CS 4820, Spring 2019 **Homework 4, Problem 1**
Name: Kevin Klaben NetID: kek228 Collaborators: Quinn Lui

(1) Parentheses *(10 points)*
You are helping out a sloppy friend with a math problem. He has written an expression of the form $a_1O_1a_2O_2 \dots O_{n-1}a_n$, where each a_i is either FALSE denoted by 0 or TRUE denoted by 1, and each O_i is the logical OR denoted by $\vee$ or the logical XOR denoted by $\oplus$. Unfortunately, he has forgotten to insert in parentheses in the expression which leads you to think of the following problem: how many ways of parenthesizing the expression leads the evaluation to be 0? Develop an efficient algorithm that takes as input an expression E of the form $a_1O_1a_2O_2 \dots O_{n-1}a_n$ and outputs the number of ways of parenthesizing E such that it evaluates to 0 (i.e, FALSE).

Example: Let the input expression E be: $0 \vee 1 \oplus 1 \vee 1$. Below are all the ways of parenthesizing E:

- $(((0 \vee 1) \oplus 1) \vee 1)$ which evaluates to 1.
- $((0 \vee (1 \oplus 1)) \vee 1)$ which evaluates to 1.
- $((0 \vee 1) \oplus (1 \vee 1))$which evaluates to 0.
- $(0 \vee ((1 \oplus 1) \vee 1))$ which evaluates to 1.
- $(0 \vee (1 \oplus (1 \vee 1)))$ which evaluates to 0.

Thus the output of your algorithm on this instance should be 2.

```
RECPAR(i,j,bool)
if (T[i][j]!=NULL and bool) or (F[i][j]!=NULL and not bool) then
  if (bool) then
    return T[i][j]
  else
    return F[i][j]
  end if
else if i==j then
  if (bool) then
    T[i][j]=$a_i$
    F[i][j]=not $a_i$ (1 if $a_i$=0, 0 if $a_i$=1)
    return T[i][j]
  else
    T[i][j]=$a_i$
    F[i][j]=not $a_i$ (1 if $a_i$=0, 0 if $a_i$=1)
    return F[i][j]
  end if
```

```
else
    falsetot=0
    for k in range(i, j-1) do
    if $O_k$==V (or) then
      ff=RECPAR(i,k,false)*RECPAR(k+1,j,false)
      falsetot=falsetot+ff
    else
      ff=RECPAR(i,k,false)*RECPAR(k+1,j,false)
      tt=RECPAR(i,k,true)*RECPAR(k+1,j,true)
      falsetot=falsetot+ff+tt
    end if
  end for
  F[i][j]=falsetot
  T[i][j]=((2*(j-i))!/((j-i)!*(j-i+1)!))-falsetot
  if (bool) then
    return T[i][j]
  else
    return F[i][j]
  end if
end if
```

Calling RECPAR(1, n, false) will yield the desired output.

Runtime Analysis:
This algorithm memoizes the running time by the 2 dimensional arrays T and F which are used to represent the number of ways to parenthesize the elements i to j that return False for F and True for T. As T and F are both n by n it takes O(n^2) time for memoizing. Overall, the first call to RECPAR will do the majority of the work as the position in the table [1][1] will be computed as well as all positions [2..n][2..n] as a result, thus all other recursive calls on this level will make at a maximum four additional recursive calls each of which will have already had the result memoized thus taking constant time. As this is the case at each level of the table there are n levels each having a first call doing most of the work with a maximum of n calls on each level each doing 4 O(n) work. Thus, for the algorithm the overall runtime will be O(n^2+4n^2)=O(n^2)

Proof of Correctness. Claim: This algorithm correctly computes the number of arrangements of parenthesis to produce false as the output.
Proof: By induction on the size of the list n.
Base Case: n=2
proof: when n is equal to 2 the call will result in identifying the one operation in the list and then will output whether of not that operation will output false or true. Thus, there is only one way of parenthesizing the list and the output of the operation is simply the only output and is the only way for false to be achieved or not and thus the algorithm will simply output whether this was false or whether it was true making the correct output for the list.

Inductive case: Given that the algorithm outputs the correct number of false orientations for each subset of the list of length n other than the actual list we want to show that the overall list is also correctly computed.
proof: The algorithm will iterate over the list of length n and split it into two sublists of at

least length 1 for all possible ways. Thus, at a maximum each sublist could be possibly n-1 in length as the other sublist could be at a minimum length 1. Thus, as we were given that the algorithm correctly computes for lists of length 1..n-1 we can be sure that the number of ways of parenthesizing to get true or to get false will be correct. The algorithm finishes by summing each of these valid sublists producing false and thus gets total number for ways of parenthesizing to receive false. Thus the algorithm will correctly compute the number of ways of parenthesizing lists of length n to achieve false.