

Priority Queues

11/7/04 02:04

Priority Queues

1

Priority Queue Operations

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

11/7/04 02:04

Priority Queues

2

Priority Queue Operations

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

11/7/04 02:04

Priority Queues

3

Priority Queue Implementations

- ◆ In class exercise - Evaluate and choose the best ADT implementation

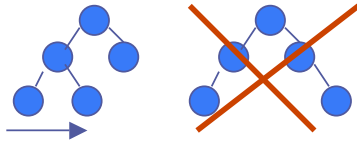
11/7/04 02:04

Priority Queues

4

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 full tree
 - Bottom level may not be filled but fills from left to right

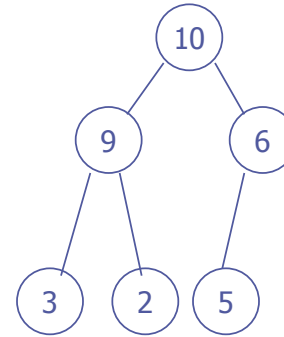


11/7/04 02:04

Priority Queues

5

Max Heap



10
9
6
3
2
5

11/7/04 02:04

Priority Queues

6

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 = $2i - 1$

11/7/04 02:04

Priority Queues

7

Heap Ordering Properties

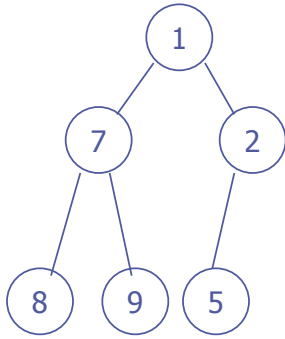
- ◆ Min Heap
 - For each node X , $X.item > (X.parent).item$
 - ◆ Smallest number is at the root
- ◆ Max Heap
 - For each node X , $X.item < (X.parent).item$
 - ◆ Largest number is at the root

11/7/04 02:04

Priority Queues

8

Min Heap



1
7
2
8
9
5

11/7/04 02:04

Priority Queues

9

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 in needed)*

11/7/04 02:04

Priority Queues

10

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 with violation of the heap property, insert item
- ◆ Otherwise, copy the hole's parent item into the hole then trickle up

