**Question 1: Beggars outside temple**

There are **A** beggars sitting in a row outside a temple. Each beggar initially has an empty pot. When the devotees come to the temple, they donate some amount of coins to these beggars. Each devotee gives a fixed amount of coin(according to their faith and ability) to some **K** beggars sitting next to each other.  
  
Given the amount **P** donated by each devotee to the beggars ranging from **L** to **R** index, where **1** <= **L** <= **R** <= **A**, find out the final amount of money in each beggar's pot at the end of the day, provided they don't fill their pots by any other means.  
For ith devotee B[i][0] = L, B[i][1] = R, B[i][2] = P, Given by the 2D array **B**

**Problem Constraints**

1 <= A <= 2 \* 105  
1 <= L <= R <= A  
1 <= P <= 103  
0 <= len(B) <= 105

**Logic –**

1. Create an array of size N with all zeros
2. Add the value (P) to the L-1th index and subtract the value (P) to Rth index.
3. Take prefix sum of the array and return the final array

**Code –**

class Solution:

    # @param A : integer

    # @param B : list of list of integers

    # @return a list of integers

    def solve(self, A, B):

        temp = [0] \* A

        for i in B:

            st = i[0]

            en = i[1]

            val = i[2]

            temp[st-1] += val

            if en == A:

                continue

            temp[en] -= val

        for i in range(1, len(temp)):

            temp[i] += temp[i-1]

        return temp

**Question 2: Rainwater trapped**

Given a vector **A** of non-negative integers representing an elevation map where the width of each bar is 1, compute how much water it is able to trap after raining.

**Problem Constraints**

1 <= **|A|** <= 100000

**Logic –**

1. Create suffix maximum array (ie – the current element = largest number among the elements to it’s right)
2. Maintain a variable containing the current max from left.
3. Add the min of current max and (i) th term of suffix maximum array to the ans variable
4. Substract height A[i] of the building from the sum.

**Code –**

class Solution:

    # @param A : tuple of integers

    # @return an integer

    def trap(self, A):

        left = []

        temp = -1

        for i in range(len(A)):

            if temp < A[i]:

                temp = A[i]

            left.append(temp)

        temp = -1

        res = 0

        for i in range(len(A)-1, -1, -1):

            if temp < A[i]:

                temp = A[i]

            res += (min(left[i], temp) - A[i])

        return res

**Question 3: Max Sum Contiguous Subarray ( Kadane Algorithm )**

Find the **contiguous non-empty** subarray within an array, **A** of length **N,** with the **largest sum**.

**Problem Constraints**

1 <= N <= 1e6  
-1000 <= A[i] <= 1000

**Logic –**

1. Maintain a sum variable, and keep adding each term to the sum.
2. If the sum becomes less than zero, make the sum = 0.
3. Maintain a “ans” variable and keep equating it with the maximum sum.

**Code –**

class Solution:

    # @param A : tuple of integers

    # @return an integer

    def maxSubArray(self, A):

        res = 0

        ans = -1000000000

        for i in range(len(A)):

            res+=A[i]

            ans = max(res, ans)

            if res <= 0:

                res = 0

        return ans

**Question 4: Add 1 to number**

Given a **non-negative** number represented as an array of digits, add **1** to the number ( increment the number represented by the digits ).

The digits are stored such that the most significant digit is at the head of the list.

**NOTE:** Certain things are intentionally left unclear in this question which you should practice asking the interviewer. For example: for this problem, the following are some good questions to ask :

* **Q:** Can the input have **0's** before the most significant digit. Or, in other words, is **0 1 2 3** a valid input?
* **A:** For the purpose of this question, **YES**
* **Q:** Can the output have **0's** before the most significant digit? Or, in other words, is **0 1 2 4** a valid output?
* **A:** For the purpose of this question, **NO**. Even if the input has zeroes before the most significant digit.

**Problem Constraints** 1 <= size of the array <= 1000000

**Logic –**

1. Maintain a variable “carry” = 1
2. Iterate the array from end and do following steps:
3. If carry = 0 append the current element (A[i]) to the ans
4. If carry = 1 and A[i] = 9 append 0 to the ans and make carry = 1
5. If carry = 1 and A[i] != 9 append A[i] + 1 to the ans
6. After the loop if carry = 1 append 1 to the ans

**Code –**

class Solution:

    # @param A : list of integers

    # @return a list of integers

    def plusOne(self, A):

        ans = []

        carry = 1

        temp = 0

        # Cancel all zero

        for i in range(len(A)):

            if A[i] != 0:

                temp = i

                break

        for i in range(len(A)-1,temp-1,-1):

            if carry == 0:

                ans.append(A[i])

            if carry == 1:

                if A[i] == 9:

                    ans.append(0)

                    carry = 1

                else:

                    ans.append(A[i]+1)

                    carry = 0

        if carry == 1:

            ans.append(1)

            carry = 0

        return reversed(ans)

**Question 5: Max Non Negative SubArray**

Given an array of integers, **A** of length **N**, find out the **maximum sum** sub-array of non negative numbers from **A**.

The sub-array should be contiguous i.e., a sub-array created by choosing the second and fourth element and skipping the third element is invalid.

Maximum sub-array is defined in terms of the sum of the elements in the sub-array.

Find and return the required subarray.

Constraints 1 <= N <= 1e5, -INT\_MAX < A[i] <= INT\_MAX

Logic –

1. If an element in array A is less than zero make the sum variable = 0.
2. Keep updating the start and end variable at every point when sum exceeds the previous maximum sum.

Code –

class Solution:

    # @param A : list of integers

    # @return a list of integers

    def maxset(self, A):

        st = 0

        start = 0

        end = 0

        sum1 = 0

        ans = -1000000000

        flag = 0

        for i in range(len(A)):

            sum1+=A[i]

            if ans < sum1:

                start = st

                end = i

                ans = sum1

            elif ans == sum1:

                if ((end-start) < (i-st)):

                    start = st

                    end = i

            if A[i] < 0:

                st = i+1

                sum1 = 0

            elif A[i] >= 0:

                flag = 1

        if flag == 0:

            return []

        return A[start:end+1]

**Question 6: Flip**

You are given a binary string **A**(i.e., with characters **0** and **1**) consisting of characters **A1**, **A2**, ..., **AN**. In a single operation, you can choose two indices, **L** and **R,** such that **1** ≤ **L** ≤ **R** ≤ **N** and flip the characters **AL**, **AL+1**, ..., **AR**. By flipping, we mean changing character **0** to **1** and vice-versa.

Your aim is to perform **ATMOST** one operation such that in the final string number of **1s** is maximized.

If you don't want to perform the operation, return an **empty** array. Else, return an array consisting of two elements denoting **L** and **R**. If there are multiple solutions, return the **lexicographically smallest** pair of **L** and **R**.

**NOTE:**Pair **(a, b)** is lexicographically smaller than pair **(c, d)** if **a** < **c** or, if **a** == **c** and **b** < **d**.

**Problem Constraints**

1 <= size of string <= 100000

Logic –

1. Find the sub array with maximum zeros in it. So that after flipping maximum number of zeros are converted to 1.
2. Convert ‘0’ to 1 and ‘1’ to -1
3. Find the subarray with maximum sum using kadane’s algorithm.

Code –

class Solution:

    # @param A : string

    # @return a list of integers

    def flip(self, A):

        flag = 0

        A = list(A)

        for i in range(len(A)):

            if A[i] == "0":

                A[i] = 1

                flag = 1

            elif A[i] ==  "1":

                A[i] = -1

        if flag == 0:

            return []

        ans = -1000000000

        sum1 = 0

        L=0

        R=0

        st = 0

        for i in range(len(A)):

            sum1+=A[i]

            if ans < sum1:

                ans = sum1

                L = st

                R = i

            if sum1 < 0:

                sum1 = 0

                st=i+1

        return [L+1, R+1]

**Question 7: Sum of all Submatrices (Contribution Technique)**

Given a 2D Matrix **A** of dimensions **N\*N**, we need to return the sum of all possible submatrices.  
  
**Problem Constraints**

1 <= N <=30

0 <= A[i][j] <= 10

Logic – Find the contribution of each element to the sum.

1. Find all possible top left corners of the subarrays = (i+1) \* (j+1)
2. Find all possible bottom right corners of the subarrays = (n-j) \* (m-i)
3. So, number of matrices containing A[i][j] = (i+1) \* (j+1) \* (n-j) \* (m-i)
4. Sum contributed = A[i][j] \* (i+1) \* (j+1) \* (n-j) \* (m-i)

Code –

class Solution:

    # @param A : list of list of integers

    # @return an integer

    def solve(self, A):

        ans = 0

        n = len(A)

        m = len(A[0])

        for i in range(n):

            for j in range(m):

                ans += ((i+1) \* (j+1) \* (n-i) \* (m-j) \* A[i][j])

        return ans

**Question 8: Sub Matrix Sum Queries**

Given a matrix of integers **A** of size **N x M** and multiple queries **Q**, for each query, find and return the submatrix sum.

Inputs to queries are **top left (b, c)** and **bottom right (d, e)** indexes of submatrix whose sum is to find out.

**NOTE:**

* Rows are numbered from top to bottom, and columns are numbered from left to right.
* Sum may be large, so return the answer mod 109 + 7.

**Problem Constraints**

1 <= N, M <= 1000  
-100000 <= A[i] <= 100000  
1 <= Q <= 100000  
1 <= B[i] <= D[i] <= N  
1 <= C[i] <= E[i] <= M

Logic – Create a prefix matrix and find the sum.

1. Create a zero matrix of size n\*m
2. Make the first column = first column of A matrix
3. Take row wise sum. prefMat[i][j] += prefMat[i-1][j]
4. Take column wise sum prefMat[i][j] += prefMat[i][j-1]
5. Now find sum using prefMat. Sum = prefMat[c][d] – prefMat[a-1][d] – prefMat[c][b-1] + prefMat[a-1][b-1], where (a,b) – top left corner and (c,d) – bottom right corner.
6. Take care if a-1 and b-1 is not negative.

Code –

class Solution:

    def solve(self, A, B, C, D, E):

        prefMat = [ [0]\*len(A[0]) for \_ in range(len(A)) ]

        for i in range(len(A[0])):

            prefMat[0][i] = A[0][i]

        for i in range(1, len(A)):

            for j in range(0, len(A[0])):

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

        for i in range(len(A)):

            for j in range(1, len(A[0])):

                prefMat[i][j] += prefMat[i][j-1]

        ans = []

        for i in range(len(B)):

            a1 = B[i] - 1

            a2 = C[i] - 1

            b1 = D[i] - 1

            b2 = E[i] - 1

            temp = prefMat[b1][b2]

            if a1 > 0:

                temp -= prefMat[a1-1][b2]

            if a2 > 0:

                temp -= prefMat[b1][a2-1]

            if a1 > 0 and a2 > 0:

                temp += prefMat[a1-1][a2-1]

            ans.append(temp%(1000000007))

        return ans

**Question 9: Search in a row wise and column wise sorted matrix**

Given a matrix of integers **A** of size **N x M** and an integer **B**.

In the given matrix every row and column is sorted in increasing order. Find and return the position of **B** in the matrix in the given form:

 If A[i][j] = B then return (i \* 1009 + j)

 If B is not present return -1.  
**Note 1:** Rows are numbered from top to bottom and columns are numbered from left to right.  
**Note 2:** If there are multiple B in A then return the smallest value of i\*1009 +j such that A[i][j]=B.

**Problem Constraints**

1 <= N, M <= 1000  
-100000 <= A[i] <= 100000  
-100000 <= B <= 100000

Logic – Start from top right corner and moving left will reduce the current value and moving down will increase the value.

1. Start with j = M-1 and i = 0.
2. If A[i][j] > B move left ie decrement j
3. If A[i][j] < B move right ie increment i
4. If A[i][j] == B, calculate (i \* 1009 + j) and store a minimum value in the ans then move left ie decrement j

Note – One can also start from bottom left corner.

Code –

class Solution:

    # @param A : list of list of integers

    # @param B : integer

    # @return an integer

    def solve(self, A, B):

        n = len(A)

        m = len(A[0])

        i = 0

        j = m-1

        res = 10000000000

        while (i<n and j>=0):

            if A[i][j] == B:

                temp = ((i+1) \* 1009 + (j+1))

                res = min(res, temp)

                j-=1

            elif A[i][j] < B:

                i+=1

            else:

                j-=1

        if res == 10000000000:

            return -1

        return res

**Question 10: Maximum Submatrix Sum (Row wise and Column wise sorted Matrix)**

Given a row-wise and column-wise sorted matrix **A**of size **N \* M**.  
Return the maximum non-empty submatrix sum of this matrix.

**Problem Constraints**

1 <= N, M <= 1000  
-109 <= A[i][j] <= 109

Logic –

1. Bottom right element of the sorted matrix is greatest element in the matrix, so it has to be included.
2. So, using the prefix matrix find sum of submatrix with fixed bottom right corner and vary the top left corner.
3. Find the max sum at each iteration.

Code –

class Solution:

    # @param A : list of list of integers

     # @return an long

    def solve(self, A):

        prefMat = [[0]\*len(A[0]) for \_ in range(len(A))]

        for i in range(len(A)):

            prefMat[i][0] = A[i][0]

        # Row wise sum

        for i in range(len(A)):

            for j in range(1, len(A[0])):

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

        # Col wize sum

        for i in range(1, len(A)):

            for j in range(len(A[0])):

                prefMat[i][j] += prefMat[i-1][j]

        ans = -100000000000000

        for i in range(len(A)):

            for j in range(len(A[0])):

                b1 = len(A)-1

                b2 = len(A[0]) - 1

                a1 = i

                a2 = j

                temp = prefMat[b1][b2]

                if a1>0:

                    temp -= prefMat[a1-1][b2]

                if a2>0:

                    temp -= prefMat[b1][a2-1]

                if a1>0 and a2>0:

                    temp += prefMat[a1-1][a2-1]

                if temp > ans:

                    ans = temp

        return ans

**Question 11: Maximum Sum Submatrix**

Given a **N \* M** 2D matrix **A**. Find the maximum sum sub-matrix from the matrix A. Return the Sum.

**Problem Constraints**

1 <= N, M <= 300  
-104 <= A[i][j] <= 104

Logic –

1. Create a column wise sum matrix.
2. Iterate the top row (tr) from 0 to n-1.
3. Iterate the bottom row (br) from tr to n-1.
4. Apply the kadane’s algorithm to find the maximum sum of the sub matrices with rows tr and br and columns from j 0 to m-1.

Note – We can also use the row wise matrix and apply kadane’s algorithm to the columns.

Code –

import sys

class Solution:

    # @param A : list of list of integers

    # @return an integer

    def solve(self, A):

        m = len(A[0])

        n = len(A)

        colsum = [[0 for j in range(m)] for i in range(n)]

        # Create a column wise sum

        for i in range(n):

            for j in range(m):

                if i==0:

                    colsum[i][j] = A[i][j]

                else:

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

        ans = -sys.maxsize

        # Iterate the top row from 0 to n

        for tr in range(n):

            # Iterate the bottom row from tr to n

            for br in range(tr, n):

                sum1 = 0

                # Apply kadanes algorithm on the sum of the columns with rows tr and br

                for j in range(m):

                    if tr >= 1:

                        sum1 = sum1 + colsum[br][j] - colsum[tr-1][j]

                    else:

                        sum1 = sum1 + colsum[br][j]

                    ans = max(ans, sum1)

                    if sum1 < 0:

                        sum1 = 0

        return ans

**Question 12: Maximum Sum Square Matrix**

Given a 2D integer matrix **A** of size N x N, find a B x B submatrix where B<= N and B>= 1, such that **the** **sum of all the elements in the submatrix is maximum**.  
  
**Problem Constraints**

1 <= N <= 103.

1 <= B <= N

-102 <= A[i][j] <= 102.

Logic –

1. Top left corners of all possible sub square matrices goes from (0, 0) to (N-B, N-B)
2. If (i,j) is the top left corner, then bottom right corner is (i+B-1, j+B-1)
3. Using the prefix Matrix find the sum using these 2 co-ordinates.

Code –

import sys

class Solution:

    def getSum(self, prefMat, x1, y1, x2, y2):

        temp = prefMat[x2][y2]

        if (x1>0):

            temp -= prefMat[x1-1][y2]

        if (y1>0):

            temp -= prefMat[x2][y1-1]

        if (x1>0 and y1>0):

            temp += prefMat[x1-1][y1-1]

        return temp

    def solve(self, A, B):

        n = len(A)

        ans = -sys.maxsize - 1

        prefMat = [[0 for i in range(n)] for i in range(n)]

        for i in range(n):

            prefMat[0][i] = A[0][i]

        for i in range(1,n):

            for j in range(n):

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

        for i in range(n):

            for j in range(1, n):

                prefMat[i][j] += prefMat[i][j-1]

        for i in range(n-B+1):

            for j in range(n-B+1):

                ans = max(ans, self.getSum(prefMat, i,j, i+B-1, j+B-1))

        return ans

**Question 13: Merge Intervals**

Given a set of non-overlapping intervals, insert a new interval into the intervals (merge if necessary).

You may assume that the intervals were initially sorted according to their start times.

**Problem Constraints**

0 <= **|intervals|** <= 100000

Logic – There are 3 possible cases for the given problem.

1. Interval is on complete left. In this case put the current interval in the ans variable and check next interval.
2. If the interval is overlapping. In this case change the elements of the current interval to (min(current.start, lastInterval.start), max(current.end, lastInterval.end)).
3. Interval is on complete right. In this case add all the remaining elements to the ans array and return ans.

Code –

# Definition for an interval.

# class Interval:

#     def \_\_init\_\_(self, s=0, e=0):

#         self.start = s

#         self.end = e

class Solution:

    def insert(self, intervals, newInterval):

        ans = []

        for i in range(len(intervals)):

            # If Interval at left

            if newInterval.start > intervals[i].end:

                ans.append(intervals[i])

            # If interval at right

            elif intervals[i].start > newInterval.end:

                ans.append(newInterval)

                while(i<len(intervals)):

                    ans.append(intervals[i])

                    i+=1

                return ans

            # If overlap

            else:

                newInterval.start = min(newInterval.start, intervals[i].start)

                newInterval.end = max(newInterval.end, intervals[i].end)

        ans.append(newInterval)

        return ans

**Question 14: First missing Integer**

Given an unsorted integer array, **A** of size **N**. Find the first missing positive integer.

