1.Kaushik has three phases in his life - cooking, sleep and coding. Each phase of his life gives him sadness as well as happiness.
The sleep and the coding phase give him happiness whereas the cooking phase makes him sad. So, each day, he is in exactly one of the three phases.
You will be given N which denotes the number of days and K which denotes the exact number of days Kaushik needs to be happy from given N days, can you tell him in how many ways he can be happy?
Since the output value can be very large, take modulo with 1000000007(109+7).

Input format :
The first line of the input contains T, denoting the number of test cases.

The next T lines contain two space-separated integers N and K.

Output format :
For each test case, output a single integer, the number of ways modulo 109+7.

Code constraints :
$1 \leq T \leq 105$

$1 \leq N \leq 106$

$1 \leq K \leq 106$

$K \leq N$

Sample test cases :
Input 1 :
3
1 1
2 1
3 2
Output 1 :
2
4
12
Input 2 :
2
999664 514014
630049 238724
Output 2 :
472920788
16397085

Input 3 :
2
764904 1080
826435 203672
Output 3 :
256521359
502877437

SOLUTION (PHYTON):

```
def fact(xy):
    a=[1]*(10**6+1)
    f=1
    for i in range(1,10**6+1):
        f=(f*i)%m
        a[i]=f
    return(a)
def mul(x,y,m):
    return(((x%m)*(y%m))%m)
def inv(x, y, m) :

    if (y == 0) :
        return 1

    p = inv(x, y // 2, m) % m
    p = (p * p) % m

    if(y % 2 == 0) :
 return p
    else :
        return ((x * p) % m)
m=1000000007
t=int(input())
a=fact(m)
for i in range(0,t):

    n,k=[int(x) for x in input().split()]
    d=mul(a[k],a[n-k],m)
    inv1=inv(d,m-2,m)
    ans=mul(a[n],inv1,m)
    output=mul(ans,inv(2,k,m),m)
    print(output)
```

---

2. Problem Statement
You are given an integer array height of length n. There are n vertical lines drawn such that the two endpoints of the ith line are (i, 0) and (i, height[i]).
Your task is to find two lines that, along with the x-axis, form a container that can hold the maximum amount of water.
Write a program to calculate the maximum amount of water that can be stored in such a container.

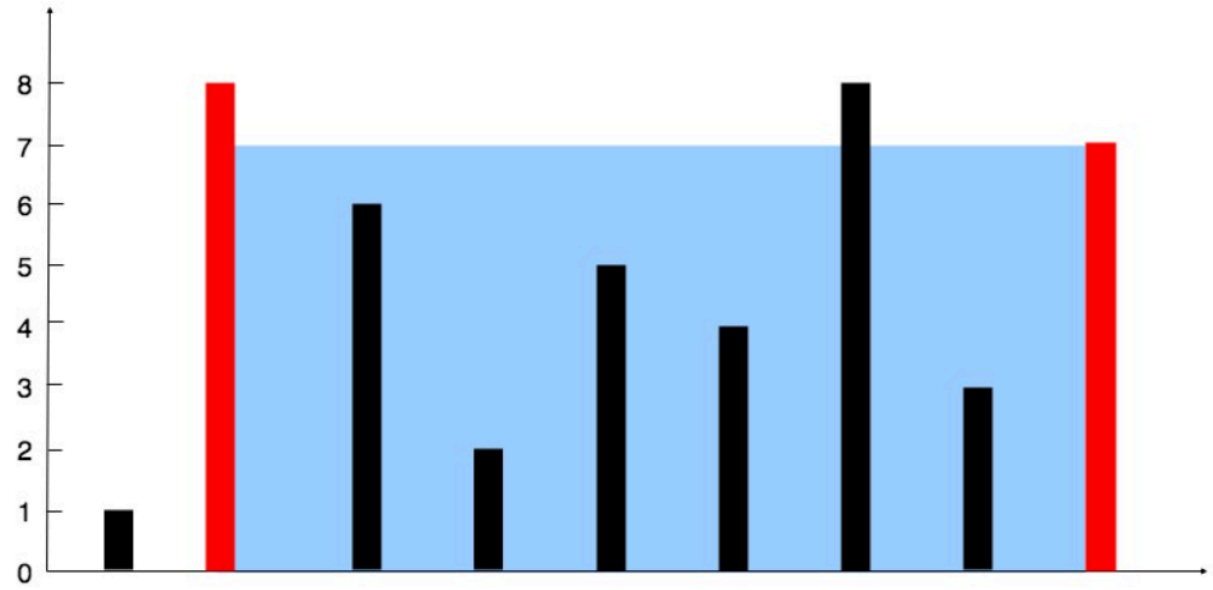
Example
Input:

height = [1, 8, 6, 2, 5, 4, 8, 3, 7]

Output:

49

Explanation:



The given vertical lines are represented by an array [1, 8, 6, 2, 5, 4, 8, 3, 7]. In this case, the max area of water (blue section) the container can contain is 49.

