Comp3010 Assignment 2

Task 1

(a)

Since "Envelope" is 8 character word meaning it has indexes of 0 being the first character of the word and 7 being the last character. The breaks 1 and 3 from breaklist indicate the subproblems that will be created i.e.

Word[i, b], word[b+1, j]

For break 1 the subproblems created are:

Word[0, 1], word[2, 7]

For break 3 the subproblems created are:

Word[0,3], word[4, 7]

However, it is important to note that depending on the ordering of the breaks to compute the total cost the cost and indexing of each break can vary i.e. choosing to break from 1,3 or 3,1.

When BL = {1,3} the total cost of the breaks are:

8 + 6 = 14 with the resulting substrings of "en", "ve", "lope" which is the cost of word[0,7] = 8 and adding the cost of the remaining substring which is calculated by (cost(str) – cost(str.substring)) thus (8 - 2) = 6 .

When BL = {3,1} the total cost of the breaks are:

8 + 4 = 12 with the resulting substrings of "ope", "en", "vel" which is the cost of word[0,7] = 8 and adding the cost of the remaining substring which is calculated by (cost(str) – (cost(str.substring)) thus (8 – 4) = 4.

Coincidentally for this example it highlights how the optimal cost is calculated from a right to left ordering of the breaks, however this is not necessarily the correct approach for each example. The ordering of breaks performed is dictated by the cost of each break resulting in the lowest totalcost.

(b)

The recursive formula for calculating the totalcost(word[i,j], BL):

Return cost(str[i,j], b) + (cost(str[i,j], b) – cost(str[i,b],b) | | cost(str[b+1, j], b)) + totalcost(str.substring(i, b) | | str.substring(b+1, j) , BL{b})

It could either be str[i, b] or str[b+1, j] chosen depending on which substring returns the better cost.

(c)

Inputs that would result in totalcost returning 0 would be:

- Word[i,j] == EMPTY, would return 0 as it is an empty string that can have no cost associated to it.

- BL == {EMPTY} or have reached the end of the list, an empty BL would return 0 as there is no breaks to be made and once the end of the list is reached we can return 0 to indicate the end and build back up the totalcost.
- Word[i,j] if i is >= to j would return 0 as you can't have a valid where its ending index is less than its starting index and if i and j are equal then you cannot make any breaks upon the string as it would be a single character.
- If i is > b or j < than b would result in 0 as that would not satisfy the cost condition and thus input values of 0 and when the totalcost is calculated would result in a total of 0.

(d)

The simplest non-zero subproblem to calculate the totalcost without any recursive calls would be when there is only a single breakpoint to compute. This is because the cost of the first break is equal to the length of the initial word[i,j] before any breaks are made so:

If BL == 1 idx{

Return str.length;

}

(e)

		Word[0, n]
BL	**b_0** **b_1** **b_2** **.** **.** **.** **.** **.** **b_i**	

Example – "Envelope"

Word[n, j]

		E	**N**	**V**	**E**	**L**	**O**	**P**	**E**
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
b_i	**0**	0	2	3	4	5	6	7	8
	1	0	0	2	3	4	5	6	7
	2	0	0	0	2	3	4	5	6
	3	0	0	0	0	2	3	4	5
	4	0	0	0	0	0	2	3	4
	5	0	0	0	0	0	0	2	3
	6	0	0	0	0	0	0	0	2
	7	0	0	0	0	0	0	0	0

The cost of each potential break is calculated from the formula j – i + 1, thus for the breaks in breaklist = {1,3} would result in the substrings word[i, b], word[b+1, j]. So for break 1 there would be the resulting substrings of word[0, 1] = cost of 2, word[2, 7] = cost of 6 and for break 3 there would

be the resulting substrings of word[0, 3] = cost of 4, word[4, 7] = cost of 4. The totalcost can now be computed with these individual costs.

(f)

A non-zero entry is computed by calling the cost function inside the return statement for totalcost where each time the recursive call is made upon totalcost a substring of str is sent through the function where the cost will be calculated for that string. This will continue until there are no more breaks to be made. The cost is calculated by j-i+1 i.e. 7-0+1 = 8 for word[0,7] in “Envelope”.

(g)

A table entry will be non-zero if it satisfies the condition of i <= b < j where then a value is calculated through the formula (j-i+1). So any index from word[i,j] as long as it meets the conditions and doesn’t satisfy the zero case conditions will have a value as seen in the table entries in e.