Dheeraj Satya Sushant Viswanadham
(z5204820)

COMP3121 Assignment 4 – Question 1

1) The string of Boolean expression will be in the order of: "symbol OPERATOR symbol" where we can have multiple such expressions. To count up the number of ways that we can put parentheses in the expression such that it will evaluate to '*true*' then we need to come up with a method to find all the possible variations (or combinations) that the subproblems can take and then order them in order of precedence. Once we have this, we can then evaluate them and place our parentheses in such an order that the overall expression evaluates to be *true*.

See given diagram to see how the NAND and NOR operators work (acquired from the homework questions booklet):

NAND	*true*	*false*
true	*false*	*true*
false	*true*	*true*

NOR	*true*	*false*
true	*false*	*false*
false	*false*	*true*

Put simply, we need to consider the contiguous substrings of the Boolean expression between every i^{th} and j^{th} true/false symbol. Once we have these substrings, we then must address the subproblems of counting the number of placements of brackets that will:

1) Evaluate the subexpression to TRUE; and
2) Evaluate the subexpression to FALSE (as it depends on the operator used in question such as XOR).

Once we have this, we can then simply calculate the number of placements of the brackets that will make the overall expression evaluate to *true*.

Note that we will not be able to 'sort' the expression as such by the order of OPERATORS as this will ultimately change the given Boolean expression making our solution invalid. Hence, what we can do is to simply start from the left of the given Boolean expression and go across. If we only have one operand and no operators given, and if that operand is the symbol TRUE then the base case will be 1 as we can simply place our brackets around that symbol and 0 otherwise (as the symbol is FALSE (F) otherwise). This will however be a special case as (T) and ((T)) will be 1 pair of brackets as they are equivalent, same for (T and T). Once we have 2 or more symbols in the expression, we can then look at the specific operator in between and place a pair of parentheses there and evaluate that expression. If true, then we increase our "trueCount" counter variable by 1 otherwise if false, then we increase our "falseCount" counter variable by 1. Then go across to the next adjacent operand (if available) and place a parenthesis there and see if that evaluates to true or false. If we have 3 operands, then we can also see all possible combinations that we can place our parentheses using combinatorics, particularly by using Catalan numbers (denoted by C_n) which will be done recursively as we check for these possible combinations and evaluate the expressions accordingly. In this way, we will firstly recursively solve the subproblem for the subexpression TRUE and after we have gone through the entire expression, go through again for evaluating subexpressions when they are FALSE. Simply put:

We will have *m* number of symbols (and *m – 1* operators as a result). We have already covered the base case where T(i, i) is "1" if our symbol itself is *true* and 0 if symbol is *false*. We will do the reverse for F(i, i). Then our general case will be to split our expression into 2 around our pivot operator so that

everything to the left will have its own bracket and everything to the right will have its own bracket. We then evaluate the two subproblems and combine the results and we must make sure it is in-line with the pivot operator in question and what we want the result to evaluate to (either T or F). We would be solving both problems in parallel time where the overall complexity would be $O(n^3)$ as there are n^2 different ranges that our i and j could cover where we are evaluating our two splitting functions (one for True and the other for False) at each stage.

As a note, the formula for obtaining the Catalan number is as follows:

$$C_n = \frac{1}{n+1} * \binom{2n}{n}.$$

Where *n* is the number of OPERATORS and $n \geq 0$.

We can see that it works by using a simple example:

$$true\ OR\ false\ AND\ false\ OR\ true$$

Which can be split into the following bracket placements:

1) $((true\ OR\ false)\ AND\ false)\ OR\ true$
2) $(true\ OR\ false)\ AND\ (false\ OR\ true)$
3) $true\ OR\ ((false\ AND\ false) OR\ true)$
4) $(true\ OR\ (false\ AND\ false))\ OR\ true$
5) $true\ OR\ (false\ AND\ (false\ OR\ true))$

If we use the formula, we can see that it works as we have three operators so our n = 3:

$$C_3 = \frac{1}{3+1} * \binom{2(3)}{3} = \frac{1}{4} * \binom{6}{3} = 5\ bracket\ placements.$$

Additionally, we can find our Catalan number by using dynamic programming where we can store each value at each stage within an array and refer to this if we have a higher n. We can find the required number of different combinations that the brackets can be placed in and evaluate them by following the steps prescribed above at each stage to solve within each subproblem. We are traversing through the expression and counting the number of operators which will be our n. At each stage of this traversal, we are filling in our Catalan number table using a recursive formula for each n^{th} operator to be used above it and evaluating that expression to fill the slot in the table. Hence, the time complexity for obtaining the Catalan number will be $O(n^2)$ as we have 2 for loops overall. Hence, by using our Catalan numbers to optimally derive all possible combinations of bracket placements, and evaluating the expression for each combination, we will arrive at our answer in $O(n^3)$ time.

End of Solution