2. Breaking a String

For the given problem of breaking a string, small-ish subproblems would be to find out the cost of breaking even smaller strings into parts. The trivial case is the substring where no cut is possible. This can be solved using Dynamic Programming by constructing the matrices given below. We will create the matrix with dimension of number of cuts + 1 (i.e. number of substrings after cut). Ultimately, no matter in which order the string is cut, the final output substrings would always be the same. e.g. substring with indices 1-2, 3-10, 11-12, 13-20.

Consider, cut[ ] = {2, 8, 10}, substring_length[ ] = {2, 6, 2, 10} and length of string = 20 initially. Let L1, L2, L3, L4 be the length of those substrings where L1 = 2, L2 = 6, L3 = 2, L4 = 10.

L1	L2	L3	L4
0	8	18	38
	0	8	26
		0	12
			0

Cost Matrix

	0	1	2	3
0		0	1	2
1			1	2
2				2
3				

Helper Matrix

In general, the table can be filled using the following rule:

$$T[i, j] = \begin{cases} 0 & \text{if } i = j \\ \min_{i \le k < j} T[i, k] + T[k + 1, j] + \sum_{z=i}^{j} substring\_length[z] & \text{if } i < j \end{cases}$$

where, $\sum_{z=i}^{j} substring\_length[z]$ gives the length of the substring which is to be cut.

A particular cell (highlighted in green) represents the minimum cost required to break the string including substrings L1, L2, L3, L4. Solution path can be found out using the helper matrix which stores the index of optimum cut choice (e.g. 0 for cut at 2, 1 for cut at 8, 2 for cut 10 at etc.)

Space complexity: 2D matrix is needed for DP, which has dimensions of n + 1, where n is number of cuts. So, its $\boldsymbol{O(n^2)}$.
Time complexity: We have to fill $\ominus(n^2)$ cells, and each of the cell takes at most $O(n)$ time due to the min over all possible k values. So, its $\boldsymbol{O(n^3)}$.