Tree:

What: - A container object
  - Composed of a TNodes
    - Each TNode holds __________
  - One “root” TNode pointer
    - The __________ TNode in the Tree
  - Zero or more “left” and “right” child TNodes
    - “_______” tree:
      ▪ ______ = ______ ➝ “left” and “right”
      - Each points down the Tree
      - Serve as the infrastructure of the Tree
      - Similar to “pre” and “next” of List
  - Optional “parent” pointer
    - Points _____ the Tree
  - Search Tree:
    - Order implemented:
      ▪ Smaller items go left.
Tree terminology and properties:
- __________: the TNode without a parent,
  o entry point into the Tree
- __________: a TNode without a child
- __________: TNodes that share the same parent
- __________: a TNode in a search path that follows from an ancestor.
- __________: a TNode in a search path that terminates with a descendent.
- __________:
  o $1 + \text{height of tallest child}$
  o $0$ for a leaf
- __________:
  o left-height – right-height
  o -1 height of non-existent children.
- __________: the TNodes visited from the root to a leaf followed when searching for an element.
  o does not visit all TNodes.
- __________: visiting every TNode in the Tree
  o __________: visit, go left, go right
  o __________: go left, visit, go right
  o __________: go left, go right, visit
- __________: the action to take when at a TNode.

Spindly trees: __________
Bushy trees: __________
# of TNodes in an optimally balanced Tree: all leaf TNodes at the same level.

<table>
<thead>
<tr>
<th># of TNodes to check for unsuccessful search or to perform insert in an optimally balanced Tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>15</td>
</tr>
<tr>
<td>31</td>
</tr>
<tr>
<td>63</td>
</tr>
<tr>
<td>127</td>
</tr>
<tr>
<td>255</td>
</tr>
<tr>
<td>511</td>
</tr>
<tr>
<td>1023</td>
</tr>
<tr>
<td>2047</td>
</tr>
<tr>
<td>4095</td>
</tr>
</tbody>
</table>

More on Tree __________:
Use of __________:
- ______________________________

Use of a __________:
- ______________________________

Use of a post__________:
- ______________________________.
Removing from Tree:
- “Real” remove is a challenge
  o delayed until hw8
- “__________” remove
  o removal algorithm of hw7
  o __________ the item is not there
  o lookup with return __________
  o writing Tree __________ display item
  o item is really __________
    - __________ the structure of Tree
  o no changing of height, balance
  o __________ flag will true or false.
Implementation Decisions:
Tree class, TNode class

Loop based solution of Insert (for example):
- Tree Insert method, TNode Insert method
  - current/working TNode to refer to TNode of
  - starts at “________”
  - reassigned to ____________ of current as you go
  - algorithm of hw7
  - height and balance need to be updated as you go
  - “__________” pointer is needed.

Recursive solution for Insert:
- Tree Insert method, TNode Insert method
  - Tree method
    - operates at ________
    - delegates to _________ for any further task
  - TNode method
    - operates at ________
    - delegate to _________ for any further task
  - algorithm of hw8
  - no “________” is needed to go ______ the Tree as
    you update height and balance.
Recursion Example: (decout)

#include <stdio.h>

/* Algorithm:
  1. Isolate last digit of the number to print
  2. Call a function (decout) to print all prior digits
  3. Print digit isolated.
*/
void decout (unsigned long number) {

}

main () {
    decout (1234);

    /* note: $$ 1000: return address */
    fputc (', stdout);
}

| Stack frame for last recursive decout call | remainder |
|                                           | quotient  |
|                                           | number    |
|                                           | ...       |
|                                           | return address |
| Stack frame for next recursive decout call | remainder |
|                                           | quotient  |
|                                           | number    |
|                                           | ...       |
|                                           | return address |
| Stack frame for first recursive decout call | remainder |
|                                           | quotient  |
|                                           | number    |
|                                           | ...       |
|                                           | return address |
| Stack frame for initial decout call       | remainder |
|                                           | quotient  |
|                                           | number    |
|                                           | ...       |
|                                           | return address |
|                                           | Label/description | Value on RTS |

Output: __________ (as each stack frame is popped).
Implications:
- Parameters are distinct from original variable that is passed to function call: call by _____
- Local variables are created _____________________
- With recursion local variables exist _______ times, one copy for each ________________.
- Debugger/Java exceptions use the RTS ____________________
- Infinite recursion will _____________________.
- Overwriting a RTS array could mean overwriting return addresses, and then you can’t ____________________.
- Parameters act just like ________________.
- More resources within your allotted resources are consumed with recursion than with a loop, those resources (___________) are allocated to you when ____________________.
- RTS allocation is the _________ efficient allocation possible:
  o Allocation and decallocation occur with assignment of ________________:
    ▪ PC is Program Counter: address in the text section for the instruction about to execute.
    ▪ SP is the Stack Pointer: address in the RTS that points to the top of the RTS.
      • Call resets ___, allocating __________
      • Call resets ___, which ______________
      • Return: restores the ___
      • Return: stores _________________
Constants:
- The value associated with symbol can ____ change.
- Syntax: “const” keyword
- Parameters:
  o Parameter is read-only
    ▪ Prefix “const” to the parameter declaration.
    ▪ Enforced in method and in any methods called using parameter.
    ▪ Originating value may or may not be const (mostly not).
    ▪ Look but don’t touch.
    ▪ Provided documentation to the user giving assurance that original item passed won’t change.
    ▪ When to use:
      • __________________________!
- Member methods:
  o Object whose member method is being called is read only.
    ▪ Add “const” after parameter list before open curly brace to the method body.
    ▪ When to use:
      • __________________________!
- Return result:
  o Result is read-only
    ▪ Prefix “const” to the return type of the method.
    ▪ When to use:
      • When object is still needed in its original form elsewhere.