11/7/04 02:04

Priority Queues

11

Min Heap Enqueue

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

Enqueue 13



Enqueue 16



11/7/04 02:04

Priority Queues

12

Min Heap Enqueue

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

Enqueue 19



11/7/04 02:04

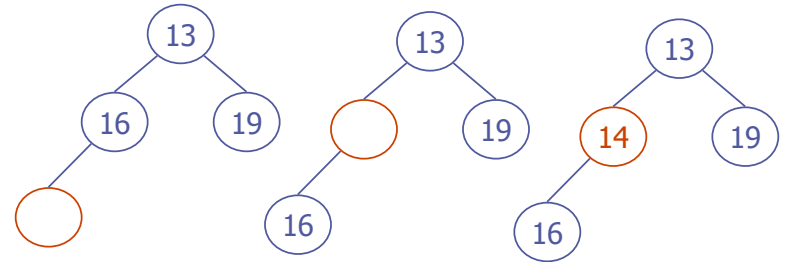
Priority Queues

13

Min Heap Enqueue

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

Enqueue 14



11/7/04 02:04

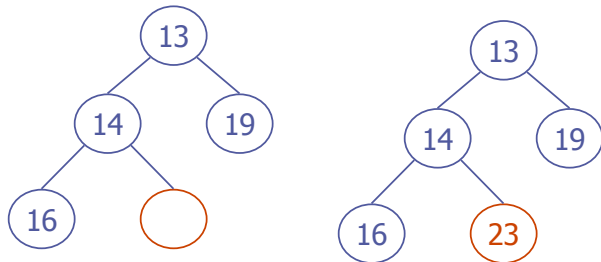
Priority Queues

14

Min Heap Enqueue

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

Enqueue 23



11/7/04 02:04

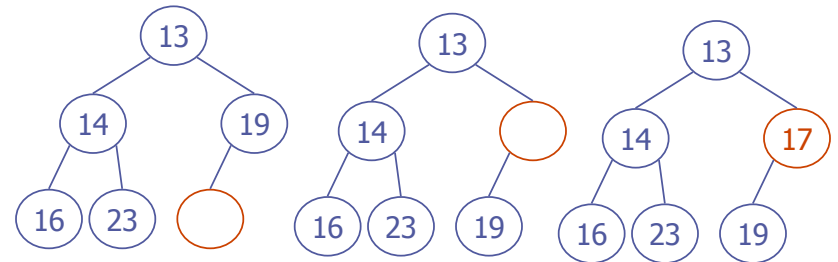
Priority Queues

15

Min Heap Enqueue

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

Enqueue 17



11/7/04 02:04

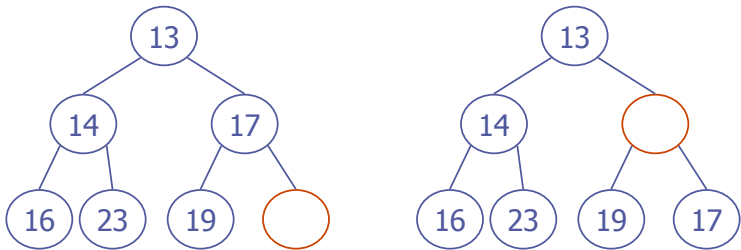
Priority Queues

16

Min Heap Enqueue

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

Enqueue 6



11/7/04 02:04

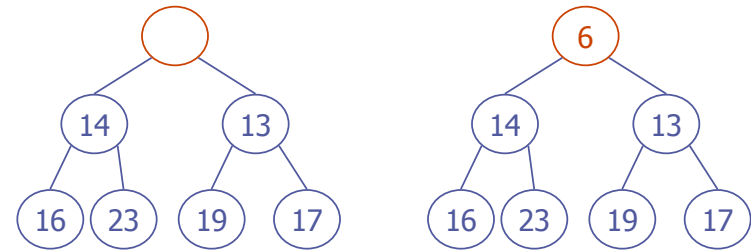
Priority Queues

17

Min Heap Enqueue

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

Enqueue 6



11/7/04 02:04

Priority Queues

18

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

11/7/04 02:04

Priority Queues

19

Max Heap Enqueue

1, 2, 3, 4, 5, 6

11/7/04 02:04

Priority Queues

20

Max Heap Enqueue

1, 2, 3, 4, 5, 6

11/7/04 02:04

Priority Queues

21

Max Heap Enqueue

1, 2, 3, 4, 5, 6

11/7/04 02:04

Priority Queues

22

Max Heap Enqueue

1, 2, 3, 4, 5, 6

11/7/04 02:04

Priority Queues

23

Max Heap Enqueue

1, 2, 3, 4, 5, 6

11/7/04 02:04

Priority Queues

24

Min Heap Enqueue

```
void enqueueHeap ( itemtype item )
  if heap not full
    if heap empty           Running time =
      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++
```

11/7/04 02:04

Priority Queues

25

Heap

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

11/7/04 02:04

Priority Queues

26

Heap Dequeue

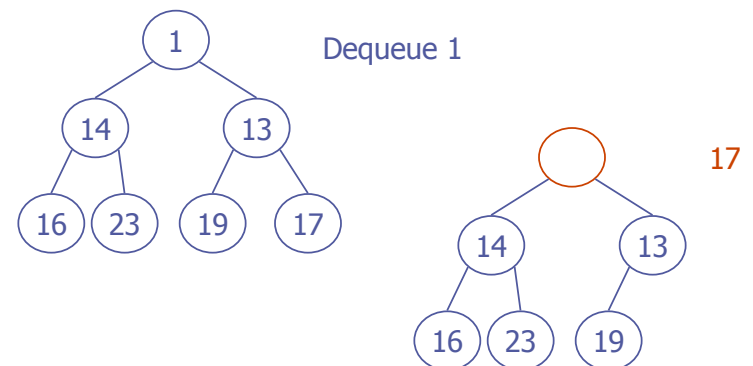
- ◆ Dequeue highest priority item
 - Makes a hole at the root
 - Want to remain a complete tree so place last item in the heap into the hole
 - ◆ If item can be placed in hole without violation of the heap property, then done
 - ◆ Otherwise, trickle down. Put the smaller of the holes children into the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

