

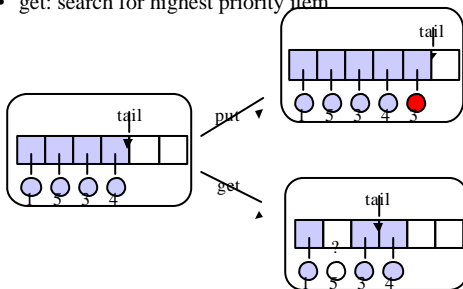
Priority Queues & Heaps

Operations

- put: insert item with a priority
- get: remove the item with highest priority, and break ties with FIFO ordering
- For simplicity:
 - we use an int as the object *and* the priority
 - In practice: need <int priority, Object elt> pair

Niave Solution: Prio-Q As Array

- put: add item to end of array
- get: search for highest priority item

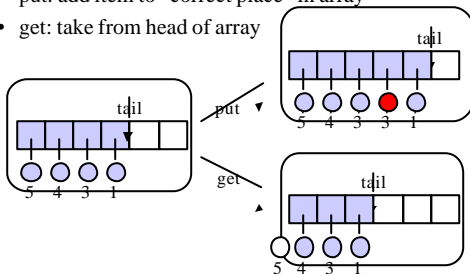


Prio-Q as Array

- put: $O(1)$ assuming no array doubling
- get: $O(n)$
 - $O(n)$ to find element, $O(1)$ to patch hole

Prio-Q as Sorted Array

- Maintain array in sorted order, largest first
- put: add item to “correct place” in array
- get: take from head of array

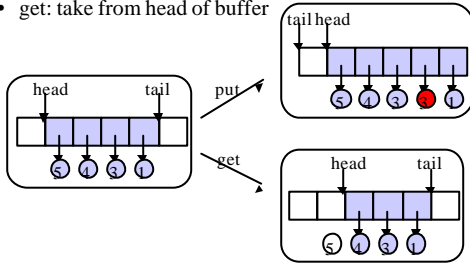


Prio-Q as Sorted Array

- put: $O(n)$
 - $O(n)$ to find correct position + $O(n)$ to shift remaining elements out of the way
 - Can we use binary search?
- get: $O(n)$
 - Need to slide all elements to the left

Prio-Q as Sorted Circular Buffer

- Maintain circular buffer in sorted order, largest first
- put: add item to "correct place" in buffer
- get: take from head of buffer



Prio-Q as Sorted Circular Buffer

- put: $O(n)$
 - $O(n)$ to find correct position + $O(n)$ to shift remaining elements out of the way
 - Can we use binary search?
- get: $O(1)$

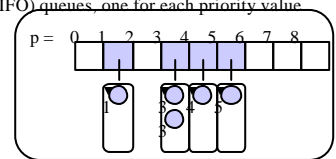
Prio-Q as Sorted Linked List

- put: $O(n)$
 - $O(n)$ to find item (linear search) + $O(1)$ to splice it out
- get: $O(1)$

Special Case: at most p priorities

- Many applications only use a fixed number of priorities:
 - 8-Puzzle with # tiles correct as heuristic: p in 0..8
 - Java task priorities: usually 1..10
 - Email priority: very low, low, normal, high, urgent
- Cool implementation:
 - Use array of p (FIFO) queues, one for each priority value

put: $O(1)$
 get: $O(P)$
 $P = \#$ priorities

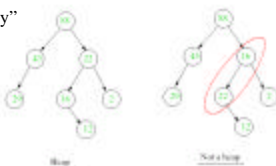


General Solution: Heap

A *heap* is a tree in which

1. an integer is stored at each node
2. integer stored at node is \geq integer stored at any of its children

"the heap property"



For now, we only care about *binary* trees

Heaps in Reality

- Ages of people in a family tree
 - Parent older than children
 - But your uncle may be younger than you
- Salaries
 - boss makes more than underlings
 - but temp worker in one division can make more than a supervisor in a different division

Running Example: Crime Family

- Nodes contain name and “ruthlessness” (= # crimes committed, an integer)
- Boss is always more ruthless than a subordinate



Heap of Priority Queue Elements

Prio-Q as Heap

- get: must return element with largest priority
- put: insert somewhere in the heap and maintain/restore the heap property
 - “heapify” = restore the heap property

get

- Highest priority always at root
- Remove root, then fill hole

Solution #1:

- promote the highest-priority child
- recurse
- until you promote a leaf element
- delete the now-empty leaf

Heapify #1: Restore the heap property

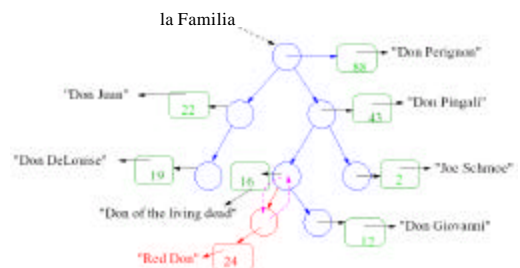


Heapifying after removing root element

put

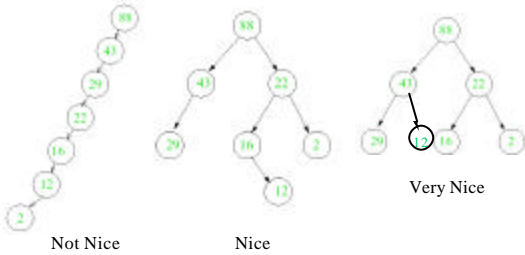
- Insert new element into a new leaf node, anywhere in the tree
- Then “bubble up” the new element until heap property is restored
 - compare new element against parent
 - if needed, swap then repeat in the new position
 - worst case: bubble up all the way to the root

