CS5800 – ALGORITHMS

MODULE 5. DATA STRUCTURES & GRAPHS

Lesson 3: Priority Queues & Heaps

Ravi Sundaram

Topics

- Priority Queue – what & why?
- Naïve implementation & $\Omega(n\log n)$ Lower Bound
- Binary Heap
  - Array implementation
  - Insertion – add as last element and percolate up
  - Deletion – replace with last element and percolate appropriately
- Building heap in O(n) time
- Summary

Priority Queues

- A Priority Queue is a queue where elements have a priority, or key, so that elements with higher priority (a smaller key) are at the front of the queue.
- Priority Queue is an abstract data structure that supports the following operations:
  - Insert(x) – inserts element x, assume key is same as x.
  - FindMin() – returns (pointer to) element with minimum key
  - Delete(x) – removes element x, assume pointer is same as x
  - DeleteMin() – removes element with minimum key (can be effected by Delete(FindMin()) so we ignore for rest of lecture
  - ChangeKey(x,k) – changes key of element x to k (can be effected by a delete followed by insert with changed key, so we ignore for rest of lecture)

- Applications.
  - Dijkstra's shortest path algorithm.
  - Prim's MST algorithm.
  - Event-driven simulation.
  - Huffman encoding.
  - Heapsort.
  - . . .

Naïve implementation

- Can implement as sorted block in array
- Can implement as unsorted linked list

	Array	Linked List
Insert	$\Theta(n)$	$\Theta(1)$
FindMin	$\Theta(1)$	$\Theta(n)$
Delete	$\Theta(n)$	$\Theta(1)$

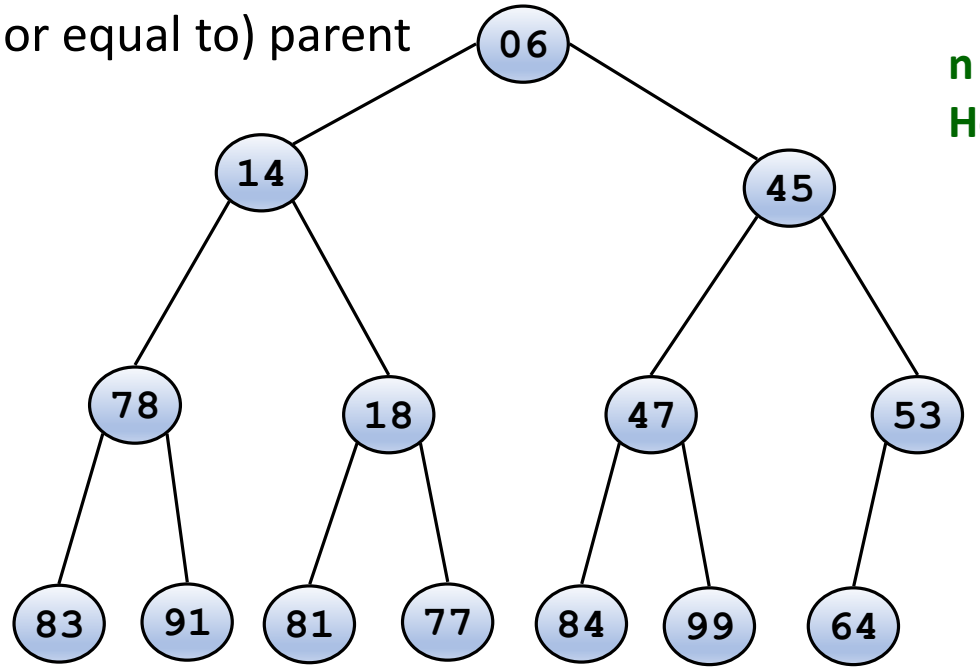
Lower bound

- Can we do better?
- Can we implement each operation to be O(1)?
- Theorem: At least one of the three operations must be $\Omega(\log n)$
- Pf: Observe that we can sort n elements by inserting them into a heap and repeatedly removing the minimum. This takes n of each of the 3 operations. And we know that sorting has an $\Omega(n \log n)$ lower bound and hence at least one of these 3 operations must take at least $\Omega(\log n)$ ■

	Array	Linked List	Binary Heap
Insert	$\Theta(n)$	$\Theta(1)$	O(logn)
FindMin	$\Theta(1)$	$\Theta(n)$	O(logn)
Delete	$\Theta(n)$	$\Theta(1)$	O(logn)

