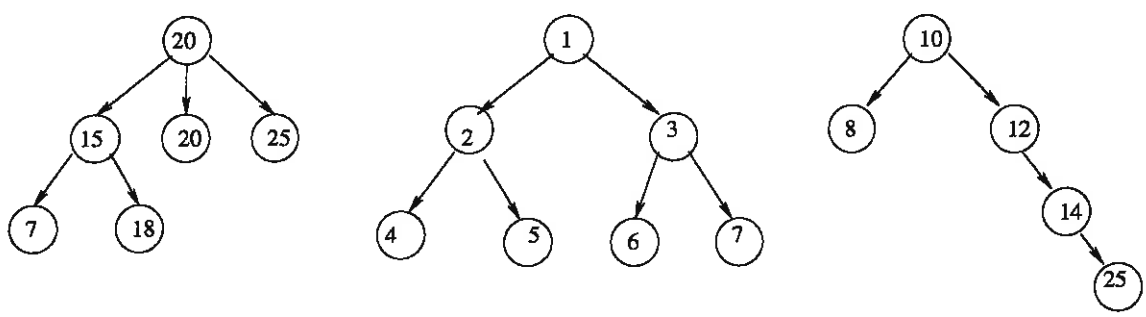
EECS 268 Home-Work 7 – Fall 2012

Do not write you name on this answer sheet (only your KU-ID). Total: 55 points

Name ~~KU-ID~~: Prasad kulkarni

1. State true or false: **(5 points)**
   - The inorder traversal of a binary search tree T will visit its nodes in sorted search-key order. True
   - The minimum height of a binary tree with n nodes is $log_2(n+1)$. True
   - The average case complexity of insertion in a binary search tree is $O(log_2 n)$. True
   - Updating the *key* item of a Table node is not permitted. True
   - A *heap* can become unbalanced with insertions and deletions over time. False

2. Given the following tree categories: (a) binary tree, (b) complete binary tree, (c) full binary tree, (d) general tree, and (e) binary search tree,
   Apply all the applicable categories to the following trees. **(5 points)**

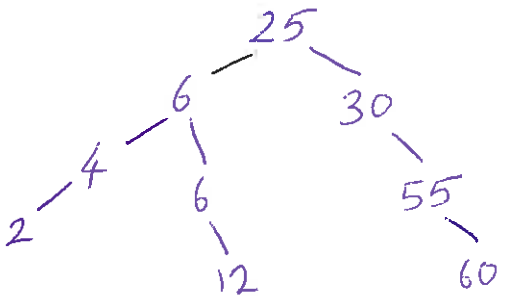


(d) general tree

(a) binary tree
(b) complete binary tree
(c) full binary tree
(d) general tree

(a) binary tree
(d) general tree
(e) binary search tree

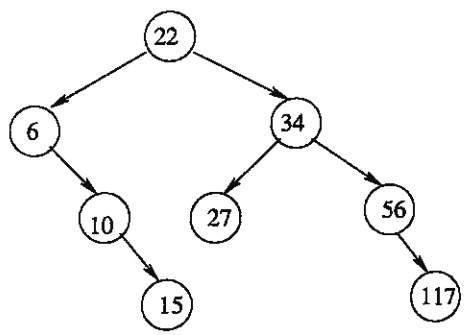
3. Arrange the following nodes in a binary search tree form in the order given: 25, 6, 4, 30, 2, 55, 6, 60, 12. **(4 points)**
Is the tree balanced? Why or why not? **(2 points)**
Print out the nodes in (a) pre-order, (b) post-order, (c) in-order. **(3 points)**



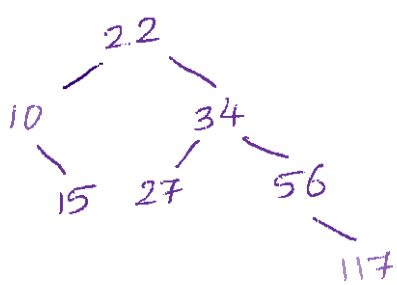
Tree is not balanced, because there are nodes in the tree where the height of its subtrees differ by one than one.
eg. Node 30 : height of left subtree = 0
height of right subtree = 2

Pre-order: 25 6 4 2 6 12 30 55 60
In-order: 2 4 6 6 12 25 30 55 60
Post-order: ~~2 4 6 12 6 30 55 60 25~~
2 4 12 6 6 60 55 30 25

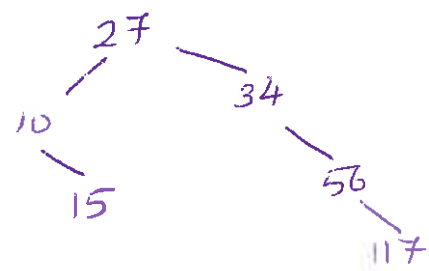
4. Given the following binary search tree, illustrate the binary tree structure after *each* of the following node deletions in the order: 6, 22, 27. **(5 points)**



