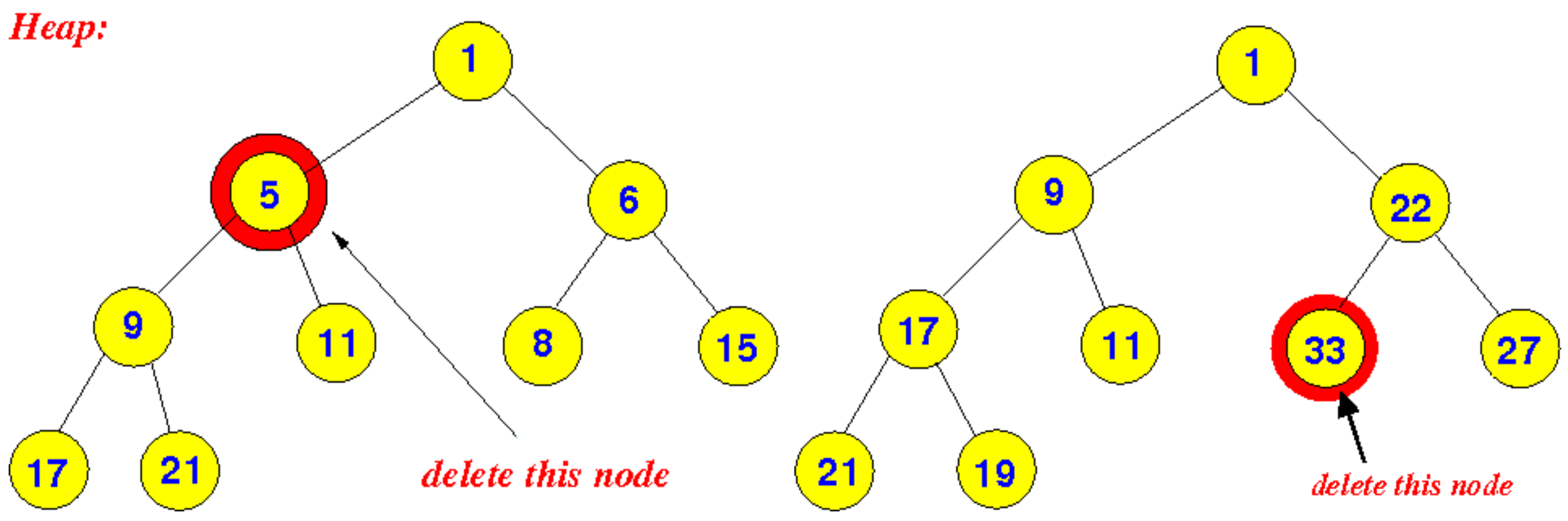
- Restore heap order
  - swap downwards
  - swap with smaller child
  - stop when finding larger children
  - or reach a leaf
- $O(\log n)$





