WEEK 4

TASK 1 :

AIM:

To determine whether the two halves of a given even-length string are alike by checking if both halves contain an equal number of vowels.

ALGORITHM :

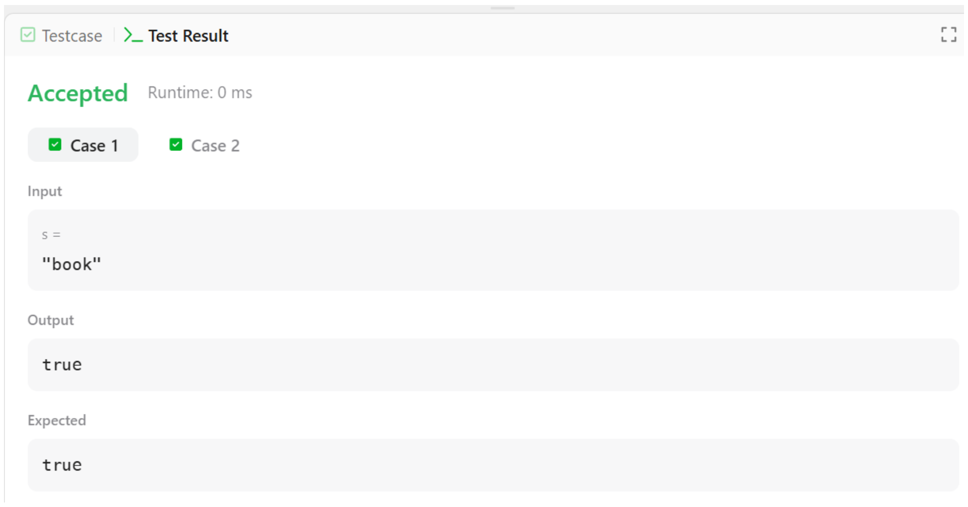
1. start
2. Read the input string s.
3. Find the midpoint of the string using mid = length / 2.
4. Initialize two counters countA and countB to zero.
5. Define a string containing all vowels (both uppercase and lowercase).
6. Traverse from index 0 to mid - 1:
7. If the character in the first half is a vowel, increment countA.
8. If the corresponding character in the second half is a vowel, increment countB.
9. Compare countA and countB.
10. If both counts are equal, return true; otherwise, return false.
11. stop

PROGRAM :

```
class Solution {

   public boolean halvesAreAlike(String s) {

      int mid = s.length() / 2;

      int countA = 0, countB = 0;

      String vowels = "aeiouAEIOU";

      for (int i = 0; i < mid; i++) {

         if (vowels.indexOf(s.charAt(i)) != -1) {

            countA++;

         }

         if (vowels.indexOf(s.charAt(i + mid)) != -1) {

            countB++;

         }

      }
```

```
        return countA == countB;
    }
}
```

Output:



Testcase | Test Result

Accepted Runtime: 0 ms

Case 1 Case 2

Input

```
s =
"book"
```

Output

```
true
```

Expected

```
true
```

RESULT :

The program successfully determines whether both halves of the string have the same number of vowels and prints true or false accordingly.

Task2 :

AIM:

To check whether a given string is a **lapindrome**. A string is said to be a lapindrome if, after splitting it into two halves (ignoring the middle character if the length is odd), both halves contain the same characters with the same frequency.

ALGORITHM :

