**Q1. Find Pivot Index Given an array of integers nums, calculate the pivot index of this array.**

class Solution {

    public int pivotIndex(int[] nums) {

        int s = 0 , l = 0 , r = 0;

        for(int i : nums) s += i;

        for(int i = 0; i<nums.length; i++){

            l += nums[i];

            r = s - l + nums[i];

            if(l == r)

            return i;

        }

       return -1;

    }

}

**Q2. Minimum Value to Get Positive Step by Step Sum Given an array of integers nums, you start with an initial positive value startValue. In each iteration, you calculate the step by step sum of startValue plus elements in nums (from left to right). Return the minimum positive value of startValue such that the step by step sum is never less than 1.**

class Solution {

    public int minStartValue(int[] nums) {

        int sum = 0, ans = 1;

        for(int num:nums)

        {

            sum+=num;

            if(sum<0)

                ans = Math.max(ans,Math.abs(sum)+1);

        }

        return ans;

    }

}

**Q3. Running Sum of 1d Array Given an array nums. We define a running sum of an array as runningSum[i] = sum(nums[0]…nums[i]). Return the running sum of nums.**

class Solution {

    public int[] runningSum(int[] nums) {

        for(int i=1;i<nums.length;i++)

        {

            nums[i]=nums[i-1]+nums[i];

        }

        return nums;

    }

}

**Q4. Find the Highest Altitude There is a biker going on a road trip. The road trip consists of n + 1 points at different altitudes. The biker starts his trip on point 0 with altitude equal 0. You are given an integer array gain of length n where gain[i] is the net gain in altitude between points i and i + 1 for all (0 <= i < n). Return the highest altitude of a point.**

class Solution {

    public int largestAltitude(int[] gain) {

        int h = 0,c=0;

        for (int i = 0; i < gain.length; i++)

        {

            c += gain[i];

            h = Math.max(h, c);

        }

        return h;

    }

}

**Q5. Minimum Size Subarray Sum Given an array of positive integers nums and a positive integer target, return the minimal length of a subarray whose sum is greater than or equal to target. If there is no such subarray, return 0 instead.**

class Solution {

    public int minSubArrayLen(int target, int[] nums) {

        int i=0;

        int sum=0;

        int len=Integer.MAX\_VALUE;

        for(int j=0;j<nums.length;j++) {

            sum+=nums[j];

            while(sum>=target) {

                len = Math.min(len,j-i+1);

                sum-=nums[i];

                i++;

            }

        }

        if(len == Integer.MAX\_VALUE) return 0;

        return len;

    }

}

**Q6. Subarray Sum Equals K Given an array of integers nums and an integer k, return the total number of subarrays whose sum equals to k. A subarray is a contiguous non-empty sequence of elements within an array.**

class Solution {

    public int subarraySum(int[] nums, int k) {

        int sum=0,res=0;

        HashMap<Integer,Integer> map=new HashMap<>();

        map.put(0,1);

        for(int i=0;i<nums.length;i++)

        {

            sum+=nums[i];

            if(map.containsKey(sum-k))

            res+=map.get(sum-k);

            map.put(sum,map.getOrDefault(sum,0)+1);

        }

        return res;

    }

}

**Q7. Apply Operations to Make All Array Elements Equal to Zero You are given a 0-indexed integer array nums and a positive integer k. You can apply the following operation on the array any number of times: Choose any subarray of size k from the array and decrease all its elements by 1. Return true if you can make all the array elements equal to 0, or false otherwise.**

class Solution {

    public boolean checkArray(int[] nums, int k) {

        int increment = 0;

        for(int i = 0; i < nums.length; i++) {

            if(increment > nums[i])

                return false;

            nums[i] -= increment;

            increment += nums[i];

            if(i - k + 1 >= 0) {

                increment -= nums[i - k + 1];

            }

        }

        return increment == 0;

    }

}

**Q8. Number of Submatrices That Sum to Target Given a matrix and a target, return the number of nonempty submatrices that sum to target. A submatrix x1, y1, x2, y2 is the set of all cells matrix[x][y] with x1 <= x <= x2 and y1 <= y <= y2. Two submatrices (x1, y1, x2, y2) and (x1' , y1' , x2' , y2') are different if they have some coordinate that is different: for example, if x1 != x1'.**

class Solution {

    public int numSubmatrixSumTarget(int[][] A, int target) {

        int res = 0, m = A.length, n = A[0].length;

        for (int i = 0; i < m; i++)

            for (int j = 1; j < n; j++)

                A[i][j] += A[i][j - 1];

        Map<Integer, Integer> counter = new HashMap<>();

        for (int i = 0; i < n; i++) {

            for (int j = i; j < n; j++) {

                counter.clear();

                counter.put(0, 1);

                int cur = 0;

                for (int k = 0; k < m; k++) {

                    cur += A[k][j] - (i > 0 ? A[k][i - 1] : 0);

                    res += counter.getOrDefault(cur - target, 0);

                    counter.put(cur, counter.getOrDefault(cur, 0) + 1);

                }

            }

        }

        return res;

    }

}