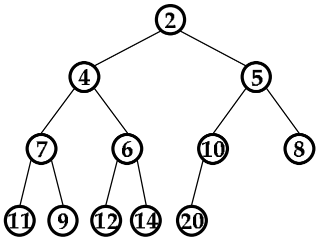
CSE 373: Heaps
(other operations and variations)

Chapter 6

Heaps: Quick Recap

Heaps:

- structure is a complete binary tree
- each node must be smaller than its descendants

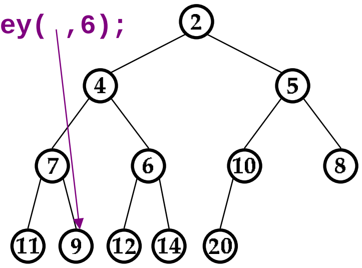


- main operations are **`insert()`** and **`deleteMin()`**
- heaps have a compact array-based representation

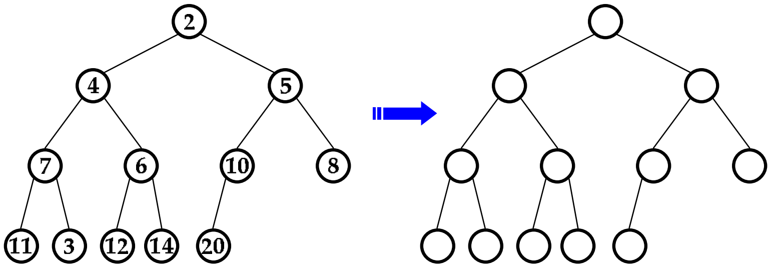
Other Operations: decreaseKey()

decreaseKey() – lowers a node's value
(while preserving heap ordering)

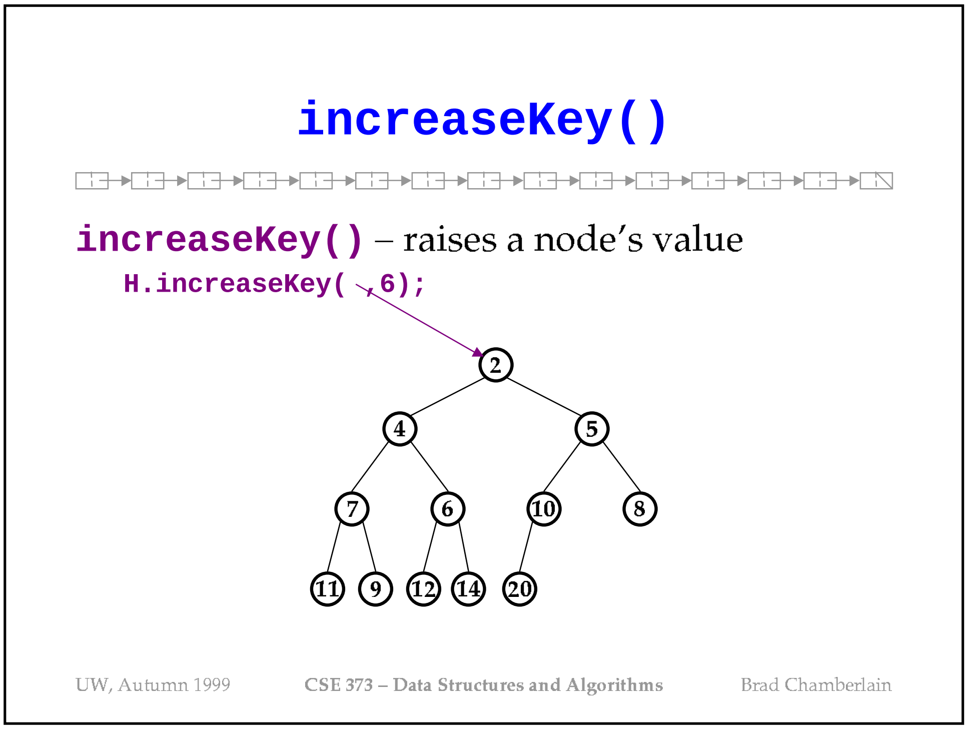
```
H.decreaseKey( ,6);
```



decreaseKey() – Continued



running time?

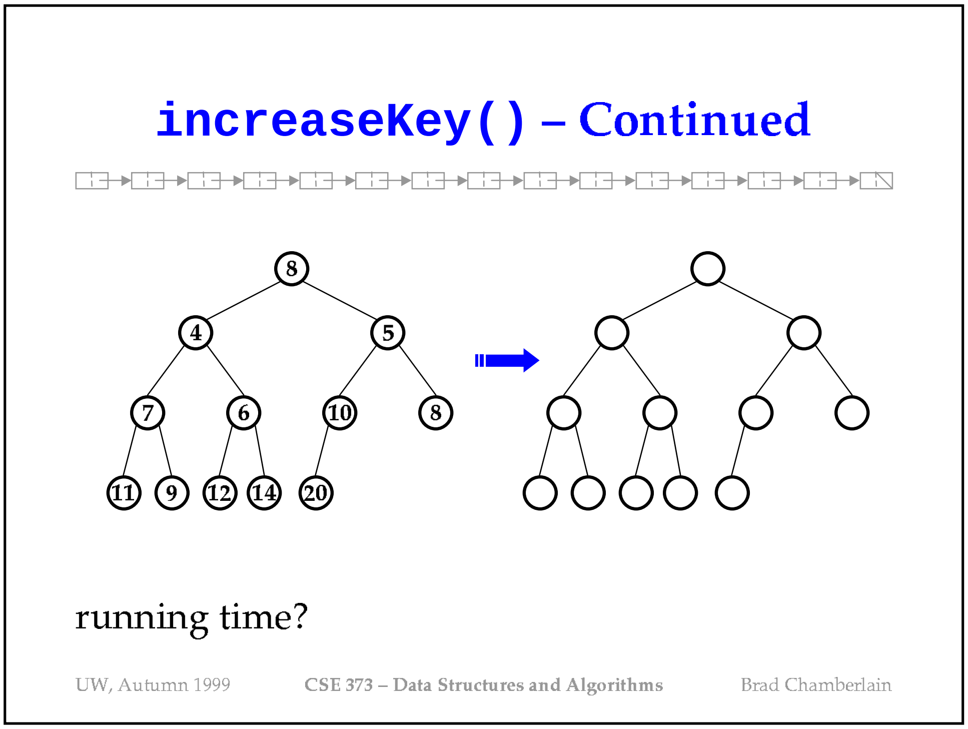


increaseKey()