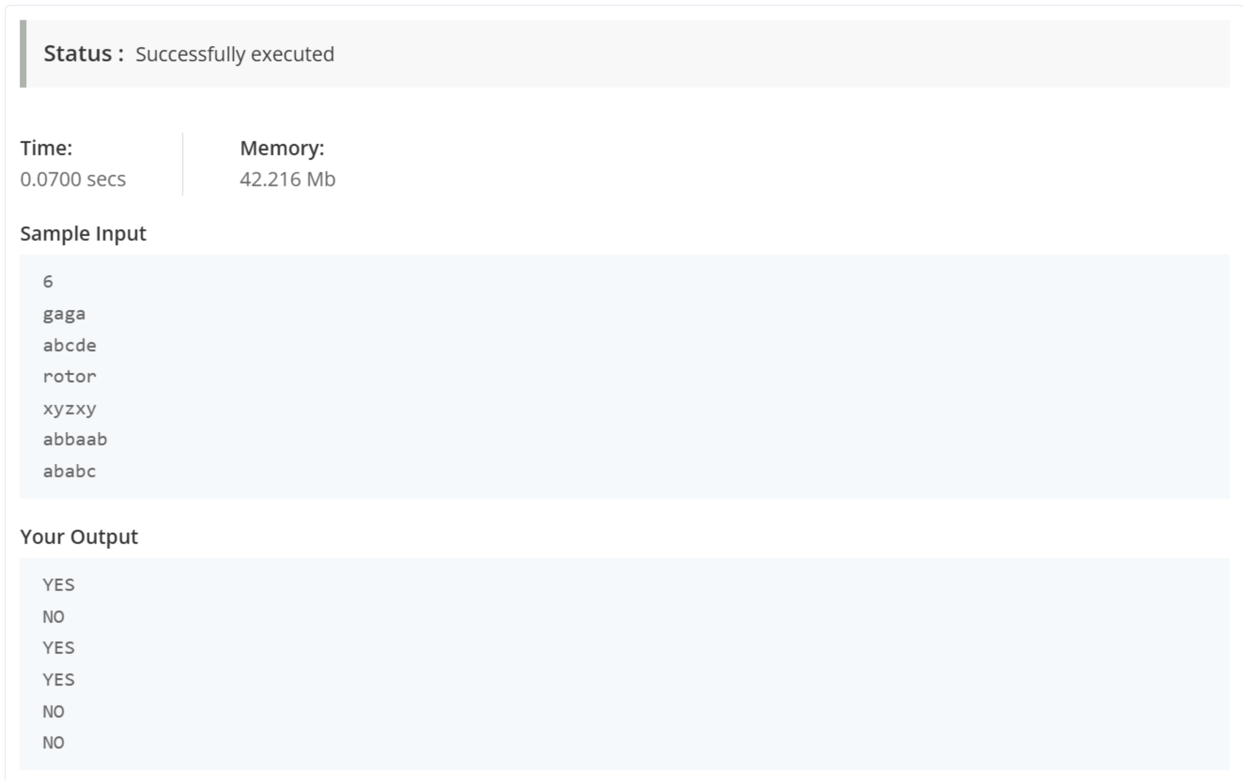
```
1. Start.
2. Read the number of test cases T.
3. For each test case:
    a. Read the string S.
    b. Find the length n of the string.
    c. Calculate the middle index mid = n / 2.
4. Create two frequency arrays of size 26 (for lowercase English letters).
5. Count the frequency of characters in the first half of the string.
6. Count the frequency of characters in the second half of the string:
    a. If the string length is odd, skip the middle character.
7. Compare both frequency arrays:
    a. If all frequencies match, the string is a lapindrome.
    b. Otherwise, it is not a lapindrome.
8. Print "YES" if the string is a lapindrome, else print "NO".
9. Stop
```

PROGRAM :

```
import java.util.*;

import java.lang.*;

import java.io.*;

class Codechef {

    public static void main (String[] args) throws java.lang.Exception {

        Scanner sc = new Scanner(System.in);

        int T = sc.nextInt();

        while (T-- > 0) {

            String s = sc.next();
            int n = s.length();

            int[] freq1 = new int[26];

            int[] freq2 = new int[26];

            int mid = n / 2;


            for (int i = 0; i < mid; i++) {

                freq1[s.charAt(i) - 'a']++;

            }

            for (int i = (n % 2 == 0 ? mid : mid + 1); i < n; i++) {

                freq2[s.charAt(i) - 'a']++;

           }

            boolean isLapindrome = true;

            for (int i = 0; i < 26; i++) {

                if (freq1[i] != freq2[i]) {

                    isLapindrome = false;

                    break;

                }

            }

          if (isLapindrome)

                System.out.println("YES");

            else

                System.out.println("NO");

        }
```

```
        sc.close();

    }

}
```

Output:



```
Status : Successfully executed

Time:           Memory:
0.0700 secs     42.216 Mb

Sample Input
6
gaga
abcde
rotor
xyzxy
abbaab
ababc

Your Output
YES
NO
YES
YES
NO
NO
```

Result:

To check whether the given string is a lapindrome or not has successfully executed

TASK 3 :

AIM :

To compare two lists of integers (triplets) element by element and calculate the score of each list based on comparison rules.

ALGORITHM :

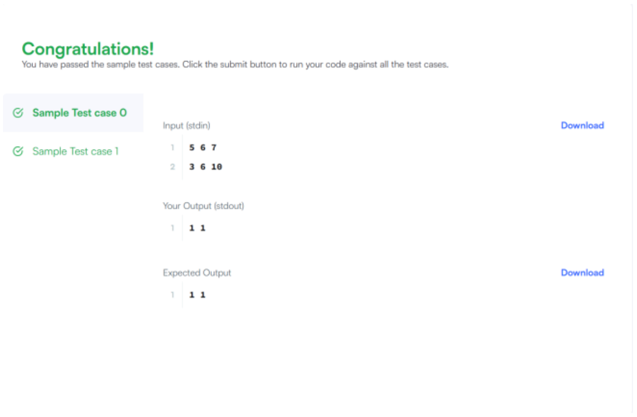
1. Initialize two integer variables scoreA and scoreB to 0.
2. Iterate through the elements of both lists a and b from index 0 to 2.
3. For each index:
4. If a[i] > b[i], increment scoreA by 1.
5. If a[i] < b[i], increment scoreB by 1.
6. If a[i] == b[i], do nothing.
7. Store scoreA and scoreB in a list.
8. Return the list containing the final scores.

