





```
[39]:
'''
Better solution

Note:
    If arr has zero & rev numbers, this is the better solution.
    If arr has zero & rev & -rev numbers, this is the optimal solution.

Time Complexity : O( n )
Space Complexity : O( 1 )
'''
def find_longest_subarr_len_BS(arr, k):
    n = len(arr)
    longest_subarr_len = float('-inf')
    running_sum = 0
    running_sum_map = dict()

    for i in range(0, n):
        running_sum += arr[i]

        if running_sum == k:
            length = i + 1
            longest_subarr_len = max(length, longest_subarr_len)

        remainder = running_sum - k
        length = 1
        running_sum_map[remainder] = 1
        longest_subarr_len = max(length, longest_subarr_len)

        if running_sum not in running_sum_map:
            running_sum_map[running_sum] = 1

    return longest_subarr_len

In [40]:
test_cases = [{'k': 3, 'arr': [1,2,-3]},
              {'k': 5, 'arr': [2,3,5]},
              {'k': 10, 'arr': [2,3,5,1,9]},
              {'k': 0, 'arr': [-1,-1,1]},
              {'k': 3, 'arr': [1,2,3,1,1,1,4,2,3]},
              {'k': 7, 'arr': [1,2,-3,1,0,-4,1,4,0,1,0,4,3]},
              {'k': 0, 'arr': [-1,0,1,1,-1,0]}]

for to in test_cases:
    arr = to['arr']
    k = to['k']
    print(find_longest_subarr_len_BS(arr, k))

2
2
3
3
12
12
6
```

```
In [41]:
'''
Optimal solution

Note:
    If arr has zero & rev numbers, this is the optimal solution.

Time Complexity : O( 2n )
Space Complexity : O( 1 )
'''
def find_longest_subarr_len_OS(arr, k):
    n = len(arr)
    longest_subarr_len = float('-inf')
    flexi_sum = arr[0]
    i = j = 0

    while i < n:
        i += 1
        while i <= n and flexi_sum > k:
            flexi_sum -= arr[i]
            i += 1

        if flexi_sum == k:
            length = j + 1
            longest_subarr_len = max(length, longest_subarr_len)
            j += 1

        if j < n:
            flexi_sum += arr[j]

    return longest_subarr_len

In [42]:
test_cases = [{'k': 0, 'arr': [-1,0,1,1,-1,-1,0]},
              {'k': 3, 'arr': [1,2,3,1,1,1,4,2,3]}]

for to in test_cases:
    arr = to['arr']
    k = to['k']
    print(find_longest_subarr_len_OS(arr, k))

3
```

### Problem Statement 14: Count Subarray sum Equals K

Given an array of integers and an integer k, return the total number of subarrays whose sum equals k.

**Note:** A subarray is a contiguous non-empty sequence of elements within an array.

**Example 1:**  
Input: arr = [3, 1, 2, 4], k = 6  
Output: 2  
Explanation: The subarrays that sum up to 6 are [3, 1, 2] and [2, 4].

**Example 2:**  
Input: arr = [1,2,3], k = 3  
Output: 2  
Explanation: The subarrays that sum up to 3 are [1, 2], and [3].

```
In [43]:
'''
Brute Force solution

Time Complexity : O( n^2 )
Space Complexity : O( 1 )
'''
def find_all_subarr_with_given_sum_BFS(arr, k):
    n = len(arr)
    subarr_counter = 0

    for i in range(0, n):
        rsum = 0

        for j in range(i, n):
            rsum += arr[j]
            if rsum == k:
                subarr_counter += 1

    return subarr_counter

In [44]:
test_cases = [{'k': 3, 'arr': [1,2,3,-3,1,1,4,2,-3]},
              {'k': 5, 'arr': [2,3,5]},
              {'k': 10, 'arr': [2,3,5,1,9]},
              {'k': 1, 'arr': [2,3,5,1,9]},
              {'k': 1, 'arr': [-1,1]},
              {'k': 3, 'arr': [1,2,3,1,1,1,4,2,3]},
              {'k': 0, 'arr': [1,2,-3,1,0,-4,1,4,0,1]}]

for to in test_cases:
    arr = to['arr']
    k = to['k']
    print(find_all_subarr_with_given_sum_BFS(arr, k))

8
2
2
3
5
6
```

```
In [45]:
'''
Optimal solution

Note:
    If arr has zero & rev numbers, this is the optimal solution.

Time Complexity : O( 2n )
Space Complexity : O( 1 )
'''
def find_all_subarr_with_given_sum_OS(arr, k):
    n = len(arr)
    rsum = 0
    subarr_map = {}
    subarr_counter = 0

    for i in range(n):
        rsum += arr[i]

        rsum = rsum - k

        subarr_counter += rsum_map.get(rsum, 0)

        rsum_map[rsum] = rsum_map.get(rsum, 0) + 1

    return subarr_counter

In [46]:
test_cases = [{'k': 3, 'arr': [1,2,3,-3,1,1,4,2,-3]},
              {'k': 5, 'arr': [2,3,5]},
              {'k': 10, 'arr': [2,3,5,1,9]},
              {'k': 1, 'arr': [-1,1]},
              {'k': 3, 'arr': [1,2,3,1,1,1,4,2,3]},
              {'k': 0, 'arr': [1,2,-3,1,0,-4,1,4,0,1]}]

for to in test_cases:
    arr = to['arr']
    k = to['k']
    print(find_all_subarr_with_given_sum_OS(arr, k))

8
2
2
3
5
6
```

