Task 2: Experiments (30%)

The purpose of this experiment is to highlight this striking difference in running time, using methods you developed in task1 to create a skewed BST and a (roughly) balanced BST, both with a large number of nodes, and measuring the worst-case execution time of searching on each. To do that, we will assume that the order in which the numbers are inserted is our choice, to produce the desired shape, and we will search for a number that is not in the tree, to force the worst case.

1. Estimate the asymptotic running time of searching in a skewed BST and a balanced BST. Fill the following table with the big-O running times of each case. skewed tree balanced tree.

	Skewed BST	Balanced BST
search	O(N)	O(LogN)

2. Using the `skewedTree()` method you developed in task1, write some code to finish the following experiments. You can simply copy `BST.java` over and rename the file to `Experiments.java` and modify the `main()` in the file. Fill in the following chart with the times in nanoseconds measured. You may need to adjust the values of n according to your platform. That is, if your program takes too long to complete, or if you run out of memory, etc., reduce the range of n as needed. If on the other hand, your program completes the execution fast, increase the range of n as much as possible to observe behavior for large n. In any case, make sure you have at least 5 columns of data. (**No points if you have less than 4 columns of data.**)

Search	n=200	n= 400	n= 600	n= 800	n= 1000
Skewed BST (nanoseconds)	43847	55453	66828	69791	74897

3. Explain how the growth of these measurements compares with your big-O running time. If your measurements do not match your conjecture, investigate the reason by looking carefully at the code of the BST class, and explain what happened. (No points if you do not say something like “when n is multiplied by

xxx the running time is yyy". Partial points if your measurements do not match and you say it, but you do not explain what you did to investigate the reason.)

My conjecture big-O of `skewedTree()` is O(N) and the measurements big-O is O(N). The runtime is increased when the input is increased. For the input, the biggest number that could run is about 1000. If the inputs are too big, they will run out of memory.

4. Using the `balancedTree()` method you developed in task1, write some code to finish the following experiments. You can continue to modify `main()` in `Experiments.java`. Fill in the following chart with the times in nanoseconds measured. You may need to adjust the values of n according to your platform. That is, if your program takes too long to complete, or if you run out of memory, etc., reduce the range of n as needed. If on the other hand, your program completes the execution fast, increase the range of n as much as possible to observe behavior for large n. In any case, make sure you have at least 5 columns of data. (**No points if you have less than 4 columns of data.**)

Search	n= $5,000$	n=10,000	n=15,000	n= $20,000$	n=25,0000
Balanced BST (nanoseconds)	33413	38260	43707	51103	56247

5. Explain how the growth of these measurements compares with your big-O running time. If your measurements do not match your conjecture, investigate the reason by looking carefully at the code of the BST class, and explain what happened. (No points if you do not say something like "when n is multiplied by xxx the running time is yyy". Partial points if your measurements do not match and you say it, but you do not explain what you did to investigate the reason.

   My conjecture big-O of `balancedTree()` is O(LogN) and the measurements big-O is O(LogN). The runtime is increased when the input is increased. For the `balancedTree()` method, it could insert bigger numbers up to 30,000.