

Unit Outline

Unit #2: Priority Queues CPSC 221: Algorithms and Data Structures

Will Evans and Jan Manuch

2016W1

- ▶ Rooted Trees, Briefly
- ▶ Priority Queue ADT
- ▶ Heaps
 - ▶ Implementing Priority Queue ADT
 - ▶ Focus on Create: Heapify
 - ▶ Brief introduction to *d*-Heaps

2 / 32

Learning Goals

- ▶ Provide examples of appropriate applications for priority queues and heaps
- ▶ Manipulate data in heaps
- ▶ Describe and apply the Heapify algorithm, and analyze its complexity

Rooted Trees



- ▶ Family Trees
- ▶ Organization Charts
- ▶ Classification trees (a.k.a. keys)
 - ▶ What kind of flower is this?
 - ▶ Is this mushroom poisonous?
- ▶ File directory structure
 - ▶ folders, subfolders in Windows
 - ▶ directories, subdirectories in UNIX
- ▶ Non-recursive call graphs

3 / 32

4 / 32

Tree Terminology

root:

leaf:

child:

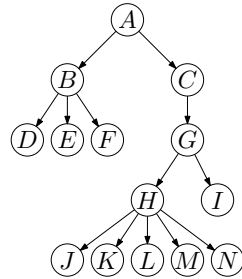
parent:

sibling:

ancestor:

descendent:

subtree:



5 / 32

Tree Terminology Reference

root: the single node with no parent

leaf: a node with no children

child: a node pointed to by me

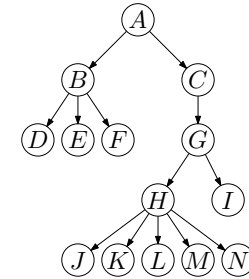
parent: the node that points to me

sibling: another child of my parent

ancestor: my parent or my parent's ancestor

descendent: my child or my child's descendent

subtree: a node and its descendants

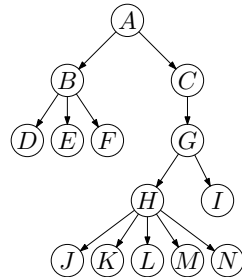


6 / 32

More Tree Terminology

depth: Number of edges on path from root to node

depth of *H*?



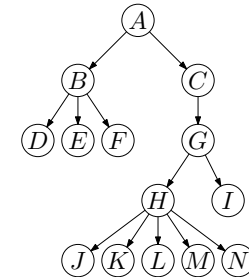
7 / 32

More Tree Terminology

height: Number of edges on longest path from node to descendent or, for whole tree, from root to leaf

height of tree?

height of *G*?

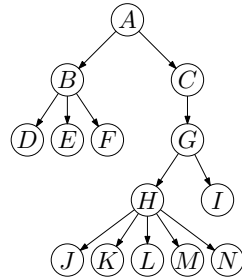


8 / 32

More Tree Terminology

(downward) degree: Number of children of a node

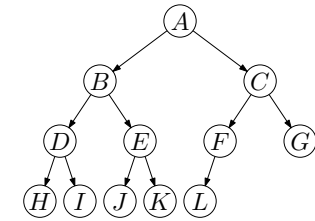
degree of B ?



One More Tree Terminology Slide

binary: each node has degree at most 2

d -ary: degree at most d



complete: as many nodes as possible for its height (each row filled in)

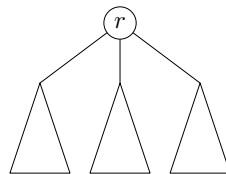
nearly complete: each row except the last one is filled in, all nodes in the last row are as far left as possible

9 / 32

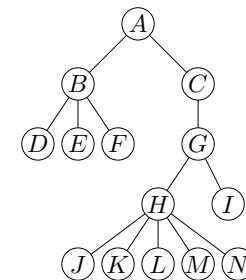
10 / 32

Longest Path

Find the longest *undirected* path in a tree



Longest Path Example



11 / 32

12 / 32

Back to Queues

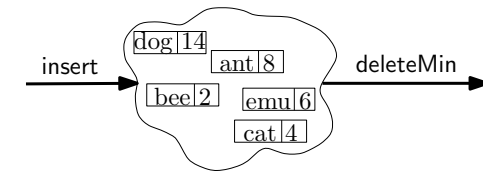
- ▶ Applications
 - ▶ ordering CPU jobs
 - ▶ simulating events
 - ▶ picking the next search site
- ▶ But we don't want FIFO ...
 - ▶ *short* jobs should go first
 - ▶ *earliest* (simulated time) events should go first
 - ▶ *most promising* sites should be searched first

13 / 32

Priority Queue ADT

- ▶ Priority Queue operations

- ▶ create
- ▶ destroy
- ▶ insert
- ▶ deleteMin
- ▶ is_empty



- ▶ Priority Queue property: For two elements in the queue, x and y , if x has a lower priority value than y , x will be deleted before y .

