## Section 1: True / False (2 points each, 30 pts total)

Circle the word TRUE or the word FALSE. If neither is circled, both are circled, or it impossible to tell which is circled, your answer will be considered wrong.

1. TRUE or FALSE
   Computer scientists always draw *tree* structures with the root at the top and leaves at the bottom because they don't get out enough to remember how trees grow in nature.
2. TRUE or FALSE
   In a makefile, the default target is the last one in the file.
3. TRUE or FALSE
   Local variables declared in a function are stored on the stack.
4. TRUE or FALSE
   The function $n \log n^2$ is $O(n^2)$
5. TRUE or FALSE
   The function $n \log n^2$ is $O(n \log n)$
6. TRUE or FALSE
   The function $n \log n^2$ is $\Omega(n \log n)$
7. TRUE or FALSE
   The function $n \log n^2$ is $\Omega(n^2)$
8. TRUE or FALSE
   The function $4n^2 + n^2 \log 2n$ is $O(n^2)$
9. TRUE or FALSE
   The function $4n^2 + n^2 \log 2n$ is $O(n^3)$
10. TRUE or FALSE
    If an algorithm consists of a program segment that is $O(n^2)$, sequentially followed by a segment that is $O(n)$, the entire algorithm is $O(n^3)$
11. TRUE or FALSE
    A function can be both $O(n^2)$ and $\Omega(n^3)$
12. TRUE or FALSE
    An in-order traversal of a binary search tree will process the smallest value first.
13. TRUE or FALSE
    In the function declaration "**`foo(int arg1, int * & arg2)`**", a pointer to `arg2` is being passed by reference.
14. TRUE or FALSE
    Both individual functions and entire classes can be templated.
15. TRUE or FALSE
    For container classes that support iterators, the function **`end()`** return an iterator that points to the last element in the container.

---

## Section 2: Short Answer (varying point values. 48 pts total)

16. (**6 points**) Show that the function $5n^3 + 3n + 4$ is $O(n^3)$
(your answer should include a specific $n_0$ and $c$, as well as an algebraic justification that the limit holds for *all* $n > n_0$; you do not need any inductive proof)

**For all $n \geq 1$: $3n^3 \geq 3n$, and $4n^3 \geq 4$**
**$\rightarrow$ $5n^3 + 3n^3 + 4n^3 = 12n^3 \geq 5n^3 + 3n + 4$**
**So, for $c = 12$ and $n_0 = 1$, $5n^3 + 3n + 4$ is $O(n^3)$**

For questions 17 and 18, Give asymptotic worst-case running times for each of the following code fragments. Characterize the running time as closely as possible (e.g., don't say $O(n3)$ if $O(n^2)$ also works). Express running times as a function of $n$. Make sure you give a satisfactory explanation—it is worth a significant fraction of the points.

17. **(6 points)**

```
for (x = 0; x < n; x++) {
    for (y = x; y < n; y++) {
        sum += x * y;
    }
}
```

Running time: O(___**$n^2$**____)
Justification:
**The inner loop runs 0 times, then n times, then n-1, n-2... down to 1. So, the number of times the body of the inner loop is executed is sum(1..n), which is $O(n^2)$. Another way to think about it is the average number of inner turns is n/2, so total is (n/2)n, again $O(n^2)$**

18. **(6 points)**

```
sum = 0;
for (a = 0; a < n; a++) {
    sum += a;
}
prod = 1;
for (a = sum; a > 0; a--) {
    prod *= a;
}
```

Running time: O(___**$n^2$**____)
Justification:
**The first loop executes n times, and the second loop executes sum(0..n-1)=$n^2$ times,so the complete code runs in $O(n + n^2)$, which by the rule of sums, is is still $O(n^2)$**

19. **(2 points)** What is the maximum height of a binary search tree with *n* nodes?

