Q1. 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;
    }
}
```

Q2. Maximum Subarray Given an integer array nums, find the subarray with the largest sum, and return its sum.

```
class Solution {
    public int maxSubArray(int[] nums) {
        int n=nums.length;
        int max = Integer.MIN_VALUE;
        int sum = 0;
        for (int i = 0; i < n; i++) {
            sum += nums[i];
            if (sum > max) {
                max = sum;
            }
            if (sum < 0) {
                sum = 0;
            }
        }
        return max;
    }
}
```

Q3. Find Subsequence of Length K With the Largest Sum You are given an integer array nums and an integer k. You want to find a subsequence of nums of length k that has the largest sum. Return any such subsequence as an integer array of length k. A subsequence is an array that can be derived from another array by deleting some or no elements without changing the order of the remaining elements.

```
class Solution {
    public int[] maxSubsequence(int[] nums, int k) {
        PriorityQueue<int[]> pq = new PriorityQueue<>((a,b) -> b[0] - a[0]);
        for(int i=0; i<nums.length; i++)
            pq.offer(new int[]{nums[i], i});
        List<int[]> l = new ArrayList<>();
        while(k-- != 0)
            l.add(pq.poll());
        Collections.sort(l, (a,b) -> a[1] - b[1]);
        int[] res = new int[l.size()];
        int index = 0;
        for(int[] i: l)
            res[index++] = i[0];
        return res;
    }
}
```

Q4. Longest Increasing Subsequence Given an integer array nums, return the length of the longest strictly increasing subsequence.

```
class Solution {
    public int lengthOfLIS(int[] nums) {
        int[] a=new int[nums.length];
        Arrays.fill(a,1);
        for(int i=0;i<nums.length;i++)
            for(int j=0;j<i;j++)
                if(nums[i]>nums[j])
                    a[i]=Math.max(a[i],a[j]+1);
        int ml=Arrays.stream(a).max().orElse(0);
        return ml;
    }
}
```

Q5. 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;
    }
}
```

Q6. Contiguous Array Given a binary array nums, return the maximum length of a contiguous subarray with an equal number of 0 and 1.

Q7. Maximum Sum of Two Non-Overlapping Subarrays Given an integer array nums and two integers firstLen and secondLen, return the maximum sum of elements in two non-overlapping subarrays with lengths firstLen and secondLen. The array with length firstLen could occur before or after the array with length secondLen, but they have to be nonoverlapping. A subarray is a contiguous part of an array.

```
class Solution {
    public int maxSumTwoNoOverlap(int[] A, int L, int M) {
        int sums[] = new int[A.length+1];

        for(int i=1;i<=A.length;i++)
            sums[i] = A[i-1]+sums[i-1];

        int maxLval = 0;
        int ans=0;
        for(int i=L;i<=A.length-M;i++)
        {
            maxLval = Math.max(maxLval,sums[i]-sums[i-L]);
            ans = Math.max(ans,sums[i+M]-sums[i]+maxLval);
        }
        int maxRval = 0 ;
        for(int i=M;i<=A.length-L;i++)
        {
            maxRval = Math.max(maxRval,sums[i]-sums[i-M]);
            ans = Math.max(ans,sums[i+L]-sums[i]+maxRval);
        }
        return ans;
    }
}
```

Q8. Sum of Subsequence Widths The width of a sequence is the difference between the maximum and minimum elements in the sequence. Given an array of integers nums, return the sum of the widths of all the non-empty subsequences of nums. Since the answer may be very large, return it modulo 109 + 7. A subsequence is a sequence that can be derived from an array by deleting some or no elements without changing the order of the remaining elements. For example, [3,6,2,7] is a subsequence of the array [0,3,1,6,2,2,7].

```
class Solution {
    public int sumSubseqWidths(int[] nums) {
        Arrays.sort(nums);
        long c = 1;
        long result = 0;
        long mod = 1000000007L;
        for (int i = 0, j = nums.length - 1; i < nums.length; i++, j--) {
            result = (result + (nums[i] * c) - (nums[j] * c)) % mod;
            c = (c * 2) % mod;
        }
        return (int) ((result + mod) % mod);
    }
}
```