14 / 32

Applications of the Priority Q

- ▶ Hold jobs for a printer in order of length
- ▶ Store packets on network routers in order of urgency
- ▶ Simulate events
- ▶ Select symbols for compression
- ▶ Sort numbers
- ▶ Anything *greedy*: an algorithm that makes the “locally best choice” at each step

15 / 32

Priority Q Data Structures

- ▶ Unsorted list
 - ▶ insert time:
 - ▶ deleteMin time:
- ▶ Sorted list
 - ▶ insert time:
 - ▶ deleteMin time:

16 / 32

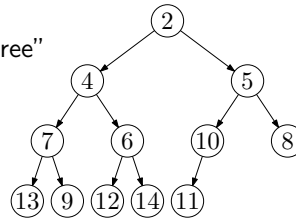
Binary Heap Priority Q Data Structure

Heap-order property: parent's key \leq children's keys.

- ▶ minimum is always at the top

Structure property: "nearly complete tree"

- ▶ depth is always $O(\log n)$
- ▶ next open location always known

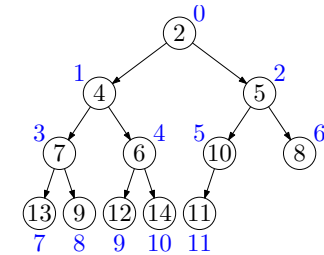


WARNING: This has NO SIMILARITY to the "heap" you hear about when people say "things you create with new go on the heap".

Nifty Storage Trick

Navigation using indices:

- ▶ $\text{left_child}(i) =$
- ▶ $\text{right_child}(i) =$
- ▶ $\text{parent}(i) =$
- ▶ $\text{root} =$
- ▶ $\text{next free position} =$

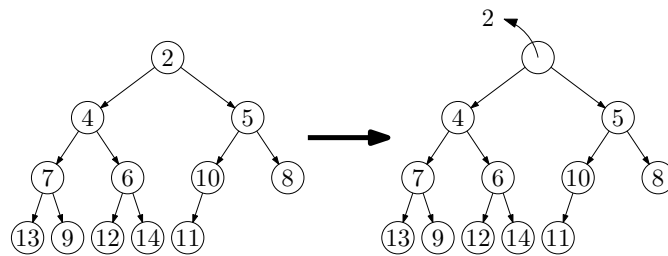


0	1	2	3	4	5	6	7	8	9	10	11	12
2	4	5	7	6	10	8	13	9	12	14	11	

17 / 32

18 / 32

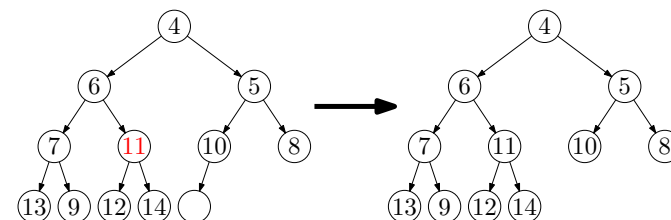
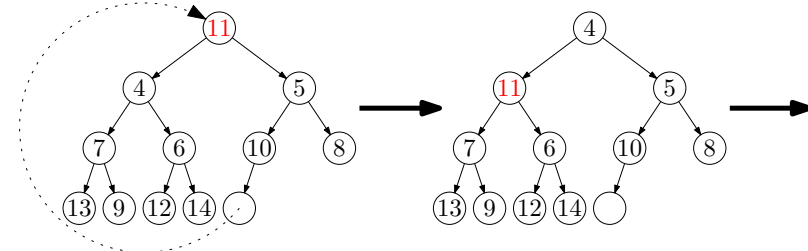
DeleteMin



Invariants violated! No longer "nearly complete"

Swap (Heapify) Down

Move last element to root then swap it down to its proper position.