Input format :
The first line of input consists of an integer n, representing the number of vertical lines.

The second line of input consists of n space-separated integers, representing the heights of the vertical lines.

Output format :
The output prints the maximum amount of water that can be stored by the container formed by two lines and the x-axis.

Code constraints :
2 ≤ n ≤ 1000 (number of vertical lines)

0 ≤ height[i] ≤ 1000 (height of each line)

Sample test cases :
Input 1 :
4
1 5 4 3
Output 1 :
6
Input 2 :
5
3 1 2 4 5
Output 2 :
12

SOLUTION (JAVA):

```
public class Main {
    public static int maxArea(int[] height, int n) {
        int left = 0, right = n - 1;
        int maxWater = 0;

        while (left < right) {
            int h = Math.min(height[left], height[right]);
            int width = right - left;
            int area = h * width;
            maxWater = Math.max(maxWater, area);

            if (height[left] < height[right]) {
                left++;
            } else {
                right--;
            }
        }

        return maxWater;
```

```
  }

   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
      int n = scanner.nextInt();

      int[] height = new int[1000];
      for (int i = 0; i < n; ++i) {
         height[i] = scanner.nextInt();
      }

      int result = maxArea(height, n);
      System.out.print(result);

      scanner.close();
   }
}
```

---

3. Problem Statement
Given a sequence of numbers in an array A. The task is to break short the array such the twice minimum value in the array should be greater than the maximum value in the array. The number of values removed from the array should be minimal. The removal of elements from the array can be done from the start or from the end of the array as per the condition mentioned above.

Input format :
The first line contains a single integer n, which represents the number of integers in the list.

The second set of input contains n integers on n lines, which represents the list of elements.

Output format :
The output consists of a single integer, which represents the minimum number of elements that need to be removed from the list to make the remaining elements in the list satisfy the condition that the minimum element is at least half of the maximum element.

Sample test cases :
Input 1 :
7
2
4
5
6
8

9
1
Output 1 :
3
Input 2 :
7
3
6
4
8
5
2
10
Output 2 :
5

SOLUTION (Python):

```
def findMin(arr, low, high):

    if low > high:
        return 0


    subarr = arr[low:high+1]
    if 2 * min(subarr) <= max(subarr):

        L = 1 + findMin(arr, low + 1, high)

        R = 1 + findMin(arr, low, high - 1)


        return min(L, R)


    return 0
if __name__ == '__main__':

    arr=[]
    n=int(input())
    for x in range(0,n):
        ele=int(input())
        arr.append(ele)
    print(findMin(arr, 0, len(arr) - 1))
```

---

4. KMP(Knuth Morris Pratt) algorithm For Pattern Search

Slide the pattern over text one by one and check for a match. If a match is found, then slides by 1 again to check for subsequent matches. Given a text txt[0..n-1] and a pattern pat[0..m-1], write a function search(char pat[], char txt[]) that prints all occurrences of pat[] in txt[]. You may assume that n > m.

Input format :
The input consists of the string and the pattern.

Output format :
The output prints the index at which the pattern is found.

Sample test cases :
Input 1 :
ABABDABACDABABCABAB
ABABCABAB
Output 1 :
Found pattern at index 10

