Arrays

Problem Statement 1: Linear Search

Given an array, and an element num the task is to find if num is present in the given array or not. If present print the index of the element or print -1.

```
Example 1:
Input: arr[]= 1 2 3 4 5, num = 3
Output: 2
Explanation: 3 is present in the 2nd index

Example 2:
Input: arr[]= 5 4 3 2 1, num = 5
Output: 0
Explanation: 5 is present in the 0th index
```

In [1]:
```python
"""
Time Complexity  : O( n )
Space Complexity : O( 1 )
"""
def linear_search(arr, num):
    n = len(arr)

    for i in range(0, n):
        if arr[i] == num:
            return i

    return -1
```

In [2]:
```python
test_cases = [{'arr': [1,2,3,4,5], 'num': 3},
              {'arr': [5,4,3,2,1], 'num': 5},
              {'arr': [5,4,3,2,1], 'num': 7}]

for tc in test_cases:
    arr = tc['arr']
    num = tc['num']
    print('arr:', arr, 'num:', num, 'Index:', linear_search(arr, num))
```

```
arr: [1, 2, 3, 4, 5] num: 3 Index: 2
arr: [5, 4, 3, 2, 1] num: 5 Index: 0
arr: [5, 4, 3, 2, 1] num: 7 Index: -1
```

Problem Statement 2: Check if an Array is Sorted

Given an array of size n, write a program to check if the given array is sorted in (ascending / Increasing / Non-decreasing) order or not. If the array is sorted then return True, Else return False.

Note: *Two consecutive equal values are considered to be sorted.*

```
Example 1:
Input: N = 5, array[] = {1,2,3,4,5}
Output: True.
Explanation: The given array is sorted.

Example 2:
Input: N = 5, array[] = {5,4,6,7,8}
Output: False.
Explanation: The given array is not sorted. Element 5 is not smaller than or equal to its next
elements.
```

In [3]:
```python
"""
Time Complexity  : O( n )
Space Complexity : O( 1 )
"""
def is_sorted(arr):
    n = len(arr)

    for i in range(1, n):
        if arr[i-1] > arr[i]:
            return False

    return True
```

In [4]:
```python
test_cases = [{'arr': [1,2,3,4,5]},
              {'arr': [4,6,5,7,8]}]

for tc in test_cases:
    arr = tc['arr']
    print('arr:', arr, 'is sorted:', is_sorted(arr))
```

```
arr: [1, 2, 3, 4, 5] is sorted: True
arr: [4, 6, 5, 7, 8] is sorted: False
```

Problem Statement 3: Find the largest & smallest elements in an array

Given an array, find the largest and the smallest elements in the array.

```
Example 1:
Input: arr[] = {2,5,1,3,0};
Output: (0, 5)

Example2:
Input: arr[] = {8,10,5,7,9};
Output: (5, 10)
```

In [5]:
```python
"""
Time Complexity  : O( n )
Space Complexity : O( 1 )
"""
def find_min_max(arr):
    mini = maxi = arr[0]

    n = len(arr)
    for i in range(1, n):
        mini = min(arr[i], mini)
        maxi = max(arr[i], maxi)

    return mini, maxi
```

In [6]:
```python
test_cases = [{'arr': [2,5,1,3,0]},
              {'arr': [8,10,5,7,9]}]

for tc in test_cases:
    arr = tc['arr']
    print('arr:', arr, '(min, max):', find_min_max(arr))
```

```
arr: [2, 5, 1, 3, 0] (min, max): (0, 5)
arr: [8, 10, 5, 7, 9] (min, max): (5, 10)
```

Problem Statement 4: Find the second largest & smallest elements in an array

Given an array, find the second smallest and second largest element in the array. Print '-1' in the event that either of them doesn't exist.

```
Example 1:
Input: [1,2,4,7,7,5]
Output: (2, 5)

Example 2:
Input: [1]
Output: (-1, -1)
```

In [7]:
```python
"""
Brute force solution

Time Complexity  : O( n log n )  [For sorting the array]
Space Complexity : O( 1 )
"""
def find_2nd_mini_maxi_BFS(arr):
    arr.sort()

    #If it is guaranteed that the array has no duplicates
    #return arr[1], arr[-2]

    mini = float('inf')
    maxi = float('-inf')
    n = len(arr)

    for i in range(0, n):
        if arr[i] != arr[0]:
            mini = min(arr[i], mini)

        if arr[i] != arr[-1]:
            maxi = max(arr[i], maxi)

    mini = -1 if mini == float('inf') else mini
    maxi = -1 if maxi == float('-inf') else maxi

    return mini, maxi
```

In [8]:
```python
test_cases = [{'arr': [1]},
              {'arr': [1,1,1,1,1,1]},
              {'arr': [1,1,3]},
              {'arr': [1,2,4,7,7,5]},
              {'arr': [8,10,5,7,9]}]


print('arr:', test_cases[0]['arr'], '\t\t\t(min, max):', find_2nd_mini_maxi_BFS(arr = test_cases[0]['arr']))
print('arr:', test_cases[1]['arr'], '\t(min, max):',      find_2nd_mini_maxi_BFS(arr = test_cases[1]['arr']))
print('arr:', test_cases[2]['arr'], '\t\t\t(min, max):', find_2nd_mini_maxi_BFS(arr = test_cases[2]['arr']))
print('arr:', test_cases[3]['arr'], '\t(min, max):',      find_2nd_mini_maxi_BFS(arr = test_cases[3]['arr']))
print('arr:', test_cases[4]['arr'], '\t\t(min, max):',    find_2nd_mini_maxi_BFS(arr = test_cases[4]['arr']))
```

```
arr: [1]                          (min, max): (-1, -1)
arr: [1, 1, 1, 1, 1, 1]           (min, max): (-1, -1)
arr: [1, 1, 3]                    (min, max): (3, 1)
arr: [1, 2, 4, 5, 7, 7]           (min, max): (2, 5)
arr: [5, 7, 8, 9, 10]             (min, max): (7, 9)
```

In [9]:
```python
"""
Better solution

Time Complexity  : O( 2n )
Space Complexity : O( 1 )
"""
def find_2nd_mini_maxi_BS(arr):
    n = len(arr)

    mini = maxi = arr[0]
    for i in range(1, n):
        mini = min(arr[i], mini)
        maxi = max(arr[i], maxi)

    second_mini = float('inf')
    second_maxi = float('-inf')
    for i in range(0, n):
        if arr[i] != mini:
            second_mini = min(arr[i], second_mini)

        if arr[i] != maxi:
            second_maxi = max(arr[i], second_maxi)

    second_mini = -1 if second_mini == float('inf') else second_mini
    second_maxi = -1 if second_maxi == float('-inf') else second_maxi

    return second_mini, second_maxi
```

In [10]:
```python
test_cases = [{'arr': [1]},
              {'arr': [1,1,1,1,1,1]},
              {'arr': [1,1,3]},
              {'arr': [1,2,4,7,7,5]},
              {'arr': [8,10,5,7,9]}]


print('arr:', test_cases[0]['arr'], '\t\t\t(min, max):', find_2nd_mini_maxi_BS(arr = test_cases[0]['arr']))
print('arr:', test_cases[1]['arr'], '\t(min, max):',      find_2nd_mini_maxi_BS(arr = test_cases[1]['arr']))
print('arr:', test_cases[2]['arr'], '\t\t\t(min, max):', find_2nd_mini_maxi_BS(arr = test_cases[2]['arr']))
print('arr:', test_cases[3]['arr'], '\t(min, max):',      find_2nd_mini_maxi_BS(arr = test_cases[3]['arr']))
print('arr:', test_cases[4]['arr'], '\t\t(min, max):',    find_2nd_mini_maxi_BS(arr = test_cases[4]['arr']))
```

```
arr: [1]                          (min, max): (-1, -1)
arr: [1, 1, 1, 1, 1, 1]           (min, max): (-1, -1)
arr: [1, 1, 3]                    (min, max): (3, 1)
arr: [1, 2, 4, 7, 7, 5]           (min, max): (2, 5)
arr: [8, 10, 5, 7, 9]             (min, max): (7, 9)
```