Binary Heap: Definition & Properties

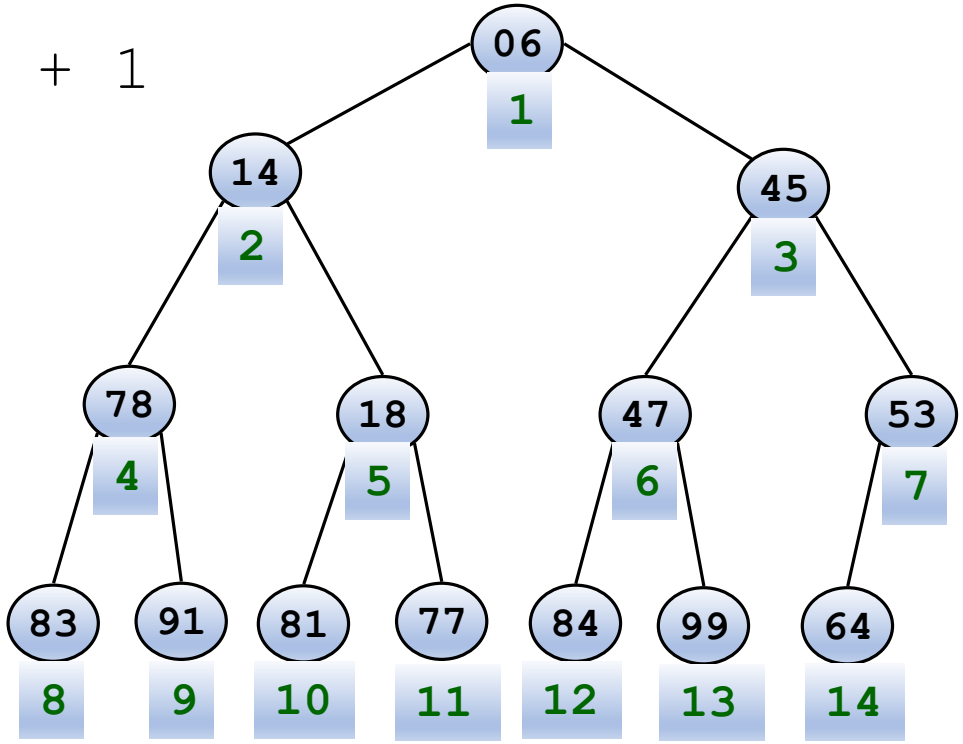
- Binary heap – two properties – structure and order
  - Almost complete binary tree.
    - filled on all levels, except last, where filled from left to right
  - Min-heap ordered.
    - every child greater than (or equal to) parent
- Properties.
  - Min element is in root.
  - Heap with n elements has height = $\lfloor \log_2 n \rfloor$.



