Reducing Dishes

Task: A Chef collected data on the satisfaction level of n dishes. Chef can cook any dish in 1 unit of time.

Like-time coefficient of a dish is defined as the time taken to cook that dish including previous dishes multiplied by its satisfaction level i.e. time[i] * satisfaction[i].

Return the maximum sum of like-time coefficient that the Chef can obtain after dishes preparation.

Dishes can be prepared in any order and the chef can discard some dishes to get this maximum value.

How?

1. Sort the array from the smallest to the biggest number.
2. If all numbers are negative, return 0 (worst case)
3. If all numbers >= 0, calculate the time * All values (Best case)
4. If there are both negative and positive numbers: calculate the time[i] * satisfaction[i] from the last item in the sorted array (A[len(A)-1] = index 0, and so on).

   If the total sum goes greater when we add items, keep scan the array.

   If the total sum lower than the last sum, break and return the last sum (maximum).

Example: input: [-1, -8, 0, 5, -9]

Sort: [-9, -8, -1, 0, 5]
Both negative & positive numbers

$5 \cdot 1 = 5$ ✓
$0 \cdot 1 + 5 \cdot 2 = 10$ ✓
$-1 \cdot 1 + 0 \cdot 2 + 5 \cdot 3 = 14$ ✓
$-8 \cdot 1 + (-1) \cdot 2 + 0 \cdot 3 + 5 \cdot 4 = 10$ ✗ → return 14 //

The Algorithm:

```
def multSum(A):  // It takes an array of integers, multiplies each integer by its index+1 (unit of time), and return the sum of all products
    sum = 0
    length = len(A)

    for time in range(length):
        sum += ((time+1) * A[time])

    return sum
```

```
def maxSatisfaction(satisfaction):

    length = len(satisfaction)
    index = length-1
    sum = 0
    max = 0

    satisfaction.sort()

    if satisfaction[-1] <= 0  // Worst case, All numbers are 0/negative
        return 0

    elif satisfaction[0] >= 0:  // Best case, All numbers >= 0
        for i in range(length):
            sum += (i+1) * satisfaction[i]
        return sum

    else:  // Both negative & positive numbers
        while index >= 0:
            current_sum = multSum(satisfaction[index:length])

            if current_sum < max:
                break

            else:
                max = current_sum

            index -= 1

        return max
```

Check If It Is a Good Array

Definitions:

- **Divisor:** If $a, b \in \mathbb{Z} \wedge b \mid a \wedge b > 0$ then b called divisor of a.
- **Common Divisor:** If $a, b \in \mathbb{Z} \wedge q \mid a \wedge q \mid b \wedge q > 0$ then q is a common divisor of a and b.
- **gcd(a, b):** The greates common divisor of a and b.
- **Extended Euclidean Algorithm:** finds integer coefficients x and y such that:

$$d = gcd(a, b) = ax + by$$

Task:

Given an array `nums` of positive integers. Your task is to select some subset of `nums`, multiply each element by an integer and add all these numbers. The array is said to be good if you can obtain a sum of 1 from the array by any possible subset and multiplicand.

Return `True` if the array is good otherwise return `false`.

How?

If we found two integers a, b, which have $gcd(a, b) = 1$, then we can sy that there is a Linear combination: $1 = ax + by$ //

we have array of numbers ($1 <= nums.length <= 10^5$).
So we can say that if $gcd(nums) = 1$, then a subset of nums exist and make the requirment to be True. Otherwise, there is no subset like this and the answer is false.

Algorithm:

```
import numpy as np

def isGoodArray(nums):
    '''
    :type nums: List[int]
    :rtype: bool
    '''
    result = np.gcd.reduce(nums)

    if result == 1:
        return True

    return false
```

Examples:

calculate gcd of all nums items by reduce function.

(1) nums = [12, 5, 7, 23]

gcd(12, 5) = 1

the first two items

gcd(1, 7) = 1

gcd(1, 23) = 1 True

(2) nums = [3, 6, 12, 9]

gcd(3, 6) = 3

gcd(3, 12) = 3

gcd(3, 9) = 3 ≠ 1 False

Split Array Largest Sum

Task: Given an array `nums` which consist of non-negative integers and an integer `m`, you can split the array into `m` non-empty continuous subarrays.

