Priority Queues

Priority Queue Operations

- Enqueue items with a priority
- Dequeue item with the highest priority
- Highest - return item with the highest priority

Priority Queue Operations

- Uses for a priority queue
  - Printers
  - Office hours
  - Emergency room

Priority Queue Implementations

- In class exercise - Evaluate and choose the best ADT implementation
  - Data Structure
    - Array based list
    - Linked list based list
    - Tree
  - Operations
    - Enqueue - enqueue with a priority
    - Dequeue - dequeue highest priority
    - Highest - return highest priority

Priority Queue Implementations

	Array	Linked List	BST
Insert at front or end (unsorted)			n/a
Search for item & deQ			n/a
Search for highest			n/a

OR

Insert sorted
deQ highest item
Return highest item

Priority Queue Implementations

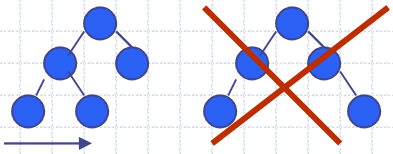
	Array	Linked List	BST
Insert at front or end (unsorted)			n/a
Search for item to deQ			n/a
Search for highest			n/a

OR

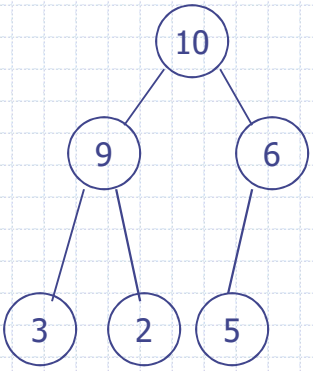
Insert sorted
deQ highest item
Return highest item

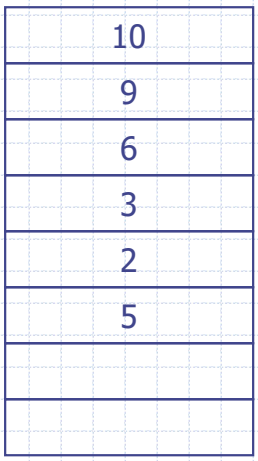
Binary Heaps/Heaps

- Similar to a BST
- Heap is sorted in a much weaker sense however the sorting is sufficient for a priority queue
- Tree is always a complete tree
  - Bottom level may not be filled but fills from left to right



Max Heap





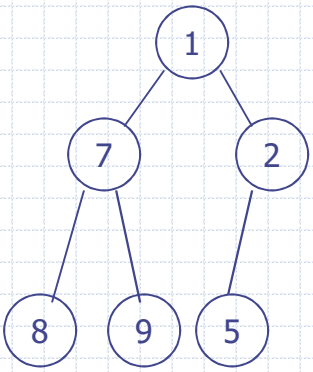
Heaps

- Full binary trees are easily stored in an array
  - Must know max size but no wasted space
    - Parent = (i-1)/2
    - Left Child = 2i + 1
    - Right Child = 2i + 2

Heap Ordering Properties

- Min Heap
  - For each node X, X.item >= (X.parent).item
    - Smallest number is at the root
- Max Heap
  - For each node X, X.item <= (X.parent).item
    - Largest number is at the root

Min Heap



1
7
2
8
9
5

Heap ADT

HEAP class

int MAXNODES	*set to current array size*
int numNodes	*current #of items in heap*
*itemtype *Arr*	*dynamically allocated space for MAXNODES items (can realloc if needed)*

Heap - Enqueue

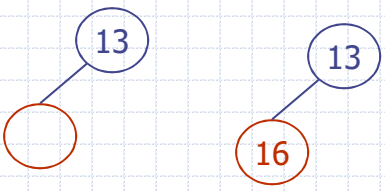
- Create a hole (empty node) in the next available complete tree location
  - Remember to fill bottom layer from left to right
- If the item can be inserted into the hole without violation of the heap property, insert item
- Otherwise, copy the holes parent item into the hole then trickle up