11/7/04 02:04

Priority Queues

27

Min Heap Dequeue

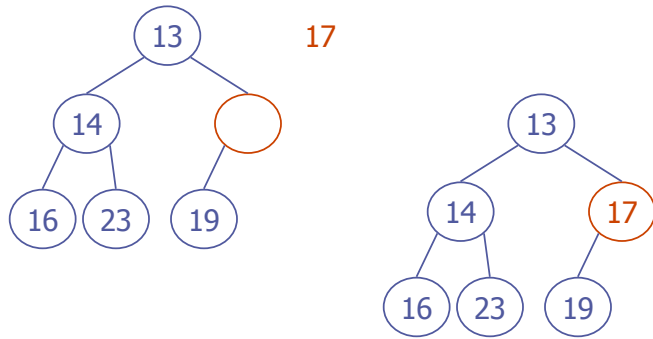


11/7/04 02:04

Priority Queues

28

Min Heap Dequeue

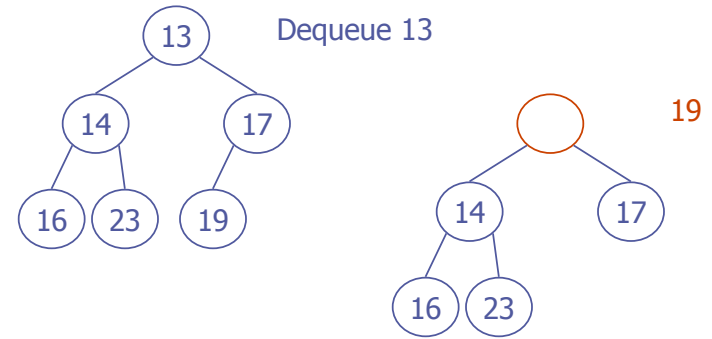


11/7/04 02:04

Priority Queues

29

Min Heap Dequeue

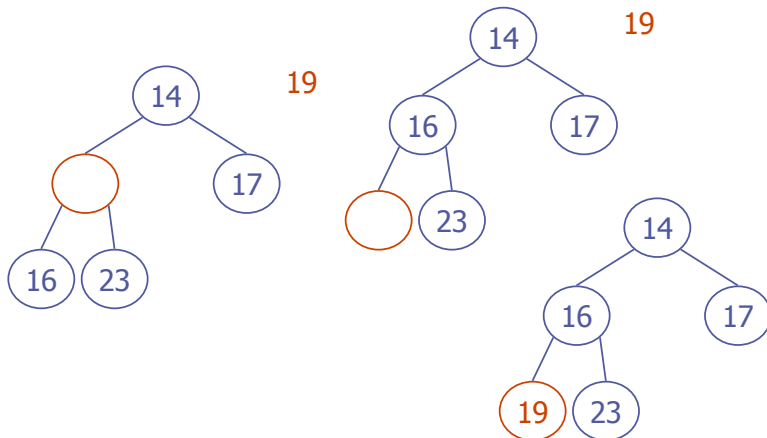


11/7/04 02:04

Priority Queues

30

Min Heap Dequeue

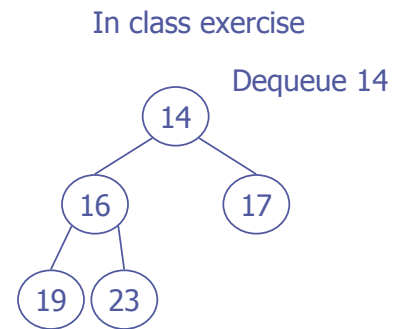


11/7/04 02:04

Priority Queues

31

Min Heap Dequeue



11/7/04 02:04

Priority Queues

32

Heap Notes

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

11/7/04 02:04

Priority Queues

33

Building a Heap

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

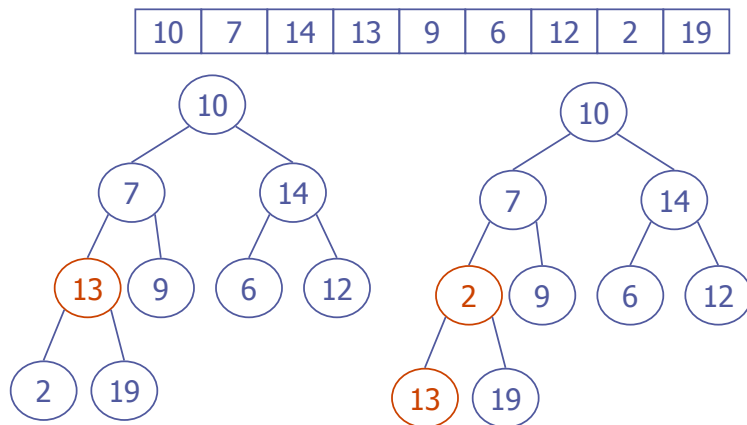
for ($x = (N/2) - 1$; $x \geq 0$; $x++$)
Trickle down

