Part1. B

Each time when the function is calling recursively, we are either decreasing value of ‘street’ or ‘avenue’ by 1, continuing till one of these variables reaches 1, which is the base case. So, each time when the function is calling itself, we have 2 subproblems ‘street-1’ and ‘avenue-1’. On the other hand, the total number of recursive calls is the sum of initial values of street and avenue.

Each time, each subproblem is divided into 2 other subproblems. The total number of subproblems will be 2^(street+avenue)-1 and there is why:

The number of subproblems at each level will be like 2^, 2^2,…2^(street+avenue-1), which created series of geometric series.

By using the formula of sum of a geometric series, the total number of subproblems will be:

2^0+2^1+2^2+…+2^(street+avenue-1)= 2^(street+avenue)-1

Thus, the time complexity of numPathsFromHome function is O(2^(street+avenue)), which is an exponential function that for the bigger values for street and avenue the function exponentially becomes very big.

Part2.B

In each recursive call, the function checks the items of array at indices ‘indexInS’ and ‘indexInT’, so there is comparison.

In the best case scenario, S is a subsequence of the first part of T, so the number of times the recursive function has been called is equal to the length of S.

In the worst case scenario, S is either not he subsequence of T or it is a subsequence of the end part of the T, so we need to traverse all the T and the number of comparisons (function is called) is equal to the length of the T.

Thus the recursive calls are made based on the lengthof the arrays S and T. and the time complexity is like O(max(lengthofS, lenghtofT)). Based on the formula we can also say that the average case time complexity is O(max(lengthofS, lenghtofT)/2)

In fact the number of comparisons (number of tiems function is called) depends on the length of T and S.

We can say that the time complexity of function ‘hasSubsequence’ is linear time. The running time of the function increases linearly based on the length of the inout. The length of the array increases the number of comparison and recursive calls also will increase