PROGRAM :

```
public class Solution {

  public static void main(String[] args) throws IOException {

    BufferedReader bufferedReader = new BufferedReader(new
InputStreamReader(System.in));

    BufferedWriter bufferedWriter = new BufferedWriter(new
FileWriter(System.getenv("OUTPUT_PATH")));

    List<Integer> a = Stream.of(bufferedReader.readLine().replaceAll("\\s+$", "").split(" "))

      .map(Integer::parseInt)

      .collect(toList());

    List<Integer> b = Stream.of(bufferedReader.readLine().replaceAll("\\s+$", "").split(" "))

      .map(Integer::parseInt)

      .collect(toList());
```

```
        List<Integer> result = Result.compareTriplets(a, b);

        bufferedWriter.write(

            result.stream()

                .map(Object::toString)

                .collect(joining(" "))

            + "\n"

        );

        bufferedReader.close();

        bufferedWriter.close();

    }

}
```

OUTPUT :



Congratulations!
You have passed the sample test cases. Click the submit button to run your code against all the test cases.

Sample Test case 0

Sample Test case 1

Input (stdin) Download

```
5 6 7
3 6 10
```

Your Output (stdout)

```
1 1
```

Expected Output Download

```
1 1
```

RESULT :

The program successfully compares the two triplets and outputs the final scores of both participants as an integer list.

AIM : 4

To check whether an integer array contains any duplicate elements.

ALGORITHM:

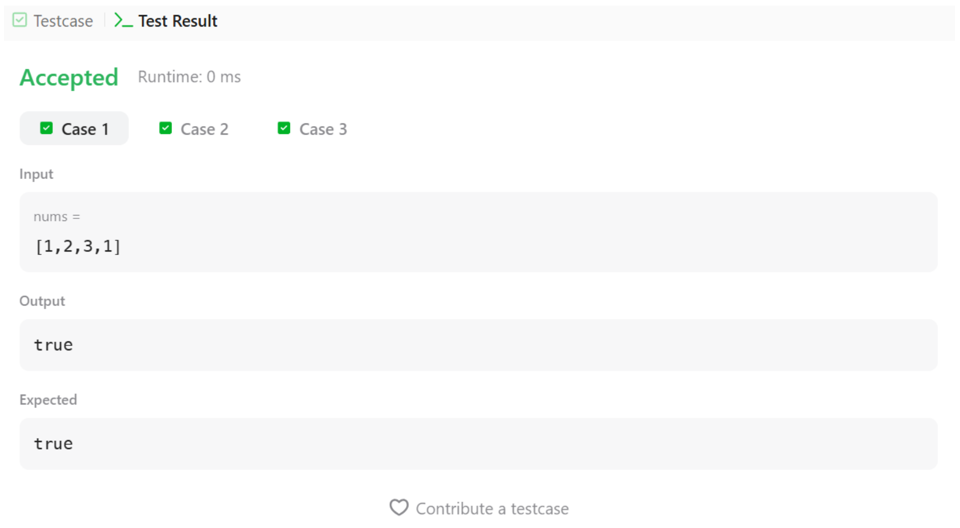
1. Create an empty HashSet to store visited elements.
2. Traverse each element in the given array.
3. Check if the element already exists in the HashSet.
4. If it exists, return true (duplicate found).
5. If it does not exist, add the element to the HashSet.
6. If the loop completes without finding duplicates, return false.

PROGRAM :

```
import java.util.HashSet;

class Solution {

    public boolean containsDuplicate(int[] nums) {

        HashSet<Integer> seen = new HashSet<>();

        for (int num : nums) {

            if (seen.contains(num)) {

                return true;

            }

            seen.add(num);

        }

        return false;

    }

}
```

```
public class Main {

    public static void main(String[] args) {

        Solution sol = new Solution();

        int[] nums1 = {1, 2, 3, 1};

        int[] nums2 = {1, 2, 3, 4};

        int[] nums3 = {1, 1, 1, 3, 3, 4, 3, 2, 4, 2};

        System.out.println(sol.containsDuplicate(nums1));

        System.out.println(sol.containsDuplicate(nums2));

        System.out.println(sol.containsDuplicate(nums3));

    }

}
```

OUTPUT :



Accepted Runtime: 0 ms

Input

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

Output

```
true
```

Expected

```
true
```

RESULT :

The program correctly identifies arrays with duplicate values and prints true if duplicates exist, otherwise prints false**.**

TASK 5:

AIM :

To convert a given time from 12-hour AM/PM format into 24-hour (military) time format.

