School of Computer Science

COMP SCI 1103/2103 Algorithm Design & Data Structure

Heaps and Heap Sort

Overview

- Last lecture:
  - (binary) heap
  - insert, deleteMin
  - siftUp, siftDown

- This lecture:
  - build heap
  - heap sort

Build heap

- We can build a binary heap by inserting the *n* elements one after the other.
  - Implies runtime O(n log n)

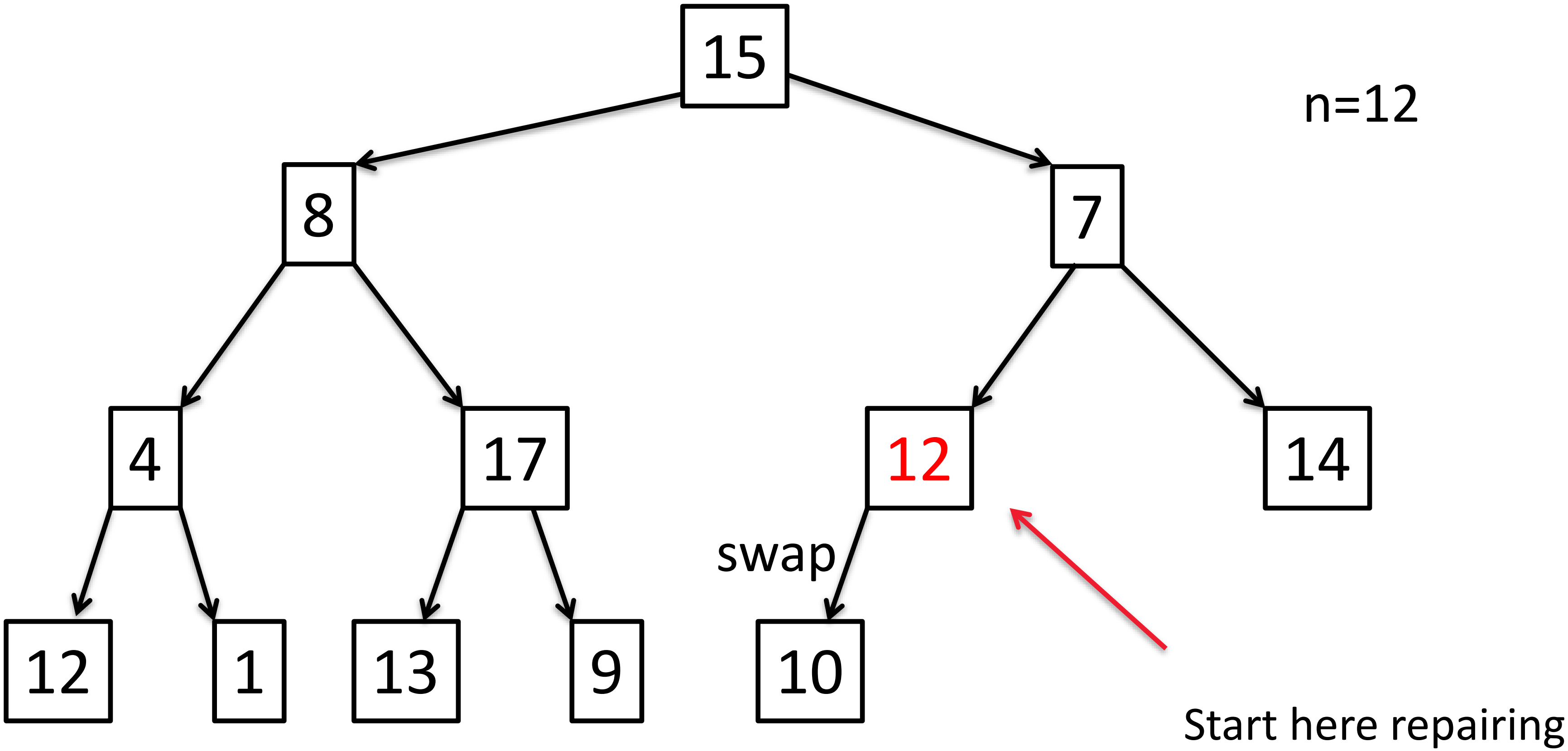
Build heap

- We can build a binary heap by inserting the n elements one after the other.
  - Implies runtime O(n log n)

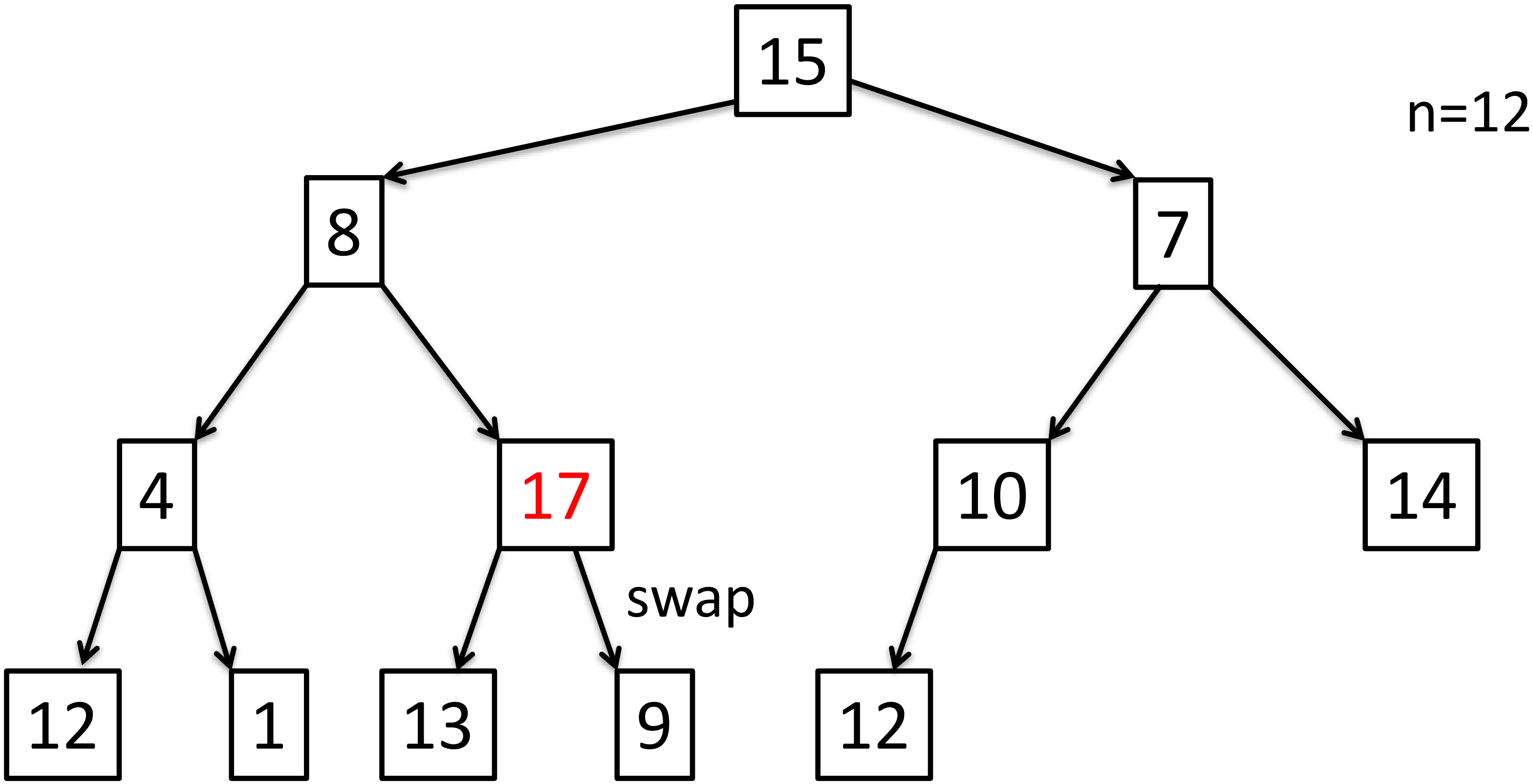
- Asssume the heap property holds for all subtrees of height k, we can establish the heap property for height k+*1* by siftDown
- buildHeapBackwards