In [11]:
```python
"""
Optimal solution

Time Complexity  : O( n )
Space Complexity : O( 1 )
"""
def find_2nd_mini_maxi_OS(arr):
    n = len(arr)

    mini, second_mini = arr[0], float('inf')
    second_maxi, maxi = float('-inf'), arr[0]

    for i in range(1, n):
        #Finds the second smallest element
        if arr[i] < mini:
            second_mini = mini
            mini = arr[i]
        elif arr[i] > mini and arr[i] < second_mini:
            second_mini = arr[i]

        #Finds the second largest element
        if arr[i] > maxi:
            second_maxi = maxi
            maxi = arr[i]
        elif arr[i] < maxi and arr[i] > second_maxi:
            second_maxi = arr[i]

    second_mini = -1 if second_mini == float('inf') else second_mini
    second_maxi = -1 if second_maxi == float('-inf') else second_maxi

    return second_mini, second_maxi
```

In [12]:
```python
test_cases = [{'arr': [1]},
              {'arr': [1,1,1,1,1,1]},
              {'arr': [1,1,3]},
              {'arr': [1,2,4,7,7,5]},
              {'arr': [8,10,5,7,9]}]


print('arr:', test_cases[0]['arr'], '\t\t\t(min, max):', find_2nd_mini_maxi_OS(arr = test_cases[0]['arr']))
print('arr:', test_cases[1]['arr'], '\t(min, max):',      find_2nd_mini_maxi_OS(arr = test_cases[1]['arr']))
print('arr:', test_cases[2]['arr'], '\t\t\t(min, max):', find_2nd_mini_maxi_OS(arr = test_cases[2]['arr']))
print('arr:', test_cases[3]['arr'], '\t(min, max):',      find_2nd_mini_maxi_OS(arr = test_cases[3]['arr']))
print('arr:', test_cases[4]['arr'], '\t\t(min, max):',    find_2nd_mini_maxi_OS(arr = test_cases[4]['arr']))
```

```
arr: [1]                          (min, max): (-1, -1)
arr: [1, 1, 1, 1, 1, 1]           (min, max): (-1, -1)
arr: [1, 1, 3]                    (min, max): (3, 1)
arr: [1, 2, 4, 7, 7, 5]           (min, max): (2, 5)
arr: [8, 10, 5, 7, 9]             (min, max): (7, 9)
```

Problem Statement 5: Reverse an array in-place

Given an array, reverse it in-place (i.e. without using extra space).

```
Example 1:
Input: [1,2,4,7,7,5]
Output: [5,7,7,4,2,1]

Example 2:
Input: [1]
Output: [1]
```

In [13]:
```python
"""
Time Complexity  : O( n )
Space Complexity : O( 1 )
"""
def inplace_reverse(arr):
    i = 0
    j = len(arr) - 1

    while i < j:
        arr[i], arr[j] = arr[j], arr[i]
        i += 1
        j -= 1
```

In [14]:
```python
test_cases = [{'arr': [1,2,4,7,7,5]},
              {'arr': [1]}]

for tc in test_cases:
    arr = tc['arr']
    inplace_reverse(arr)
    print('Reversed arr:', arr)
```

```
Reversed arr: [5, 7, 7, 4, 2, 1]
Reversed arr: [1]
```

Problem Statement 6: Move all Zeros to the end of the array

You are given an array of integers, your task is to move all the zeros in the array to the end of the array and move non-negative integers to the front **by maintaining their order**.

```
Example 1:
Input: 1 ,0 ,2 ,3 ,0 ,4 ,0 ,1
Output: 1 ,2 ,3 ,4 ,1 ,0 ,0 ,0
Explanation: All the zeros are moved to the end and non-negative integers are moved to front
by maintaining order

Example 2:
Input: 1,2,0,1,0,4,0
Output: 1,2,1,4,0,0,0
Explanation: All the zeros are moved to the end and non-negative integers are moved to front
by maintaining order
```

In [15]:
```python
"""
Time Complexity  : O( n )
Space Complexity : O( 1 )
"""
def move_zeroes(arr):
    n = len(arr)

    for i in range(0, n):
        if arr[i] == 0:
            j = i
            break

    for i in range(j+1, n):
        if arr[i] != 0:
            arr[i], arr[j] = arr[j], arr[i]
            j += 1
```

In [16]:
```python
test_cases = [{'arr': [1,0,2,3,0,4,0,1]},
              {'arr': [1,2,0,1,0,4,0,0]}]

for tc in test_cases:
    arr = tc['arr']
    move_zeroes(arr)
    print('New arr:', arr)
```

```
New arr: [1, 2, 3, 4, 1, 0, 0, 0]
New arr: [1, 2, 1, 4, 0, 0, 0, 0]
```

Problem Statement 7: Remove duplicates in-place from Sorted Array

Given an integer array sorted in **ascending order**, remove the duplicates in place such that each unique element appears only once. **The relative order of the elements should be kept the same**.

If there are k elements after removing the duplicates, then the first k elements of the array should hold the final result. It does not matter what you leave beyond the first k elements.

Note: *Return k after placing the final result in the first k slots of the array.*

```
Example 1:
Input: arr[1,1,2,2,2,3,3]
Output: arr[1,2,3,_,_,_,_]
Explanation: Total number of unique elements are 3, i.e [1,2,3] and Therefore return 3 after
assigning [1,2,3] in the beginning of the array.

Example 2:
Input: arr[1,1,1,2,2,3,3,3,3,4,4]
Output: arr[1,2,3,4,_,_,_,_,_,_,_]
Explanation: Total number of unique elements are 4, i.e[1,2,3,4] and Therefore return 4 after
assigning [1,2,3,4] in the beginning of the array.
```

In [17]:
```python
def remove_duplicates(arr):
    n = len(arr)

    j = 0
    for i in range(0, n):
        if arr[i] != arr[j]:
            j += 1
            arr[i], arr[j] = arr[j], arr[i]

    return j+1
```

In [18]:
```python
test_cases = [{'arr': [1,1,2,2,2,3,3]},
              {'arr': [1,1,1,2,2,3,3,3,3,4,4]}]

arr = test_cases[0]['arr']
print(f"arr: {arr}\t\tUnique Elements: {remove_duplicates(arr)}\tNew arr: {arr}")
arr = test_cases[1]['arr']
print(f"arr: {arr}\tUnique Elements: {remove_duplicates(arr)}\tNew arr: {arr}")
```

```
arr: [1, 1, 2, 2, 2, 3, 3]              Unique Elements: 3      New arr: [1, 2, 3, 2, 2, 1, 3]
arr: [1, 1, 1, 2, 2, 3, 3, 3, 3, 4, 4]  Unique Elements: 4      New arr: [1, 2, 3, 4, 2, 1, 3, 3, 3, 1, 4]
```

Problem Statement 8: Rotate array by K elements

Given an array of integers, rotating array of elements by k elements either left or right.

```
Example 1:
Input: arr = [1,2,3,4,5,6,7], k = 2, right
Output: arr = [6,7,1,2,3,4,5]
Explanation: array is rotated to right by 2 position.

Example 2:
Input: arr = [3,7,8,9,10,11], k = 3, left
Output: arr = [9,10,11,3,7,8]
Explanation: Array is rotated to right by 3 position.
```

In [19]:
```python
"""
Better solution

Time Complexity  : O( n )
Space Complexity : O( k )
"""
def rotate_BS(arr, k, direction):
    n = len(arr)
    k = k % n

    if direction == 'LEFT':
        tmp_arr = arr[:k]

        for i in range(k, n):
            arr[i - k] = arr[i]

        for i in range(n-k, n):
            arr[i] = tmp_arr[i - k]
    else:
        tmp_arr = arr[n-k:]

        for i in range(n-k-1, -1, -1):
            arr[i + k] = arr[i]

        for i in range(0, k):
            arr[i] = tmp_arr[i]
```

In [20]:
```python
test_cases = [{'arr': [1,2,3,4,5,6,7], 'k': 2, 'direction': 'RIGHT'},
              {'arr': [3,7,8,9,10,11], 'k': 3, 'direction': 'LEFT'}]

for tc in test_cases:
    print(f"arr: {tc['arr']}\tk: {tc['k']}\tdirection:  {tc['direction']}\t", end='')
    rotate_BS(arr=tc['arr'], k=tc['k'], direction=tc['direction'])
    print(f"New arr: {tc['arr']}")
```

```
arr: [1, 2, 3, 4, 5, 6, 7]      k: 2    direction:  RIGHT       New arr: [6, 7, 1, 2, 3, 4, 5]
arr: [3, 7, 8, 9, 10, 11]       k: 3    direction:  LEFT        New arr: [9, 10, 11, 3, 7, 8]
```

