Recursive Solution:

The main objective for this problem is to maximize the score obtained by matching the base pairs. Lets consider a string of base pair sequence, we need to find how many maximum legal base pairs are present in it.

This can be reduced to a optimal substructure by considering any two indices *i,j* from the given sequence. When there is a legal pair between any two bases at indices *i,j*, then we increment our function value by 1 and look for a pair inside the remaining sequence after pairing i with j.

Also, it may happen that there is some other base at index k ($i < k < j$) which is more favourable to be paired with the base at index i. Thus, our objective function should incorporate this case and compute the score for all possible values of k.

This can be solved using inside fashion by defining a function which tells us whether the given pair is Legal or not and considering the least sub-problem to be the internal one.

So we can form following recurrence relation:

$$M(i,j) = max \begin{cases} \mathrm{M}(i{+}1,\ j{-}1) + 1, & \text{if } x_i \text{ \& } x_j \text{ forms a legal base pair} \\ \mathrm{M}(i{+}1,\ j), & \\ \mathrm{M}(i,\ j{-}1), & \\ max_{i\,<\,k\,<\,j}[\mathrm{M}(i,\ k) + \mathrm{M}(k{+}1,\ j)] & \end{cases}$$

The 1$^{\text{st}}$ line computes the score recursively for remaining inner sequence if a legal pair is formed. The 2$^{\text{nd}}$ and 3$^{\text{rd}}$ lines gives the score if either base i or base j remains unpaired with any other base in the given sequence. While the last max function computes the parallel pairs. The recursion will continue until we get a pair of indices with length 0 or 1. Which will occur when $i \geq j$ or M(i,i) and M(i, i-1) and so on. Therefore, The Base case here will be:

$$\mathrm{S}(i,j) = 0 \text{ for } i \geq j$$

To implement this function in efficient way, instead of using recursive calls, we can use dynamic programming and fill the table diagonally.

Pseudo-code:

Here we assume that a sub routine **IS_PAIR_LEGAL(x_i,x_j)** to check whether a pair is legal or not is predefined. It checks x_i,x_j belongs to one of the pairs of **(A,U), (U,A), (G,C), (C,G)**. If it is legal, it returns **TRUE**.

Also, we store the index value of matched base in a pair in table S. If *(i,j)* forms a pair, we store j in S(i,j). If there is a bifurcation at index k we, store k at S(i,j). And finally, if there is no any pair between indices (i,j), we store -1 in S(i,j). Initially, we define "printStr" as an empty string array of length n. It is the string of matched pairs of parenthesis. Usage: First **call STRUCT_PRED** (*sequence*) with input sequence. Then **call PRINT_PARENTHESIS** $(1, n, S, printStr)$

Algorithm 1 STRUCT_PRED (*sequence*)

```
n = length[sequence] -1;
for i = 1 to n do
   M[i,i] = 0;
end for
for l = 2 to n do
   for i = 1 to (n - l + 1) do
      j = i + l - 1;
      M[i,j] = 0;
      if IS_PAIR_LEGAL(sequence[i],sequence[j]) then
         q = M[i+1, j-1] + 1;
         if q ≥ M[i,j] and q != 0 then
            M[i, j] = q;
            S[i, j] = j;
         end if
      end if
      for k = i to j - 1 do
         qk = M[i, k] + M[k + 1, j];
         if qk > M[i,j] then
            M[i, j] = qk;
            S[i, j] = k;
         end if
      end for
      if M[i,j] = 0 then
         S[i, j] = -1;
      end if
   end for
end for
return M , S
```

Algorithm 2 PRINT_PARENTHESIS $(i, j, S, printStr)$

---

if i > j **then**
  return
end if
if S[i,j] = -1 **or** S[i,j] = i **then**
  printStr[i] = ".";
  PRINT_PARENTHESIS $(i + 1, j, S, printStr)$
else if S[i,j] = j **then**
  printStr[i] = "{";
  printStr[j] = "}";
  PRINT_PARENTHESIS $(i + 1, j - 1, S, printStr)$
else
  k = S[i,j] ;
  printStr[i] = "{";
  printStr[k] = "}";
  PRINT_PARENTHESIS $(i + 1, k - 1, S, printStr)$
  PRINT_PARENTHESIS $(k + 1, j, S, printStr)$
end if
return