Min Heap Enqueue

13, 16, 19, 14, 23, 17, 6

Enqueue 13

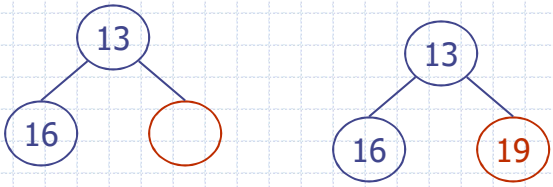
Enqueue 16



Min Heap Enqueue

13, 16, 19, 14, 23, 17, 6

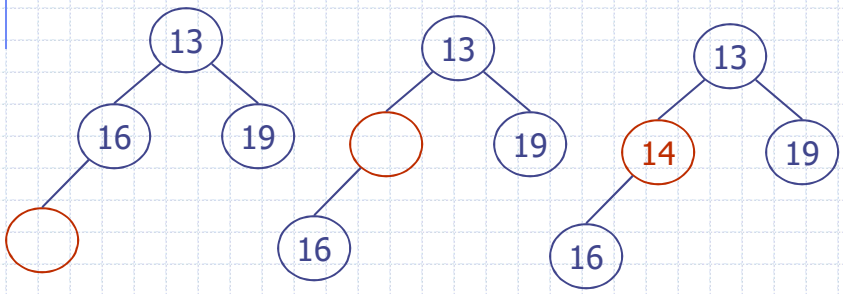
Enqueue 19



Min Heap Enqueue

13, 16, 19, 14, 23, 17, 6

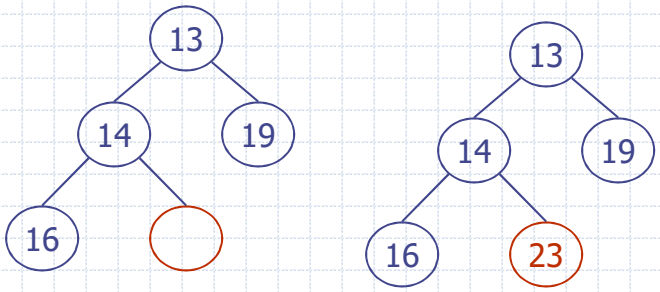
Enqueue 14



Min Heap Enqueue

13, 16, 19, 14, 23, 17, 6

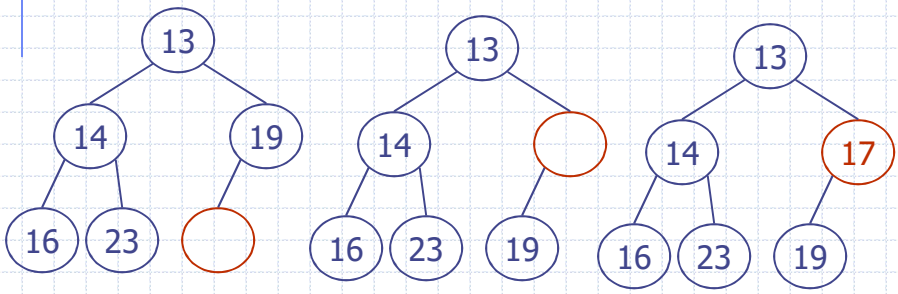
Enqueue 23



Min Heap Enqueue

13, 16, 19, 14, 23, 17, 6

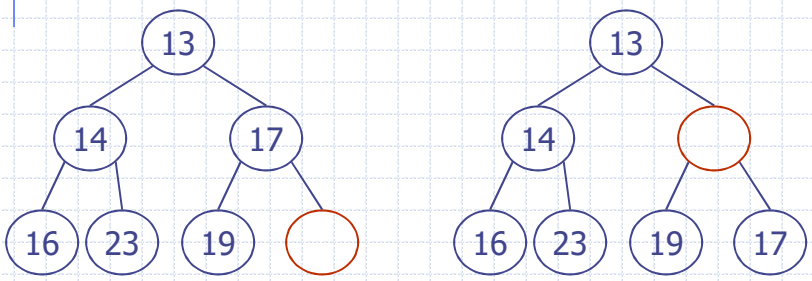
Enqueue 17



Min Heap Enqueue

13, 16, 19, 14, 23, 17, 6

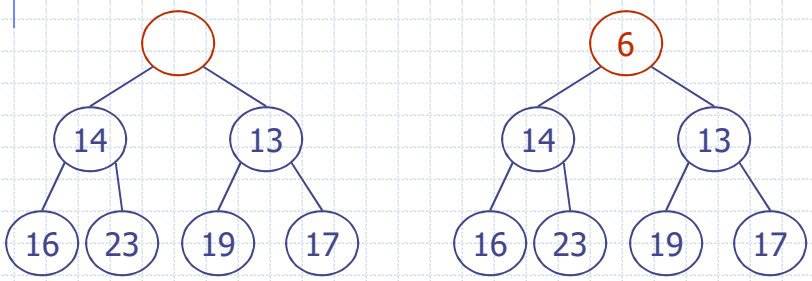
Enqueue 6



Min Heap Enqueue

13, 16, 19, 14, 23, 17, 6

Enqueue 6



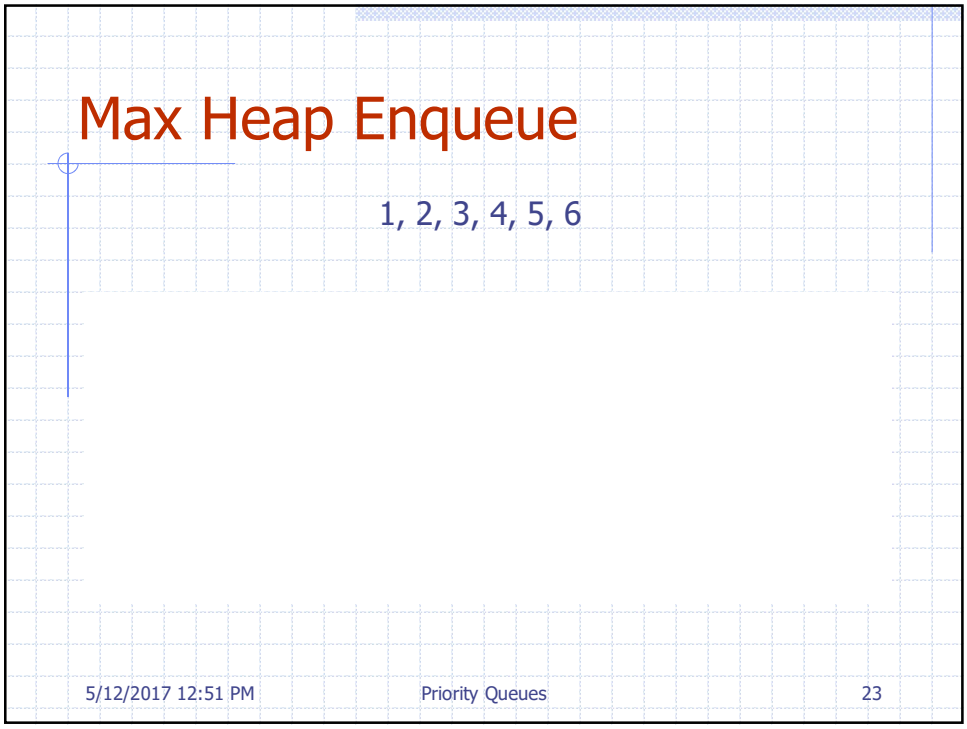
Max Heap Enqueue

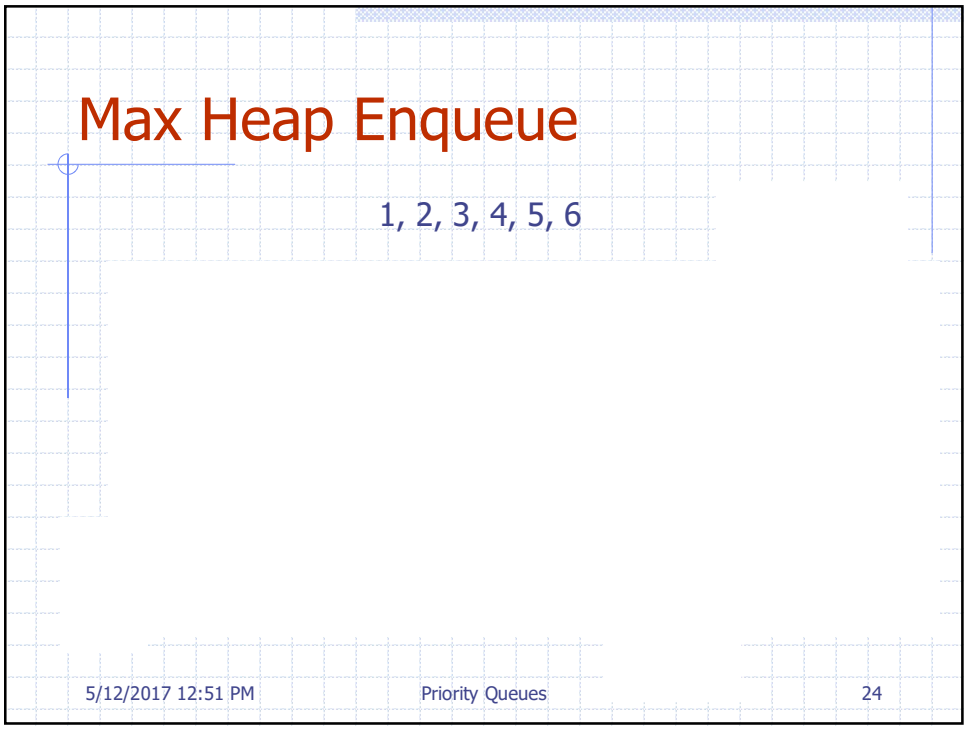
- In class exercise - Enqueue the following numbers into a max heap (remember that the **largest** number is at the root)