In [21]:
```python
"""
Optimal solution

Time Complexity  : O( 2n )
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 rotate_OS(arr, k, direction):
    n = len(arr)
    k = k % n

    if direction == 'LEFT':
        reverse_inplace(arr, 0, k-1)
        reverse_inplace(arr, k, n-1)
        reverse_inplace(arr, 0, n-1)
    else:
        reverse_inplace(arr, 0, n-k-1)
        reverse_inplace(arr, n-k, n-1)
        reverse_inplace(arr, 0, n-1)
```

In [22]:
```python
test_cases = [{'arr': [1,2,3,4,5,6,7], 'k': 2, 'direction': 'RIGHT'},
              {'arr': [3,7,8,9,10,11], 'k': 3, 'direction': 'LEFT'}]

for tc in test_cases:
    print(f"arr: {tc['arr']}\tk: {tc['k']}\tdirection:  {tc['direction']}\t", end='')
    rotate_OS(arr=tc['arr'], k=tc['k'], direction=tc['direction'])
    print(f"New arr: {tc['arr']}")
```

```
arr: [1, 2, 3, 4, 5, 6, 7]      k: 2    direction:  RIGHT       New arr: [6, 7, 1, 2, 3, 4, 5]
arr: [3, 7, 8, 9, 10, 11]       k: 3    direction:  LEFT        New arr: [9, 10, 11, 3, 7, 8]
```

Problem Statement 9: Count max consecutive 1's in an array

Given an array that contains only 1's and 0's, return the count of maximum consecutive 1's in the array.

```
Example 1:
Input: arr = [1, 1, 0, 1, 1, 1]
Output: 3
Explanation: There are 2 consecutive 1's and 3 consecutive 1's in the array out of which
maximum is 3.

Example 2:
Input: arr = [1, 0, 1, 1, 0, 1]
Output:
Explanation: There are 2 consecutive 1's in the array.
```

In [23]:
```python
"""
Time Complexity  : O( n )
Space Complexity : O( 1 )
"""
def find_max_consecutive_ones(arr):
    n = len(arr)
    one_count = 0
    max_one_count = 0

    for i in range(0, n):
        if arr[i] == 1:
            one_count += 1
        else:
            one_count = 0

        max_one_count = max(one_count, max_one_count)

    return max_one_count
```

In [24]:
```python
test_cases = [{'arr': [1,1,0,1,1,1]},
              {'arr': [1,0,1,1,0,1]}]

for tc in test_cases:
    print('arr:', tc['arr'], "Max consecutive 1's:", find_max_consecutive_ones(arr = tc['arr']))
```

```
arr: [1, 1, 0, 1, 1, 1] Max consecutive 1's: 3
arr: [1, 0, 1, 1, 0, 1] Max consecutive 1's: 2
```

Problem Statement 10: Find the number that appears once, while others appear twice

Given a non-empty array of integers arr, every element appears twice except for one. Find that single one.

```
Example 1:
Input Format: arr = [2,2,1]
Result: 1
Explanation: In this array, only the element 1 appear once and so it is the answer.

Example 2:
Input Format: arr = [4,1,2,1,2]
Result: 4
Explanation: In this array, only element 4 appear once and the other elements appear twice.
```

In [25]:
```python
"""
Better solution

Time Complexity  : O( n + n/2 + 1 ) ~ O( n )
Space Complexity : O( n/2 + 1 )     ~ O( n )
"""
def get_single_element_BS(arr):
    n = len(arr)
    freq_map = dict()

    for i in range(0, n):
        freq_map[ arr[i] ] = freq_map.get(arr[i], 0) + 1

    for num, freq in freq_map.items():
        if freq == 1:
            return num

    return -1
```

In [26]:
```python
test_cases = [{'arr': [2,2,1]},
              {'arr': [4,1,2,1,2]}]


print('arr:', test_cases[0]['arr'], "\t\tNum that appears once:", get_single_element_BS(arr = test_cases[0]['ar
print('arr:', test_cases[1]['arr'], "\tNum that appears once:", get_single_element_BS(arr = test_cases[1]['arr'
```

```
arr: [2, 2, 1]          Num that appears once: 1
arr: [4, 1, 2, 1, 2]    Num that appears once: 4
```

In [27]:
```python
"""
Optimal solution

Note:
    XOR Property 1: a ^ a = 0
    XOR Property 2: 0 ^ a = a

Time Complexity  : O( n )
Space Complexity : O( 1 )
"""
def get_single_element_OS(arr):
    n = len(arr)
    result = 0

    for i in range(0, n):
        result = result ^ arr[i]

    return result
```

In [28]:
```python
test_cases = [{'arr': [2,2,1]},
              {'arr': [4,1,2,1,2]}]


print('arr:', test_cases[0]['arr'], "\t\tNum that appears once:", get_single_element_OS(arr = test_cases[0]['ar
print('arr:', test_cases[1]['arr'], "\tNum that appears once:", get_single_element_OS(arr = test_cases[1]['arr'
```

```
arr: [2, 2, 1]          Num that appears once: 1
arr: [4, 1, 2, 1, 2]    Num that appears once: 4
```

Problem Statement 11: Find the missing number in an array

Given an integer N and an array of size N-1 containing N-1 numbers between 1 to N. Find the number (between 1 to N), that is not present in the given array.

```
Example 1:
Input Format: N = 5, arr = [1,2,4,5]
Result: 3
Explanation: In the given array, number 3 is missing. So, 3 is the answer.

Example 2:
Input Format: N = 3, arr = [1,3]
Result: 2
Explanation: In the given array, number 2 is missing. So, 2 is the answer.
```

In [29]:
```python
"""
Time Complexity  : O( n ) [For summing the array]
Space Complexity : O( 1 )
"""
def missing_number(arr, n):
    return (n*(n+1))//2 - sum(arr)
```

In [30]:
```python
test_cases = [{'arr': [1,2,4,5], 'n': 5},
              {'arr': [1,3], 'n': 3}]

arr = test_cases[0]['arr']
n = test_cases[0]['n']
print('arr:', arr, "\tn:", n, "\tMissing num:", missing_number(arr, n))

arr = test_cases[1]['arr']
n = test_cases[1]['n']
print('arr:', arr, "\t\tn:", n, "\tMissing num:", missing_number(arr, n))
```

```
arr: [1, 2, 4, 5]       n: 5    Missing num: 3
arr: [1, 3]             n: 3    Missing num: 2
```

In [31]:
```python
"""
Note:
    XOR Property 1: a ^ a = 0
    XOR Property 2: 0 ^ a = a

Time Complexity  : O( n )
Space Complexity : O( 1 )
"""
def missing_number(arr, n):
    xor1 = 0
    xor2 = 0

    for i in range(0, n-1):
        xor1 ^= arr[i]
        xor2 ^= i+1

    xor2 ^= n
    return xor1 ^ xor2
```

In [32]:
```python
test_cases = [{'arr': [1,2,4,5], 'n': 5},
              {'arr': [1,3], 'n': 3}]

arr = test_cases[0]['arr']
n = test_cases[0]['n']
print('arr:', arr, "\tn:", n, "\tMissing num:", missing_number(arr, n))

arr = test_cases[1]['arr']
n = test_cases[1]['n']
print('arr:', arr, "\t\tn:", n, "\tMissing num:", missing_number(arr, n))
```

```
arr: [1, 2, 4, 5]       n: 5    Missing num: 3
arr: [1, 3]             n: 3    Missing num: 2
```

Problem Statement 12: Union of Two Sorted Arrays

Given two sorted arrays, arr1, and arr2 of size n and m. Find the union of two sorted arrays.

