PROBLEM SOLVING

SmartCliff
Career Mobility Solutions

SDE Readiness Training

Exercise No. : **7**

Topics Covered : **Array, Priority Queue, List, Hashmap, Treemap, Heap, LinkedHashMap**

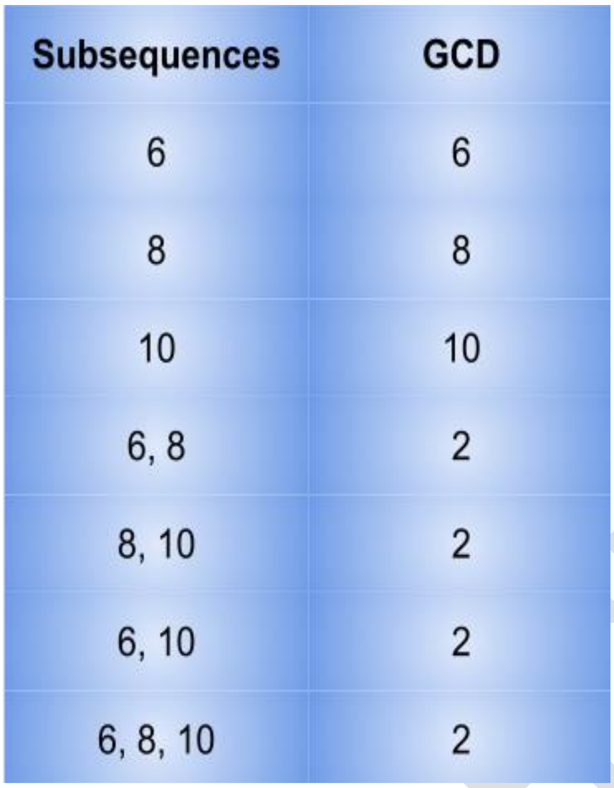
Date : **22-05-2024**

Solve the following problems

Q No.	Question Detail	Level
1	**Different Sub sequences GCD** **Problem statement :** You are given an array "ARR" of positive integers. Your task is to find the number of different GCD's in all the non-empty subsequences of the given array. Note: The GCD of a list of numbers is defined as the greatest number that perfectly divides all the members of the given list. For example, the GCD of 6, 8, and 10 is 2. A subsequence of the array is a list that can be made by using some elements of the given array. All the elements can also be part of a subsequence. For example: the array [1, 2] has subsequences: [1], [2] and [1, 2]. Similarly for the array [1, 2, 3], one of the subsequences is [1, 3]. **Sample Input 1:** 2 3 6 8 10 3 1 2 1 **Sample Output 1:** 4 2 **Explanation For Sample Input 1:** In the first test case, The subsequences and their GCD's are:	Hard

Little practice is worth more than a ton of theory

The different GCD’s from the above table are 6, 8, 10, 2. So, the answer is 4.

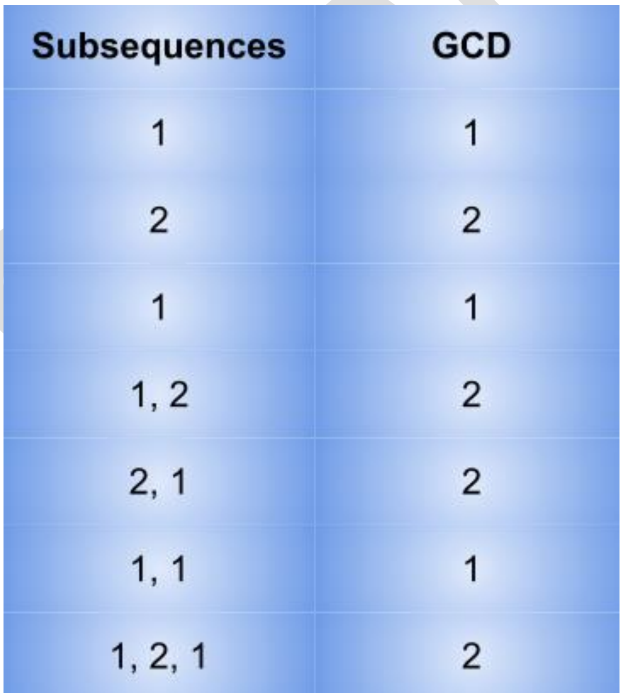


Subsequences	GCD
6	6
8	8
10	10
6, 8	2
8, 10	2
6, 10	2
6, 8, 10	2

In the second test case,

The subsequences and their GCD’s are:

The different GCD’s from the above table are 1, 2. So, the answer is 2.



Subsequences	GCD
1	1
2	2
1	1
1, 2	2
2, 1	2
1, 1	1
1, 2, 1	2

Sample Input 2:

2

	4 5 10 15 20 3 2 5 10 **Sample Output 2:** 4 4 **Explanation For Sample Input 1:** In the first test case, The different GCD's are 5, 10, 15, 20. So, the answer is 4. In the second test case, The different GCD's are 1, 2, 5, 10. So, the answer is 4. **Constraints:** 1 <= T <= 10 1 <= N <= 10^4 1 <= ARR[i] <= 10^5 Where 'T' is the number of test cases, 'N', denotes the size of the array 'ARR', and 'ARR[i]' denotes the elements of the array.	
2	**24 Game** **Problem statement :** Henry is feeling lonely, so he started playing online games. While searching for fun, he found an exciting game. In this game, Ninja has to choose four cards at random. On each card, there is a number between 1 to 9, both inclusive. For Henry to win, he has to make the number 24 using the number on cards and the following operator *, /, +, -, (,). Help Henry to find whether he will win the game or not, on the basis of his selection. If Henry can win the game, print true otherwise, print false. **Example:-** If the cards Ninja chooses are 4, 1, 8, 7. Then Ninja can make 24 by (8 - 4) * (7 - 1). Hence Ninja can win, so you have to return true. **Note:-** The division operator '/' represents actual division, not integer division. For example, 4 / (1 - ⅔) = 12.	Hard

