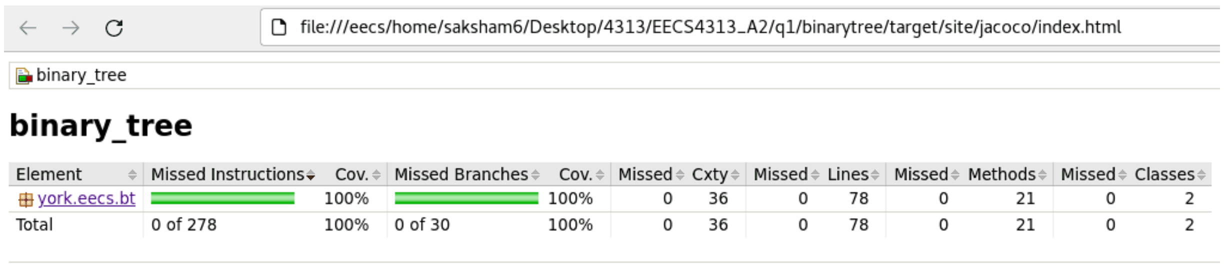
Question 1

Here we will see the screenshot and coverage for the three parts:

a) **Writing the Test cases manually** - I was able to achieve 100% coverage for the code, trying different types of inputs and so on.



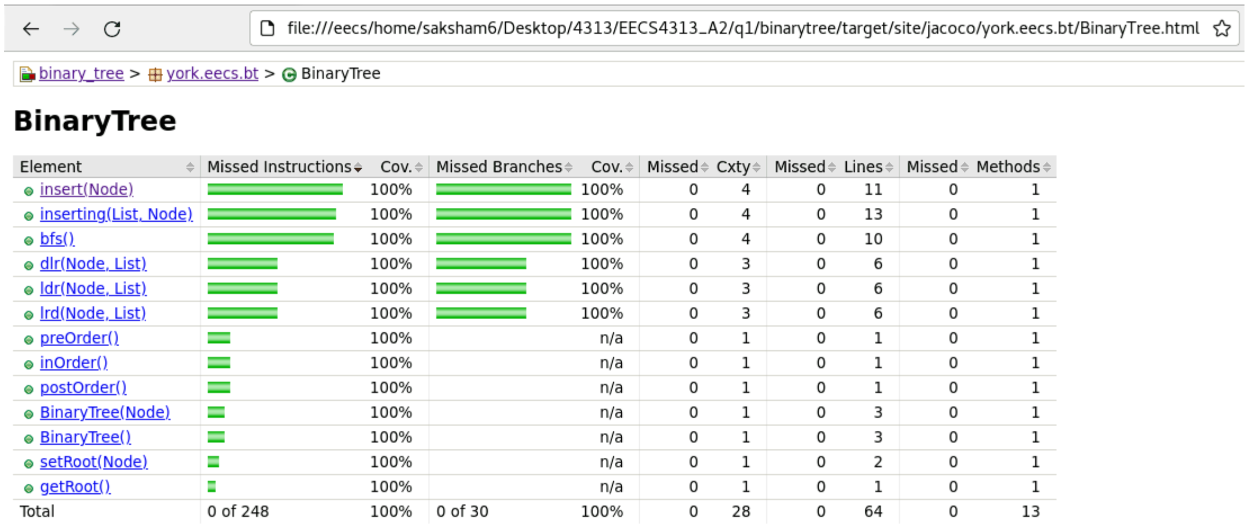
file:///eecs/home/saksham6/Desktop/4313/EECS4313_A2/q1/binarytree/target/site/jacoco/index.html

binary_tree

binary_tree

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
york.eecs.bt	🟩	100%	🟩	100%	0	36	0	78	0	21	0	2
Total	0 of 278	100%	0 of 30	100%	0	36	0	78	0	21	0	2

Further unfolding the BinaryTree class.