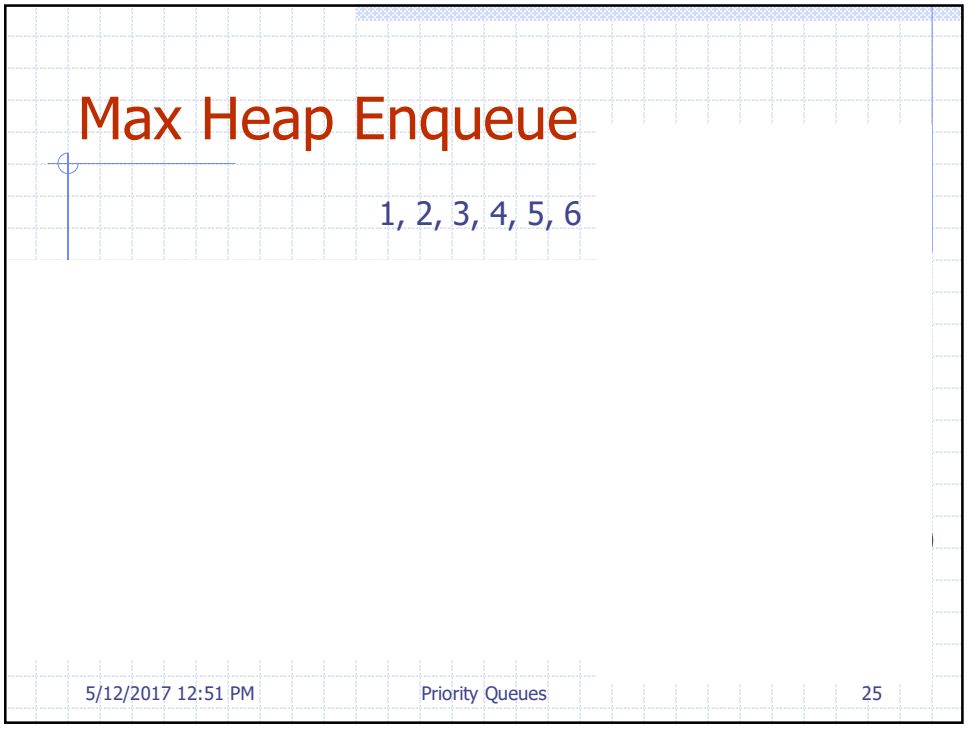
1, 2, 3, 4, 5, 6

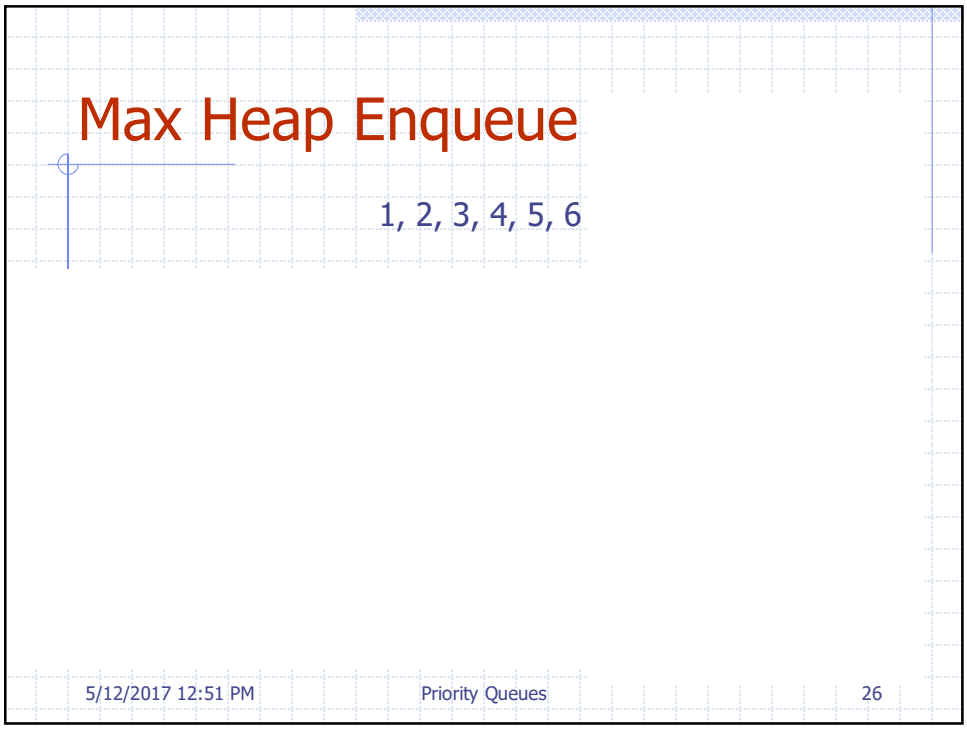
Max Heap Enqueue

1, 2, 3, 4, 5, 6


Max Heap Enqueue
1, 2, 3, 4, 5, 6
5/12/2017 12:51 PM
Priority Queues
23


Max Heap Enqueue
1, 2, 3, 4, 5, 6
5/12/2017 12:51 PM
Priority Queues
24


Max Heap Enqueue
1, 2, 3, 4, 5, 6
5/12/2017 12:51 PM
Priority Queues
25


Max Heap Enqueue
1, 2, 3, 4, 5, 6
5/12/2017 12:51 PM
Priority Queues
26

Min Heap Enqueue

```
void enqueueHeap ( itemtype item )
    if heap not full
        if heap empty
            arr[numNodes] = item
            ++numNodes
        else
            x = numNodes
            while (x > 0 && arr[(x-1)/2] > item)
                arr[x] = arr[(x-1)/2]
                x = (x-1)/2
            arr[x] = item
            ++numNodes
```

Running time =

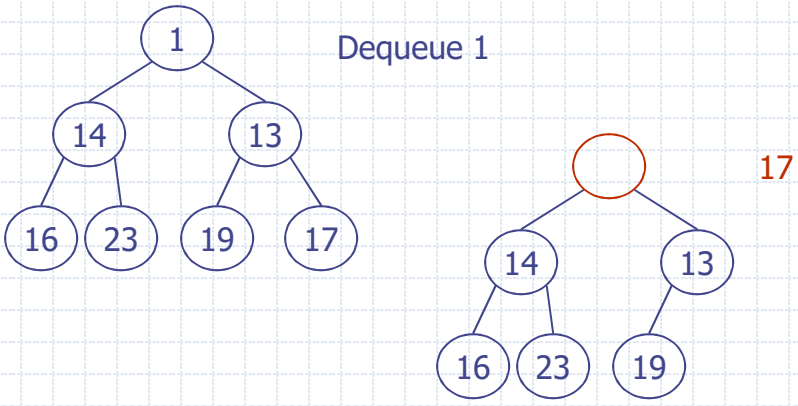
Heap

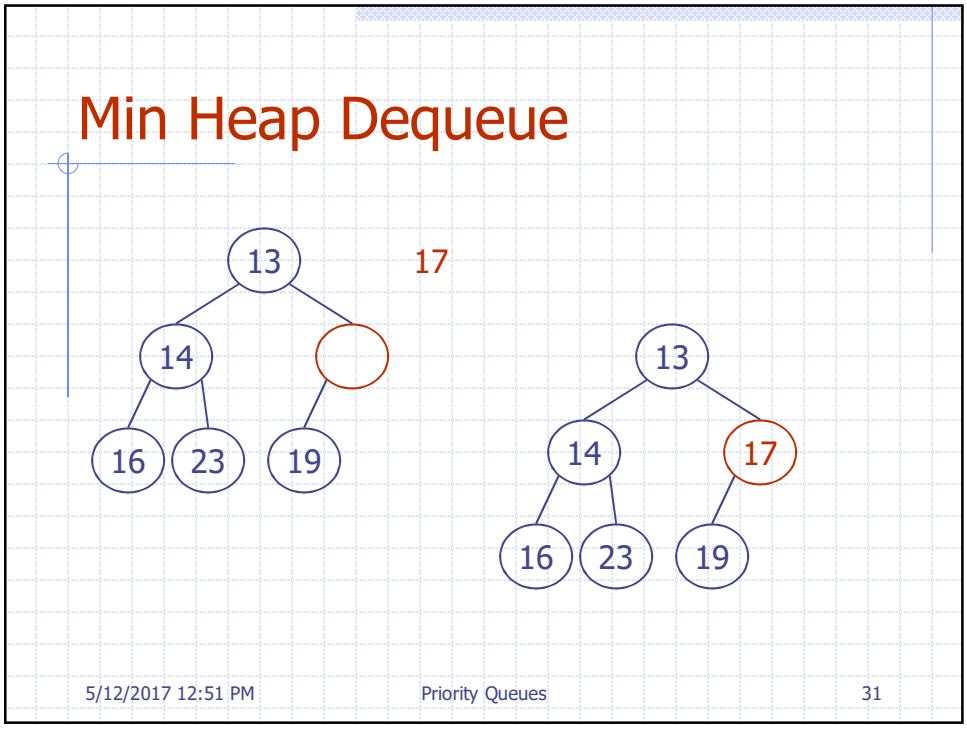
- Finding the node with the highest priority is easy - it is just at the root
  - Running time = O(1)