### Problem Statement 15: Maximum Subarray Sum in an Array (Kadane's Algorithm)

Given an integer array arr, find the contiguous subarray (containing at least one number) which has the largest sum and returns its sum and prints the subarray.

**Example 1:**  
Input: arr = [-2,1,-3,4,-1,2,1,-5,4]  
Output: 6  
Explanation: [-2,1,-3,4,-1,2,1] has the largest sum = 6.

**Examples 2:**  
Input: arr = [1]  
Output: 1  
Explanation: Array has only one element and which is giving positive sum of 1.

```
In [47]:
'''
Brute Force solution

Time Complexity : O( n^2 )
Space Complexity : O( 1 )
'''
def find_max_subarr_sum_BFS(arr):
    n = len(arr)
    max_subarr_sum = 0

    for i in range(0, n):
        rsum = 0

        for j in range(i, n):
            rsum += arr[j]
            max_subarr_sum = max(rsum, max_subarr_sum)

    return max_subarr_sum

In [48]:
test_cases = [{'arr': [-2,1,-3,4,-1,2,1,-5,4]},
              {'arr': [-2,-3,4,-1,-2,1,5,-3]},
              {'arr': [1]}]

for to in test_cases:
    arr = to['arr']
    print('arr:', arr, '\tMax sub arr sum:', find_max_subarr_sum_BFS(arr))

arr: [-2, 1, -3, 4, -1, 2, 1, -5, 4]    Max sub arr sum: 6
arr: [-2, -3, 4, -1, -2, 1, 5, -3]    Max sub arr sum: 7
arr: [1]                             Max sub arr sum: 1
```

```
In [49]:
'''
Optimal Solution (Kadane's Algorithm)

Time Complexity : O( n )
Space Complexity : O( 1 )
'''
def find_max_subarr_sum_OS(arr):
    n = len(arr)
    max_subarr_sum = curr_sum = arr[0]

    for i in range(1, n):
        curr_sum = max(arr[i], curr_sum + arr[i])

        max_subarr_sum = max(curr_sum, max_subarr_sum)

    return max_subarr_sum

In [50]:
test_cases = [{'arr': [-2,1,-3,4,-1,2,1,-5,4]},
              {'arr': [-2,-3,4,-1,-2,1,5,-3]},
              {'arr': [1]}]

for to in test_cases:
    arr = to['arr']
    print('arr:', arr, '\tMax sub arr sum:', find_max_subarr_sum_OS(arr))

arr: [-2, 1, -3, 4, -1, 2, 1, -5, 4]    Max sub arr sum: 6
arr: [-2, -3, 4, -1, -2, 1, 5, -3]    Max sub arr sum: 7
arr: [1]                             Max sub arr sum: 1
```

```
In [51]:
'''
Optimal solution (Kadane's Algorithm)

Time Complexity : O( n )
Space Complexity : O( 1 )
'''
def find_max_subarr_sum_OS2(arr):
    n = len(arr)
    curr_sum = arr[0]
    curr_side = curr_wide = 0
    max_subarr_sum = max_subarr_wide = 0

    for i in range(1, n):
        if arr[i] > curr_sum + arr[i]:
            curr_sum = arr[i]
            curr_side = curr_wide = i
        else:
            curr_sum = curr_sum + arr[i]
            curr_wide = i

        if curr_sum > max_subarr_sum:
            max_subarr_sum = curr_sum
            max_subarr_wide = curr_wide

    return max_subarr_sum, arr[max_subarr_wide : max_subarr_wide + 1]

In [52]:
test_cases = [{'arr': [-2,1,-3,4,-1,2,1,-5,4]},
              {'arr': [-2,-3,4,-1,-2,1,5,-3]},
              {'arr': [1]}]

for to in test_cases:
    arr = to['arr']
    print('arr:', arr, '\tMax sub arr:', find_max_subarr_sum_OS2(arr))

arr: [-2, 1, -3, 4, -1, 2, 1, -5, 4]    Max sub arr: 6 [4, -1, 2, 1]
arr: [-2, -3, 4, -1, -2, 1, 5, -3]    Max sub arr: 7 [4, -1, -2, 1, 5]
arr: [1]                             Max sub arr: 1 [1]
```