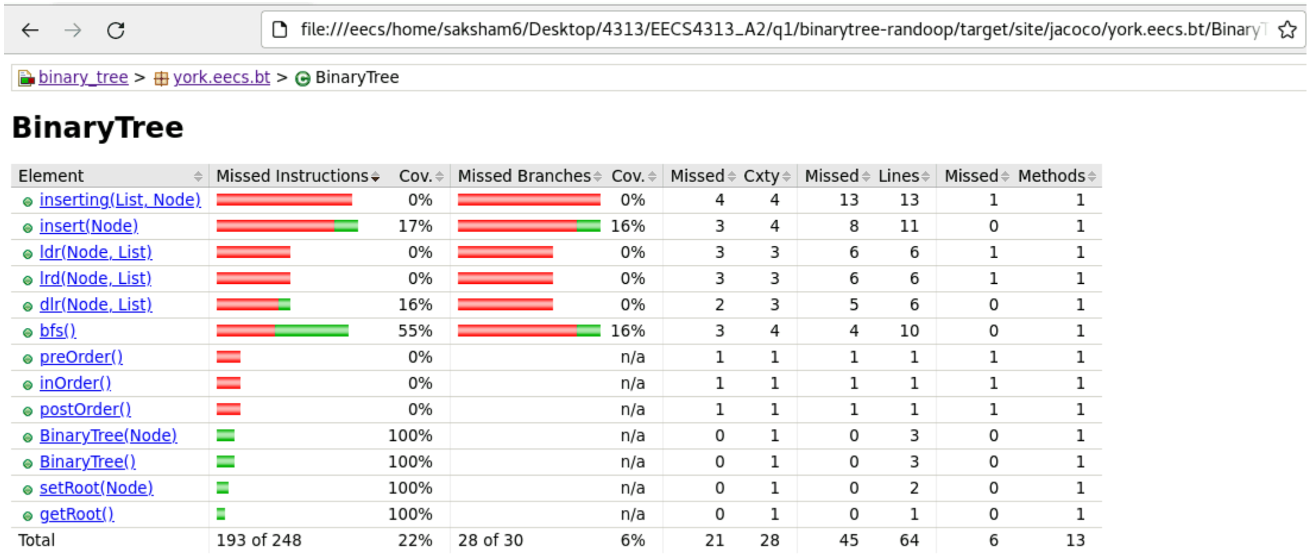
file:///eecs/home/saksham6/Desktop/4313/EECS4313_A2/q1/binarytree/target/site/jacoco/york.eecs.bt/BinaryTree.html

binary_tree > york.eecs.bt > BinaryTree

BinaryTree

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
insert(Node)	🟩	100%	🟩	100%	0	4	0	11	0	1
inserting(List, Node)	🟩	100%	🟩	100%	0	4	0	13	0	1
bfs()	🟩	100%	🟩	100%	0	4	0	10	0	1
dlr(Node, List)	🟩	100%	🟩	100%	0	3	0	6	0	1
ldr(Node, List)	🟩	100%	🟩	100%	0	3	0	6	0	1
lrd(Node, List)	🟩	100%	🟩	100%	0	3	0	6	0	1
preOrder()	🟩	100%		n/a	0	1	0	1	0	1
inOrder()	🟩	100%		n/a	0	1	0	1	0	1
postOrder()	🟩	100%		n/a	0	1	0	1	0	1
BinaryTree(Node)	🟩	100%		n/a	0	1	0	3	0	1
BinaryTree()	🟩	100%		n/a	0	1	0	3	0	1
setRoot(Node)	🟩	100%		n/a	0	1	0	2	0	1
getRoot()	🟩	100%		n/a	0	1	0	1	0	1
Total	0 of 248	100%	0 of 30	100%	0	28	0	64	0	13

b) **Using Randoop** – After using randoop we were only able to achieve 22% of coverage in the code, which was not very desirable using an automated test generation suite. Overall was 19% coverage.



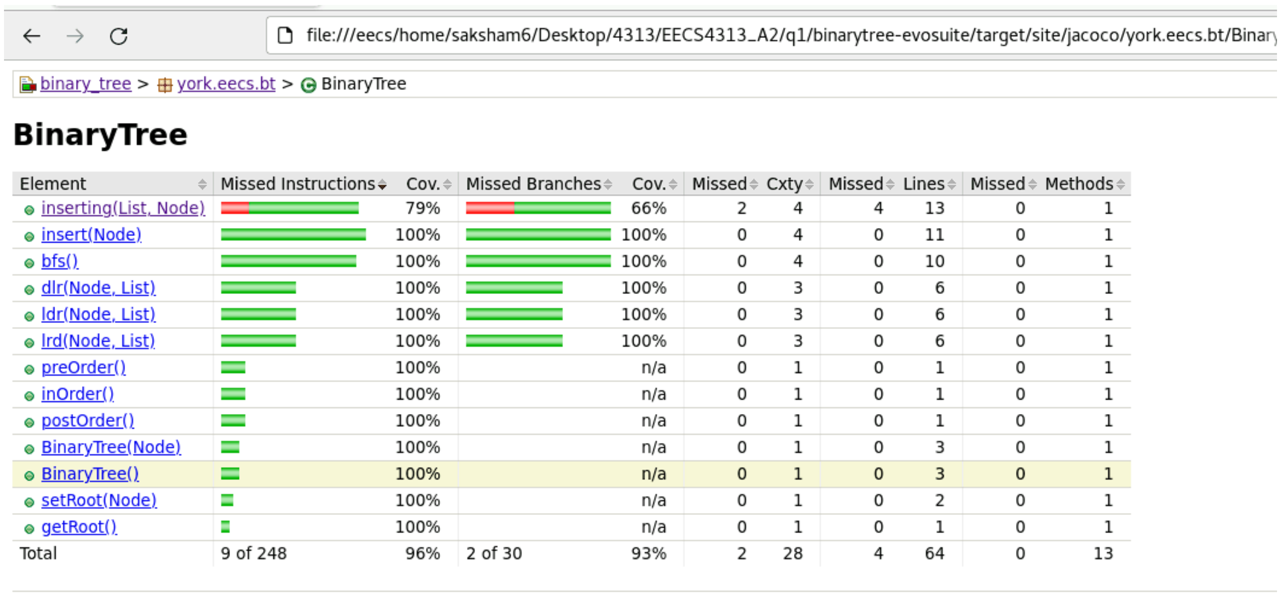
file:///eecs/home/saksham6/Desktop/4313/EECS4313_A2/q1/binarytree-randoop/target/site/jacoco/york.eecs.bt/Binary