Write an algorithm to minimize the largest sum among these `m` subarrays.

Example:

- Input: nums = [7, 2, 5, 10, 8], m = 2

  possible combinations:

  7 ; 2, 5, 10, 8 → sums: 7 ; 25

  7, 2 ; 5, 10, 8 → sums: 9 ; 23

  7, 2, 5 ; 10, 8 → sums: 14 ; 18 min

  7, 2, 5, 10 ; 8 → sums: 24 ; 8

  → The largest sums: 25, 23, 18, 24

  → The minimum among largest sum = 18 //

- Input: nums = [1, 4, 4], m = 3

  possible combinations: 1 ; 4 ; 4

  The largest sum 4. One result -> It is the min = 4

How?

At Bruth-force method we can scan all options, but running time will be large.

Instead, let's apply greedy algorithm which try to find the best solution.

First step: If m = 1, we know there is only answer and it the maximum one.
The result will be the sum of all numbers in the array.

If m = len(nums), the result will be the maximum number in nums.

second step: After define right & left limits, we can assume that result is between those boundaries. If so, we can apply binary search to find the minimum among largest results.

right (high)
left (low)

Third step: Define middle : ⌊(low + high) / 2⌋ (low boundary)

fourth step: Check if it possible to divide the array into m subarrays while the max sum of any sub array doesn't exceed the middle.

If True → the answer lay between low and middle.

If false → the answer lay between middle and high

Example Tracing:

Input: nums = [7, 2, 5, 10, 8], m = 2

low = 10, high = 32 → middle = (32 + 10) // 2 = 21

7 + 2 + 5 <= 21 ; 10 + 8 <= 21 → m = 2 ✓ result = 21

low = 10, high = 20 → middle = (20 + 10) // 2 = 15

7 + 2 + 5 <= 15 ; 10 <= 15 ; 8 <= 15 → m = 3 ✗

low = 16, high = 20 → middle = 18

7 + 2 + 5 <= 18 ; 10 + 8 <= 18 → m = 2 ✓ result = 18

low = 16, high = 17 → middle = 16

7 + 2 + 5 <= 16 ; 10 <= 16 ; 8 <= 16 → m = 3 ✗

low = 17, high = 17 → middle = 17

7 + 2 + 5 <= 17 ; 10 <= 17 ; 8 <= 17 → m = 3 ✗

return result (18) //

Algorithm:

```
def possibleSplitIntoMPieces(nums, limit, m):

    subArrays_counter = 0
    sum = 0

    for num in nums:

        if (sum + num) > limit:
            subArrays_counter += 1
            sum = num

        else:
            sum += num

    subArrays_counter += 1
    if m >= subArrays_conter:
        return True
    else
        return false


def splitArrays(nums, m):

    high = sum(nums)
    low = max(nums)
    nums_length = len(nums)


    if m == nums_length:
        return low
    if m == 1:
        return high

    while low <= high:
        middle = (low + high) //2

        if possibleSplitIntoMPieces(nums, middle, m):
            high = middle -1
            result = middle

        else
            low = middle +1

    return result
```

Find XOR Sum of All Pairs Bitwise AND

Definitions:

OR	AND	XOR
$x+y$	$x \cdot y$	$x \oplus y$ $\bar{x} \cdot y + x \cdot \bar{y}$ $(x+y)(\bar{x}+\bar{y})$

$(\oplus \equiv 1)$

6 AND 1

↓

1 1 0

&

0 0 1

0 0 0 = 0

6 AND 3

↓

1 1 0

&

0 1 1

0 1 0 = 2

Distributive formula: $(a \cdot b) \oplus (a \cdot c) = a \cdot (b \oplus c)$

$(a \cdot b) \oplus (a \cdot c) \oplus (a \cdot d) = a \cdot (b \oplus c \oplus d)$

Task: The XOR sum of a list is the bitwise XOR of all its elements. If the list only contains one element, then its XOR sum will be equal to this element.

- for example, the XOR sum of [1,2,3,4] is equal to 1 XOR 2 XOR 3 XOR 4 = 4, and the XOR sum of [3] is equal to 3.

You are given two 0-indexed arrays arr1 and arr2 that consist only of non-negative integers.