  $$\textbf{for } i \coloneqq \lfloor n/2 \rfloor \textbf{ downto } 1 \textbf{ do } siftDown(i)$$

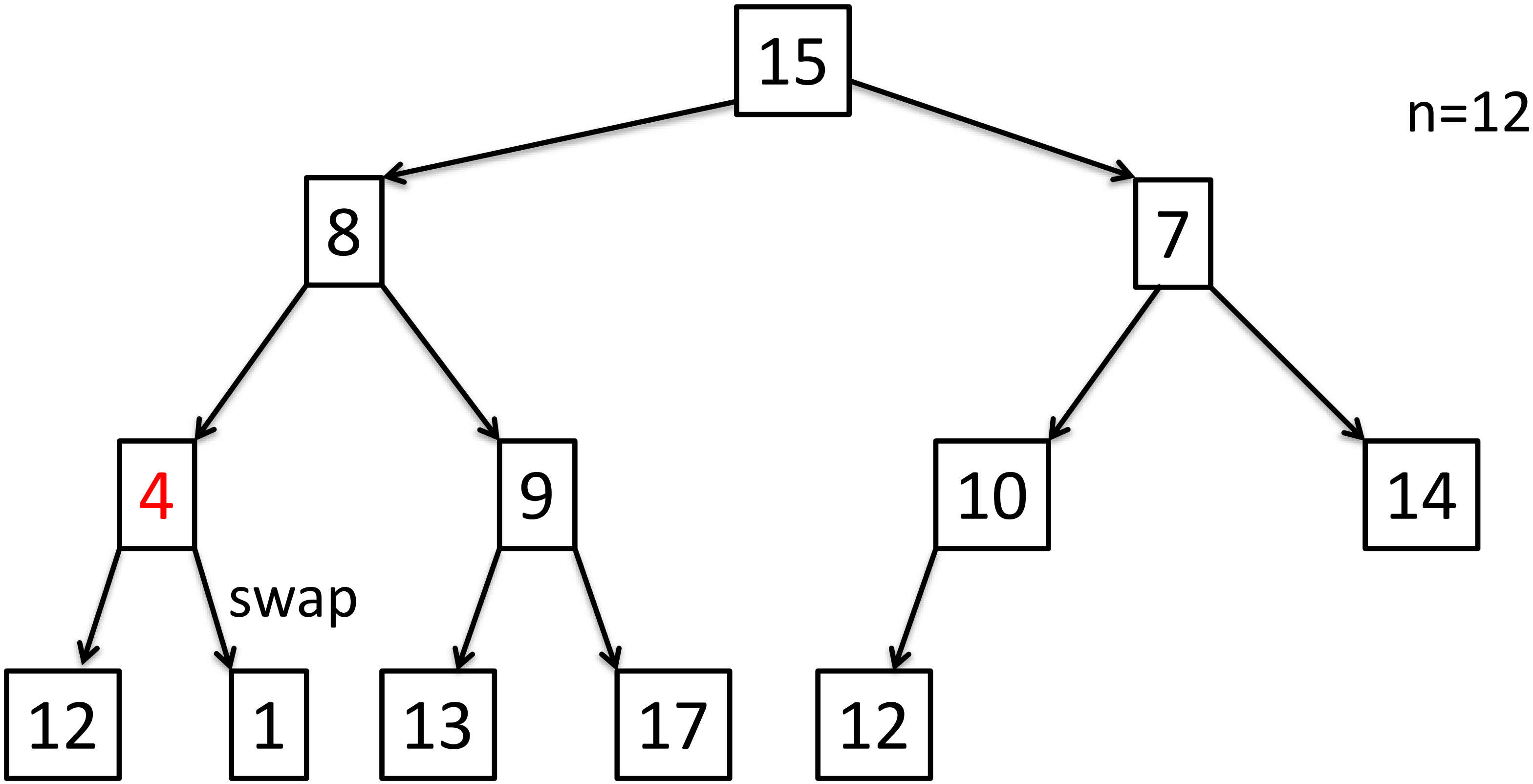
Example – buildHeapBackwards