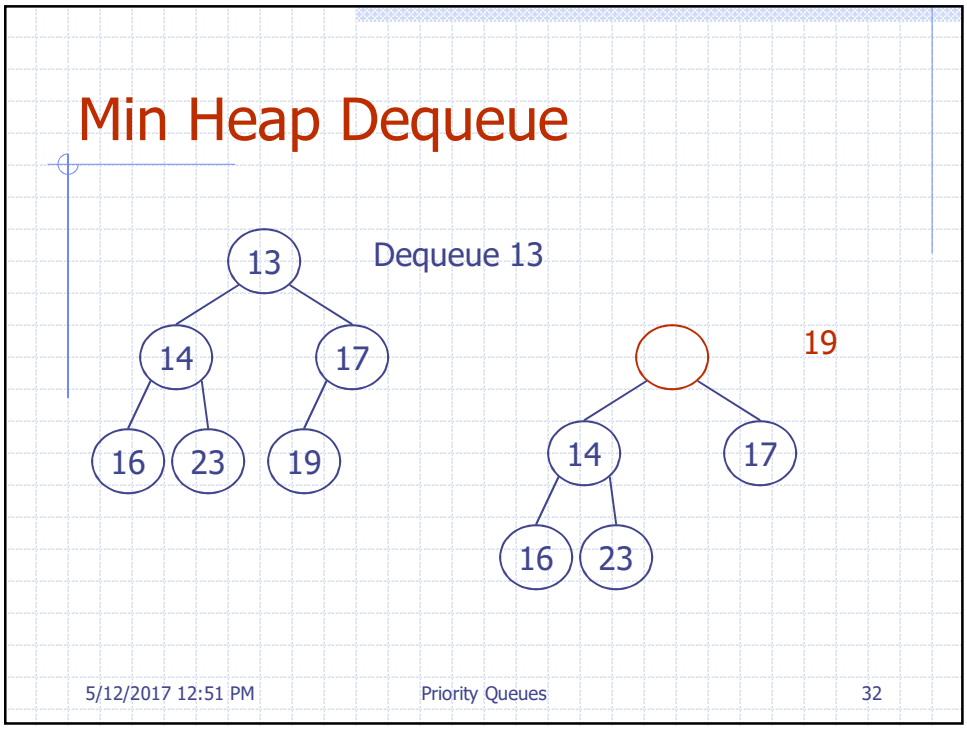
Heap Dequeue

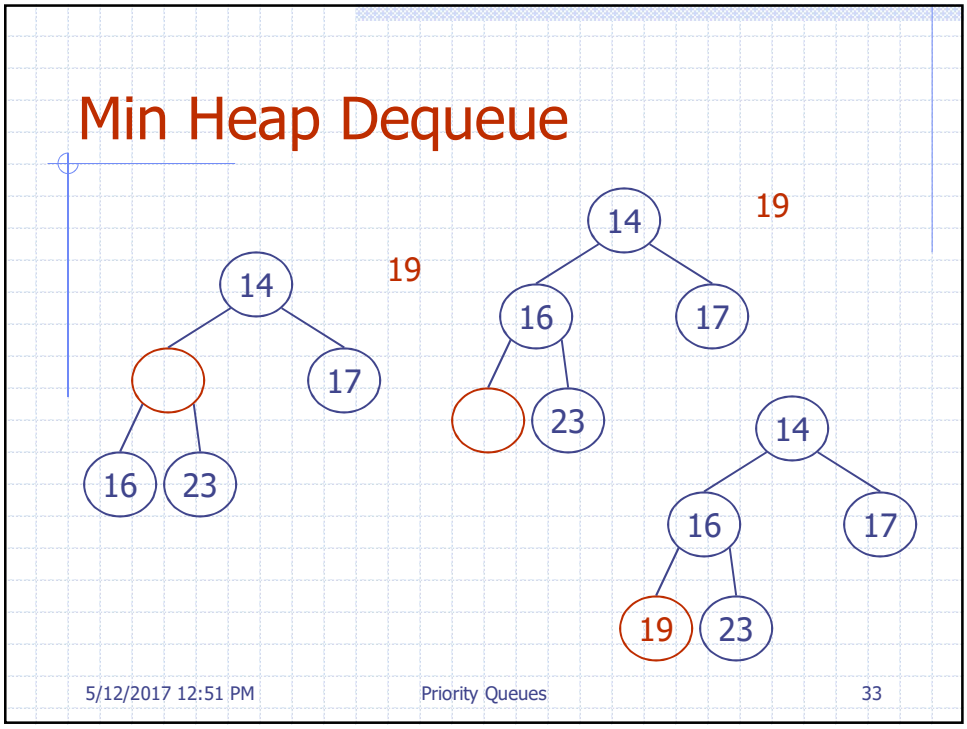
- Dequeue highest priority item
  - Makes a hole at the root
  - Want to remain a complete tree so place last item in the heap into the hole
    - If item can be placed in hole without violation of the heap property, then done
    - Otherwise, trickle down. Put the highest priority of the holes children into the hole

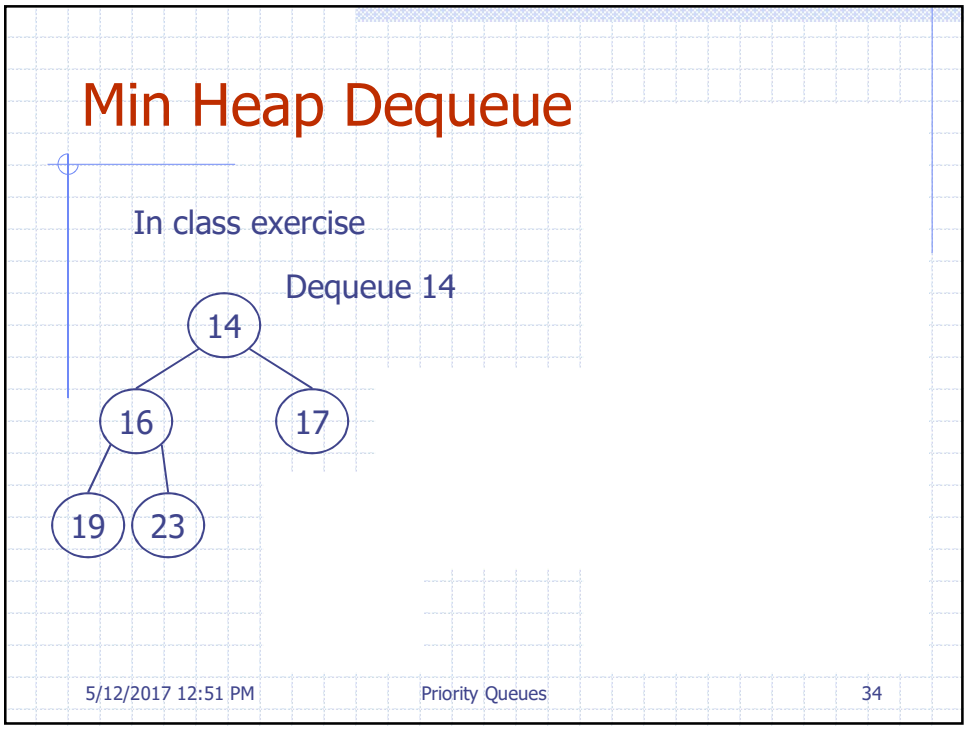
Min Heap Dequeue




Min Heap Dequeue
13
14
16
23
19
17
13
14
17
16
23
19
5/12/2017 12:51 PM
Priority Queues
31