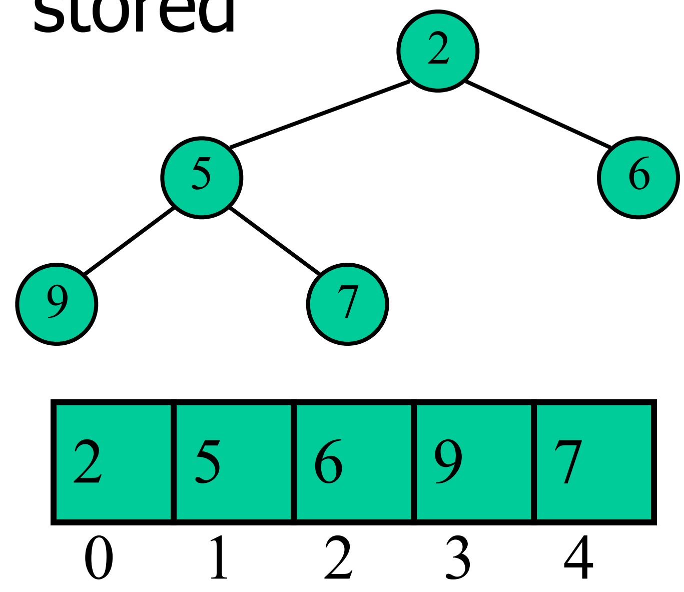
General Removal

- swap with last node
- delete last node
- may need to upheap or downheap



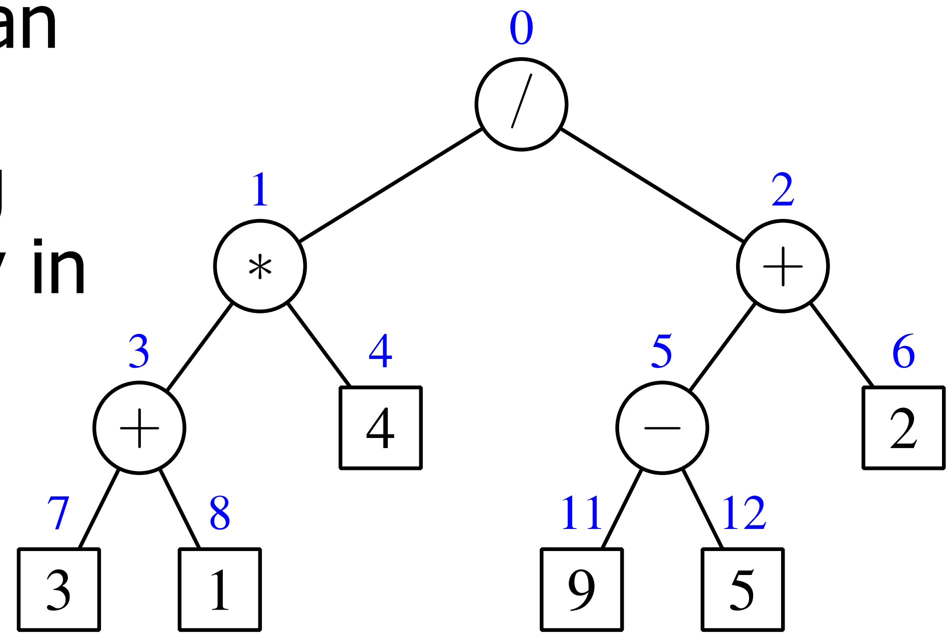
Array-based Heap

- Heap is a complete binary tree, thus is particularly suited for array-based implementation
- Array/ArrayList of length n for heap with n keys
- node at index i
  - left child $2i + 1$
  - right child $2i + 2$
- peek – element at 0
- poll – remove 0
- no links/references stored



Array-based Binary Tree

- The numbering can then be used as indices for storing the nodes directly in an array



/	*	+	+	4	−	2	3	1			9	5		
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Heap-based PriorityQueue

```
public class ArrayHeap<E extends
Comparable<E>> extends
ArrayBinaryTree<E> implements
PriorityQueue<E>{
  E peek();
  E poll();
}
```

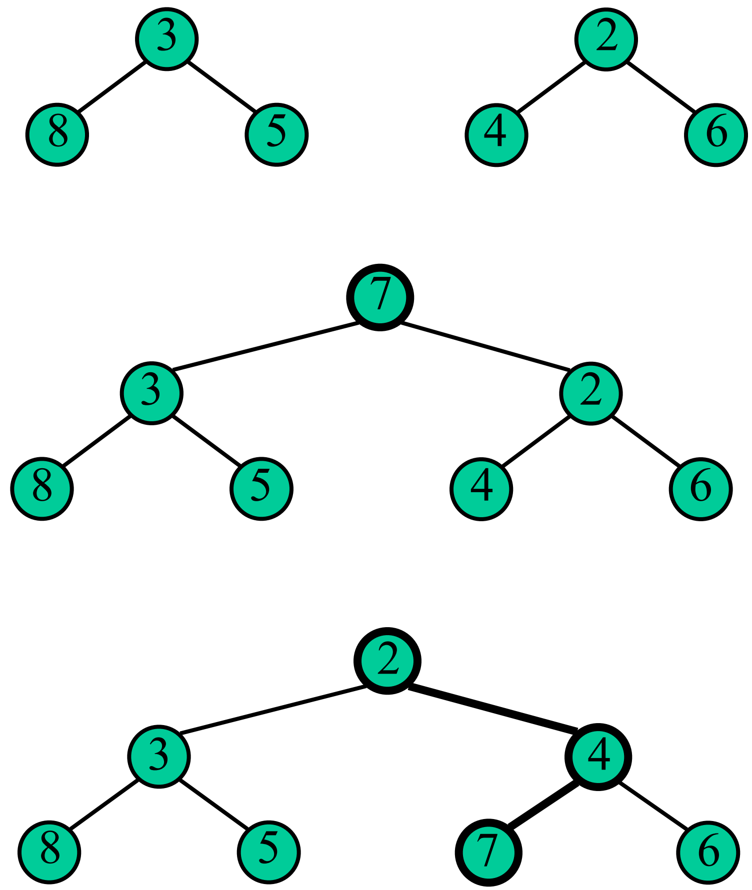
Write poll at chalkboard

Update Key

- What should happen when you change the key of an existing element in a heap?
- What are the cases?
  - `increaseKey`
  - `decreaseKey`