	Sample Input 1:- 2 4 1 8 7 1 2 1 2 **Sample Output 1:-** True False **Explanation Of Sample Input 1:-** Test case 1:- Here, we can make 24 by (8 - 4) * (7 - 1). Hence we will return true. Test case 2:- Here, there is no way to make 24 using these cards, so ninja can't win, hence return false. **Sample Input 2 :-** 2 6 7 8 9 1 2 3 4 **Sample Output 2:-** True True **Explanation Of Sample Input 2:-** Test case 1:- Here, we can make 24 by (8 - 4) * (7 - 1). Hence we will return true. Test case 2:- Here, there is no way to make 24 using these cards, so ninja can't win, hence return false. **Constraints:-** 1 <= T <= 3000 1 <= NUMS[i] <= 9 where 0 <= i <= 4	
3	**Running Median** **Problem statement :** You are given a stream of 'N' integers. For every 'i-th' integer added to the running list of integers, print the resulting median. Print only the integer part of the median. **Sample Input 1 :** 6 6 2 1 3 7 5 **Sample Output 1 :**	Hard

	6 4 2 2 3 4 **Explanation of Sample Output 1 :** S = {6}, median = 6 S = {6, 2} -> {2, 6}, median = 4 S = {6, 2, 1} -> {1, 2, 6}, median = 2 S = {6, 2, 1, 3} -> {1, 2, 3, 6}, median = 2 S = {6, 2, 1, 3, 7} -> {1, 2, 3, 6, 7}, median = 3 S = {6, 2, 1, 3, 7, 5} -> {1, 2, 3, 5, 6, 7}, median = 4 **Sample Input 2 :** 5 5 4 3 2 1 **Sample Output 2 :** 5 4 4 3 3 **Constraints:** 0<=N<=10^5 1 <= ARR[i] <= 10^5 Where 'T' is the number of test cases, 'N', denotes the size of the array 'ARR', and 'ARR[i]' denotes the elements of the array.	
4	**Settle Debt** **Problem statement** A group of friends went on a trip and sometimes lent each other money. Each transaction among them is represented by the tuple (X, Y, Z) which means person 'X' gave person 'Y' $Z. Given a list of 'N' transactions between a group of friends, return the minimum number of transactions required to settle the debt. Example: Alice paid for Bill's lunch for $10. Then later Chris gave Alice $5 for a taxi ride. Assuming Alice, Bill, and Chris are person 0, 1, and 2 respectively (0, 1, 2 are the person's ID), the transaction can be represented as [[0, 1, 10], [2, 0, 5]].So here the minimum number of transactions to settle the debt is 2. **Sample Input 1:** 1 2 0 1 10 2 0 5	Hard

Sample Output 1:

2

Explanation 1:

For the first test case,

Person-0 gave person-1 $10.

Person-2 gave person-0 $5.

Therefore, Two transactions are needed. One way to settle the debt is person-1 pays person-0 and person-2 $5 each.

Sample Input 2:

```
1
4
0 1 10
1 0 1
1 2 5
2 0 5
```

Sample Output 2:

```
1
```

Explanation 2:

For the first test case,

Person-0 gave person-1 $10.

Person-1 gave person-0 $1.

Person-1 gave person-2 $5.

Person-2 gave person-0 $5.

Therefore, only 1 transaction is needed. Person-1 only needs to give person-0 $4, and all debt is settled.

Constraints:

1 <= T <= 10

1 <= N <= 9

Where 'T' represents the number of test cases, and 'N' represents the number of transactions among the friends.

5 **Meeting Rooms III** Hard

Problem statement : You are given an integer n. There are n rooms numbered from 0 to n - 1. You are given a 2D integer array meetings where meetings[i] =

[$start_i$, end_i] means that a meeting will be held during the **half-closed** time interval [$start_i$, end_i). All the values of $start_i$ are **unique**.

Meetings are allocated to rooms in the following manner:

1. Each meeting will take place in the unused room with the **lowest** number.
2. If there are no available rooms, the meeting will be delayed until a room becomes free. The delayed meeting should have the **same** duration as the original meeting.
3. When a room becomes unused, meetings that have an earlier original **start** time should be given the room.

Return *the **number** of the room that held the most meetings.* If there are multiple rooms, return *the room with the **lowest** number.*

A **half-closed interval** [a, b) is the interval between a and b **including** a and **not including** b.

Example 1:

Input: n = 2, meetings = [[0,10],[1,5],[2,7],[3,4]]

Output: 0

Explanation:

- At time 0, both rooms are not being used. The first meeting starts in room 0.
- At time 1, only room 1 is not being used. The second meeting starts in room 1.
- At time 2, both rooms are being used. The third meeting is delayed.
- At time 3, both rooms are being used. The fourth meeting is delayed.
- At time 5, the meeting in room 1 finishes. The third meeting starts in room 1 for the time period [5,10).
- At time 10, the meetings in both rooms finish. The fourth meeting starts in room 0 for the time period [10,11).

Both rooms 0 and 1 held 2 meetings, so we return 0.

Example 2:

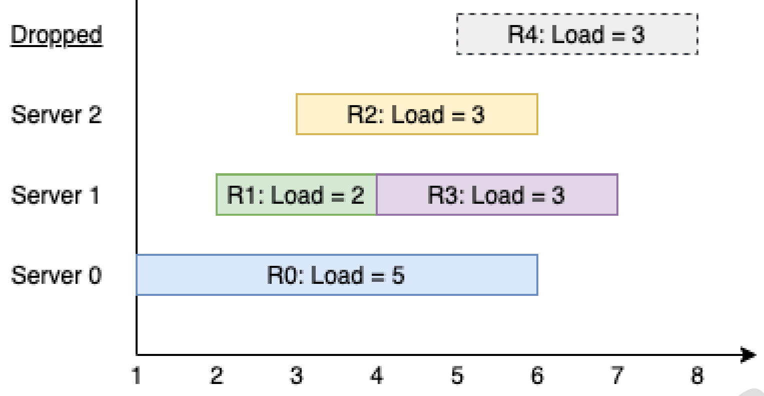
Input: n = 3, meetings = [[1,20],[2,10],[3,5],[4,9],[6,8]]

Output: 1

Explanation:

- At time 1, all three rooms are not being used. The first meeting starts in room 0.
- At time 2, rooms 1 and 2 are not being used. The second meeting starts in room 1.
- At time 3, only room 2 is not being used. The third meeting starts in room 2.
- At time 4, all three rooms are being used. The fourth meeting is delayed.

	- At time 5, the meeting in room 2 finishes. The fourth meeting starts in room 2 for the time period [5,10). - At time 6, all three rooms are being used. The fifth meeting is delayed. - At time 10, the meetings in rooms 1 and 2 finish. The fifth meeting starts in room 1 for the time period [10,12). Room 0 held 1 meeting while rooms 1 and 2 each held 2 meetings, so we return 1. **Constraints:** • 1 <= n <= 100 • 1 <= meetings.length <= 10^5 • meetings[i].length == 2 • 0 <= $start_i$ < end_i <= 5 * 10^5 • All the values of $start_i$ are **unique**.	
6	**Find Servers That Handled Most Number of Requests** **Problem statement :** You have k servers numbered from 0 to k-1 that are being used to handle multiple requests simultaneously. Each server has infinite computational capacity but **cannot handle more than one request at a time**. The requests are assigned to servers according to a specific algorithm: • The ith (0-indexed) request arrives. • If all servers are busy, the request is dropped (not handled at all). • If the (i % k)th server is available, assign the request to that server. • Otherwise, assign the request to the next available server (wrapping around the list of servers and starting from 0 if necessary). For example, if the ith server is busy, try to assign the request to the (i+1)th server, then the (i+2)th server, and so on. You are given a **strictly increasing** array arrival of positive integers, where arrival[i] represents the arrival time of the ith request, and another array load, where load[i] represents the load of the ith request (the time it takes to complete). Your goal is to find the **busiest server(s)**. A server is considered **busiest** if it handled the most number of requests successfully among all the servers. Return *a list containing the IDs (0-indexed) of the* ***busiest server(s)***. You may return the IDs in any order. **Example 1:**	Hard



Input: k = 3, arrival = [1,2,3,4,5], load = [5,2,3,3,3]

Output: [1]

Explanation:

All of the servers start out available.

The first 3 requests are handled by the first 3 servers in order.

Request 3 comes in. Server 0 is busy, so it's assigned to the next available server, which is 1.

Request 4 comes in. It cannot be handled since all servers are busy, so it is dropped.

Servers 0 and 2 handled one request each, while server 1 handled two requests.

Hence server 1 is the busiest server.

Example 2:

Input: k = 3, arrival = [1,2,3,4], load = [1,2,1,2]

Output: [0]

Explanation:

The first 3 requests are handled by first 3 servers.

Request 3 comes in. It is handled by server 0 since the server is available.

Server 0 handled two requests, while servers 1 and 2 handled one request each.

Hence server 0 is the busiest server.

Example 3:

Input: k = 3, arrival = [1,2,3], load = [10,12,11]

Output: [0,1,2]

Explanation: Each server handles a single request, so they are all considered the busiest.

Constraints:

- $1 <= k <= 10^5$

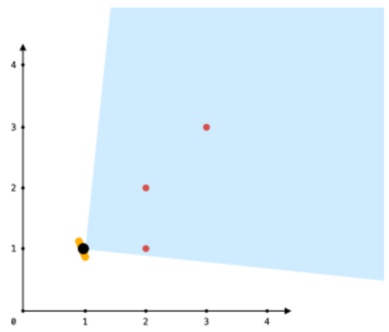
	• 1 <= arrival.length, load.length <= 10^5 • arrival.length == load.length • 1 <= arrival[i], load[i] <= 10^9 • arrival is **strictly increasing**.	
7	**Minimum Number of Refueling Stops** **Problem statement :** A car travels from a starting position to a destination which is target miles east of the starting position. There are gas stations along the way. The gas stations are represented as an array stations where stations[i] = [positioni, fueli] indicates that the ith gas station is positioni miles east of the starting position and has fueli liters of gas. The car starts with an infinite tank of gas, which initially has startFuel liters of fuel in it. It uses one liter of gas per one mile that it drives. When the car reaches a gas station, it may stop and refuel, transferring all the gas from the station into the car. Return the minimum number of refueling stops the car must make in order to reach its destination. If it cannot reach the destination, return -1. Note that if the car reaches a gas station with 0 fuel left, the car can still refuel there. If the car reaches the destination with 0 fuel left, it is still considered to have arrived. **Example 1:** Input: target = 1, startFuel = 1, stations = [] Output: 0 Explanation: We can reach the target without refueling. **Example 2:** Input: target = 100, startFuel = 1, stations = [[10,100]] Output: -1 Explanation: We can not reach the target (or even the first gas station). Example 3:	Hard

	Input: target = 100, startFuel = 10, stations = [[10,60],[20,30],[30,30],[60,40]] Output: 2 Explanation: We start with 10 liters of fuel. We drive to position 10, expending 10 liters of fuel. We refuel from 0 liters to 60 liters of gas. Then, we drive from position 10 to position 60 (expending 50 liters of fuel), and refuel from 10 liters to 50 liters of gas. We then drive to and reach the target. We made 2 refueling stops along the way, so we return 2. **Constraints:** 1 <= target, startFuel <= 10^9 0 <= stations.length <= 500 1 <= position_i < position_{i+1} < target 1 <= fueli < 10^9	
8	**Maximum Number of Visible Points** **Problem statement :** You are given an array points, an integer angle, and your location, where location = [posx, posy] and points[i] = [xi, yi] both denote integral coordinates on the X-Y plane. Initially, you are facing directly east from your position. You cannot move from your position, but you can rotate. In other words, posx and posy cannot be changed. Your field of view in degrees is represented by angle, determining how wide you can see from any given view direction. Let d be the amount in degrees that you rotate counterclockwise. Then, your field of view is the inclusive range of angles [d - angle/2, d + angle/2]. You can see some set of points if, for each point, the angle formed by the point, your position, and the immediate east direction from your position is in your field of view.	Hard

There can be multiple points at one coordinate. There may be points at your location, and you can always see these points regardless of your rotation. Points do not obstruct your vision to other points.

Return the maximum number of points you can see.

Example 1:

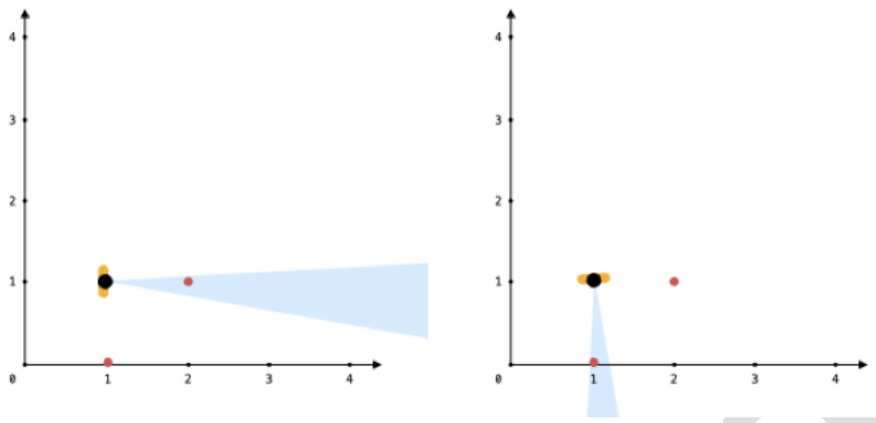


Input: points = [[2,1],[2,2],[3,3]], angle = 90, location = [1,1]
Output: 3
Explanation: The shaded region represents your field of view. All points can be made visible in your field of view, including [3,3] even though [2,2] is in front and in the same line of sight.

Example 2:

Input: points = [[2,1],[2,2],[3,4],[1,1]], angle = 90, location = [1,1]
Output: 4
Explanation: All points can be made visible in your field of view, including the one at your location.

Example 3:

	 Input: points = [[1,0],[2,1]], angle = 13, location = [1,1] Output: 1 Explanation: You can only see one of the two points, as shown above. **Constraints:** 1 <= points.length <= 10^5 points[i].length == 2 location.length == 2 0 <= angle < 360 0 <= posx, posy, xi, yi <= 100	
9	**Ninja Jump** **Problem statement :** Ninja is assigned a task to reach the last stone by his master. These stones are numbered with some value and in the form of an array. He is allowed to jump either odd-numbered jumps or even-numbered jumps and has to reach the last stone. So your task is to find the number of starting index from which he may start jumping so he reaches the last stones. You are provided with the given array and you have to find the number of starting index of the array from which Ninja can reach the end of the array by jumping some number of times. For jumping you have to follow below instructions: You may jump forward from index 'i' to index 'j' (with i < j) in the following way:	Hard

During odd-numbered jumps (i.e., jumps 1, 3, 5, ...), you jump to the index 'j' such that 'arr[i] <= arr[j]' and 'arr[j]' is the smallest possible value. If there are multiple such indices 'j', you can only jump to the smallest such index j.

During even-numbered jumps (i.e., jumps 2, 4, 6, ...), you jump to the index 'j' such that 'arr[i] >= arr[j]' and 'arr[j]' is the largest possible value. If there are multiple such indices 'j', you can only jump to the smallest such index 'j'.

It may be the case that for some index 'i', there are no legal jumps.

Sample Input 1:

```
2
5
20 23 22 24 25
5
5 6 4 4 7
```

Sample Output 1:

```
2
3
```

Explanation for sample input 1:

Test Case 1:

For the first test case, if we start from index i = 0, we can make our first jump to i = 2, since it is smaller than all other array values but then we cannot jump anymore.

If we start from the index i = 1, or i = 2 we can make the first jump to i = 3 then we cannot jump anymore.

If we start from index i = 3 we have to make only one jump and we reach the end.

And if we start from index i = 4 we are already at the end.

So we return '2' as our answer as there are '2' good starting index.

Test Case 2:

For the test case if we start From starting index i = 0, we make jumps to i = 1, i = 2, i = 3

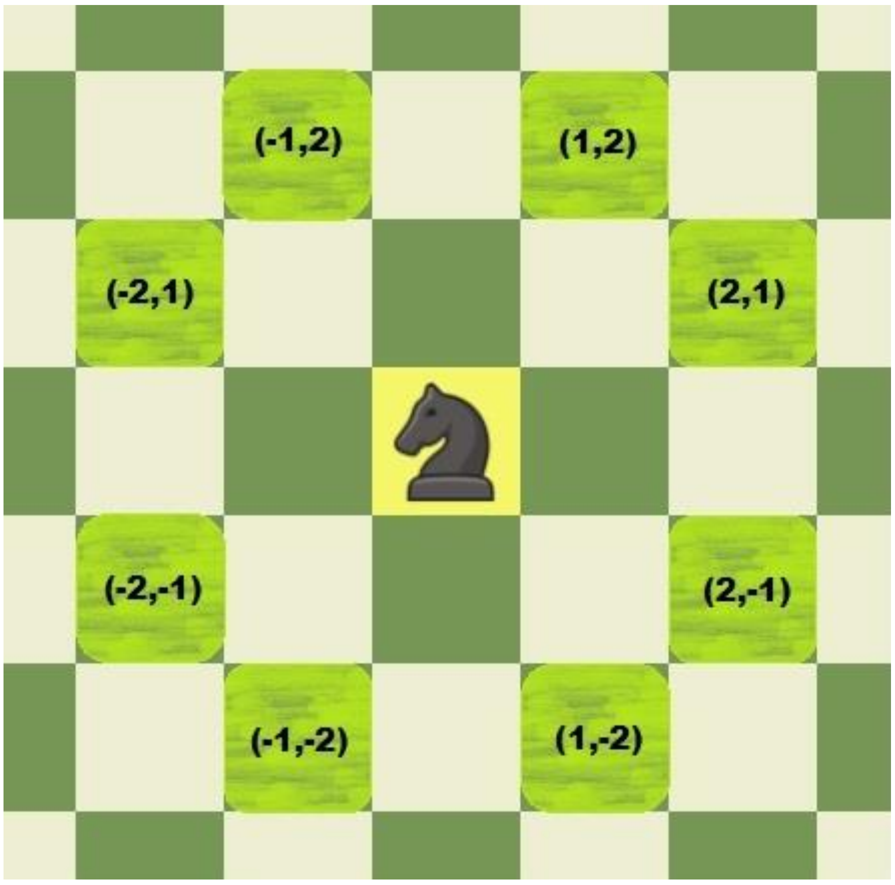
From starting index i = 1, we jump to i = 4, so we reach the end.

From starting index i = 2, we jump to i = 3, and then we can't jump anymore.

From starting index i = 3, we jump to i = 4, so we reach the end.

From starting index i = 4, we are already at the end.

In total, there are 3 different starting indices i = 1, i = 3, and i = 4, where we can reach the end with some number of jumps so we return '3' as our answer.

	Sample Input 2: 1 5 5 1 3 4 2 **Sample Output 2:** 3 **Explanation of sample input 2:** As we can reach the end by starting from index 1, 2, 4	
10	**Minimum Knight Moves** **Problem statement :** You are given an infinite chessboard (ie: the x-coordinates and y-coordinates can be anything between -infinity to +infinity). You have a knight placed at coordinates '(0, 0)'. Find the minimum number of steps needed to move the knight to '(X, Y)'. The knight has 8 possible moves, each move is two units in a cardinal direction, then one unit in an orthogonal direction. For example : As depicted in the photo below, the knight currently at (0, 0) can move to any of the 8 positions: (1, 2), (2, 1), (2, -1), (1, -2), (-1, -2), (-2, -1), (-2, 1), (-1, 2).  (-1,2) (1,2) (-2,1) (2,1) (-2,-1) (2,-1) (-1,-2) (1,-2)	Hard

Little practice is worth more than a ton of theory

Example :

If X = 1 and Y = -1, then we need to find out the minimum number of steps to move the knight from (0, 0) to (1, -1).

We need at least 2 steps to move the knight to the desired position.

First move: (0, 0) -> (2, 1)

Second move: (2,1) -> (1, -1)

Here we can see that there are many ways, but we need at least 2 steps. Therefore we will return the value 2.

Sample Input 1 :

```
2
1 1
1 0
```

Sample Output 1 :

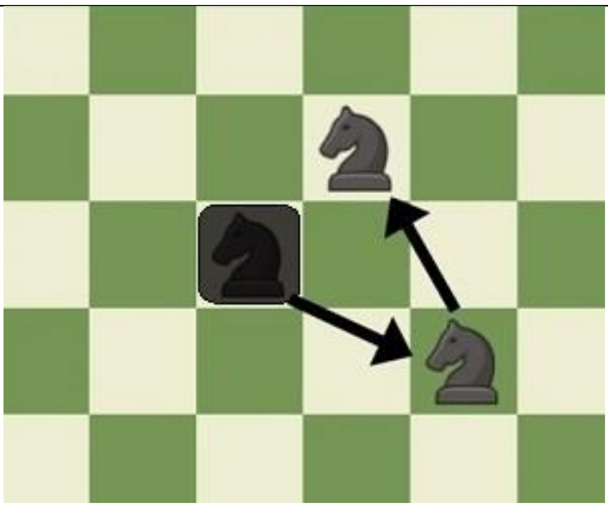
```
2
3
```

Explanation For Sample Input 1 :

For test case 1 :

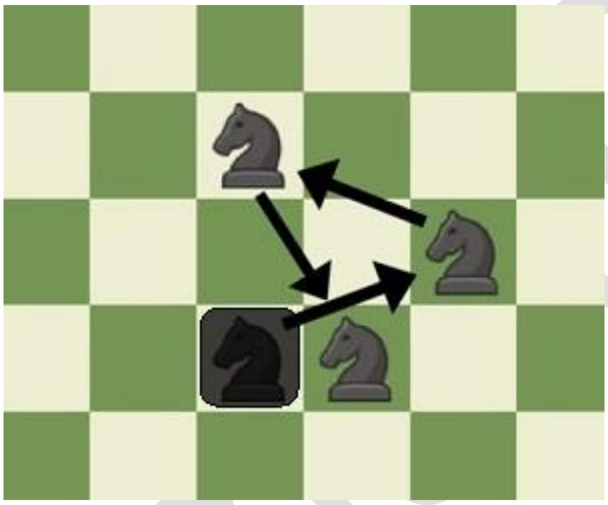
(0, 0) to (2, -1) to (1,1), therefore 2 steps are required. The other possible way is (0, 0) to (-1, 2) to (1, 1), but we require at least 2 steps to move from (0,0) to (1,-1).

Hence return value 2. Refer the image for better understanding:



For test case 2 :

(0, 0) to (2, 1) to (0, 2) to (1, 0), therefore 3 steps are required. Refer the image for better understanding:



Sample Input 2 :

2

12 5

5 12

Sample Output 2 :

7

7

Constraints :

1 <= T <= 10

-100 <= X, Y <= 100

11 **Reverse Subarray To Maximize Array Value** Hard

Problem statement : You are given an array of integers 'NUMS'. The beauty of this array can be defined as:
The sum of absolute difference of each two consecutive elements.
In one operation you can reverse one subarray of 'NUMS'. Your task is to find maximum beauty by performing the operation exactly once.

Sample Input 1:

```
1
5
5 4 1 7 2
```

Sample Output 1:

```
17
```

Explanation for sample input 1:
The current beauty of the array is |5 - 4| + |4 - 1| + |1 - 7| + |7 - 2| = 15, after reversing the subarray [4, 1, 7, 2] beauty becomes = |5 - 2| + |2 - 7| + |7 - 1| + |1 - 4| = 17, which is the maximum beauty we can obtain.

Sample Input 2:

```
2
4
4 3 7 1
5
4 5 3 7 6
```

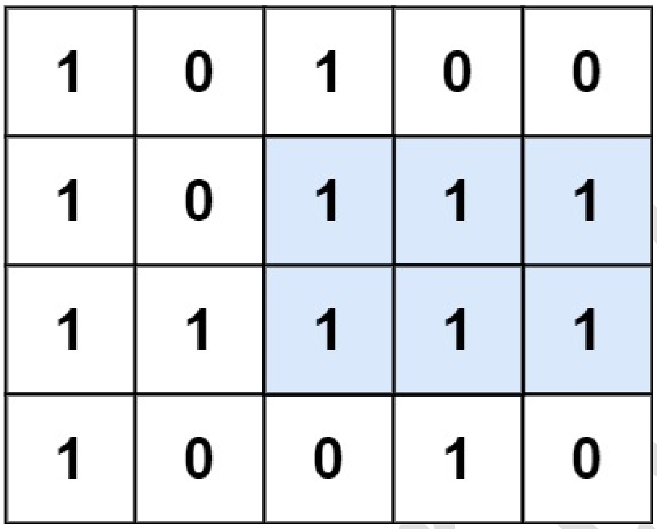
Sample Output 2:

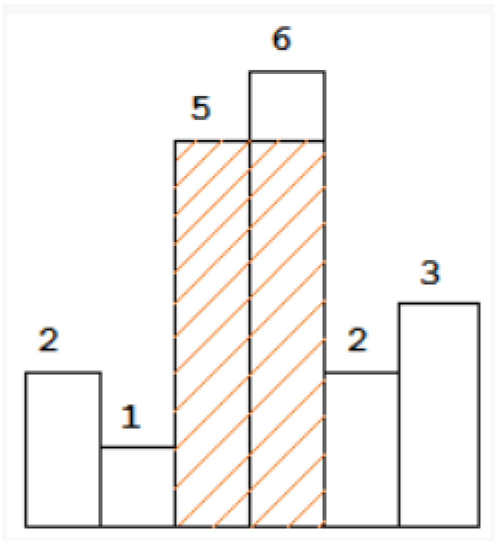
```
13
10
```

Explanation for sample input 2:
For test case 1, the initial beauty value is 1 + 4 + 6 = 11, on reversing the subarray [3, 7, 1] the beauty becomes 3 + 6 + 4 = 13.

For test case 2, reverse the subarray [5, 3, 7] to get a beauty value of 10.

Constraints:
1 <= 'T' <= 10
1 <= 'N' <= 10^5
1 <= NUMS[i] <= 10^8

	Where ‘N’ is the size of the array ‘NUMS’ and ‘NUMS[i]’ is the ith element of the array.	
12	**Maximal Rectangle** **Problem statement :** Given a rows x cols binary matrix filled with 0's and 1's, find the largest rectangle containing only 1's and return its area. **Example 1:**  Input: matrix = [["1","0","1","0","0"],["1","0","1","1","1"],["1","1","1","1","1"],["1","0","0","1","0"]] Output: 6 Explanation: The maximal rectangle is shown in the above picture. **Example 2:** Input: matrix = [["0"]] Output: 0 **Example 3:** Input: matrix = [["1"]] Output: 1 **Constraints:** rows == matrix.length	Hard

	cols == matrix[i].length 1 <= row, cols <= 200 matrix[i][j] is '0' or '1'.	
13	**Largest rectangle in a histogram** **Problem statement :** You have been given an array/list 'HEIGHTS' of length ‘N. 'HEIGHTS' represents the histogram and each element of 'HEIGHTS' represents the height of the histogram bar. Consider that the width of each histogram is 1. You are supposed to return the area of the largest rectangle possible in the given histogram. **For example :** In the below histogram where array/list elements are {2, 1, 5, 6, 2, 3}.  The area of largest rectangle possible in the given histogram is 10. **Sample Input 1 :** 2 10 1 0 1 2 2 2 2 1 0 2 10 1 2 1 0 1 1 0 0 2 2 **Sample Output 1 :** 8 4 **Explanation For Sample Input 1 :**	Hard

Little practice is worth more than a ton of theory

In the first test case, the area of the largest rectangle of the given histogram is 8 in the rectangle starting from index 4 to index 7 in the given array/list.

In the second test case, the area of the largest rectangle of the given histogram is 4 in the rectangle starting from index 9 to index 10 in the given array/list.

Sample Input 2 :

```
2
10
8 6 3 5 0 0 4 10 2 5
10
6 1 8 10 5 7 0 4 5 8
```

Sample Output 2 :

```
12
20
```

Explanation For Sample Input 2 :

In the first test case, the area of the largest rectangle of the given histogram is 12.

In the second test case, the area of the largest rectangle of the given histogram is 20.

Constraints :

1 <= T <= 10

1 <= N <= 10^6

0 <= HEIGHTS[i] <= 10^9

Where 'T' is the number of test cases.
'N' is the length of the given array/list.
And, HEIGHTS[i] denotes the height of the 'ith' histogram bar.

14 **Longest Harmonious Subsequence** Hard

Problem statement

You are given an array 'ARR' of 'N' integers. Your task is to find the longest harmonious subsequence of this array.

A sequence is Harmonious if the difference between the maximum and the minimum of all the elements is exactly 1.

For example

[3, 4, 3, 3, 3, 3, 4] is a harmonic array as the maximum of all the elements is 4 and minimum of all the elements is 3. So, the difference between the maximum and the minimum = 4 - 3 = 1.

Sample Input 1:

```
2
4
1 2 2 1
4
1 2 3 4
```

Sample Output 1:

```
4
2
```

Explanation for Sample Input 1:

In the first test case, the given array is [1, 2, 2, 1]. If we take the complete array, then the maximum of all the elements is 2 and the minimum of all the elements is 1. So, the difference between the maximum and the minimum = 2 - 1 = 1. Hence, the longest Harmonic subsequence is [1, 2, 2, 1] and its length is 4.

In the second test case, the given array is [1, 2, 3, 4]. If we take the complete array, then the maximum of all the elements is 4 and the minimum of all the elements is 1.

So, the difference between the maximum and the minimum = 4 - 1 = 3. So, it is not a Harmonic subsequence. If we take subsequence as [1, 2], then the maximum of all the elements is 2 and the minimum of all the elements is 1.

So, the difference between the maximum and the minimum = 2 - 1 = 1. So, it is a harmonic subsequence and this is the longest harmonic subsequence. If we take subsequence as [2, 3] or [3, 4], then we will get the same answer.

Sample Input 2:

```
2
2
1 2
10
1 2 2 3 4 5 1 1 1 1
```

Sample Output 2:

```
2
```

	7 **Constraints:** 1<= T <= 100 1 <= N <= 10^4 1 <= ARR[i] <= 10^8	
15	**Sliding Window Maximum** You are given an array of integers nums, there is a sliding window of size k which is moving from the very left of the array to the very right. You can only see the k numbers in the window. Each time the sliding window moves right by one position. Return the max sliding window. **Example 1:** Input: nums = [1,3,-1,-3,5,3,6,7], k = 3 Output: [3,3,5,5,6,7] Explanation: Window position Max --------------- ----- [1 3 -1] -3 5 3 6 7 3 1 [3 -1 -3] 5 3 6 7 3 1 3 [-1 -3 5] 3 6 7 5 1 3 -1 [-3 5 3] 6 7 5 1 3 -1 -3 [5 3 6] 7 6 1 3 -1 -3 5 [3 6 7] 7 **Example 2:** Input: nums = [1], k = 1 Output: [1] **Constraints:** 1 <= nums.length <= 105 -104 <= nums[i] <= 104 1 <= k <= nums.length	Hard

16	**Merge k Sorted Lists** You are given an array of k linked-lists lists, each linked-list is sorted in ascending order. Merge all the linked-lists into one sorted linked-list and return it. **Example 1:** Input: lists = [[1,4,5],[1,3,4],[2,6]] Output: [1,1,2,3,4,4,5,6] Explanation: The linked-lists are: [1->4->5, 1->3->4, 2->6] merging them into one sorted list: 1->1->2->3->4->4->5->6 **Example 2:** Input: lists = [] Output: [] Example 3: Input: lists = [[]] Output: [] **Constraints:** k == lists.length 0 <= k <= 10^4 0 <= lists[i].length <= 500 -10^4 <= lists[i][j] <= 10^4 lists[i] is sorted in ascending order. The sum of lists[i].length will not exceed 10^4.	Hard

17	**Most Stones Removed with Same Row or Column**	Hard

Problem statement : On a 2-D plane, we place *'n'* stones at some integer coordinate points. Each coordinate point may have at most one stone.

A stone can be removed if it shares either the same row or the same column as another stone that has not been removed.

You are given an array ***'stones'*** of length 'n' where 'stones[i]' = '[ri, ci]' represent the ith stone’s location i.e 'ri' is the row coordinate of the 'ith' stone and 'ci' is the column coordinate of the 'ith' stone. Your task is to return the largest possible number of stones that can be removed.

Example:

Input: 'stones' = [[0,1] [1,0] [0,0]]

Output: 2

Explanation:

We can remove the 1st stone at [0,1] as it has the same row as the 3rd stone [0, 0]. And remove the 2nd stone because it has the same column [0,1] as the 3rd stone.

Sample Input 1 :

```
5
2 0
2 1
3 1
3 2
5 5
```

Expected Answer:

3

Explanation of the Sample Input 1:

For this test case:

The answer will be 3:

	We can remove the stone at [2,1] because it has the same row as the stone [2,0], stone at [3,1] because it has the same column as [2,1] and stone at [3,2] because it shares the same row as [3,1]. **Sample Input 2 :** 1 1 1 **Expected Answer:** 0 ***Constraints:*** 1 <= n <= 10^4 0 <= stones[i][j] <= 10^3	
18	**Length of the Longest Valid Substring** **Problem statement :** You are given a string word and an array of strings forbidden. A string is called **valid** if none of its substrings are present in forbidden. Return *the length of the* ***longest valid substring*** *of the string* word. A **substring** is a contiguous sequence of characters in a string, possibly empty. **Example 1:** **Input:** word = "cbaaaabc", forbidden = ["aaa","cb"] **Output:** 4 **Explanation:** There are 11 valid substrings in word: "c", "b", "a", "ba", "aa", "bc", "baa", "aab", "ab", "abc" and "aabc". The length of the longest valid substring is 4. It can be shown that all other substrings contain either "aaa" or "cb" as a substring. **Example 2:** **Input:** word = "leetcode", forbidden = ["de","le","e"] **Output:** 4 **Explanation:** There are 11 valid substrings in word: "l", "t", "c", "o", "d", "tc", "co", "od", "tco", "cod", and "tcod". The length of the longest valid substring is 4. It can be shown that all other substrings contain either "de", "le", or "e" as a substring.	Hard

Constraints:

- 1 <= word.length <= 10^5
- word consists only of lowercase English letters.
- 1 <= forbidden.length <= 10^5
- 1 <= forbidden[i].length <= 10
- forbidden[i] consists only of lowercase English letters.

19 **Text Justification** Hard

Problem statement : Given an array of strings words and a width maxWidth, format the text such that each line has exactly maxWidth characters and is fully (left and right) justified.

You should pack your words in a greedy approach; that is, pack as many words as you can in each line. Pad extra spaces ' ' when necessary so that each line has exactly maxWidth characters.

Extra spaces between words should be distributed as evenly as possible. If the number of spaces on a line does not divide evenly between words, the empty slots on the left will be assigned more spaces than the slots on the right.

For the last line of text, it should be left-justified, and no extra space is inserted between words.

Note:

- A word is defined as a character sequence consisting of non-space characters only.
- Each word's length is guaranteed to be greater than 0 and not exceed maxWidth.
- The input array words contains at least one word.

Example 1:

Input: words = ["This", "is", "an", "example", "of", "text", "justification."], maxWidth = 16

Output:

```
[
   "This    is    an",
   "example  of text",
   "justification.  "
]
```

Example 2:

Input: words = ["What","must","be","acknowledgment","shall","be"], maxWidth = 16

Output:

```
[
  "What   must   be",
  "acknowledgment  ",
  "shall be        "
]
```

Explanation: Note that the last line is "shall be   " instead of "shall   be", because the last line must be left-justified instead of fully-justified.

Note that the second line is also left-justified because it contains only one word.

Example 3:

Input: words = ["Science","is","what","we","understand","well","enough","to","explain","to","a",s"computer.","Art","is","everything","else","we","do"], maxWidth = 20

Output:

```
[
  "Science  is  what we",
  "understand      well",
  "enough to explain to",
  "a  computer.  Art is",
  "everything  else  we",
  "do                  "
]
```

Constraints:

- 1 <= words.length <= 300
- 1 <= words[i].length <= 20
- words[i] consists of only English letters and symbols.
- 1 <= maxWidth <= 100

words[i].length <= maxWidth

20 **Design a Text Editor** — Hard

Problem statement : Design a text editor with a cursor that can do the following:

- **Add** text to where the cursor is.
- **Delete** text from where the cursor is (simulating the backspace key).

- **Move** the cursor either left or right.

When deleting text, only characters to the left of the cursor will be deleted. The cursor will also remain within the actual text and cannot be moved beyond it. More formally, we have that 0 <= cursor.position <= currentText.length always holds.

Implement the TextEditor class:

- TextEditor() Initializes the object with empty text.
- void addText(string text) Appends text to where the cursor is. The cursor ends to the right of text.
- int deleteText(int k) Deletes k characters to the left of the cursor. Returns the number of characters actually deleted.
- string cursorLeft(int k) Moves the cursor to the left k times. Returns the last min(10, len) characters to the left of the cursor, where len is the number of characters to the left of the cursor.
- string cursorRight(int k) Moves the cursor to the right k times. Returns the last min(10, len) characters to the left of the cursor, where len is the number of characters to the left of the cursor.

Example 1:

Input

["TextEditor", "addText", "deleteText", "addText", "cursorRight", "cursorLeft", "deleteText", "cursorLeft", "cursorRight"]

[[], ["leetcode"], [4], ["practice"], [3], [8], [10], [2], [6]]

Output

[null, null, 4, null, "etpractice", "leet", 4, "", "practi"]

Explanation

```
TextEditor textEditor = new TextEditor(); // The current text is "|". (The '|'
character represents the cursor)
textEditor.addText("leetcode"); // The current text is "leetcode|".
textEditor.deleteText(4); // return 4
                          // The current text is "leet|".
                          // 4 characters were deleted.
textEditor.addText("practice"); // The current text is "leetpractice|".
textEditor.cursorRight(3); // return "etpractice"
                           // The current text is "leetpractice|".
```

// The cursor cannot be moved beyond the actual text and thus did not move.
// "etpractice" is the last 10 characters to the left of the cursor.
textEditor.cursorLeft(8); // return "leet"
// The current text is "leet|practice".
// "leet" is the last min(10, 4) = 4 characters to the left of the cursor.
textEditor.deleteText(10); // return 4
// The current text is "|practice".
// Only 4 characters were deleted.
textEditor.cursorLeft(2); // return ""
// The current text is "|practice".
// The cursor cannot be moved beyond the actual text and thus did not move.
// "" is the last min(10, 0) = 0 characters to the left of the cursor.
textEditor.cursorRight(6); // return "practi"
// The current text is "practi|ce".
// "practi" is the last min(10, 6) = 6 characters to the left of the cursor.

Constraints:

- 1 <= text.length, k <= 40
- text consists of lowercase English letters.
- At most 2 * 10^4 calls **in total** will be made

to addText, deleteText, cursorLeft and cursorRight.