**max height = n – 1: essentially a linked list.**

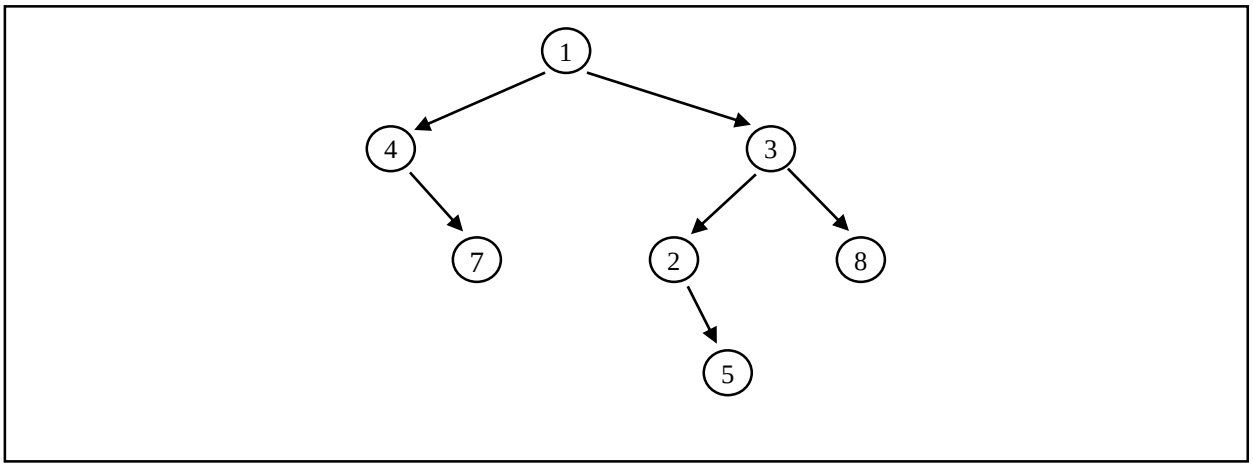
20. **( 2 points)** What is the minimum height of a binary search tree with *n* nodes?

**min height = log(n).**

21. (**3 points**) What is the main difference between a plain binary tree and a binary search tree?

**The binary search tree (BST) is a binary tree with a special additional property: for any given node, all of the nodes in that node's left subtree have keys less than the node, and all of the nodes in the right subtree have keys that are greater**

Questions 22 thru 25 refer to the following binary tree:




22. **(2 points)** The height of this tree is: _____**3**_____

23. **(2 points)** The depth of the node labeled '7' is: _____**2**_____

24. **(2 points)** The height of the node labeled '3' is: _____**2**_____

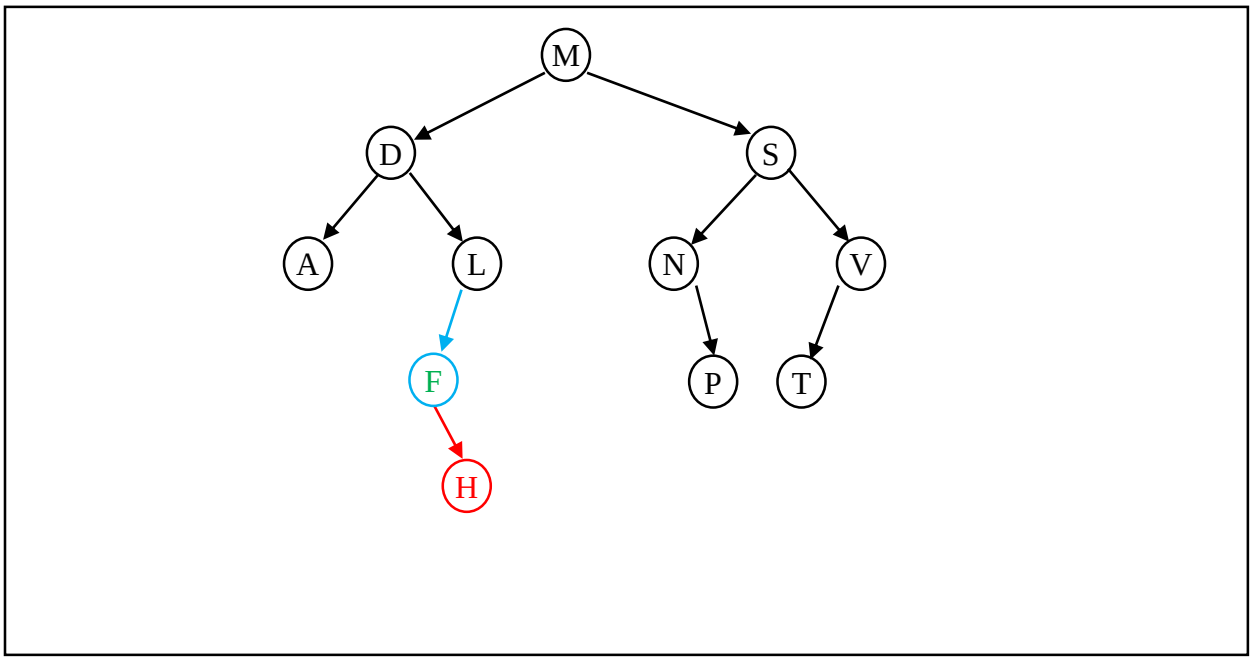
25. **(4 points)** Print out the order the nodes would be visited for a postorder traversal:

_7_, _4_, _5_, _2_, _8_, _3_, _1_,

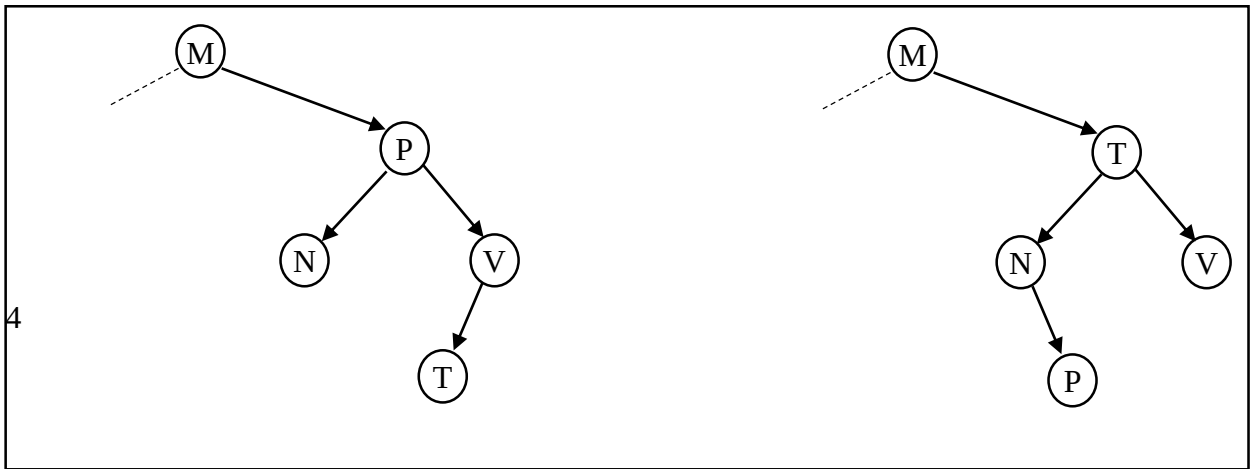
26. (**3 points**) Differentiate the experimental versus analytical approaches to determining algorithmic complexity.

**experimental approach: implement the algorithm and run it, measuring time taken;**
**analytical approach: examine the code and mathematically determine how many times each step is executed, at what cost**

Questions 27 thru 29 refer to the following binary search tree, with nodes in alphabetical order:




27. **(3 points)** First insert into the above tree a node with the value "F"
(you can draw on the diagram above)
*(See new blue node "F" above)*
28. **(3 points)** Then insert into the tree a node with the value "H"
(again, you can add to the diagram above)
*(See new red node "H" above)*
29. **(4 points)** Draw the resulting tree when you delete the node "S" from the *original* tree
(draw the complete new tree below):



## Section 3: Design and Coding (22 points total)

Note: for this section, you are being asked to write more substantial sections of code. Clarity of code structure and efficiency of design count.