SOLUTION (JAVA)

```
import java.util.Scanner;

class KMPAlgorithm {
    public static int[] computeLPSArray(String pat) {
        int M = pat.length();
        int[] lps = new int[M];
        int length = 0;
        int i = 1;

        while (i < M) {
            if (pat.charAt(i) == pat.charAt(length)) {
                length++;
                lps[i] = length;
                i++;
            } else {
                if (length != 0) {
                    length = lps[length - 1];
                } else {
                    lps[i] = 0;
                    i++;
                }
            }
```

```
        }
        return lps;
    }

    public static void KMPSearch(String pat, String txt) {
        int M = pat.length();
        int N = txt.length();
        int[] lps = computeLPSArray(pat);
        int i = 0;
        int j = 0;

        while (i < N) {
            if (pat.charAt(j) == txt.charAt(i)) {
                i++;
                j++;
            }

            if (j == M) {
                System.out.println("Found pattern at index " + (i - j));
                j = lps[j - 1];
            } else if (i < N && pat.charAt(j) != txt.charAt(i)) {
                if (j != 0) {
                    j = lps[j - 1];
                } else {
                    i++;
                }
            }
        }
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String txt = scanner.nextLine();
        String pat = scanner.nextLine();
        KMPSearch(pat, txt);
    }
}
```

---

5. Given an array Arr[N] of N integers and a positive integer K. The task is to divide the array into two sub-arrays from right after the Kth position and slide the left sub-array of K elements to the end.

Example 1

Input:

5 -- Value of N

{10, 20, 30, 40, 50} -- Elements of Arr []

2 -- Value of K

Output:

30 40 50 10 20

Explanation:

Arr[] = {10, 20, 30, 40, 50} and K = 2 (2nd position)

Divide array from after 2nd position and add left sub-array {10,20} to the end.

So the output is 30 40 50 10 20.

Example 2

Input:

4 -- Value of N

{10, 20, 30, 40} -- Elements of Arr []

1 -- Value of K

Output:

20 30 40 10

Explanation:

Arr[] = {10, 20, 30, 40} and K=1 (1st position)

Divide array from after 1st position and add left sub-array {10} to the end.

So the output is 20 30 40 10.

Example 3

Input:

4 -- Value of N

{10, 20, 30, 40} -- Elements of Arr[]

3 -- Value of K

Output:

40 10 20 30

Explanation:

Arr[] = {10, 20, 30, 40} and K=3 (3rd position)

Divide the array from after the 3rd position and add left sub-array {10, 20, 30} to the end.

So the output is 40 10 20 30.

Input format :
The first line of input consists of a single positive integer N representing the size of Arr[].

The second line consists of N space-separated integers representing the array elements.

The third line consists of a single positive integer K.

Output format :
The output must be N integers separated by a single space character representing the modified array.

Code constraints :
$1 \leq N \leq 100$

$-100 \leq Arr[i] \leq 100$

1 ≤ K < N

Sample test cases :
Input 1 :
5
10 20 30 40 50
2
Output 1 :
30 40 50 10 20
Input 2 :
4
10 20 30 40
1
Output 2 :
20 30 40 10
Input 3 :
4
10 20 30 40
3
Output 3 :
40 10 20 30

SOLUTION (JAVA):

```
import java.util.Scanner;

public class Main {
    public static void reverseArray(int[] arr, int start, int end) {
        while (start < end) {
            int temp = arr[start];
            arr[start] = arr[end];
            arr[end] = temp;
            start++;
            end--;
        }
    }

    public static void slideAndSwap(int[] arr, int n, int k) {
        reverseArray(arr, 0, k - 1);
        reverseArray(arr, k, n - 1);
        reverseArray(arr, 0, n - 1);
    }
```

```
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        int N = scanner.nextInt();
        int[] Arr = new int[N];

        for (int i = 0; i < N; i++) {
            Arr[i] = scanner.nextInt();
        }

        int K = scanner.nextInt();

        slideAndSwap(Arr, N, K);

        for (int i = 0; i < N; i++) {
            System.out.print(Arr[i] + " ");
        }

        scanner.close();
    }
}
```

---

6. Problem Statement

Given an unsorted array A of size N that contains only non-negative integers, find a continuous sub-array that adds to a given number S and return the left and right indexes of that sub-array. In the case of multiple subarrays, return the subarray indexes that come first when moving from left to right.

Note: Both the indexes in the array should be according to 1-based indexing. You have to return an array list consisting of two elements, left and right. If no such subarray exists, return an array consisting of element -1.

Example 1

Input:

N = 5, S = 12

A[5] = {1,2,3,7,5}

Output:

2 4

Explanation:

The sum of elements from the 2nd position to the 4th position is 12.

Example 2

Input:

N = 10, S = 15

A[10] = {1,2,3,4,5,6,7,8,9,10}

Output:

1 5

Explanation:

The sum of elements from the 1st position to the 5th position is 15.

Input format :
The first line of input consists of the length of the array, N.