Note: *Elements in the union should be in ascending order.*

```
Example 1:
Input: arr1 = [1,2,3,4,5], arr2 = [2,3,4,4,5]
Output: [1,2,3,4,5]

Example 2:
Input: arr1 = [1,2,3,4,5,6,7,8,9,10], arr2 = [2,3,4,4,5,11,12]
Output: [1,2,3,4,5,6,7,8,9,10,11,12]
```

In [33]:
```python
"""
Better solution

Time Complexity  : O( 2(n+m) )
Space Complexity : O( n+m )
"""
def find_union_BS(arr1, arr2):
    result = set()

    for num in arr1:
        result.add(num)

    for num in arr2:
        result.add(num)

    #Set to list: O(N)
    return list(result)
```

In [34]:
```python
test_cases = [{'arr1': [1,2,3,4,5], 'arr2': [2,3,4,4,5]},
              {'arr1': [1,2,3,4,5,6,7,8,9,10], 'arr2': [2,3,4,4,5,11,12]}]

arr1 = test_cases[0]['arr1']
arr2 = test_cases[0]['arr2']
print('arr1:', arr1, "\t\t      arr2:", arr2, "\t    Union:", find_union_BS(arr1, arr2))

arr1 = test_cases[1]['arr1']
arr2 = test_cases[1]['arr2']
print('arr1:', arr1, "arr2:", arr2, "Union:", find_union_BS(arr1, arr2))
```

```
arr1: [1, 2, 3, 4, 5]                  arr2: [2, 3, 4, 4, 5]         Union: [1, 2, 3, 4, 5]
arr1: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] arr2: [2, 3, 4, 4, 5, 11, 12] Union: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,
12]
```

In [35]:
```python
"""
Optimal solution

Time Complexity  : O( n+m )
Space Complexity : O( 1 )
"""
def find_union_OS(arr1, arr2):
    i = j =0
    n, m = len(arr1), len(arr2)
    result = []

    while i < n and j < m:
        if arr1[i] <= arr2[j]:
            if len(result) == 0 or result[-1] != arr1[i]:
                result.append( arr1[i] )
            i += 1
        else:
            if len(result) == 0 or result[-1] != arr2[j]:
                result.append( arr2[j] )
            j += 1

    while i < n:
        if len(result) == 0 or result[-1] != arr1[i]:
            result.append( arr1[i] )
        i += 1

    while j < m:
        if len(result) == 0 or result[-1] != arr2[j]:
            result.append( arr2[j] )
        j += 1

    return result
```

In [36]:
```python
test_cases = [{'arr1': [1,2,3,4,5], 'arr2': [2,3,4,4,5]},
              {'arr1': [1,2,3,4,5,6,7,8,9,10], 'arr2': [2,3,4,4,5,11,12]}]

arr1 = test_cases[0]['arr1']
arr2 = test_cases[0]['arr2']
print('arr1:', arr1, "\t\t      arr2:", arr2, "\t    Union:", find_union_OS(arr1, arr2))

arr1 = test_cases[1]['arr1']
arr2 = test_cases[1]['arr2']
print('arr1:', arr1, "arr2:", arr2, "Union:", find_union_OS(arr1, arr2))
```

```
arr1: [1, 2, 3, 4, 5]                  arr2: [2, 3, 4, 4, 5]         Union: [1, 2, 3, 4, 5]
arr1: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] arr2: [2, 3, 4, 4, 5, 11, 12] Union: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,
12]
```

Problem Statement 13: Longest Subarray with sum K

Given an array and a sum k, we need to print the length of the longest subarray that sums to k.

```
Example 1:
Input: arr = [2,3,5], k = 5
Result: 2
Explanation: The longest subarray with sum 5 is [2, 3]. And its length is 2.

Example 2:
Input: arr = [2,3,5,1,9], k = 10
Result: 3
Explanation: The longest subarray with sum 10 is [2, 3, 5]. And its length is 3.

Example 3:
Input: arr = [-1, 1, 1], k = 1
Result: 3
Explanation: The longest subarray with sum 1 is [-1, 1, 1]. And its length is 3.
```

In [37]:
```python
"""
Brute force solution

Time Complexity  : O( n^2 )
Space Complexity : O( 1 )
"""
def find_longest_subarr_len_BFS(arr, k):
    n = len(arr)
    longest_subarr_len = float('-inf')

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

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

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

    return longest_subarr_len
```

In [38]:
```python
test_cases = [{'k': 5, 'arr': [2,3,5]},
              {'k': 10, 'arr': [2,3,5,1,9]},
              {'k': 1, 'arr': [-1,1,1]},
              {'k': 3, 'arr': [1,2,3,1,1,1,1,4,2,3]},
              {'k': 0, 'arr': [1,2,-3,1,0,-4,1,4,0,1]}]

for tc in test_cases:
    arr = tc['arr']
    k = tc['k']
    print(find_longest_subarr_len_BFS(arr, k))
```

```
2
3
3
3
8
```

In [39]:
```python
"""
Better solution

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

Time Complexity  : O( n )
Space Complexity : O( n )
"""
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
        if remainder in running_sum_map:
            length = i - running_sum_map[ remainder ]
            longest_subarr_len = max(length, longest_subarr_len)

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

    return longest_subarr_len
```

In [40]:
```python
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': [1,2,3,1,1,1,1,4,2,3]},
              {'k': 7, 'arr': [1,2,-3,1,0,-4,1,4,0,1,0,4,3]},
              {'k': 0, 'arr': [-1,0,1,1,-1,-1,0]}]

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

```
2
2
3
2
3
12
6
```

In [41]:
```python
"""
Optimal solution

Note:
    If arr has zero & +ev 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 j < n:
        while i <= j and flexi_sum > k:
            flexi_sum -= arr[i]
            i += 1

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

        j += 1

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

    return longest_subarr_len
```

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

for tc in test_cases:
    arr = tc['arr']
    k = tc['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]:
```python
"""
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]:
```python
test_cases = [{'k': 3, 'arr': [1,2,3,-3,1,1,1,4,2,-3]},
              {'k': 5, 'arr': [2,3,5]},
              {'k': 10, 'arr': [2,3,5,1,9]},
              {'k': 1, 'arr': [-1,1,1]},
              {'k': 3, 'arr': [1,2,3,1,1,1,1,4,2,3]},
              {'k': 0, 'arr': [1,2,-3,1,0,-4,1,4,0,1]}]

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

```
8
2
2
3
5
6
```

In [45]:
```python
"""
Optimal solution

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

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

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

        rem = rsum - k

        subarr_counter += rsum_map.get(rem, 0)

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

    return subarr_counter
```

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

for tc in test_cases:
    arr = tc['arr']
    k = tc['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: [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]:
```python
"""
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]:
```python
test_cases = [{'arr': [-2,1,-3,4,-1,2,1,-5,4]},
              {'arr': [-2,-3,4,-1,-2,1,5,-3]},
              {'arr': [1]}]

for tc in test_cases:
    arr = tc['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]:
```python
"""
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]:
```python
test_cases = [{'arr': [-2,1,-3,4,-1,2,1,-5,4]},
              {'arr': [-2,-3,4,-1,-2,1,5,-3]},
              {'arr': [1]}]

for tc in test_cases:
    arr = tc['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]:
```python
"""
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_sidx = curr_eidx = 0

    max_subarr_sum = arr[0]
    max_subarr_sidx = max_subarr_eidx = 0

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

        if curr_sum > max_subarr_sum:
            max_subarr_sum = curr_sum

            max_subarr_sidx = curr_sidx
            max_subarr_eidx = curr_eidx

    return max_subarr_sum, arr[max_subarr_sidx : max_subarr_eidx + 1]
```

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

for tc in test_cases:
    arr = tc['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 will 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>

```
Example 1:
Input:  arr = [2,6,5,8,11], target = 14
Output: YES    (for 1st variant)
        [1, 3] (for 2nd variant)

Example 2:
Input:  arr = [2,6,5,8,11], target = 15
Result: NO       (for 1st variant)
        [-1, -1] (for 2nd variant)
```

In [53]:
```python
"""
Better solution

Time Complexity  : O( n )
Space Complexity : O( n )
"""

def two_sum_BS(arr, target):
    n = len(arr)
    hashmap = dict()

    for i in range(0, n):
        rem = target - arr[i]
        if rem in hashmap:
            return hashmap[ rem ], i

        hashmap[ arr[i] ] = i

    return -1, -1
```

In [54]:
```python
test_cases = [{'arr': [2,6,5,8,11], 'target': 14},
              {'arr': [2,6,5,8,11], 'target': 15}]

for tc in test_cases:
    arr = tc['arr']
    target = tc['target']
    print('arr:', arr, 'target:', target, '2 Sum:', two_sum_BS(arr, target))
```

```
arr: [2, 6, 5, 8, 11] target: 14 2 Sum: (1, 3)
arr: [2, 6, 5, 8, 11] target: 15 2 Sum: (-1, -1)
```

In [55]:
```python
"""
Optimal (space) solution

Time Complexity  : O( n log n )
Space Complexity : O( 1 )
"""

def two_sum_OS(arr, target):
    arr = sorted(enumerate(arr), key = lambda x: x[1])

    n = len(arr)
    i, j = 0, n-1

    while i < j:
        if arr[i][1] + arr[j][1] == target:
            return arr[i][0], arr[j][0]
        elif arr[i][1] + arr[j][1] < target:
            i += 1
        else:
            j -= 1

    return (-1, -1)
```

In [56]:
```python
test_cases = [{'arr': [2,6,5,8,11], 'target': 14},
              {'arr': [2,6,5,8,11], 'target': 15}]

for tc in test_cases:
    arr = tc['arr']
    target = tc['target']
    print('arr:', arr, 'target:', target, '2 Sum:', two_sum_OS(arr, target))
```

```
arr: [2, 6, 5, 8, 11] target: 14 2 Sum: (1, 3)
arr: [2, 6, 5, 8, 11] target: 15 2 Sum: (-1, -1)
```

Problem Statement 17: 3 Sum - Find triplets that add up to a zero

Given an integer array nums, return all the triplets [nums[i], nums[j], nums[k]] such that i!=j, i!=k, and j!=k, and nums[i] + nums[j] + nums[k] == 0.

Note: The solution set must not contain duplicate triplets.