Merging Two Heaps

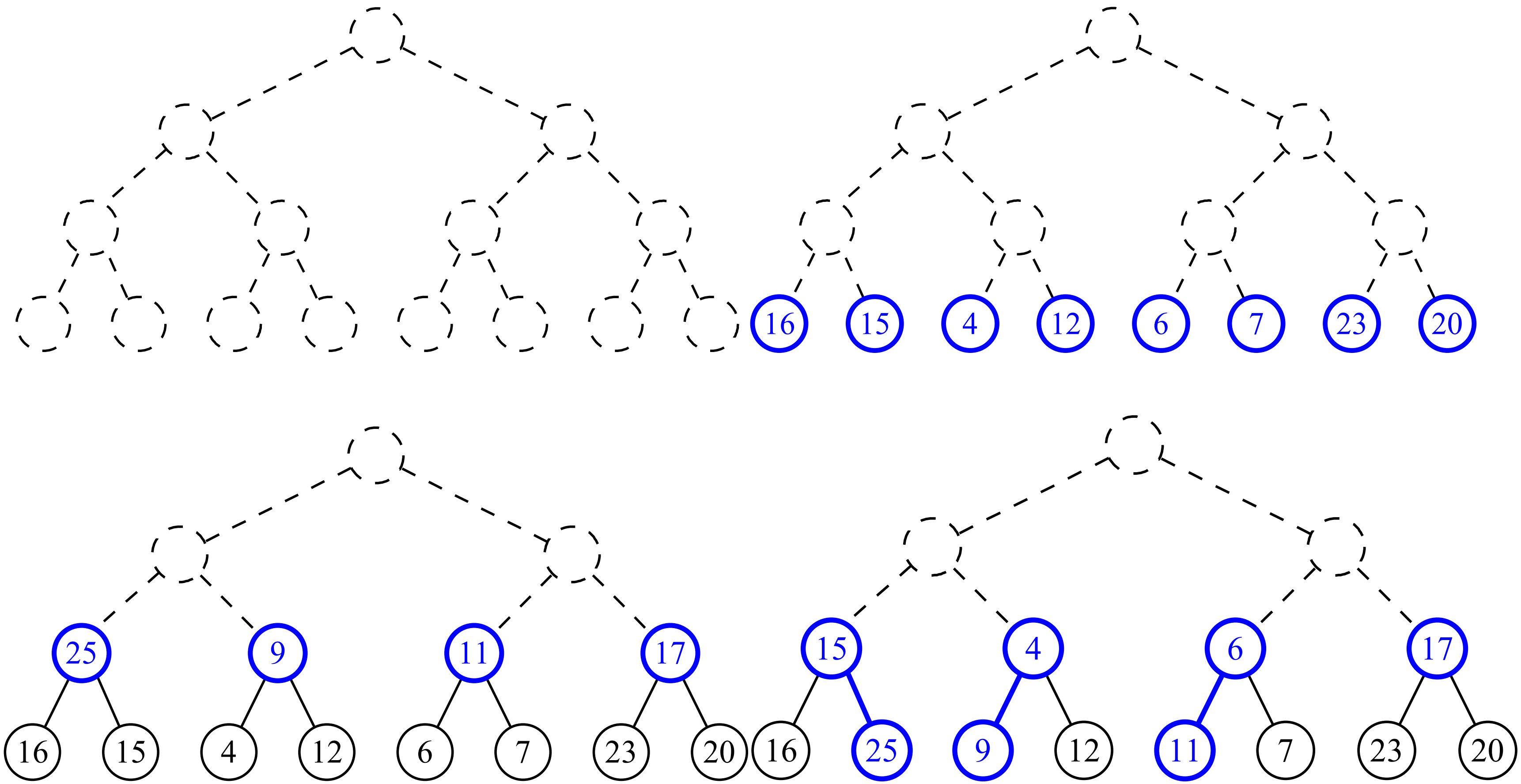
- Given two heaps and a new key k
- Create a new heap with k as root and the two heaps as subtrees
- downheap on k to restore heap order
- $O(logn)$