The second line of input consists of the sum of the sub-arrays to be searched, S.

The third line of input consists of N space-separated array elements.

Output format :
The output prints two space-separated integers, representing the start and end index positions of the subarray respectively whose sum is S.

If no such subarray exists, return an array consisting of element -1.

Refer to the sample output for the formatting specifications.

Code constraints :
In this scenario, the test cases fall under the following constraints:

1 ≤ N ≤ 105

1 ≤ Ai ≤ 109

Sample test cases :
Input 1 :
5
12
1 2 3 7 5
Output 1 :
2 4
Input 2 :
10
15
1 2 3 4 5 6 7 8 9 10
Output 2 :
1 5
Input 3 :
3
22
1 4 5
Output 3 :
-1

SOLUTION (JAVA):

```
import java.util.Scanner;

class Main {
    static void subarraySum(int[] arr, int n, int s, int[] result) {
        int sum = 0;
```

```
        result[0] = 0;
        result[1] = -1;
        int i = 0, st = 0;

        while (i < n) {
            sum += arr[i];

            while (sum > s) {
                sum -= arr[st];
                st++;
            }

            if (sum == s) {
                result[0] = st + 1;
                result[1] = i + 1;
                return;
            }
            i++;
        }
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int l = sc.nextInt();
        int sum = sc.nextInt();
        int[] arr = new int[l];

        for (int i = 0; i < l; i++) {
            arr[i] = sc.nextInt();
        }

        int[] result = new int[2];
        subarraySum(arr, l, sum, result);

        if (result[1] == -1) {
            System.out.print("-1");
        } else {
            System.out.print(result[0] + " " + result[1]);
        }
    }
}
```

---

7. There is a bag with three types of gemstones: Ruby of type R, Garnet of type G, and Topaz of type T. Write a program to find the total number of possible arrangements to make a series of gemstones where no two gemstones of the same type are adjacent to each other.

Example 1

Input:

1-Count of R i.e. Ruby

1-Count of G i.e. Garnet

0-Count of T i.e. Topaz

Output:

2

Explanation:

Arrangements are RG and GR.

Example 2

Input:

1-Count of R i.e. Ruby

1-Count of G i.e. Garnet

1-Count of T i.e. Topaz

Output:

6

Explanation:

Arrangements are RGTR, GRTR, RGRT, RTGR, RTRG and TRGR.

Input format :
The first line of input contains an integer - count of R.

The second line of input contains an integer - count of G.

The third line of input contains an integer - count of T.

Output format :
The output prints the number of arrangements.

Sample test cases :
Input 1 :
1
1
0
Output 1 :
2
Input 2 :
1
1
1
Output 2 :
6

SOLUTION (JAVA):

```
class Main {
    static int countWays(int p, int q, int r, int last) {
        if (p < 0 || q < 0 || r < 0)
            return 0;

        if (p == 1 && q == 0 && r == 0 && last == 0)
            return 1;
        if (p == 0 && q == 1 && r == 0 && last == 1)
            return 1;
        if (p == 0 && q == 0 && r == 1 && last == 2)
            return 1;

        if (last == 0)
            return countWays(p - 1, q, r, 1) + countWays(p - 1, q, r, 2);
        if (last == 1)
```

```
            return countWays(p, q - 1, r, 0) + countWays(p, q - 1, r, 2);
        if (last == 2)
            return countWays(p, q, r - 1, 0) + countWays(p, q, r - 1, 1);

        return 0;
    }

    static int neo(int p, int q, int r) {
        return countWays(p, q, r, 0) +
                countWays(p, q, r, 1) +
                countWays(p, q, r, 2);
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int p = sc.nextInt();
        int q = sc.nextInt();
        int r = sc.nextInt();
        System.out.print(neo(p, q, r));
    }
}
```

---

8. Problem Statement

Given two sorted arrays nums1 and nums2 of size m and n respectively, return the median of the two sorted arrays.

Example

Input

2

1 2

2

3 4

Output

2.5

Explanation

merged array = [1,2,3,4] and median is (2 + 3) / 2 = 2.5

Input format :
The first line contains an integer m, representing the size of the first sorted array nums1.

The second line contains m integers separated by a space, representing the elements of the first sorted array nums1 in non-descending order.

The third line contains an integer n, representing the size of the second sorted array nums2.

The fourth line contains n integers separated by a space, representing the elements of the second sorted array nums2 in non-descending order.