ALGORITHM:

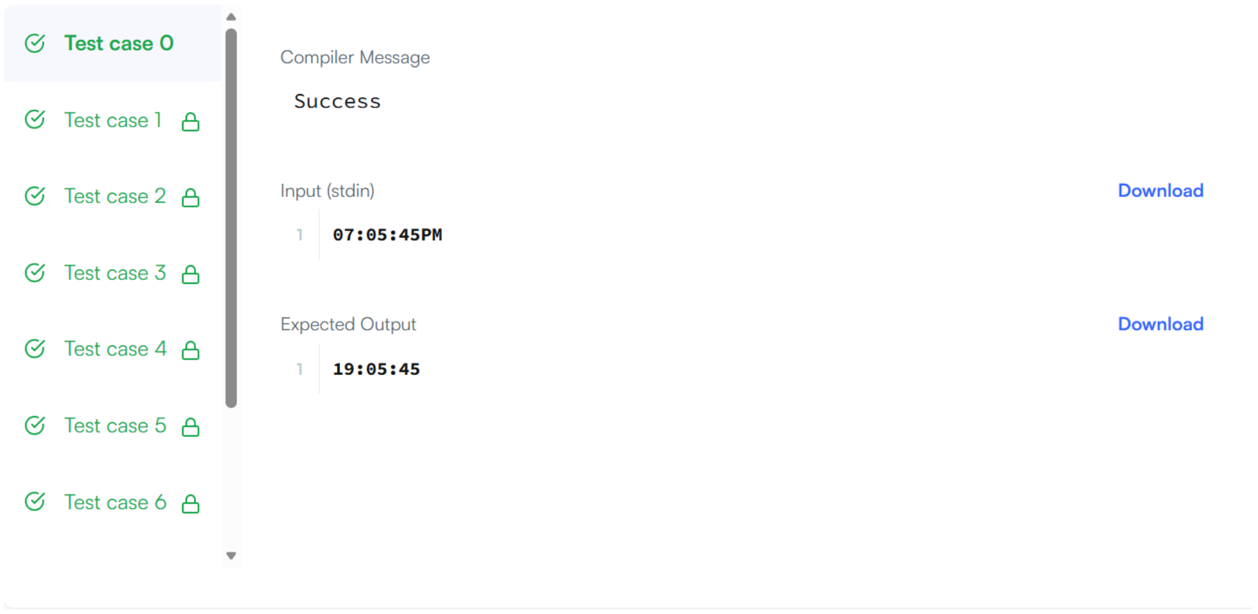
1. Read the input time string in 12-hour format.
2. Extract the hour part, minutes–seconds part, and AM/PM indicator.
3. If the time is AM and the hour is 12, change the hour to 00.
4. If the time is PM and the hour is not 12, add 12 to the hour.
5. Format the hour to two digits and combine it with minutes and seconds.
6. Display the converted 24-hour time.

PROGRAM :

```
import java.util.Scanner;

public class Solution {

   public static String timeConversion(String s) {

      String ampm = s.substring(8); // "AM" or "PM"

      int hour = Integer.parseInt(s.substring(0, 2));

      String minutesSeconds = s.substring(2, 8); // ":MM:SS"

      if (ampm.equals("AM")) {

         if (hour == 12) {

            hour = 0; // midnight case

         }

      } else { // PM case
```

```
            if (hour != 12) {

                hour += 12; // add 12 for afternoon/evening

            }


        String hourStr = String.format("%02d", hour);

        return hourStr + minutesSeconds;

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        String s = sc.nextLine(); // read input string

        System.out.println(timeConversion(s));

        sc.close();

    }

}
```

OUTPUT :



Compiler Message

```
Success
```

Input (stdin)

```
07:05:45PM
```

Expected Output

```
19:05:45
```

RESULT :

The program successfully converts the given 12-hour format time into its equivalent 24-hour format and prints the result.

TASK 6 :

AIM :

To move all zero elements in the given array to the end while maintaining the relative order of non-zero elements.

