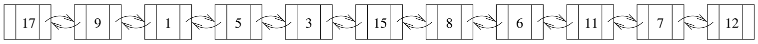
Binary Search Trees: Data Structures for Ordered Sets

(CLRS 12.1-12.3)

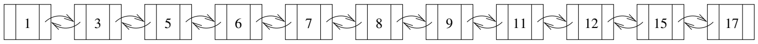
- At a high level, we have a set of elements S and we want to represent S with a data structure such that the following operations are supported efficiently:
  - Search(e): Return (pointer to) element e in S (if $e \in S$)
  - Insert(e): Insert element e in S
  - Delete(e): Delete element e from S
  - Successor(e): Return (pointer to) minimal element in S larger than e
  - Predecessor(e): Return (pointer to) maximal element in S smaller than e
- The first implementation that comes to mind is the ordered array:

1	3	5	6	7	8	9	11	12	15	17

  - Search can be performed in $O(n)$ time by scanning through array or in $O(\log n)$ time using binary search
  - Predecessor/Successor can be performed in $O(\log n)$ time like searching
  - Insert/Delete takes $O(n)$ time since we need to expand/compress the array after finding the position of e
- Perhaps an unordered list?



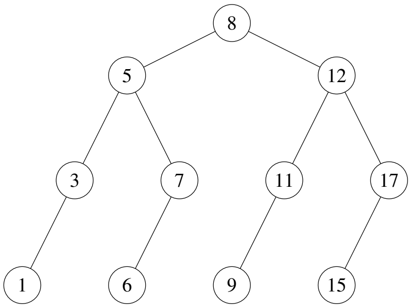
  - Search takes $O(n)$ time since we have to scan the list
  - Predecessor/Successor takes $O(n)$ time
  - Insert takes $O(1)$ time since we can just insert e at beginning of list
  - Delete takes $O(n)$ time since we have to perform a search before spending $O(1)$ time on deletion
- How about ... an ordered list ?



  - Search takes $O(n)$ time since we cannot perform binary search
  - Predecessor/Successor takes $O(n)$ time
  - Insert/Delete takes $O(n)$ time since we have to perform a search to locate the position of insertion/deletion

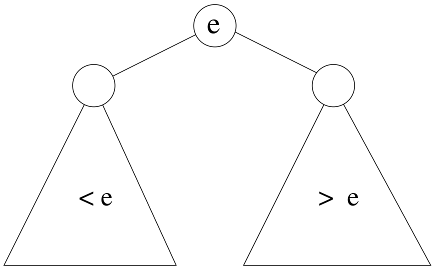
Binary search tree implementation

- We want to combine the advantages of sorted arrays (fast search) with the advantages of lists (fast insert and delete)
- Binary search naturally leads to definition of binary search tree



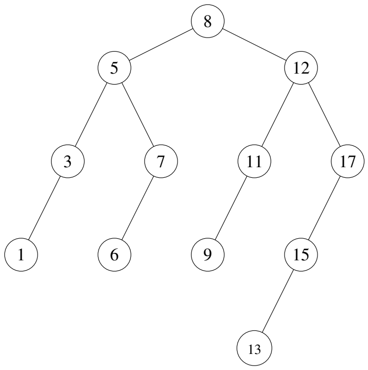
- Formal definition of search tree:

  - Binary tree with elements in nodes
  - If node v holds element e then
    - All elements in left subtree $\leq e$
    - All elements in right subtree $> e$



- *Search*(e) in $O(height)$: Compare with e and recursively search in left or right subtree
- *Insert*(e) in $O(height)$: Search for e and insert at place where search path terminates (Note: height may increase)

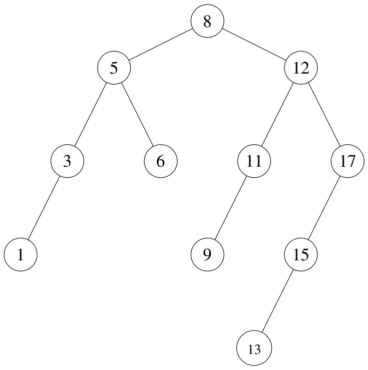
  Example: Insertion of 13



- *Delete*(e) in $O(height)$: Search for node v containing e,

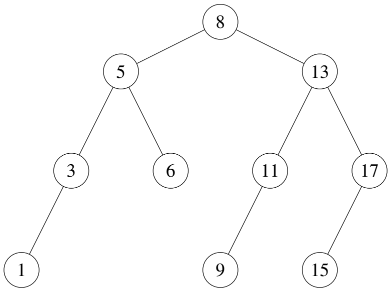
  1. v is a leaf: Delete v
  2. v is internal node with one child: Delete v and attach $child(v)$ to $parent(v)$

     Example: Delete 7

  3. v is internal node with two children:
     - exchange e in v with successor e' in node v' (minimal element in right subtree, found by following left branches as long as possible in right subtree)
     - v' node can be deleted by case 1 or 2

       Example: Delete 12

- Class work: How do you find the successor/predecessor of a given node?
- Class work: Assume you have a BST of n elements. How long does it take to sort them?
- Note:
  - Running time of all operations depend on height of tree.
  - Intuitively the tree will be nicely balanced if we do insertion and deletion randomly.
  - In worst case the height can be O(n).