n = 14
Height = 3

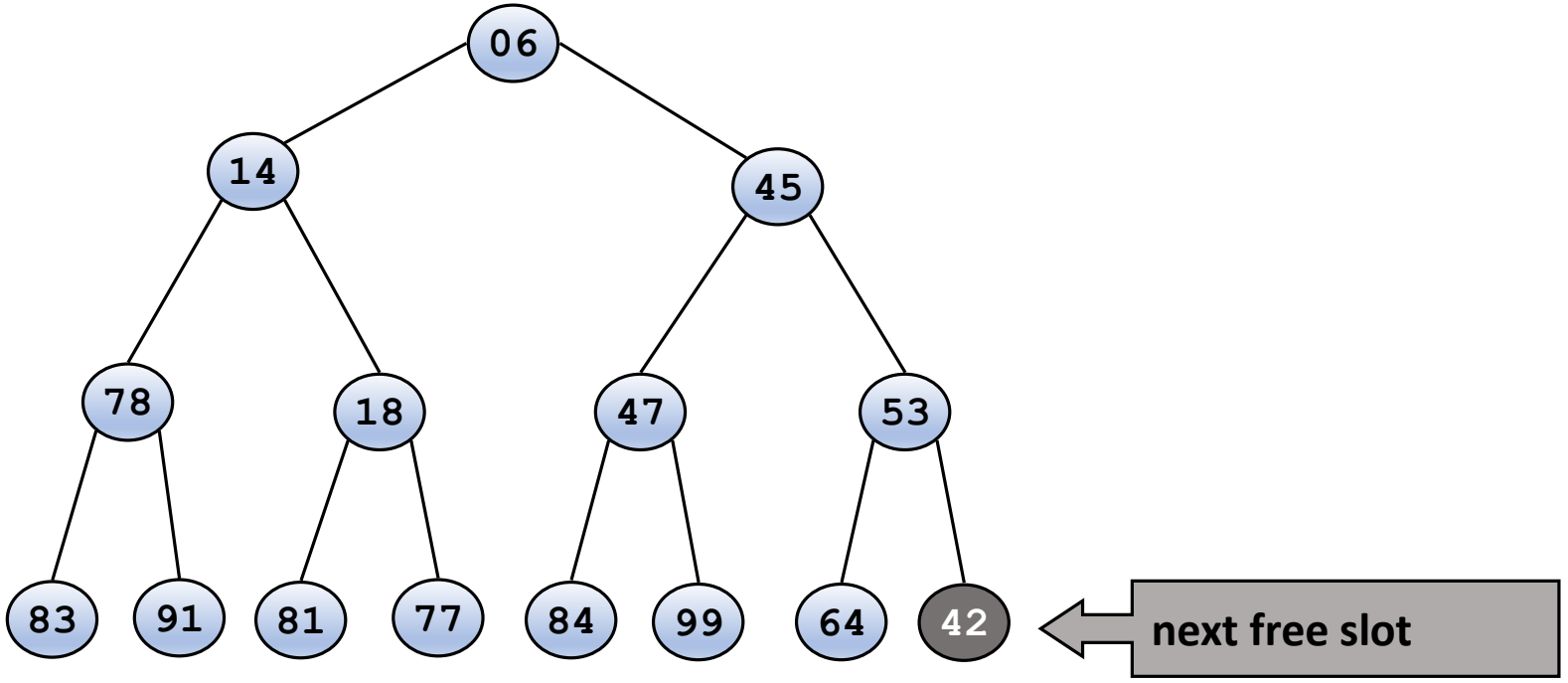
Binary Heaps: Array Implementation

- Use an array: no need for explicit parent or child pointers.
  - `Parent(i) = ⌊i/2⌋`
  - `Left(i) = 2i`
  - `Right(i) = 2i + 1`



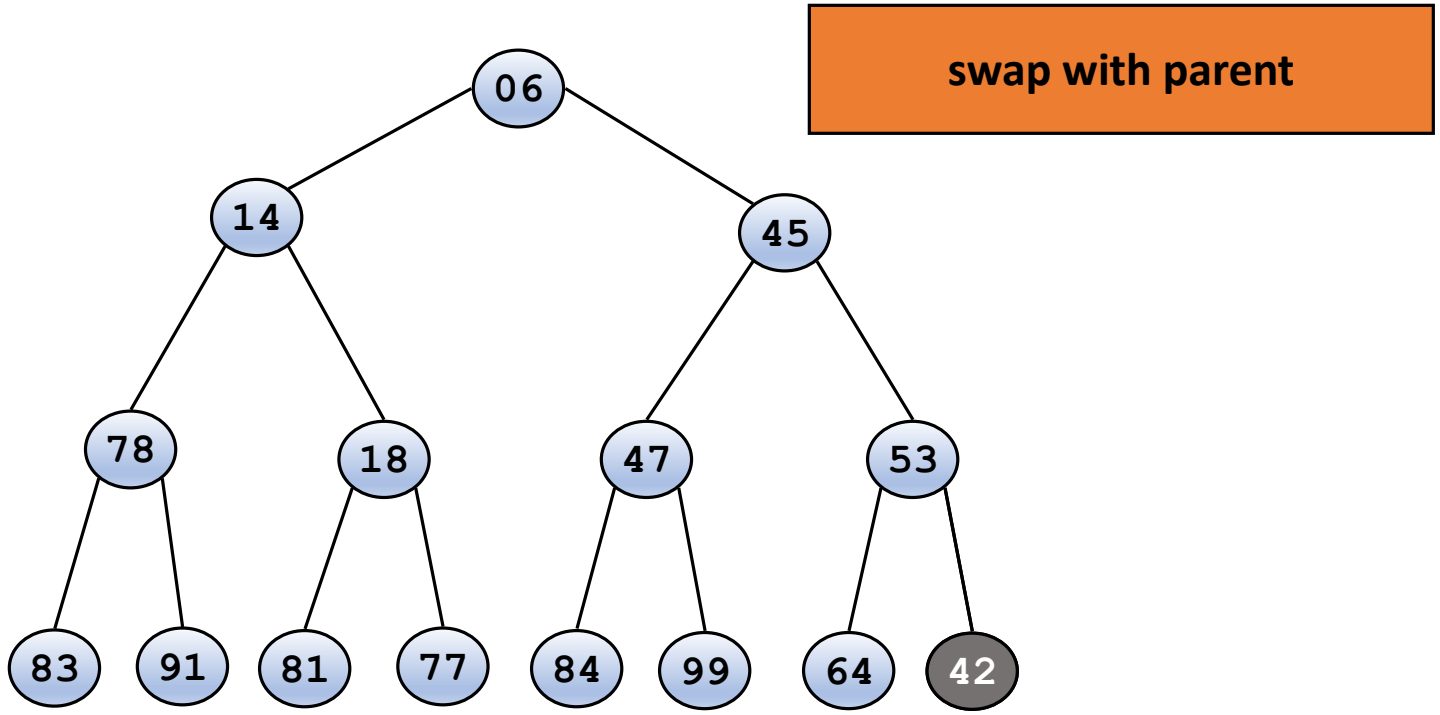
Binary Heap: Insertion

- Insert element x into heap.
  - Insert into next available slot.
  - Percolate up until it is heap ordered.