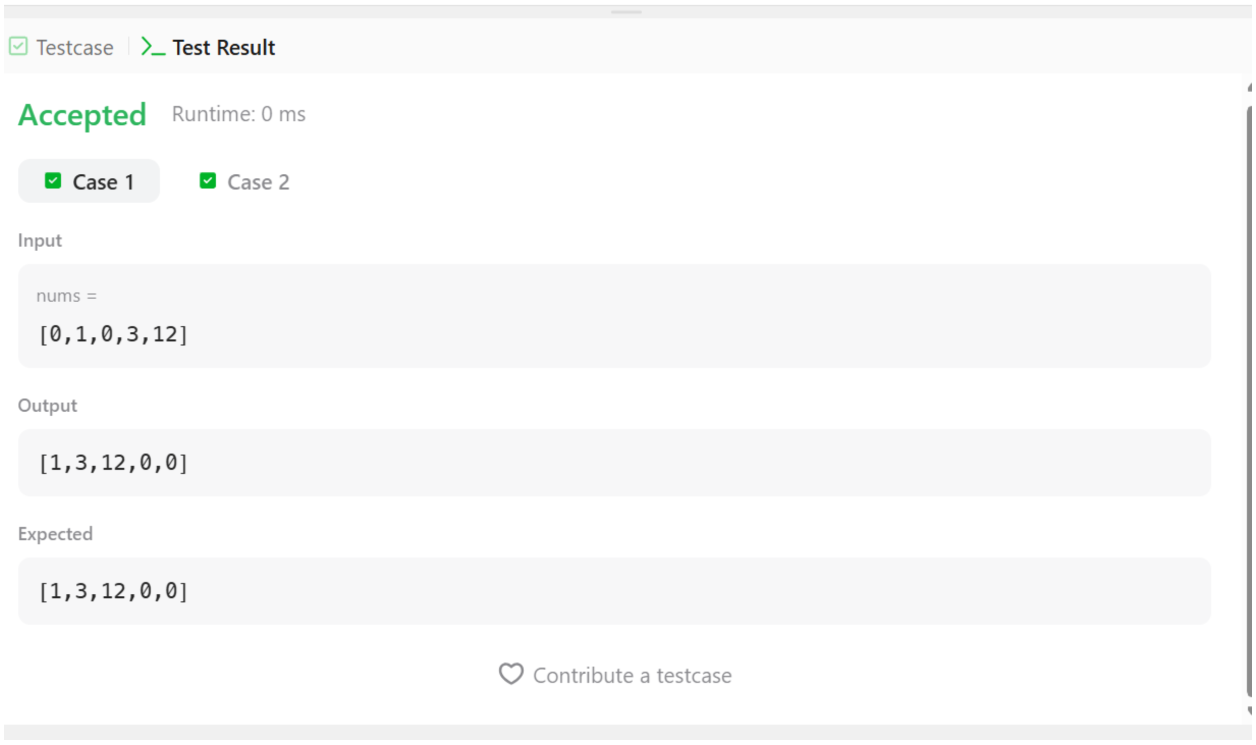
ALGORITHM:

1. Initialize a pointer j to 0 to track the position of non-zero elements.
2. Traverse the array using index i.
3. If nums[i] is not zero, swap it with nums[j].
4. Increment j after each swap.
5. Continue until all elements are processed.

PROGRAM :

```
class Solution {

public void moveZeroes(int[] nums) {

int j = 0;

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

if (nums[i] != 0) {

int temp = nums[i];

nums[i] = nums[j];

nums[j] = temp;

j++;

}

          }

                    }

}
```

OUT PUT :



Accepted Runtime: 0 ms

Case 1 Case 2

Input

```
nums =
[0,1,0,3,12]
```

Output

```
[1,3,12,0,0]
```

Expected

```
[1,3,12,0,0]
```

RESULT :

The program rearranges the array by shifting all zero values to the end and keeps the order of non-zero elements unchanged.

TASK 7 :

AIM:
To calculate the absolute difference between the sums of the primary and secondary diagonals of a square matrix**.**

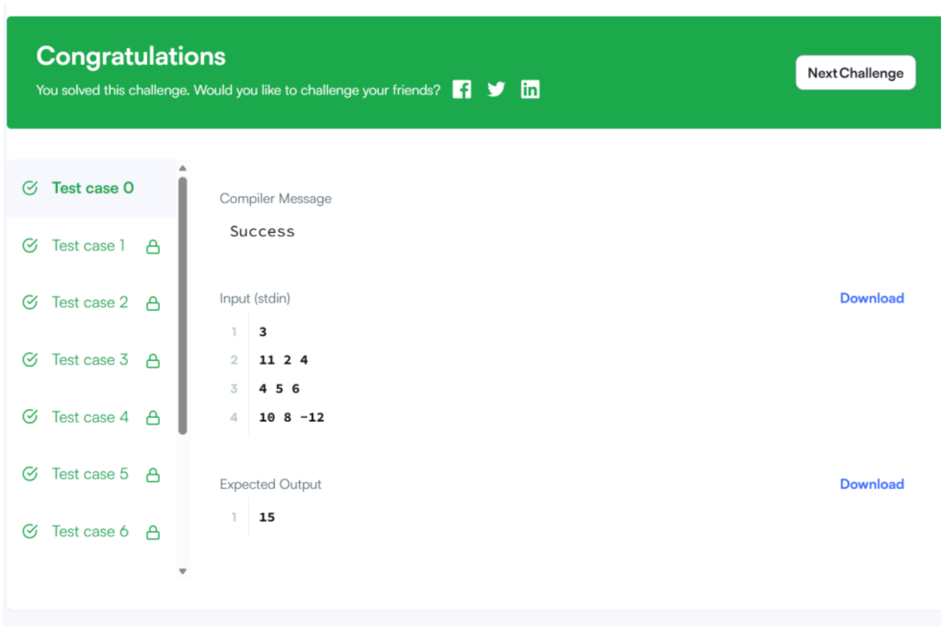
ALGORITHM:

1. Read the size n of the square matrix and its elements.
2. Initialize two variables to store the sums of the primary and secondary diagonals.
3. Traverse the matrix using a single loop.
4. Add elements arr[i][i] to the primary diagonal sum.
5. Add elements arr[i][n − i − 1] to the secondary diagonal sum.
6. Find and return the absolute difference between the two sums.