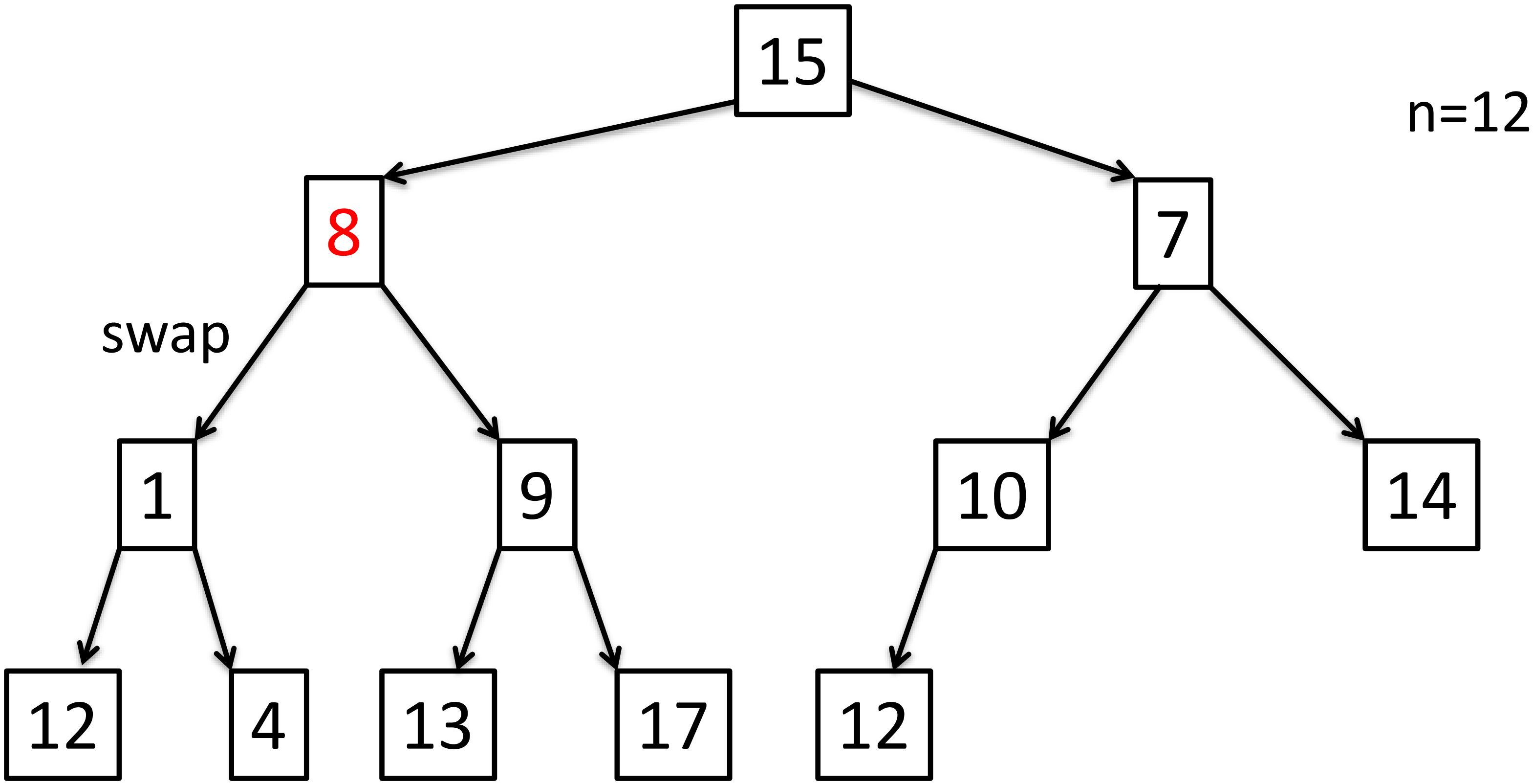
Example – buildHeapBackwards



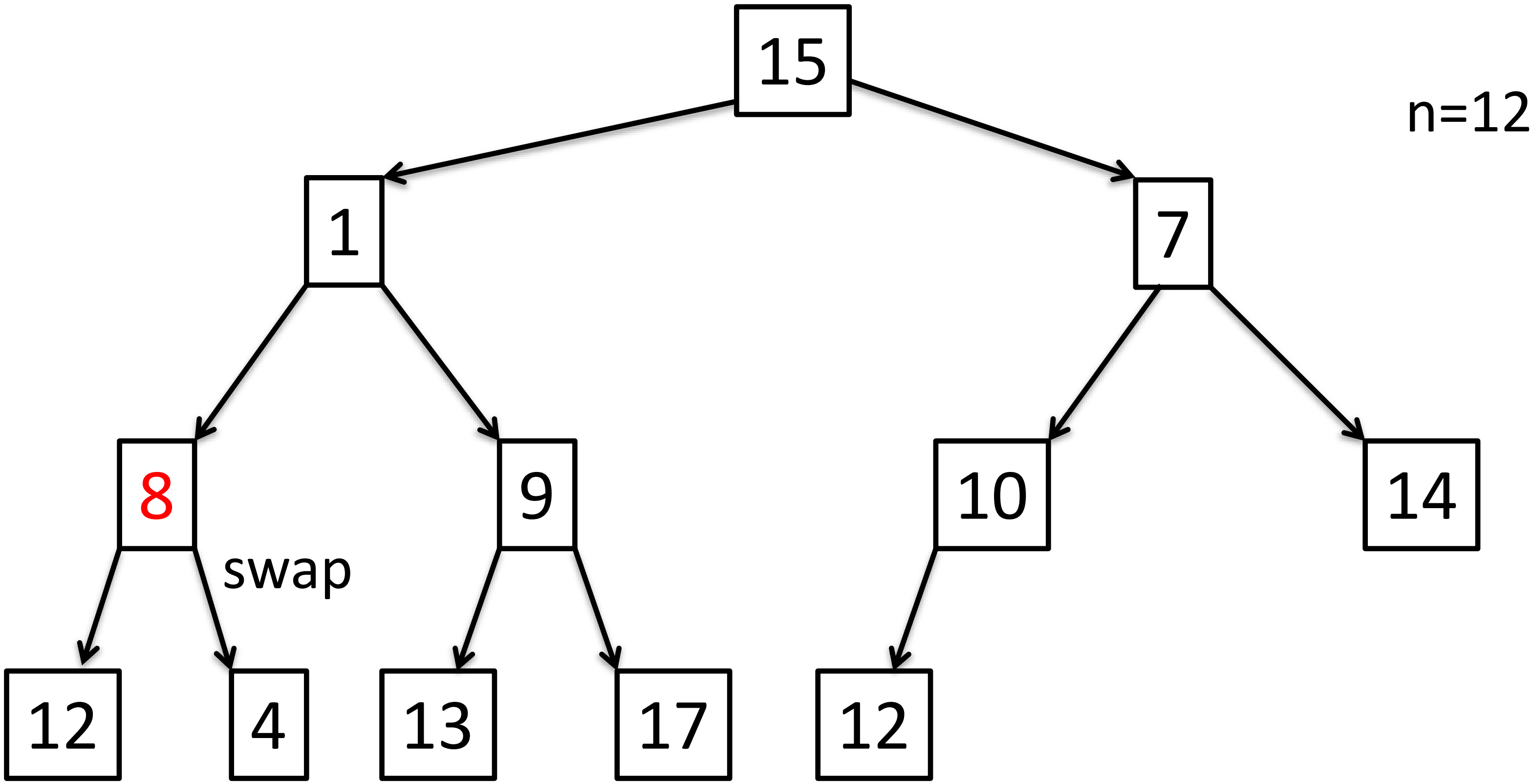
Example – buildHeapBackwards



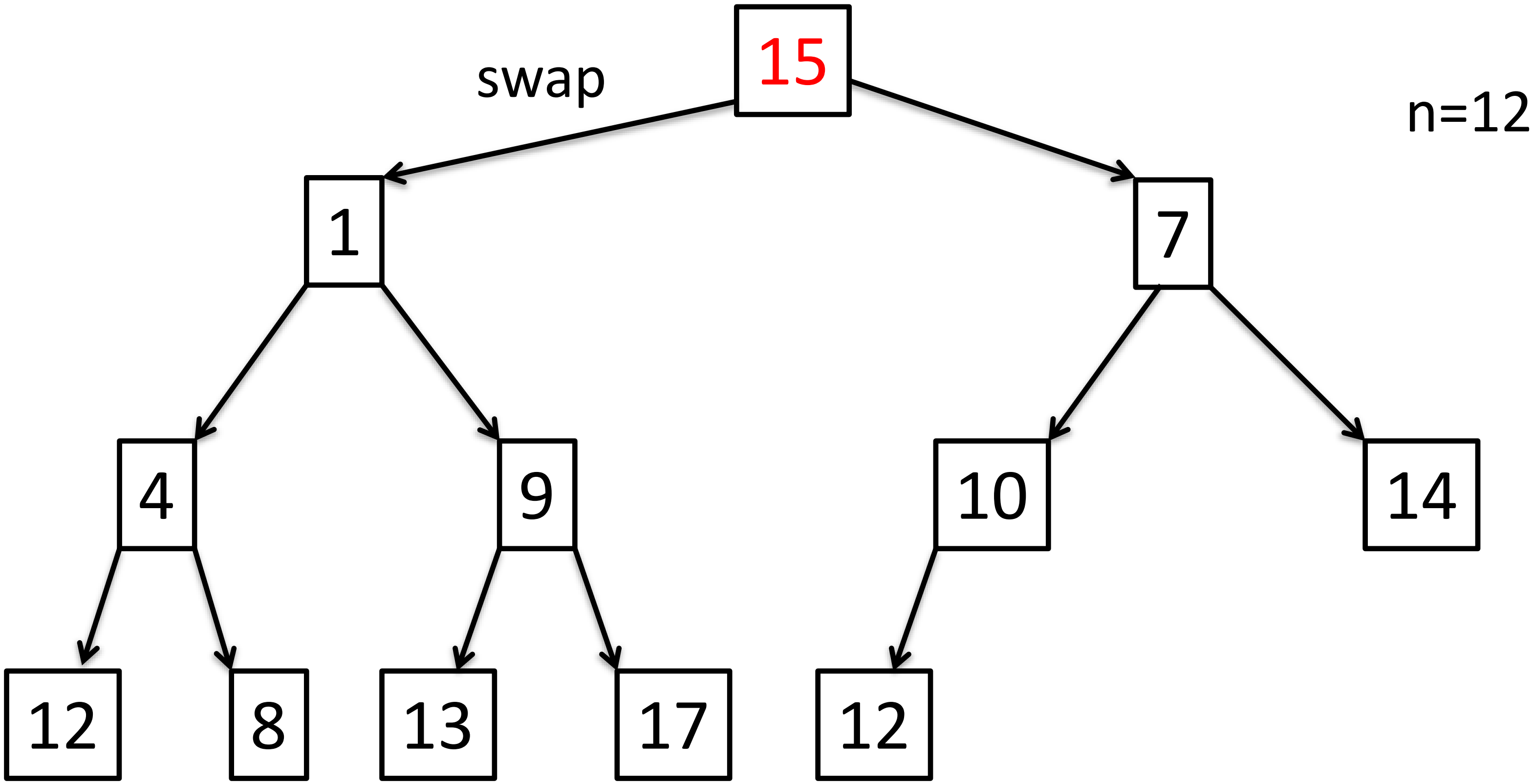
Example – buildHeapBackwards



