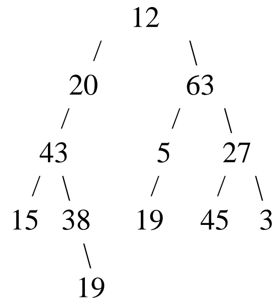
COP 3502 Quiz #4 Version A (Binary Search Trees, AVL Trees, Tries, Heaps, Hash Tables) Solutions

1) (6 pts) Provide the Preorder, Inorder and Postorder traversals of the following binary tree:

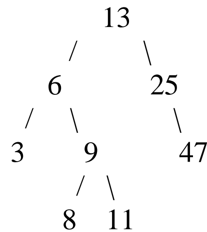


Pre-Order: **12, 20, 43, 15, 38, 19, 63, 5, 19, 27, 45, 3** **(Grading: 2 pts if correct, 1 pt if mostly correct, per traversal, 0 pts if 6 or fewer items correct)**

In-Order: **15, 43, 38, 19, 20, 12, 19, 5, 63, 45, 27, 3**

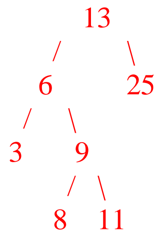
Post-Order: **15, 19, 38, 43, 20, 19, 5, 45, 3, 27, 63, 12**

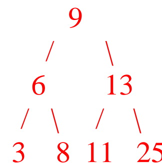
2) (5 pts) Show the result of deleting 47 from the **AVL** tree shown below.



Solution

Initial tree is, we rebalance at 13, putting 9 at root to get:





Grading: 1 pt valid BST (if not valid BST 0 out of 5 automatically)
2 pts valid AVL (just check balances)
1 pt 9 at root, 1 pt for everything else being correct

3) (10 pts) Consider building a trie where each trie node stores an added component called numWords. This integer would represent the number of nodes corresponding to the ends of valid words within that sub-trie. If we add this item to the struct, then it becomes possible to write a function that takes in the root of the trie and returns **the sum of the lengths of all the valid words in the trie.** For example, if the trie stored the words "act", "action", "after", and "bacon", then calling the function sumWordLengths with the root of the trie as the parameter should return 19 (the sum of the lengths of the 4 words). If we pass in the node corresponding to the prefix "a", the function should return 11, because within that subtree the nodes corresponding to valid words follow the paths "ct", "ction" and "fter". You may write the function recursively or iteratively. (I recommend recursively!)

Please use the struct definition and function prototype shown below:

```
typedef struct trie {
    int isWord;
    int numWords;
    struct trie* next[26];
} trie;

int sumWordLengths(trie* root) {

    // Grading: 2 pts.
    if (root == NULL) return 0;

    // Grading: 1 pt accumulator, 1 pt loop, 3 pts adding all
    //          rec calls.
    int res = 0, i;
    for (i=0; i<26; i++)
        res += sumWordLengths(root->next[i]);

    // 1 pt for res, 2 pts for what you add to it. (Give 1 of these
    //                pts if they say root->numWords only.)
    return res + (root->numWords - root->isWord);
}
```

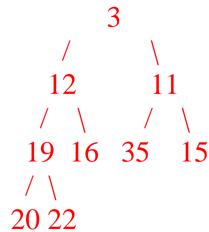
Note: There is probably another way to do this where you "cut off" words before you get to NULL, so be careful when grading. If you're not sure, plug their function into my posted code and check. There could also be an approach I didn't think of. This can definitely be done iteratively, but the code is longer and I think most students would have some trouble with that approach. (It would be like a BFS through the trie where you are storing depths and just adding the depth for each node where isWord is 1.)

4) (8 pts) Consider the binary heap represented by the array stored below. Draw the original heap as a tree and then show the result of removing the minimum item from the heap as another tree. Draw a square around the original heap and a circle around the heap AFTER the minimum item is removed.

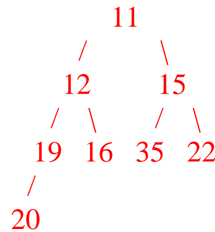
Index	1	2	3	4	5	6	7	8	9
Value	3	12	11	19	16	35	15	20	22

Solution

Original heap:



Delete min gets rid of 3, replaces with 22 and runs percolate down:



Grading: 2 pts original drawing.
2 pts 11 at root of second drawing.
1 pt for 12, 1 pt for 15, 1 pt for 22, 1 pt for res

5) (6 pts) Let a hash table, which uses quadratic probing to place its elements into the table, have a length of 127. If an element to insert has a hash value of 121, list the first 6 indexes into the array where we will look to insert the element AFTER index 121, in the case of collisions.

Solution

122, 125, 3, 10, 19, 30

Grading: 1 pt per slot, number has to be in the correct slot to get credit.