PROGRAM :

```
import java.util.Scanner;

public class Solution {

   public static int diagonalDifference(int[][] arr) {

      int n = arr.length;

      int primaryDiagonal = 0;

      int secondaryDiagonal = 0;

      for (int i = 0; i < n; i++) {

         primaryDiagonal += arr[i][i];

         secondaryDiagonal += arr[i][n - i - 1];

      }

      return Math.abs(primaryDiagonal - secondaryDiagonal);

   }

   public static void main(String[] args) {

      Scanner sc = new Scanner(System.in);
    int n = sc.nextInt(); // size of matrix
      int[][] arr = new int[n][n];
      for (int i = 0; i < n; i++) {
         for (int j = 0; j < n; j++) {
            arr[i][j] = sc.nextInt();
         }
     }
      System.out.println(diagonalDifference(arr));
      sc.close();
   }
}
```

OUTPUT :



Compiler Message

```
Success
```

Input (stdin)

```
3
11 2 4
4 5 6
10 8 -12
```

Expected Output

```
15
```

RESULT:

The program successfully computes and displays the absolute difference between the sums of the two diagonals of the given square matrix.

TASK 8 :

AIM:
To find the transpose of a given matrix by converting rows into columns.

ALGORITHM:

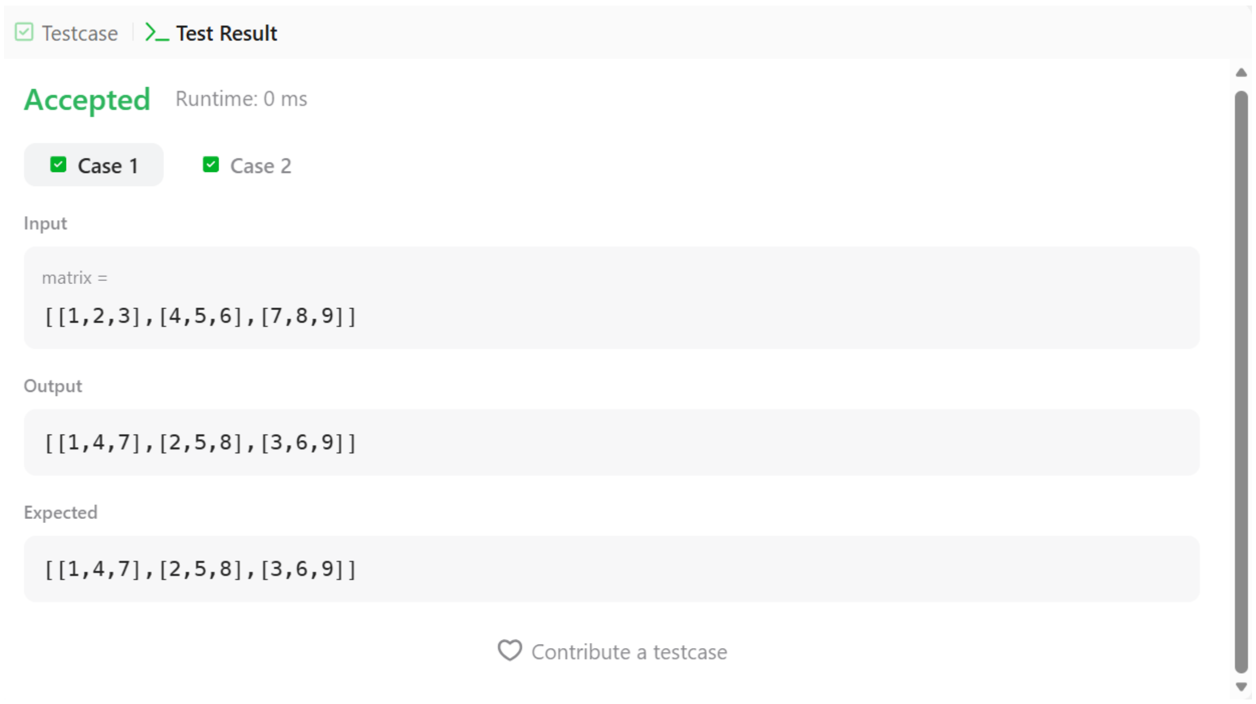
1. Read the number of rows m and columns n of the matrix.
2. Create a new matrix of size n × m to store the transpose.
3. Traverse the original matrix using nested loops.
4. Assign each element matrix[i][j] to result[j][i].
5. After completing the traversal, return the transposed matrix.