Consider the list containing the result of arr[i] AND arr[j] (bitwise AND) for every (i, j) pair where $0 <= i < arr1.length$ and $0 <= j < arr2.length$.

Return the XOR sum of the aforementioned list.

How? Given two list, for example: [a, b, c]; [d, e].

The instructions: $a \cdot b \oplus a \cdot d \oplus b \cdot d \oplus b \cdot e \oplus c \cdot d \oplus c \cdot e =$

$\rightarrow (a \cdot d \oplus b \cdot d \oplus c \cdot d) \oplus (a \cdot e \oplus b \cdot e \oplus c \cdot e)$

use the distributive formula: $d \cdot (a \oplus b \oplus c) \oplus e \cdot (a \oplus b \oplus c) =$

$\rightarrow (a \oplus b \oplus c) \cdot (d \oplus e)$ Mean XOR of each array and then AND between results

Algorithm:

```
import functools

def xor(num1, num2):

    result = num1 ^ num2
    return result


def get XORSum(arr1, arr2):

    xor_arr1 = functools.reduce(xor, arr1)
    xor_arr2 = functools.reduce(xor, arr2)

    and_arr1_arr2 = xor_arr1 & xor_arr2

    return and_arr1_arr2
```

Example:

Input: arr1 = [1, 2, 3] , arr2 = [5, 6]

$1 \cdot 5 \wedge 1 \cdot 6 \wedge 2 \cdot 5 \wedge 2 \cdot 6 \wedge 3 \cdot 5 \wedge 3 \cdot 6 =$

$= (1 \cdot 5 \wedge 2 \cdot 5 \wedge 3 \cdot 5) \wedge (1 \cdot 6 \wedge 2 \cdot 6 \wedge 3 \cdot 6) =$

$= 5 \cdot (1 \wedge 2 \wedge 3) \wedge 6 \cdot (1 \wedge 2 \wedge 3) =$

$= (1 \wedge 2 \wedge 3) \cdot (5 \wedge 6) = 0$

Chalkboard XOR Game

Task: You are given an array of integers `nums` represents the numbers written on a chalkboard.
Alice and Bob take turns erasing exactly one number from the chalkboard with Alice starting first. If erasing a number causes the bitwise XOR of all elements of the chalkboard to become 0, then that player loses.
The bitwise XOR of one element is that element itself, and the bitwise XOR of no elements is 0.
Also, if any player starts their turn with the bitwise XOR of all the elements of the chalkboard equal to 0, then that player wins.
Return true if and only if Alice wins the game, assuming both players play optimally.

How? Identities $A \oplus A = 0$; $A \oplus B \neq 0$; $A \oplus B = B \oplus A$

1. If After erase a number, XOR of all elements is equal to zero, the player will loss.
2. If the player erase the last number — he will lose.

case 1:

- If any player starts their turn with the bitwise XOR of all the elements of the chalkboard equal to 0, then that player wins.
- The array: $a_1, a_2, a_3, \ldots, a_n$

  If Alice pick a_k, the XOR will be $a_1 \oplus \cdots a_{k-1} \oplus a_{k+1} \oplus a_n$.

  $(a_1 \oplus \cdots a_{k-1} \oplus a_{k+1} \oplus a_n) \oplus a_k = 0$ only if
  $a_1 \oplus \cdots a_{k-1} \oplus a_{k+1} \oplus a_n = a_k$
  And if apply on each next i pick.

→ The important part is that if the XOR of elements is 0, then Alice can control the game, she just need to avoid to pick a_k such as:

$$a_1 \oplus \cdots a_{k-1} \oplus a_{k+1} \oplus a_n = a_k$$

case 2: $a_1 \oplus a_2 \oplus \cdots \oplus a_n \neq 0$

→ mean that Alice will lose only if she got odd turns to play.

So for wining, the require should be that number of turns should be ever of Alice's win.

Algorithm:

```
import functools

def xor(num1, num2):

    result = num1 ^ num2
    return result

def xorGame(nums)

    xor_result = functools.reduce(xor, nums) == 0
    queue_result = (len(nums) % 2) == 0

    if xor_result:
        return True

    return queue_result
```

Minimum Insertion steps to make a string Palindrome

Definition: LCS - Longest Common string!

