COMP3010 Ass2 Task 1

a. Since we assume to be working from left to right, we end up with the problem being Cost(word[1,3]), following the given algorithm, we end up with (3-1+1) which will just = 3. Therefore the cost of the first break will = 3, and the cost of the second break leaves us with 7-3 characters remaining. This means that the second subproblem would be Cost(word[4,7]) which = (7-4+1) leaving us with a cost of 4. The most efficient way of retrieving our desired break is to therefore break left to right as the cost is only 3.

b.

```
fastestBreak = (str, break, n1=str.length, n2=break.length) {
    //if the string is empty, return
    If(n1 == 0 || n1 < n2)
        Return 0;
    If(str.substring(0, break.length == break) {
        //remove substring from original and call function with new
        numbers
        Return fastestBreak(str.substring(break.length), break) + 1;

    }

    //call same function with one less letter than the original string

    Return fastestBreak(str.substring(1), break);

}
```

c. If we use any value where i>b>=j, or simply put, any i>j
    a. This would be for example if we used Cost(word[4, 3])

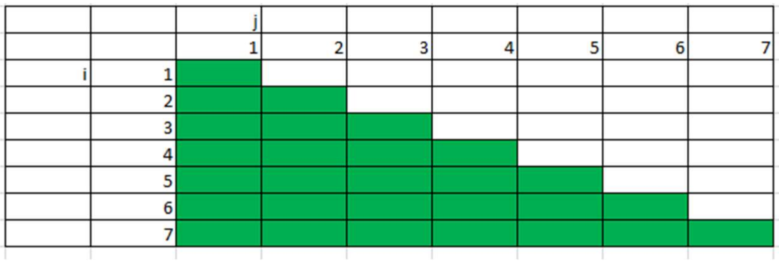
d. Assuming we are going left to right, this would mean that Cost(word[1,1]) would give the lowest cost because the result would be (1-1+1) = 1 and since this algorithm is running once, no recursive call will be made

e. I would be the rows, j would be the columns and breakList would be the value stored for the length of the breaks that have been made and what will be used to compare in the function. I is the rows because when you start at word[I, j] you are staying at index I and searching through till you reach j (which would be the columns) and then you increment I afterwards. This results in a left to right scan through our dynamic table.

f. Say we already have the value of word[4, 6] in our table stored, using the lookup table if we were given the task of working out [4,7] we would simply find the result of [4,6] in the table and do one more computation (that is all it would take to go from index j=6 to j=7), this is because effectively it stores all previous calculations in the result and we only need that one more calculation to run to get the end result, thus saving a lot of time.

g. Every I that is <= j will have a non 0 entry, which would look like this in a table (green represents non zero)

    a.



		j						
		1	2	3	4	5	6	7
i	1	🟩						
	2	🟩	🟩					
	3	🟩	🟩	🟩				
	4	🟩	🟩	🟩	🟩			
	5	🟩	🟩	🟩	🟩	🟩		
	6	🟩	🟩	🟩	🟩	🟩	🟩	
	7	🟩	🟩	🟩	🟩	🟩	🟩	🟩