Question 2

Part a)

1. Function called once
2. `
3. Local variable n assigned to input argument once
4. Initialise result once
5. Initialise i once
6. Compare i against n n+1 times
7. Increment i n times
8. Perform operation X n times
9. Assign result n times
10. Return result 1 time

T(n) = 1 + 1 + 1 + 1 + 1 + n+1 + n + nX + n + 1

T(n) = nX + 3n + 7

Part b)

Results:

T(100): 9.051501E-6 seconds

T(1000): 4.3421222E-4 seconds

T(10000): 0.0348994561 seconds

The algorithm is O(n^2) due to the nature of strings being immutable.

We can best demonstrate this using T(100), T(1000) and T(10000). If the algorithm is O(n) then we should expect a 1:10 runtime ratio between each subsequent runtime, for O(n^2) we expect 1:100 and O(n^3) we expect 1:1000.

T(100):T(1000) is aprox 1:50 and T(1000) :T(10000) is aprox 1:80. Both of these measurements are in the same order of magnitude as expected for O(n^2) and we can conclude the algorithm is O(n^2) – the slight differences will be based on the linear and constant parts of the equation.

Part C)

T(100): 2.019324E-6 seconds

T(1000): 1.623322E-5 seconds

T(10000): 1.699821E-4 seconds

The algorithm is O(n) due to the nature of stringbuffer being able to append a string in O(k) time where k is the length of the string to be appended.

We can best demonstrate this using T(100), T(1000) and T(10000). If the algorithm is O(log(n)) then we should expect a 1:2 runtime ratio between each subsequent runtime, for O(n) we expect 1:10 and O(n^2) we expect 1:100.

Ratio of T(100):T(1000) is aprox 8 and T(1000):T(10000) is aprox 10. Both are in the order of magnitude expected from an O(n) algorithm and we can conclude the algorithm is O(n).