Given two sequences, find the length of longest subsequence present in both of them. A subsequence that appear in the same relative order, but not necessarily contiguous.

IDEA:

Compute the table bottom-up.

Time = $O(mn)$.

		A	B	C	B	D	A	B
	0	0	0	0	0	0	0	0
B	0	0	1	1	1	1	1	1
D	0	0	1	1	1	2	2	2
C	0	0	1	2	2	2	2	2
A	0	1	1	2	2	2	3	3
B	0	1	2	2	3	3	3	4
A	0	1	2	2	3	3	4	4

$$c[i,j] = \begin{cases} 0 & \text{if } i=0 \text{ or } j=0 \\ c[i-1,j-1]+1 & \text{if } x[i]=y[j], \\ \max\{c[i-1,j], c[i,j-1]\} & \text{otherwise.} \end{cases}$$

c[i–1, j–1]	c[i–1, j]
c[i, j–1]	c[i, j]

Computing the length of an LCS

```
LCS-LENGTH (X, Y)
1.  m ← length[X]
2.  n ← length[Y]
3.  for i ← 1 to m
4.      do c[i, 0] ← 0
5.  for j ← 0 to n
6.      do c[0, j ] ← 0
7.  for i ← 1 to m
8.      do for j ← 1 to n
9.          do if x_i = y_j
10.                 then c[i, j ] ← c[i–1, j–1] + 1
11.             else if c[i–1, j ] ≥ c[i, j–1]
12.                     then c[i, j ] ← c[i– 1, j ]
13.                 else c[i, j ] ← c[i, j–1]
14.  return c
```

c[m,n] contains the length of an LCS of X and Y.

Time: $O(mn)$

Task: Given a string s. In one step you can insert any character at any index of the string.

Return the minimum number of steps to make s palindrome.

A palindrome string is one that reads the same backward as well as forward.

How? The idea is to use the opposit target of the LCS algorithm.

→ Given S. create reverse S.
The optimal state is where result of LCS is len(S), means we have a palindrome.

If not, it means we have to insert letters.

→ Now, the return value will be the longest common subsequence.

we have to find the uncommon letters.

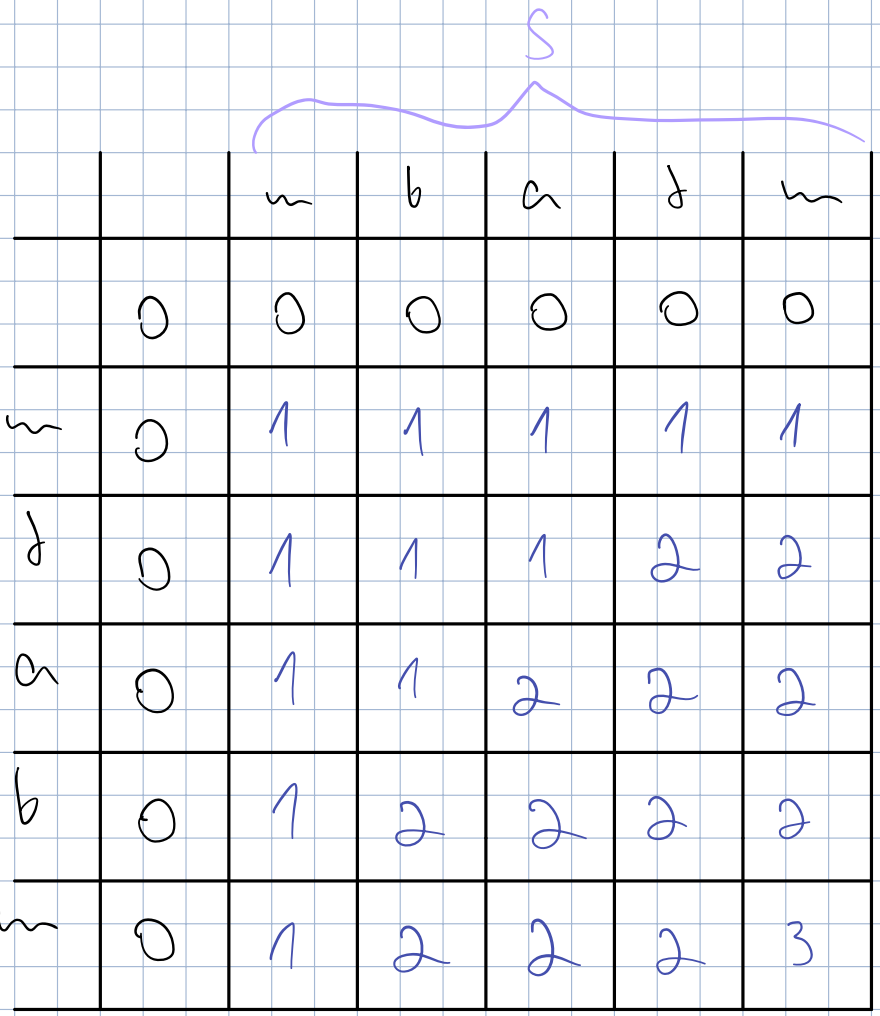
⟶ len(S) - (return value of LCS) = Number of insertions