### Problem Statement 16: Two Sum - Check if a pair with given sum exists in Array

Given an array of integers and an integer target.

- 1st variant: Return YES if there exist two numbers such that their sum is equal to the target. Otherwise, return NO.
- 2nd variant: Return indices of the two numbers such that their sum is equal to the target. Otherwise, we return [-1, -1].

**Note:** You are not allowed to use the same element twice. Example: If the target is equal to 6 and num[1] = 3, then nums[1] + nums[1] = target is not a solution.</p></div>
<div data-bbox=



```
In [75]: """
Time Complexity : O(3n)
Space Complexity : O(1)
"""
def reverse_inplace(arr, i, j):
    while i < j:
        arr[i], arr[j] = arr[j], arr[i]
        i += 1
        j -= 1

def next_greater_permutation(arr):
    n = len(arr)

    #Step 1: Find the break point
    idx = -1
    for i in range(n-2, -1, -1):
        if arr[i] < arr[i+1]:
            idx = i
            break
    else:
        #If break point does not exist, reverse the whole array and return
        reverse_inplace(arr, 0, n-1)
        return arr

    #Step 2: Find the next greater element and swap it with arr[idx]
    for i in range(n-1, idx, -1):
        if arr[i] > arr[idx]:
            arr[i], arr[idx] = arr[idx], arr[i]
            break

    # Step 3: reverse the right half
    reverse_inplace(arr, idx+1, n-1)

    return arr
```

```
In [76]: test_cases = [{'arr': [1,3,2]},
                      {'arr': [3,2,1]]}

for tc in test_cases:
    arr = tc['arr']
    print('arr:', arr, end='')
    print(' ', 'Next Permutation:', next_greater_permutation(arr))

arr: [1, 3, 2] Next Permutation: [2, 1, 3]
arr: [3, 2, 1] Next Permutation: [1, 2, 3]
```

## Problem Statement 24: Set Matrix Zero

Given a matrix if an element in the matrix is 0 then you will have to set its entire column and row to 0 and then return the matrix.

Example 1:  
Input: matrix =   
[[1,1,1],  
[1,0,1],  
[1,1,1]]  
Output: matrix =   
[[1,0,1],  
[0,0,0],  
[1,0,1]]  
Explanation: Since matrix[2][2]=0. Therefore the 2nd column and 2nd row will be set to 0.

