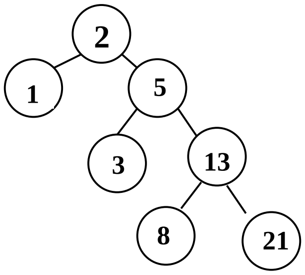
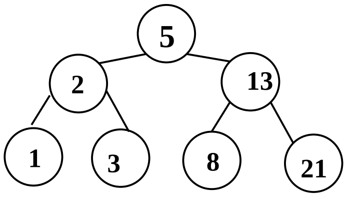
CS 3050 Homework # 3. **Name : KEY**

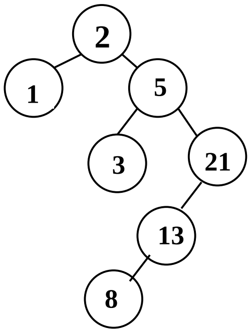
Submitted to Canvas, due at 11:59pm on March 7, 2018.

1. For the set A = {1, 2, 3, 5, 8, 13, 21} of keys, draw three binary search trees of different heights.

Answer:
(Student answers may vary)







2. Suppose that we have numbers between 1 and 1000 in a binary search tree, and we want to search for the number 363. Which of the following sequences could not be the sequence of nodes examined? Explain why.

a. 2, 252, 401, 398, 330, 344, 397, 363.
b. 924, 220, 911, 244, 898, 258, 362, 363.
c. 925, 202, 911, 240, 912, 245, 363. , (912 is invalid)
d. 2, 399, 387, 219, 266, 382, 381, 278, 363.
e. 935, 278, 347, 621, 299, 392, 358, 363. (299 is invalid)

3. For a binary search tree T of n nodes, determine the number of internal nodes in O(n) running time in psuedo code.
(1) Give a recursive algorithm.

```
TREE-COUNT-INTERNAL(T)
   i = 0
   if(T[left])!= NULL)
      i = i + 1 + TREE-COUNT-INTERNAL(T[left])
```

```
    if(T[right])!= NULL)
        if(T[left] != NULL)
            i = i + TREE-COUNT-INTERNAL(T[right])
        else
            i = i + 1 + TREE-COUNT-INTERNAL(T[right])
return i
```

(2) Give a non-recursive algorithm.

Students answers may vary depending on if they used a queue or stack like data structure.

4. Given a Binary Search Tree T and two integers a and b on the tree, print all elements in T between a and b in psuedo code.

```
printAllNodes(T,a,b)
        if a > b
                swap(a,b)
        if T == Null
                return 0
        if a <= T->key
                printAllNodes(T->left,a,b)
        if a <= T-key and b >= T->key
                print(T->key)
        if T->key <= b
                printAllNodes(T->right,a,b)
        return
```

5. Suppose that instead of each node x keeping the attribute x:p, pointing to x’s parent, it keeps x:succ, pointing to x’s successor. Give pseudocode for SEARCH, INSERT, and DELETE on a binary search tree T using this representation. These procedures should operate in time O(h), where h is the height of the tree T . (Hint: You may wish to implement a subroutine that returns the parent of a node. This is the same question of 12.3-5 in the textbook.)

Answer:

```
/*     insert(inserted)    //O(h)
 * 1    def parent, predecessor, successor
 * 2    def curr = root
 * 3    while(curr)
 * 4        parent = curr
 * 5        if(inserted->key < curr->key)
 * 6           successor = curr
 * 7           curr = curr->left
 * 8       else
 *            predecessor = curr
 * 9           curr = curr->right
 * 10   if(!parent)
 * 11       root    =   inserted
 * 12   else
          (inserted->key < parent->key?   parent->left    :   parent->right)
                 =   inserted;
 * 13   inserted->sucessor = successor
 * 14   if(predecessor)
 * 15       predecessor->successor = inserted
 */

/*      search(key)     O(h)
 * 1    def curr = root
 * 2    while(curr  && key != curr->key)
 * 3        curr = (key < curr->key?    curr->left  :   curr->right)
 * 4    return curr
 */


/*      delete(target)  O(h)
 * 1    def pred = predessor(target)
 * 2    if(pred)
 * 3        pred->successor = target->successor
 * 4    if(!target->left)
 * 5        transplant(target, target->right)
 * 6    else if (!target->right)
 * 7        transplant(target, target->left)
 * 8    else
 *         def replacer = minimum(target->right)
 * 9        if(parent(target) != target)
 * 10           transplant(replacer, replacer->right)
 * 11           replacer->right = target->right
 * 12       transplant(target, replacer)
 * 13       replacer->left  =  target->left
```