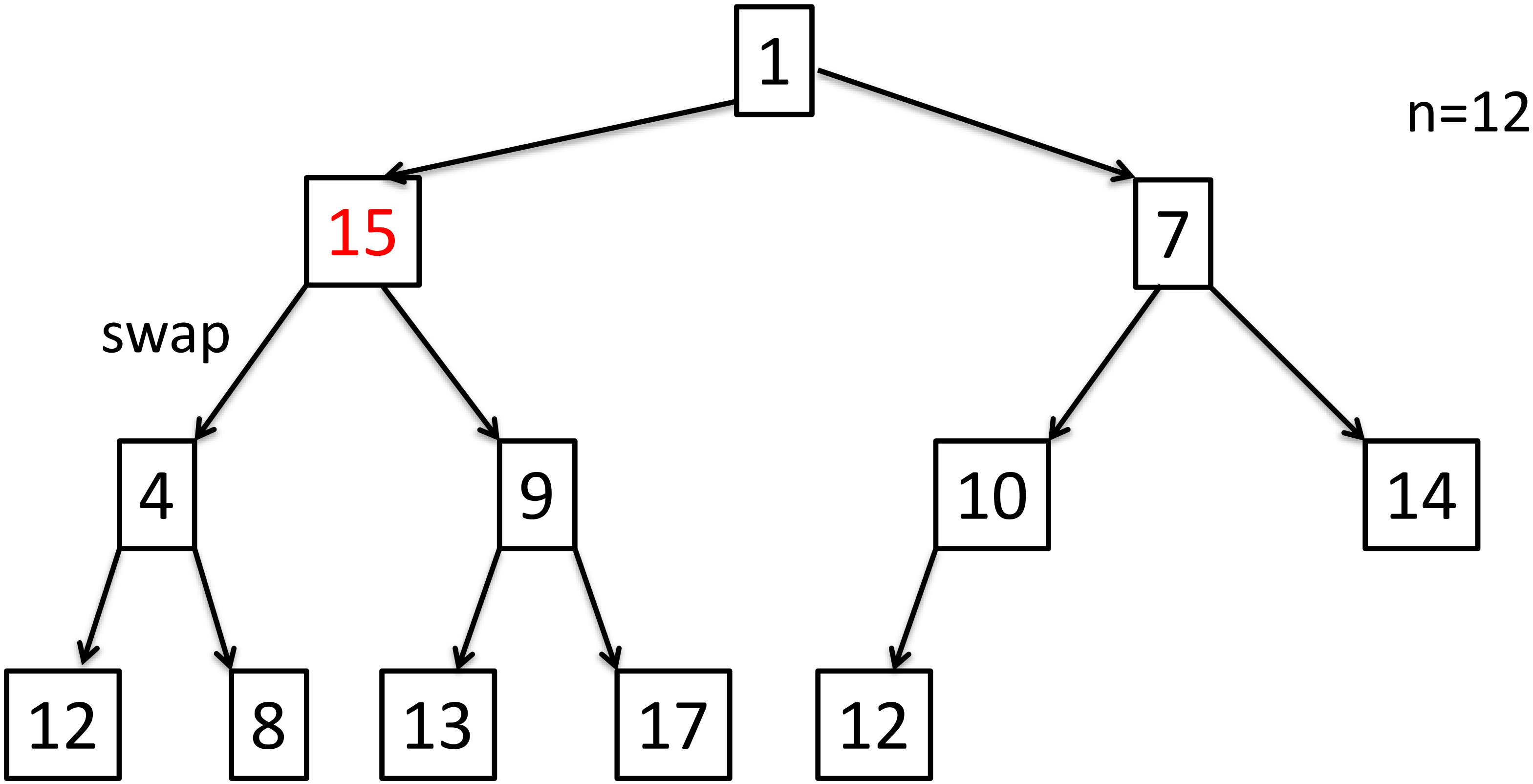
Example – buildHeapBackwards



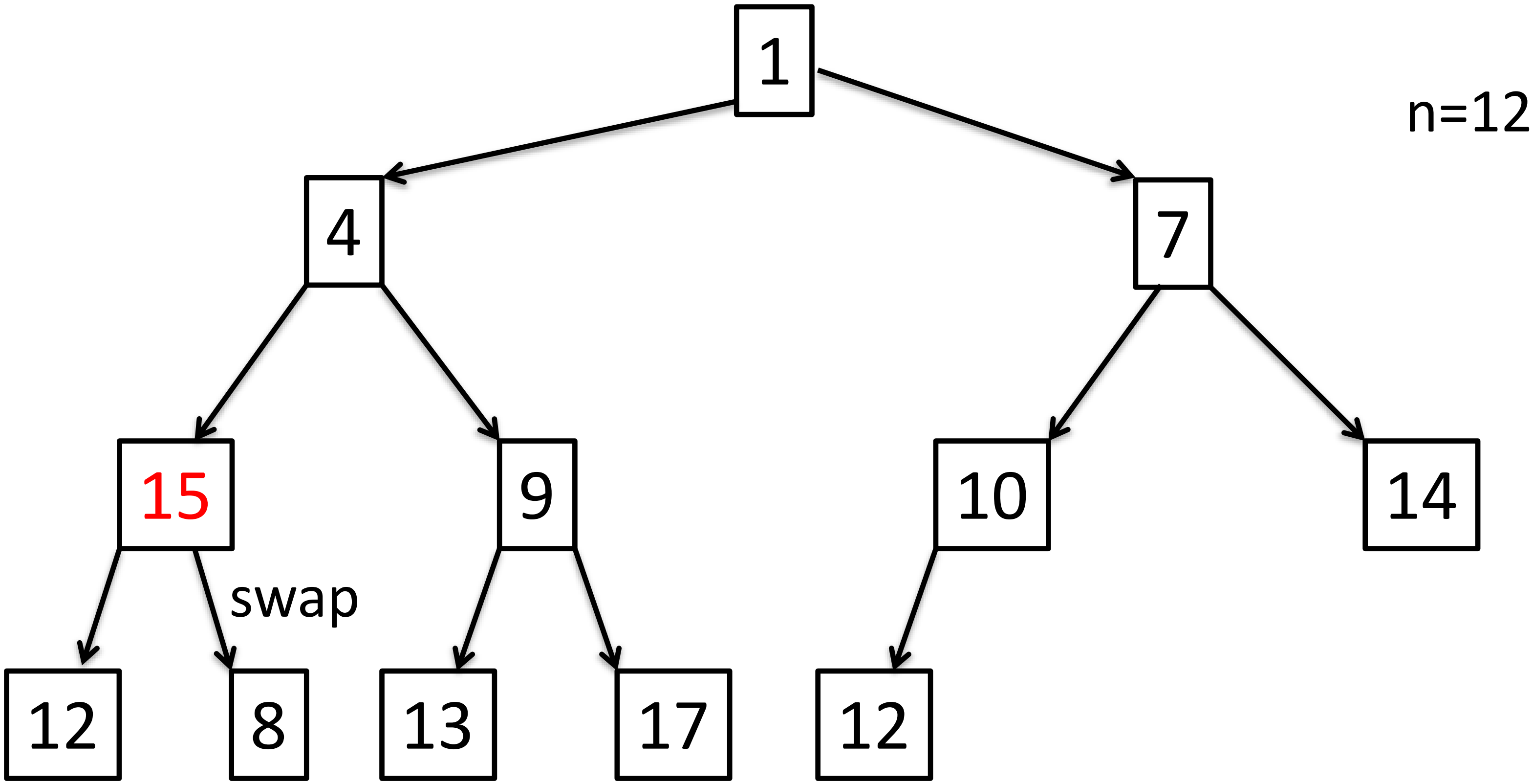
Example – buildHeapBackwards



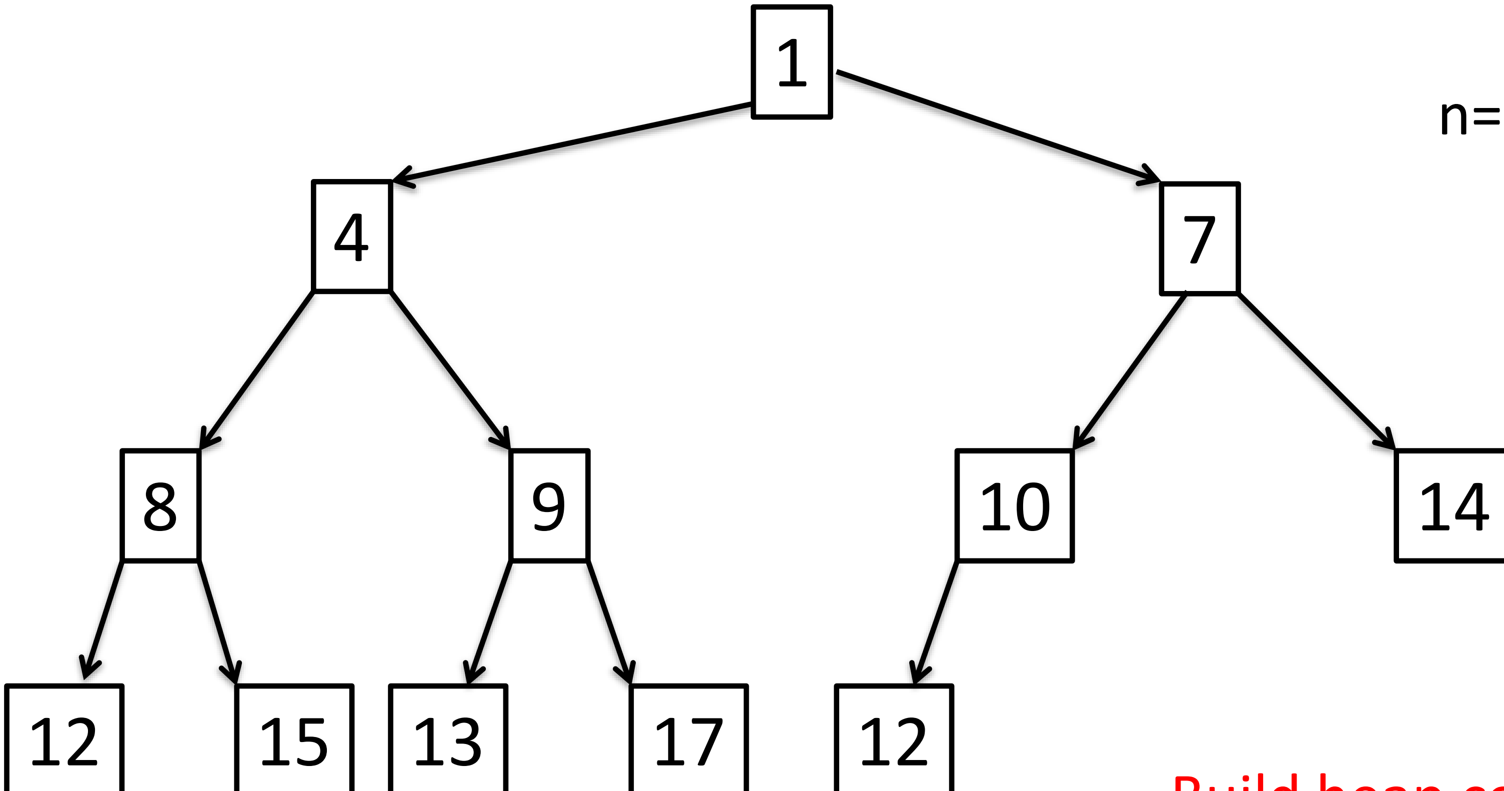
Example – buildHeapBackwards



Example – buildHeapBackwards



Example – buildHeapBackwards



n=12

Build heap completed

Theorem

- BuildHeapBackwards establishes a heap.

