Heaps / Priority Queue Implementation

MSCI 240: Algorithms & Data Structures

lecture summary

no more content after these slides!

heaps

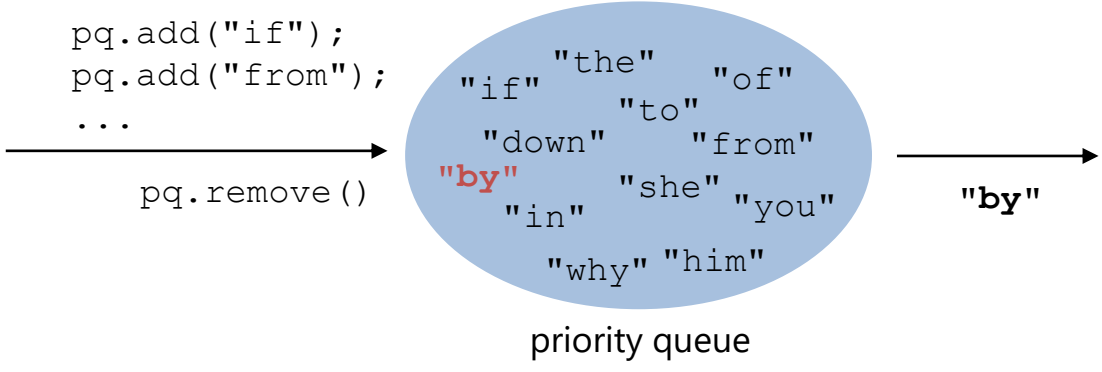
- inserting
- removing
- complexity

priority queue implementation with min-heap array

Topic	Building Java Programs	Algorithms (Sedgewick)
classes, ADTs	chapter 8	1.2
arrays	chapter 7	
ArrayList<T>	chapter 10	1.3
Stack/Queue	chapter 14, (11)	1.3
LinkedList	chapter 16	1.3
Complexity	chapter 13	1.4
Searching		pp. 46-47
Sorting		chapter 2.1-2.3
Recursion	chapter 12	1.1 (p. 25)
Binary Trees	chapter 17	chapter 3.1-3.2
Dictionaries	chapter 18.1	chapter 3.4
Graphs	N/A (Wikipedia good)	chapter 4.1
Heaps/Priority Queues	chapter 18.2	chapter 2.4

priority queue: a collection of ordered elements that provides fast access to the minimum (or maximum) element

`add(element)`	adds in order
`peek()`	returns minimum or "highest priority" value
`remove()`	removes/returns minimum value
`isEmpty()`, `clear()`, `size()`	



recall: binary search trees

add/insert complexity

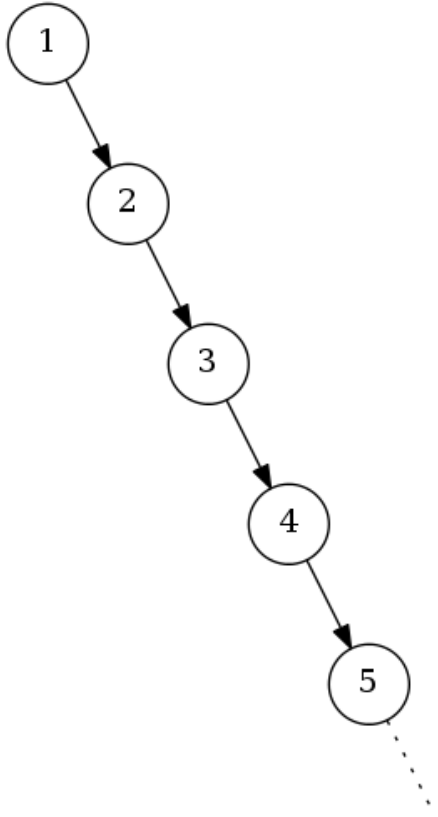
average case → $O(\log n)$

worst case → $O(n)$

search/contains complexity

average case → $O(\log n)$

worst case → $O(n)$



heap

also a binary tree

not a binary search tree

difference?