Min Heap Dequeue
13
14
17
16
23
19
Dequeue 13
19
14
17
16
23
5/12/2017 12:51 PM
Priority Queues
32


Min Heap Dequeue
19
14
17
16
23
19
14
16
17
23
14
16
17
19
23
5/12/2017 12:51 PM
Priority Queues
33


Min Heap Dequeue
In class exercise
Dequeue 14
14
16
17
19
23
5/12/2017 12:51 PM
Priority Queues
34

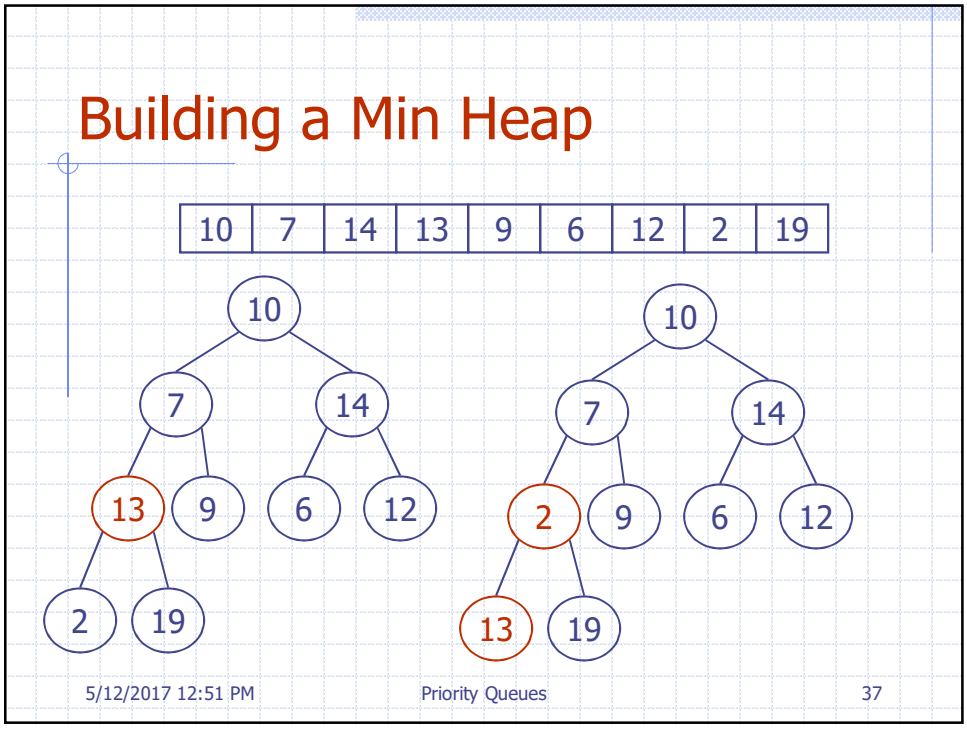
Heap Notes

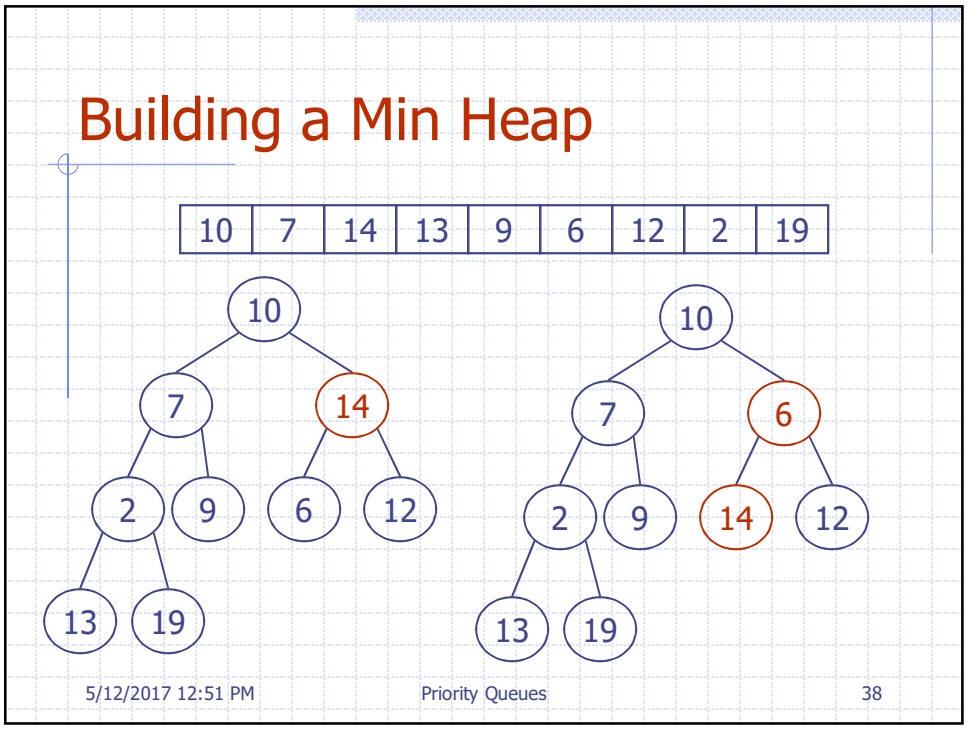
- Common error when writing code for dequeue
  - Check number of children before finding child with highest priority value
- Searching for an item requires looking at all nodes since there is no particular ordering
  - Search is not too common

