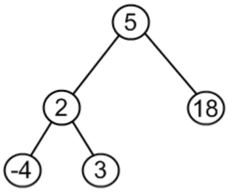
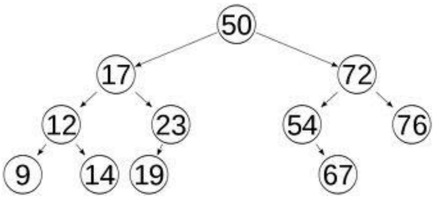
CSC115 Lecture 19

Heaps

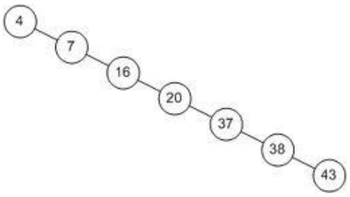
First:
Remember Binary Search Trees

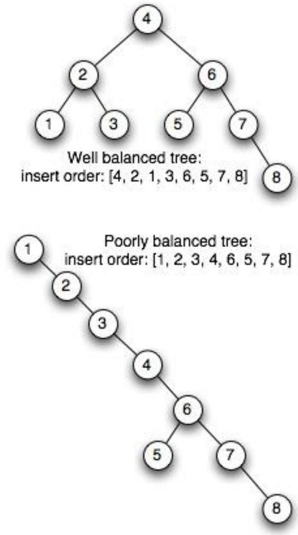
- For each node n:
  - n’s value is greater than all values in its left subtree T_L
  - n’s value is less than all values in its right subtree T_R
  - Both T_L and T_R are binary search trees





Problem with Binary Search Trees



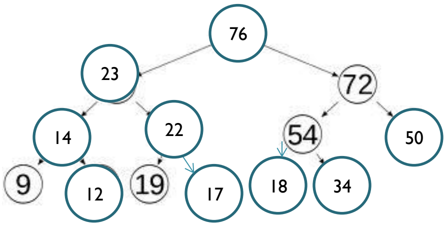

4
2
6
1
3
5
7
8
Well balanced tree:
insert order: [4, 2, 1, 3, 6, 5, 7, 8]
Poorly balanced tree:
insert order: [1, 2, 3, 4, 6, 5, 7, 8]
1
2
3
4
6
5
7
8

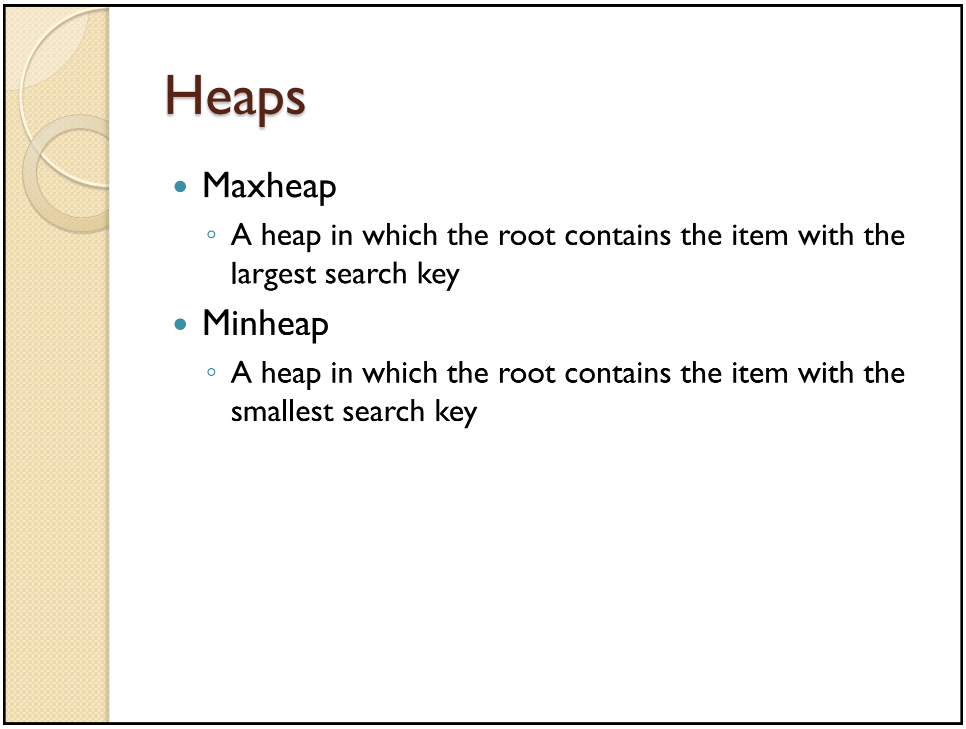
Heaps

- A heap is a complete binary tree
  - That is empty

  or
  - Whose root contains a search key greater than or equal to the search key in each of its children, and
  - Whose root has heaps as its subtrees

