**Objective**:

To compare the Data Structures: Binary search tree and Red black tree for two input random sequences (random numbers between 1 and 100 inclusive) each of length 500 and empirically determine the Height of both the trees (average case for searching operation) and average number of comparisons to be done for a successful search (worst case scenario).

**Procedure:**

1. Create a datastructures program package with one class (CompareBSTnRBT.java) in Eclipse
2. Determine the value of height and average comparisons for each data structure (binary search tree and red black tree) and for each sequence.
3. Check which data structure performs well for different sequences.

**Technology used:** Java

**Tool used:** Eclipse

**Source Code Filename:** CompareBSTnRBT.java

**Explanation:**

Data structures Binary search tree and Red Black tree maintains the binary search tree property i.e., given any node the left child is less than the node and the right child is greater than the node. In our case we will increment the count field if we encounter a number that is already inserted as a node in the tree. The main difference between the trees is that Red Black tree is a self-balancing tree whereas Binary search tree is not. When a part of the input sequence is already sorted then the binary search tree will form long chains of nodes that can cause searches to take linear time, but Red black tree being a self-balancing tree guarantees a search operation in logarithmic time

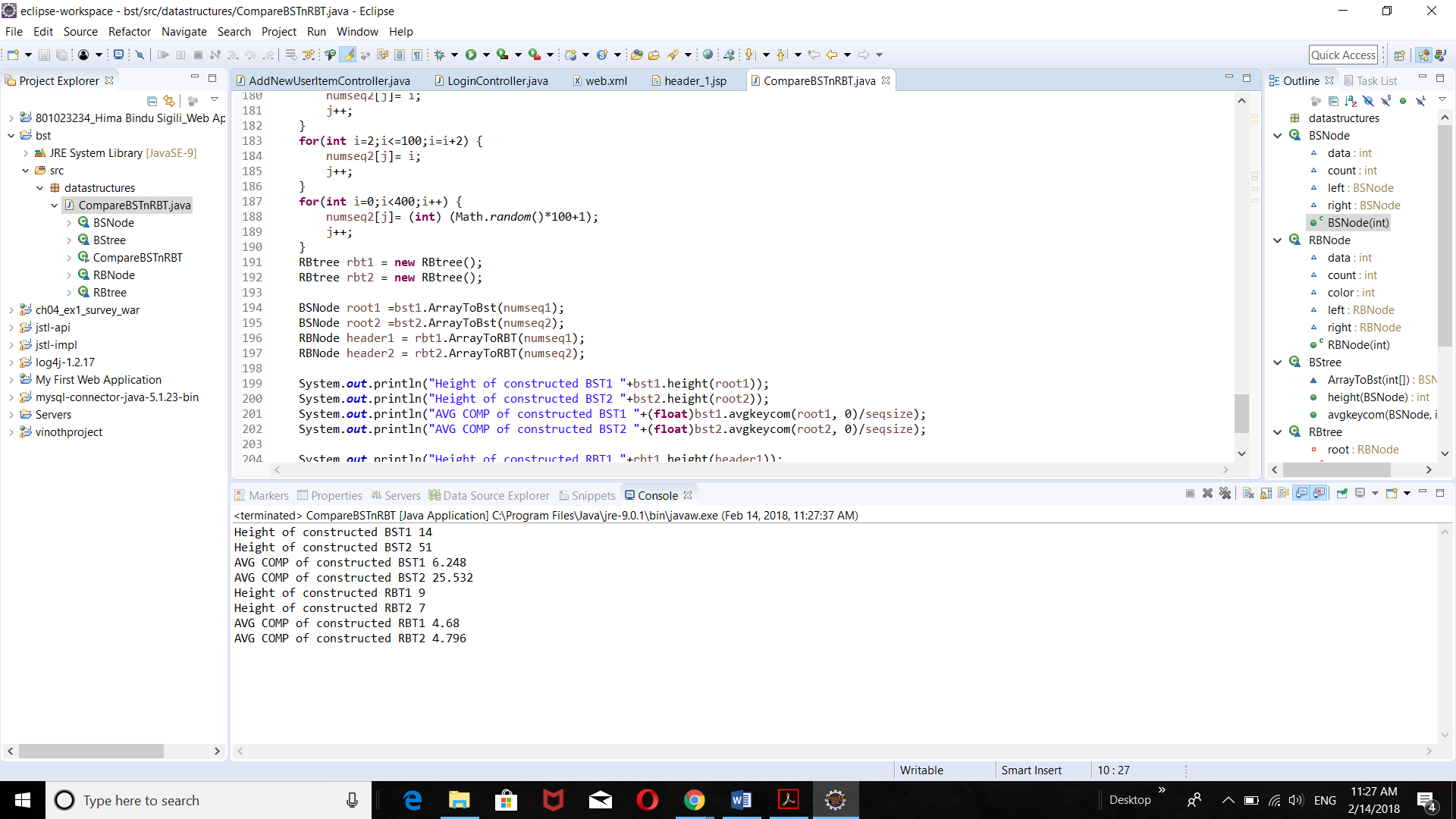
The Comparison of Binary search tree and Red black tree program is explained below.

**Program Explanation:**

1. Create a java package “datastructures”
2. Create a class CompareBSTnRBT with a main method and four classes one for Red Black tree node(with data, left child, right child, color and count as attributes) and one for Binary search tree node(same attributes as Red black node except the color field) and one for the insert operation for Red Black tree( having height and average key comparisons methods ) and a similar one for Binary search tree.
3. Create the two input sequences, sequence 1 will be the 500 random numbers between 0 and 100(inclusive) and sequence 2 will be the first 50 odd numbers followed by first 50 even numbers followed by 400 random numbers between 1 and 100 inclusive.
4. ArraytoBST method iteratively calls the insert method and the insert method recursively calls itself and performs the insertion of nodes into the binary search tree. Similarly, the Red black tree has ArraytoRBT method to perform the insertion into the red black tree.
5. Finally, I printed the results to console. The tree for which the fewest number of average key comparisons is considered as the effective balancing data structure.

**Screen Shots of output:**

**For an instance of random sequences:**



**Observations:**

As per the tabular column below, the minimum number of average key comparisons for the second sequence is for the Red Black tree. As the sequence has more sorted values long chains of node will be formed in the binary search tree resulting in the large height (linear time complexity n) and as the red black tree is self-balancing the height will be maintained (time complexity nlogn) and is effective in the worst case and average case.

For Sequence 1:

Height and Average key comparisons for successful search of Binary search tree (Average case and worst-case scenarios) is similar to that of Red black tree.

For Sequence 2:

Height and Average key comparisons for successful search of Binary search tree (Average case and worst-case scenarios) is greater than that of Red black tree.

From the above observation we can say that Red Black trees are more efficient in search operations compared to the Binary search trees.

**Tabular column:**

|  |  |  |  |
| --- | --- | --- | --- |
| Sequence | Data Structure | Height | Average Comparisons  For successful search |
| 1 | Binary Search Tree | 15 | 7.19 |
| 1 | Red Black Tree | 8 | 4.77 |
| 2 | Binary Search Tree | 51 | 25.094 |
| 2 | Red Black Tree | 7 | 4.784 |

**Conclusion/Inference:**

From the above observation it can be inferred that for the second sequence the self-balancing tree Red Black tree is more efficient than the binary search tree.