19 / 32

20 / 32

DeleteMin Code

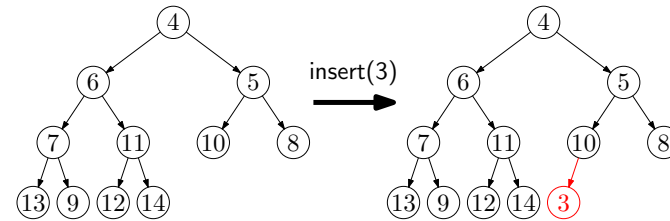
```
int deleteMin() {
    assert(!isEmpty());
    int returnVal = Heap[0];
    Heap[0] = Heap[n-1];
    n--;
    swapDown(0);
    return returnVal;
}
```

Runtime:

```
void swapDown(int i) {
    int s = i;
    int left = i * 2 + 1;
    int right = left + 1;
    if( left < n &&
        Heap[left] < Heap[s] )
        s = left;
    if( right < n &&
        Heap[right] < Heap[s] )
        s = right;
    if( s != i ) {
        int tmp = Heap[i];
        Heap[i] = Heap[s];
        Heap[s] = tmp;
        swapDown(s);
    }
}
```

21 / 32

Insert

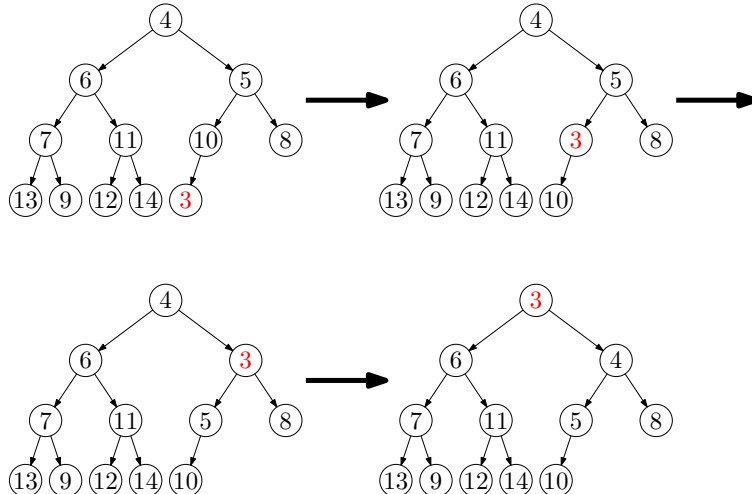


Invariant violated! Child has smaller key than parent.

22 / 32

Swap (Heapify) Up

Put new element last then swap it up to its proper position.



23 / 32

Insert Code

```
void insert(int x) {
    assert(!isFull());
    Heap[n] = x;
    n++;
    swapUp(n-1);
}
```

Runtime:

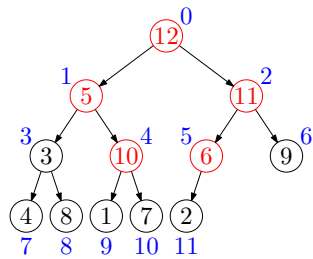
```
void swapUp(int i) {
    if( i == 0 ) return;
    int p = (i - 1)/2;
    if( Heap[i] < Heap[p] ) {
        int tmp = Heap[i];
        Heap[i] = Heap[p];
        Heap[p] = tmp;
        swapUp(p);
    }
}
```

24 / 32

Heapify: Build a Heap from a non-Heap Array

1. Start with the input array.

12	5	11	3	10	6	9	4	8	1	7	2
----	---	----	---	----	---	---	---	---	---	---	---

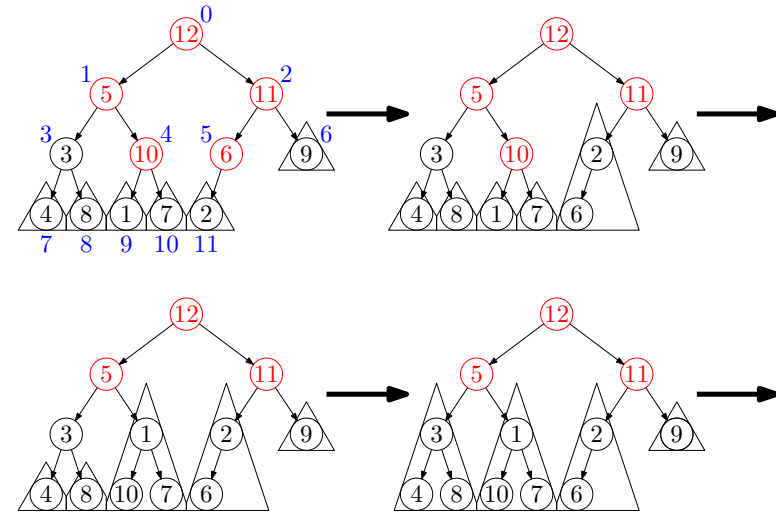


Invariant violated!