Algorithm:

```
def minInsertion(s):

    reverse_s = s[::-1]

    str_length = len(s)
    matrix_dimension = len(s)+1

matrix = [[0 for col in range(matrix_dimension)] for row in range(matrix_dimension)]

    for i in range(1, matrix_dimension)
        for j in range(1, matrix_dimension)

            if s[i-1] == reverse_s[j-1]:
                matrix[i][j] = matrix[i-1][j-1] + 1
            elif matrix[i-1][j] >= matrix[i][j-1]:
                matrix[i][j] = matrix[i-1][j]
            else:
                matrix[i][j] = matrix[i][j-1]

    return str_length - matrix[-1][-1]
```

Example:

Input: s = "mbadm"



		m	b	a	d	m
	0	0	0	0	0	0
m	0	1	1	1	1	1
d	0	1	1	1	2	2
a	0	1	1	2	2	2
b	0	1	2	2	2	2
m	0	1	2	2	2	3

Output: len(s) - matrix[-1][-1] = 5 - 3 = 2 //

Shortest Common Subsequence

Task: Given two strings `str1` and `str2`, return the shortest string that has both `str1` and `str2` as subsequences. If there are multiple valid strings, return any of them.

A string `s` is a subsequence of string `t` if deleting some number of characters from `t` (possibly 0) results in the string `s`.

How?

1. Find LCS (Longest Common Subsequence).
2. Run on each string (str1, str2) seperatly one after one, until getting first letter from step 1. concat them to the return result.
3. Concat to the return result the ith common character.
4. repeat on steps 2, 3 until finish scanning str1, str2.
5. Concat the rest of str1, str2 to result.

Algorithm:

```
def LCS(S1, S2) // return the Longest Common string
                // return type: str

def shortestCommonSupersequence(str1, str2)

    lcs_string = LCS(str1, str2)

    i=0
    j=0
    result=""

    for letter in lcs_string:
        while(str1[i] != letter):
            result += str1[i]
            i+=1
        while(str2[j] != letter):
            result += str2[j]
            j+=1

        result+=letter
        i+=1
        j+=1

    result += str1[i:]
    result+=str2[j:]
    return result
```

Example:

Input: str1 = "abcac"
str2 = "cab"

—> lcs = ab

for loop: "a" —> str1 concat to result: ""
str2 concat to result: "c"

result = "c" —> result = "ca"

for loop: "b" —> str1 concat to result: ""
str2 concat to result: ""

result = "ca" —> result = "cab"

end for

str1 concat to result: "ac" —> result = "cabac"
str2 concat to result: "" —> result = "cabac"

return "cabac"

Edit Distance

Task: Given two strings `word1` and `word2`, return the minimum number of operation required to convert `word1` to `word2`.

You have the following three operations premitted on a word:

- Insert a character.
- Delete a character.
- Replace a character.

How? (*similar to lcs)

1. Initialize a matrix with dimensions `[len(word2)][len(word1)]`
2. Initialize all matrix values to 0.
3. Initialize first row and column values to their index.
4. If `word1[i] == word2[j]`: don't change anything.
5. else! check which operation will be with minimum cost:
   - Insertion (take left value +1)
   - Remove (take upper value +1)
   - Replace (take slant value +1)
6. The value at `matrix[-1][-1]` is the minimum operations.