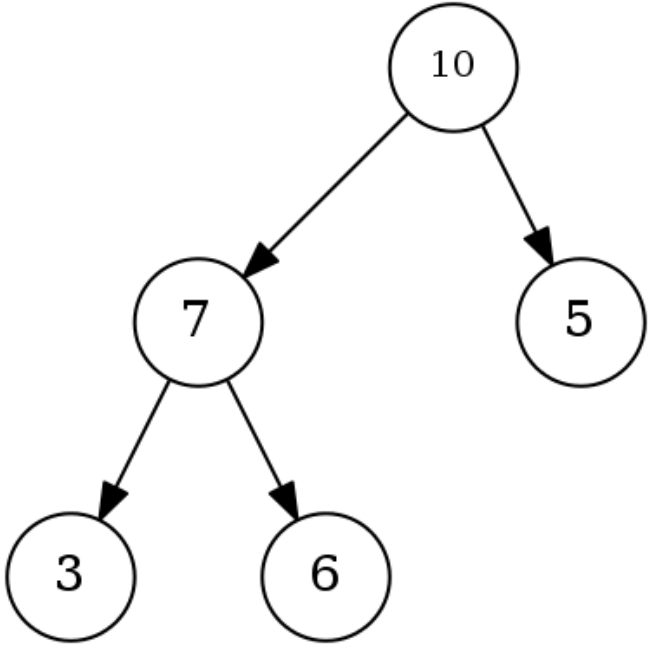
always balanced

two kinds: max heap, min heap

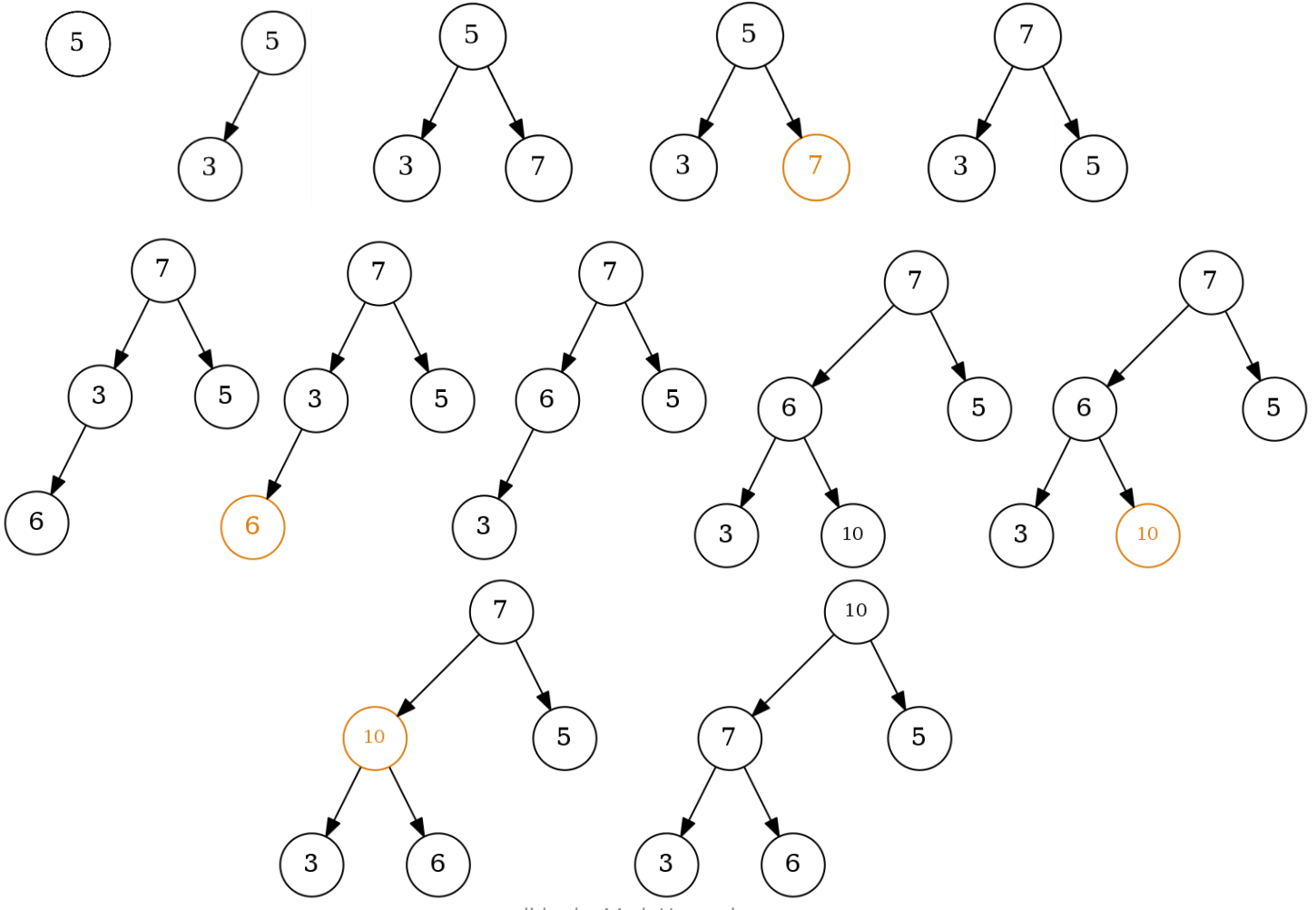
max-heap property: for all nodes n other than the root node, the value of n's parent is $\geq$ the value of the node n