2. Fix the heap-order property bottom up. Use swapDown.

```
for( i=n/2-1; i >=0; i-- ) swapDown(i);
```

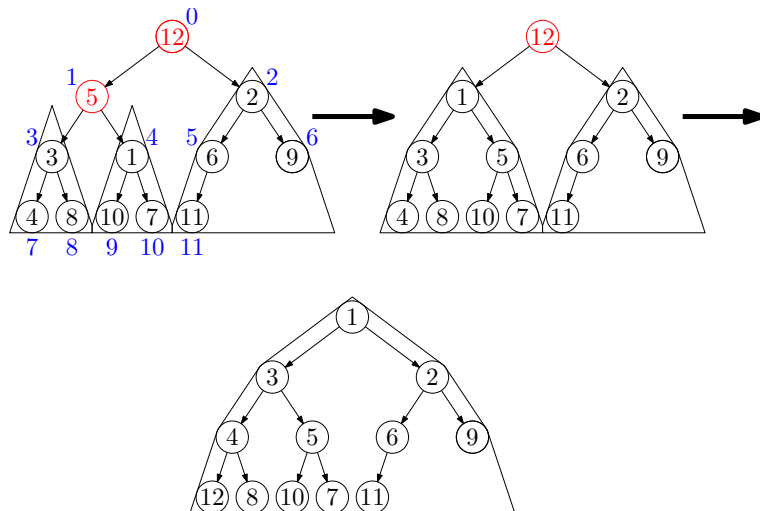
Heapify Example...



25 / 32

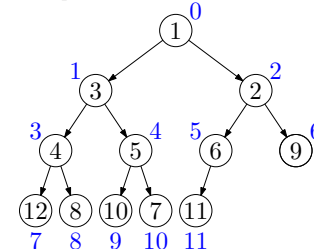
26 / 32

Heapify Example



Heapify Runtime

swapDown on a heap of height h takes at most _____ steps.



Let H be the height of the heap.

swapDown is called

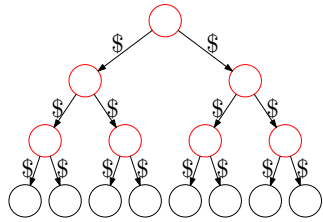
once	on heap of height	H
≤ 2 times	on heap of height	$H - 1$
≤ 4 times	on heap of height	$H - 2$
\vdots		
$\leq 2^{H-1}$ times	on heap of height	1

$$\text{Total \# steps} \leq \sum_{h=1}^H h 2^{H-h} = 2^H \sum_{h=1}^H h/2^h \leq 2^{H+1} = O(n)$$

27 / 32

28 / 32

Heapify Runtime: Charging Scheme



Possible **violations**. How much time to fix them?
 Place a dollar on each edge of the heap. One dollar pays for one step of `swapDown`. By induction, we can show that when `swapDown` is called on a node v , both children of v have a path (the rightmost path) to a leaf that is uncharged. The edges on the left child's rightmost path plus the edge to the left child pay for the steps of `swapDown` at v . The edges on the right child's rightmost path plus the edge to the right child form the uncharged path available to the parent of v .

29 / 32

Thinking about Binary Heaps

Observations

- ▶ finding a child/parent index is a multiply/divide by two
- ▶ `deleteMin` and `insert` access far-apart array locations
- ▶ `deleteMin` accesses all children of visited nodes
- ▶ `insert` accesses only parent of visited nodes
- ▶ `insert` is at least as common as `deleteMin`

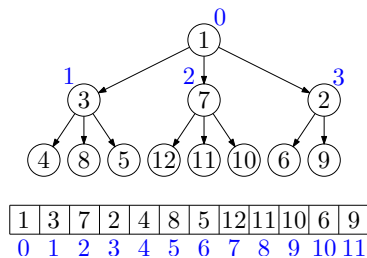
Realities

- ▶ division and multiplication by powers of two are fast
- ▶ far-apart array accesses ruin cache performance
- ▶ with huge data sets, disk I/O dominates

30 / 32

Solution: d -Heaps

Nearly complete d -ary trees (representable by array) with Heap-order property.



Good choices for d :

- ▶ fit one set of children on a memory page/disk block
- ▶ fit one set of children in a cache line
- ▶ optimize performance based on ratio of `inserts/deleteMins`
- ▶ make d a power of two for efficiency

31 / 32

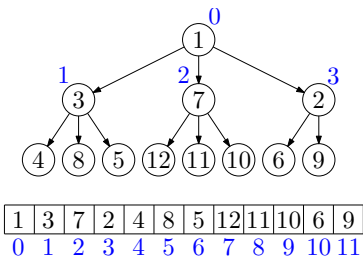
d -Heap Navigation

▶ j th-child(i) =

▶ parent(i) =

▶ root =

▶ next free position =



32 / 32