**Note:**Your algorithm should run in O(n) time and use constant space.

**Problem Constraints**

1 <= N <= 1000000

-109 <= A[i] <= 109

Logic 1 – Create an array with all values = 0. Fill true for all the indices present in the given array. Iterate the array and find the first index with value = False.

Logic 2 – Swap Method – Place the elements between 1 and N to their correct index

1. Start with i = 0. Check if current element (A[i]) is between [1, len(A)]. If not check next element (i+=1).
2. If yes (1<= A[i] <= len(A)), then check if A[A[i]-1] = A[i], if not swap the element.
3. If Yes (A[A[i]-1] = A[i]) it means the element is at correct index. So check next element (i+=1).
4. Iterate the array and return the index that does not satisfy A[index] = index+1.

Code –

class Solution:

    # @param A : list of integers

    # @return an integer

    def firstMissingPositive(self, A):

        n = len(A)

        i = 0

        while(i<n):

            if A[i] >= 1 and A[i] <=n:

                correctIdx = A[i] - 1

                if A[correctIdx] != A[i]:

                    # Swap

                    A[correctIdx], A[i] = A[i], A[correctIdx]

                else:

                    i+=1

            else:

                i+=1

        for i in range(len(A)):

            if A[i] != (i+1):

                return i+1

        return n+1

**Question 15: Maximum Absolute Difference**

You are given an array of **N** integers, **A1, A2, .... AN**.  
  
Return the maximum value of f(i, j) for all 1 ≤ i, j ≤ N. f(i, j) is defined as |A[i] - A[j]| + |i - j|, where |x| denotes absolute value of x.

**Problem Constraints**

1 <= N <= 100000

-109 <= A[i] <= 109

Logic – Get rid of the modulus sign. 4 cases will be formed.

1. If A[i] > A[j] and i>j then ans = (A[i]+i) – (A[j]+j)
2. If A[i] < A[j] and i>j then ans = (A[j]-j) – (A[i]-i)
3. If A[i] > A[j] and i<j then ans = (A[i]-i) – (A[j]-j)
4. If A[i] < A[j] and i<j then ans = (A[j]+j) – (A[i]+i)
5. 2 cases are redundant. So we consider only cases 1 and 3.
6. Maximize the left parts (A[i]+i) and (A[i]-i) and minimize the respective left parts (A[j]+j) and (A[j]-j) and find 2 answers. As we want maximum difference we return the greatest difference.

Code –

import sys

class Solution:

    # @param A : list of integers

    # @return an integer

    def maxArr(self, A):

        mx1 = mx2 = -1 \* sys.maxsize

        mn1 = mn2  = sys.maxsize

        for i in range(len(A)):

            mx1 = max(mx1, (A[i]+i))

            mx2 = max(mx2, (A[i]-i))

            mn1 = min(mn1, (A[i]+i))

            mn2 = min(mn2, (A[i]-i))

        return max(mx1-mn1, mx2-mn2)

**Question 16: Row with maximum number of ones**

Given a binary sorted matrix **A** of size **N x N**. Find the row with the **maximum** number of **1**.

**NOTE:**

* If two rows have the maximum number of 1 then return the row which has a **lower index**.
* Rows are numbered from top to bottom and columns are numbered from left to right.
* Assume **0-based** indexing.
* Assume each row to be sorted by values.
* Expected time complexity is O(rows).

**Problem Constraints**

1 <= N <= 1000

0 <= A[i] <= 1

Logic –

1. Start checking for zero from the top right corner (i=0 and j = n-1).
2. If the current element (A[i][j]) is equal to one decrement j (Move Left) and update the ans = current row (i).
3. If current element is equal to zero then move to the bottom (i+=1).

Code –

class Solution:

    # @param A : list of list of integers

    # @return an integer

    def solve(self, A):

        n = len(A)

        i = 0

        j = n-1

        ans = 0

        while(i<n and j>=0):

            if A[i][j] == 1:

                j-=1

                ans = i

            else:

                i+=1

        return ans

**Question 17: Minimum Swaps**

Given an array of integers **A** and an integer **B**, find and return the minimum number of swaps required to bring all the numbers less than or equal to **B** together.

**Note:** It is possible to swap any two elements, not necessarily consecutive.

**Problem Constraints**

1 <= length of the array <= 100000  
-109 <= A[i], B <= 109

Logic – Sliding window approach

1. Count total number elements less than or equal to B in the array A.
2. Apply Sliding window technique with length of window = number of elements less than or equal to B.
3. Number of swaps = Number of elements less than or equal to B in array A – number of elements less than or equal to B in the window.
4. Find the minimum number of swaps in every iteration. (ans = min(ans, swaps)).

Code –

class Solution:

    def solve(self, A, B):

        n = len(A)

        cnt = 0

        # Count elements in A less than or equal to B

        for i in A:

            if i<=B:

                cnt += 1

        temp = 0

        # Check no of elements less than or equal to B in the first window

        for i in range(cnt):

            if A[i] <= B:

                temp+=1

        ans = cnt - temp

        l = 0

        r = cnt

        while(r<n):

            if A[r] <= B:

                temp+=1

            if A[l] <= B:

                temp-=1

            ans = min(ans, cnt-temp)

            l+=1

            r+=1

        return ans

**Question 18: Minimum swaps 2**

Given an array of integers **A** of size **N** that is a permutation of **[0, 1, 2, ..., (N-1)]**. It is allowed to swap any two elements (not necessarily consecutive).

Find the minimum number of swaps required to sort the array in ascending order.

**Problem Constraints**

1 <= N <= 100000  
0 <= A[i] < N

Logic – Sort the array using the logic used in question 14. As the elements are between [0 to N-1], we can place the elements in their correct indexes by swapping them.

1. If A[i] != i, Swap A[i] and A[A[i]]
2. Else check next element.

Code –

class Solution:

    # @param A : list of integers

    # @return an integer

    def solve(self, A):

        n = len(A)

        i = 0

        cnt = 0

        while(i<n):

            if A[i] == i:

                i+=1

            else:

                cnt += 1

                A[A[i]], A[i] = A[i], A[A[i]]

        return cnt

**Question 19: Maximum Distance**

Given an array, **A** of integers of size **N**. Find the maximum value of j - i such that A[i] <= A[j].

**Problem Constraints**

1 <= N <= 105

-109 <= A[i] <= 109

Logic –

1. Create an array B of tuples containing (A[i], i).
2. Sort the array B on the basis of the first value of the tuple. ie A[i]
3. Now the 2nd condition is satisfied (A[i]<=A[j]) if we iterate from the end.
4. Iterate from the end and maintain a variable maxj = maximum value of B[i][j] from the end.
5. Answer = min (answer, B[i][j])

Code –

import sys

class Solution:

    # @param A : tuple of integers

    # @return an integer

    def maximumGap(self, A):

        n = len(A)

        B = []

        for i in range(n):

            B.append((A[i], i))

        B.sort()

        ans = 0

        maxj = -sys.maxsize - 1

        for i in range(n-1, -1, -1):

            maxj = max(maxj, B[i][1])

            ans = max(ans, maxj-B[i][1])

        return ans

**Question 20: Merge Overlapping Intervals**

Given a collection of intervals, merge all overlapping intervals.  
  
**Problem Constraints**

1 <= Total number of intervals <= 100000.

Logic –

1. Sort the array on the basis of start values of the intervals.
2. Iterate and check if the last element in answer is overlapping with current element.
3. If No – Append the element and move forward.
4. If Yes – Change the start of the last element to min(lastElement.start, currentElement.start) and end to max(lastElement.end, currentElement.end).

Code –

# Definition for an interval.

# class Interval:

#     def \_\_init\_\_(self, s=0, e=0):

#         self.start = s

#         self.end = e

import functools

class Solution:

    def mycmp(self, a, b):

        if a.start > b.start:

            return 1

        elif a.start < b.start:

            return -1

        else:

            return 0

    # @param intervals, a list of Intervals

    # @return a list of Interval

    def merge(self, intervals):

        ans = []

        # sort intervals

        intervals = sorted(intervals, key=functools.cmp\_to\_key(self.mycmp))

        ans.append(intervals[0])

        n = len(intervals)

        for i in range(1,n):

            st = intervals[i].start

            en = intervals[i].end

            # If not overlapping

            if ans[-1].end < st:

                ans.append(intervals[i])

            else:

                ans[-1].start = min(st, ans[-1].start)

                ans[-1].end = max(en, ans[-1].end)

        return ans

**Question 21: Maximum Chunks to Make Sorted**

Given an array of integers **A** of size **N** that is a permutation of **[0, 1, 2, ..., (N-1)]**, if we split the array into some number of "chunks" (partitions), and individually sort each chunk. After concatenating them in order of splitting, the result equals the sorted array.

What is the **most** number of chunks we could have made?

**Problem Constraints**

1 <= N <= 100000  
0 <= A[i] < N

Logic – Smallest leftmost chunk is the smallest index at which A[0, 1,…i] contains all elements up to i. We can check that if max of A[0,1,…i] is equal to I, only then we can take it as a separate chunk. ( Note – This is possible only because elements in array A are between 0 and N-1 )

1. Iterate the array and maintain a variable that holds maximum value from left.
2. If current index = maximum value from left, increment the answer.

Code –

class Solution:

    # @param A : list of integers

    # @return an integer

    def solve(self, A):

        cnt = 0

        maxtillNow = -1

        for i in range(len(A)):

            maxtillNow = max(maxtillNow, A[i])

            if i == maxtillNow:

                cnt+=1

        return cnt

**Question 22: Single Number**

Given an array of integers **A**, every element appears twice except for one. Find that integer that occurs once.  
  
**NOTE:** Your algorithm should have a linear runtime complexity. Could you implement it without using extra memory?

**Problem Constraints**

* 2 <= |A| <= 2000000
* 0 <= A[i] <= INTMAX

Logic – Take XOR of all elements in the array. XOR of same number becomes zero and the single number is the number which will be left in the end.

Code –

class Solution:

    # @param A : tuple of integers

    # @return an integer

    def singleNumber(self, A):

        ans = 0

        for i in A:

            ans ^= i

        return ans

**Question 23: Single Number 2**

Given an array of integers, every element appears thrice except for one, which occurs once.  
  
Find that element that does not appear thrice.  
  
**NOTE:** Your algorithm should have a linear runtime complexity.  
  
Could you implement it without using extra memory?

**Problem Constraints**

* 2 <= A <= 5\*106
* 0 <= A <= INTMAX

Logic – Count the no of set bits at the ith bit (i=0 to 31) and take the modulo with 3.

* 1. Check all 32 bits of all the numbers in the array A (I = 0 to 31).
  2. Count the number of elements in the array which have the ith bit set (cnt = A[j] & (1<<i) != 0)
  3. Take modulo (%) of the count (cnt) with 3.
  4. If the cnt % 3 == 0 means that the ith bit of the single number is not set.
  5. If the cnt % 3 != 0 means that the ith bit of the single number is set. Therefore, in this case set the ith bit of the ans.

Note – If some numbers appear x times we can take the modulo of cnt with x and do the other steps as it is.

Code –

class Solution:

    # @param A : tuple of integers

    # @return an integer

    def singleNumber(self, A):

        ans = 0

        for i in range(32):

            cnt = 0

            # Check if ith bit is set or not

            # incrememt cnt if it is set

            for j in range(len(A)):

                # bit is set

                if (A[j] & (1<<i) != 0):

                    cnt+=1

            if (cnt % 3 != 0):

                ans |= (1<<i)

        return ans

**Question 24: Number of 1 Bits**

Write a function that takes an integer and returns the number of 1 bits it has.  
  
**Problem Constraints**

1 <= A <= 109

Logic –

1. When a AND operation is performed between A and A-1, the set bit from the left or the most significant bit (MSB) is toggled to 0 (1 -> 0 and 0 -> 0).
2. So, we perform this operation till the number becomes equal to 0.

Code –

class Solution:

    # @param A : integer

    # @return an integer

    def numSetBits(self, A):

        M = A

        cnt = 0

        while(M > 0):

            M = M&M-1

            cnt+=1

        return cnt

**Question 25: Add Binary Strings**

Given two binary strings, return their sum (also a binary string).

**Example:**

a = "100"

b = "11"

Return a + b = "111".

Logic –

1. Make the Strings Equal, by adding (|len(A)-len(B)|) zeros to right of the smaller number.
2. Maintain a carry variable equal to zero. Consider 2 conditions: If carry = 0 and carry =1
3. Create the conditions by applying the basic rules of binary addition.
4. 0+0 =0
5. 1+0 = 1
6. 0+1 = 1
7. 1+1 = 0 and carry = 1
8. 1+1+1 = 1 and carry = 1

Code –

class Solution:

    def addBinary(self, A, B):

        n = len(A)

        m = len(B)

        k = m

        if (n>m):

            temp = '0'\*(n-m)

            B = temp + B

            k = n

        elif (m>n):

            temp = '0'\*(m-n)

            A = temp + A

        carry = 0

        ans = ''

        for i in range(k-1, -1, -1):

            if carry == 0:

                if A[i] == '0' and B[i] == '0':

                    carry = 0

                    ans = '0' + ans

                elif (A[i] == '1' and B[i] == '0') or (A[i] == '0' and B[i] == '1'):

                    carry = 0

                    ans = '1' + ans

                elif (A[i] == '1' and B[i] == '1'):

                    carry = 1

                    ans = '0' + ans

            elif carry == 1:

                if A[i] == '0' and B[i] == '0':

                    carry = 0

                    ans = '1' + ans

                elif (A[i] == '1' and B[i] == '0') or (A[i] == '0' and B[i] == '1'):

                    carry = 1

                    ans = '0' + ans

                else:

                    carry = 1

                    ans = '1' + ans

        if carry == 1:

            ans = '1' + ans

        return ans

**Question 26: Single Number 3**

Given an array of positive integers **A**, two integers appear only once, and all the other integers appear twice.  
Find the**two integers that appear only once**.

**Problem Constraints**

2 <= |A| <= 100000  
1 <= A[i] <= 109

Logic –

1. Find the XOR of all the elements in the array. Now we have to separate 2 numbers from this answer.
2. To find the 2 numbers find the position of the first set bit from the right. Then left shift 1 to that position. (1 << idx)
3. Check if A[i] & (1<<idx) == 0.
4. If above condition is True, XOR it with a variable x1.
5. If it is False, XOR it with a variable x2.
6. x1 and x2 are the required numbers separated from the XOR of all elements.

Code –

class Solution:

    # @param A : list of integers

    # @return a list of integers

    def solve(self, A):

        ans = 0

        for i in A:

            ans ^= i

        # Find first set bit from right

        idx = 0

        M = ans

        while(M>0):

            if M%2==1:

                break

            M/=2

            idx+=1

        temp = 1<<idx

        x1 = 0

        x2 = 0

        for i in A:

            if i & temp == 0:

                x1 ^= i

            else:

                x2 ^= i

        if x1>x2:

            return [x2, x1]

        return [x1, x2]

**Question 27 Count Total Set Bits**

Given a positive integer **A**, the task is to count the total number of set bits in the binary representation of all the numbers from **1** to **A**.

Return the count modulo **109 + 7**.

**Problem Constraints**

1 <= A <= 109

Logic –

1. Find the position of most significant bit of A (idx).
2. Number of 1s in one iteration = (2^(idx-1) \* idx) + (A-temp+1). [Try this by an example (A=11)].
3. Now reduce A to A – (2^idx)
4. Repeat the same process again till A becomes equal to 0.

Code –

class Solution:

    # @param A : integer

    # @return an integer

# Find the postion of most significant bit

    def findRightBit(self, A):

        idx = 0

        while(A>0):

            A>>=1

            idx +=1

        idx-=1

        return idx

    def solve(self, M):

        # if M<=0:

        #     return 0

        # # Using recursion

        # idx = self.findRightBit(M)

        # # Cal 2^(idx-1)\*idx + (n-(2^idx)+1)

        # temp = 1 << (idx)

        # ans = (temp\*idx//2) + (M-temp+1) + self.solve(M-temp)

        # return ans%1000000007

        # Without Recursion

        ans = 0

        A = M

        while(A>0):

            # Find idx

            idx = self.findRightBit(A)

            temp = 1 << (idx)

            ans += (temp\*idx//2) + (A-temp+1)

            A = A - (temp)

        return ans%1000000007

**Question 28 Strange Equality**

Given an integer **A**.  
Two numbers, X and Y, are defined as follows:

* X is the **greatest number smaller** than A such that the XOR sum of X and A is the same as the sum of X and A.
* Y is the **smallest number greater** than A, such that the XOR sum of Y and A is the same as the sum of Y and A.

Find and **return** the XOR of X and Y.

**NOTE 1:** XOR of X and Y is defined as X ^ Y where '^' is the BITWISE XOR operator.

**NOTE 2:** Your code will be run against a maximum of 100000 Test Cases.  
  
**Problem Constraints**

1 <= A <= 109

Logic – A + B = (A ^ B) + 2 \* (A & B)

1. Following the above equation, if xor sum and the sum of 2 numbers are equal, their bitwise AND should be zero.
2. Therefore, in the number P such that P & Q is 0, all the set bits of number Q must be unset in the number P since 1 & 0 = 0.
3. The unset bits of A can have any orientation in X, that is, they can either be 0 or be 1.  
   So to find the smallest number (Y) greater than A, the answer is the next power of 2 greater than A.
4. And to find the greatest number(X) smaller than A, set all the unset bits of A to 1.

Code –

class Solution:

    # @param A : integer

    # @return an integer

    # A+B = (A^B) + 2(A&B)

    def solve(self, temp):

        cnt = 1

        A = temp

        while(A>0):

            A//=2

            cnt+=1

        return ((1<<cnt)-1) ^ temp

**Question 29 Divide Integers**

Divide two integers without using multiplication, division and mod operator.

Return the floor of the result of the division.

Also, consider if there can be overflow cases i.e output is greater than INT\_MAX, return INT\_MAX.