```
Example 1:
Input: nums = [-1,0,1,2,-1,-4]
Output: [[-1,-1,2],[-1,0,1]]

Example 2:
Input: nums = [-1,0,1,0]
Output: [[-1,0,1]]
```

In [57]:
```python
"""
Optimal (space) solution

Time Complexity  : O( n log n ) + O( n^2 )
Space Complexity : O( 1 )
"""
def three_sum(arr):
    arr.sort()

    n = len(arr)
    ans = []

    for i in range(n):
        if i != 0 and arr[i-1] == arr[i]:
            continue

        j = i+1
        k = n-1

        while j < k:
            total_sum = arr[i] + arr[j] + arr[k]

            if total_sum < 0:
                j += 1
            elif total_sum > 0:
                k -= 1
            else:
                ans.append( (arr[i], arr[j], arr[k]) )

                j += 1
                k -= 1

                while j < k and arr[j] == arr[j - 1]:
                    j += 1
                while j < k and arr[k] == arr[k + 1]:
                    k -= 1

    return ans
```

In [58]:
```python
test_cases = [{'arr': [-1,0,1,2,-1,-4]},
              {'arr': [-1,0,1,0]}]

for tc in test_cases:
    arr = tc['arr']
    print('arr:', arr, '3 Sum:', three_sum(arr))
```

```
arr: [-4, -1, -1, 0, 1, 2] 3 Sum: [(-1, -1, 2), (-1, 0, 1)]
arr: [-1, 0, 0, 1] 3 Sum: [(-1, 0, 1)]
```

Problem Statement 18: 4 Sum - Find quads that add up to a target value

Given an array of N integers, your task is to find unique quads that add up to give a target value. In short, you need to return an array of all the unique quadruplets [arr[a], arr[b], arr[c], arr[d]] such that their sum is equal to a given target.

Note: The solution set must not contain duplicate quadruplets.

```
Example 1:
Input: arr = [1,0,-1,0,-2,2], target = 0
Output: [[-2,-1,1,2],[-2,0,0,2],[-1,0,0,1]]

Example 2:
Input: arr = [4,3,3,4,4,2,1,2,1,1], target = 9
Output: [[1,1,3,4],[1,2,2,4],[1,2,3,3]]
```

In [59]:
```python
"""
Optimal (space) solution

Time Complexity  : O( n log n ) + O( n^3 )
Space Complexity : O( 1 )
"""
def four_sum(arr, target):
    arr.sort()

    n = len(arr)
    ans = []

    for i in range(0, n):
        if i != 0 and arr[i-1] == arr[i]:
            continue

        for j in range(i+1, n):
            if j != 1 and arr[j-1] == arr[j]:
                continue

            k = j+1
            l = n-1

            while k < l:
                target_sum = arr[i] + arr[j] + arr[k] + arr[l]

                if target_sum < target:
                    k += 1
                elif target_sum > target:
                    l -= 1
                else:
                    ans.append( [arr[i], arr[j], arr[k], arr[l]] )
                    k += 1
                    l -= 1

                    while k < l and arr[k-1] == arr[k]:
                        k += 1
                    while k < l and arr[l] == arr[l+1]:
                        l -= 1

    return ans
```

In [60]:
```python
test_cases = [{'arr': [1,0,-1,0,-2,2], 'target': 0},
              {'arr': [4,3,3,4,4,2,1,2,1,1], 'target': 9}]

for tc in test_cases:
    arr = tc['arr']
    target = tc['target']
    print('arr:', arr, '4 Sum:', four_sum(arr, target))
```

```
arr: [-2, -1, 0, 0, 1, 2] 4 Sum: [[-2, -1, 1, 2], [-2, 0, 0, 2], [-1, 0, 0, 1]]
arr: [1, 1, 1, 2, 2, 3, 3, 4, 4, 4] 4 Sum: [[1, 1, 3, 4], [1, 2, 2, 4], [1, 2, 3, 3]]
```

Problem Statement 19: Buy And Sell Stocks

You are given an array of prices where prices[i] is the price of a given stock on an ith day. You want to maximize your profit by choosing a single day to buy one stock and choosing a different day in the future to sell that stock. Return the maximum profit you can achieve from this transaction. If you cannot achieve any profit, return 0.

```
Example 1:
Input: prices = [7,1,5,3,6,4]
Output: 5
Explanation: Buy on day 2 (price = 1) and sell on day 5 (price = 6), profit = 6-1 = 5.
Note: That buying on day 2 and selling on day 1 is not allowed because you must buy before you
sell.

Example 2:
Input: prices = [7,6,4,3,1]
Output: 0
Explanation: In this case, no transactions are done and the max profit = 0.
```

In [61]:
```python
"""
Time Complexity  : O( n )
Space Complexity : O( 1 )
"""
def find_max_profit(arr):
    n = len(arr)
    min_price = float('inf')
    max_profit = float('-inf')

    for i in range(0,n):
        min_price = min(arr[i], min_price)
        max_profit = max(arr[i] - min_price, max_profit)

    return max_profit
```

In [62]:
```python
test_cases = [{'arr': [7,1,5,3,6,4]},
              {'arr': [7,6,4,3,1]}]

for tc in test_cases:
    arr = tc['arr']
    print('arr:', arr, 'Max profit:', find_max_profit(arr))
```

```
arr: [7, 1, 5, 3, 6, 4] Max profit: 5
arr: [7, 6, 4, 3, 1] Max profit: 0
```

Problem Statement 20: Leaders in an Array

Given an array, print all the elements which are leaders. A Leader is an element that is greater than all of the elements on its right side in the array.

```
Example 1:
Input: arr = [4, 7, 1, 0]
Output: [7, 1, 0]
Explanation: Rightmost element is always a leader. 7 and 1 are greater than the elements in
their right side.

Example 2:
Input: arr = [10, 22, 12, 3, 0, 6]
Output: [22, 12, 6]
Explanation: 6 is a leader. In addition to that, 12 is greater than all the elements in its
right side (3, 0, 6), also 22 is greater than 12, 3, 0, 6.
```

In [63]:
```python
"""
Time Complexity  : O( 2n )
Space Complexity : O( 1 )
"""
def find_leaders(arr):
    n = len(arr)
    maxi = float('-inf')
    leaders = []

    for i in range(n-1, -1, -1):
        if arr[i] > maxi:
            leaders.append(arr[i])
            maxi = arr[i]

    inplace_reverse(leaders)

    return leaders
```

In [64]:
```python
test_cases = [{'arr': [4, 7, 1, 0]},
              {'arr': [10, 22, 12, 3, 0, 6]}]

for tc in test_cases:
    arr = tc['arr']
    print('arr:', arr, 'Leaders:', find_leaders(arr))
```

```
arr: [4, 7, 1, 0] Leaders: [7, 1, 0]
arr: [10, 22, 12, 3, 0, 6] Leaders: [22, 12, 6]
```

Problem Statement 21: Find the Majority Element (Moore's Voting Algorithm)

Given an array of N integers, write a program to return an element that occurs more than N/2 times (majority element) in the given array. Return -1 if no such element exists in the array.