Example!

לדוגמא, ניתן להפוך את המחרוזת **Saturday** למחרוזת **Sunday** על-ידי 3 פעולות עריכה:

- מחיקה של האות a מהמקום השני.
- מחיקה של האות t מהמקום השני.
- החלפת האות r באות n במקום השלישי.
- על שאר התווים לא נבצע אף פעולה.

Saturday → Sturday→ Surday→ Sunday

		S	a	t	u	r	d	a	y
	0	1	2	3	4	5	6	7	8
S	1	0	1	2	3	4	5	6	7
U	2	1	1	2	2	3	4	5	6
N	3	2	2	2	3	3	4	5	6
D	4	3	3	3	3	4	3	4	5
A	5	4	3	4	4	4	4	3	4
Y	6	5	4	4	5	5	5	4	3

כש"ש שווין בין שתי אותיות שמתכלים על תתי המחרוזות שבאלכסון לו.

במידה ולא נמתכל על האינדקס שמעלה, השמאלי ובאלכסון ניקח האינדקס שהכי מינימלי מביניהם.

edit distance = 3

Delete - למעלה
replace - באלכסון
Insert - משמאל

בכל פעם משווים בין תתי המחרוזות ובודקים איזו פעולה (מחיקה / הוספה / החלפה)

Algorithm:

```
def initMatrix(len1, len2):

    mat = [[0] * len2 for _ in range(len1)]

    for i in range(len1):
        mat[i][0] = i

    for i in range(len2):
        mat[0][i] = i

    return mat
```

```
def minDistance(word1, word2):

    len1 = len(word1)
    len2 = len(word2)

    matrix = initMatrix(len1+1, len2+1)

    for i in range(1, len1+1):
        for j in range(1, len2+1):
            if word1[i-1] == word2[j-1]:
                matrix[i][j] = matrix[i-1][j-1]
            else:
                upper_element = matrix[i-1][j]
                left_element = matrix[i][j-1]
                upper_left_element = matrix[i-1][j-1]

                matrix[i][j] = 1 + min(upper_element, left_element,
                                       upper_left_element)

    return matrix[-1][-1]
```

Find Minimum in Rotated Sorted Array II

Task: Suppose an array of length n sorted in ascending order is rotated between 1 and n times. for example, the array nums = [0,1,4,4,5,6,7] might become:

- [4,5,6,7,0,1,4] if it was rotated 4 times.
- [0,1,4,4,5,6,7] if it was rotated 7 times.

Notice that rotating an array [a[0], a[1], ..., a[n-1]] 1 time results in the array [a[n-1], a[0], a[1], ..., a[n-2]].

Given the sorted rotated array nums that may contain duplicates, return the minimum element of this array.

You must decrease the overall operation steps as much as possible.

How? It is a classic Binary search with a twist.

After rotate the array, we have 3 possibilities:

1. The array is sorted (option to duplicated values).
2. The array isn't sorted, and min located at the right to the middle.
3. The array isn't sorted, and min located at the left to the middle.

At each iteration check in which part the min located, and change high/low respectivly.

After changing high/low value we still leave with the same 3 options but with smaller amount of numbers.

Keep iterate until coverage (while low < high).

Algorithm:

```
def findMin(nums):

    low=0
    high=len(nums)-1

    while low<high:

        mid=(low+high)//2

        if nums[mid]>nums[high]:
            low=mid+1

        elif nums[low]>nums[mid]
            high=mid
            low+=1

        else: // The array is sorted
            high-=1 // maybe there are duplicated values.

    return nums[low]
```

Examples:

- 0 1 2 3 4 5 6
  [4,5,6,7,0,1,4]

  low=0, high=6 -> mid=3

  nums[mid] > nums[high] (7>4)

  -> low=4, high=6, mid=5

  Sorted Array else: high=5

  low=4, high=5, mid=4

  else: high=4

return nums[low] - nums[4]=0

candy

Task: There are n children standing in a line. Each child is assigned a rating value given in the integer array ratings.

