COP 3502
10/27/23

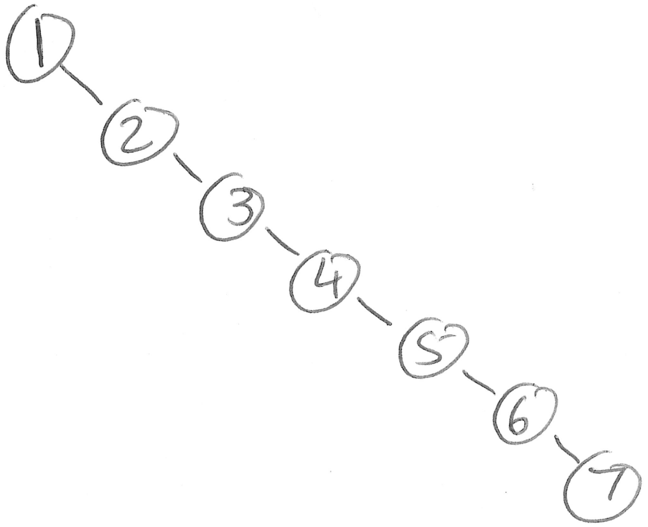
Name: ____________________
UCFID: ____________________
NID: ____________________

1) (5 pts) ALG (Binary Trees)

Draw a single **binary search tree** that meets all the following conditions:

- The tree contains 7 nodes.
- The tree's pre-order traversal is the same as its in-order traversal.
- The tree does not contain any duplicate values.

If it is not possible to draw such a tree, say so and explain why not.



1) (10 pts) ALG (Binary Trees)

Consider a function that takes in a pointer to a binary tree node and returns a pointer to a binary tree node defined below:

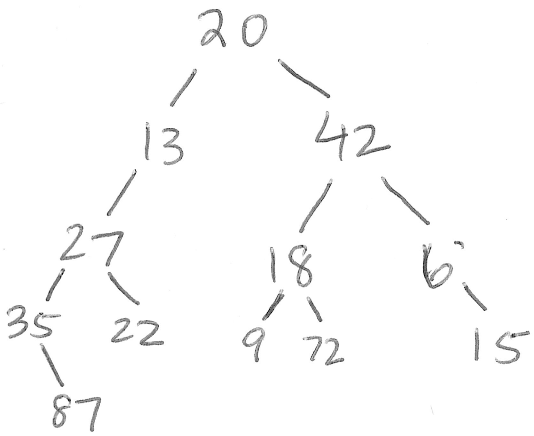
```
typedef struct bintreenode {
    int data;
    struct bintreenode* left;
    struct bintreenode* right;
} btreenode;

btreenode* somefunction(btreenode* root) {
    if (root == NULL) return NULL;
    somefunction(root->left);
    somefunction(root->right);
    btreenode* tmp = root->left;
    root->left = root->right;
    root->right = tmp;
    return root;
}
```

Swap left + right ptrs

Let the pointer tree point to the root node of the tree depicted below:

```
                 20
            /          \
          42            13
       /     \            \
      6       18           27
     /       /  \         /  \
   15      72    9       22   35
                              /
                            87
```

If the line of code `tree = somefunction(tree)` were executed, draw a picture of the resulting binary tree that the pointer tree would point to.



mirror image

1) (10 pts) DSN (Binary Trees)

The goal of a function named *legacyCount()* is to take the root of a binary tree (*root*) and return the number of nodes that contain a value greater than at least one of their ancestors. For example, this function would return **4** for the following tree, since **60** is greater than both of its ancestors (3 and 14), **40** is greater than two of its ancestors (3 and 14) (even though 40 isn't greater than its parent!), **28** is greater than both of its ancestors (9 and 14), and **2** is greater than one of its ancestors (1).

```
          14
         /   \
        3      9
       / \      \
      1   60     28
     /      \
    2        40
```

Our node struct is as follows:

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

To make the code work, *legacyCount*() is a wrapper function for a recursive function called *legacyHelper*(). Included below is the code for legacyCount() as well as the function signature for *legacyHelper*(). Write all of the code for the *legacyHelper*() function. Note: If *root* is NULL, you should return 0.

```
int legacyCount(node *root) {
    if (root == NULL) return 0; ✓
    return legacyHelper(root->left, root->data) +
           legacyHelper(root->right, root->data);
}

int legacyHelper(node* root, int minAncestor) {
     if (root == NULL) return 0;
     int ans = 0;
     if (root->data > minAn~~s~~cestor)
          ans = 1;
     else
          minAncestor = root->data
     return ~~Te~~ ans + legacyHelper(root->right, minAncestor)
               + legacyHelper(root->left, minAncestor);


}
```

1) (10 pts) DSN (Binary Trees)

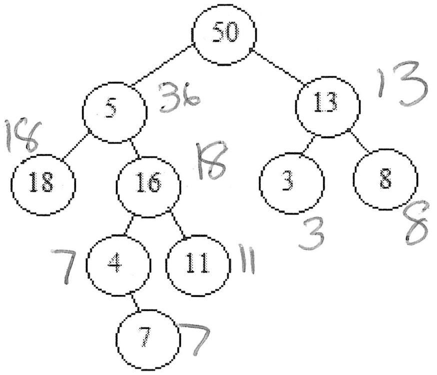
Demonstrate your understanding of recursion by rewriting the following recursive function, which operates on a binary tree, as an **iterative** function. In doing so, you must abide by the following restrictions:

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 res *2;
        else if (root -> left == NULL) {
            res *= 3;
            root = root -> right;
        }
        else if (root -> right == NULL) {
            res *=4;
            root = root -> left;
        }
        else if (root -> right -> date > root -> left -> date) {
            res *= 5;
            root = root -> right;
        }
        else {
            res *= 6;
            root = root -> left;
    }   }
    return res;
}
```

1) (10 pts) ALG (Binary Trees)

What does the function call `solve(root)` print out if root is pointing to the node storing 50 in the tree shown below? The necessary struct and function are provided below as well. Please fill in the blanks shown below. (Note: the left pointer of the node storing 50 points to the node storing 5, and all of the pointers shown correspond to the direction they are drawn in the picture below.)

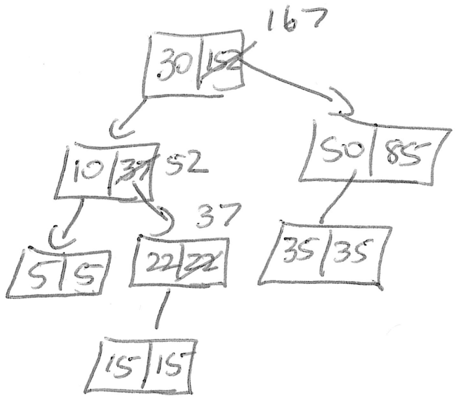


```
typedef struct bstNode {
    int data;
    struct bstNode *left;
    struct bstNode *right;
} bstNode;

int solve(bstNode* root) {

    if (root == NULL) return 0;

    int res = root->data;
    int left = solve(root->left);
    int right = solve(root->right);

    if (left+right > res)
        res = left+right;

    printf("%d, ", res);
    return res;
}
```

18, 7, 7, 11, 18, 36, 3, 8, 13, 50,

1) (10 pts) DSN (Binary Search Trees)

A modified BST node stores the `sum` of the `data` values in its sub-tree. **Complete** writing the insert function shown below ***recursively***, so that it takes in a pointer to the root of a binary search tree, *root*, and an integer, *value*, inserts a node storing value in it into the tree and returns a pointer to the root of the resulting tree. Notice that this task is more difficult than a usual binary tree insert since the sum values in several nodes must be updated as well. The struct used to store a node is shown below.



```
typedef struct bstNode {
   struct bstNode * left, * right;
   int data;
   int sum;
} bstNode;

bstNode* insert(bstNode * root, int value){


    if (root == NULL) {
        bstNode* res = malloc(sizeof(bstNode));

        res->data = Value ;

        res->sum = Value ;

        res->left = NULL ;

        res->right = NULL;
        return res;
    }

    if (value <= root->data)
        root->left = insert(root->left, value);         ;
    else
        root->right = insert(root->right, value)        ;
    root-> Sum += Value;                                ;

    return root;
}
```

Spring 2021 **Data Structures Exam, Part B**

Name: ______________________

UCFID: ______________________

NID: _________________________

1) (10 pts) DSN (Binary Trees)

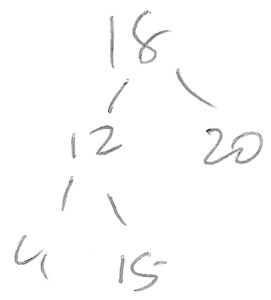
Write a function named *find_below*() that takes a pointer to the root of a binary tree (*root*) and an integer value (*val*) and returns the greatest value in the tree that is strictly less than *val*. If no such value exists, simply return *val* itself. Note that the tree passed to your function will **not** necessarily be a binary **search** tree; it's just a regular binary tree.

For example:

```
        18            find_below(root, 196) would return 22
       /  \           find_below(root, 1) would return 1
      7    4          find_below(root, 4) would return 1
     / \              find_below(root, 22) would return 18
    1   22            find_below(root, 20) would return 18
         \            find_below(root, 8) would return 7
          8           find_below(root, -23) would return -23
```

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 find_below(node *root, int val)
{
      if (root == NULL) return val;
      ~~int numCand = 0;~~  int nC = 0;
      int alt[3]; alt[0]=val; alt[1] = val; alt[2] = val;
      if (root->date < val){alt[0] = root->data; nC++; }
      int leftS = find_below(~~root, val~~); (root->left, val);
      if (leftS < val){alt[1] = leftS; nC++; }
      int rightS = find_below(root->right, val);
      if (rightS < val){alt[2] = rightS; nc++; }

      if (nC ==0) return val;
      ~~Int res~~    ~~int res = =~~ ~~int res = alt[0];~~
            ~~for (int i = 1; i<3; i++) if (alt[~~
            int res = val+1;
}
            for (int i=0; i<3; i++) {
                  if ( alt[i] == val) continue;
                  if ( res == val+1) res = alt[i]
```

```
    if (~~re~~alt[i] > res) res = alt[i];
}
return res;
```