```
Example 1:
Input: arr = [3,2,3]
Output: 3

Example 2:
Input:  arr = [2,2,1,1,1,2,2]
Output: 2

Example 3:
Input:  arr = [4,4,2,4,3,4,4,3,2,4]
Output: 4
```

In [65]:
```python
"""
Brute force solution

Time Complexity  : O( n log n )
Space Complexity : O( 1 )
"""
def find_majority_element_BFS(arr):
    arr.sort()

    n = len(arr)
    candidate = arr[n//2]
    counter = 0

    for i in range(0, n):
        if arr[i] == candidate:
            counter += 1

    return candidate if counter > n//2 else -1
```

In [66]:
```python
test_cases = [{'arr': [3,2,3]},
              {'arr': [2,2,1,1,1,2,2]},
              {'arr': [4,4,2,4,3,4,4,3,2,4]}]

for tc in test_cases:
    arr = tc['arr']
    print('arr:', arr, 'Majority element:', find_majority_element_BFS(arr))
```

```
arr: [2, 3, 3] Majority element: 3
arr: [1, 1, 1, 2, 2, 2, 2] Majority element: 2
arr: [2, 2, 3, 3, 4, 4, 4, 4, 4, 4] Majority element: 4
```

In [67]:
```python
"""
Better solution

Time Complexity  : O( 2n )
Space Complexity : O( n )
"""
def find_majority_element_BS(arr):
    n = len(arr)
    hashmap = dict()

    for i in range(0,n):
        hashmap[ arr[i] ] = hashmap.get(arr[i], 0) + 1

    for n, f in hashmap.items():
        if f > n//2:
            return n

    return -1
```

In [68]:
```python
test_cases = [{'arr': [3,2,3]},
              {'arr': [2,2,1,1,1,2,2]},
              {'arr': [4,4,2,4,3,4,4,3,2,4]}]

for tc in test_cases:
    arr = tc['arr']
    print('arr:', arr, 'Majority element:', find_majority_element_BS(arr))
```

```
arr: [3, 2, 3] Majority element: 3
arr: [2, 2, 1, 1, 1, 2, 2] Majority element: 2
arr: [4, 4, 2, 4, 3, 4, 4, 3, 2, 4] Majority element: 4
```

In [69]:
```python
"""
Optimal solution (Moore's Voting Algorithm)

Time Complexity  : O( 2n )
Space Complexity : O( 1 )
"""
def find_majority_element_OS(arr):
    n = len(arr)

    candidate = None
    count = 0

    for i in range(0,n):
        if count == 0:
            candidate = arr[i]
            count = 1
        elif arr[i] == candidate:
            count += 1
        else:
            count -= 1

    count = 0
    for i in range(0,n):
        if arr[i] == candidate:
            count += 1

    return candidate if count > n//2 else -1
```

In [70]:
```python
test_cases = [{'arr': [3,2,3]},
              {'arr': [2,2,1,1,1,2,2]},
              {'arr': [4,4,2,4,3,4,4,3,2,4]}]

for tc in test_cases:
    arr = tc['arr']
    print('arr:', arr, 'Majority element:', find_majority_element_OS(arr))
```

```
arr: [3, 2, 3] Majority element: 3
arr: [2, 2, 1, 1, 1, 2, 2] Majority element: 2
arr: [4, 4, 2, 4, 3, 4, 4, 3, 2, 4] Majority element: 4
```

Problem Statement 22: Find the Majority Element II (Moore's Voting Algorithm)

Given an array of N integers, write a program to return the element(s) that occurs more than N/3 times (majority element) in the given array. Return [] if no such element exists in the array.

```
Example 1:
Input: arr = [1,2,2,3,2]
Output: [2]
Explanation: Here we can see that the Count(1) = 1, Count(2) = 3 and Count(3) = 1. Therefore,
the count of 2 is greater than N/3 times. Hence, 2 is the answer.

Example 2:
Input: arr = [11,33,33,11,33,11]
Output: [11, 33]
Explanation: Here we can see that the Count(11) = 3 and Count(33) = 3. Therefore, the count of
both 11 and 33 is greater than N/3 times. Hence, 11 and 33 is the answer.
```

In [71]:
```python
"""
Better solution

Time Complexity  : O( n )
Space Complexity : O( n )
"""
def find_majority_element_2_BS(arr):
    n = len(arr)
    majority_elements = []
    num_freq_map = dict()

    for i in range(0,n):
        num_freq_map[ arr[i] ] = num_freq_map.get(arr[i], 0) + 1

        if num_freq_map[arr[i]] == n//3+1:
            majority_elements.append( arr[i] )

        if len(majority_elements) == 2:
            break

    return majority_elements
```

In [72]:
```python
test_cases = [{'arr': [1,2,2,3,2]},
              {'arr': [11,33,33,11,33,11]},
              {'arr': [2,1,1,3,1,4,5,6]}]

for tc in test_cases:
    arr = tc['arr']
    print('arr:', arr, 'Majority element:', find_majority_element_2_BS(arr))
```

```
arr: [1, 2, 2, 3, 2] Majority element: [2]
arr: [11, 33, 33, 11, 33, 11] Majority element: [33, 11]
arr: [2, 1, 1, 3, 1, 4, 5, 6] Majority element: [1]
```

In [73]:
```python
"""
Optimal solution (Moore's Voting Algorithm)

Time Complexity  : O( 2n )
Space Complexity : O( 1 )
"""
def find_majority_element_2_OS(arr):
    n = len(arr)
    majority_elements = []
    candidate1 = candidate2 = None
    counter1 = counter2 = 0

    for i in range(0,n):
        if counter1 == 0 and arr[i] != candidate2:
            candidate1 = arr[i]
            counter1 = 1
        elif counter2 == 0 and arr[i] != candidate1:
            candidate2 = arr[i]
            counter2 = 1
        elif arr[i] == candidate1:
            counter1 += 1
        elif arr[i] == candidate2:
            counter2 += 1
        else:
            counter1 -= 1
            counter2 -= 1

    counter1 = counter2 = 0
    for i in range(0, n):
        if arr[i] == candidate1:
            counter1 += 1
        if arr[i] == candidate2:
            counter2 += 1

    if counter1 > n//3:
        majority_elements.append(candidate1)
    if counter2 > n//3:
        majority_elements.append(candidate2)

    return majority_elements
```

In [74]:
```python
test_cases = [{'arr': [1,2,2,3,2]},
              {'arr': [11,33,33,11,33,11]},
              {'arr': [2,1,1,3,1,4,5,6]}]

for tc in test_cases:
    arr = tc['arr']
    print('arr:', arr, 'Majority element:', find_majority_element_2_OS(arr))
```

```
arr: [1, 2, 2, 3, 2] Majority element: [2]
arr: [11, 33, 33, 11, 33, 11] Majority element: [11, 33]
arr: [2, 1, 1, 3, 1, 4, 5, 6] Majority element: [1]
```

Problem Statement 23: Sort an array of 0s, 1s and 2s (Dutch National Flag Sorting Algorithm)

Given an array consisting of only 0s, 1s, and 2s. Write a program to in-place sort the array without using inbuilt sort functions. (Expected: Single pass-O(N) and constant space)

```
Input: nums = [2,0,2,1,1,0]
Output: [0,0,1,1,2,2]

Input: nums = [2,0,1]
Output: [0,1,2]

Input: nums = [0]
Output: [0]
```

In [75]:

```python
"""
Note:
    [1] if mid = 0, swap(low, mid); low ++; mid ++
    [2] if mid = 1, mid ++
    [3] if mid = 2, swap(high, mid); high--

Time Complexity  : O( n )
Space Complexity : O( 1 )
"""
def dnf_sort(arr):
    n = len(arr)
    low = 0
    mid = 0
    high = n-1

    while mid <= high:
        if arr[ mid ] == 0:
            arr[ low ], arr[ mid ] = arr[ mid ], arr[ low ]
            mid += 1
            low += 1
        elif arr[ mid ] == 1:
            mid += 1
        else:
            arr[ mid ], arr[ high ] = arr[ high ], arr[ mid ]
            high -= 1
```

In [76]:

```python
test_cases = [{'arr': [2,0,2,1,1,0]},
              {'arr': [2,0,1]},
              {'arr': [0]},
              {'arr': [1,1,2,0,0,1,2,2,1,0]}]

for tc in test_cases:
    arr = tc['arr']
    dnf_sort(arr)
    print('arr:', arr)
```

```
arr: [0, 0, 1, 1, 2, 2]
arr: [0, 1, 2]
arr: [0]
arr: [0, 0, 0, 1, 1, 1, 1, 2, 2, 2]
```

Problem Statement 24: Rearrange Array Elements by Sign

There's an array 'A' of size 'N' with an equal number of positive and negative elements. Without altering the relative order of positive and negative elements, you must return an array of alternately positive and negative values.

Note: Start the array with positive elements.