You are giving candies to these children subjected to the following requirements:

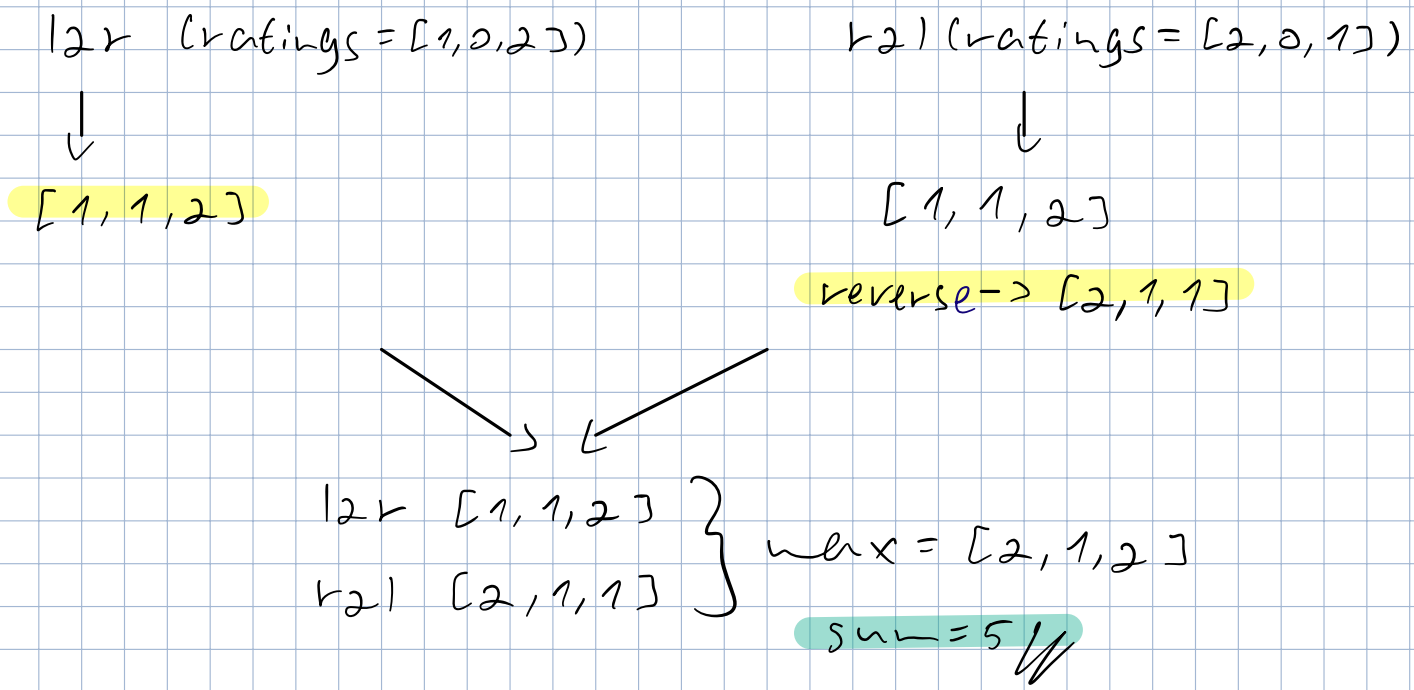
- Each child must have at least one candy.
- Children with a higher rating get more candies than their neighbors.

Return the minimum number of candies you need to have to distribute the candies to the children.

How?

1. Because we have to consider in both of neighbores (if exist) of each child, we have to calculate how much candy we give to each child:
   - one time calculate from right to left.
   - second time calculte from left to right (reverse)
2. After we calculat two possibilities, we have to consider in both of them. It means, just take the maximum value (number of candies) to each child.

Example: Input: ratings = [1, 0, 2]



Algorithm:

```
def candies_array(ratings):

    candies = [0] * len(ratings)
    candies[0] = 1

    for i in range(1, len(ratings)):

        if ratings[i] <= ratings[i-1]:
            candies[i] = 1
        elif ratings[i] > ratings[i-1]:
            candies[i] = candies[i-1] + 1

    return candies
```

```
def candy(ratings):

    if len(rating) == 1:
        return 1

    lr = candies_array(ratings)

    ratings.reverse()
    rl = candies_array(ratings)

    rl.reverse()

    sum_of_candies = 0

    for i in range(0, len(ratings)):
        sum_of_candies += max(lr[i], rl[i])

    return sum_of_candies
```