**NOTE: INT\_MAX = 2^31 - 1**  
  
**Problem Constraints**

-231 <= A, B <= 231-1

B != 0

Logic – Determine the most significant bit in the answer. Iterate on the bit position ‘i’ from 31 to 1 and find the first bit for which divisor « i is less than the dividend.

1. Iterate each bit from 1st to 31st bit.
2. If B\*(2^i) = B << i <= A then set the ith bit of the answer and decrease A to A-(B << i).
3. Repeat the process for all the bits.

Code –

import sys

class Solution:

    # @param A : integer

    # @param B : integer

    # @return an integer

    def divide(self, A, B):

        ans = 0

        if B>0:

            tempB = B

        else:

            tempB = ~B + 1

        if A > 0:

            temp = A

        else:

            temp = ~A + 1

        for i in range(31, -1, -1):

            temp2 = tempB << i

            if temp2 <= temp:

                ans |= (1 << i)

                temp -= temp2

        if (A > 0 and B>0) or (A<0 and B<0):

            maxInt = (1<<31) - 1

            if ans < maxInt:

                return ans

            else:

                return maxInt

        else:

            return -1\*ans

**Question 30 Smallest XOR**

Given two integers **A** and **B**, find a number **X** such that **A** xor **X** is minimum possible, and the number of set bits in **X** equals **B**.

**Problem Constraints**

0 <= A <= 109  
0 <= B <= 30

Logic – Count set bits in A and consider 3 possible cases. noOfBitsA > B, noOfBitsA < B, noOfBitsA = B.

1. Calculate number of set bits in A (noOfBitsA).
2. If set bits in A is greater than B then A has (noOfBitsA – B) extra set bits in it. Therefore, we have to set them.
3. If set bits in A is less than B then we need to set (B - noOfBitsA) bits from right.
4. If noOfBitsA = B, then A is the ans as A^A = 0, and no of set bits required = B.

Code –

class Solution:

    # @param A : integer

    # @param B : integer

    # @return an integer

    def cal(self, A):

        cnt = 0

        while(A>0):

            A = A&A-1

            cnt+=1

        return cnt

    def solve(self, A, B):

        # Cal no of bits in A

        noOfBitsA = self.cal(A)

        if noOfBitsA > B:

            temp = noOfBitsA-B

            ans = A

            for \_ in range(temp):

                ans = ans&ans-1

            return ans

        elif B > noOfBitsA:

            i = 0

            temp = B - noOfBitsA

            ans = A

            while(temp>0):

                if A%2==0:

                    temp-=1

                    # set ith bit

                    ans |= (1<<i)

                A//=2

                i+=1

            return ans

        else:

            return A

**Question 31 Reverse Bits**

Reverse the bits of an **32** bit unsigned integer A.

**Problem Constraints**

0 <= A <= 232

Logic – Get the bits of A from right and left shift them i times (i goes from 31 to 0).

1. The remainder of A with 2 will give the right most bit. Shift this i times ( i = 31 to 0 ).
2. Perform the OR operation with answer, to set/reset the ith bit depending upon the value of A%2. If A%2 == 0 reset (Make it 0) the ith bit, else set it.
3. Decrement i and right shift A to get the next right most bit.

Code –

class Solution:

    # @param A : unsigned integer

    # @return an unsigned integer

    def reverse(self, A):

        temp = A

        i = 31

        ans = 0

        while(temp>0):

            ans |= (temp%2) << i

            i-=1

            temp>>=1

        return ans

**Question 32 Rearrange Array**

Rearrange a given array so that Arr[i] becomes Arr[Arr[i]] with O(1) extra space.

**Example:**

Input : [1, 0]

Return : [0, 1]

Lets say N = size of the array. Then, following holds true :

* **All elements in the array are in the range [0, N-1]**
* **N \* N does not overflow for a signed integer**

Logic-Store old and new values in the form of a+bn, such that a+bn % n = a and a+bn //n = b

* 1. Convert every element in A to a+bn = A[i] + (A[A[i]]%n)\*n, where a = A[i] and b = A[A[i]]%n = old value of A[A[i]].
  2. To swap we need to return the new value of the every element, so divide each element with n.

Note – We can also store the old value as // and new value as %. It is also a valid way of representation.

Code –

class Solution:

    # @param A : list of integers

    # Modify the array A which is passed by reference.

    # You do not need to return anything in this case.

    # % gives old value

    # / gives new value

    def arrange(self, A):

        n = len(A)

        # Store a+bn in each element of A

        # a -> Old value

        # b -> New value

        for i in range(n):

            A[i] = A[i] + (A[A[i]]%n)\*n

        for i in range(n):

            A[i] = A[i]//n

        return A

**Question 33 A, B and Modulo**

Given two integers **A** and **B**, find the **greatest possible positive integer M**, such that A % M = B % M.

**Problem Constraints**

1 <= A, B <= 109  
A != B

Logic –

1. We will try to find the pattern by fixing the value of A and taking different B values.
2. Suppose A = 5 and B = 6. The maximum possible value of M is 1.  
   Suppose A = 5 and B = 7. The maximum possible value of M is 2.  
   Suppose A = 5 and B = 8. The maximum possible value of M is 3.  
   Suppose A = 5 and B = 9. The maximum possible value of M is 4.  
   … so on.
3. It is easy to observe that the maximum value equals the absolute difference between A and B.

Code –

class Solution:

    # @param A : integer

    # @param B : integer

    # @return an integer

    def solve(self, A, B):

        if B>A:

            return B-A

        return A-B

**Question 34 Implement Power Function**

Implement **pow(A, B) % C**.

In other words, given **A**, **B** and **C**, **Find (AB % C).**

**Note:** The remainders on division cannot be negative. In other words, make sure the answer you return is non-negative.  
  
**Problem Constraints**

-109 <= A <= 109  
0 <= B <= 109  
1 <= C <= 109

Logic –

1. If B is odd the ans = ans \* A and reduce B by 1.
2. If B is even ans = ans \* A \* A and reduce B by B//2.

Note – The code below is bit different but follows the same principle. We can also apply recursion here if asked.

Code –

class Solution:

    # @param A : integer

    # @param B : integer

    # @param C : integer

    # @return an integer

    def pow(self, A, B, C):

        # Using recursion

        if A == 0:

            return 0

        if B==0:

            return 1

        x = self.pow(A, B//2, C) % C

        if B&1==0:

            return (x\*x) % C

        else:

            return (A\*x\*x) % C

        # Without recursion

        # ans = 1

        # while(B>0):

        #     if B&1:

        #         ans = (ans\*A) % C

        #     A = (A\*A) % C

        #     B >>= 1

        # return ans % C

**Question 35 Prime Modulo inverse**

Given two integers **A** and **B**. Find the value of **A-1 mod B** where **B** is a prime number and **gcd(A, B) = 1**.

**A-1 mod B** is also known as modular multiplicative inverse of **A** under modulo **B**.  
  
**Problem Constraints**

1 <= A <= 109  
1<= B <= 109  
B is a prime number

Logic – Fermat’s theorem: If p is prime number and if a is any integer, then Ap-1 congruent to 1 % p.

1. Using the Fermat’s theorem A-1 % B = AB-2 % B

Code –

class Solution:

    # @param A : integer

    # @param B : integer

    # @return an integer

    def findPow(self, A, B, C):

        ans = 1

        while(B>0):

            if B&1:

                ans = (ans \* A) % C

            B>>=1

            A = (A\*A) %C

        return ans

    def solve(self, A, B):

        return self.findPow(A, B-2, B)

**Question 36 Pair Sum Divisible by M**

Given an array of integers **A** and an integer **B**, find and return the number of pairs in **A** whose sum is divisible by **B**.

Since the answer may be large, return the answer modulo (109 + 7).  
  
**Problem Constraints**

1 <= length of the array <= 100000  
1 <= A[i] <= 109  
1 <= B <= 106

Logic – Find modulus of the array, and find number of pairs where the sum is 0 or the sum is B. Follow below steps to find the number of pairs.

1. Create a hash map which holds the frequency of A[i]%B with all values from 0 to B-1 = 0.
2. If frequency of A[i]%B = 0 is n, then number of pairs = n \* n-1 //2.
3. Now, the number of pairs with i and B-1-i can be found out by freq[i] \* freq[B-1-i]
4. If B is even then the mid element from 1 to B-1 will form n \* n-1 // 2 pairs. Where n = frequency of mid element.

Code –

class Solution:

    # @param A : list of integers

    # @param B : integer

    # @return an integer

    def solve(self, A, B):

        freq = {}

        for i in range(B):

            freq[i] = 0

        for i in range(len(A)):

            freq[A[i]%B] += 1

        ans = 0

        ans += ((freq[0] \* (freq[0]-1)) // 2)

        i = 1

        j = B-1

        while(j>i):

            ans += (freq[i] \* freq[j])

            j-=1

            i+=1

        if j==i:

            ans += ((freq[i] \* (freq[i]-1)) // 2)

        return ans % 1000000007

**Question 37 Count of Divisors**

Given an array of integers **A**, find and return the **count of divisors** of each element of the array.

**NOTE:** The order of the resultant array should be the same as the input array.

**Problem Constraints**

1 <= length of the array <= 100000  
1 <= A[i] <= 106

Logic – Sieve of Eratosthenes method

1. Create a sieve of Eratosthenes from 1 to max(A) which holds the divisors from 1 to max(A).
2. Steps to create Seive.
3. Iterate i from 2 to maxA+1
4. Start j = i+i
5. Start adding 1 to the spf[j], and increment j by I (j+=i)
6. For more details refer gfg.

Code –

class Solution:

    # @param A : list of integers

    # @return a list of integers

    def solve(self, A):

        n = len(A)

        maxA = max(A)

        spf = [2 for i in range(maxA+2)]

        spf[1] = 1

        spf[0] = 0

        for i in range(2, maxA+1):

            j = i+i

            while(j < maxA+1):

                spf[j] += 1

                j+=i

        # print(spf)

        ans = []

        for i in A:

            ans.append(spf[i])

        return ans

**Question 38 Very Large Power**

Given two Integers **A**, **B**. You have to calculate (A ^ (B!)) % (1e9 + 7).

"^" means power,

"%" means "mod", and

"!" means factorial.

**Problem Constraints**

1 <= A, B <= 5e5

Logic – Fermat’s theorem: If p is prime number and if a is any integer, then Ap-1 congruent to 1 % p.

1. AB! = AB!-(p-1)-(p-1)-(p-1)…. = AB!%(p-1) , as Ap-1 makes no difference to the answer B! can be replaced by B! % (P-1).
2. So the final answer can be reduced = AB!%(p-1) % p.

Code –

class Solution:

    def fact(self, B, P):

        # returns B! % (P)

        ans = 1

        i = 1

        while(i<=B):

            ans = ((ans%P) \* (i%P))

            i+=1

        return ans % P

    def findPow(self, A, B, C):

        # returns A^B % C

        ans = 1

        while(B>0):

            if (B&1):

                ans = (ans%C) \* (A%C)

            A = (A%C) \* (A%C)

            B>>=1

        return ans % C

    def solve(self, A, B):

        P = 1000000007

        # X = B! % (P-1)

        X = self.fact(B, P-1)

        # ans = A^X % P

        ans = self.findPow(A, X, P)

        return ans

**Question 39 Delete One**

Given an integer array **A** of size **N**. You have to delete **one** element such that the GCD**(Greatest common divisor)** of the remaining array is maximum.  
  
Find the **maximum** value of GCD.

**Problem Constraints**

2 <= N <= 105  
1 <= A[i] <= 109

Logic –

1. Create a suffix array that will store gcd of numbers from the end of the array.
2. Iterate the array and maintain a variable which will store the gcd of all numbers from 1st element to ith element.
3. Find the GCD of current GCD and GCD of the elements from nth element to (current element + 1)th and find the maximum GCD in every iteration.

Note – Find the GCD of the elements from nth element to (current element + 1)th using GCD suffix array.

Code –

class Solution:

    # @param A : list of integers

    # @return an integer

    def gcd(self, a,b):

        while(b):

            a, b = b, a%b

        return abs(a)

    def solve(self, A):

        n = len(A)

        suffGCD = [0 for i in range(n)]

        suffGCD[n-1] = A[n-1]

        for i in range(n-2, -1, -1):

            suffGCD[i] = self.gcd(suffGCD[i+1], A[i])

        temp = 0

        ans = 0

        for i in range(n-1):

            ans = max(ans, self.gcd(temp, suffGCD[i+1]))

            temp = self.gcd(temp, A[i])

        ans = max(ans, self.gcd(temp, 0))

        return ans

**Question 40 Enumerating GCD**

You are given a number **A** and a number **B**. Greatest Common Divisor (GCD) of all numbers between **A** and **B** inclusive is taken (GCD**(A, A+1, A+2 ... B)**).

As this problem looks a bit easy, it is given that numbers **A** and **B** can be in the range of **10100**.

You have to return the value of GCD found.

The greatest common divisor of 2 numbers, A and B, is the largest number, D that divides both A and B perfectly.

**Problem Constraints**

1 <= A <= B <= 10100

Logic – GCD of A and A+1 is always 1.

1. If A=B then answer = A = B, because GCD of 2 same numbers is the number itself.
2. GCD of A, A+1 = 1 so the total GCD has to be 1.

Proof: We know that if a number D divides both A and B, it will also divide abs(A-B)

Now, if B = A+1, let's assume D to be a number that divides both A and B.

D should also divide abs(A-B) = abs(A - (A+1)) = 1

But the only number to perfectly divide 1 is 1 itself. Hence D has to be 1.

Code –

class Solution:

    # @param A : string

    # @param B : string

    # @return a strings

    def solve(self, A, B):

        if A==B:

            return A

        return "1"

**Question 41 Greatest Common Divisor**

Given 2 non-negative integers A and B, find gcd(A, B)

GCD of 2 integers A and B is defined as the greatest integer 'g' such that 'g' is a divisor of both A and B. Both A and B fit in a 32 bit signed integer.

**Note:**DO NOT USE LIBRARY FUNCTIONS.

**Problem Constraints**

0 <= A, B <= 109

Logic – GCD(a,b) = GCD(b,a%b)

1. Keep swapping a and b with b and a%b till b becomes 0. Then a is the answer.

Code –

class Solution:

    # @param A : integer

    # @param B : integer

    # @return an integer

    def gcd(self, A, B):

        while(B):

            A, B = B, A%B

        return A

**Question 42 Pubg**

There are **N** players, each with strength **A[i]**. when player **i** attack player **j**, player **j** strength reduces to **max(0, A[j]-A[i])**. When a player's strength reaches **zero**, it loses the game, and the game continues in the same manner among other players until only **1** survivor remains.  
  
Can you tell the **minimum health** last surviving person can have?  
  
**Problem Constraints**

* 1 <= N <= 100000
* 1 <= A[i] <= 1000000

Logic – Let’s consider if there were only 2 people with strength A and B (A<=B). then A would attack B, leading to A, B-A.  
It would continue until it gets smaller than A or A, B%A. Then the process would repeat as A%(B%A), B%A, and so on…Therefore, find GCD of all numbers in the array.

Code –

class Solution:

    # @param A : list of integers

    # @return an integer

    def gcd(self, a, b):

        while(b>0):

            a, b = b, a%b

        return a

    def solve(self, A):

        ans = 0

        for i in A:

            ans = self.gcd(i, ans)

        return ans

**Question 43 Largest Coprime Divisor**

You are given two positive numbers **A** and **B** . You need to find the maximum valued integer **X** such that:

 **X** divides **A** i.e. A % X = 0

 **X** and **B** are co-prime i.e. gcd(X, B) = 1  
  
**Problem Constraints**

1 <= A, B <= 109

Logic – Remove all the common prime factors between A and B to find the answer X.

We can try to remove the common factors of **A** and **B** from **A** by finding the greatest common divisor (gcd) of **A** and **B** and dividing **A** with that gcd.

* Mathematically, **A** = **A** / gcd(**A**, **B**) —— *STEP1*

Now, we repeat *STEP1* till we get gcd(**A**, **B**) = 1.  
Atlast, we return **X** = **A**

*How does this work ?*

On doing prime factorization of **A** and **B**, we get :

* **A** = *p1x1* **.** *p2x2* **.** … **.** *pkxk*
* **B** = *q1y1* **.** *q2y2* **.** … **.** *qlyl*

Where pi ; i = 1, 2, …, k are prime factors of **A** and xi ; i = 1, 2, …, k are their respective powers  
Similarly, qj ; i = 1, 2, …, l are prime factors of **B** and yi ; j = 1, 2, …, l are their respective powers

Let ri ; i = 1, 2, …, m be the common factors of **A** and **B**. By repeating *STEP1*, we are reducing the respective powers of ri by at least one each time (Proof of this is left to the reader). Therefore, we reach a point where powers of ri become zero, and the product of the rest prime-powers in **A** form the largest divisor of **A** that is co-prime with **B**.

Code –

class Solution:

    # @param A : integer

    # @param B : integer

    # @return an integer

    def gcd(self, A, B):

        while(B):

            A, B = B, A%B

        return A

    def cpFact(self, A, B):

        gcd = self.gcd(A, B)

        while(gcd != 1):

            A = A//gcd

            gcd = self.gcd(A, B)

        return A

**Question 44 Divisor Game**

Scooby has 3 three integers **A, B, and C**.

Scooby calls a positive integer special if it is divisible by B and it is divisible by **C**. You need to tell the number of special integers less than or equal to **A**.

**Problem Constraints**

1 <= A, B, C <= 109

Logic –

1. After taking some iterations, it can be found that number of integers divisible by B and C and less than A = A/lcm(B,C).
2. Because the lcm is the least number divisible by B and C. We need to count the multiples of this lcm less than A, which can be found by A/lcm(B,C)

Code –

class Solution:

    # @param A : integer

    # @param B : integer

    # @param C : integer

    # @return an integer

    def gcd(self, A,B):

        while(B!=0):

            A,B = B,A%B

        return A

    def solve(self, A, B, C):

        lcm = B \* C // self.gcd(B,C)

        return (A//lcm)

        # for i in range(A, -1, -1):

        #     if (i%lcm==0):

        #         if i==0:

        #             return 0

        #         return (A//i)

**Question 45 All GCD Pair**

Given an array of integers **A** of size **N** containing GCD of every possible pair of elements of another array.