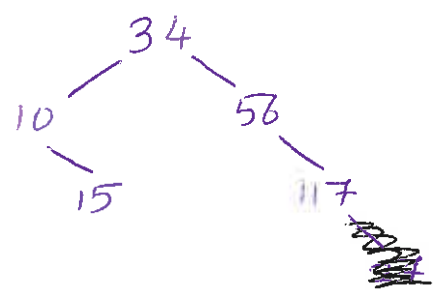
delete 6



delete 22 ⟹



⇓ delete 27



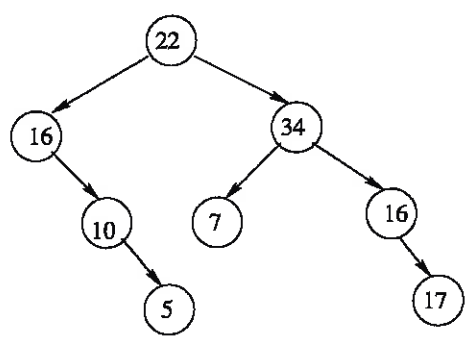
5. Given a pointer-based representation of binary tree (as on class slides 30–47), write an algorithm to deallocate all nodes in the tree. The *TreeNode* structure is as below. **(5 points)**

```
class TreeNode {
private:
    TreeNode() {};
    TreeNode(const int& nodeItem, TreeNode *left = NULL,
             TreeNode *right = NULL): item(nodeItem),
             leftChildPtr(left), rightChildPtr(right) {}

    int item;                  // data portion
    TreeNode *leftChildPtr;    // pointer to left child
    TreeNode *rightChildPtr;   // pointer to right child
    friend class BinaryTree;   // friend class
};
```

```
void BinaryTree :: deallocateTree (TreeNode *& treePtr)
{
    // Post-order traversal
    if (treePtr != NULL) {
        deallocateTree (treePtr -> leftChildPtr);
        deallocateTree (treePtr -> rightChildPtr);
        delete treePtr;
    } treePtr = NULL;
}
```

6. Illustrate (with figure) the array representation of the following binary tree. **(5 points)**



$\#nodes = 2^h - 1$
$= 2^4 - 1 = 15$

left-child $= 2i + 1$
right-child $= 2i + 2$ } for node i

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
22	16	34	—	10	7	16	—	—	—	5	—	—	—	17

7. Explain the Table implementation you will choose in the following cases (sorted/unsorted array/LL, BST). Why? **(4 points)**

(a) English dictionary; only needs a retrieval operation.

✓ Sorted array implementation

— both sorted array and BST provide an avg. case efficiency for retrieval of $O(\log_2 n)$

But sorted array is easier to implement and more space efficient

(b) Word processor spell checker; compares words in your document with words in its dictionary; can add new words to its dictionary when necessary; needs frequent retrieval and occasional insertion.

✓ Binary Search Tree is preferred.

Can also use sorted array if ease of implementation is an over-riding concern.

8. Why is a binary search tree not particularly suitable for implementing a priority queue? What data structure would you choose instead? Why? **(5 points)**

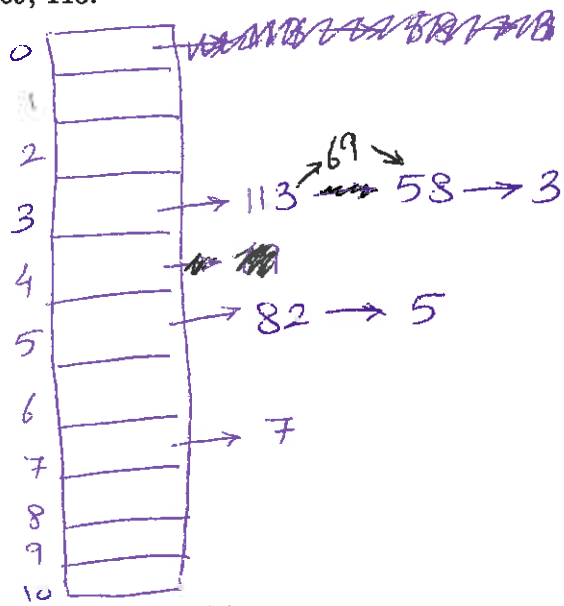
Although a BST provides $O(\log_2 n)$ time operations (insert, delete, search), in the worst-case, the complexity is $O(n)$.

The worst-case can happen if the tree is not balanced. BST cannot be guaranteed to be balanced. Keeping a BST balanced is a complex operation.

Priority-queue has certain restrictions on its semantics, or the set of operations it allows. More precisely, items can only be removed in ascending or descending order. This restriction makes the "heap" datastructure more suitable, since the more "relaxed" semantics of heap allow it be maintain balance relatively easily.