Output format :
The output contains a double value, representing the median of the two sorted arrays.

If the combined length of the two arrays is even, then the median is the average of the two middle elements.

Otherwise, it is the middle element. The output should be accurate up to one decimal place.

Code constraints :
$1 \leq m \leq 1000$

$1 \leq n \leq 1000$

$-106 \leq nums1[i], nums2[i] \leq 106$

Sample test cases :
Input 1 :
2
1 2
2
3 4
Output 1 :
2.5
Input 2 :
4
1 -3 5 7

4
2 4 -6 8
Output 2 :
-1.0

SOLUTION (JAVA):

```
class Main{

    static double findMedianOfSortedArrays(int nums1[], int m, int nums2[], int n) {
        int N1 = m;
        int N2 = n;
        if (N1 < N2) return findMedianOfSortedArrays(nums2, n, nums1, m);  // Make sure A2 is the shorter one.

        int lo = 0, hi = N2 * 2;
        while (lo <= hi) {
            int mid2 = (lo + hi) / 2;   // Try Cut 2
            int mid1 = N1 + N2 - mid2;  // Calculate Cut 1 accordingly

            double L1 = (mid1 == 0) ? Integer.MIN_VALUE : nums1[(mid1-1)/2];          // Get L1, R1, L2, R2 respectively
            double L2 = (mid2 == 0) ? Integer.MIN_VALUE : nums2[(mid2-1)/2];
            double R1 = (mid1 == N1 * 2) ? Integer.MAX_VALUE : nums1[(mid1)/2];
            double R2 = (mid2 == N2 * 2) ? Integer.MAX_VALUE : nums2[(mid2)/2];

            if (L1 > R2) lo = mid2 + 1;                // A1's lower half is too big; need to move C1 left (C2 right)
            else if (L2 > R1) hi = mid2 - 1;           // A2's lower half too big; need to move C2 left.
            else return (Math.max(L1,L2) + Math.min(R1, R2)) / 2;        // Otherwise, that's the right cut.
        }
        return -1 ;
    }

    public static void main(String a[]){
        Scanner sc = new Scanner(System.in);
        int m = sc.nextInt() ;
        int nums1[] = new int [m] ;
        for(int i = 0 ; i < m ; i++)
            nums1[i] = sc.nextInt() ;
        int n = sc.nextInt() ;
        int nums2[] = new int[n] ;
        for(int i = 0 ; i < n ; i++)
            nums2[i] = sc.nextInt() ;
```

```
        System.out.print(findMedianOfSortedArrays(nums1, m, nums2, n));
    }
}
```

---

9. Problem Statement

Write a program using recursion to return the number of ways to express x as a sum of nth powers of unique natural numbers.

Example 1

Input:

10

2

Output:

1

Explanation:

x = 10

n = 2

10 = 12 + 32, hence we have only 1 possibility.

Example 2

Input:

100

2

Output:

3

Explanation:

x = 100

n = 2

100 = 102 OR 62 + 82 OR 12 + 32 + 42 + 52 + 72, hence total 3 possibilities.

Function Specifications: int getAllWays(int, int, int)

Input format :
The first line of input contains the integer x.

The second line contains the integer n.

Output format :
The output is the number of ways to express x as a sum of n-th powers of unique natural numbers in a single line.

Sample test cases :
Input 1 :
100
2
Output 1 :
3
Input 2 :
10
2
Output 2 :
1

SOLUTION (JAVA):

```
import java.util.Scanner ;

class Main {

    public static int getAllWays(int remainingSum, int power, int base){

            int result = (int)Math.pow(base, power);

            if (result == remainingSum)
            return 1 ;

            if (result > remainingSum)
            return 0 ;

            // Two recursive calls one to include current base's
            // power in sum another to exclude
            return getAllWays(remainingSum - result, power, base + 1) +
                    getAllWays(remainingSum, power, base + 1) ;
    }
        // Driver Code
    public static void main(String[] args){

        Scanner sc = new Scanner(System.in) ;
            int x = sc.nextInt() ;
            int n = sc.nextInt() ;
            System.out.print( getAllWays(x, n, 1) ) ;
    }
}
```

---

10. Problem Statement

You are embarking on a circular journey through several gas stations, each stocked with a specific amount of gas and requiring a certain amount of gas to reach the next station.

