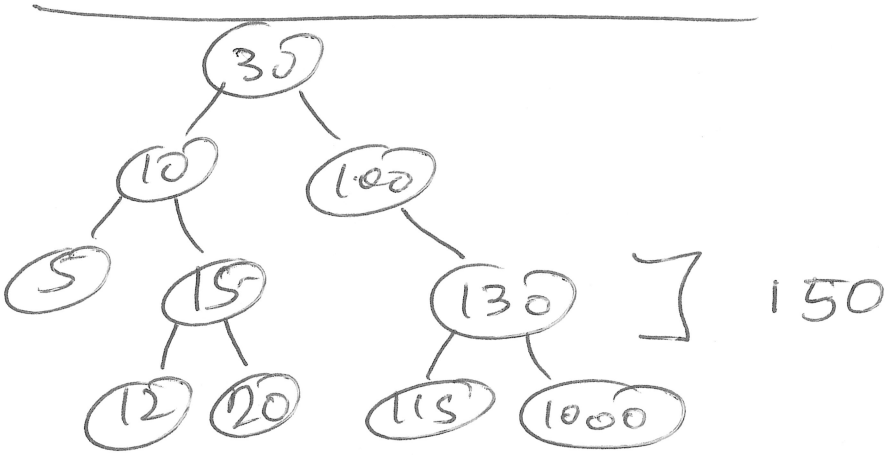
COP 3502 ≤ 10/27/2025

1) Sample Binary Tree Qs from Foundation Exam. (Tracing, Coding, Traversals)
2) Show how to practice via posted code.

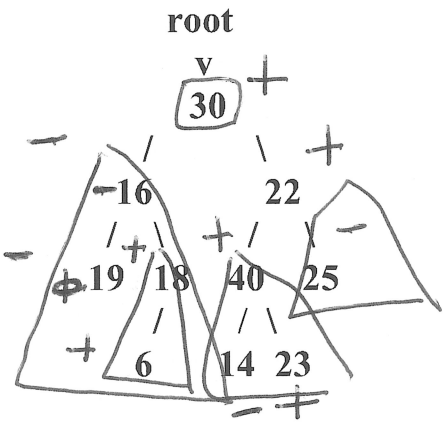
Sum of depth example



1) (10 pts) ALG (Binary Trees)

The following code, when passed the root of a binary tree and returns a result calculated by adding the values of some nodes and subtracting the values of others. If the function whatDoesItDo is called on the root of the binary tree shown below, for each value in the tree, indicate whether it gets added (A) or subtracted (S), **with respect to the original call on the root of the tree below.** No need to state the final return value. (**Grading Note:** +1 for each correct slot, +0 for each slot left blank, -1 for each incorrect slot, minimum score is 0.)

```
#include <stdio.h>
#include <stdlib.h>
typedef struct bintreenode {
    int data;
    struct bintreenode* left;
    struct bintreenode* right;
} bintreenode;

int whatDoesItDo(bintreenode* root) {
    if (root == NULL) return 0;
    if (root->left == NULL && root->right == NULL) return root->data;
    if (root->left == NULL) return root->data + whatDoesItDo(root->right);
    if (root->right == NULL) return root->data + whatDoesItDo(root->left);
    if (root->left->data > root->right->data)
        return root->data + whatDoesItDo(root->left) - whatDoesItDo(root->right);
    return root->data + whatDoesItDo(root->right) - whatDoesItDo(root->left);
}
```

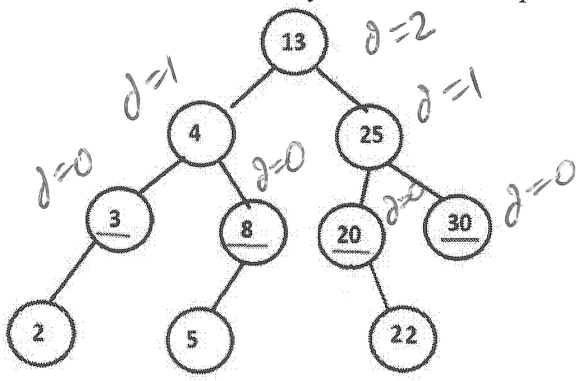


For each open slot, either write the letter 'A' for added, or 'S' for subtracted.

30	A	16	S	22	A
19	S	18	A	40	A
25	S	6	A	14	S
23	A				

1) (10 pts) DSN (Binary Trees)

Write a **recursive** function named `sumAtDepth` that takes a pointer to the root of a binary tree, `root`, and non-negative integer, `depth`, and returns the sum of all the values in the nodes that are at a level `depth` below the root. For example, if you pass the root of the following binary tree and `depth` = 2, the function should return 61 (= 3 + 8 + 20 + 30) since the each of the nodes storing 3, 8, 20 and 30 are 2 levels below the root node of the tree. You may assume that depth is a non-negative integer.



You must write your solution in a **single** function. You cannot write any helper functions.