e.g., add 5, 3, 7, 6, 10

"swim" nodes up if they don't satisfy max-heap property



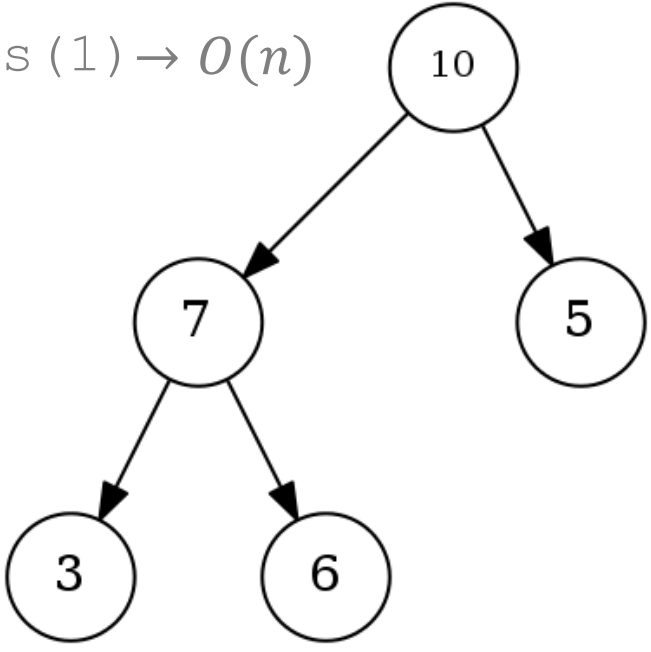
e.g., add 5, 3, 7, 6, 10 (stages)



max element always at top

complexity

- add → # swaps? → $O(\log n)$
- search/contains → e.g., `contains(1)` → $O(n)$
- remove?