9. Use open hashing and the hash-function: h(x) = x mod 11 to insert the following elements in the hash-table: 3, 5, 7, 82, 58, 69, 113.
(3 points)

$3 \% 11 = 3$
$5 \% 11 = 5$
$7 \% 11 = 7$
$82 \% 11 = 5$
$58 \% 11 = 3$
$69 \% 11 = 3$
$113 \% 11 = 3$



10. Use closed hashing to and the hash-function: h(x) = x mod 11 to insert the following elements in the hash-table: 3, 5, 7, 82, 58, 69, 113.
(4 points)
Use the following collision resolution techniques (one at a time): (a) Linear probing (b) Quadratic probing

Linear probing

(a)

0	
1	
2	
3	3
4	58
5	5
6	82
7	7
8	69
9	113
10	

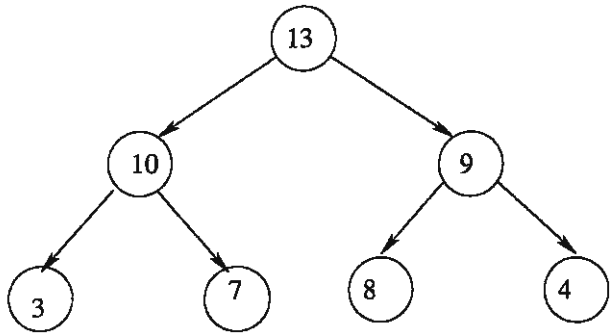
82 : 5 → 6
58 : 3 → 4
69 : 3 → 4 → 5 → 6 → 7 → 8
113 : 3 → 4 → 5 → 6 → 7 → 8 → 9

(b) Quadratic probing: $h_i(x) = (h_0(x) + i^2) \bmod m$

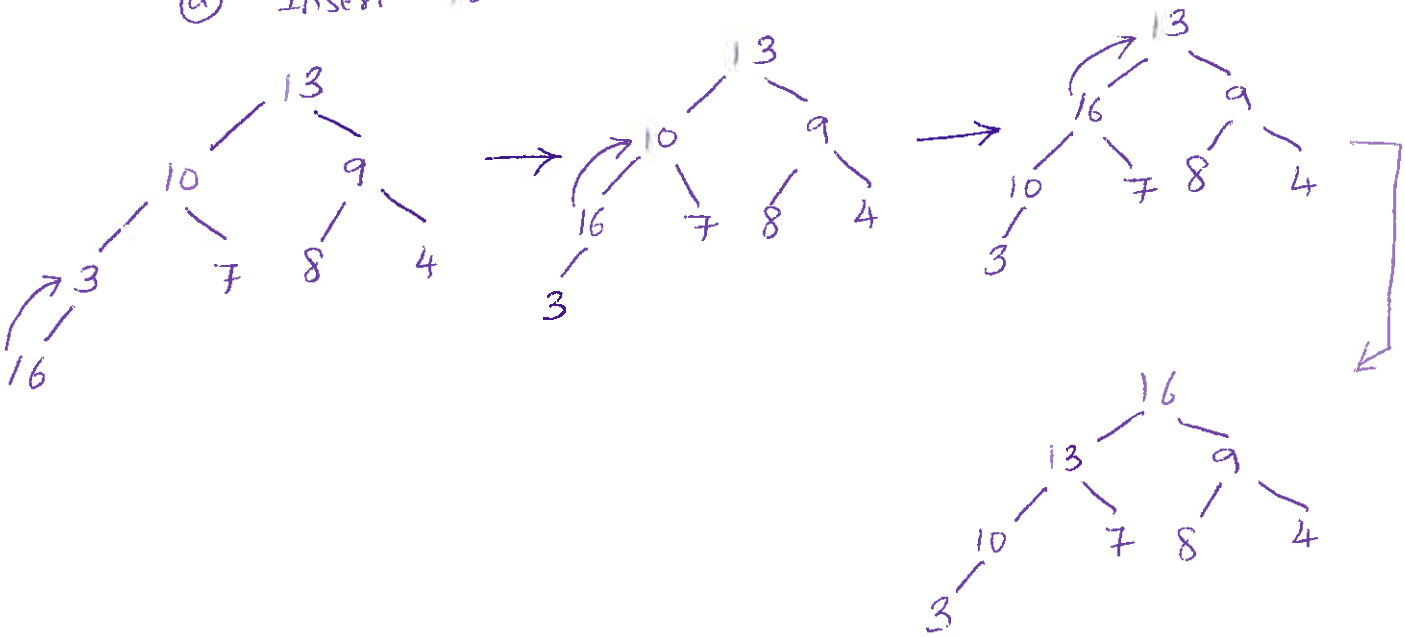
0	
1	69
2	
3	3
4	58
5	5
6	82
7	7
8	113
9	
10	

82 : 5 → 6
58 : 3 → 4
69 : 3 → 4 → 7 → 1
113 : 3 → 4 → 7 → 1 → 8

11. Given the following max-heap, illustrate the *individual* steps in which the heap changes for each for the following operations: (a) Insert 16, (b) delete. **(5 points)**



(a) Insert 16



(b) Delete