Bubble up



Prio-Q as Heap

- put: $O(n)$ worst case, but $O(\log n)$ if tree is nice
- get: $O(n)$ worst case, but $O(\log n)$ if tree is nice



Keeping the Tree Fat & Short

A *complete binary tree* is a tree in which:

1. Nodes are numbered in a bfs fashion (the "position")
2. Node at position n is filled only if all nodes in positions less than n are filled

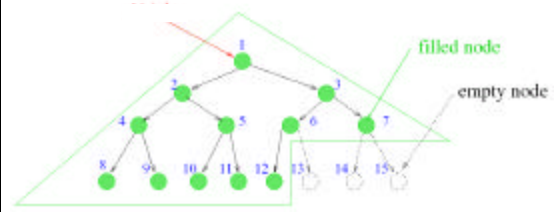


Heap As Complete Binary Tree

- Will keep tree short and fat, and us happy
- Get: will ruin the complete binary tree property, since the bottom-right-most node might not be the one to get promoted
- Put: need to create new node in the first empty position
 - easy to name (next empty position = current size + 1)
 - how to find the position once we have the number?

Quick Diversion: Binary Numbers

- Finding empty position 13 given root (position 1)
- $13 = 8 + 4 + 0 + 1 = 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 1101$ (binary)



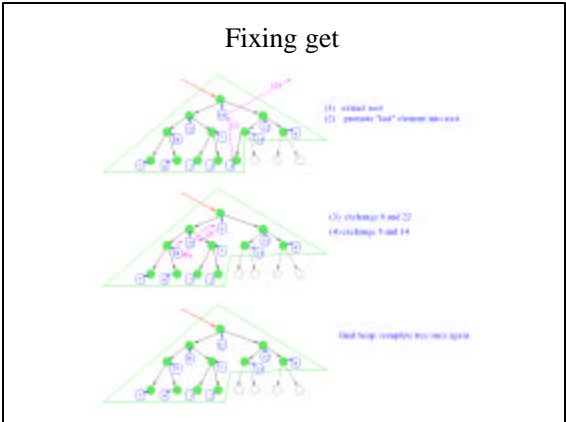
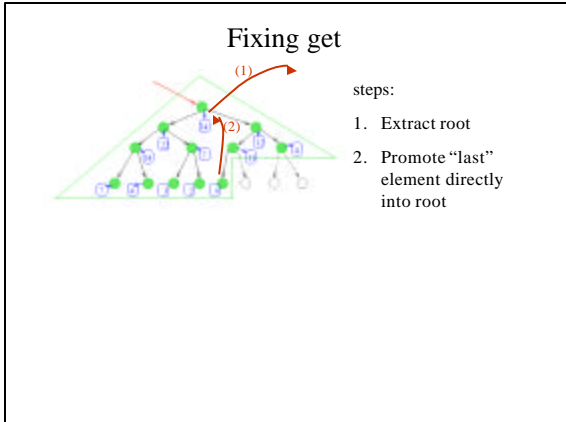
So we can find any position in $\log n$ steps

put

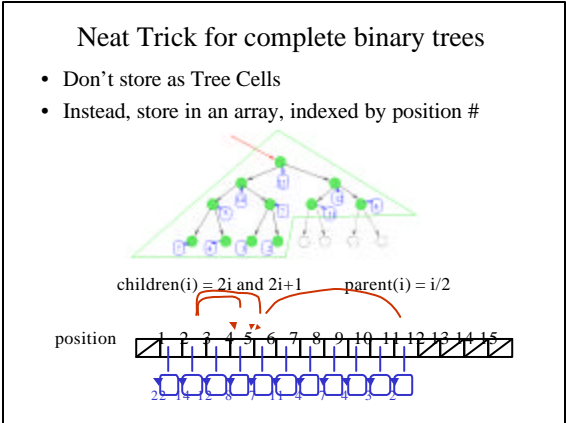
- If current size = n , insert at position $n' = n + 1$
 - Convert n' to binary
 - substitute R for 1, and L for 0
 - follow links to the position
- Insert new element
- Bubble up, as before

get is still broken





- ### Prio-Q as Heap
- Use a heap that is a complete binary tree
 - Keep track of size
 - get: $O(\log n)$
 - remove root, promote last leaf node, bubble down
 - put: $O(\log n)$
 - insert into "last+1" leaf node, bubble up



- ### Heap Construction
- build-heap: Given n elements in an array, rearrange to build a heap
 - Obvious solution:
 - Start with empty array
 - Do heap insert for each element
 - $1 + n * \log n \rightarrow O(n \log n)$
 - There is a faster way...