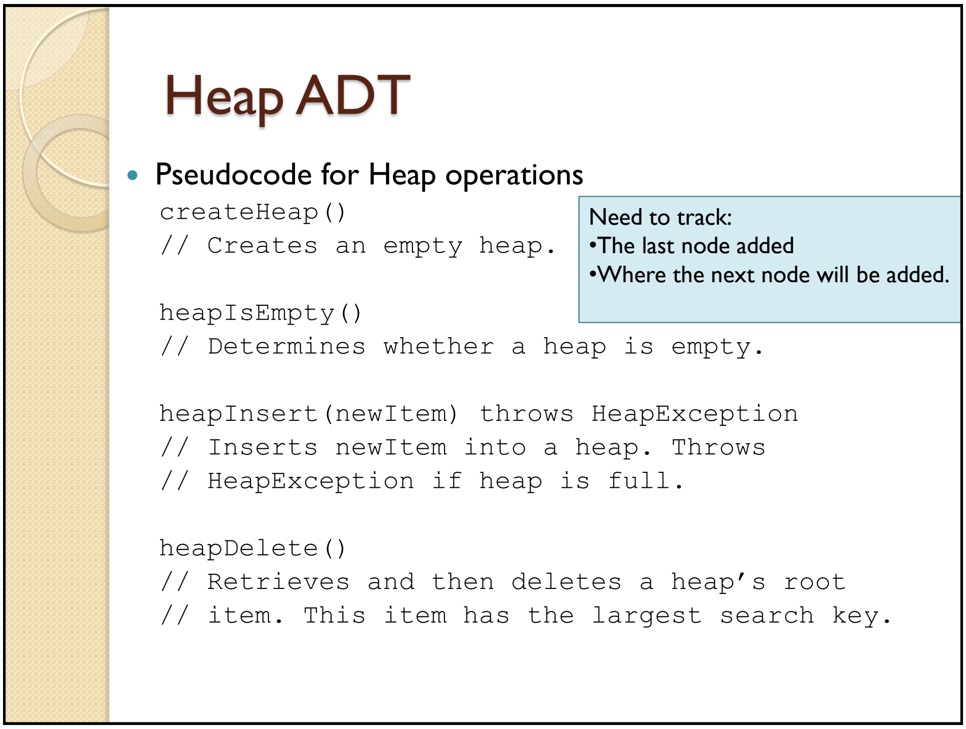
Heap





Heaps

- Maxheap
  - A heap in which the root contains the item with the largest search key
- Minheap
  - A heap in which the root contains the item with the smallest search key



Heap ADT

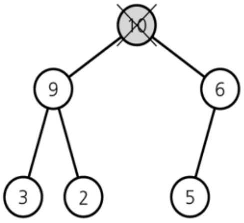
- Pseudocode for Heap operations

```
createHeap()
// Creates an empty heap.

heapIsEmpty()
// Determines whether a heap is empty.

heapInsert(newItem) throws HeapException
// Inserts newItem into a heap. Throws
// HeapException if heap is full.

heapDelete()
// Retrieves and then deletes a heap’s root
// item. This item has the largest search key.
```

Need to track:
•The last node added
•Where the next node will be added.

Heaps: `heapDelete`

- Step 1: Return the item in the root
  - Results in disjoint heaps

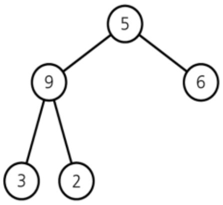


(a)

a) Disjoint heaps

Heaps: `heapDelete`

- Step 2: Copy the item from the last node into the root
  - Results in a semiheap

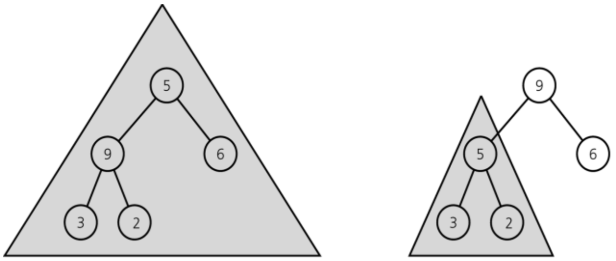


(b)

b) a semiheap

Heaps: heapDelete

- Step 3: Transform the semiheap back into a heap
  - Performed by the recursive algorithm heapRebuild