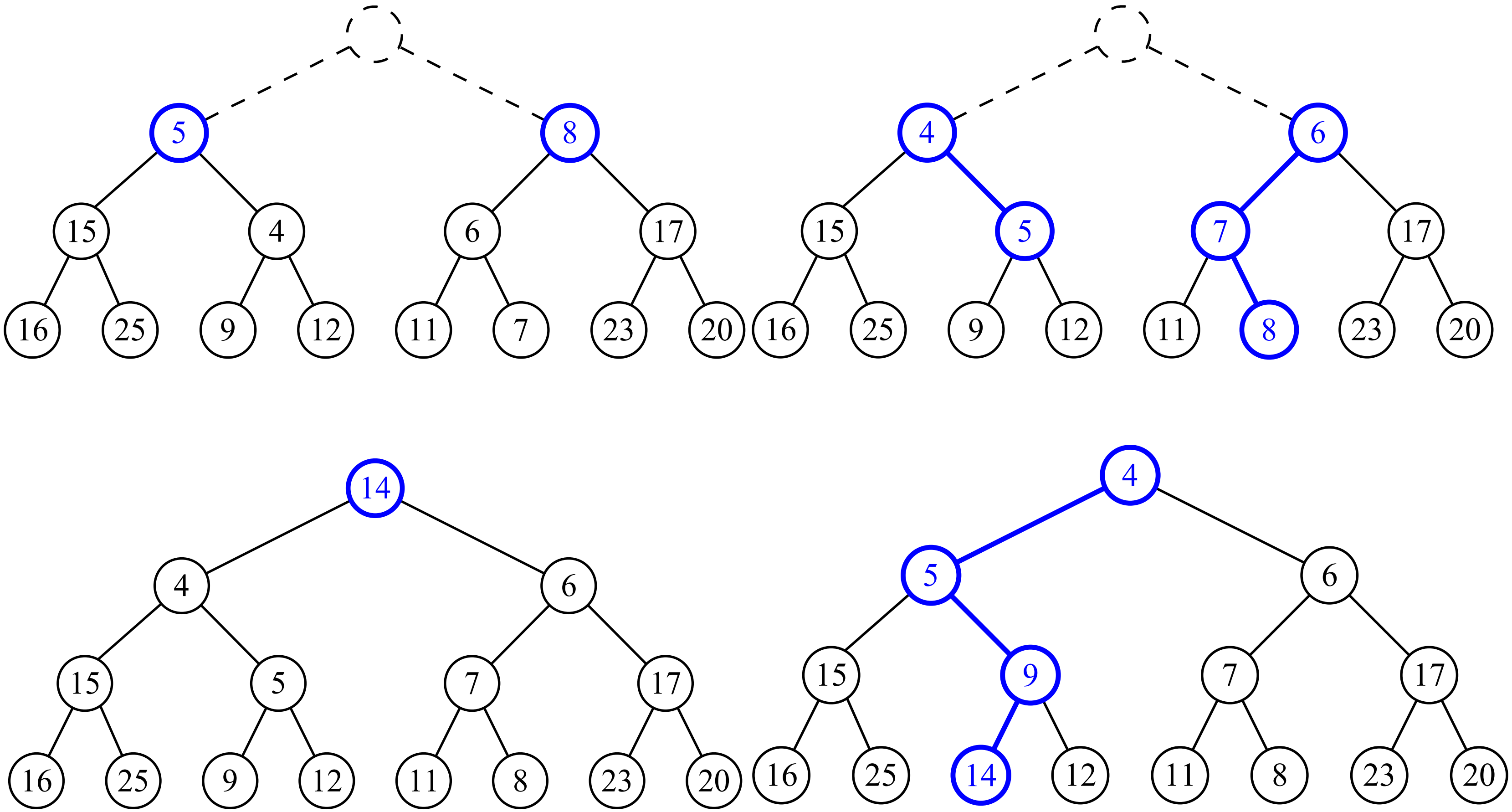
Bottom-up Construction

- Complexity of constructing a heap with n elements?
  - Call insert n times - $O(nlogn)$
  - When does $O(nlogn)$ occur?
- More efficient alternative
  1. construct $(n+1)/2$ elementary heaps storing one entry each
  2. merge pairwise into $(n+1)/4$ larger heaps

heapify


16 15 4 12 6 7 23 20
25 9 11 17
16 15 4 12 6 7 23 20
15 4 6 17
16 25 9 12 11 7 23 20

heapify


5 8 4 6
15 4 6 17 15 5 7 17
16 25 9 12 11 7 23 20 16 25 9 12 11 8 23 20
14 4
4 6 5 6
15 5 7 17 15 9 7 17
16 25 9 12 11 8 23 20 16 25 14 12 11 8 23 20

Analysis

- $n/4 + n/8 + \ldots + 1 = O(n)$ merges
  - but O() ignores constants
  - O(n) yes, but really n/2 merges
- Each merge is $O(logn)$ which would suggest O(nlogn)
  - but first merge cost is 1 comparison
  - figuring the max number of comparisons for each merge
- n/4*1 + n/8*2 +n/16*3 … + 1*logn = O(n)