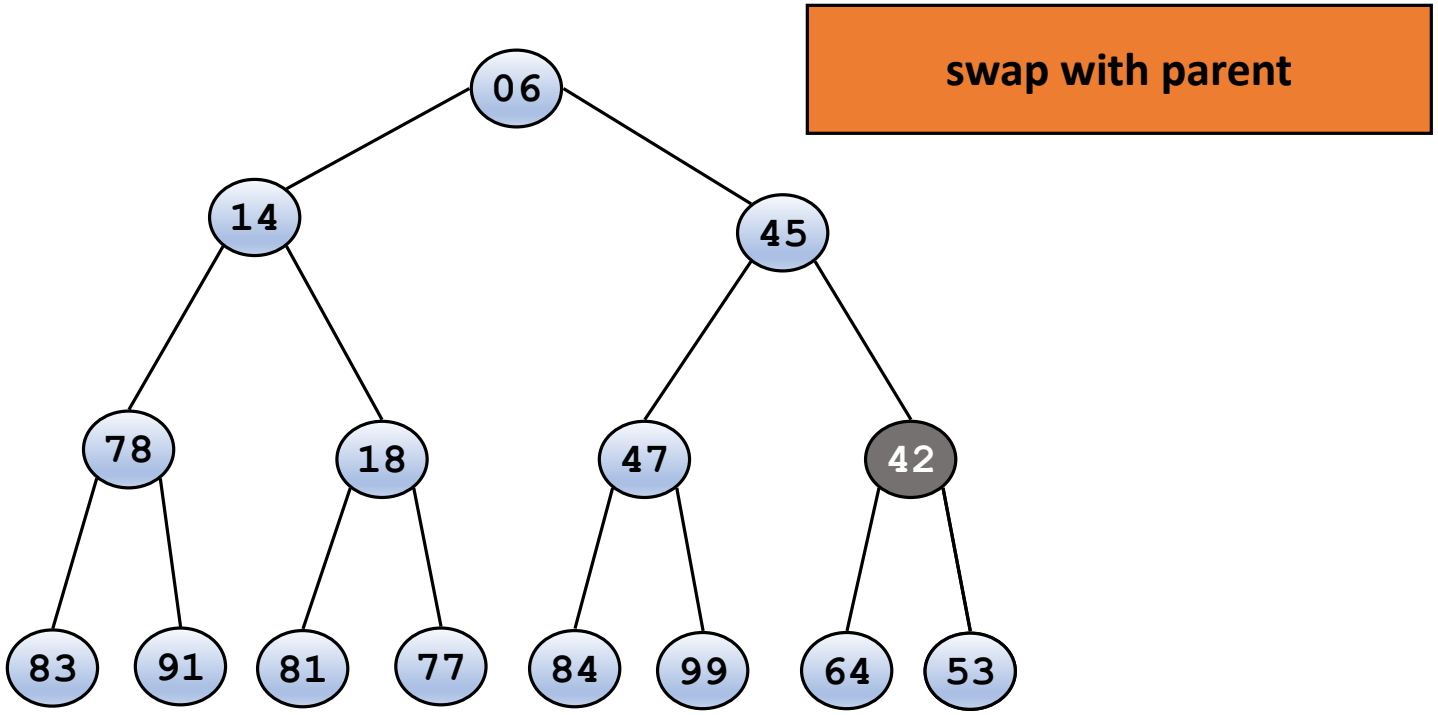
Binary Heap: Insertion

- Insert element x into heap.
  - Insert into next available slot.
  - Percolate up until it is heap ordered.