PROGRAM :

```
class Solution {

  public int[][] transpose(int[][] matrix) {

    int m = matrix.length;

    int n = matrix[0].length;

    int[][] result = new int[n][m];

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

      for (int j = 0; j < n; j++) {

        result[j][i] = matrix[i][j];

      }

    }

    return result;

  }

  public static void main(String[] args) {



 Solution sol = new Solution();

    int[][] matrix1 = {{1,2,3},{4,5,6},{7,8,9}};
```

```
        int[][] matrix2 = {{1,2,3},{4,5,6}};

        int[][] result1 = sol.transpose(matrix1);

        int[][] result2 = sol.transpose(matrix2);

        printMatrix(result1); // [[1,4,7],[2,5,8],[3,6,9]]

        printMatrix(result2); // [[1,4],[2,5],[3,6]]

    }

    private static void printMatrix(int[][] matrix) {

        for (int[] row : matrix) {

            for (int val : row) {

                System.out.print(val + " ");

            }

            System.out.println();

        }

    }

}
```

OUTPUT :



```
Accepted    Runtime: 0 ms

Case 1    Case 2

Input
matrix =
[[1,2,3],[4,5,6],[7,8,9]]

Output
[[1,4,7],[2,5,8],[3,6,9]]

Expected
[[1,4,7],[2,5,8],[3,6,9]]
```

RESULT :

The program successfully computes and displays the transpose of the given matrix, where rows are converted into columns and columns into rows.

TASK 9 :

AIM :

To compute the block sum for each element in a matrix using a given range k, where each block sum includes all elements within k distance from that element.

ALGORITHM:

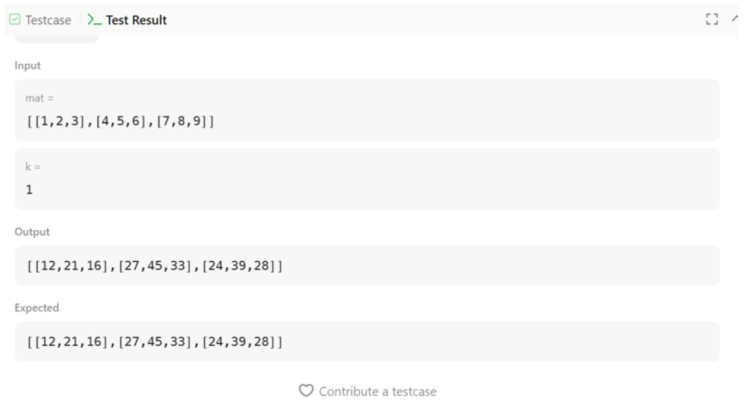
1. Read the matrix dimensions m and n.
2. Create a prefix sum matrix of size (m+1) × (n+1).
3. Compute prefix sums so that each cell stores the sum of elements from the top-left corner to that position.
4. For each element in the original matrix, determine the top-left and bottom-right boundaries of the block using k.
5. Use the prefix sum matrix to calculate the block sum efficiently.
6. Store the computed block sum in the result matrix and return it.

PROGRAM :

```
class Solution {

  public int[][] matrixBlockSum(int[][] mat, int k) {

    int m = mat.length, n = mat[0].length;

    int[][] prefix = new int[m+1][n+1];

    for (int i=1; i<=m; i++) {

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

        prefix[i][j] = mat[i-1][j-1] + prefix[i-1][j] + prefix[i][j-1] - prefix[i-1][j-1];

      }

    }

    int[][] ans = new int[m][n];


 for (int i=0; i<m; i++) {
```

```
            for (int j=0; j<n; j++) {
                int r1 = Math.max(0, i-k), c1 = Math.max(0, j-k);
                int r2 = Math.min(m-1, i+k), c2 = Math.min(n-1, j+k);
                ans[i][j] = prefix[r2+1][c2+1] - prefix[r1][c2+1] - prefix[r2+1][c1] + prefix[r1][c1];
            }
        }
        return ans;
    }
}
```

OUTPUT :



```
Input
mat =
[[1,2,3],[4,5,6],[7,8,9]]
k =
1
Output
[[12,21,16],[27,45,33],[24,39,28]]
Expected
[[12,21,16],[27,45,33],[24,39,28]]
```

RESULT :

The program successfully calculates the block sum matrix, where each element represents the sum of all elements within distance k from the corresponding position in the original matrix.

TASK 10 :

AIM :

To rotate the elements of a given matrix anticlockwise by r rotations layer by layer.

ALGORITHM:

1. Read the matrix dimensions m, n and the number of rotations r.
2. Determine the number of layers in the matrix.
3. For each layer, store its boundary elements in a list.
4. Rotate the list by r positions using modulo operation.
5. Place the rotated elements back into the matrix in the same layer order.
6. Repeat for all layers.
7. Display the final rotated matrix.