Your mission is to find the gas station where you can initiate your journey in such a way that you can circumnavigate the entire route clockwise without running out of gas. If a feasible starting gas station exists, provide its index; otherwise, indicate that no such station can fulfill the requirement by returning -1.

Given two integer arrays gas and cost, return the starting gas station's index if you can travel around the circuit once in the clockwise direction, otherwise return -1.

Example 1

Input:

gas = [1, 2, 3, 4, 5]

cost = [3, 4, 5, 1, 2]

Output:

3

Explanation:

Start at station 3 (index 3) and fill up with 4 units of gas. Your tank = 0 + 4 = 4

Travel to station 4. Your tank = 4 - 1 + 5 = 8

Travel to station 0. Your tank = 8 - 2 + 1 = 7

Travel to station 1. Your tank = 7 - 3 + 2 = 6

Travel to station 2. Your tank = 6 - 4 + 3 = 5

Travel to station 3. The cost is 5. Your gas is just enough to travel back to station 3.

Therefore, return 3 as the starting index.

Example 2

Input

gas =[2, 3, 4]

cost = [3, 4]

Output:

-1

Explanation:

You can't start at station 0 or 1, as there is not enough gas to travel to the next station.

Let's start at station 2 and fill up with 4 units of gas. Your tank = 0 + 4 = 4

Travel to station 0. Your tank = 4 - 3 + 2 = 3

Travel to station 1. Your tank = 3 - 3 + 3 = 3

You cannot travel back to station 2, as it requires 4 units of gas but you only have 3.

Therefore, you can't travel around the circuit once no matter where you start.

Input format :
The first line of input consists of an integer N, representing the number of gas stations.
The second line of input consists of N space-separated integers representing the amount of gas available at each station.
The third line of input consists of N space-separated integers representing the cost of traveling from each station to the next.

Output format :
The output prints an integer representing the index of the starting gas station that allows you to complete the circuit, or -1 if no such station exists.

Code constraints :
$1 \leq N \leq 10^5$
$0 \leq gas[i], cost[i] \leq 10^4$

Sample test cases :

Input 1 :

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

Output 1 :

```
3
```

Input 2 :

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

Output 2 :

```
-1
```

SOLUTION (JAVA):

```
import java.util.Scanner;

class GasStation {
    public static int canCompleteCircuit(int[] gas, int[] cost) {
        int totalGas = 0; // Total gas available
        int currentGas = 0; // Gas in the current segment
        int startStation = 0; // Starting station index

        for (int i = 0; i < gas.length; ++i) {
            totalGas += gas[i] - cost[i]; // Calculate total gas balance

            currentGas += gas[i] - cost[i]; // Add gas of the current station

            // If the gas in the current segment becomes negative, reset and move to the next station
            if (currentGas < 0) {
                currentGas = 0;
                startStation = i + 1;
            }
        }

        // If total gas balance is non-negative, a solution is possible
        if (totalGas >= 0) {
            return startStation;
        } else {
            return -1;
        }
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int n = scanner.nextInt();

        int[] gas = new int[n];
```

```
        int[] cost = new int[n];

        for (int i = 0; i < n; ++i) {
            gas[i] = scanner.nextInt();
        }

        for (int i = 0; i < n; ++i) {
            cost[i] = scanner.nextInt();
        }

        int result = canCompleteCircuit(gas, cost);
        System.out.println(result);

        scanner.close();
    }
}
```

---

11. Given a string s and a dictionary of words dict, add spaces in s to construct a sentence where each word is a valid dictionary word.
Return all such possible sentences.

For example, given

s = "snakesandladder",

dict = ["snake", "snakes", "and", "sand", "ladder"]

A solution is ["snakes and ladder", "snake sand ladder"].

Input format :
The first line contains an integer T, denoting the number of test cases.
Every test case has 3 lines.
The first line contains an integer N, number of words in the dictionary.
The second line contains N strings denoting the words of the dictionary.
The third line contains a string s.
Output format :
For each test case, print all possible strings.
Code constraints :
1 <= T <= 100
1 <= N <= 20
1 <= Length of each word in dictionary <= 15
1 <= Length of s <= 1000
Sample test cases :
Input 1 :

```
1
5
snake snakes and sand ladder
snakesandladder
```