11/7/04 02:04

Priority Queues

34

Building a Min Heap

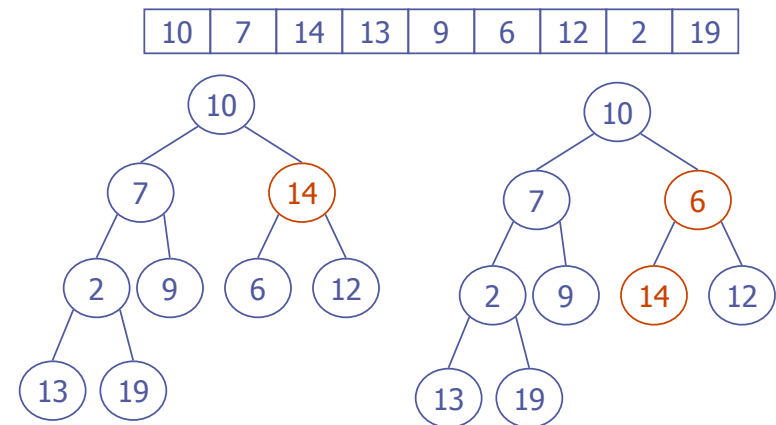


11/7/04 02:04

Priority Queues

35

Building a Min Heap

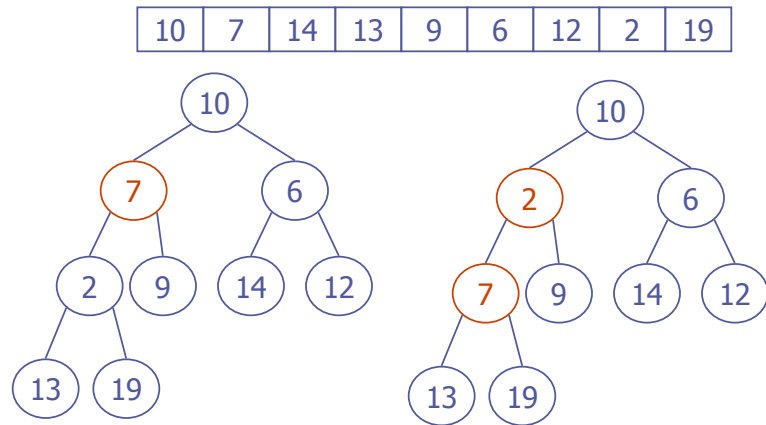


11/7/04 02:04

Priority Queues

36

Building a Min Heap

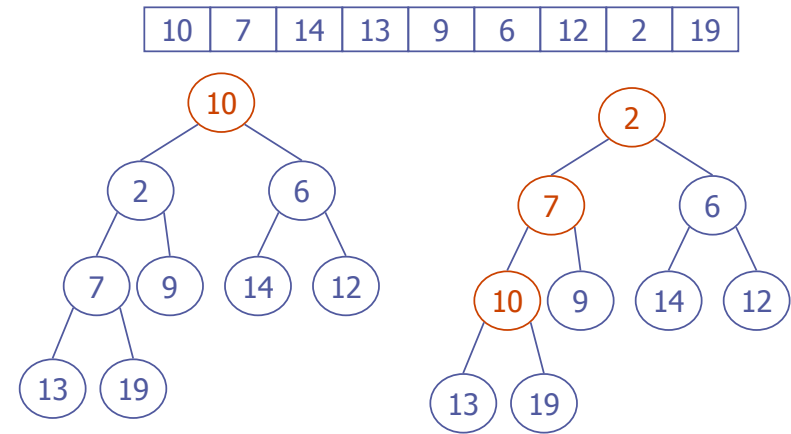


11/7/04 02:04

Priority Queues

37

Building a Min Heap



11/7/04 02:04

Priority Queues

38

Building a Heap

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

11/7/04 02:04

Priority Queues

39

Heapsort

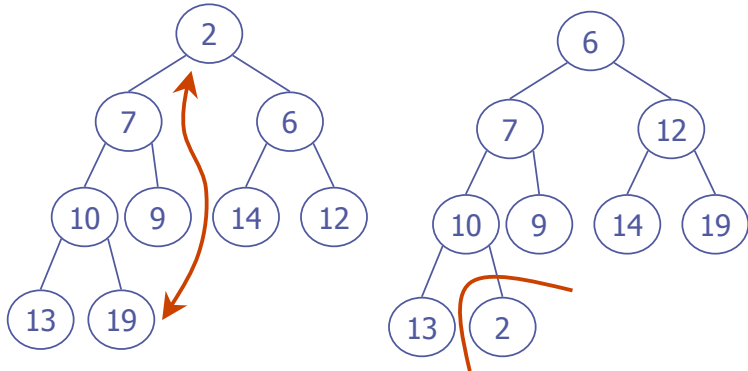
- ◆ Build heap for the opposite of what you want
 - Max heap for ascending order
 - Min heap for descending order
- ◆ Take root and place in last array position, then think of array as 1 smaller
- ◆ Trickle down from root to rebuild heap
- ◆ Continue until all items are moved