Example 2:  
Input: matrix =   
[[0,1,2,0],  
[3,4,5,2],  
[1,3,1,5]]  
Output: matrix =   
[[0,0,0,0],  
[0,4,5,0],  
[0,3,1,0]]  
Explanation: Since matrix[0][0]=0 and matrix[0][3]=0. Therefore 1st row, 1st column and 4th column will be set to 0

```
In [77]: """
Better solution
Time Complexity : O(2n)
Space Complexity : O(1)
"""
def set_zeros_B2(matrix):
    rows, columns = len(matrix), len(matrix[0])

    row_flag = [False] * rows
    col_flag = [False] * columns

    for r in range(rows):
        for c in range(columns):
            if matrix[r][c] == 0:
                row_flag[r] = True
                col_flag[c] = True

    for r in range(rows):
        for c in range(columns):
            if row_flag[r] == col_flag[c]:
                matrix[r][c] = 0

    # Not gonna count this
    for r in range(rows):
        print('\n')
        for c in range(columns):
            print(matrix[r][c], ' ', end='')

In [78]: test_cases = [{'matrix': [[1,1,1],
                                   [1,0,1],
                                   [1,1,1]]},
                      {'matrix': [[0,1,2,0],
                                   [3,4,5,2],
                                   [1,3,1,5]]},
                      {'matrix': [[1,1,1,1],
                                   [1,0,1,1],
                                   [1,1,0,1],
                                   [0,1,1,1]]}]

for tc in test_cases:
    matrix = tc['matrix']
    set_zeros_B2(matrix)
    print('\n', '#'*100, sep='')

1 0 1
0 0 0
#####

0 0 0 0
0 4 5 0
0 3 1 0
#####

0 0 0 1
0 0 0 0
0 0 0 0
0 0 0 0
#####

In [79]: """
Optimal solution
Time Complexity : O(2n)
Space Complexity : O(1)
"""
def set_zeros_O3(matrix):
    rows, columns = len(matrix), len(matrix[0])

    first_row_zero_flag = False

    #Step 1: Traverse the matrix and mark 1st row & col accordingly
    for r in range(rows):
        for c in range(columns):
            if matrix[r][c] == 0:
                matrix[0][c] = 0

                if r == 0:
                    first_row_zero_flag = True
                else:
                    matrix[r][0] = 0

    #Step 2: Mark with 0 from (1,1) to (n-1, m-1)
    for r in range(1, rows):
        for c in range(1, columns):
            if matrix[0][c] == 0 or matrix[r][0] == 0:
                matrix[r][c] = 0

    #Step 3: Mark the 1st col as 0s if [0][0] is 0
    if matrix[0][0] == 0:
        for r in range(rows):
            matrix[r][0] = 0

    #Step 4: Mark the 1st row as 0s if first_row_zero_flag is True
    if first_row_zero_flag == True:
        for c in range(1, columns):
            matrix[0][c] = 0

    # Not gonna count this
    for r in range(rows):
        print('\n')
        for c in range(columns):
            print(matrix[r][c], ' ', end='')

In [80]: test_cases = [{'matrix': [[1,1,1],
                                   [1,0,1],
                                   [1,1,1]]},
                      {'matrix': [[0,1,2,0],
                                   [3,4,5,2],
                                   [1,3,1,5]]},
                      {'matrix': [[1,1,1,1],
                                   [1,0,1,1],
                                   [1,1,0,1],
                                   [0,1,1,1]]},
                      {'matrix': [[1, 2, 3, 4],
                                   [ 5, 0, 7, 6],
                                   [0,10,11,12],
                                   [13,14,15, 0]]}]

for tc in test_cases:
    matrix = tc['matrix']
    set_zeros_O3(matrix)
    print('\n', '#'*100, sep='')

1 0 1
0 0 0
#####

0 0 0 0
0 4 5 0
0 3 1 0
#####

0 0 0 1
0 0 0 0
0 0 0 0
0 0 0 0
#####

0 0 0 0
#####

0 0 3 0
0 0 0 0
0 0 0 0
0 0 0 0
#####

In [81]: def reverse(arr):
    n = len(arr)
    i, j = 0, n-1
    while i < j:
        arr[i], arr[j] = arr[j], arr[i]
        i += 1
        j -= 1

def rotate(matrix, direction):
    rows, columns = len(matrix), len(matrix[0])

    if direction == 'LEFT':
        for r in range(rows):
            reverse(matrix[r])
        for r in range(rows):
            for c in range(r+1, columns):
                matrix[c][r], matrix[r][c] = matrix[c][r], matrix[r][c]
    else:
        for r in range(rows):
            for c in range(r+1, columns):
                matrix[r][c], matrix[c][r] = matrix[c][r], matrix[r][c]
        for r in range(rows):
            reverse(matrix[r])

    # Not gonna count this
    for r in range(rows):
        print('\n')
        for c in range(columns):
            print(matrix[r][c], ' ', end='')

In [82]: test_cases = [{'matrix': [[1,2,3],
                                   [4,5,6],
                                   [7,8,9]], 'direction': 'LEFT'},
                      {'matrix': [[1,2,3],
                                   [4,5,6],
                                   [7,8,9]], 'direction': 'RIGHT'},]

for tc in test_cases:
    matrix = tc['matrix']
    direction = tc['direction']
    rotate(matrix, direction)
    print('\n', '#'*100, sep='')

3 6 9
2 5 8
1 4 7
#####

7 4 1
8 5 2
9 6 3
#####

In [ ]:
```