Output 1 :

```
snake sand ladder
snakes and ladder
```

SOLUTION (JAVA):

```
import java.util.*;

class DictionaryWord {

   public static void main(String[] args) {
      Scanner s=new Scanner(System.in);
      int t=s.nextInt();
      for (int j = 0; j < t; j++) {
         int n = s.nextInt();
         String[] dict = new String[n];
         for (int i = 0; i < n; i++) {
            dict[i] = s.next();
         }
         String str = s.next();
         findSol(str, str.length(), "", dict);
      }
   }
   public static void findSol(String str,int n,String res,String[] dict)
   {
      for (int i = 1; i <=n; i++) {
         String s=str.substring(0,i);
         if(Arrays.asList(dict).contains(s))
         {
            if (i==n)
            {
               res=res+s;
                System.out.println(res);
                return;
            }
            String temp=str.substring(i,n);
            findSol(temp,temp.length(),res+s+" ",dict);
         }
      }
   }
}
```

12.

Love for prime never ends. Salman comes with a problem with primes where he will be giving you X prime numbers and Q queries of form A_i, B_i, for each query print the number of integers between A_i and B_i (both inclusive) that are divisible by at least one of the X given primes.

Input format :

The first line consists of two space-separated integers - X and Q.

The second line consists of X primes.

The following Q lines, each contain two space-separated integers - A_i, B_i.

Output format :

Print Q lines, denoting the answer to each of the Q queries.

Code constraints :

$1 \leq X \leq 10$

$1 \leq Q \leq 100$

$1 \leq A \leq B \leq 10^9$

Each prime $\leq 10^7$

Sample test cases :

Input 1 :

```
6 2
2 3 5 7 11 37
1 1
1 2
```

Output 1 :

```
0
1
```

SOLUTION (Python):

```
from itertools import combinations
from math import gcd
k,t=map(int,input().split())
l=list(map(int,input().split()))
def lcm(m):
    lc=m[0]
    for i in m[1:]:
        lc=lc*i//gcd(i,lc)
    return lc
def div(n):
    total=0
    for i in range(1,k+1):
        for j in combinations(l,i):
            c=n//lcm(j)
            if(i%2==0):
                total-=c
            else:
                total+=c
    return total
for i in range(t):
```

```
x,y=map(int,input().split())
print(div(y)-div(x-1))
```

---

13. Given a matrix of dimension m * n where each cell in the matrix can have values 0, 1 or 2 which has the following meaning:
0: Empty cell

1: Cells have fresh oranges

2: Cells have rotten oranges

We have to determine what is the minimum time required so that all the oranges become rotten.

A rotten orange at index [i,j] can rot other fresh orange at indexes [i-1, j], [i+1, j], [i, j-1], [i, j+1] (up, down, left and right). If it is impossible to rot every orange then simply print "All oranges cannot rot".

Input format :
The first line of input consists of two space-separated integers, m and n.
The following m lines consist of n elements representing the matrix.
Output format :
The output prints the minimum time required so that all the oranges become rotten.
If it is impossible to rot every orange, print "All oranges cannot rot".
Code constraints :
1 ≤ n, m ≤ 500
Sample test cases :
Input 1 :
```
3 5
2 1 0 2 1
1 0 1 2 1
1 0 0 2 1
```
Output 1 :
```
All oranges can become rotten in 2 time frames.
```

Input 2 :
```
3 5
2 1 0 2 1
0 0 1 2 1
1 0 0 2 1
```
Output 2 :
```
All oranges cannot rot
```