Find and return the original numbers used to calculate the GCD array in any order. For example, if original numbers are **{2, 8, 10}** then the given array will be **{2, 2, 2, 2, 8, 2, 2, 2, 10}.**  
  
**Problem Constraints**

1 <= N <= 10000

1 <= A[i] <= 109

Logic –

1. Sort the array in decreasing order.
2. Highest element will always be one of the original numbers. Keep that number and remove it from the array.
3. Compute GCD of the element taken in the previous step with the current element starting from the greatest and discard the GCD value from the given array.

Code –

class Solution:

    # @param A : list of integers

    # @return a list of integers

    def gcd(self, A, B):

        while(B):

            A, B = B, A%B

        return A

    def solve(self, A):

        ans = {}

        B = []

        A.sort(reverse = True)

        for i in range(len(A)):

        # Remove the elements formed by taking GCD of 2 original elements from the hashmap

            if A[i] in ans and ans[A[i]] > 0:

                ans[A[i]] -= 1

            else:

                # Take GCD of current element and other elements in the B

                for j in range(len(B)):

                    temp = self.gcd(A[i], B[j])

                    if temp not in ans:

                        ans[temp] = 0

# if A and B are 2 numbers then gcd(A,B) = gcd(B,A), and both should be considered

                    ans[temp] += 2

                # Add the current element (original element) to array B

                B.append(A[i])

        return B

**Question 46 Count of Divisors**

Given an array of integers **A**, find and return the **count of divisors** of each element of the array.

**NOTE:** The order of the resultant array should be the same as the input array.

**Problem Constraints**

1 <= length of the array <= 100000  
1 <= A[i] <= 106

Logic – Count the number of prime factors the number has. Let say A = p1x1 p2x2 p3x3 p4x4, then number of divisors = (x1+1) \* (x2+1) \* (x3+1) \* (x4+1).

1. To find the number of prime factors first create a sieve which will hold the smallest prime factor of a number.
2. Using this we can find number of prime factors in the number. This can be done by dividing the number by each prime factor till the number of that prime factors in that number becomes zero (ie Remainder of the number with that prime factor becomes non-zero).

Code –

class Solution:

    # @param A : list of integers

    # @return a list of integers

    # Approach 1 - Prime factorization method

    def count(self, spf, N):

        ans = 1

        temp = N

        while(N>1):

            x = spf[N]

            cnt = 1

            while(N%x == 0):

                N = N//x

                cnt+=1

            ans = ans \* (cnt)

        return ans

    def solve(self, A):

        n = len(A)

        maxA = max(A)

        spf = [i for i in range(maxA+1)]

        i = 2

        while(i\*i <= maxA):

            if spf[i] == i:

                j = i\*i

                while(j<=maxA):

                    if spf[j] == j:

                        spf[j] = i

                    j+=i

            i+=1

        ans = []

        for i in range(n):

            ans.append(self.count(spf, A[i]))

        return ans

**Question 47 Prime Sum**

Given an even number **A** ( greater than 2 ), return two prime numbers whose sum will be equal to the given number.

If there is more than one solution possible, return the lexicographically smaller solution.

If [a, b] is one solution with a <= b, and [c,d] is another solution with c <= d, then

[a, b] < [c, d], If a < c OR a==c AND b < d.

**NOTE:** A solution will always exist. Read Goldbach's conjecture.  
  
**Problem Constraints**

4 <= A <= 2\*107

Logic –

1. Create a sieve and find iterate from the end of the sieve and check if sieve[i] and sieve[A-i] = 0, as shown in the code.

Code –

class Solution:

    # @param A : integer

    # @return a list of integers

    def primesum(self, A):

        seive = [0 for i in range(A+1)]

        i = 2

        while(i\*i<A):

            if (seive[i] == 0):

                j = i\*i

                while(j<=A):

                    seive[j] = 1

                    j+=i

            i+=1

        for i in range(A-2, -1, -1):

            if seive[i] == 0 and seive[A-i]==0:

                return ([A-i, i])

**Question 48 Lucky Numbers**

A **lucky number** is a number that has exactly **2 distinct** prime divisors.

You are given a number **A,** and you need to determine the **count** of lucky numbers between the range 1 to A (both inclusive).  
  
**Problem Constraints**

1 <= A <= 50000

Logic – Create a sieve of prime factors and find the numbers with 2 prime factors.

Code –

class Solution:

    # @param A : integer

    # @return an integer

    def solve(self, A):

        seive = [0 for i in range(A+1)]

        i = 2

        while(i<=A):

            if (seive[i] == 0):

                j = i+i

                while(j<=A):

                    seive[j] += 1

                    j+=i

            i+=1

        # print(seive)

        ans = 0

        for i in range(1, A+1):

            if seive[i] == 2:

                ans += 1

        return ans

**Question 49 Prime Subsequences**

Given an array **A** having **N** positive numbers. You have to find the number of **Prime subsequences** of **A**.

A Prime subsequence is one that has **only prime** numbers, for example [2, 3], [5] are the Prime subsequences where [2, 4] and [1, 2, 3, 4] are not.

Logic – Find the number of prime numbers in the array (x), then 2^x -1 is the answer.

Code –

class Solution:

    # @param A : list of integers

    # @return an integer

    def isPrime(self, N):

        if N == 1:

            return False

        i = 2

        while(i\*i<=N):

            if (N%i == 0):

                return False

            i+=1

        return True

    def solve(self, A):

        ans = 1

        for i in range(len(A)):

            if  self.isPrime(A[i]):

                ans = ans \* 2

        return (ans - 1) % 1000000007

**Question 50 Number of Open Doors**

Given an integer **A,** which denotes the number of doors in a row numbered 1 to A. All the doors are closed initially.

A person moves to and fro, changing the states of the doors as follows: the person opens a door that is already closed and closes a door that is already opened.

In the first go, he/she alters the states of doors numbered 1, 2, 3, … , A.  
In the second go, he/she alters the states of doors numbered 2, 4, 6 ….  
In the third go, he/she alters the states of doors numbered 3, 6, 9 …  
This continues till the A'th go in, which you alter the state of the door numbered A.

Find and return the **number of open doors** at the end of the procedure.

**Problem Constraints**

1 <= A <= 109

Logic –

1. The perfect squares have exactly odd number of divisors, therefore all perfect squares would be open at the end.
2. So, we have to return number of perfect squares from 1 to A. This is equal to floor value of square root of A.

Code –

import math

class Solution:

    # @param A : integer

    # @return an integer

    def solve(self, A):

        return (math.floor(math.sqrt(A)))

**Question 51 Prime Addition**

You are given an even number N and you need to represent the given number as the sum of primes. The prime numbers do not necessarily have to be distinct. It is guaranteed that at least one possible solution exists.

You need to determine the minimum number of prime numbers needed to represent the given number.

**Input**

The first argument contains an integer N,the number you need to represent (3<=N<=10^9).

**Output**

Return an integer which is the minimum number of prime numbers needed to represent the given number N.

Logic –

Goldbach's conjecture states that every even integer greater than 2 can be expressed as the sum of two primes.

Bonus : This conjecture is not proven yet but verified uptil 4\*10^18 integers.

Code –

class Solution:

    # @param A : integer

    # @return an integer

    def solve(self, A):

        if A==2:

            return 1

        return 2

**Question 52 Distinct Primes**

You have given an array A having N integers. Let say G is the product of all elements of A.

You have to find the number of distinct prime divisors of G.

**Input Format**

The first argument given is an Array A, having N integers.

**Output Format**

Return an Integer, i.e number of distinct prime divisors of G.

Logic – Similar to count of divisors.

1. Create a sieve containing smallest prime factor of numbers from 1 to maximum element in given array A.
2. Find prime factors for each number in the given array and add it to hash set. Return the length of the hash map.
3. Prime factors can be found by dividing the smallest prime factor till the remainder of the number with that prime factor becomes zero. After it becomes zero, select the next smallest prime factor.

Code –

class Solution:

    # @param A : list of integers

    # @return an integer

    def solve(self, A):

        n = max(A)

        spf = [0 for i in range(n+1)]

        spf[0] = spf[1] = 1

        i = 2

        while(i<=n):

            j = i

            while(j<=n):

                if spf[j] == 0:

                    spf[j] = i

                j+=i

            i+=1

        # print(spf)

        lst = {}

        for N in A:

            while(N>1):

                x = spf[N]

                while(N%x == 0):

                    N = N//x

                lst[x] = 1

        return len(lst)

**Question 53 Sorted Permutation Rank**

Given a string **A**. Find the rank of the string amongst its permutations **sorted lexicographically**.  
Assume that no characters are repeated.

**Note:**The answer might not fit in an integer, so return your answer % 1000003  
  
**Problem Constraints**

1 <= **|A|** <= 1000

Logic –

1. As no characters are repeated the maximum size of array is 26.
2. Iterate the array, check the elements to the right which are smaller than the current element.
3. The words formed by these characters will come before the current character in the terms of rank. So we add all the possible permutations to the rank.
4. We repeat this for the entire array.

Code -

class Solution:

    # @param A : string

    # @return an integer

    def fact(self, A):

        ans = 1

        while(A>1):

            ans = ans \* A

            A-=1

        return ans

    def findRank(self, A):

        rank = 1

        for i in range(len(A)):

            temp = ord(A[i])

            count = 0

            for j in range(i+1, len(A)):

                if ord(A[j]) < temp:

                    count+=1

            rank += count \* self.fact(len(A)-1-i)

        return rank % 1000003

**Question 54 Sorted Permutation Rank with Repeats**

Given a string **A**, find the rank of the string amongst its permutations sorted lexicographically. Note that the characters might be repeated. If the characters are repeated, we need to look at the rank in unique permutations. Look at the example for more details.

**NOTE:**

* The answer might not fit in an integer, so return your answer % 1000003 where 1000003 is a prime number.
* String A can consist of both lowercase and uppercase letters. Characters with lesser ASCII values are considered smaller, i.e., 'a' > 'Z'.

**Problem Constraints**

1 <= len(A) <= 1000000

Logic –

1. Create a hash map containing the character as key and its frequency as the value.
2. Iterate the array and count the number of smaller elements on the right by using the hash map.
3. The number of permutations with a character C as the first character = number of permutations possible with remaining N-1 character = (N-1)! / (p1! \* p2! \* p3! ...) where p1, p2, p3 are the number of occurrences of repeated characters.
4. So, we need to find the number of repeated characters in the array. We can find that by iterating the hash map and calculating the product of factorial of each element.
5. After that we can remove the current element from the hash map as it is already counted.
6. Repeat it for all the elements in the array.

Code –

class Solution:

    # @param A : string

    # @return an integer

    def fact(self, A):

        ans = 1

        while(A>1):

            ans = ans \* A

            A-=1

        return ans

# rank = rank + (no of smaller elements on right \* ((curr\_len-1)!)) // (factorial of no of repeating elements)

    def findRank(self, A):

        hash = {}

        for i in A:

            if i not in hash:

                hash[i] = 1

                continue

            hash[i] += 1

        ans = 1

        curr\_len = len(A) - 1

        for i in range(len(A)):

            # Find no of smaller char on the right of i

            right\_cnt = 0

            for key, value in hash.items():

                if ord(key) < ord(A[i]):

                    right\_cnt += value

            den = 1

            # Find the denominator ie fact of repeated

            for key, value in hash.items():

                if value <= 1:

                    continue

                else:

                    den = den \* self.fact(value)

            hash[A[i]] -= 1

            ans += (self.fact(curr\_len) \* right\_cnt) // den

            curr\_len -= 1

        return ans % 1000003

**Question 55 Compute nCr % m**

Given three integers **A**, **B,** and **C**, where **A** represents **n**, **B** represents **r,** and **C** represents **m**, find and return the value of **nCr % m** where **nCr % m = (n!/((n-r)!\*r!))% m**.

**x!** means factorial of x i.e. **x! = 1 \* 2 \* 3... \* x.**

**Problem Constraints**

1 <= A \* B <= 106

1 <= B <= A

1 <= C <= 106

Logic – nCr = n-1Cr + n-1Cr-1

There are 2 ways to solve this:

1. Dynamic Programming
   1. Create a matrix (A+1, B+1) with the first column 1 and other elements 0.
   2. Iterate from rows 1 to A and columns 1 to A and add mat[i-1][j] and mat[i-1][j-1] to mat[i][j]. This way the last element will contain the value of nCr
2. Recursion
   1. The base case here is if A becomes equal to zero or A equals B, the ans = 1.
   2. If B becomes 1 return A.
   3. Then we can execute the same function for smaller inputs as shown in the code. (**return ((self.solve(A-1, B, C) % C) + ((self.solve(A-1, B-1, C)) % C )) % C**)

Code –

class Solution:

    # @param A : integer

    # @param B : integer

    # @param C : integer

    # @return an integer

    def solve(self, A, B, C):

        mat = [[0 for j in range(B+1)] for i in range(A+1)]

        for i in range(A+1):

            mat[i][0] = 1

        for i in range(1, A+1):

            for j in range(1, B+1):

                mat[i][j] = (mat[i-1][j] % C) + (mat[i-1][j-1] % C)

        # print(mat)

        return mat[A][B] % C

    # With Recursion

    # def solve(self, A, B, C):

    #     if B == 0 or A==B:

    #         return 1

    #     if B == 1:

    #         return A

    #     return ((self.solve(A-1, B, C) % C) + ((self.solve(A-1, B-1, C)) % C )) % C

**Question 56 Compute nCr % p [Solve for bigger inputs]**

Given three integers **A**, **B,** and **C**, where **A** represents **n**, **B** represents **r,** and **C** represents **p** and **p** is a prime number greater than equal to **n**, find and return the value of **nCr % p** where **nCr % p = (n! / ((n-r)! \* r!)) % p.**

x! means factorial of x i.e.**x! = 1 \* 2 \* 3... \* x.**

**NOTE:** For this problem, we are considering 1 as a prime.  
  
**Problem Constraints**

* 1 <= A <= 106
* 1 <= B <= A
* A <= C <= 109+7

Logic – Fermatts theorem

1. nCr = n! / (r! \* (n-r)!)
2. Find the maximum between n-r and r and cancel it from the numerator.
3. The factorial of the remaining term (ie n-r or r) will be the denominator.
4. Then use Fermatts theorem to find the value of (denominator)-1.
5. (denominator)-1 % p = (denominator)p-2 % pcan be found using Ax % p.
6. Then ans = ((numerator % p) \* (denominator)p-2 % p ) % p, where numerator is equal to n \* n-1 \* n-2 \* …. n-r+1 and denominator = factorial of the minimum between n-r or r

Code –

class Solution:

    # @param A : integer

    # @param B : integer

    # @param C : integer

    # @return an integer

    def findPow(self, A, B, C):

        # returns A^B % C

        ans = 1

        while(B>0):

            if B&1:

                ans = (ans % C) \* (A % C)

            A = (A % C) \* (A % C)

            B>>=1

        return ans % C

    def solve(self, A, B, C):

        if C == 1:

            return 0

        if A == 1:

            return 1

        mx = max(A - B, B)

        mn = min(A - B, B)

        denominator = 1

        numerator = 1

        for n in range(mx + 1, A + 1):

            numerator = (numerator \* n)

            if numerator >= C:

                numerator %= C

        for d in range(2, mn + 1):

            denominator = (denominator \* d)

            if denominator >= C:

                denominator %= C

        return (numerator \* self.findPow(denominator, C - 2, C)) % C

**Question 57 Tower of Hanoi**

In the classic problem of the Towers of Hanoi, you have **3** towers numbered from 1 to 3 (left to right) and **A** disks numbered from 1 to A (top to bottom) of different sizes which can slide onto any tower.  
The puzzle starts with disks sorted in ascending order of size from top to bottom (i.e., each disk sits on top of an even larger one).  
You have the following constraints:

* Only one disk can be moved at a time.
* A disk is slid off the top of one tower onto another tower.
* A disk cannot be placed on top of a smaller disk.

You have to find the solution to the **Tower of Hanoi** problem.  
You have to return a 2D array of dimensions **M x 3**, where M is the minimum number of moves needed to solve the problem.  
In each row, there should be 3 integers (disk, start, end), where:

* **disk** - number of disk being moved
* **start** - number of the tower from which the disk is being moved
* **stop** - number of the tower to which the disk is being moved

Logic –

1. To place the disk at the bottom from B to D, we will have to put all the discs above it from B to C first then we place last disc from B to D and then the remaining discs from C to D.
2. We do this using recursion. In the first step we write the base condition to exit the recursion. Then we call the same function to place A-1 discs from B to C using the disc D. Then we place the Ath disc from B to D. Then the remaining A-1 discs are placed from C to D via B.

Code –

class Solution:

    # @param A : integer

    # @return a list of list of integers

    def helper(self, A, B, D, C, temp):

        if A == 0:

            return temp

        self.helper(A-1, B, C, D, temp)

        temp.append([A, B, D])

        self.helper(A-1, C, D, B, temp)

    def towerOfHanoi(self, A):

        temp = []

        self.helper(A, 1, 3, 2, temp)

        return temp

**Question 58 Gray Code**

The gray code is a binary numeral system where two successive values differ in only one bit.

Given a non-negative integer **A** representing the total number of bits in the code, print the sequence of gray code.

A gray code sequence must begin with **0**.

**Problem Constraints**

1 <= A <= 16

Logic –

* 1. Gray Code(A) = Gray Code(A-1) with (A-1)th not set + All elements of Gray Code(A-1) with (A-1)th set.
  2. Base case is [0, 1] at A=1.

Code –

class Solution:

    # @param A : integer

    # @return a list of integers

    def grayCode(self, A):

        if A==1:

            temp = []

            temp.append(0)

            temp.append(1)

            return temp

        x = self.grayCode(A-1)

        temp = []

        for i in x:

            temp.append(i)

        for i in range(len(x)-1, -1, -1):

            temp.append(x[i] + (1<<(A-1)))

        return temp

**Question 59 Kth Symbol**

On the first row, we write a 0. Now in every subsequent row, we look at the previous row and replace each occurrence of 0 with 01, and each occurrence of 1 with 10.

Given row number **A** and index **B**, return the **Bth** indexed symbol in row **A**. (The values of **B** are 1-indexed.).

**Problem Constraints**

1 <= A <= 20

1 <= B <= 2A - 1

Logic –

* 1. If k is odd then it means that the number at that position is formed by toggling the k//2 the element of N-1th operation.
  2. We can divide the Func(N, K) into Func(N-1, K//2). If x = Func(N-1, K//2) is even then return x. If x is odd then toggle and return x.

Code –

class Solution:

    # @param A : integer

    # @param B : integer

    # @return an integer

    def helper(self, A, B):

        if A==0 or B==0:

            return 0

        x = self.helper(A-1, B//2)

        if B%2==0:

            return x

        else:

            if x==0:

                return 1

            else:

                return 0

    def solve(self, A, B):

        return(self.helper(A, B-1))

**Question 60 Merge 2 sorted array**

Given two sorted integer arrays **A** and **B**, merge B and A as one sorted array and return it as an output.

**Problem Constraints**

-1010 <= A[i], B[i] <= 1010

Logic –

* 1. Take 2 pointers i and j. Take the smallest element and increment the respective pointer till one of the array is completely checked.
  2. Append the elements of the other array.

Code –

class Solution:

    # @param A : tuple of integers

    # @param B : tuple of integers

    # @return a list of integers

    def solve(self, A, B):

        ans = []

        i = j = 0

        while(i<len(A) and j<len(B)):

            if A[i] > B[j]:

                ans.append(B[j])

                j+=1

            elif B[j] > A[i]:

                ans.append(A[i])

                i+=1

            else:

                ans.append(A[i])

                ans.append(B[j])

                i+=1

                j+=1

        if i == len(A):

            while(j<len(B)):

                ans.append(B[j])

                j+=1

        elif j == len(B):

            while(i<len(A)):

                ans.append(A[i])

                i+=1

        return ans

**Question 61 Kth Smallest Index**

Find the **Bth** smallest element in given array **A** .

**NOTE**: Users should try to solve it in less than equal to B swaps.  
  
**Problem Constraints**

1 <= |A| <= 100000

1 <= B <= min(|A|, 500)

1 <= A[i] <= 109

Logic –

1. Perform Selection sort or bubble sort for B times. Then return the B-1th element in the array.

Code –

class Solution:

    # @param A : tuple of integers

    # @param B : integer

    # @return an integer

    def kthsmallest(self, A, B):

        temp = []

        for i in A:

            temp.append(i)

        for i in range(B):

            for j in range(len(A)-1, 0, -1):

                if temp[j] < temp[j-1]:

                    temp[j], temp[j-1] = temp[j-1], temp[j]

        return temp[B-1]

**Question 62 Array with consecutive elements**

Given an array of positive integers **A**, check and return whether the array elements are consecutive or not.

**NOTE:** Try this with O(1) extra space.

**Problem Constraints**

1 <= length of the array <= 100000  
1 <= A[i] <= 10^9

Logic –

1. Sort the array and check if the consecutive elements are having a difference of 1 or not.

Code –

class Solution:

    # @param A : list of integers

    # @return an integer

    def solve(self, A):

        mn = min(A)

        mx = max(A)

        A.sort()

        temp = 0

        for i in range(1, len(A)):

            if A[i] != A[i-1]+1:

                return 0

        return 1

**Question 63 MaxMod**

Given an array **A** of size **N**, Groot wants you to pick **2** indices **i** and **j** such that

1. **1 <= i, j <= N**
2. **A[i] % A[j]** is maximum among all possible pairs of **(i, j)**.

Help Groot in finding the maximum value of **A[i] % A[j]** for some **i**, **j**.

**Problem Constraints**

1 <= N <= 100000  
0 <= A[i] <= 100000

Logic – Since we need the maximum value of A[i]%A[j], we should take A[i] and A[j] to be the last elements in the sorted array(ie the largest and the second largest value of the array)

1. We have to take the largest and second-largest distinct number of the array, which need not be the last two elements of the sorted array. There can also be duplicates. So we have to iterate the array and find max1 and max2 in the array.
2. This can be done by changing the max2 to max1 every time a element greater than A[i] is found as shown in the code.

Code –

import sys

class Solution:

    # @param A : list of integers

    # @return an integer

    def solve(self, A):

        if len(A) == 1:

            return 0

        max1 = -sys.maxsize

        max2 = -sys.maxsize

        for i in range(len(A)):

            if A[i] > max1:

                max2 = max1

                max1 = A[i]

            elif A[i] > max2 and A[i] < max1:

                max2 = A[i]

        if max2 == -sys.maxsize:

            return 0

        return max2

**Question 64 Chocolate Distribution**

Given an array **A** of **N** integers where the i-th element represent the number of chocolates in the i-th packet.

There are **B** number of students, the task is to distribute chocolate packets following below conditions:

1. Each student gets one packet.

2. The difference between the number of chocolates given to any two students is minimum.

Return the minimum difference (that can be achieved) between the student who gets minimum number of chocolates and the student who gets maximum number of chocolates.

**Problem Constraints**

0 <= N <= 10^5

1 <= A[i] <= 10^7

0 <= B <= 10^5

Logic – Sort the array. Find the minimum difference (Difference between last and first element of the subarray) between all subarrays of size B.

Code –

import sys

class Solution:

    # @param A : list of integers

    # @param B : integer

    # @return an integer

    def solve(self, A, B):

        if B==0 or len(A)==0:

            return 0

        ans = sys.maxsize

        A.sort()

        for i in range(len(A)-B+1):

            ans = min(ans, A[i+B-1]-A[i])

        return ans

**Question 65 Inversion Count in an array**

Given an array of integers **A**. If i < j and A[i] > A[j], then the pair (i, j) is called an inversion of A. Find the total number of inversions of A modulo (109 + 7).  
  
**Problem Constraints**

1 <= length of the array <= 100000

1 <= A[i] <= 10^9

Logic – Same as merge sort algorithm

1. Suppose we know the number of inversions in the left half and the right half of the array, lets call them a and b.
2. The inversions in the elements in the left half which are greater than the elements in the right half are not counted in a+b.
3. Lets assume that both left half and right half are sorted. We can just merge the two arrays and whenever we find that a[i] > a[j] (where i is the pointer of left half of the array and j is the pointer of the right half of the array) we simply increment the number of inversions.
4. The final answer will be a + b + number of inversions found during merge step.

Code –

class Solution:

    def merge(self, A, l, y, r):

        p1 = l

        p2 = y+1

        arr = []

        cnt = 0

        while(p1<=y and p2<=r):

            if A[p1] > A[p2]:

                arr.append(A[p2])

                p2+=1

                cnt+=(y-p1+1) # Count number of inversions

          elif A[p2] > A[p1]:

                arr.append(A[p1])

                p1+=1

            else:

                arr.append(A[p1])

                p1+=1

        while(p1<=y):

            arr.append(A[p1])

            p1+=1

        while(p2<=r):

            arr.append(A[p2])

            p2+=1

        for i in range(r-l+1):

            A[l+i] = arr[i]

        return cnt

    def helper(self, A, l, r):

        if l==r:

            return 0

        mid = (l+r) // 2

        a = self.helper(A, l, mid)

        b = self.helper(A, mid+1, r)

        c = self.merge(A, l, mid, r)

        return (a + b + c) % 1000000007

    def solve(self, A):

        return self.helper(A, 0, len(A)-1) % 1000000007

**Question 66 Largest Number**

Given an array **A** of non-negative integers, arrange them such that they form the largest number.

**Note:** The result may be very large, so you need to return a string instead of an integer.

**Problem Constraints**

1 <= len(A) <= 100000  
0 <= A[i] <= 2\*109

Logic – Use a modified Sort

1. In the sort function, at every step 2 elements of the array are compared and according to it the sorting function sorts the array. This condition can be modified in the sorting function.
2. Convert the 2 integers to be compared at every step of the sorting process into strings.
3. Then compare the addition of the strings (str(a)+str(b)) with the reverse addition of the strings (str(b)+str(a)).

Code –

import functools

class Solution:

    # @param A : tuple of integers

    # @return a strings

    def compare(self, a, b):

        temp1 = str(a) + str(b)

        temp2 = str(b) + str(a)

        if (temp1 > temp2):

            return -1

        elif (temp2 > temp1):

            return 1

        else:

            return 0

    def largestNumber(self, A):

        temp = sorted(A, key=functools.cmp\_to\_key(self.compare))

        ans = ""

        flag = -1

        for i in temp:

            if i!=0:

                flag = 1

            ans += str(i)

        if flag==-1:

            return 0

        return ans

**Question 67 Unique Elements**

You are given an array **A** of **N** elements. You have to make all elements unique. To do so, in one step you can increase any number by one.

Find the minimum number of steps.  
  
**Problem Constraints**

1 <= N <= 105  
1 <= A[i] <= 109

Logic –

1. Sort the array.
2. Iterate the array from i=1 to n-1 and check if the previous element is equal to the current element, then add 1 to the current element and increment the count by 1.
3. If in the above process the current element becomes greater than the next element, then convert the next element to current element+1 and increment the count accordingly.

Code –

class Solution:

    # @param A : list of integers

    # @return an integer

    def solve(self, A):

        A.sort()

        i = 1

        cnt = 0

        while(i<len(A)):

            if A[i] == A[i-1]:

                cnt += 1

                A[i] += 1

            elif A[i-1] > A[i]:

                cnt += A[i-1] + 1 - A[i]

                A[i] = A[i-1] + 1

            i+=1

        return cnt

**Question 68 Reverse Pairs**

Given an array of integers **A**, we call **(i, j)** an **important reverse pair** if **i < j** and **A[i] > 2\*A[j]**.

Return the number of important reverse pairs in the given array **A**.

**Problem Constraints**

1 <= length of the array <= 105

-2 \* 109 <= A[i] <= 2 \* 109

Logic – Similar to merge sort algorithm

1. We will do a usual merge sort, but before calling the merge function, we will calculate the number of pairs using two pointers, considering that the two arrays are sorted individually.
2. Likewise, we will do this till our merge sort ends, i.e., the array becomes sorted.

Code –

class Solution:

    def merge(self, A, l, y, r):

        p1 = l

        p2 = y+1

        arr = []

        cnt = 0

        i = y

        j = r

        while(p1<=y and p2<=r):

            if A[p1] > A[p2]:

                arr.append(A[p2])

                p2+=1

            elif A[p2] > A[p1]:

                arr.append(A[p1])

                p1+=1

            else:

                arr.append(A[p1])

                p1+=1

        while(i>=l and j>y):

            if A[i] > 2\*A[j]:

                cnt+=(j-y)

                i-=1

            elif A[i] <= 2\*A[j]:

                j-=1

        while(p1<=y):

            arr.append(A[p1])

            p1+=1

        while(p2<=r):

            arr.append(A[p2])

            p2+=1

        for i in range(r-l+1):

            A[l+i] = arr[i]

        return cnt

    def helper(self, A, l, r):

        if l==r:

            return 0

        mid = (l+r) // 2

        a = self.helper(A, l, mid)

        b = self.helper(A, mid+1, r)

        c = self.merge(A, l, mid, r)

        return (a + b + c)

    def solve(self, A):

        return self.helper(A, 0, len(A)-1)

**Question 69 B Closest Points to Origin**

We have a list **A** of points **(x, y)** on the plane. Find the **B** closest points to the origin **(0, 0)**.

Here, the distance between two points on a plane is the **Euclidean distance**.

You may return the answer in any order. The answer is guaranteed to be unique (except for the order that it is in.)

**NOTE:** Euclidean distance between two points **P1(x1, y1)** and **P2(x2, y2)** is **sqrt( (x1-x2)2 + (y1-y2)2 )**.  
  
**Problem Constraints**

1 <= B <= length of the list A <= 105  
-105 <= A[i][0] <= 105  
-105 <= A[i][1] <= 105

Logic – Use a modified sort

1. Sort the given array according to the distance between them and the origin.
2. After sorting return the first B elements.

Code –

import functools

import math

class Solution:

    # @param A : list of list of integers

    # @param B : integer

    # @return a list of list of integers

    def myCmp(self, a, b):

        x1 = a[0]

        y1 = a[1]

        x2 = b[0]

        y2 = b[1]

        x = math.sqrt((x1\*x1) + (y1\*y1))

        y = math.sqrt((x2\*x2) + (y2\*y2))

        if x>y:

            return 1

        elif y>x:

            return -1

        return 0

    def solve(self, A, B):

        C = sorted(A, key = functools.cmp\_to\_key(self.myCmp))

        ans = []

        for i in range(B):

            ans.append(C[i])

        return ans

**Question 70 Sum the Difference**

Given an integer array, **A** of size **N**.

You have to find all possible non-empty subsequences of the array of numbers and then, for each subsequence, find the difference between the largest and smallest numbers in that subsequence. Then add up all the differences to get the number.

As the number may be large, output the number modulo 1e9 + 7 (1000000007).

**NOTE:** Subsequence can be non-contiguous.

**Problem Constraints**

1 <= N <= 10000

1<= A[i] <=1000

Logic –

1. Sort the array. Then we can find the number of sub sequences in which each element is minimum and maximum.
2. The ith is smallest in (2N-i-i) subsequence and largest in (2i) subsequence (where N = length of the array)
3. We can add all maximum values (2i \* A[i]) and minimum values (2N-i-i \* A[i]) and return their difference.

Code –

class Solution:

    # @param A : list of integers

    # @return an integer

    def solve(self, A):

        A.sort()

        mx = 0

        mn = 0

        mod = 1000000007

        for i in range(len(A)):

            mx += ((1<<i)\*A[i]) % mod

            mn += ((1<<(len(A)-1-i)) \* A[i]) % mod

        return (mx-mn) % mod

**Question 71 Quick Sort**

Given an integer array **A**, sort the array using **QuickSort**.  
  
**Problem Constraints**

1 <= |A| <= 105

1 <= A[i] <= 109

Logic –

There are many different versions of quickSort that pick pivot in different ways:

-> Always pick first element as pivot.  
-> Always pick last element as pivot (implemented below)  
-> Pick a random element as pivot.  
-> Pick median as pivot.

The key process in quickSort is partition().  
Target of partitions is, given an array and an element x of array as pivot, put x at its correct position in sorted array and put all smaller elements (smaller than x) before x, and put all greater elements (greater than x) after x.

Repeat the same procedure for the array on the left of the element x and right of the element x.

Average Case Time Complexity : O(NlogN)  
Worst Case : O(N2)

Code –

class Solution:

    # @param A : list of integers

    # @return a list of integers

    def rearrange(self, A, l, r):

        p1 = l+1

        p2 = r

        while(p1<=p2):

            if (A[p1] < A[l]):

                p1+=1

            elif (A[p2] > A[l]):

                p2-=1

            else:

                A[p1], A[p2] = A[p2], A[p1]

                p2-=1

                p1+=1

        A[p2], A[l] = A[l], A[p2]

        return p2

    def helper(self, A, l, r):

        if l == r:

            return

        x = self.rearrange(A, l, r)

        if x==l:

            self.helper(A, x+1, r)

        elif x==r:

            self.helper(A, l, x-1)

        else:

            self.helper(A, l, x-1)

            self.helper(A, x+1, r)

    def solve(self, A):

        self.helper(A, 0, len(A)-1)

        return A

**Question 72 Maximum Unsorted Array**

Given an array **A** of non-negative integers of size **N**. Find the minimum sub-array **Al, Al+1 ,..., Ar** such that if we sort(in ascending order) that sub-array, then the whole array should get sorted. If **A** is already sorted, output **-1**.  
  
**Problem Constraints**

1 <= N <= 1000000  
1 <= A[i] <= 1000000

Logic –

1. Assume that **Al**, …, **Ar** is the minimum-unsorted-subarray which is to be sorted.  
   then min(**Al**, …, **Ar**) >= max(**A0**, …, **Al-1**)  
   and max(**Al**, …, **Ar**) <= min(**Ar+1**, …, **AN-1**)
2. Using this property we can find the maximum unsorted subarray.
3. First iterate from the left and find a point “i” with A[i+1] < A[i] and iterate from right and find a point “j” with A[j-1] > A[j].
4. Find the maximum and minimum element in this range([i, j]).
5. Iterate from the left and break when a A[x] greater than Amin (A[x] > Amin) is found. The value of x after breaking the loop is the left bound of the Maximum unsorted subarray. We are breaking the loop here because the rule mentioned in the first step is first violated at this x (index nearest to first element violating the property mentioned in step 1).
6. Similarly, we can find the right bound by iterating from the right and breaking the loop when A[x] less than Amax (A[x] < Amax) is found. We are breaking the loop here because the rule mentioned in the first step is first violated at this x (index nearest to last element violating the property mentioned in step 1).

Code –

class Solution:

    # @param A : list of integers

    # @return a list of integers

    def subUnsort(self, A):

        n = len(A)

        i = 0

        j = n-1

        while(i<n-1):

            if A[i+1] < A[i]:

                break

            i+=1

        # Check if array is already sorted

        if i==n-1:

            return [-1]

        while(j<n):

            if A[j-1] > A[j]:

                break

            j-=1

        Amax = -1

        Amin = 1000000

        for x in range(i, j+1):

            Amax = max(Amax, A[x])

            Amin = min(Amin, A[x])

        a1 = i

        a2 = j

        for x in range(i):

            if A[x] > Amin:

                a1 = x

                break

        for x in range(len(A)-1, j, -1):

            if A[x] < Amax:

                a2 = x

                break

        return [a1, a2]

**Question 73 Maximum and Minimum Magic**

Given an array of integers **A** of size **N** where **N** is even.

Divide the array into two subsets such that

1.Length of both subset is equal.

2.Each element of A occurs in exactly one of these subset.

**Magic number = sum of absolute difference of corresponding elements of subset.**

**Note:** You can reorder the position of elements within the subset to find the value of the magic number.

For Ex:-

subset 1 = {1, 5, 1},

subset 2 = {1, 7, 11}

Magic number = abs(1 - 1) + abs(5 - 7) + abs(1 - 11) = 12

Return an array **B** of size **2**, where B[0] = maximum possible value of Magic number modulo **109 + 7**, B[1] = minimum possible value of a Magic number modulo **109 + 7**.  
  
**Problem Constraints**

1 <= N <= 105

-109 <= A[i] <= 109

N is even

Logic –

1. Initially, sort the array.
2. For the minimum magic, find the sum of the difference between adjacent elements in pairs of two.
3. For the maximum magic, find the sum of the difference between the two elements equidistant from the front and back of the array.

Code –

class Solution:

    # @param A : list of integers

    # @return a list of integers

    def solve(self, A):

        mod = 1000000007

        A.sort()

        ans1 = 0

        for i in range(len(A)//2):

            ans1 += abs(A[i] - A[len(A)-i-1]) % mod

        ans2 = 0

        for i in range(0, len(A)-1, 2):

            ans2 += abs(A[i]-A[i+1]) % mod

        return [ans1 % mod, ans2 % mod]

**Question 73 Sorted Insert Position**

Given a sorted array A of size N and a target value B, return the index (0-based indexing) if the target is found.  
If not, return the index where it would be if it were inserted in order.

**NOTE:**You may assume no duplicates in the array. Users are expected to solve this in O(log(N)) time.  
  
**Problem Constraints**

1 <= N <= 106

Logic – Binary Search

1. Search using the binary search and if we find the element return the index and if we don’t find the element in the array then we can return l.

Code –

class Solution:

    # @param A : list of integers

    # @param B : integer

    # @return an integer

    def searchInsert(self, A, B):

        l = 0

        r = len(A)-1

        while(l<=r):

            mid = l+(r-l)//2

            if A[mid] == B:

                return mid

            elif A[mid] > B:

                r = mid-1

            else:

                l = mid+1

        return l

**Question 75 Find a Peak Element (Maxima)**

Given an array of integers **A**, find and return the peak element in it. An array element is peak if it is NOT smaller than its neighbours.

For corner elements, we need to consider only one neighbour. We ensure that answer will be unique.

**Problem Constraints**

1 <= **|A|** <= 100000

1 <= A[i] <= 109

Logic –

1) Initially let l = 0 and r = A.size()-1  
2) Repeat steps 3-4 while l<=r  
3) Set m=(l+r)/2  
4) If A[m] >= A[m-1] and A[m] >= A[m+1], A[m] is the peak element. Set ans = A[m] and exit the loop  
Else if A[m] > A[m-1] we know that the peak element has to be on the right side of A[m]. Hence, we set l =m+1  
Else if A[m] < A[m-1] we know that the peak element has to be on the left side of A[m]. Hence, we set r=m-1.  
5) Return ans  
Edge cases: where the first or last element is the peak element.

Code –

class Solution:

    # @param A : list of integers

    # @return an integer

    def solve(self, A):

        if len(A)==1:

            return A[0]

        if A[0] > A[1]:

            return A[0]

        if A[-1] > A[-2]:

            return A[-1]

        l = 1

        r = len(A)-2

        while(l<=r):

            mid = l+(r-l)//2

            if A[mid] >= A[mid-1] and A[mid] >= A[mid+1]:

                return A[mid]

            elif A[mid] > A[mid-1]:

                l = mid+1

            else:

                r = mid-1

**Question 76 Rotated Sorted Array Search**

Given a sorted array of integers **A** of size N and an integer **B**.

array A is rotated at some pivot unknown to you beforehand.

(i.e., 0 1 2 4 5 6 7 might become 4 5 6 7 0 1 2 ).

You are given a target value B to search. If found in the array, return its index otherwise, return -1.

You may assume no duplicate exists in the array.

**NOTE:**Users are expected to solve this in O(log(N)) time.  
  
**Problem Constraints**

1 <= N <= 1000000

1 <= A[i] <= 10^9

all elements in A are distinct.

Logic – Binary Search

1. Compare the middle element with the first element of the array.
2. If middle element is greater than first element then we are in the left part of the rotated sorted array. If A[mid] is in the range [A[0], A[mid]] then we can discard the elements on the right of the mid else we can discard the left elements.
3. If we are in the right part (A[mid] < A[0]) and if A[mid] is in the range of [A[mid], A[n-1]] then we can discard the elements on the left of the mid else we can discard right elements.

Code –

class Solution:

    # @param A : tuple of integers

    # @param B : integer

    # @return an integer

    def search(self, A, B):

        l = 0

        r = len(A)-1

        while(l<=r):

            mid = l + ((r-l)>>1)

            if A[mid]==B:

                return mid

            elif A[mid]>=A[0]:

                if A[mid] > B and B >= A[0]:

                    r = mid-1

                else:

                    l = mid+1

            else:

                if A[mid] < B and B <= A[-1]:

                    l = mid+1

                else:

                    r = mid-1

        return -1

**Question 77 Matrix Search**

Given a matrix of integers **A** of size **N x M** and an integer **B**. Write an efficient algorithm that searches for integer **B** in matrix **A**.

This matrix A has the following properties:

1. Integers in each row are **sorted** from left to right.
2. The first integer of each row is greater than or equal to the last integer of the previous row.

Return **1** if **B** is present in **A**, else return **0**.

**NOTE:** Rows are numbered from top to bottom, and columns are from left to right.  
  
**Problem Constraints**

1 <= N, M <= 1000  
1 <= A[i][j], B <= 106

Logic –

1. Find the row in which the element is present using binary search (Check for the first and the last element of each row).
2. Using the binary search in that array to search the element.

Code –

public class Solution {

    public int binSearch(int[] A, int B) {

        int l = 0;

        int r = A.length-1;

        while(l<=r) {

            int mid = l + ((r-l)>>1);

            if (A[mid] == B) {

                return 1;

            } else if (A[mid] > B) {

                r = mid-1;

            } else{

                l = mid+1;

            }

        }

        return 0;

    }

    public int searchMatrix(int[][] A, int B) {

        int l = 0;

        int r = A.length-1;

        int m = A[0].length-1;

        while(l<=r) {

            int mid = l + ((r-l)>>1);

            if (B >= A[mid][0] && B <= A[mid][m]){

                return binSearch(A[mid], B);

            } else if (A[mid][0] > B){

                r = mid-1;

            } else {

                l = mid+1;

            }

        }

        return 0;

    }

}

**Question 78 Median of Array**

There are two sorted arrays **A** and **B** of sizes N and M respectively.

Find the **median** of the two sorted arrays ( The median of the array formed by merging both the arrays ).

**NOTE:**

* The overall run time complexity should be O(log(m+n)).
* IF the number of elements in the merged array is even, then the median is the average of (n/2)th and (n/2+1)th element. For example, if the array is [1 2 3 4], the median is (2 + 3) / 2.0 = 2.5.

**Problem Constraints**

1 <= N + M <= 2\*106

Logic –

1)

Code –

public class Solution {

    // DO NOT MODIFY BOTH THE LISTS

    public double findMedianSortedArrays(final List<Integer> a, final List<Integer> b) {

        int n = a.size();

        int m = b.size();

        if (n > m)

            return findMedianSortedArrays(b, a); // Swapping to make A smaller

        int l = 0;

        int r = n;

        while(l<=r) {

            int cut1 = (l+r)>>1;

            int cut2 = ((n+m+1)>>1) - cut1;

            int left1 = cut1>0 ? a.get(cut1-1) : Integer.MIN\_VALUE;

            int left2 = cut2>0 ? b.get(cut2-1) : Integer.MIN\_VALUE;

            int right1 = cut1<n ? a.get(cut1) : Integer.MAX\_VALUE;

            int right2 = cut2<m ? b.get(cut2) : Integer.MIN\_VALUE;

            if (left1 <= right2 && left2 <= right1) {

                if ((n+m)%2==0) {

                    return (Math.max(left1, left2) + Math.min(right1, right2))/2.0;

                } else {

                    return Math.max(left1, left2);

                }

            }

            else if (right2 < left1) {

                r = cut1-1;

            } else {

                l = cut1+1;

            }

        }

        return 0.0;

    }

}

**Question 79 Matrix Median**

Given a matrix of integers **A** of size N x M in which each row is sorted.

Find and return the overall median of matrix A.

**NOTE**: No extra memory is allowed.

**NOTE**: Rows are numbered from top to bottom and columns are numbered from left to right.

**Problem Constraints**

1 <= N, M <= 10^5

1 <= N\*M <= 10^6

1 <= A[i] <= 10^9

N\*M is odd

Logic –

1. The median of a matrix has ((n\*m)+1)/2 number of elements greater than or equal to itself.
2. So we have to find the number which has ((n\*m)+1)/2 number of elements greater or equal to itself.
3. We can find that using binary search.
4. First, we have to find the range of the median. The range is between minimum element of the array and the maximum element array.
5. For each mid element we find the number of elements greater than or equal to itself. If it is greater than ((n\*m)+1)/2 then we discard the elements on the right of the mid(r=mid).If it is less than ((n\*m)+1)/2 then we discard the elements on the left of the mid(l=mid+1).

Note - number of elements greater than or equal to itself = Upper bound of the element in the array, which can be found applying binary search in the array.

Code –

public class Solution {

    public int binSearch(int[] A, int K) {

        int l = 0;

        int r = A.length-1;

        while(l<=r) {

            int mid = l + ((r-l)>>1);

            if (A[mid] <= K) {

                l = mid+1;

            } else {

                r = mid-1;

            }

        }

        return l;

    }

    public int findMedian(int[][] A) {

        int n = A.length;

        int m = A[0].length;

        int l = A[0][0];

        int r = -1;

        for (int i=0;i<n;i++){

            r = Math.max(A[i][m-1], r);

            l = Math.min(A[i][0], l);

        }

        int temp = (n\*m+1)/2;

        int ans = -1;

        while(l<r) {

            int mid = l + ((r-l)/2);

            int x = 0;

            for(int i=0;i<n;i++) {

                x+= (binSearch(A[i], mid));

            }

            if(x<temp) {

                l = mid+1;

            } else {

                r = mid;

            }

        }

        return l;

    }

}

**Question 80 Square root of a number**

Given an integer **A**.

Compute and return the **square root of A**.

If **A** is not a perfect square, return **floor(sqrt(A)).**

**DO NOT USE SQRT FUNCTION FROM THE STANDARD LIBRARY.**

**NOTE:**Do not use the sqrt function from the standard library. Users are expected to solve this in O(log(A)) time.  
  
**Problem Constraints**

0 <= A <= 1010

Logic –

1. Use the binary search approach. Calculate mid ((l+r)/2) at every iteration and if mid\*mid > the given number, then any number greater than or equal mid cannot be the answer, so reduce the right range to mid-1.
2. Similarly if mid\*mid is less than the given number then mid can be a possible answer, but numbers less than mid cannot be the answer, so we reduce the left to mid+1 and keep a record of mid.

Code –

class Solution:

    # @param A : integer

    # @return an integer

    def sqrt(self, A):

        if A==0:

            return 0

        l = 0

        r = A

        ans = 0

        while(l<=r):

            mid = l+((r-l)>>1)

            if mid\*mid > A:

                r = mid-1

            else:

                ans = mid

                l = mid+1

        return ans

**Question 81 Rotated Sorted Array Search**

Given a sorted array of integers **A** of size N and an integer **B**.

array A is rotated at some pivot unknown to you beforehand.

(i.e., 0 1 2 4 5 6 7 might become 4 5 6 7 0 1 2).

You are given a target value B to search. If found in the array, return its index otherwise, return -1.

You may assume no duplicate exists in the array.

**NOTE:**Users are expected to solve this in O(log(N)) time.

**Problem Constraints**

1 <= N <= 1000000

1 <= A[i] <= 10^9

all elements in A are distinct.

Logic –

1. Use the binary search approach. Compare the middle element with the first element and if A[mid] > first element means the mid element is in left half of the array. So, if the element to be searched is between A[mid] and first element then we reduce the right range as the answer has to be in this range

(A[mid] > B > first element. Where B is element to be searched). If not then we increase the left range as the answer has to be in this range.

1. Similarly, if A[mid] in the right range of the array (A[mid] < A[0]) we check if B is between (A[mid] < B < A[n-1]). If yes then the answer has to be in this range so reduce the range by increasing the left bound. If the element B is not present in the given range then we reduce the right bound.

Code –

public class Solution {

    // DO NOT MODIFY THE ARGUMENTS WITH "final" PREFIX. IT IS READ ONLY

    public int search(final int[] A, int B) {

        int l = 0;

        int r = A.length-1;

        int n = A.length;

        while(l<=r){

            int mid = l+((r-l)>>1);

            if (A[mid] == B) {

                return mid;

            } else if (A[mid] > A[0]){

                if (A[mid] > B && B >= A[0]) {

                    r = mid-1;

                } else{

                    l = mid+1;

                }

            } else {

                if (A[mid] < B && B <= A[n-1]) {

                    l = mid+1;

                } else {

                    r = mid-1;

                }

            }

        }

        return -1;

    }

}

**Question 82 Special Integer**

Given an array of integers **A** and an integer **B**, find and return the maximum value K such that there is no subarray in A of size K with the sum of elements greater than B.  
  
**Problem Constraints**

1 <= |A| <= 100000

1 <= A[i] <= 10^9

1 <= B <= 10^9

Logic – Binary Search + Sliding window approach

1. For the size of the subarray (k) check if the maximum sum of subarrays of length k is less than or equal to B. If it is less than or equal to B then that k is a possible answer, so we store that k and reduce the range of the k by increasing the value of left bound to mid+1.
2. Similarly, if maximum sum of subarrays of length k is greater than B, then a subarray of size greater than k will never be a answer. So we reduce the range of k by reducing the right bound to mid-1.
3. We can find the maximum sum of subarrays of length k using Sliding window approach, and vary the value of k using binary search.

Code –

public class Solution {

    public boolean SWA(int[] A, int K, int B) {

        long sum1 = 0;

        for(int i=0;i<K;i++) {

            sum1+=A[i];

        }

        long ans = sum1;

        for (int i=1;i<(A.length-K+1);i++) {

            sum1 = (long) (sum1 - A[i-1] + A[i+K-1]);

            ans = (long) Math.max(sum1, ans);

        }

        return ans<=B;

    }

    public int solve(int[] A, int B) {

        int n = A.length;

        int l = 0;

        int r = n;

        int ans = -1;

        while(l<=r){

            int mid = l + ((r-l)>>1);

            boolean x = SWA(A, mid, B);

            if (x){

                ans=mid;

                l = mid+1;

            } else{

                r = mid-1;

            }

        }

        return ans;

    }

}

**Question 83 Ath Magical Number**

You are given three positive integers, **A**, **B,** and **C**.

Any positive integer is magical if divisible by either **B** or **C**.

Return the **Ath** smallest magical number. Since the answer may be very large, return modulo **109 + 7**.  
  
**Problem Constraints**

1 <= A <= 109

2 <= B, C <= 40000

Logic –

Say L = lcm(B, C), the least common multiple of B and C, and let f(x) be the number of magical numbers less than or equal to x.  
A well known result says that **L = (B\*C)/gcd(B,C)**, and that we can calculate the function gcd.

Then **f(x) = x/B + x/C - x/L** (floor of all the divisions)

There are x/B numbers B, 2B, 3B…. that are divisible by B, there are x/C numbers divisible by C, and we need to subtract the x/L numbers divisible by B and C that we double-counted.

Finally, the answer must be between **min (B, C)** and **A \* min (B, C)**.

If x increases f(x) increases, we can use binary search on x to find the Ath number.

Algorithm:  
1) Low=1 and High = A \* min(B,C)  
2) Find mid = (low + high)/2  
3) Find f(mid) let it be count  
4) If count>=A then mark it as a answer and try to find smaller number which implies high = mid-1  
5) Else low = mid+1  
6) Repeat steps 2 to 5 until low<=high

Code –

public class Solution {

    public int gcd(int A, int B) {

        while(B!=0) {

            int temp = B;

            B = A%B;

            A = temp;

        }

        return A;

    }

    public long count(long x, int B, int C) {

        long lcm = (B\*C) / gcd(B,C);

        return ((x/B) + (x/C) - (x/lcm));

    }

    public long solve(int A, int B, int C) {

        long l = (long) Math.min(B,C);

        long r = (long) A\*Math.min(B,C);

        long ans = 0;

        while(l<=r){

            long mid = l + ((r-l)>>1);

            long x = count(mid, B, C);

            if (x>=A){

                ans = mid;

                r = mid-1;

            } else{

                l = mid+1;

            }

        }

        return (int) (ans % (1000 \* 1000 \* 1000 + 7));

    }

}

**Question 84 Kth Price**

Given the price list at which tickets for a flight were purchased, figure out the kth smallest price for the flight. kth smallest price is the minimum possible n such that there are at least k price elements in the price list with value <= n. In other words, if the price list was sorted, then A[k-1] ( k is 1 based, while the array is 0 based ).

**NOTE** You are not allowed to modify the price list ( The price list is read only ). Try to do it using constant extra space.

**Example:**

A : [2 1 4 3 2]

k : 3

Answer : 2

**Constraints :**

* 1 <= number of elements in the price list <= 1000000
* 1 <= k <= number of elements in the price list

Logic – Binary Search approach

1. We count the number of elements in array less than or equal to the middle element.
2. If the count is greater than or equal to K then it is a possible answer and all the elements greater than mid are also the possible answers but mid element is the best possible answer so we consider it and reduce the right bound.
3. If the count is less than K then mid is not the required answer and any number less than mid cannot be a answer. So, we increase the left bound.

Code –

class Solution:

    # @param A : tuple of integers

    # @param B : integer

    # @return an integer

    def count(self, A, k):

        # Returns the no of elements in array less than or equal to k

        cnt = 0

        for i in A:

            if i<=k:

                cnt+=1

        return cnt

    def solve(self, A, B):

        l = 1

        r = max(A)

        ans = len(A)

        while(l<=r):

            mid = l + ((r-l)>>1)

            x = self.count(A,mid)

            if x>=B:

                ans = mid

                r = mid-1

            else:

                l = mid+1

        temp = 1000000000

        return ans

**Question No 85 Search in Bitonic Array**

Given a bitonic sequence **A** of **N** distinct elements, write a program to find a given element **B** in the bitonic sequence in **O(logN)** time.

**NOTE:**

 A Bitonic Sequence is a sequence of numbers which is first strictly increasing then after a point strictly decreasing.  
  
**Problem Constraints**

3 <= N <= 105

1 <= A[i], B <= 108

Given array always contain a bitonic point.

Array A always contain distinct elements.

Logic –

* 1. Find the bitonic point. If A[mid] > A[mid+1] and A[mid] > A[mid-1] then we can conclude that mid is the bitonic point.
  2. Then apply binary search for the element in left part and apply reverse binary search in right part of the array.

Code –

class Solution:

    # @param A : list of integers

    # @param B : integer

    # @return an integer

    def revbin(self, A, l, r, B):

        low = l

        high = r

        while(low<=high):

            mid = low + ((high-low)>>1)

            if A[mid]==B:

                return mid

            elif A[mid]>B:

                low = mid+1

            else:

                high = mid-1

        return -1

    def bin(self, A, l, r, B):

        low = l

        high = r

        while(low<=high):

            mid = low + ((high-low)>>1)

            if A[mid]==B:

                return mid

            elif A[mid]>B:

                high = mid-1

            else:

                low = mid+1

        return -1

    def solve(self, A, B):

        l=0

        r=len(A)-1

        while(l<=r):

            mid = l + ((r-l)>>1)

            if A[mid] == B:

                return mid

            # Mid is the bitonic point

            elif A[mid]>A[mid-1] and A[mid]>A[mid+1]:

                if A[mid] < B:

                    return -1

                x = self.revbin(A, mid+1, len(A)-1, B)

                y = self.bin(A, 0, mid-1, B)

                if x==-1 and y==-1:

                    return -1

                elif x==-1:

                    return y

                else:

                    return x

            # left part

            elif(A[mid] > A[mid-1] and A[mid] < A[mid+1]):

                l=mid+1

            # right part

            elif ((A[mid] < A[mid-1] and A[mid] > A[mid+1])):

                r = mid-1

        return -1

**Question No 86 Painter's Partition Problem**

Given 2 integers **A** and **B** and an array of integers **C** of size **N**. Element **C[i]** represents the length of **ith** board.  
You have to paint all **N** boards **[C0, C1, C2, C3 … CN-1]**. There are **A** painters available and each of them takes **B** units of time to paint **1 unit** of the board.

Calculate and return the minimum time required to paint all boards under the constraints that **any painter will only paint contiguous sections of the board.**

**NOTE:**  
**1.** 2 painters cannot share a board to paint. That is to say, a board cannot be painted partially by one painter, and partially by another.  
**2.** A painter will only paint contiguous boards. This means a configuration where painter 1 paints boards 1 and 3 but not 2 is invalid.  
  
Return the **ans % 10000003**.

**Problem Constraints**

1 <= A <= 1000  
1 <= B <= 106  
1 <= N <= 105  
1 <= C[i] <= 106

Logic – Use Binary Search

1. The minimum possible time for the work to complete is maximum value of element in A.
2. The maximum possible time for the work to complete is sum of all elements in A.
3. Therefore, range of the answer is min(A) to sum(A) (l=min(A), r=sum(A)).
4. For each mid ((l+r)/2) we have to check that if the work is to be completed in mid units of time, then how many workers do we need. If this number is less than or equal to the available painters then all values above mid are the possible answers but mid is the best possible answer as the work can be done in less time by using other painters.
5. But if this value is greater than the number of painters then all values below mid can never be the answers as the work needs more time as the painters are falling short.

Code –

class Solution:

    # @param A : integer

    # @param B : integer

    # @param C : list of integers

    # @return an integer

    def checkFunc(self, C, mid, A):

        painter = 0

        sum1 = 0

        for i in range(len(C)):

            sum1+=C[i]

            if sum1>mid:

                sum1 = C[i]

                painter+=1

                if painter == A:

                    return False

        return True

    def paint(self, A, B, C):

        if len(C) <= A:

            return (max(C)\*B) % 10000003

        l = max(C)

        r = sum(C)

        ans = -1

        while(l<=r):

            mid = (l+r)//2

            if self.checkFunc(C, mid, A):

                ans = mid

                r = mid-1

            else:

                l = mid+1

        return (ans\*B) % 10000003

**Question No 87 - Aggressive cows**

Farmer John has built a new long barn with **N** stalls. Given an array of integers **A** of size **N** where each element of the array represents the location of the stall and an integer **B** which represents the number of cows.

His cows don't like this barn layout and become aggressive towards each other once put into a stall. To prevent the cows from hurting each other, John wants to assign the cows to the stalls, such that the minimum distance between any two of them is as large as possible. What is the largest minimum distance?  
  
**Problem Constraints**

2 <= N <= 100000  
0 <= A[i] <= 109  
2 <= B <= N

Logic – Similar to above problem.

1. The minimum possible answer in this case is 1, as we can place all cows at a distance of 1 unit.
2. The maximum possible answer is max in the array – min in the array.
3. For each mid we have to check how many cows can be placed with least distance of mid. If we can place cows greater than or equal to the given cows then all the answers above mid are the possible answers (and the mid is the best possible answer), as we can further increase the distance between the cows.
4. But if we can place cows less than the given number of cows then all the numbers above mid and mid cannot be the answer, as we would need more space to place the extra cows.

Code –

class Solution:

    # @param A : list of integers

    # @param B : integer

    # @return an integer

    def checkFunc(self, A, k, B):

        cow = 1

        last\_placed = A[0]

        for i in range(1, len(A)):

            if A[i]-last\_placed >= k:

                last\_placed = A[i]

                cow+=1

                if cow==B:

                    return 1

        return 0

    def solve(self, A, B):

        l = 1

        A.sort()

        r = A[-1]-A[0]

        ans = -1

        while(l<=r):

            mid = l + ((r-l>>1))

            # Check if I can place cows with least distance of mid

            x = self.checkFunc(A, mid, B)

            if (x==1):

                ans = mid

                l=mid+1

            else:

                r = mid-1

        return ans

**Question No 88 Allocate books**

Given an array of integers **A** of size **N** and an integer **B**.

The College library has **N** books. The **ith** book has **A[i]** number of pages.

You have to allocate books to **B** number of students so that the maximum number of pages allocated to a student is minimum.

A book will be allocated to exactly one student.

Each student has to be allocated at least one book.

Allotment should be in contiguous order, for example: A student cannot be allocated book 1 and book 3, skipping book 2.

Calculate and return that minimum possible number.

**NOTE:** Return -1 if a valid assignment is not possible.  
  
**Problem Constraints**

1 <= N <= 105  
1 <= A[i], B <= 105

Logic –

1. The minimum answer is maximum of the array as 1 book can be distributed to 1 student.
2. The maximum answer is sum of the array as all the books can be distributed to 1 student.
3. For every mid we have to find the number of students to which we can distribute at most mid number of pages. If this number is less than or equal to B it means all the values above mid can be the answer but mid is the best possible answer. So, we store the value of mid and reduce the right bound to r = mid-1.
4. If this number is greater than B, it means all the answers below mid cannot be the answers as they will also need more students than B. So we increase the left bound to l=mid+1.

Code –

class Solution:

    # @param A : list of integers

    # @param B : integer

    # @return an integer

    def checkFunc(self, A, mid, B):

        # Check whether mid pages be allocated to B students

        student = 1

        pages\_allocated = 0

        for i in range(len(A)):

            if (pages\_allocated + A[i] <= mid):

                pages\_allocated += A[i]

            else:

                pages\_allocated = A[i]

                student += 1

        # print(student, B, mid, end = ' ')

        return student

    def books(self, A, B):

        # print(len(A), B)

        n = len(A)

        if n<B:

            return -1

        l = max(A)

        r = sum(A)

        ans = max(A)

        while(l<=r):

            mid = l + ((r-l)>>1)

            x = self.checkFunc(A, mid, B)

            if x <= B:

                ans = mid

                r = mid-1

            else:

                l = mid+1

        return ans

**Question No 89 Food Packet Distribution**

The government wants to set up **B** distribution offices across **N** cities for the distribution of food  
packets. The population of the ithcity is **A[i]**. Each city must have at least 1 office, and people can go to an office of their own city. We want to maximize the minimum number of people who can get food in any single office.  
  
**Problem Constraints**

1 <= N <= 105

1 <= A[i] <= 106

1 <= B <= 5 x 105

Logic – Similar to above problems.

If we allot x offices to a city, then a[i]/x number of people can go to a single office.

Hence, for checking if a given number of people can go to a single office, we can add a[i]/x for all cities and check if it is greater than B.

Code –

class Solution:

    # @param A : list of integers

    # @param B : integer

    # @return an integer

    def check(self, A, mid):

        sum1 = 0

        for i in A:

            if (i//mid) == 0:

                return -1

            sum1 += (i//mid)

        return sum1

    def solve(self, A, B):

        l = 1

        r = min(A)

        ans = l

        if sum(A) < B:

            return 0

        while(l<=r):

            mid = (l+r) >> 1

            x = self.check(A, mid)

            if x<B:

                r = mid-1

            else:

                ans = mid

                l = mid+1

        return ans

**Question No 90 Subarray with given sum**

Given an array of positive integers A and an integer B, find and return first continuous subarray which adds to B.

If the answer does not exist return an array with a single element "-1".

First sub-array means the sub-array for which starting index in minimum.  
  
**Problem Constraints**

1 <= length of the array <= 100000  
1 <= A[i] <= 109  
1 <= B <= 109

Logic –

1. Find the prefix sum and find the elements having the difference equal to B in the prefix sum using the 2 pointer method.
2. Start with i=0 and j=1.If pref[j] - pref[j] < B then increment the j pointer. Else increment the I pointer. After incrementing i if i==j then increment j and i can never be equal to j.

Code –

class Solution:

    # @param A : list of integers

    # @param B : integer

    # @return a list of integers

    def solve(self, A, B):

        pref = []

        pref.append(0)

        sum1 = 0

        for i in A:

            sum1 += i

            pref.append(sum1)

        i = 0

        j = 1

        while(j<=len(A)):

            if pref[j] - pref[i] < B:

                j+=1

            elif pref[j] - pref[i] > B:

                i+=1

                if i==j:

                    j+=1

            else:

                return A[i:j]

        return [-1]

**Question No 91 Pairs with given difference**

Given an one-dimensional integer array **A** of size **N** and an integer **B**.

**Count all distinct pairs with difference equal to B**.

Here a pair is defined as an integer pair (x, y), where x and y are both numbers in the array and their absolute difference is **B**.  
**Problem Constraints**

1 <= N <= 104

0 <= A[i], B <= 105

Logic –

1. Formulate a two pointer approach to the this problem.
2. We will first sort the given array and use two pointers i and j with i = 0, j = 1.

3)We will have three conditions:

1. A[j] - A[i] < B --> We will increase the pointer j.

2. A[j] - A[i] > B --> We will increase the pointer i.

3. A[j] - A[i] = B --> We will increase both the pointers and add 1 to the answer.

Code –

class Solution:

    # @param A : list of integers

    # @param B : integer

    def solve(self, A, B):

        cnt = 0

        i = 0

        j = 1

        A.sort()

        while(j<len(A)):

            if (A[j] - A[i]) < B:

                x = A[j]

                while(j<len(A) and A[j]==x):

                    j+=1

            elif (A[j] - A[i]) > B:

                x = A[i]

                while(i<j and A[i]==x):

                    i+=1

                if i==j:

                    j+=1

            else:

                cnt+=1

                x = A[j]

                while(j<len(A) and A[j]==x):

                    j+=1

        return cnt

**Question No 92 Container with most water**

Given n non-negative integers A[0], A[1], ..., A[n-1] , where each represents a point at coordinate (i, A[i]).

**N** vertical lines are drawn such that the two endpoints of line i is at (i, A[i]) and (i, 0).

Find two lines, which together with x-axis forms a container, such that the container contains the most water.

**Note:** You may not slant the container.  
  
**Problem Constraints**

0 <= N <= 105

1 <= A[i] <= 105

Logic – 2 pointer method

1. Start with i=0 and j =n-1.
2. If A[i] is less than A[j] then we can calculate the amount of water, compare it with the current answer and store the maximum among them and increment i, as A[i] will not have a better answer.
3. Similarly, if A[i] is greater than A[j] then store the maximum answer and decrement j.

Code –

class Solution:

    # @param A : list of integers

    # @return an integer

    def maxArea(self, A):

        i = 0

        j = len(A)-1

        ans = 0

        while(i<j):

            if A[i] < A[j]:

                temp = A[i] \* (j-i)

                ans = max(ans, temp)

                i+=1

            else:

                temp = A[j] \* (j-i)

                ans = max(ans, temp)

                j-=1

        return ans

**Question No 93 3 Sum Zero**

Given an array **A** of **N** integers, are there elements a, b, c in S such that **a + b + c = 0**

Find all unique triplets in the array which gives the sum of zero.

**Note:**

Elements in a triplet (a,b,c) must be in non-descending order. (ie, a ≤ b ≤ c) The solution set must not contain duplicate triplets.  
  
**Problem Constraints**

0 <= N <= 7000

-108 <= A[i] <= 108

Logic –

1. Sort the array. Then traverse the array (assume variable - i) and apply 2 pointer (assume j and k) from the i+1th element to the last element to find A[j]+A[k]-A[i].

Code –

class Solution:

    # @param A : list of integers

    def threeSum(self, A):

        ans = []

        A.sort()

        i = 0

        while(i<len(A)):

            B = -1 \* A[i]

            j = i+1

            k = len(A)-1

            while(k>j):

                if A[j] + A[k] == B:

                    ans.append([A[i], A[j], A[k]])

                    x = A[j]

                    y = A[k]

                    while(k>j and A[j] == x):

                        j+=1

                    while(k>j and A[k] == y):

                        k-=1

                elif A[j] + A[k] > B:

                    k-=1

                else:

                    j+=1

            temp = A[i]

            while(i<len(A) and temp == A[i]):

                i+=1

        return ans

**Question No 94 Pairs with given sum II**

Given a sorted array of integers (not necessarily distinct) **A** and an integer **B**, find and return how many pair of integers ( A[i], A[j] ) such that i != j have sum equal to B.

Since the number of such pairs can be very large, return number of such pairs modulo (109 + 7).  
  
**Problem Constraints**

1 <= **|A|** <= 100000

1 <= A[i] <= 10^9

1 <= B <= 10^9

Logic –

1. Use 2 pointer method. Start with i=0 and j = n-1.
2. If A[i] +A[j] > B then decrement j, else increment i.
3. If A[i]+A[j] == B then find the count. There are 2 possible cases:
4. A[i] == A[j]. In this case we count the number of A[i]s in the array (j-i+1), and number of pairs can be found by the formula (n\*(n-1)/2)
5. Else we find the number of A[i]s and A[j]s separately and multiply them to find the possible pairs.

Code –

class Solution:

    # @param A : list of integers

    # @param B : integer

    # @return an integer

    def solve(self, A, B):

        i = 0

        j = len(A)-1

        cnt = 0

        while(j>i):

            if A[i]+A[j] == B:

                if A[i] == A[j]:

                    n = (j-i+1)

                    cnt += (n\*(n-1)//2)

                    return cnt % (1000000007)

                else:

                    x = A[i]

                    y = A[j]

                    cnt1 = 0

                    cnt2 = 0

                    while(A[i]==x):

                        cnt1+=1

                        i+=1

                    while(A[j]==y):

                        cnt2+=1

                        j-=1

                    cnt += (cnt1\*cnt2)

            elif A[i]+A[j] > B:

                j-=1

            else:

                i+=1

        return cnt % (1000000007)

**Question No 95 Array 3 pointers**

You are given 3 sorted arrays **A, B and C**.

Find **i, j, k** such that : **max(abs(A[i] - B[j]), abs(B[j] - C[k]), abs(C[k] - A[i]))** is **minimized**.

Return the **minimum max(abs(A[i] - B[j]), abs(B[j] - C[k]), abs(C[k] - A[i]))**.  
  
**Problem Constraints**

0 <= len(A), len(B), len(c) <= 106

0 <= A[i], B[i], C[i] <= 107

Logic – Windowing strategy

1. Take 3 pointers X, Y and Z
2. Initialize them to point to the start of arrays A, B and C
3. Find min of X, Y and Z.
4. Compute max(X, Y, Z) - min(X, Y, Z).
5. If new result is less than current result, change it to the new result.
6. Increment the pointer of the array which contains the minimum.

Code –

class Solution:

    # @param A : tuple of integers

    # @param B : tuple of integers

    # @param C : tuple of integers

    # @return an integer

    def minimize(self, A, B, C):

        i = 0

        j = 0

        k = 0

        ans = 10000000

        while(i<len(A) and j<len(B) and k<len(C)):

            mx = max(A[i], B[j], C[k])

            mn = min(A[i], B[j], C[k])

            ans = min(ans, mx-mn)

            if i<len(A) and mn == A[i]:

                i+=1

            elif j<len(B) and mn == B[j]:

                j+=1

            elif k<len(C) and mn == C[k]:

                k+=1

        return ans

**Question No 96 Sort by color**

Given an array with **N** objects colored **red, white, or blue**, sort them so that objects of the same color are adjacent, with the colors in the **order red, white, and blue.**

We will use the integers **0, 1, and 2 to represent red, white, and blue**, respectively.

**Note:** Using the library sort function is not allowed.  
  
**Problem Constraints**

1 <= N <= 1000000  
0 <= A[i] <= 2

Logic –

1. Swap the 0s to the start of the array maintaining a pointer, and 2s to the end of the array.  
   1s will automatically be in their right position.

Code –

class Solution:

    # @param A : list of integers

    # @return a list of integers

    def sortColors(self, A):

        i = 0

        j = 0

        k = len(A)-1

        while(k>=j):

            if A[j] == 0:

                A[i], A[j] = A[j], A[i]

                i+=1

                j+=1

            elif A[j] == 1:

                j+=1

            else:

                A[k], A[j] = A[j], A[k]

                k-=1

        return A

**Question No 97 Closest pair from sorted arrays**

Given two sorted arrays of distinct integers, **A** and **B**, and an integer **C**, find and return the pair whose sum is closest to C and the pair has one element from each array.

More formally, find A[i] and B[j] such that **abs((A[i] + B[j]) - C)** has minimum value.

If there are multiple solutions find the one with minimum i and even if there are multiple values of j for the same i then return the one with minimum j.

Return an array with two elements **{A[i], B[j]}**.  
  
**Problem Constraints**

1 <= length of both the arrays <= 100000

1 <= A[i], B[i] <= 109

1 <= C <= 109

Logic – This problem can be solved by using two pointer approach.

1. Let say l = 0 i.e. it point at the start of array A and r = len(B) - 1 i.e. it points to end of array B.  
   Also maintain a variable, let say, dif, which stores the minimum of abs(A[i]+B[j]-C).
2. Now, If abs(A[l]+B[r]-C) < dif then update the value of dif and the value of two A[i] and A[j] and there indices.
3. If abs(A[l]+B[r]-C) == dif then we need to update the indixes such that l and r are minimum as possible in case of multiple solution.
4. Also, increment l and decrement r accordingly.

Code –

class Solution:

    # @param A : list of integers

    # @param B : list of integers

    # @param C : integer

    # @return a list of integers

    def solve(self, A, B, C):

        i = 0

        j = len(B) - 1

        ansi = 0

        ansj = len(B) - 1

        sum1 = abs(A[i] + B[j] - C)

        while(i<len(A) and j>=0):

            if sum1 > abs(A[i] + B[j] - C):

                sum1 = abs(A[i] + B[j] - C)

                ansi = i

                ansj = j

            elif sum1 == abs(A[i] + B[j] - C):

                if ansi == i:

                    ansj = min(ansj, j)

            if A[i] + B[j] < C:

                i+=1

            elif A[i] + B[j] > C:

                j-=1

            else:

                i+=1

                j-=1

        return [A[ansi], B[ansj]]

**Question No 98 3 Sum**

Given an array **A** of **N** integers, find three integers in **A** such that the sum is closest to a given number **B**. Return the sum of those three integers.

Assume that there will only be one solution.  
  
**Problem Constraints**

* -108 <= B <= 108
* 1 <= N <= 104
* -108 <= A[i] <= 108

Logic –

1. Iterate the array (use i as iterator), then use 2 pointer in the right half of the ith element in the array with j = i+1 and k = n-1.
2. Find the minimum value of |A[i]+A[j]+A[k] - B| at every step. If a pair with A[i]+A[j]+A[k] = B is found we can directly return that pair as it will be the closest to B.
3. If A[i]+A[j]+A[k] > B then decrement k, else increment j.

Code –

import sys

class Solution:

    # @param A : list of integers

    # @param B : integer

    # @return an integer

    def threeSumClosest(self, A, B):

        A.sort()

        i = 0

        ans = 1000000000

        while(i<len(A)-2):

            j = i+1

            k = len(A)-1

            while(j<k):

                s = A[i] + A[j] + A[k]

                # Find the minimum value of the answer

                if abs(s-B) < abs(ans-B):

                    ans = s

                if s < B:

                    j+=1

                elif s > B:

                    k-=1

                else:

                    return s

            i+=1

        return ans

**Question No 99 Another Count Rectangles**

Given a sorted array of distinct integers **A** and an integer **B**, find and return how many rectangles with distinct configurations can be created using elements of this array as length and breadth whose area is lesser than B.

(Note that a rectangle of 2 x 3 is different from 3 x 2 if we take configuration into view)  
  
**Problem Constraints**

1 <= **|A|** <= 100000  
1 <= A[i] <= 109  
1 <= B <= 109

Logic – As the array is sorted we can use two pointer method

* 1. We can use the 2 pointer method with one pointer i=0 and second pointer

j =n-1. If the area (A[i]\*A[j]) is less than B then we can conclude that 2\*(j-i) rectangles can be formed which have area less than B. After calculation increment i.

* 1. If area is greater than or equal to B then simply decrement j to reduce the area.

Code –

class Solution:

    # @param A : list of integers

    # @param B : integer

    # @return an integer

    def solve(self, A, B):

        i = 0

        j = len(A)-1

        cnt = 0

        while(j > i):

            if A[i]\*A[j] < B:

                cnt+= (2 \* (j-i))

                i+=1

            else:

                j-=1

        for x in A:

            if x\*x < B:

                cnt+=1

            else:

                break

        return cnt % (1000000007)

**Question No 100 Longest Consecutive Sequence**

Given an unsorted integer array **A** of size N.

Find the length of the longest set of consecutive elements from array A.

**Problem Constraints**

1 <= N <= 106

-106 <= A[i] <= 106

Logic –

1. Create a HashMap of all elements and iterate over all keys in the HashMap.
2. At every iteration check if the current element is a consecutive part of any other sequence (ie A[i]-1 exists in HashMap or not).
3. If it exists we can ignore the element as it will be counted in the next iteration or it has already been counted in previous iteration.

Code –

class Solution:

    # @param A : tuple of integers

    # @return an integer

    def longestConsecutive(self, A):

        hashA = {}

        ans = 0

        for i in range(len(A)):

            hashA[A[i]] = 1

        for i in hashA.keys():

            # Checking if the current is not a consecutive element of any other element

            if i-1 not in hashA:

                temp = i

                temp2 = 1

                # Checking for number of consecutive elements

                while temp+1 in hashA:

                    temp2+=1

                    temp+=1

                ans = max(ans, temp2)

        return ans

**Question No 101 Shaggy and distances**

Shaggy has an array **A** consisting of N elements. We call a pair of distinct indices in that array a special if elements at those indices in the array are equal.

Shaggy wants you to find a special pair such that the distance between that pair is minimum. Distance between two indices is defined as |i-j|. If there is no special pair in the array, then return -1.  
  
**Problem Constraints**

1 <= |A| <= 105

Logic –

1. Iterate over the array and store each element in the HashMap, and if an element already exists in the HashMap then take the distance between them and update the HashMap with the current index.
2. Answer will be minimum of all distances.
3. We are storing the most recent found index of each element because that will be the closest to the left of the next found index.

Code –

class Solution:

    # @param A : list of integers

    # @return an integer

    def solve(self, A):

        hashA = {}

        ans = len(A)

        flag = 0

        for i in range(len(A)):

            if A[i] not in hashA:

                hashA[A[i]] = i

            else:

                flag = 1

                ans = min(ans, i-hashA[A[i]])

                hashA[A[i]] = i

        if flag == 0:

            return -1

        return ans

**Question No 102 Sub-array with 0 sum**

Given an array of integers **A**, find and return whether the given array contains a non-empty subarray with a sum equal to 0.

If the given array contains a sub-array with sum zero return 1, else return 0.  
  
**Problem Constraints**

1 <= |A| <= 100000

-10^9 <= A[i] <= 10^9

Logic –

1. The idea is to iterate through the array, and for every element **A[i]**,  
   calculate sum of elements from 0 to i (this can simply be done as sum += arr[i]).
2. If the current sum has been seen before, then there is a zero-sum array.
3. Hashing is used to store the sum values so that we can quickly store sum and  
   find out whether the current sum is seen before or not.

Code –

import java.util.HashMap;

public class Solution {

    public int solve(int[] A) {

        long sum1 = 0;

        int N = A.length;

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

        map.put(sum1,1);

        for (int i=0;i<N;i++) {

            sum1+=A[i];

            if (map.containsKey(sum1)) {

                return 1;

            } else {

                map.put(sum1, 1);

            }

        }

        return 0;

    }

}

**Question No 103 Largest Continuous Sequence Zero Sum**

Given an array **A** of **N** integers.

Find the largest continuous sequence in a array which sums to zero.  
  
**Problem Constraints**

1 <= N <= 106

-107 <= A[i] <= 107

Logic –

There are two steps:

1. Create prefix sum array where ith index in this array represents total sum from 1 to ith index element.

2. Iterate all elements of prefix sum array and use hashing to find two elements where value at ith index == value at jth index but i != j.

3. IF element is not present in hash then fill hash table with current element.

Code –

class Solution:

    # @param A : list of integers

    # @return a list of integers

    def lszero(self, A):

        pref = []

        sum1 = 0

        pref.append(0)

        for i in A:

            sum1 += i

            pref.append(sum1)

        hashA = {}

        ansi = 0

        ansj = 0

        for i in range(len(A)+1):

            if pref[i] in hashA:

                # Longest sequence is required

                if i-hashA[pref[i]] > ansj-ansi:

                    ansi = hashA[pref[i]]

                    ansj = i

            else:

                hashA[pref[i]] = i

        return A[ansi:ansj]

**Question No 104 Distinct Numbers in Window**

You are given an array of **N** integers, **A1, A2 ,..., AN** and an integer **B**. Return the count of distinct numbers in all windows of size **B**.

Formally, return an array of size **N-B+1** where **i'th** element in this array contains number of distinct elements in sequence **Ai, Ai+1 ,..., Ai+B-1.**

**NOTE:**if **B** > **N**, return an empty array.  
  
**Input Format**

First argument is an integer array A  
Second argument is an integer B.

Logic – Hashmap and Sliding window Algorithm

1. If you have solution for window [i, i+k-1], we can build solution for window [i+1, i+k].
2. If we have a HashMap where we can maintain count of all keys and number of distinct keys, then we just have to reduce count of key A[i] and increasing count of A[i+k]. If count of some key has been reduced to zero, we need to remove that key.

Code –

class Solution:

    # @param A : list of integers

    # @param B : integer

    # @return a list of integers

    def dNums(self, A, B):

        N = len(A)

        if B>N:

            return []

        hashA = {}

        ans = []

        # Create a hash of First B elements

        for i in range(B):

            if A[i] not in hashA:

                hashA[A[i]] = 0

            hashA[A[i]] += 1

        ans.append(len(hashA))

        # Using sliding window

        for i in range(B, N):

            # add element next to window

            if A[i] not in hashA:

                hashA[A[i]] = 1

            else:

                hashA[A[i]] += 1

            # delete the first element of the window

            if hashA[A[i-B]] == 1:

                del hashA[A[i-B]]

            else:

                hashA[A[i-B]] -= 1

            ans.append(len(hashA.keys()))

        return ans

**Question No 105 Sort Array in given Order**

Given two arrays of integers **A** and **B**, Sort **A** in such a way that the relative order among the elements will be the same as those are in **B**.  
For the elements not present in **B**, append them at last in sorted order.

Return the array A after sorting from the above method.

**NOTE**: Elements of B are unique.  
  
**Problem Constraints**

1 <= length of the array A <= 100000

1 <= length of the array B <= 100000

-10^9 <= A[i] <= 10^9

Logic –

1. Loop through A, store the count of every number in a HashMap (key: number, value: count of number).
2. Loop through B, check if it is present in HashMap, put in the output array as many times in Aand remove the number from HashMap.
3. Sort the rest of the numbers in HashMap and put them in the output array.

Code –

class Solution:

    # @param A : list of integers

    # @param B : list of integers

    # @return a list of integers

    def solve(self, A, B):

        hashA = {}

        # Create Frequency Hashmap of A

        for i in range(len(A)):

            if A[i] not in hashA:

                hashA[A[i]] = 0

            hashA[A[i]] += 1

        ans = []

        for i in range(len(B)):

            # If B[i] is present in array A then append it in ans and delete it from the hashmap

            if B[i] in hashA:

                for x in range(hashA[B[i]]):

                    ans.append(B[i])

                del hashA[B[i]]

        temp = []

        # append the remaining keys of the hashmap to temp

        for key, val in hashA.items():

            for i in range(val):

                temp.append(key)

        # Sort temp and append to ans

        temp.sort()

        for i in temp:

            ans.append(i)

        return ans

**Question No 106 Colorful Number**

Given a number **A,** find if it is **COLORFUL** number or not.

If number **A** is a **COLORFUL** number return **1** else, return **0**.

What is a **COLORFUL** Number:

A number can be broken into different contiguous sub-subsequence parts.

Suppose, a number 3245 can be broken into parts like 3 2 4 5 32 24 45 324 245.

And this number is a COLORFUL number, since product of every digit of a contiguous subsequence is different.  
  
**Problem Constraints**

1 <= A <= 2 \* 109

Logic –

1. Create an array of digits of the number.
2. Apply sliding window with window lengths of 1 to N.
3. Use a HashMap to store all the products and if a product repeats, return 0.

Code –

class Solution:

    # @param A : integer

    # @return an integer

    def colorful(self, A):

        arr = []

        hashA = {}

        # Create an array of digits of the number

        while(A>0):

            arr.append(A%10)

            A//=10

        N = len(arr)

        # Apply sliding window with all lengths from 1 to N

        for k in range(1, N+1):

            prod = 1

            # Calculate product for first window

            for i in range(k):

                prod \*= arr[i]

            if prod in hashA:

                return 0

            else:

                hashA[prod] = 1

            # Calculate product for rest of the array

            for i in range(k, N):

                if arr[i] != 0 and arr[i-k] != 0:

                    prod //= (arr[i-k])

                    prod \*= arr[i]

                if prod in hashA:

                    return 0

                else:

                    hashA[prod] = 1

        return 1

**Question No 107 Check Palindrome - II**

Given a string **A** consisting of lowercase characters.

Check if characters of the given string can be rearranged to form a **palindrome**.

Return 1 if it is possible to rearrange the characters of the string **A** such that it becomes a palindrome else return 0.

**Problem Constraints**

1 <= |A| <= 105

A consists only of lower-case characters.

Logic –

1. Create a HashMap and count number of characters in the string.
2. For an even length palindrome, the frequency of all characters must be divisible by 2.
3. For an odd length palindrome, except for 1 character the frequency of all characters must be divisible by 2.

Code –

class Solution:

    # @param A : string

    # @return an integer

    def solve(self, A):

        hashA = {}

        for i in A:

            if i not in hashA:

                hashA[i] = 0

            hashA[i] += 1

        # For even number of characters

        if len(A) & 1 == 0:

            for key, val in hashA.items():

                if val % 2 != 0:

                    return 0

        # For odd number of characters

        else:

            flag = 0

            for key, val in hashA.items():

                if val % 2 != 0 and flag == 0:

                    flag = 1

                    hashA[key] -= 1

                elif val % 2 != 0 and flag == 1:

                    return 0

        return 1

**Question No 108 Interesting Array**

You have an array **A** with **N** elements. We have **two** types of operation available on this array :

1. We can split an element **B** into two elements, **C** and **D,** such that B = C + D.
2. We can merge two elements, **P** and **Q,** to one element, **R,** such that R = P ^ Q i.e., XOR of P and Q.

You have to determine whether it is possible to convert array A to size 1, **containing a single element equal to 0** after several splits and/or merge?  
  
**Problem Constraints**

1 <= N <= 100000

1 <= A[i] <= 106

Logic –

1. We can find the XOR value of all the elements in the array and if it is even then we can split it into 2 equal halves and then take their XOR to make the answer equal to zero.
2. But if the XOR value comes out to be odd then we can’t divide it in 2 equal halves so the answer will be 1.
3. So, if the XOR value of all elements is even then return “YES” else return “NO”.

Code –

class Solution:

    # @param A : list of integers

    # @return a strings

    def solve(self, A):

        ans = 0

        for i in A:

            ans ^= i

        if ans&1 ==0:

            return "Yes"

        else:

            return "No"

**Question No 109 Min XOR value**

Given an integer array **A** of **N** integers, find the pair of integers in the array which have minimum **XOR** value. Report the minimum **XOR** value.  
  
**Problem Constraints**

2 <= length of the array <= 100000  
0 <= A[i] <= 109

Logic –

The first step is to sort the array. The answer will be the minimum value of X[i] XOR X[i+1] for every i.

**Proof:**  
Let’s suppose that the answer is not X[i] XOR X[i+1], but A XOR B and there exists C in the array such as A <= C <= B.

Next is the proof that either A XOR C or C XOR B is smaller than A XOR B.

Let A[i] = 0/1 be the i-th bit in the binary representation of A  
Let B[i] = 0/1 be the i-th bit in the binary representation of B  
Let C[i] = 0/1 be the i-th bit in the binary representation of C

*This is with the assumption that all of A, B and C are padded with 0 on the left until they all have the same length*

Example: A = 169, B = 187, C = 185

A = 101010012  
B = 101110112  
C = 101110012

Let i be the leftmost (biggest) index such that A[i] differs from B[i]. There are 2 cases now:  
1) C[i] = A[i] = 0,  
then (A XOR C)[i] = 0 and (A XOR B)[i] = 1  
This implies (A XOR C) < (A XOR B)  
2) C[i] = B[i] = 1,  
then (B XOR C)[i] = 0 and (A XOR B)[i] = 1  
This implies (B XOR C) < (A XOR B)

**Time complexity:** O(N \* logN) to sort the array and O(N) to find the smallest XOR  
**Space complexity:** O(N)

Code –

import sys

class Solution:

    # @param A : list of integers

    # @return an integer

    def findMinXor(self, A):

        A.sort()

        ans = sys.maxsize

        for i in range(len(A)-1):

            ans = min(ans, A[i]^A[i+1])

        return ans

**Question No 110 Different Bits Sum Pairwise**

We define f(X, Y) as the number of different corresponding bits in the binary representation of **X** and **Y**.  
For example, f(2, 7) = 2, since the binary representation of 2 and 7 are 010 and 111, respectively. The first and the third bit differ, so f(2, 7) = 2.

You are given an array of N positive integers, A1, A2,..., AN. Find sum of f(Ai, Aj) for all pairs (i, j) such that 1 ≤ i, j ≤ N. Return the answer modulo 109+7.

**Problem Constraints**

1 <= N <= 105

1 <= A[i] <= 231 - 1

Logic – Count the number of set and unset bit for all bits from (1 to N)

1. Iterate over all 32 bits(i goes from 0 to 31) of all the elements in the array (j goes from 0 to len(A)).
2. Check the ith bit of each element of the array and count if it is 0 (x0) or 1(x1). The final count for that bit is equal to x0\*x1.
3. Similarly, we count for all bits and multiply it by 2 as 2 elements will form 2 pairs with itself. Eg 2, 7 will have 2 pairs 2,7 and 7,2.

Code –

class Solution:

    # @param A : list of integers

    # @return an integer

    def cntBits(self, A):

        cnt = 0

        for i in range(32):

            x0 = 0

            x1 = 0

            # Check the ith bit for each element

            for j in range(len(A)):

                if (A[j] & (1<<i)) != 0:

                    x1+=1

                else:

                    x0+=1

            cnt+=(x0\*x1)

        return (2\*cnt) % 1000000007

Day 45 1 problem logic left QNo 78

Done till Day 50. Day 51, 52 and 53 remaining to be done