Program Set #3
Total Points: 30

Three problems must be implemented for full credit. The required (starred) problem marked in the set must be implemented by everyone. See the 2436 Grading and Submission Guide Sheets for additional grading/submission information. Partial credit will be given. (10 points each)

Section One- Choose two problems.

1. Write a C++ program to see the various "guesses" the computer makes when guessing a number using a binary search. The program should allow for the input of N representing the largest positive integer in a possible set of numbers. Then an integer R will be input representing the target number. The program will use a binary search strategy to "find" that value. Each line of output will display not only each of the program's guesses, but the low value and high values that were used to determine the guesses. All guesses are displayed until the target is reached or not found. For example, if 11-21-31 is displayed, the 21 indicates the guess while the 11 and the 41 indicate the current lower and upper bounds. If there is an odd number of items in the remaining range of numbers, the guess will be the middle number. For example, if there were 11 numbers remaining, the guess would be the 6th number in that list. If there are an even number of items remaining, the guess will be the lesser of the two middle numbers. For example, if 20 numbers remain, the guess would be the 10th number in the list.

Input from the keyboard two positive integers two positive integers N and R. N represents the largest possible integer in the range [1,100000] and R represents the target. For each case, output to the screen three sets of integers printed on separate lines until the target is found. The first is the low value of the current range, the second is the computer's guess, and the third is the high number in the current range. A dash (-) will separate the numbers with no extra spaces. If the target is found, output to the screen the string "GOT IT!"; otherwise, output "NOT GOT IT!". Finally, the program should ask if the user wants to run the program again (Check case). Refer to the sample output below.

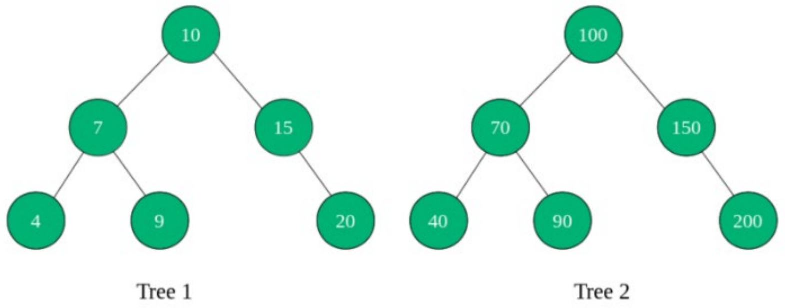
Sample Run:

```
Enter largest range value [1-100000]: 10
Enter search target: 7

1-5-10
6-8-10
6-6-7
7-7-7
GOT IT!

Run Again (Y/N): y

Enter largest range value [1-100000]: 100
Enter search target: 32

1-50-100
1-25-49
```

```
26-37-49
26-31-36
32-34-36
32-32-33
GOT IT!

Run Again (Y/N): N
```

Name the program: `ShowBinSearchXX.cpp`, where XX are your initials.

2. Write a C++ program to check if two binary search trees are identical in structure (shape) not content (values). For example, the following trees have the same structure:



Input from the keyboard two integer trees of up to 10 values on separate lines. Output to the screen if the trees have the same shape or not. The program must use a tree data structure. Finally, the program should ask if the user wants to run the program again (Check case). Refer to the sample output below.

Sample Run:

```
Enter tree 1: 10 7 15 9 4 20
Enter tree 2: 100 70 40 90 150 200

Same structure.

Run Again (Y/N): Y

Enter tree 1: 3 4 5 6 7
Enter tree 2: 5 10 1 3 4

Not same structure.

Run Again (Y/N): n
```

Name the program: `TreeStructureXX.cpp`, where XX are your initials.

3. Write a C++ program that takes any word, like `INVITATIONAL`, builds a binary search tree in alpha order, allowing duplicate letters, which will be inserted to the left when the duplicate is encountered, and then calculates and reports the following information:

- Depth - distance from root to farthest leaf node
- Number of Leaf Nodes - number of nodes with no children
- Number of External Nodes - number of potential nodes (right or left child null pointers)
- Internal Path Length (IPL) - sum of all distances from each internal node to the root
- External Path Length (EPL) - sum of all distances from external nodes to the root

For example, the tree for the word `INVITATIONAL` is:

```
        I
      /   \
    I       N
   /       /  \
  A       N   V
 / \     /   /
A   I   L   T
           /
          T
         /
        O
```