`increaseKey()` – raises a node's value

```
H.increaseKey( , 6);
```

UW, Autumn 1999 CSE 373 – Data Structures and Algorithms Brad Chamberlain



increaseKey() – Continued

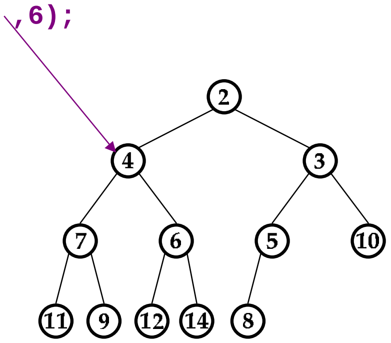
running time?

UW, Autumn 1999 CSE 373 – Data Structures and Algorithms Brad Chamberlain

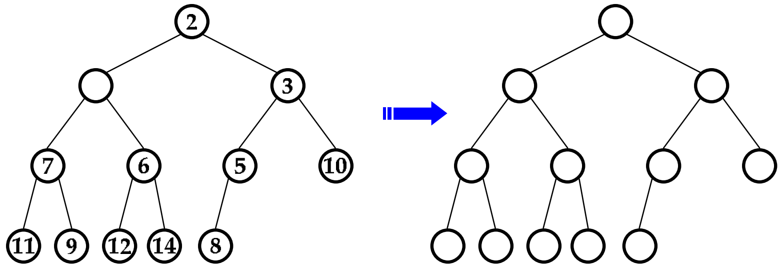
delete()

delete() – removes a node from the heap

```
H.delete( ,6);
```



delete() – Continued



running time?

Let’s Write a Heap Routine…

buildHeap()

buildHeap() – creates a heap from an array

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

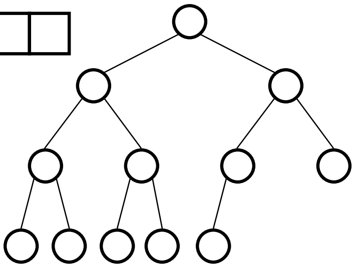
Straightforward Implementation: **insert()** elements into an empty heap one at a time

running time?

buildHeap() – Continued

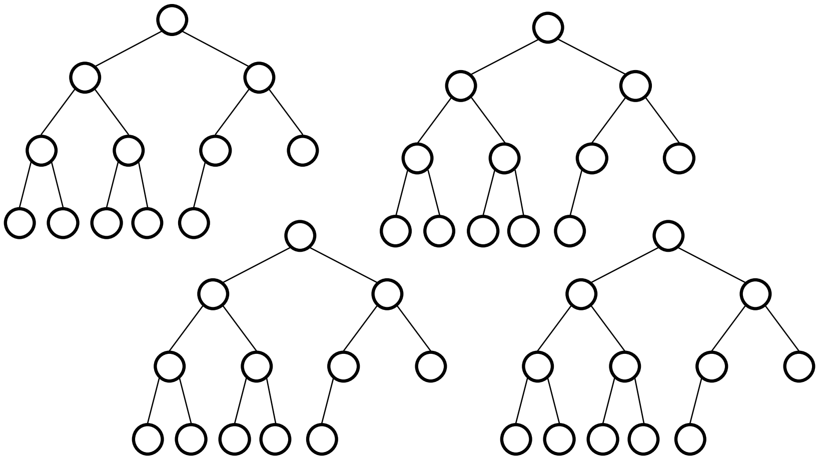
Better Implementation: Treat input array as a heap and "percolate down" first $n/2$ values

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

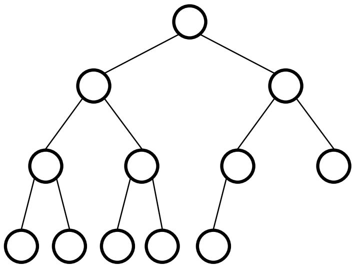


running time?

buildHeap() – even more



`buildHeap()` running time



MaxHeaps

MaxHeaps: the dual of the Heaps we've defined

- support fast `insert()` and `deleteMax()` ops
- work exactly the same as (Min)Heaps

Why is `deleteMax()` expensive on a normal minheap?

What's the running time?

d-Heaps

d-Heaps: Just like normal heaps but with *d* children rather than 2

Intuition: tree will be shallower so ops will be faster

However…

- more comparisons need to be made when percolating down
- finding parent/children may be slower

What about asymptotic running time?

Bottom Line: 4-heaps ***may*** outperform 2-heaps

Merging Heaps

How to merge heaps effectively?

Straightforward method: copy both arrays into a single array and use **`buildHeap()`**

running time?

Advanced methods:

- pointer-based imbalanced heaps
  - *leftist heaps* – a bit like AVL trees; O(log *n*) merge
  - *skew heaps* – like Splay trees; O(log *n*) amortized ops
  - *binomial queues* – O(log *n*) merge, but ~O(1) insert