30. **(6 points)** Write a code snippet with a for-loop that uses an iterator to print out all of the values stored in a STL **`list<int>`** object called **`listOfInts`**. Be sure to declare your iterator object correctly.

```
list<int> listOfInts;

… // Assume there will be code here to fill listOfInts with numbers

// Now, your code here to print out all the values in listOfInts;
// You must use an iterator!!!

    list<int>::iterator it;

    for (it = listOfInts.begin(); it != listOfInts.end(); it++) {
        cout << *it << endl;
    }
```

31. **(10 points)** You are given the following C++ class definition for a singly-linked list node class (**Node**):

```
class Node {
  public:
    int data;
    Node *next;
}
```

Write the code body for the function **swap_with_next()**, which is called with two arguments: **list**, a pointer to the first **Node** of a singly-linked list; and **data**, a value to search for in the list. When the Node with the given value is found, it should then be swapped with the Node immediately following it. The **next** field of the last node in the list is **NULL**. To keep the solution simple, you can assume that the list is not empty, that the value you are searching for is in the list, and that it will not be the first element or last element in the list.

```
void swap_with_next(Node *list, int data) {

    // taking full advantage of assumption that list isn't empty,
    // and that we do not need to handle swap at head
    Node *prev, *curr, *next, *nextNext;
    for (prev = list, curr = prev->next; curr->next != NULL;
           prev = curr, curr = curr->next) {
        if (curr->data == data) {
            next = curr->next;
            nextNext = next->next;
            prev->next = next;
            next->next = curr;
            curr->next = nextNext;
            return;
        }

}
```

32. **(6 points)** You are given the following C++ class definition for **`TNode`** (short for "tree node"), that is used to represent nodes in a binary search tree ("BST"):

```
class TNode {
  public:
    int data;
    TNode *left, *right;
}
```

Every **`TNode`** contains a distinct data value. Write the code body for the recursive function **`contains()`**, which is called with two arguments: **`root`**: a pointer to the root node of a tree or subtree, and **`data`**: the value to search for in the tree. The function returns **`true`** if the value was found in the tree/subtree, **`false`** otherwise. Note again that this is a proper BST. Also note that either/both the pointers **`root->left`** and **`root->right`** can be **`NULL`**. To keep the solution simple, you can assume that the tree initially passed in is not empty (i.e., main() promises not to call the initial invocation of **`contains()`** with **`NULL`**). As to whether **`contains()`** ever recursively calls itself with **`NULL`**, as many of our in-class examples did, is up to you. Don't forget the "return" statement at the end!

```
bool contains(TNode *root, int data) {

    if (root == NULL)
        return false;
    else if (data < root->data) {
        return contains(root->left, data);
    } else if (data > root->data) {
        return contains(root->right, data);
    } else {
        return true;
    }

}
```

33. **Extra Credit (5 points)**
34.

Give asymptotic worst-case running times, as a function of *n*, for the following code. Characterize the running time as closely as possible:

```
int sum = 0;
for (i = 0; i < n; i++) {
    for (int j = i; j < n; j++) {
        for (int x = i; x < j; x++) {
            sum += x;
        }
    }
}
```

Be careful—this is trickier than it seems! That's why it's extra credit.

Running time: O(___**$n^3$**____)
Justification:

**The innermost loop runs j-i times,but since j ranges from i to n, for any given i, it runs 0 times, then 1, then 2 .. n-i-1 times. So it runs a total of sum(0..n-i-1). Now, define i′ = n-i, then our terms become sum(0..i′-1)where i′ ranges from 0..n-1. sum(0..i) is O($i^2$), so our outer loop makes the whole function $\text{sum}_{i:0..n}(i^2)$, which is O($n^3$)**