11/7/04 02:04

Priority Queues

40

Heapsort - Sort in descending order

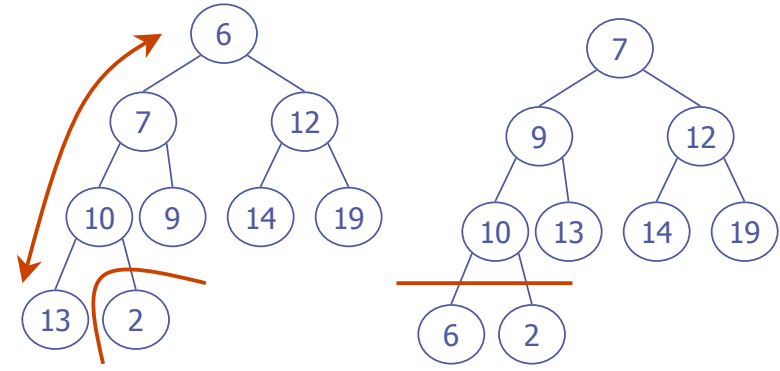


11/7/04 02:04

Priority Queues

41

Heapsort - Sort in descending order

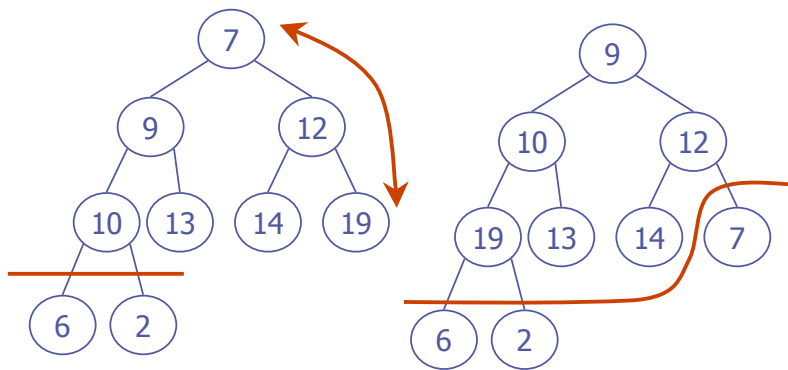


11/7/04 02:04

Priority Queues

42

Heapsort - Sort in descending order

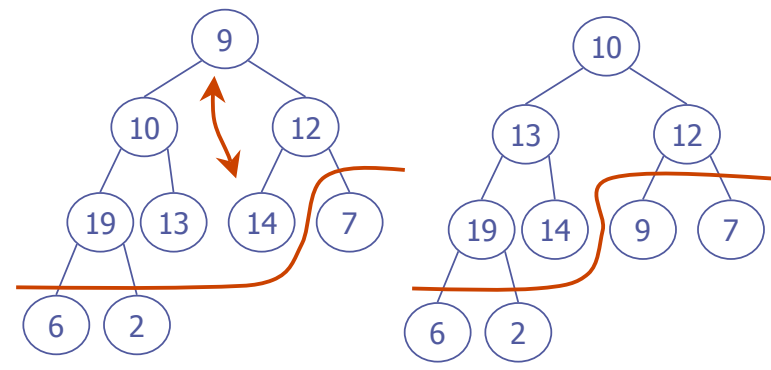


11/7/04 02:04

Priority Queues

43

Heapsort - Sort in descending order

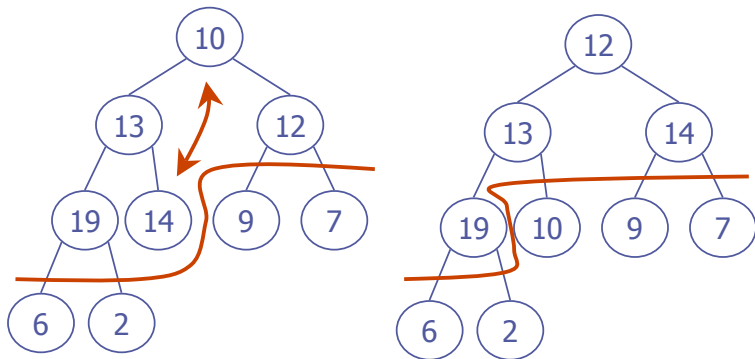


11/7/04 02:04

Priority Queues

44

Heapsort - Sort in descending order