```
Example 1:
Input: arr = [1,2,-4,-5]
Output: [1, -4, 2, -5]

Example 2:
Input: arr = [1,2,-3,-1,-2,3]
Output: [1, -3, 2, -1, 3, -2]
```

In [77]:

```python
"""
Time Complexity  : O( n )
Space Complexity : O( 1 )
"""
def rearrange_by_sign(arr):
    n = len(arr)

    pos_idx, neg_idx = 0, 1
    result = [0] * n

    for i in range(0,n):
        if arr[i] >= 0:
            result[pos_idx] = arr[i]
            pos_idx += 2
        else:
            result[neg_idx] = arr[i]
            neg_idx += 2

    return result
```

In [78]:

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

for tc in test_cases:
    arr = tc['arr']
    print('arr:', arr, 'Re-arranged arr:', rearrange_by_sign(arr))
```

```
arr: [1, 2, -4, -5] Re-arranged arr: [1, -4, 2, -5]
arr: [1, 2, -3, -1, -2, 3] Re-arranged arr: [1, -3, 2, -1, 3, -2]
```

Problem Statement 25: Longest Consecutive Sequence in an Array

You are given an array of 'N' integers. You need to find the length of the longest sequence which contains the consecutive elements.

```
Example 1:
Input: [100, 200, 1, 3, 2, 4]
Output: 4
Explanation: The longest consecutive subsequence is 1, 2, 3, and 4.

Example 2:
Input: [3, 8, 5, 7, 6]
Output: 4
Explanation: The longest consecutive subsequence is 5, 6, 7, and 8.
```

In [79]:

```python
"""
Optimal Solution

Time Complexity  : O( n log n ) + O( n )
Space Complexity : O( 1 )
"""
def longest_successive_elements_OS1(arr):
    arr.sort()

    n = len(arr)
    last_smallest = float('-inf')
    counter = longest = 0

    for i in range(0, n):
        if (arr[i] - 1) == last_smallest:
            counter += 1
            last_smallest = arr[i]
        elif arr[i] != last_smallest:
            counter = 1
            last_smallest = arr[i]

        longest = max(counter, longest)

    return longest
```

In [80]:

```python
test_cases = [{'arr': [100,200,1,3,2,4]},
              {'arr': [3,8,5,7,6]},
              {'arr': [102,4,100,1,101,3,2,1,1]},
              {'arr': [100,102,100,101,101,4,3,2,3,2,1,1,1,2]}]

for tc in test_cases:
    arr = tc['arr']
    print('arr:', arr, 'Longest successive elements count:', longest_successive_elements_OS1(arr))
```

```
arr: [1, 2, 3, 4, 100, 200] Longest successive elements count: 4
arr: [3, 5, 6, 7, 8] Longest successive elements count: 4
arr: [1, 1, 1, 2, 3, 4, 100, 101, 102] Longest successive elements count: 4
arr: [1, 1, 1, 2, 2, 2, 3, 3, 4, 100, 100, 101, 101, 102] Longest successive elements count: 4
```

In [81]:

```python
"""
Optimal Solution

Time Complexity  : O( 3n )
Space Complexity : O( n )
"""
def longest_successive_elements_OS2(arr):
    n = len(arr)

    unique_arr = set()
    for num in arr:
        unique_arr.add(num)

    longest = 0
    for num in unique_arr:
        if (num - 1) not in unique_arr:
            tmp = num
            counter = 1

            while tmp+1 in unique_arr:
                counter += 1
                tmp += 1

            longest = max(counter, longest)

    return longest
```

In [82]:

```python
test_cases = [{'arr': [100,200,1,3,2,4]},
              {'arr': [3,8,5,7,6]},
              {'arr': [102,4,100,1,101,3,2,1,1]},
              {'arr': [100,102,100,101,101,4,3,2,3,2,1,1,1,2]}]

for tc in test_cases:
    arr = tc['arr']
    print('arr:', arr, 'Longest successive elements count:', longest_successive_elements_OS2(arr))
```

```
arr: [100, 200, 1, 3, 2, 4] Longest successive elements count: 4
arr: [3, 8, 5, 7, 6] Longest successive elements count: 4
arr: [102, 4, 100, 1, 101, 3, 2, 1, 1] Longest successive elements count: 4
arr: [100, 102, 100, 101, 101, 4, 3, 2, 3, 2, 1, 1, 1, 2] Longest successive elements count: 4
```

Problem Statement 26: Find next permutation

Given an array of integers, rearrange the numbers of the given array into the lexicographically next greater permutation of numbers. If such an arrangement is not possible, it must rearrange to the lowest possible order (i.e., sorted in ascending order).

```
Eample 1:
Input: arr = [1,3,2]
Output: arr = [2,1,3]
Explanation: All permutations of [1,2,3] are [[1,2,3], [1,3,2], [2,1,3], [2,3,1], [3,1,2],
[3,2,1]]. So, the next permutation just after [1,3,2] is [2,1,3].

Eample 2:
Input: arr = [3,2,1]
Output: arr = [1,2,3]
Explanation: As we see all permutations of [1,2,3], we find [3,2,1] at the last position. So,
we have to return the topmost permutation.
```

In [83]:

```python
"""
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 [84]:

```python
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 27: 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.Therfore the 2nd column and 2nd row wil 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 [85]:

```python
"""
Better solution

Time Complexity  : O( 2rc )
Space Complexity : O( r ) + O( c )
"""
def set_zeroes_BS(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] or 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 [86]:

```python
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_zeroes_BS(matrix)
    print('\n', '#'*100, sep='')
```

```


1  0  1

0  0  0

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


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 [87]:

```python
"""
Optimal solution

Time Complexity  : O( 2rc )
Space Complexity : O( 1 )
"""
def set_zeroes_OS(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 [88]:

```python
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, 8],
                          [ 0,10,11,12],
                          [13,14,15, 0]]}]

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

```


1  0  1

0  0  0

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


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  3  0

0  0  0  0

0  0  0  0

0  0  0  0
####################################################################################################
```

Problem Statement 28: Rotate Image by 90 degree

Given a matrix, your task is to rotate the matrix 90 degrees clockwise.

```
Example 1:
Input:  [[1,2,3],
         [4,5,6],
         [7,8,9]]
Output: [[7,4,1],
         [8,5,2],
         [9,6,3]]

Example 2:
Input:  [[ 5, 1, 9,11],
         [ 2, 4, 8,10],
         [13, 3, 6, 7],
         [15,14,12,16]]
Output: [[15,13, 2, 5],
         [14, 3, 4, 1],
         [12, 6, 8, 9],
         [16, 7,10,11]]
```

In [89]:

```python
"""
Time Complexity  : O( rc + rc/2 )
Space Complexity : O( 1 )
"""
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[r][c], matrix[c][r] = 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 [90]:

```python
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
####################################################################################################
```

Problem Statement 29: Spiral Traversal of Matrix

Given a Matrix, print the given matrix in spiral order.

```
Example 1:          L           R
Input: matrix = T [[01 02 03 04 05],
                   [14 15 16 17 06],
                   [13 20 19 18 07],
                 B  [12 11 10 09 08]]
Outhput: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```

In [91]:

```python
"""
Time Complexity  : O( rc )
Space Complexity : O( 1 )
"""
def print_spiral(matrix):
    rows, columns = len(matrix), len(matrix[0])

    top, bottom = 0, rows-1
    left, right = 0, columns-1

    result = []

    while top <= bottom and left <= right:
        for i in range(left, right+1):
            result.append( matrix[top][i] )
        top += 1

        for i in range(top, bottom+1):
            result.append( matrix[i][right] )
        right -= 1

        if top <= bottom:
            for i in range(right, left-1, -1):
                result.append( matrix[bottom][i] )
            bottom -= 1

        if left <= right:
            for i in range(bottom, top-1, -1):
                result.append( matrix[i][left] )
            left += 1

    return result
```

In [92]:

```python
test_cases = [{'matrix': [[ 1,  2,  3,  4, 5],
                          [14, 15, 16, 17, 6],
                          [13, 20, 19, 18, 7],
                          [12, 11, 10,  9, 8]]}]

for tc in test_cases:
    matrix = tc['matrix']
    print(print_spiral(matrix))
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
```