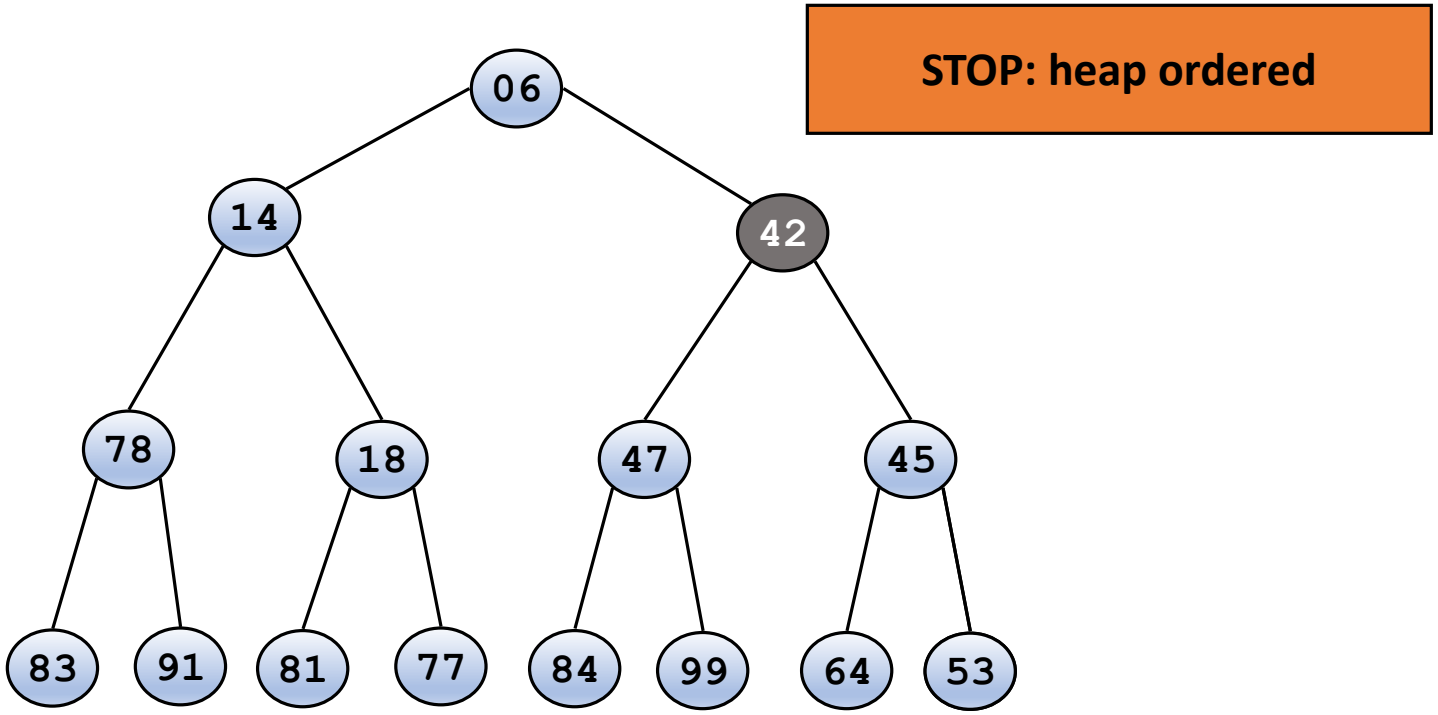
Binary Heap: Insertion

- Insert element x into heap.
  - Insert into next available slot.
  - Percolate up until it is heap ordered.



Binary Heap: Insertion

- Insert element x into heap.
  - Insert into next available slot.
  - Percolate up until it is heap ordered.



Binary Heap: Delete

- DeleteMin: delete minimum element from heap.
  - Exchange root with rightmost leaf.
  - Percolate down, exchanging with smaller child, until it is heap ordered.
  - O(logn) operations

- Delete: remove element from heap.
  - Replace with rightmost leaf
  - Percolate up or down as appropriate
  - O(logn) operations

Binary Heap: BuildHeap

- BuildHeap: given n elements build heap from scratch
  - Naïve implementation
  - Repeatedly call Insert n times
  - $\Omega(\text{nlogn})$ operations

- BuildHeap: clever implementation
  - Place given elements down in array arbitrarily with no heap ordering
  - Starting from bottom (not top!) repeatedly percolate down
  - Key observation – more elements at bottom so less movement
  - #operations = $\sum_{logn}^{0} 2^i * (logn - i)$ = $\sum_{0}^{logn} (\frac{n}{2^i}) * i$ = O(n)

Binary Heap: Heapsort

- Heapsort.
  - Insert N items into binary heap.
  - Perform N delete-min operations.
  - O(N log N) sort.
  - No extra storage.

Summary

- Priority Queue – useful data structure for greedy algorithms
- Implementable using array
- Insert, Delete, Extract-Min in O(logn) time
- Entire heap – O(n) time
- Main Takeaway: the binary heap implemented on the array is an easy realization of a priority queue, useful in many greedy contexts.