Question 4

Solution

```
l = array of word lengths
n = number of words
M = length of array

lookupTable = {}

function solve(i) {
  if (i >= n) {
    return 0, []
  }

  // memoize function using a table
  cached = lookupTable[i]
  if (cached) {
    return cached
  }

  lineLength = l[i]
  bestScore = inf
  bestSplits = []
  j = i+1

  // For each of the possible line breaks, compute the score of the remaining
  // words.
  while (lineLength <= M && j < n) {
    // compute score of rest of the words
    s, splits = solve(j)
    // add the penalty from this break
    currentScore = s + (M-lineLength)^3;

    // pick the best score we've seen
    if (currentScore < bestScore) {
      bestScore = currentScore
      bestSplits = [j]+bestSplits
    }

    lineLength += l[j] + 1
    j++
  }

  // If we hit the end of the list, return zero since we aren't counting the
  // last line.
  if (j >= n) {
    bestScore = 0
    splits = []
  }

  // Save the result in a table.
  lookupTable[i] = (bestScore, splits)

  return bestScore, splits
}
```

Running time and space

There are n possible inputs to solve since the function is memoized and uses a table they won't be recomputed meaning the bulk of the solve function will only be run n times. Most of `solve` is a constant cost, except for the while loop, which may run $O(M)$ times since each word is at least 1 letter and there can be a maximum of M letters in a line. Thus the runtime is $O(Mn)$.

As for space complexity, the lookup table stores the best possible splits and scores. There are a maximum of n entries, and there could be n possible splits if every word is on a new line. Thus the space used is $O(n(n+1)/2) \equiv O(n^2)$.