SOLUTION (JAVA):

```
import java.util.Queue;
import java.util.*;

class RotOrange
```

```
{
    public static int R;
    public static int C;
     static class Ele
     {
           int x = 0;
           int y = 0;
           Ele(int x,int y)
           {
                 this.x = x;
                 this.y = y;
           }
     }


     static boolean isValid(int i, int j)
     {
           return (i >= 0 && j >= 0 && i < R && j < C);
     }


     static boolean isDelim(Ele temp)
     {
           return (temp.x == -1 && temp.y == -1);
     }


     static boolean checkAll(int arr[][])
     {
           for (int i=0; i<R; i++)
                 for (int j=0; j<C; j++)
                       if (arr[i][j] == 1)
                             return true;
           return false;
     }


     static int rotOranges(int arr[][])
     {

           Queue<Ele> Q=new LinkedList<>();
           Ele temp;
           int ans = 0;

           for (int i=0; i < R; i++)
```

```
            for (int j=0; j < C; j++)
                  if (arr[i][j] == 2)
                        Q.add(new Ele(i,j));


            Q.add(new Ele(-1,-1));


            while(!Q.isEmpty())
            {

                  boolean flag = false;


                  while(!isDelim(Q.peek()))
                  {
                        temp = Q.peek();


                        if(isValid(temp.x+1, temp.y) &&
arr[temp.x+1][temp.y] == 1)
                        {
                              if(!flag)
                              {

                                    ans++;
                                    flag = true;
                              }

                              arr[temp.x+1][temp.y] = 2;


                              temp.x++;
                              Q.add(new Ele(temp.x,temp.y));


                              temp.x--;
                        }


                        if (isValid(temp.x-1, temp.y) &&
arr[temp.x-1][temp.y] == 1)
                        {
                                    if (!flag)
                                    {
```

```
                                    ans++;
                                    flag = true;
                               }
                               arr[temp.x-1][temp.y] = 2;
                               temp.x--;
                               Q.add(new Ele(temp.x,temp.y));
                               temp.x++;
                          }


                          if (isValid(temp.x, temp.y+1) &&
arr[temp.x][temp.y+1] == 1) {
                               if(!flag)
                               {
                                    ans++;
                                    flag = true;
                               }
                               arr[temp.x][temp.y+1] = 2;
                               temp.y++;
                               Q.add(new Ele(temp.x,temp.y));
                               temp.y--;
                            }


                          if (isValid(temp.x, temp.y-1) &&
arr[temp.x][temp.y-1] == 1)
                          {
                               if (!flag)
                               {
                                    ans++;
                                    flag = true;
                               }
                               arr[temp.x][temp.y-1] = 2;
                               temp.y--;
                               Q.add(new Ele(temp.x,temp.y));
                          }
                          Q.remove();

                    }
                    Q.remove();


                    if (!Q.isEmpty())
                    {
                          Q.add(new Ele(-1,-1));
```

```
                }


            }


            return (checkAll(arr))? -1: ans;

        }


        public static void main(String[] args)
        {
            Scanner s=new Scanner(System.in);
            R=s.nextInt();
            C=s.nextInt();
            int arr[][]=new int[R][C];
            for(int i=0;i<R;i++){
                for(int j=0;j<C;j++){
                    arr[i][j]=s.nextInt();
                }
            }

             int ans = rotOranges(arr);
             if(ans == -1)
                  System.out.println("All oranges cannot rot");
             else
                  System.out.println("All oranges can become rotten in
"+ans+" time frames.");
        }

}
```
—--
-
14.

Problem Statement

You are given an integer array nums that is sorted in non-decreasing order.

Your task is to remove some duplicates from the array in-place such that each unique element appears at most twice. The relative order of the elements should be kept the same.

Your goal is to modify the input array nums in such a way that the first k elements of nums hold the final result, where k is the count of elements after removing the duplicates. It does not matter what you leave beyond the first k elements.

Return k after placing the final result in the first k slots of nums.

Example 1

Input:

nums = [1, 1, 1, 2, 2, 3]

Output:

5

Explanation:

Your function should return k = 5, with the first five elements of nums being 1, 1, 2, 2, and 3 respectively.

Example 2

Input:

nums = [0, 0, 1, 1, 1, 1, 2, 3, 3]

Output:

7

Explanation:

Your function should return k = 7, with the first seven elements of nums being 0, 0, 1, 1, 2, 3, and 3 respectively.

Input format :

The first line of input consists of an integer n, representing the number of elements in the array.
The second line of input consists of n space-separated integers, representing the elements of the array.

Output format :

The output prints the new length k of the array, after removing duplicates.

Sample test cases :

Input 1 :

```
9
0 0 1 1 1 1 2 3 3
```

Output 1 :

```
7
```

Input 2 :

```
6
1 1 1 2 2 3
```

Output 2 :

```
5
```

SOLUTION (JAVA)

```
import java.util.Scanner;

public class Main {
    public static int removeDuplicates(int[] nums, int n) {
        int c = 0;
```

```
        for (int i = 0; i < n; i++) {
            if (i < n - 2 && nums[i] == nums[i + 2]) {
                continue;
            } else {
                nums[c++] = nums[i];
            }
        }
        return c;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int n = scanner.nextInt();

        int[] nums = new int[100000];
        for (int i = 0; i < n; ++i) {
            nums[i] = scanner.nextInt();
        }


        int result = removeDuplicates(nums, n);
        System.out.print(result);

        scanner.close();
    }
}
```

15.