Code for Encode method :

public String encode(String word)

{

StringBuffer code = new StringBuffer(""); // 1

for (String c: word.toLowerCase().split("")) //word length

{

if (morseLettersTree.contains(c)) //n

{

CodeVisitor codeVisitor = new CodeVisitor(c); //1

morseLettersTree.postOrderTraversal(codeVisitor); //n

code.append(codeVisitor.letterCode()); //1

}

else if (c.equals(" ")) //1

code.append(" "); //1

}

return code.toString().trim(); //3

}

Analysis:

Encode is O(n2) because if we add up all of the operation costs noted in the code :

O(n2 + word.length() + 2n + 6)

The biggest factor is n2.

Code for Decode method:

public String decode(String code)

{

StringBuffer word = new StringBuffer(); //1

for (String w: code.split(" ")) //Every word

{

for (String l: w.split(" ")) //Every letter

{

BinaryTreeNode<String> letterNode = morseLettersTree.root(); //1

for (String s: l.split("")) //Every character

{

if (s.equals(".")) //1

letterNode = letterNode.leftChild(); //1

else if (s.equals("-")) //1

letterNode = letterNode.rightChild(); //1

}

word.append(letterNode.element()); //1

}

word.append(" "); //1

}

return word.toString().trim(); //1

}

Analysis:

Since n is never iterated through, and we only iterate through the lengths of the words, the big O for this is O(1).