Problem Statement 30: Pascal's Triangle

This problem has 3 variations. They are stated below:
Variation 1: Given row number r and column number c. Print the element at position (r, c) in Pascal's triangle.
Variation 2: Given the row number n. Print the n-th row of Pascal's triangle.
Variation 3: Given the number of rows n. Print the first n rows of Pascal's triangle.

```
Example 1:
Input:  r = 5, c = 3
Output: (for variation 1) -> 6
        (for variation 2) -> 1 4 6 4 1
        (for variation 3) -> 1
                             1 1
                             1 2 1
                             1 3 3 1
                             1 4 6 4 1

Example 2:
Input: r = 1, c = 1
Output: (for variation 1) -> 1
        (for variation 2) -> 1
        (for variation 3) -> 1
```

In [93]:

```python
# note:
# nCr =      n!       = n x (n-1) x (n-2) x ... x (n-r)! = n x (n-1) x (n-2) x ...
#       -------------   --------------------------------   -----------------------
#        r! x (n-r)!              r! x (n-r)!                          r!

"""
Time Complexity  : O( r )
Space Complexity : O( 1 )
"""
def nCr(n, r):
    numerator = denominator = 1

    for i in range(1, r+1):
        numerator *= n
        denominator *= i
        n -= 1

    return int(numerator / denominator)
```

In [94]:

```python
"""
Variation 1 (Optimal solution)

Time Complexity  : O( col_num )
Space Complexity : O( 1 )
"""
def pascals_triangle_v1(row_num, col_num):
    return nCr( row_num-1, col_num-1 )
```

In [95]:

```python
test_cases = [{'row_num': 1, 'col_num': 1},
              {'row_num': 2, 'col_num': 2},
              {'row_num': 5, 'col_num': 3},
              {'row_num': 6, 'col_num': 4},
              {'row_num': 12, 'col_num': 6}]

for tc in test_cases:
    row_num, col_num = tc['row_num'], tc['col_num']
    print(f"row_num: {row_num}\tcol_num: {row_num}\tPascal's Triangle Element: {pascals_triangle_v1(row_num, co
```

```
row_num: 1      col_num: 1      Pascal's Triangle Element: 1
row_num: 2      col_num: 2      Pascal's Triangle Element: 1
row_num: 5      col_num: 5      Pascal's Triangle Element: 6
row_num: 6      col_num: 6      Pascal's Triangle Element: 10
row_num: 12     col_num: 12     Pascal's Triangle Element: 462
```

In [96]:

```python
"""
Variation 2 (Brute force solution)

Time Complexity  : O( r(r+1)/2 ) ~ O ( r^2 )
Space Complexity : O( 1 )
"""
def pascals_triangle_v2_BFS(row_num):
    result = []
    for col_num in range(1, row_num+1):
        result.append( nCr( row_num-1, col_num-1 ) )
    return result
```

In [97]:

```python
test_cases = [{'row_num': 1},
              {'row_num': 2},
              {'row_num': 5},
              {'row_num': 6},
              {'row_num': 12}]

for tc in test_cases:
    row_num = tc['row_num']
    print(f"row_num: {row_num}\tPascal's Triangle Row: {pascals_triangle_v2_BFS(row_num)}")
```

```
row_num: 1      Pascal's Triangle Row: [1]
row_num: 2      Pascal's Triangle Row: [1, 1]
row_num: 5      Pascal's Triangle Row: [1, 4, 6, 4, 1]
row_num: 6      Pascal's Triangle Row: [1, 5, 10, 10, 5, 1]
row_num: 12     Pascal's Triangle Row: [1, 11, 55, 165, 330, 462, 462, 330, 165, 55, 11, 1]
```

In [98]:

```python
"""
Variation 2 (Optimal solution)

Time Complexity  : O( r )
Space Complexity : O( 1 )
"""
def pascals_triangle_v2_OS(row_num):
    result = []

    prev_row_num = row_num - 1
    numerator = denominator = 1

    for col_num in range(0, row_num):
        if col_num == 0 or col_num == row_num-1:
            result.append( 1 )
        else:
            numerator *= prev_row_num
            denominator *= col_num
            prev_row_num -= 1

            result.append( int(numerator / denominator) )

    return result
```

In [99]:

```python
test_cases = [{'row_num': 1},
              {'row_num': 2},
              {'row_num': 5},
              {'row_num': 6},
              {'row_num': 12}]

for tc in test_cases:
    row_num = tc['row_num']
    print(f"row_num: {row_num}\tPascal's Triangle Row: {pascals_triangle_v2_OS(row_num)}")
```

```
row_num: 1      Pascal's Triangle Row: [1]
row_num: 2      Pascal's Triangle Row: [1, 1]
row_num: 5      Pascal's Triangle Row: [1, 4, 6, 4, 1]
row_num: 6      Pascal's Triangle Row: [1, 5, 10, 10, 5, 1]
row_num: 12     Pascal's Triangle Row: [1, 11, 55, 165, 330, 462, 462, 330, 165, 55, 11, 1]
```

In [100…

```python
"""
Variation 3 (Brute force solution)

Time Complexity  : O( ∑[1,r](r(r+1)/2) ) = O( r(r+1)(2r+1)/6 ) ~ O ( r^3 )
Space Complexity : O( 1 )
"""
def pascals_triangle_v3_BFS(row_num):
    result = []

    for row in range(1, row_num+1):
        result_row = []

        for col in range(1, row+1):
            result_row.append( nCr( row-1, col-1 ) )

        result.append( result_row )

    return result
```

In [101…

```python
test_cases = [{'row_num': 1},
              {'row_num': 2},
              {'row_num': 5},
              {'row_num': 6},
              {'row_num': 12}]

for tc in test_cases:
    row_num = tc['row_num']
    pt = pascals_triangle_v3_BFS(row_num)
    for row in pt:
        print(row)
    print('#'*100)
```

```
[1]
####################################################################################################
[1]
[1, 1]
####################################################################################################
[1]
[1, 1]
[1, 2, 1]
[1, 3, 3, 1]
[1, 4, 6, 4, 1]
####################################################################################################
[1]
[1, 1]
[1, 2, 1]
[1, 3, 3, 1]
[1, 4, 6, 4, 1]
[1, 5, 10, 10, 5, 1]
####################################################################################################
[1]
[1, 1]
[1, 2, 1]
[1, 3, 3, 1]
[1, 4, 6, 4, 1]
[1, 5, 10, 10, 5, 1]
[1, 6, 15, 20, 15, 6, 1]
[1, 7, 21, 35, 35, 21, 7, 1]
[1, 8, 28, 56, 70, 56, 28, 8, 1]
[1, 9, 36, 84, 126, 126, 84, 36, 9, 1]
[1, 10, 45, 120, 210, 252, 210, 120, 45, 10, 1]
[1, 11, 55, 165, 330, 462, 462, 330, 165, 55, 11, 1]
####################################################################################################
```

In [102…

```python
"""
Variation 3 (Optimal solution)

Time Complexity  : O ( r^2 )
Space Complexity : O( 1 )
"""
def pascals_triangle_v3_OS(row_num):
    result = []

    for row in range(1, row_num+1):
        result.append( pascals_triangle_v2_OS(row) )

    return result
```

In [103…

```python
test_cases = [{'row_num': 1},
              {'row_num': 2},
              {'row_num': 5},
              {'row_num': 6},
              {'row_num': 12}]

for tc in test_cases:
    row_num = tc['row_num']
    pt = pascals_triangle_v3_OS(row_num)
    for row in pt:
        print(row)
    print('#'*100)
```

```
[1]
####################################################################################################
[1]
[1, 1]
####################################################################################################
[1]
[1, 1]
[1, 2, 1]
[1, 3, 3, 1]
[1, 4, 6, 4, 1]
####################################################################################################
[1]
[1, 1]
[1, 2, 1]
[1, 3, 3, 1]
[1, 4, 6, 4, 1]
[1, 5, 10, 10, 5, 1]
####################################################################################################
[1]
[1, 1]
[1, 2, 1]
[1, 3, 3, 1]
[1, 4, 6, 4, 1]
[1, 5, 10, 10, 5, 1]
[1, 6, 15, 20, 15, 6, 1]
[1, 7, 21, 35, 35, 21, 7, 1]
[1, 8, 28, 56, 70, 56, 28, 8, 1]
[1, 9, 36, 84, 126, 126, 84, 36, 9, 1]
[1, 10, 45, 120, 210, 252, 210, 120, 45, 10, 1]
[1, 11, 55, 165, 330, 462, 462, 330, 165, 55, 11, 1]
####################################################################################################
```

In [ ]: