2. Iterate through the array from left to right.
3. For each element, update the currentSum by adding the current element or starting a new subarray if the current element is larger than the sum so far.
4. Update the maxSum if the currentSum becomes larger than the maxSum.
5. Finally, return the maxSum.

Day 5 / 100 :

Topic - Array

1 Problem statement: Find N Unique Integers Sum up to Zero (easy)

Given an integer n, return any array containing n unique integers such that they add up to 0.

Example 1:

Input: n = 5
Output: [-7,-1,1,3,4]
Explanation: These arrays also are accepted [-5,-1,1,2,3] , [-3,-1,2,-2,4].

Solutions :

Approach 1 -

Explanation:

1. The function sumZero takes an integer n as input and returns a vector of integers.
2. We create an empty vector called result to store the resulting array.
3. If n is odd, we include 0 in the result array. This ensures that the sum of the elements in the array is 0.
4. Next, we generate pairs of positive and negative numbers. We iterate from 1 to n/2 (inclusive) and for each number i, we add i and -i to the result array. This guarantees that the sum of each pair is 0.
5. Finally, we return the result array.
6. In the main function, we call the sumZero function with n = 5 and store the resulting array in the result variable.

7. We then print the elements of the result array using a for-each loop.

Intuition:
To create an array of n unique integers that sum up to 0, we can take advantage of the property that the sum of positive and negative integers cancels out. By creating pairs of positive and negative numbers (excluding 0 for even n), we can ensure that the sum of the array is 0.

```
class Solution {
public:
   vector<int> sumZero(int n) {
           vector<int> res;
        if(n == 0) return res;
        if(n%2 != 0) res.push_back(0); //if odd then to push 0
        for(int i=1;i<=floor(n/2);i++){
            res.push_back(i);
            res.push_back(-i);
        }
       return res;
}
};
```

2 Problem statement: [Rectangle Overlap](#) (easy)

An axis-aligned rectangle is represented as a list [x1, y1, x2, y2], where (x1, y1) is the coordinate of its bottom-left corner, and (x2, y2) is the coordinate of its top-right corner. Its top and bottom edges are parallel to the X-axis, and its left and right edges are parallel to the Y-axis.

Two rectangles overlap if the area of their intersection is positive. To be clear, two rectangles that only touch at the corner or edges do not overlap.

Given two axis-aligned rectangles rec1 and rec2, return true if they overlap, otherwise return false.

Example 1:

Input: rec1 = [0,0,2,2], rec2 = [1,1,3,3]
Output: true
Example 2:

Input: rec1 = [0,0,1,1], rec2 = [1,0,2,1]
Output: false

Solutions :

Approach 1

```
class Solution {
public:
bool isRectangleOverlap(vector<int>& rec1, vector<int>& rec2) {
    // Check if rec1 is to the left of rec2
    if (rec1[2] <= rec2[0])
        return false;

    // Check if rec1 is to the right of rec2
    if (rec1[0] >= rec2[2])
        return false;

    // Check if rec1 is below rec2
    if (rec1[3] <= rec2[1])
        return false;

    // Check if rec1 is above rec2
    if (rec1[1] >= rec2[3])
        return false;

    // If none of the above conditions are true, the rectangles
overlap
    return true;
}
};
```

Explanation:

The function takes two vectors, rec1 and rec2, representing the rectangles. The function returns a boolean value indicating whether the rectangles overlap or not.

To determine if the rectangles overlap, we need to check four conditions:

1. Check if rec1 is to the left of rec2: If the right edge of rec1 (given by rec1[2]) is less than or equal to the left edge of rec2 (given by rec2[0]), then rec1 is completely to the left of rec2 and they don't overlap.
2.
3. Check if rec1 is to the right of rec2: If the left edge of rec1 (given by rec1[0]) is greater than or equal to the right edge of rec2 (given by rec2[2]), then rec1 is completely to the right of rec2 and they don't overlap.
4.
5. Check if rec1 is below rec2: If the top edge of rec1 (given by rec1[3]) is less than or equal to the bottom edge of rec2 (given by rec2[1]), then rec1 is completely below rec2 and they don't overlap.
6.
7. Check if rec1 is above rec2: If the bottom edge of rec1 (given by rec1[1]) is greater than or equal to the top edge of rec2 (given by rec2[3]), then rec1 is completely above rec2 and they don't overlap.

If none of the above conditions are true, then the rectangles overlap.

This solution works based on the fact that for two rectangles to overlap, they must overlap on both the X-axis and the Y-axis. Checking these four conditions covers all possible scenarios where the rectangles don't overlap.