BST Number List in Console/Terminal
Cpt S 321 Homework Assignment
Washington State University

Submission Instructions:

- Create a branch called "Branch_HW1" and work in this branch for this assignment.
- Once you are done and before the deadline, tag the version that you would want us to grade with the assignment number. For example, "HW1", "HW2", etc.
- On Canvas -> Assignments -> Submit the link to your repository (a link to the tag or the branch works) by the HW deadline.
  IMPORTANT: The HW must be tagged by the due date and a link to that tag needs to be submitted via Canvas in order to receive a grade.

Assignment Instructions:

Read all the instructions *carefully* before you write any code.

Create a C# console application that fulfills the following requirements:

1. Get a list of integer numbers from the user on A SINGLE LINE
   - The numbers will be in the range [0,100]
   - The numbers will be separated by spaces
   - You may assume that the user enters a correctly formatted input string that meets these requirements
   - You may use Console.ReadLine or a similar method to get input from the user
2. Add all the numbers to a binary search tree (you must implement your own BST) in the order they were entered
   - Don't allow duplicates
   - Use the Split function on the entered string for easy parsing (split on the space character)
3. Display the numbers in sorted order (smallest first, largest last).
   - Traverse the tree in order to produce this output.
4. Display the following statistics about the tree
   - Number of items (note that this will be less than or equal to the number of items entered by the user, since duplicates won't be added to the tree). Write a function that determines this from your BST, NOT the array returned from the split. In other words, you must have a Count function in your BST implementation (you are not allowed to use any existing implementation for that).
   - Number of levels in the tree. A tree with no nodes at all has 0 levels. A tree with a single node has 1 level. A tree with 2 nodes has 2 levels. A tree with three nodes could have 2

or 3 levels. You should know why this is from your advanced data structures prerequisite course.

- Theoretical minimum number of levels that the tree could have given the number of nodes it contains (figure out the formula to calculate this)

Point Breakdown (10 points total):

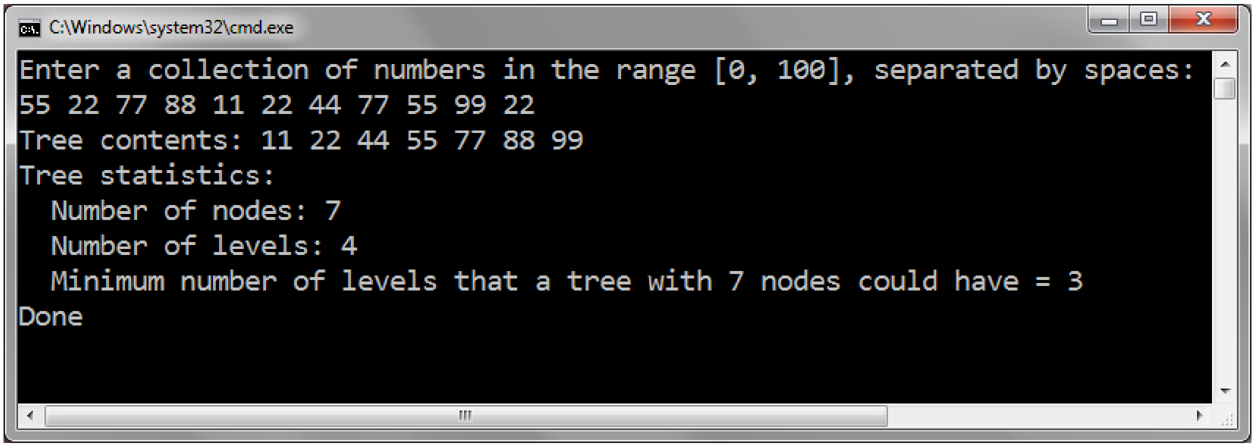
- 5 points: Fulfill all the requirements above with no inaccuracies in the output and no crashes.
- 1 point: For a "healthy" version control history, i.e., 1) the HW assignment should be built iteratively, 2) every commit should be a cohesive functionality, 3) the commit message should concisely describe what is being committed, ~~4) you should follow TDD – i.e., write and commit tests first and then implement and commit the functionality.~~
- 1 point: Code is clean, efficient and well organized.
- 1 point: Quality of identifiers.
- 1 point: Existence and quality of comments.
- ~~1 point: Existence and quality of test cases. Normal cases and edge cases are both important to test.~~

- 1 point: The repository is setup properly

General Homework Requirements	
Quality of Version Control	● Homework should be built iteratively (i.e., one feature at a time, not in one huge commit). ● Each commit should have cohesive functionality. ● Commit messages should concisely describe what is being committed. ~~● TDD should be used (i.e, write and commit tests first and then implement and commit functionality).~~ ~~● Include "TDD" in all commit messages with tests that are written before the functionality is implemented.~~ ● Use of a .gitignore. ● Commenting is done as the homework is built (i.e, there is commenting added in each commit, not done all at once at the end).
Quality of Code	● Each file should only contain one public class. ● Correct use of access modifiers. ● Classes are cohesive. ● Namespaces make sense. ● Code is easy to follow. ~~● StyleCop is installed and configured correctly for all projects in the solution and all warnings are resolved. If any warnings are suppressed, a good reason must be~~

	~~provided.~~ ● Use of appropriate design patterns and software principles seen in class.
Quality of Identifiers	● No underscores in names of classes, attributes, and properties. ● No numbers in names of classes or tests. ● Identifiers should be descriptive. ● Project names should make sense. ● Class names and method names use PascalCasing. ● Method arguments and local variables use camelCasing. ● ~~No Linguistic Antipatterns or Lexicon Bad Smells.~~
Existence and Quality of Comments	● Every method, attribute, type, and test case has a leading comment block ~~with a minimum of <summary>, <returns>, <param>, and <exception> filled in as applicable.~~ ● ~~All comment blocks use the format that is generated when typing "///" on the line above each entity.~~ ● There is useful inline commenting in addition to leading comment blocks that explains how the algorithm is implemented.
~~Existence and Quality of Tests~~	● ~~Normal, boundary, and overflow/error cases should be tested for each feature.~~ ● ~~Test cases should be modularized (i.e, you should have a separate test case for each thing you test - do not combine them into one large test case).~~ ● ~~*Note: In assignments with a GUI, we do not require testing of the GUI itself.*~~
The repository is properly setup	● Private ● All TAs and the instructor are added as Maintainers ● The readme contains name and ID

Sample Output:



```
Enter a collection of numbers in the range [0, 100], separated by spaces:
55 22 77 88 11 22 44 77 55 99 22
Tree contents: 11 22 44 55 77 88 99
Tree statistics:
  Number of nodes: 7
  Number of levels: 4
  Minimum number of levels that a tree with 7 nodes could have = 3
Done
```