# Morse Code – Dale Chetney

“If you can read this, the code is working. Well done and party on.”

## How it works –

Create a list of ***possibilities***, which contain a ***sentence*** and a ***likelihood***  
call a recursive helper method, passing in the message ***toGo*** and an empty string ***soFar****.*

Call a likelihood check, passing in ***soFar****.*

Split***soFar*** into words. For each ***word*** a ***likelihood***is multiplied by that ***word’s*** frequency with the following ***word*** divided by some large number. That ***likelihood*** is returned.

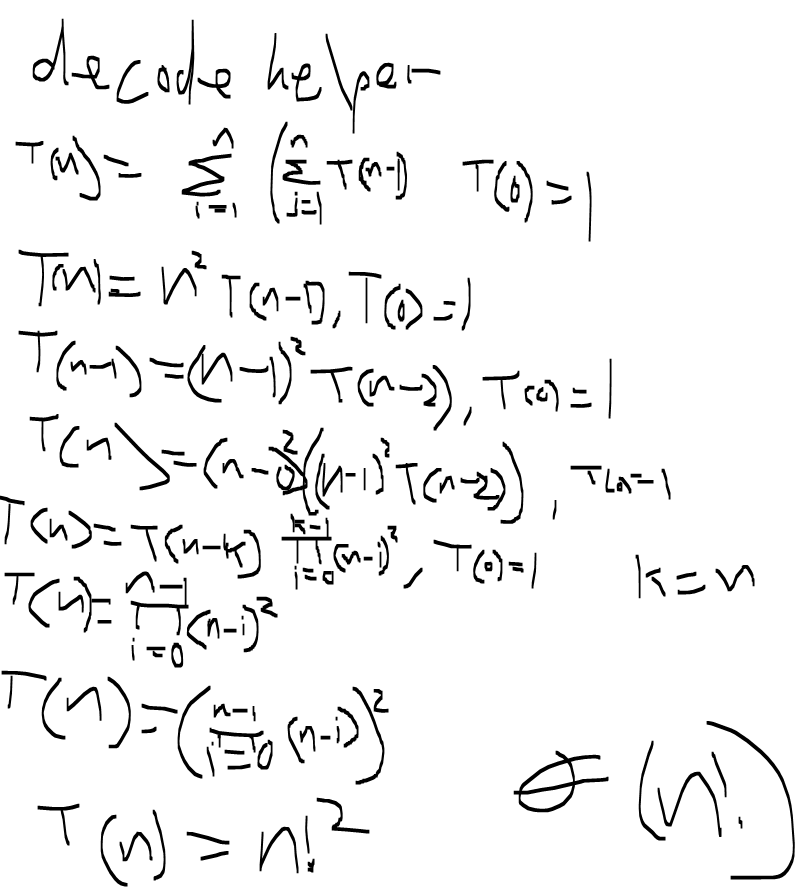
After storing the returned ***likelihood***, check to see if ***toGo*** is empty.   
 If it is, check to see if the ***likelihood*** is greater than some threshold.   
 If it is, add a new ***possibility***to the list with the current ***likelihood****.* Set ***soFar*** as   
 its ***sentence***.   
 Either way, return void.  
If ***toGo*** isn’t empty, check if the ***likelihood***is less than some threshold  
 If it is, return void.  
Loop through an ***index*** that goes from 1 to the length of ***toGo***+ 1, each time splitting ***toGo*** along that ***index***. The dictionary generates ***words*** from the first part, and for each of those ***words*** recurse, passing in the second part and a new string that is ***soFar***+ the ***word*** + a space.

Use quicksort to sort the list of ***possibilities*** by their ***likelihood*** create a list of strings and store the ***sentences*** of the ***possibilities*** in it. Return the list.

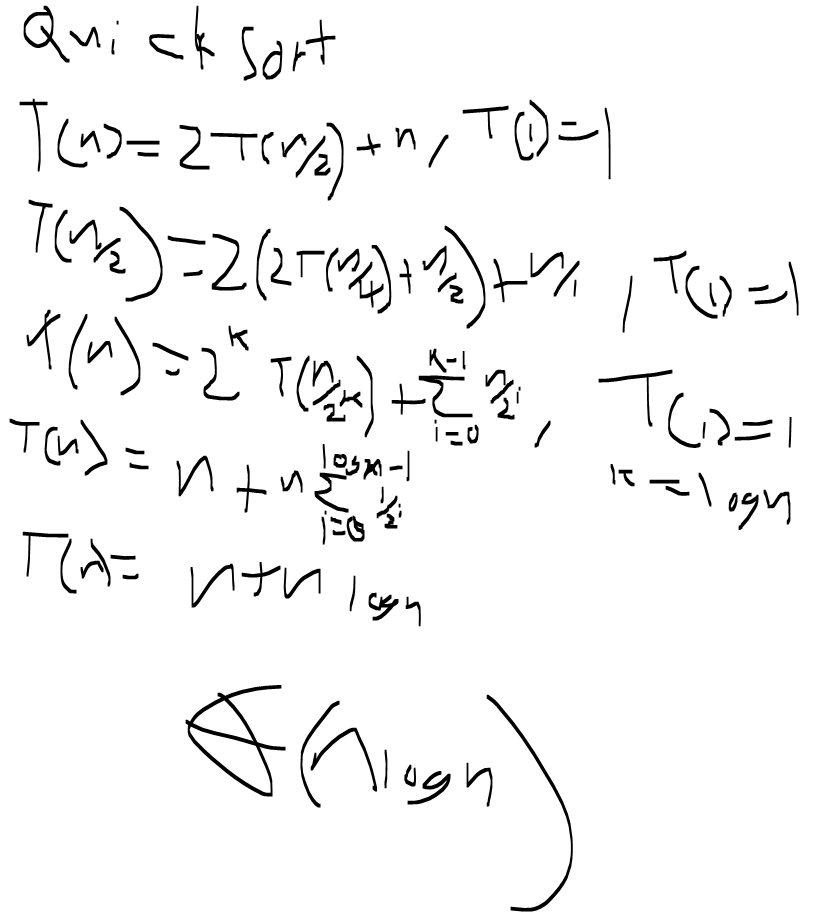
## Help –

I initially had no idea what to do, so I went to my friend Luis for help. Luis also had no idea what to do. As we were discussing our predicament, Bastien asked if we needed any help. We most certainly did. Without showing any code or pseudo-code, Bastien walked us through his approach to the problem. Therefore our solutions are likely quite similar.

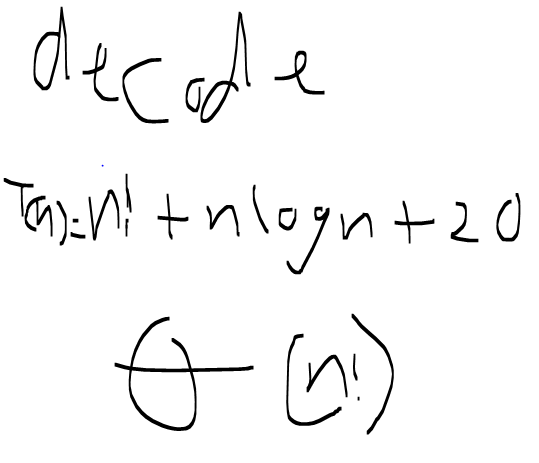
## Performance Calculations –



Therefore, because it is Θ(n!), it must be O(n!)



Therefore, because it is Θ(nlogn), it must be O(nlogn)



Therefore, because it is Θ(n!), it must be O(n!)

## Performance Tests

Messages were as follows:  
“Well done”  
“Show me the money”  
“You can’t handle the truth”  
“Life is like a box of chocolates”

Precision modifier is the float that decode() takes. It’s used as the dividend in likelihood calculations, and 10/the modifier is the threshold for pursuing and recording branches.

Possibilities generated is taken from the length of the possibility array before the top 20 is returned.

Time is given by recording the current time before the algorithm and subtracting it from the time after the algorithm. It does not take into account the time it takes to create the decoding dictionary.

|  |  |  |  |
| --- | --- | --- | --- |
| Message Length | Precision Modifier | Possibilities Generated | Time in milliseconds |
| 21 | 1000 | 119 | 156 |
| 21 | 10000 | 1 | 125 |
| 34 | 1000 | 1009 | 359 |
| 34 | 10000 | 21 | 141 |
| 53 | 1000 | 17705 | 8642 |
| 53 | 10000 | 99 | 296 |
| 77 | 1000 | 62046 | 81355 |
| 77 | 10000 | 1 | 187 |

Based on these results, it’s clear that the length of the message is of little consequence. Instead everything rests on the algorithm’s ability to identify and eliminate bad branches. This means that if I were to try and improve the algorithm, I would look for a more sophisticated likeliness identifier than bigrams.