binary_tree > york.eecs.bt > BinaryTree

BinaryTree

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
inserting(List, Node)	🟥	0%	🟥	0%	4	4	13	13	1	1
insert(Node)	🟥🟩	17%	🟥🟩	16%	3	4	8	11	0	1
ldr(Node, List)	🟥	0%	🟥	0%	3	3	6	6	1	1
lrd(Node, List)	🟥	0%	🟥	0%	3	3	6	6	1	1
dlr(Node, List)	🟥🟩	16%	🟥	0%	2	3	5	6	0	1
bfs()	🟥🟩	55%	🟥🟩	16%	3	4	4	10	0	1
preOrder()	🟥	0%		n/a	1	1	1	1	1	1
inOrder()	🟥	0%		n/a	1	1	1	1	1	1
postOrder()	🟥	0%		n/a	1	1	1	1	1	1
BinaryTree(Node)	🟩	100%		n/a	0	1	0	3	0	1
BinaryTree()	🟩	100%		n/a	0	1	0	3	0	1
setRoot(Node)	🟩	100%		n/a	0	1	0	2	0	1
getRoot()	🟩	100%		n/a	0	1	0	1	0	1
Total	193 of 248	22%	28 of 30	6%	21	28	45	64	6	13

c) **Using Evosuite** - After using Evosuite we were able to achieve 96% coverage in the BinaryTree code, which is a desirable result for an automated test generation suite. Overall was 95% coverage.



file:///eecs/home/saksham6/Desktop/4313/EECS4313_A2/q1/binarytree-evosuite/target/site/jacoco/york.eecs.bt/Binar

binary_tree > york.eecs.bt > BinaryTree

BinaryTree

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
inserting(List, Node)	🟥🟩	79%	🟥🟩	66%	2	4	4	13	0	1
insert(Node)	🟩	100%	🟩	100%	0	4	0	11	0	1
bfs()	🟩	100%	🟩	100%	0	4	0	10	0	1
dlr(Node, List)	🟩	100%	🟩	100%	0	3	0	6	0	1
ldr(Node, List)	🟩	100%	🟩	100%	0	3	0	6	0	1
lrd(Node, List)	🟩	100%	🟩	100%	0	3	0	6	0	1
preOrder()	🟩	100%		n/a	0	1	0	1	0	1
inOrder()	🟩	100%		n/a	0	1	0	1	0	1
postOrder()	🟩	100%		n/a	0	1	0	1	0	1
BinaryTree(Node)	🟩	100%		n/a	0	1	0	3	0	1
BinaryTree()	🟩	100%		n/a	0	1	0	3	0	1
setRoot(Node)	🟩	100%		n/a	0	1	0	2	0	1
getRoot()	🟩	100%		n/a	0	1	0	1	0	1
Total	9 of 248	96%	2 of 30	93%	2	28	4	64	0	13

d) **Comparison** – Out of the three I believe that Evosuite provided us with a good result depending how long it took to run and provide us a coverage result and how long it took to write test cases manually. Randoop was not good to use in this case due to certain constraints as it is feedback guided. But in any case, Junit Manual cases gave us 100% coverage which is still the best case possible.
Readability – of the code I believe is only concerned with the manually written code as the code generated are all written in a particular format making it easier to understand, but manually written code can be poor, without proper comments. Every programmer is recommended to write properly commented and documented code.
Effectiveness – For the 3 different cases we see that manually effective code is the most effective here as we achieve 100% code coverage and test with different types of input a user will be allowed to/ or will be using. But in case a user wants to save time and have the resources to give, it is recommended that to generate the test cases using Evosuite.
Usefulness – I believe Evosuite will be the most useful in this case to provide us with the quickest and easier test generation.
But we must keep in mind that depending on the specific use case for the code the best way to test out any functionality would be writing your own test cases to exactly understand the behavior of it.