- The depth of this tree is 5 since the letter "`O`" is 5 levels away from the root.
- There are 4 leaf nodes (nodes with no kids) - `A`, `I`, `L` and `O`.
- There are 13 external nodes, e.g., right or left child null pointers where a new node could be inserted. The letters `I` on level 1, `N` and `V` on level 2, and both Ts have one available null pointer for a new node, as well as all four-leaf nodes each having two available null pointers, for a total of 13 external nodes.
- The internal path length is the sum of the depths of all internal nodes, the nodes actually in the tree. The nodes `I` and `N` are at level 1, and therefore are each 1 level away from the root, for an internal path length so far of 2. `A`, `N` and `V` are at level 2, for a total of 6, `A`, `I`, `L` and `T` are level 3, totaling 12, and then `T` and `O` add 4 and 5 to the sum, for a grand internal path length total of 2 + 6 + 12 + 4 + 5 = 29.
- The external path length is calculated from the 13 external nodes, 1 at level 2 (right child pointer from the I), 2 each from level 3 (both right child pointers for `N` and `V`) for a total EPL of 6 so far, 7 each at level 4 (both pointers from `A`, `I`, and `L`, plus the right child pointer for `T`) for a level 4 EPL total of 28, plus a level 5 right child pointer from `T`, and two level 6 pointers from `O`. The total external path length is: 2 + 6 + 28 + 5 + 12 = 53.

Input from keyboard a string (word) of uppercase characters with no word greater than 25 letters. Covert to upper case on input. Output to the screen the five integers for each binary search tree resulting from the word, indicating tree depth, number of leaf nodes, number of external nodes, internal path length, and external path length all labeled and formatted appropriately. The program must use a tree data structure. Finally, the program should ask if the user wants to run the program again (Check case). Refer to the sample output below.

Sample Run:

```
Enter word: DISTRICT

Tree Statistics:

Depth:                     4
Leaf nodes:                4
External nodes:            9
Internal path length:     16
External path length:     32

Run Again (Y/N): n
```

Name the program: `TreeStatsXX.cpp`, where `XX` are your initials.

Required Problem- Comprehensive.

(4**). Given a network of up to 26 nodes (named A to Z), where every pair of nodes may be connected or disconnected, write a C++ program to draw the network in a two-dimensional diagram. A node may be connected to at most 4 other nodes. Input from the keyboard the number of nodes in the network then on separate lines input the pairs (edges) with capital letters. Assume the pairs are not sorted and convert to proper case. Output to the screen an ASCII drawing showing the actual links between the nodes. Nodes are given by the capital letters A to Z. Use `'-'`, a dash, for horizontal links and a vertical bar `'|'` for vertical links. The links should have horizontal and vertical lines that do not bend and have a non-zero length. Spaces can be added provided they don't disfigure the picture. Finally, the program should ask if the user wants to run the program again (Check case). Refer to the sample output below.

Sample Run:

```
Enter number of nodes: 7
Enter node pairs (edges):

H C
G H
A B
B F
B C
F G
C D
D A

The network:

A - B ----- F
```

```
|   |       |
D - C - H - G

Run Again (Y/N): n
```

Name the program: `DrawNetworkNodesXX.cpp`, where XX are your initials.

Extra Credit: Implement the following problem. See the 2436 Grading and Submission Guide Sheets for additional grading/submission information. Partial credit will be given. (10 points)

Write a C++ program using recursion to generate a set of squares within a square. Input from the keyboard an integer `n` in the range `[5,20]`. Error check input. Output to the screen a square with `n` uppercase `X's` on each side. Then inside that square will be another square, one cell away on each side. This pattern will continue until there is a single square in the middle, thus making a box of concentric squares. The program must use a recursive function for full credit. Finally, the program should ask if the user wants to run the program again (Check case). Refer to the sample output below.

Sample Run:

```
Enter size [5-20]: 5

XXXXX
X   X
X X X
X   X
XXXXX

Run Again (Y/N): Y

Enter size [5-20]: 10

XXXXXXXXXX
X        X
X XXXXXX X
X X    X X
X X XX X X
X X XX X X
X X    X X
X XXXXXX X
X        X
XXXXXXXXXX

Run Again (Y/N): n
```

Name the program: `RecSquaresXX.cpp`, where `XX` are your initials.