- Proof sketch:
  - Before calling siftDown(i) all nodes with indices 2i+2, …, n fulfill the heap property.
  - Show that after siftDown(i) all nodes with indices 2i, …, n fulfill the heap property.
  - i=1 implies that the array is heap-ordered.

Correctness of buildHeapBackwards

- Consider subtree rooted at i and let e be the element.
- Build can only affect the heap property of the elements in that subtree.
- Heap property in that subtree can only be violated at the children of i.
- SiftDown swaps e with the smallest of its children (implies heap property holds at the other child after the swap)
- Element e is moved down along the path until heap property is not violated at the children anymore.
- Heap property holds at the children of i and in their subtrees.
- Heap property holds at indices $2i, \ldots, n$.

Theorem

- BuildHeapBackwards establishes a heap in time O(n).

- Proof sketch:
  - There are at most 2^{ℓ} nodes of depth ℓ.
  - A call of siftDown for each of these nodes takes time $O(k-\ell)$ for depth k tree.
  - Get total runtime by summing up $\ell = 0, \ldots, k-1$

Proof runtime

Total runtime

$$O\left(\sum_{l=0}^{k-1} 2^l \cdot (k-l)\right)$$

$$= O\left(2^k \sum_{l=0}^{k-1} 2^{-k+l} \cdot (k-l)\right)$$

$$= O\left(2^k \sum_{j=1}^{k} 2^{-j} \cdot j\right)$$

$$= O(n) \qquad \square$$

Explanation

$$2^{\lfloor \log n \rfloor} \leq n \text{ and } \sum_{j=1}^{k} 2^{-j} \cdot j < 2$$

Runtime

- Creation of empty heap O(1)
- Finding the minimum element O(1)
- DeleteMin O(log n)
- Insert O(log n)
- Building Heap O(n)

Heap sort

- Want to have a sorting algorithm based on heaps that runs in time O(n log n).

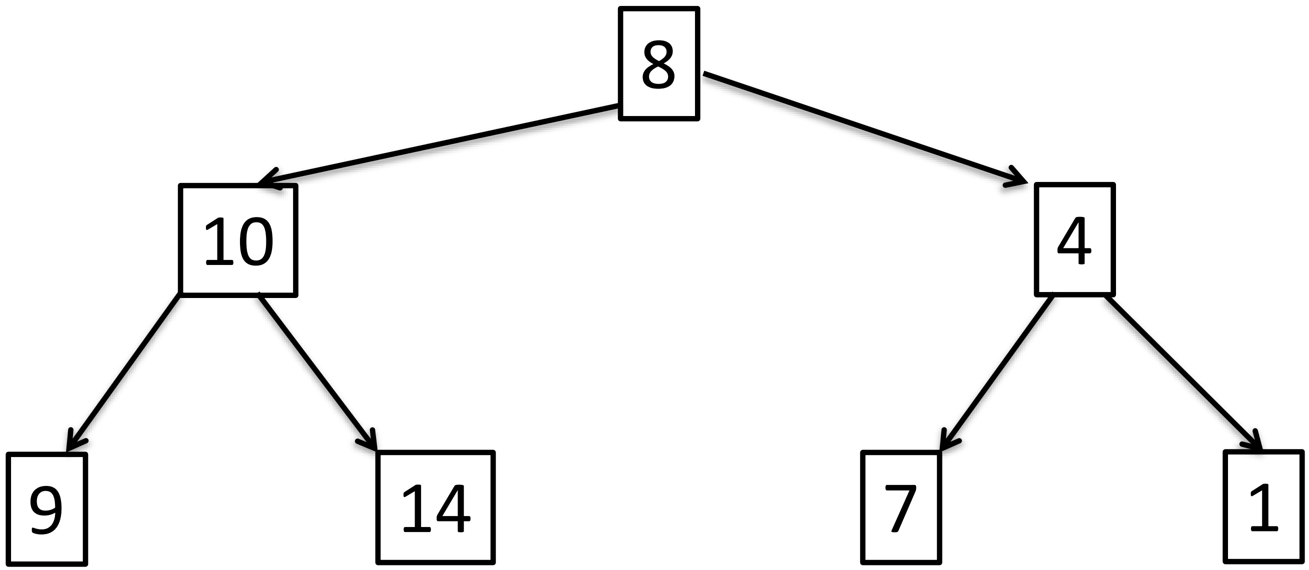
- Idea:
  - Build the heap for *n* elements in time O(n).
  - Each step pick and delete the minimum element, time O(log n)
  - Iterate until heap is empty.
- In total *n* iterations implies total runtime O(n log n)

Example – heap sort

Sort the sequence 8,10,4,9,14,7,1

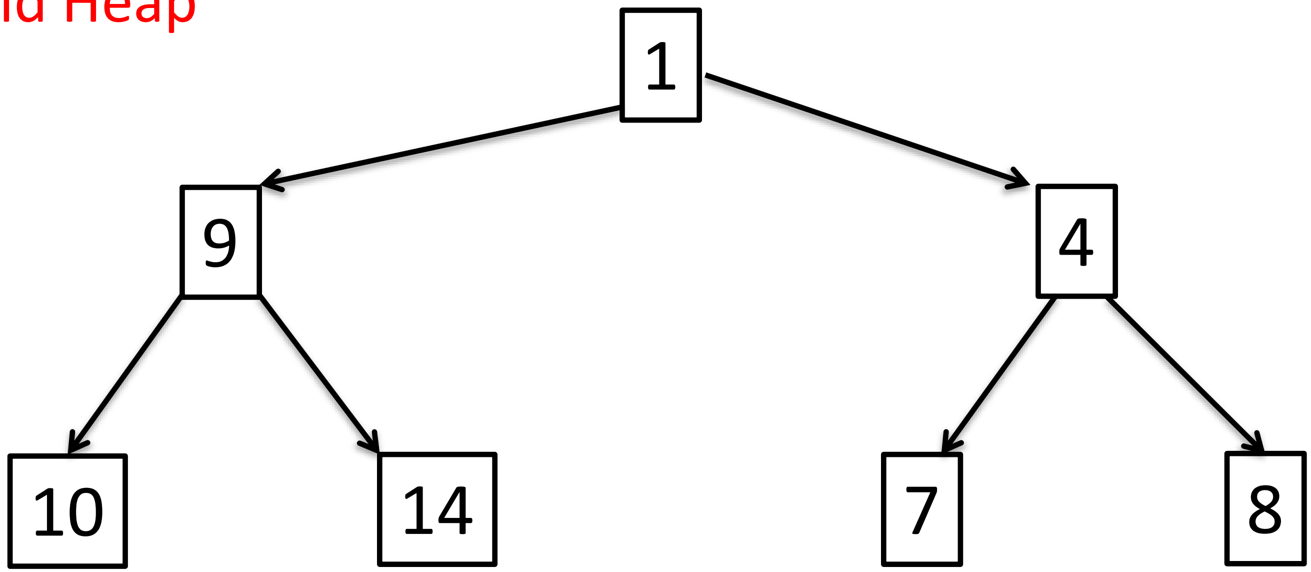
Input array:

8	10	4	9	14	7	1



Example – heap sort

Build Heap



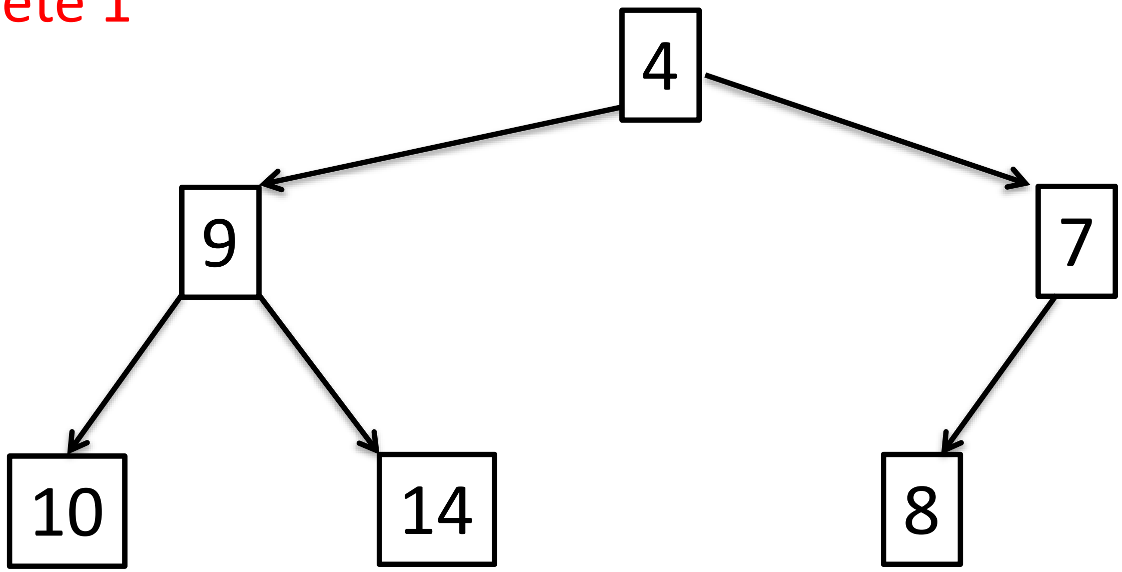
Heap array h

1	9	4	10	14	7	8

Sorted array s

Example – heap sort

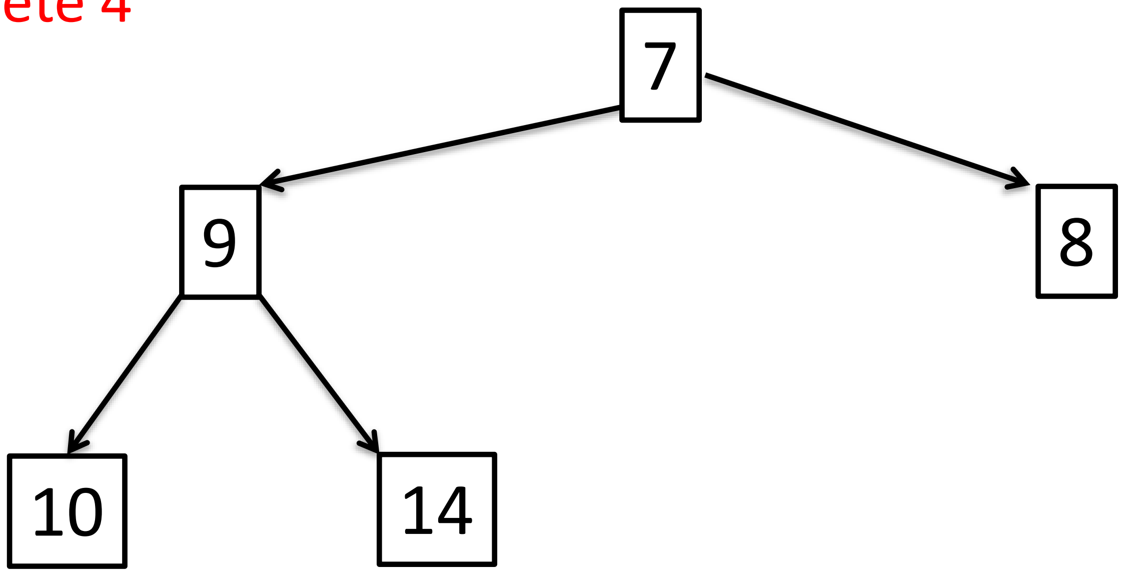
Delete 1



Heap array h	4	9	7	10	14	8	
Sorted array s	1						

Example – heap sort

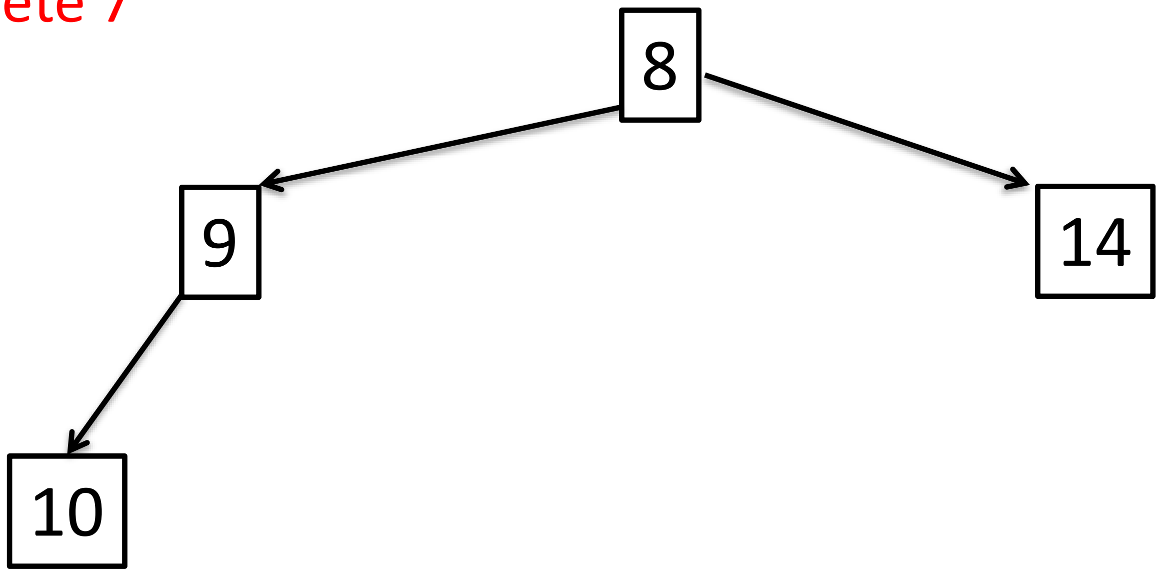
Delete 4



Heap array h	7	9	8	10	14		
Sorted array s	1	4					

Example – heap sort

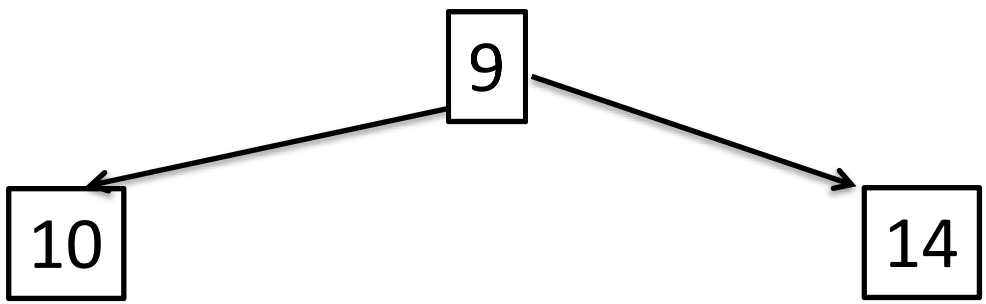
Delete 7



Heap array h	8	9	14	10			