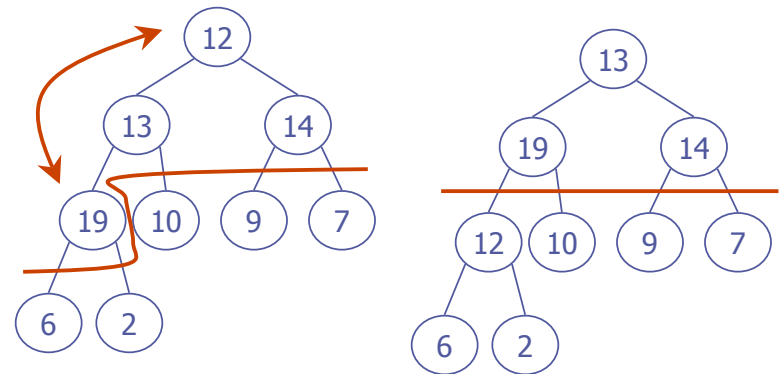


11/7/04 02:04

Priority Queues

45

Heapsort - Sort in descending order

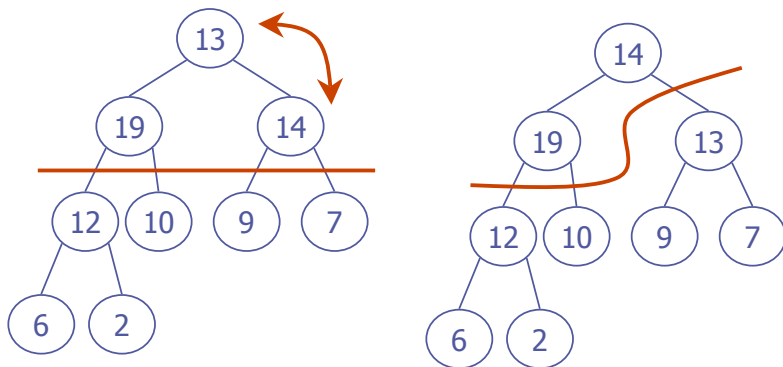


11/7/04 02:04

Priority Queues

46

Heapsort - Sort in descending order

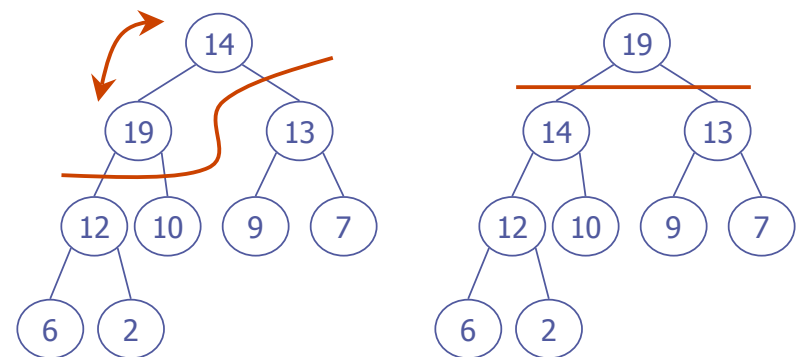


11/7/04 02:04

Priority Queues

47

Heapsort - Sort in descending order



11/7/04 02:04

Priority Queues

48

Heapsort Run Time

- ◆ Build step - $O(n)$
- ◆ Sorting step - $O(n \lg n)$
- ◆ Heapsort - $O(n) + O(n \lg n) = O(n \lg n)$