The function signature and node struct are given below.

```
typedef struct node
{
   int data;
   struct node *left;
   struct node *right;
} node;

int sumAtDepth(node *root, int depth) {
```

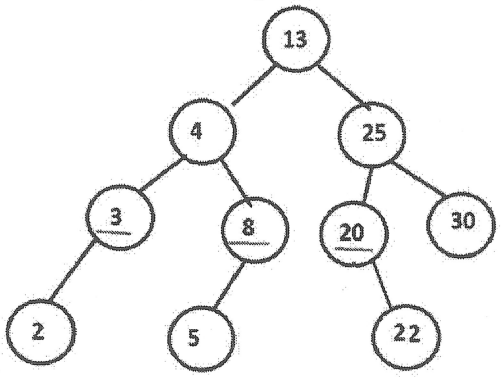
```
if (root == NULL) return 0;
if (depth == 0) return root->data;
return sumAtDepth(root->left, depth-1) +
       sumAtDepth(root->right, depth-1);
```

```
}
```

1) (10 pts) DSN (Binary Trees)

Write a function named *sumSingleParents*() that takes a pointer to the root of a binary tree (*root*) and returns the sum of all the values in the nodes with a single child.

For example, if you pass the root of the following binary tree, the function should return 31 (=3+8+20) as the nodes containing 3, 8, and 20 have only one child:



You must write your solution in a **single** function. You cannot write any helper functions.

The function signature and node struct are given below.

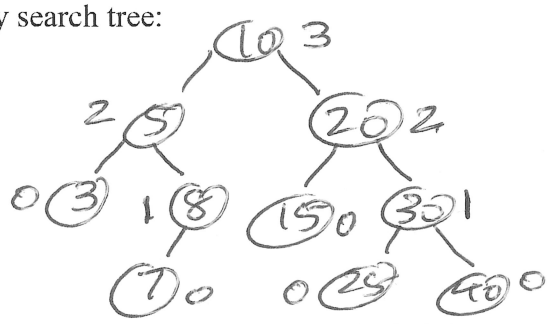
```
typedef struct node
{
   int data;
   struct node *left;
   struct node *right;
} node;

int sumSingleParents(node *root) {
      if   (root==NULL) return 0;
      if (root->left != NULL && root->right != NULL)
         return sumSingleParents(root->left) + sumSingleParents(root->right);
      if (root->left == NULL && root->right != NULL)
         return root->data + sumSingleParents(root->right);
      if (root->left != NULL && root->right == NULL)
         return root->data + sumSingleParents(root->left);
      return 0;
}
```

Prob can be 2 case instead of 3

1) (10 pts) DSN (Binary Trees)

Consider using the following struct definition for a node of a binary search tree:

```
typedef struct node {
    int data;
    int height;
    struct node* left;
    struct node* right;
} node;
```



Assume that a binary search tree has been built with the data values in each struct filled in, but the heights are uninitialized. Write a **void recursive** function, `assignHeights`, **with no helper** functions, which takes in a pointer, `root`, to the root of a binary search tree, and assigns the height component of each node in the subtree pointed to by `root` to its correct height in the tree. Recall that the height of a leaf node is 0, and that more generally, the height of a node is the maximum number of links (left or right) to follow from that node to any leaf node in that subtree. (If root is NULL, then no action should be taken.)

```
void assignHeights(node* root) {
    if (root == NULL) return;
    if (root->left == NULL && root->right == NULL) {
        root->height = 0;
        return;
    }
    int leftH = -1;
    if (root->left != NULL) {
        assignHeights(root->left);
        leftH = root->left->height;
    }
    int rightH = -1;
    if (root->right != NULL) {
        assignHeight(root->right);
        rightH = root->right->height;
    }
    if (leftH > rightH)
        root->height = leftH+1;
    else
        root->height = rightH+1;
}
```

1) (10 pts) DSN (Binary Trees)

Demonstrate your understanding of recursion by rewriting the following recursive function, which operates on a binary tree, as an **iterative** function. In doing so, you must abide by the following restrictions:

1. Do not write any helper functions in your solution.
2. Do not make any recursive calls in your solution.

```
int foo(node *root) {
    if (root == NULL) return 1;
    if (root->left == NULL && root->right == NULL) return 2;
    if (root->left == NULL) return 3 * foo(root->right);
    if (root->right == NULL) return 4 * foo(root->left);
    if (root->right->data > root->left->data) return 5 * foo(root->right);
    return 6 * foo(root->left);
}

int iterative_foo(node *root) {
```

```
int res = 1;
while (root != NULL) {
    if (root->left == NULL && root->right == NULL)
        return 2*res;
    if (root->left == NULL) {
        ~~return 3*~~ res*=3;
        root = root->right;
    }
    else if (root->right == NULL) {
        res *= 4;
        root = root->left;
    }
    else if (root->right->data > root->left->data ) {
        res *= 5;
        root = root->right;
    }
    else {
        ~~return res;~~ res*=6;
        root = root->left;
    }
}
return res;
```