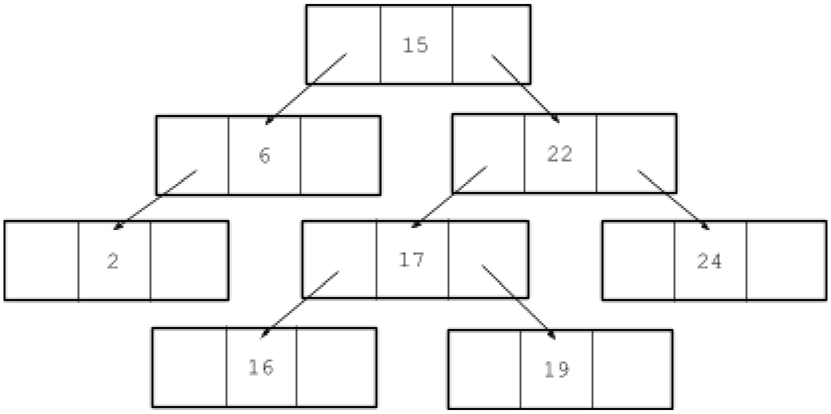
**Hannah Zhang**

**Binary Search Tree Problems**

0. Draw a picture of bst.



1. Write the function contains? which takes a binary search tree of numbers and a datum as its input and returns #t if the datum is in the tree, #f otherwise.

```
> (contains? bst 17)
#t
> (contains? bst 18)
#f
; empty tree must be false
; if the datum of a node is equal to the datum inputted → true
; if not, we keep on looking
; use previously defined abstraction functions
; this way is not logarithmic
(define (contains? bst num)
  (cond ((empty-tree? bst) #f)
        ((equal? (datum bst) num) #t)
        (else (or (contains? (left bst) num) (contains? (right
bst) num)))))
; logarithmically
; test left, test right, else return true
(define (contains2? bst num)
  (cond ((empty-tree? bst) #f)
        ((> num (datum bst)) (contains2? (right bst) num))
        ((< num (datum bst)) (contains2? (left bst) num))
        (else #t)))
```

2. Write the function inorder which takes a binary search tree as its input and returns a list of the data in ascending order.

```
> (inorder bst)
(2 6 15 16 17 19 22 24)
; it is a binary search tree so everything is organized
; use append
; append all of the datum of the left side, the datum of the
root, and all of the datum of the right side
(define (inorder bst)
  (cond ((empty-tree? bst) '())
        (else
         (append (inorder (left bst)) (append (list (datum bst))
(inorder (right bst)))))))
```

3. Write the function count-nodes which takes a binary tree (does not have to be a binary search tree) as its input and returns the number of nodes in the tree.

```
> (count-nodes bst)
8
; if empty, add zero and terminate
; if it is not empty then there is a node so add 1
(define (count-nodes bst)
  (if (empty-tree? bst)
      0
      (+ 1 (count-nodes (left bst)) (count-nodes (right bst)))))
```

4. Write the function square-tree which takes a binary tree that contains numbers as its input and returns a binary tree with the same structure and the numbers squared.

```
> (datum (left (left (right (square-tree bst)))))
256
; returns a tree
; the left and right stay the same but the datum is squared
; call recursion on left and right of the node
(define (square-tree bst)
  (cond ((empty-tree? bst) '())
        (else (make-bintree (square (datum bst)) (square-tree
(left bst)) (square-tree (right bst))))))
```