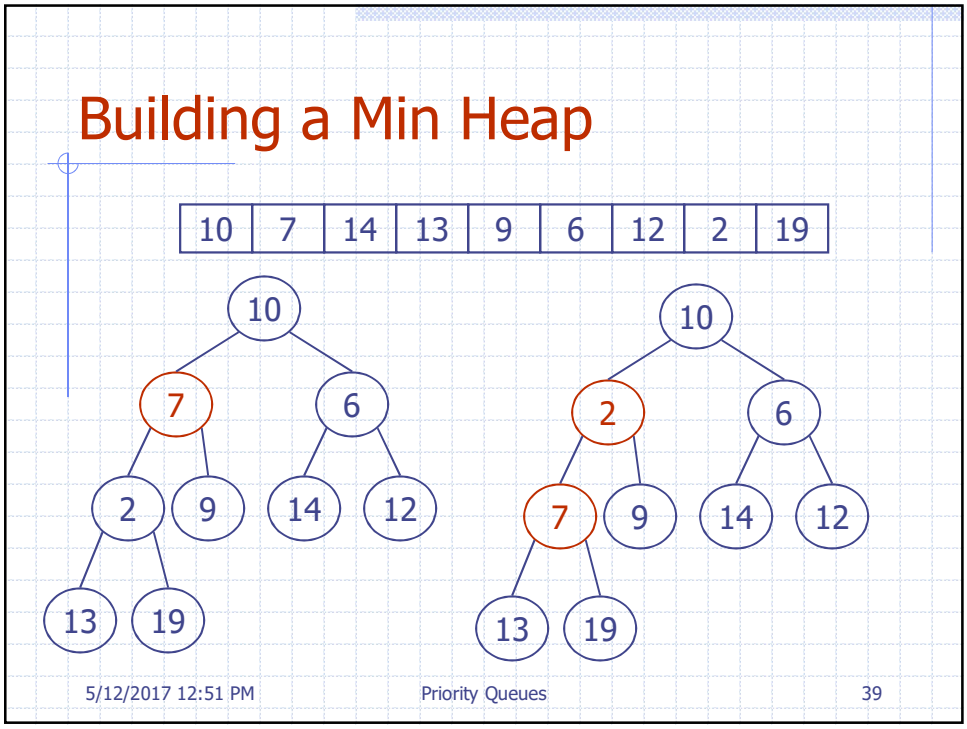
Building a Heap

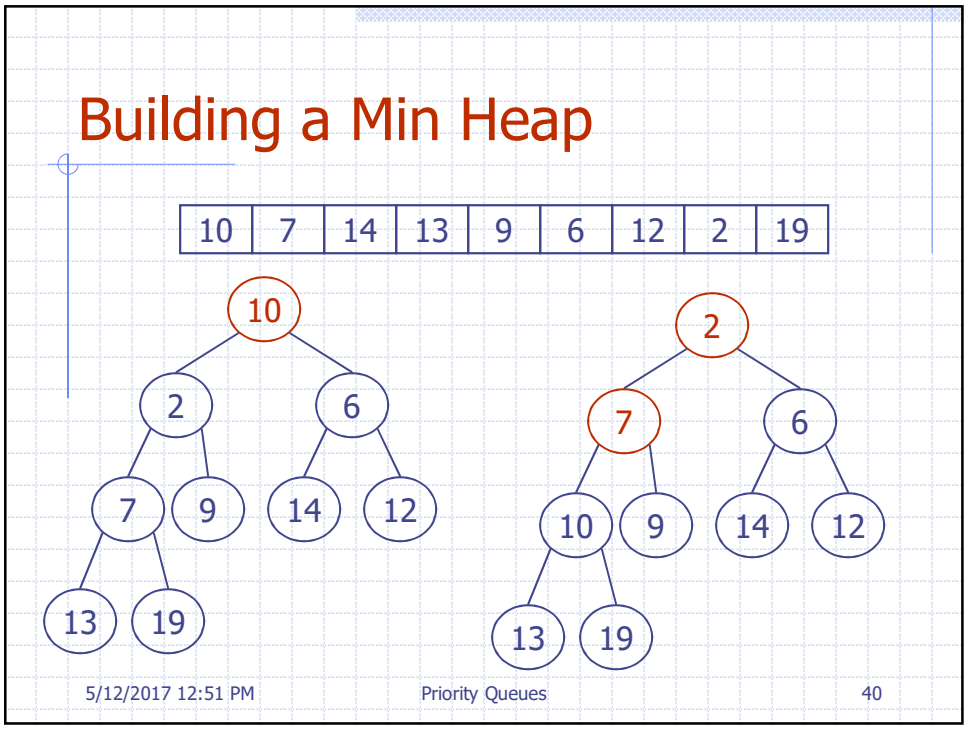
- Naïve method
  - Perform N enqueue operations - O(n lg n)
- Better solution
  - Given an array of numbers:

for (x = (N/2)-1; x >= 0; --x)
Trickle down


Building a Min Heap
10 7 14 13 9 6 12 2 19
10
7 14
13 9 6 12
2 19
10
7 14
2 9 6 12
13 19
5/12/2017 12:51 PM
Priority Queues
37


Building a Min Heap
10 7 14 13 9 6 12 2 19
10
7 14
2 9 6 12
13 19
10
7 6
2 9 14 12
13 19
5/12/2017 12:51 PM
Priority Queues
38


Building a Min Heap
10 7 14 13 9 6 12 2 19
10
7
6
2
9
14
12
13
19
10
2
6
7
9
14
12
13
19
5/12/2017 12:51 PM
Priority Queues
39

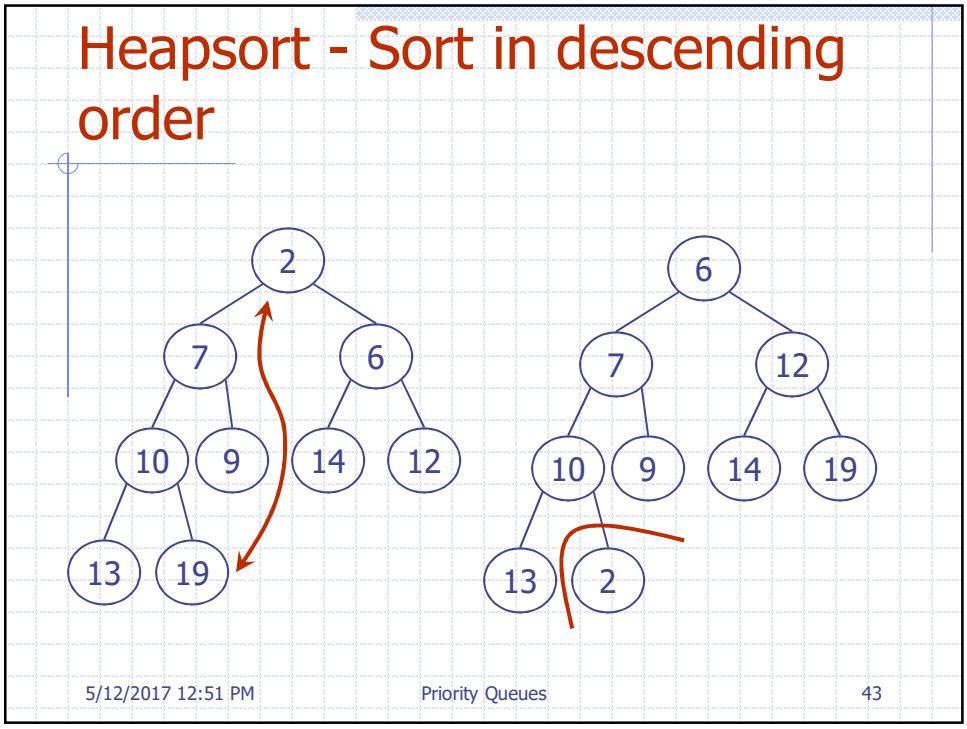

Building a Min Heap
10 7 14 13 9 6 12 2 19
10
2
6
7
9
14
12
13
19
2
7
6
10
9
14
12
13
19
5/12/2017 12:51 PM
Priority Queues
40