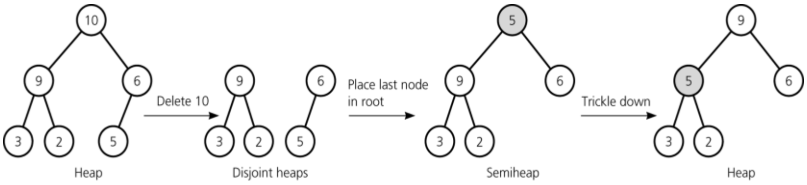


First semiheap passed to **heapRebuild** Second semiheap passed to **heapRebuild**

Recursive calls to ***heapRebuild***

Heaps: heapDelete

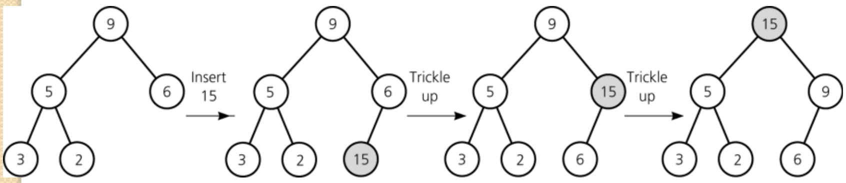
- Efficiency
  - heapDelete is O(log n)



Deletion from a heap

Heaps: `heapInsert`

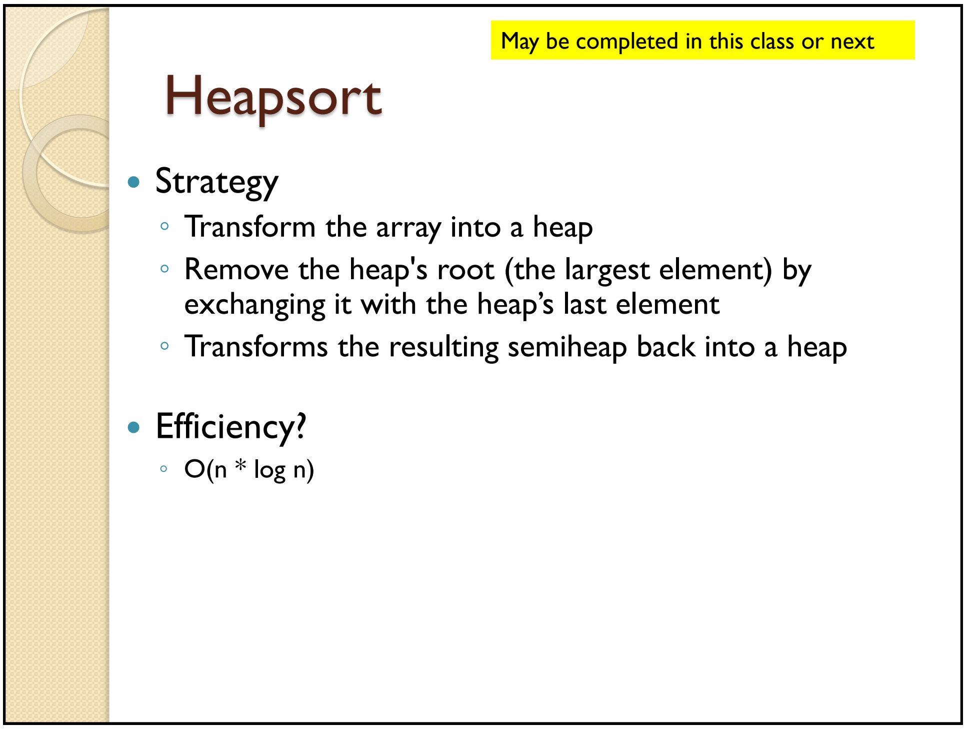
- Strategy
  - Insert `newItem` into the bottom of the tree
  - Trickle new item up to appropriate spot in the tree
- Efficiency: O(log n)



Insertion into a heap

A Few More Examples

- Will be done in class



May be completed in this class or next

Heapsort

- Strategy
  - Transform the array into a heap
  - Remove the heap's root (the largest element) by exchanging it with the heap’s last element
  - Transforms the resulting semiheap back into a heap
- Efficiency?
  - O(n * log n)