NAME (print): ____________________________________________

Honor Acknowledgment (signature): ____________________________

DO NOT SPEND MORE THAN 10 OR SO MINUTES ON ANY OF THE OTHER QUESTIONS! If you don’t see the solution to a problem right away, move on to another problem and come back to it later.

Before starting, make sure your test contains 7 pages.

If you think there is a syntax error, then ask. You may assume any include statements are provided.

<table>
<thead>
<tr>
<th>Problem</th>
<th>value</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem 1</td>
<td>12 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 2</td>
<td>10 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 3</td>
<td>12 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 4</td>
<td>22 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 5</td>
<td>14 pts.</td>
<td></td>
</tr>
<tr>
<td>TOTAL:</td>
<td>70 pts.</td>
<td></td>
</tr>
</tbody>
</table>

Whenever you see references to Node or TNode in this test, check the separate handout for their definition (unless a specific definition is included in the problem statement).
PROBLEM 1:  (Small stuff... (12 points))

Part A (5 points)
Draw the single binary tree from which the following traversals result (note: not a BST):

pre:  A C F I H B E D G
in:   F C H I B A D G E

Part B (3 points)
Give a possible order of insertion that would result in the following binary search tree (BST):

```
    I
   / \
  E   L
 /   /
C   O
  \ /
   V
  / 
 P
  \  
   S
```

Part C (4 points)
Evaluate the following postfix expression.
3 1 4 + 2 5 * + *

Also write the above expression in normal infix form with parentheses where needed (assuming normal precedence).
PROBLEM 2:  (You’ve got it all backwards (10 points))

Write the function reverse whose header is given below. The function reverses the elements of the queue q.
For example, if q is represented by (a,b,c,d), with a the first element and d the last element of the queue, then after the call reverse(q), q is represented by (d,c,b,a).
For full credit, you may use only $O(1)$ explicit additional memory. That means you may not create any vectors, queues, stacks, etc., or anything that grows with the size of the queue passed in.
Hint: Use recursion. Remember how we printed out a file or linked list in reverse order...

```c
void Reverse(Queue<int> & q)
// pre: q is represented by (a1, a2, ..., an)
// post: q represented by (an, ..., a2, a1), i.e., q is reversed
```
PROBLEM 3:  *(Add it up (12 points))*

Write a routine called \textit{SumBT} that creates a new binary tree that is the "sum" of two identically shaped binary trees.

Binary trees are defined to have the same shape if they are identical except for their info fields. That is, they have the same pattern of NULL and non-NULl pointers in their right and left fields.

\begin{verbatim}
TNode<int> * SumBT(TNode<int> * t, TNode<int> * s)
// pre: s and t are binary trees with the same shape
// post: returns a binary tree with the same shape as s or t
// but whose info fields are each the sum of the corresponding
// info fields in s and t.
{

\end{verbatim}
PROBLEM 4: (Analyses ... (22 points))

Part A (5 points) What is the worst case running time T(N) as expressed by using O(?) (or big O) for the following (crazy) code segment?

```c
int j, k, h;
int sum = 0;
for (k = 1; k <= N; k++) // N has a value from previous code
{
    for (j = 1; j <= n; j = j*2)
    {
        sum += j;
    }
}
```

Part B (5 points) Consider the following (nonsensical) Mystery function, but don’t attempt to decipher what it computes. Instead, give only the recurrence relation describing the running time of Mystery. DO NOT attempt to solve the recurrence relation or to compute big-Oh.

```c
int Mystery(int N)
{
    int tot = 0;
    int k, j;
    if (N == 0)
        return 1;
    if (N%3 == 0 || N%3 == 1)
    {
        tot += 3*Mystery(N/3);
    }
    if (N%3 == 1 || N%3 == 2)
    {
        tot += 5*Mystery(N/3);
    }
    if (N%3 == 2 || N%3 == 0)
    {
        tot += 7*Mystery(N/3);
    }
    return tot;
}
```
Part C (6 points) Assume you are given a pointer to a binary tree.
Consider the most efficient implementation of each of the following operations, given only a pointer to the root of the tree.
For example, find the maximum would be implemented as a function with the only parameter a pointer to the linked list. DO NOT modify the data structure. Also, consider only solutions that do not require more than $O(1)$ additional explicit memory, but ignore memory requirements implicit in function call mechanisms, etc.

1. Find the maximum element (tree is a BST). Running time:
2. Find the minimum element (tree is not a BST). Running time:
3. Is X an element in the tree (tree is a BST)? Running time:
4. Is X an element in the tree (tree is not a BST)? Running time:
5. Print the element that occurs most often (tree is a BST). Running time:
6. List the elements from largest to smallest (tree is a BST). Running time:

Part D (6 points)
Solve the recurrence relation given below and give the resultant big Oh. Show your work. Answers without justification will earn zero credit (or some multiple of that :-)

$T(1) = 1$

$T(N) = 2 \cdot T(N/2) + N$
PROBLEM 5:  \((Merge\ Traffic\ (14\ points))\)

Write the function \textit{Merge}, whose header is show below, that will take two ordered linked lists and merge them into a new ordered linked list.

Here our strategy is to NOT create or delete any nodes. Instead, you will take existing nodes that were linked together as sorted, linked lists and re-arrange them, by changing pointers appropriately, to transform them into a single sorted linked list.

For example if we had list\text{A} represented as
\[\text{listA} \rightarrow 3 \rightarrow 6 \rightarrow 8 \rightarrow 13 \ \n\]
and list\text{B} as
\[\text{listB} \rightarrow 1 \rightarrow 2 \rightarrow 9 \ \n\]
we should expect \textit{Merge} to return
\[\text{Merge} \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 6 \rightarrow 8 \rightarrow 9 \rightarrow 13 \ \n\]
were \(\n\) represents the NULL pointer.

\begin{verbatim}
Node * Merge(Node * listA, Node * listB)
// pre: listA and listB are each NULL-terminated, linked lists with NO
// header nodes the Nodes are ordered with the smallest first.
// Either or both may be empty.
// post: returns pointer to a NULL-terminated, linked lists with NO
// header node. The Nodes are ordered with the smallest first.
{
\end{verbatim}