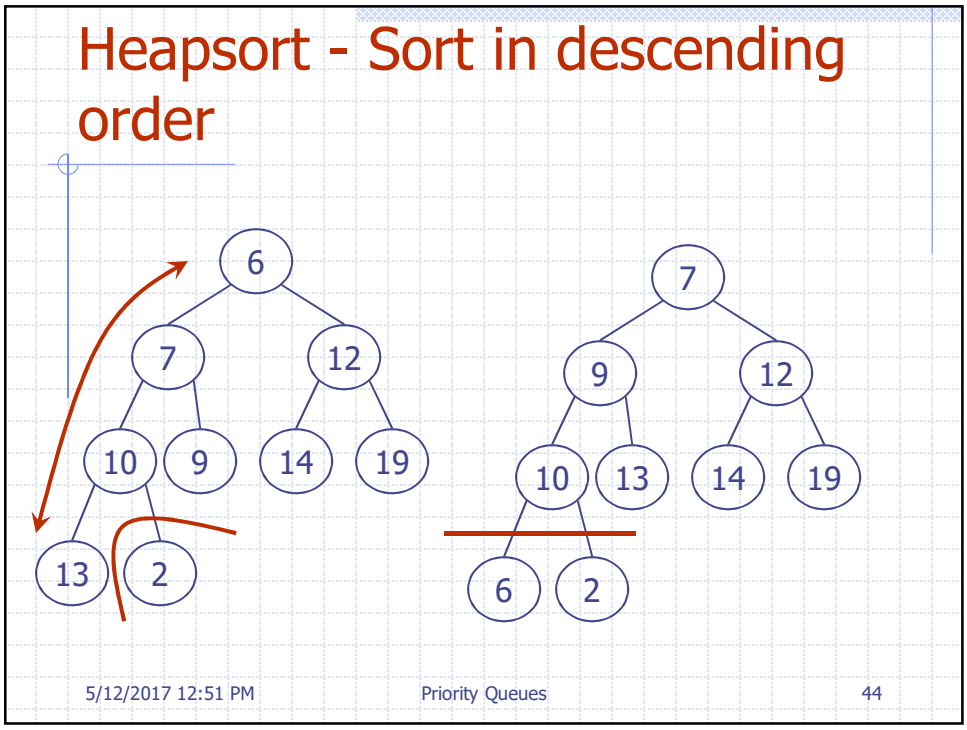
Building a Heap

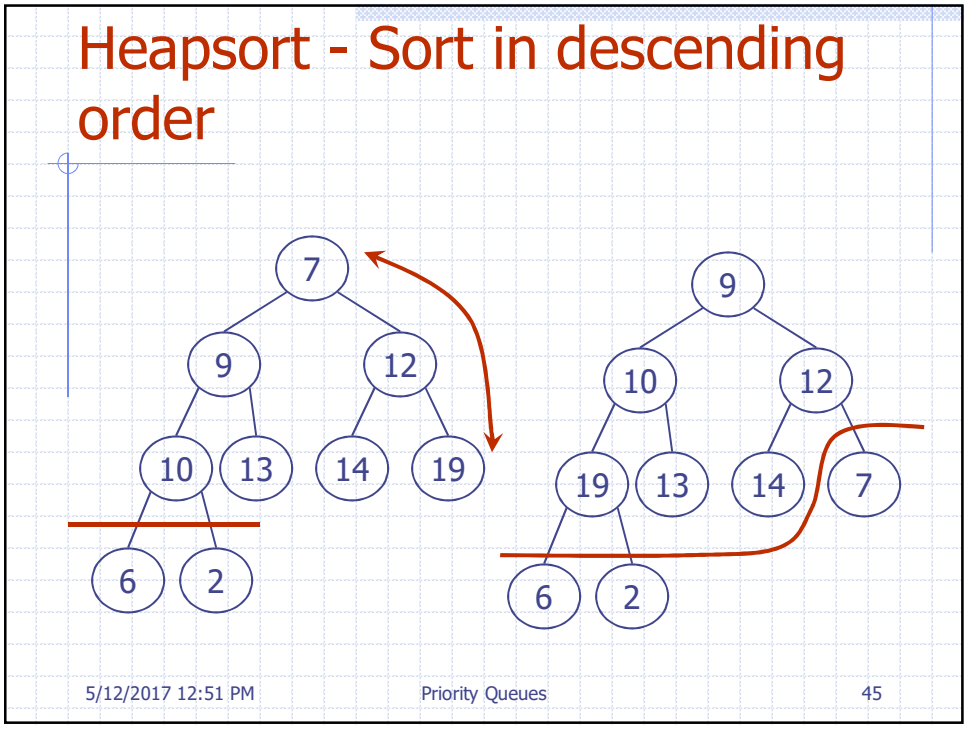
- Build heap algorithm actually runs in O(n) time
  - Beyond the scope of the class to prove

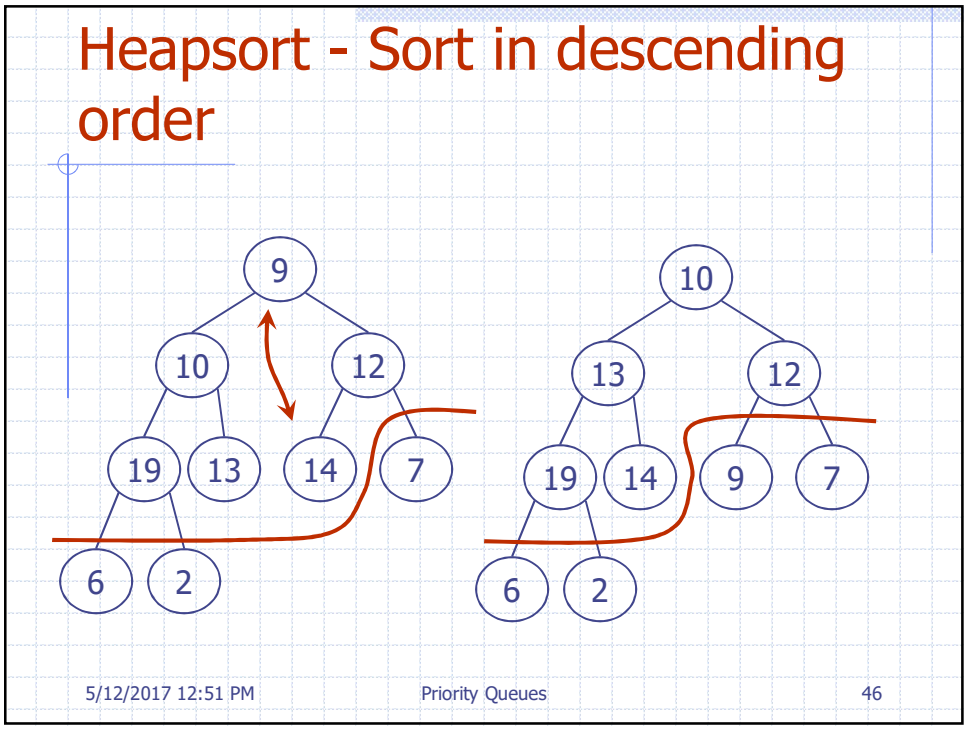
Heapsort

- Build heap for the opposite of what you want
  - Max heap for ascending order
  - Min heap for descending order
- Take root and place in last array position, then think of array as 1 smaller
- Trickle down from root to rebuild heap
- Continue until all items are moved