Sorted array s	1	4	7				

Example – heap sort

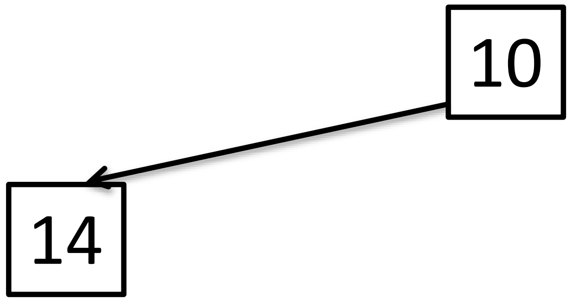
Delete 8



Heap array h	9	10	14				
Sorted array s	1	4	7	8			

Example – heap sort

Delete 9



Heap array h							
	10	14					

Sorted array s							
	1	4	7	8	9		

Example – heap sort

Delete 10

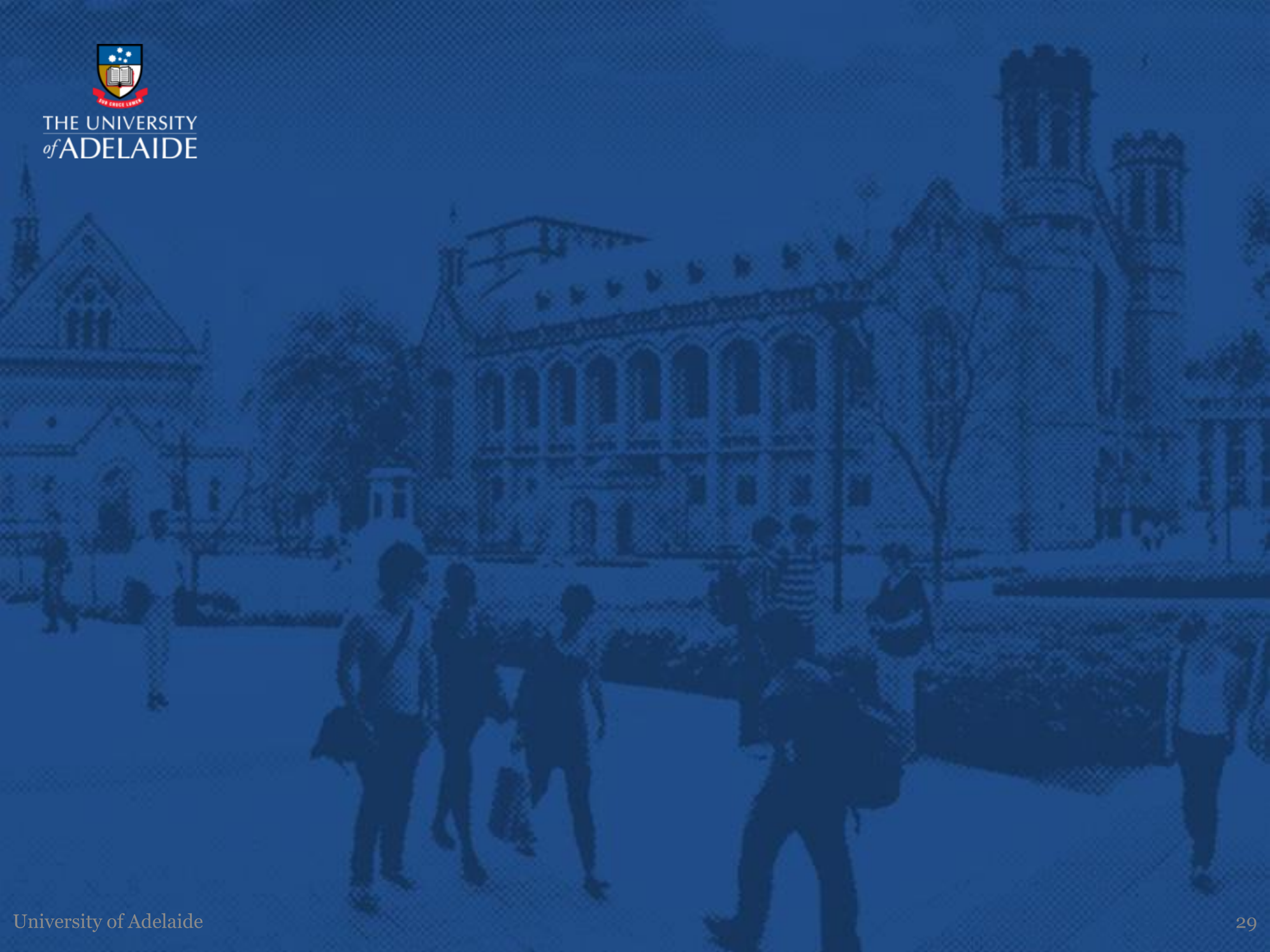
14

Heap array h	14						
Sorted array s	1	4	7	8	9	10	

Example – heap sort

Delete 14

Heap array h							
Sorted array s	1	4	7	8	9	10	14



THE UNIVERSITY of ADELAIDE