Ans 1) For the Red black BST code, the code with more branches makes it difficult to achieve 100% branch coverage. However, in the process of chasing 100% branch coverage, it really makes the developer think about the infeasible code paths, ultimately improving the quality of the code. Following this, methods consisting of balance(), moveRedRight(), moveRedLeft(), flipColors(), rotateLeft(), and rotateRight() are only accessible using other methods like delete() or put() thus, making testing difficult.. With the code structure having thought out private methods which are accessible via public interface or methods, the testability of target individual paths is easier than that in a not well-structured code. Following this, methods consisting of get(), put, deleteMax(), delete(), deleteMin(), min(), max(), floor(), ceiling(), rank(), and keys() follow public interface and private method to perform their functionality.

Ans 2) The use of Java assert statements to implement precondition checks and object invariants are consistent to check the requirements of a left-leaning Red-Black Tree. The code has an assert check() method branch at the end of methods which can hurt the requirements of the left-leaning Red-Black tree such as delete, put, etc thus, maintaining consistency. Also, having the assert check() method, it is boosting the confidence of the developer in making a less error-prone program. With the assert statement checked at the end of every error-prone method, it is taking some responsibility off from the developer to test certain integral requirements of the Red-Black Tree. Hence, helping in testing the program.

Ans3) With the assumption that the given code is part of a larger application, usually, the closed-box testing is done without having any knowledge of the inner working of the code. This is usually achieved so have an external interface that abstractly sits on the top of the main code and makes all calls to the main code. However, we have written the test with the prior knowledge of code in our hands, and tests themselves are written to test different paths and branches rather than just checking the correctness of the output. Hence, the objectives of tests written with the mentioned methods are aligning with that of the open-box testing.