//Recursive algorithm for regex union that returns an equivalent disjoint-union of regexs.

// As a result, algorithm returns a set with NO repeat regexs and NO redundant regexs.

def right\_or(w1, .., wn):

// Base case:

if (len(w1) == … == len(wn) == 1):

return or(w1, w2, .. wn)

// Inductive case:

w1 = s1 c1, … wn = sn cn

// Empty initialzed vectors

end\_zero, end\_one

// Sort s1, .., sn in vectors based on ending character

for i in range(0, n-1, 1):

if (ci == 0):

end\_zero.append(si)

if (ci == 1):

end\_one.append(si)

if (ci == s):

end\_zero.append(si)

end\_one.append(si)

// Recursively call right\_or on non-empty end\_zero and end\_one vectors.

// If both vectors are empty, return empty string.

if end\_zero != empty and end\_one != empty:

return str\_concat( right\_or(end\_zero), “0+”, right\_or(end\_one), “1”)

else if end\_zero == empty and end\_one != empty:

return str\_concat(right\_or(end\_one), “1”)

else if end\_zero != empty and end\_one == empty:

return str\_concat(right\_or(end\_zero), “0”)

else:

return empty

Good example input to try code on:

[“0s0s0”, “s0s0s”]