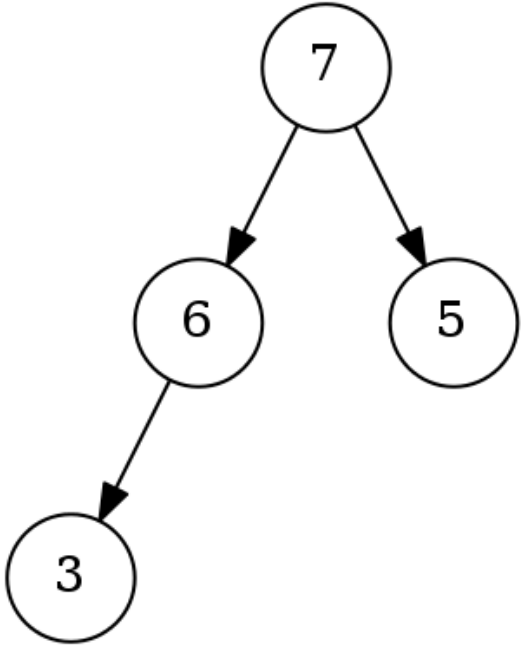


remove max

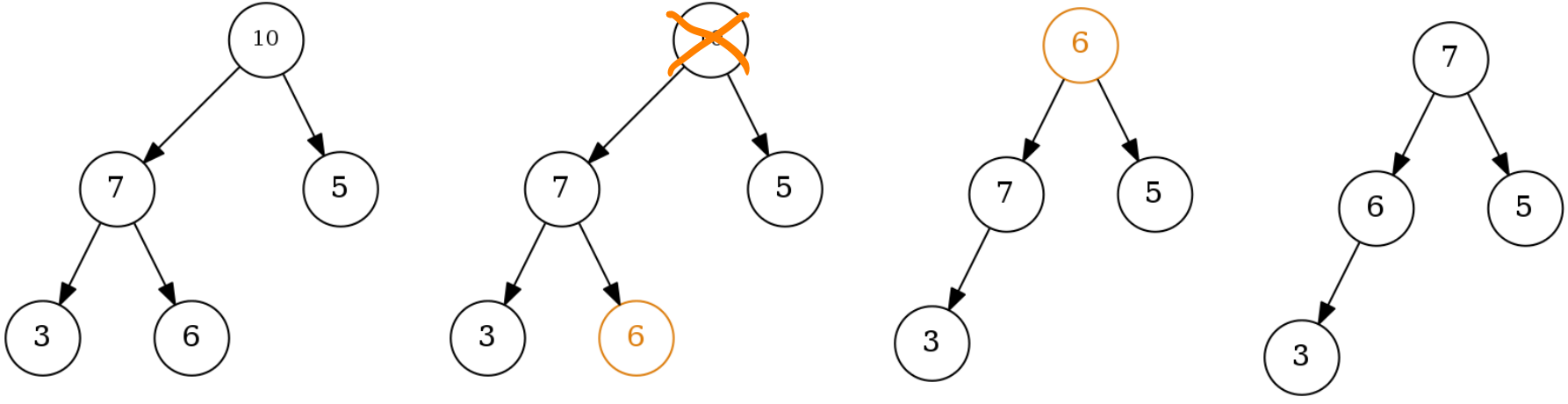
"sink" nodes down if they
don't satisfy max-heap propertv

complexity→ # swaps?→ $O(\log n)$

how would you find k^{th} largest
(e.g., Q1, Q3 quartiles, median)



e.g., remove max



heaps can be (and typically are) stored in arrays

index	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	...
value		10	7	5	3	6				