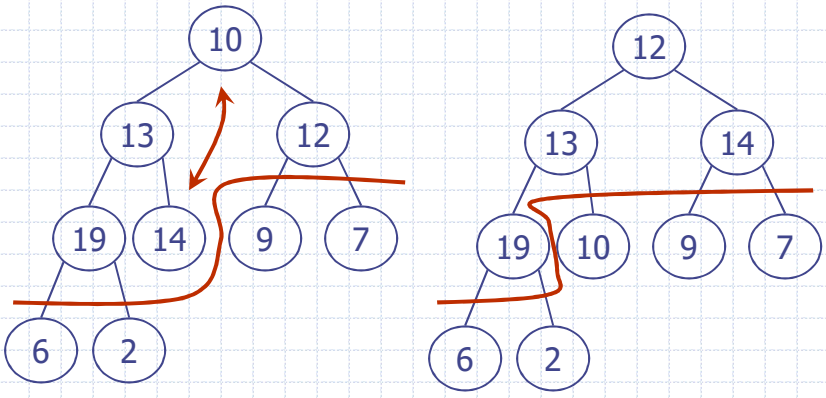

Heapsort - Sort in descending order
2
7
6
10
9
14
12
13
19
6
7
12
10
9
14
19
13
2
5/12/2017 12:51 PM
Priority Queues
43


Heapsort - Sort in descending order
6
7
12
10
9
14
19
13
2
7
9
12
10
13
14
19
6
2
5/12/2017 12:51 PM
Priority Queues
44

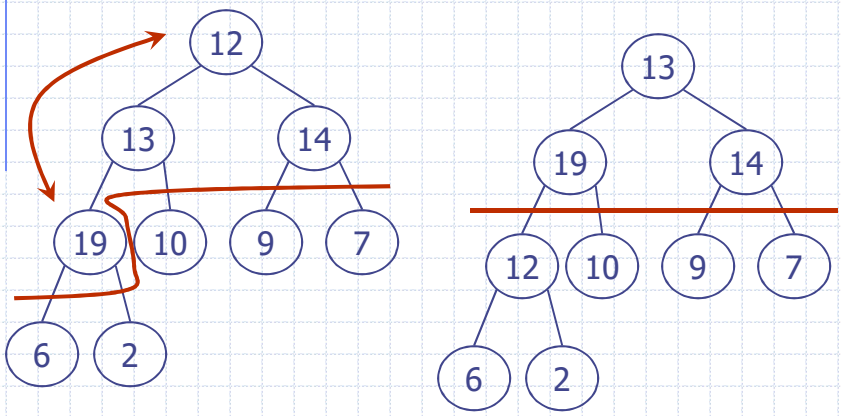

Heapsort - Sort in descending order
7
9
12
10
13
14
19
6
2
9
10
12
19
13
14
7
6
2
5/12/2017 12:51 PM
Priority Queues
45


Heapsort - Sort in descending order
9
10
12
19
13
14
7
6
2
10
13
12
19
14
9
7
6
2
5/12/2017 12:51 PM
Priority Queues
46

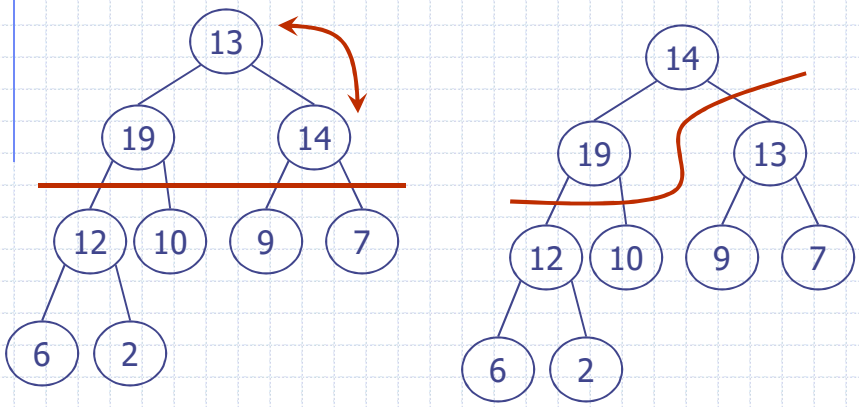
Heapsort - Sort in descending order



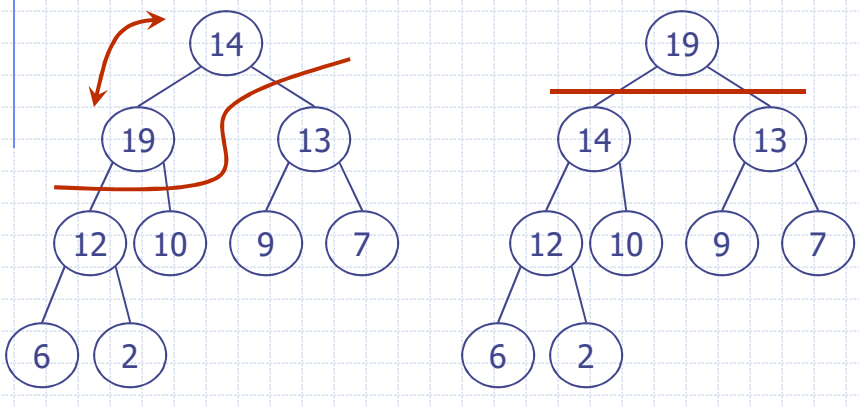
Heapsort - Sort in descending order

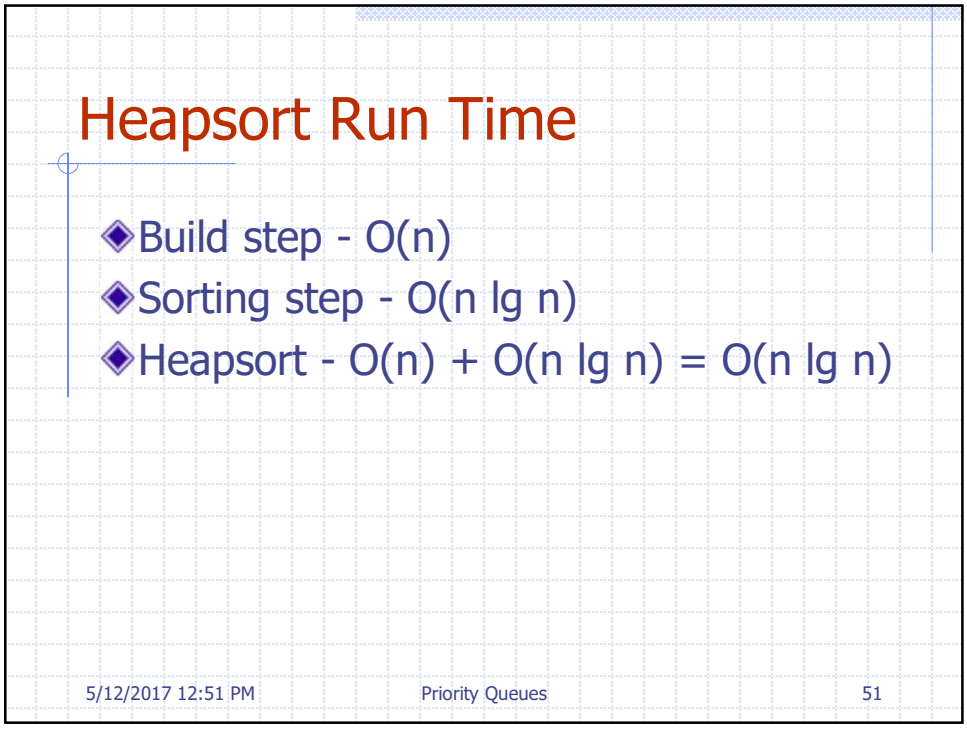


Heapsort - Sort in descending order



Heapsort - Sort in descending order




Heapsort Run Time
Build step - O(n)
Sorting step - O(n lg n)
Heapsort - O(n) + O(n lg n) = O(n lg n)
5/12/2017 12:51 PM
Priority Queues
51