Task 1

Function **getMessage** has time complexity of O(1) as all operations are constant and has space complexity of O(1) as space taken does not depend on input size.

Function **messageFind**:

First for loop has time complexity of O(m).

Second for loop runs n times. It has a second for loop inside that also runs m times. Hence this for loop has time complexity of O(nm).

The while loop runs a maximum of n + m times (pointer starts at bottom left and can only go up or left), hence has time complexity of O(n+m).

Inbuilt strip and split both do not exceed O(n+m) (these operations are performed on the text contained in encrypted.txt).

All other operations are constant. Hence, the function has a time complexity of O(nm).

The function has a space complexity of O(nm) since the largest variable that depends on input size is the grid ‘arr’ which is of size n * m.

Task 2

Function ‘**cord’** has time complexity of O(1) as all operations are constant regardless of input size. Space complexity is also O(1) as size of variables does not depend on the input either. The function uses char as the index to retrieve the corresponding character in lstChar.

Function ‘**wordSort’** uses radix sort to sort all items. First 0’s are added at the end of all strings to make them all the same size.

Calculating m has time complexity of O(N) as the for loop runs entire length of list and in each iteration constant number of operations operations are performed.

Creating ‘lst’ (adding 0’s at the end of all words) has complexity of O(MN) as a for loop runs entire length of the list and a while loop inside the for loop runs a maximum of m times in each iteration of the for loop. The space complexity does not exceed O(MN).

Performing radix sort has a time complexity of O(MN). The outer for loop runs m times. The first for loop inside the outer loop (to create lstCount) performs constant number of operations n times. The second for loop (to create lstIndex) performs constant number of operations len(lstCount) times, which is a constant number of operations since length of lstCount is 27 regardless of the input. The third for loop (to create output) performs constant number of operations n times (for each item in lst). There are also other constant operations inside the for loop. Since the outer loop runs m times, and two inner loops run n times while one of them performs constant number of operations, time complexity for performing radix sort is O(MN). Once these are created, it uses these lists to create output. As output is updated, the corresponding index in lstIndex is incremented to represent the new position. Finally, lst becomes output and output is reset.

The outer loop works backwards from m. In each iteration, it creates lstCount (which holds frequency of each character) and lstIndex (which holds the index for each character).

Finally, all 0’s are removed at the end. The outer loop runs n times (length of output). Inside the outer loop is a while loop that performs constant number of operations a maximum of m times for each i. Hence, time complexity for removing 0’s is O(MN).

Space complexity for the entire function would be O(MN) as largest input dependant variables are lst and output which contain N words, each with a maximum length of N.

Function **wordBreak**:

First for loop runs N times, hence has time complexity of O(N).

Second and third for loops run N times, and have a while loop inside that runs maximum of M times, hence both have time complexity O(NM).

Fourth for loop runs K times, hence has time complexity of O(K).

Fifth for loop runs N times. It has a while loop that runs maximum of K times. The while loop has another for loop that runs M times. There is another for loop inside the while loop (only runs if certain conditions are met) that runs M times. There is one final for loop in the while loop that runs M times. Hence, the time complexity for this entire nested loop comes down to O(NKM).

Sixth and seventh for loops run K times, hence have time complexity of O(K).

The function uses strip and split functions outside of the loops, which do not exceed the maximum time complexities.

The time complexity for the entire function comes down to O(KMN).

The space complexity for the entire function would be O(NM + K) since all variables that depend on input size are either of size M, K or N*M.