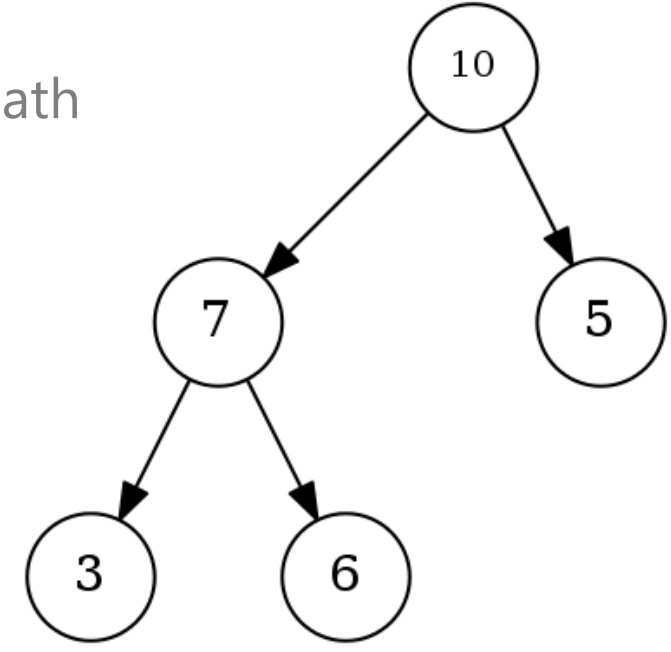
index of root = 1

leave 0 empty to simplify the math

$a[i]$'s left child @ $a[2i]$

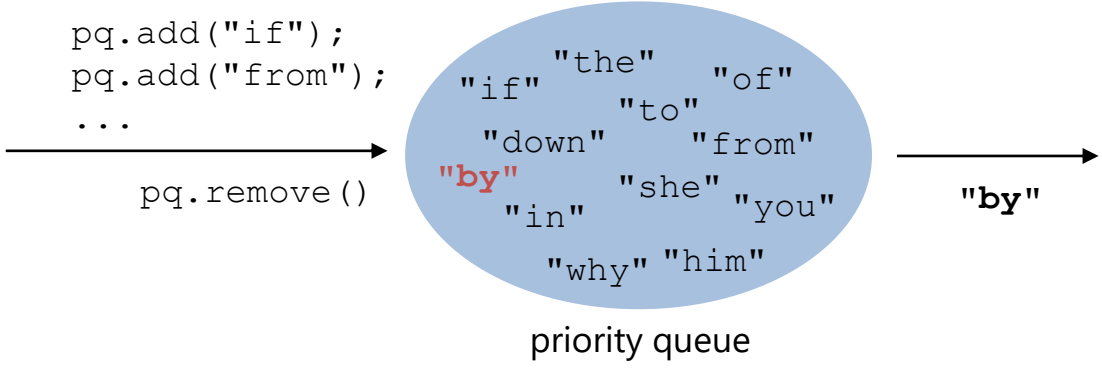
$a[i]$'s right child @ $a[2i + 1]$

$a[i]$'s parent @ $a[i/2]$



priority queue: a collection of ordered elements that provides fast access to the minimum (or maximum) element

`add(element)`	adds in order
`peek()`	returns minimum or "highest priority" value
`remove()`	removes/returns minimum value
`isEmpty()`, `clear()`, `size()`	



let's implement an `int` priority queue using a min-heap array

```
public class HeapIntPriorityQueue {
    private ArrayList<Integer> elements;

    // constructs a new empty priority queue
    public HeapIntPriorityQueue() {
        elements = new ArrayList<Integer>();
        elements.add(-1); // dummy 0th element
    }

    // ...

}
```

since we will treat the array as a complete tree/heap, and walk up/down between parents/children, these methods are helpful:

```
private int parent(int index)         { return index / 2; }
private int leftChild(int index)      { return index * 2; }
private int rightChild(int index)     { return index * 2 + 1; }
private boolean hasParent(int index) { return index > 1; }

private boolean hasLeftChild(int index) {
    return leftChild(index) < elements.size();
}

private boolean hasRightChild(int index) {
    return rightChild(index) < elements.size();
}

private void swap(int index1, int index2) {
    int temp = elements.get(index1);
    elements.set(index1, elements.get(index2));
    elements.set(index2, temp);
}
```

```
// Adds the given value to this priority queue in order.
public void add(int value) {
    elements.add(value); // add as rightmost leaf

    // "swim up" as necessary to fix ordering
    int index = elements.size() - 1;
    boolean found = false;
    while (!found && hasParent(index)) {
        int parent = parent(index);
        if (elements.get(index) < elements.get(parent)) {
            swap(index, parent);
            index = parent;
        } else {
            found = true; // found proper location; stop
        }
    }
}
```

```
// Adds the given value to this priority queue in order.
public void add(int value) {
    elements.add(value); // add as rightmost leaf

    // "swim up" as necessary to fix ordering
    int i = elements.size() - 1;
    while (hasParent(i) &&
           elements.get(i) < elements.get(parent(i))) {
        swap(i, parent(i));
        i = parent(i);
    }
}
```

heap summary

heaps

- adding (swim up)
- removing (sink down)
- complexity $O(\log n)$ for both, because remains balanced
- peek (i.e., getMax) is $O(1)$

priority queue implementation with min-heap array

- can use array to store heap, children at $2i$ and $2i + 1$

clicker questions

an array sorted in descending order is a max-heap

A. true
B. false

what would a max-heap backed by an array look like after inserting the following numbers: 4,3,9,10,1,2,5?

A.

1	2	3	4	5	6	7	…
10	9	5	4	3	2	1	…

B.

1	2	3	4	5	6	7	…
1	2	3	4	5	9	10	…

C.

1	2	3	4	5	6	7	…
1	3	2	10	4	9	5	…

D.

1	2	3	4	5	6	7	…
10	9	5	3	1	2	4	…

E.

1	2	3	4	5	6	7	…
10	5	9	2	1	4	3	…

what would a min-heap backed by an array look like after inserting the following numbers: 4,3,9,10,1,2,5?

A.

1	2	3	4	5	6	7	...
10	9	5	4	3	2	1	...

B.

1	2	3	4	5	6	7	...
1	2	3	4	5	9	10	...

C.

1	2	3	4	5	6	7	...
1	3	2	10	4	9	5	...

D.

1	2	3	4	5	6	7	...
10	9	5	3	1	2	4	...

E.

1	2	3	4	5	6	7	...
10	5	9	2	1	4	3	...

Monday, Dec 3:

review

sports day 2

Friday, Dec 14:

exam: 7:30-10:00pm, RCH 103/105 (same as midterm)