PROGRAM

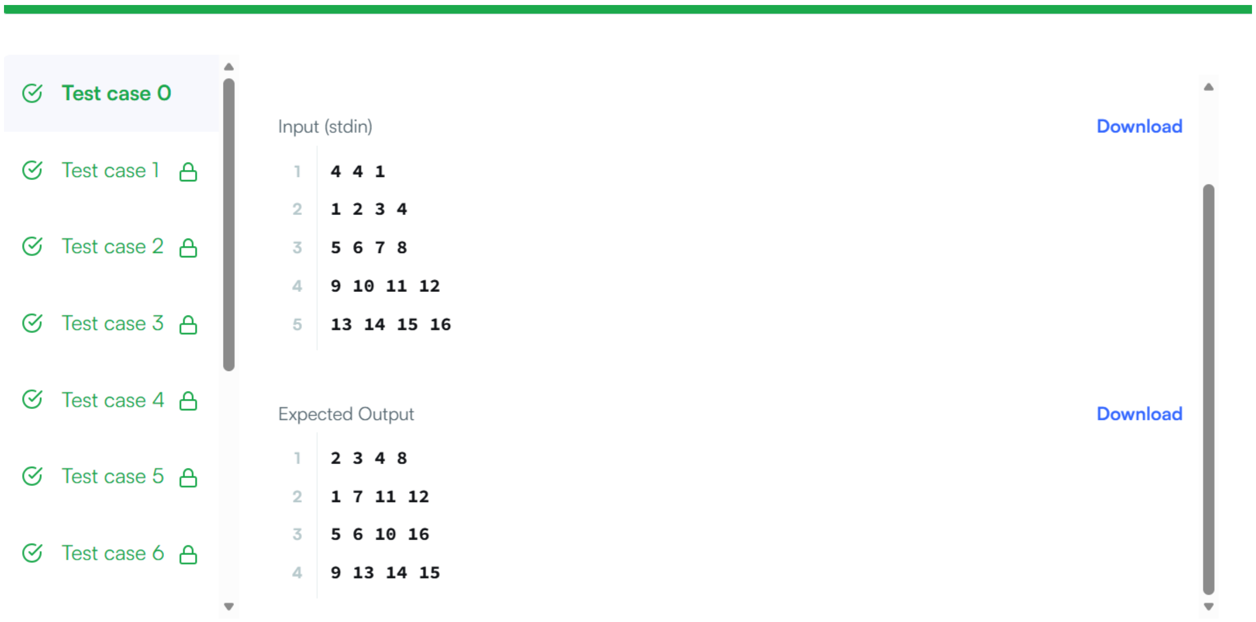
```
import java.util.*;

public class Solution {

    public static void matrixRotation(int[][] matrix, int r) {

        int m = matrix.length;

        int n = matrix[0].length;

        int layers = Math.min(m, n) / 2;

        for (int layer = 0; layer < layers; layer++) {

            List<Integer> elements = new ArrayList<>();

            for (int j = layer; j < n - layer; j++) {

                elements.add(matrix[layer][j]);

            }
```

```
for (int i = layer + 1; i < m - layer - 1; i++) {
                elements.add(matrix[i][n - layer - 1]);
            }
            for (int j = n - layer - 1; j >= layer; j--) {
                elements.add(matrix[m - layer - 1][j]);
            }
            for (int i = m - layer - 2; i > layer; i--) {
                elements.add(matrix[i][layer]);
            }
            int len = elements.size();
            int rot = r % len;
            List<Integer> rotated = new ArrayList<>();
            rotated.addAll(elements.subList(rot, len));
            rotated.addAll(elements.subList(0, rot));
            int idx = 0;
            for (int j = layer; j < n - layer; j++) {
                matrix[layer][j] = rotated.get(idx++);
            }
            for (int i = layer + 1; i < m - layer - 1; i++) {
                matrix[i][n - layer - 1] = rotated.get(idx++);
            }

  for (int j = n - layer - 1; j >= layer; j--) {
```

```
                matrix[m - layer - 1][j] = rotated.get(idx++);
            }
            for (int i = m - layer - 2; i > layer; i--) {
                matrix[i][layer] = rotated.get(idx++);
            }
        }
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                System.out.print(matrix[i][j] + " ");
            }
            System.out.println();
        }
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int m = sc.nextInt();
        int n = sc.nextInt();
        int r = sc.nextInt();
        int[][] matrix = new int[m][n];
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {

matrix[i][j] = sc.nextInt();
```

```
          }
        }
        matrixRotation(matrix, r);
        sc.close();
    }
}
```

OUTPUT :



Test case 0

Test case 1

Test case 2

Test case 3

Test case 4

Test case 5

Test case 6

Input (stdin) Download

```
4 4 1
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
```

Expected Output Download

```
2 3 4 8
1 7 11 12
5 6 10 16
9 13 14 15
```

RESULT :

The program successfully rotates the given matrix anticlockwise by the specified number of rotations and prints the updated matrix.