**Question 23:-**  BinaryToDecimal (int x[])

**Answer:-** **Basic Approach for Binary to Decimal Conversion**:

* The fundamental idea for converting a binary number to its decimal equivalent is to multiply each digit in the binary number by 2 raised to the power of its positional value and then add up these values.
* For example:
  + Input: 1100
  + Calculation: (1 \* 2^3 + 1 \* 2^2 + 0 \*2^1 + 0 \*2^0 = 8 + 4 + 0 + 0 = 12)
* - Complexity:
* - Time complexity: \(O(\log n)\)
* - Auxiliary space: \(O(1)\)
* - Note: If dealing with large binary values, consider using `long` instead of `int` to avoid errors.

**Question 24:- find the rows in a**  void max one row (int x[][]) in java

# Answer:-  To find the maximum number in each row of a 2D array (matrix) **Using a Loop** finds the maximum element in each row of a given matrix using loops:

This program initializes a 2D array called matrix, iterates through each row, and finds the maximum element in each row. The maxElement method calculates the maximum value for each row, and the printArray method displays the results.

**Question 25:- find the rows in a**  void max one row (int x[][]) in java

# Answer:-  converts a given number into words. You can customize the program by replacing the value of x with the desired number you want to convert:

 if you replace 54297 with your desired number, it will display “Fifty-Four Thousand Two Hundred Ninety-Seven

**Question 26**:- print digonal wise (int x[]) in java

### Answer:-

# . We iterate through the matrix and print the diagonal elements directly.

**Question 27:-** Sequential diagonal(int x[][]) in java

# Answer:-  Certainly! Let’s explore how to loop diagonally through a two-dimensional integer array in Java. This technique can be applied to square two-dimensional arrays of any size.

1. **Overview**:

# To loop diagonally, we’ll use row and column indices to access elements of the array.

# First, let’s determine the number of diagonal lines in the array. For an n x n array, there are 2n - 1 diagonal lines.

# Next, we find the mid-point (let’s call it midPoint) using the formula: midPoint = (diagonalLines / 2) + 1.

1. **Getting Row and Column Indices**:
2. We’ll loop from 1 to diagonalLines.

# Define a variable called itemsInDiagonal to keep track of the number of items in each diagonal line.

# If i is less than or equal to midPoint, increment itemsInDiagonal. Otherwise, decrement it.

* + Within the loop, create another loop with variable j ranging from 0 to itemsInDiagonal.

# Calculate the row and column indices based on whether i is greater than midPoint:

* + - If i is less than or equal to midPoint:
      * rowIndex = (i - j) - 1
      * columnIndex = j
    - Otherwise:
      * rowIndex = (length - 1) - j
      * columnIndex = (i - length) + j

# Use these indices to access the elements of the array.

**Question 29:-** void print LCM (int x[])

# Answer:-   **Least Common Multiple (LCM)** of an array of integers

**Question 30:-** void HCF (int x[][]) in java

# Answer:-   **Highest Common Factor (HCF)** of an array of integers. We’ll assume that the input array x contains the numbers for which we want to calculate the HCF.

1. **Using Static Method**:

# In this approach, we read two numbers from the user and calculate the HCF using a static method called hcfCal.

# The hcfCal method also uses the Euclidean algorithm to find the HCF.

**Question 31:-** find 3 largest value in array (int z[]) **without using sorting**

**Answer:-**  **Using Sorting: You can sort the array in ascending order and then retrieve the third largest element**

**and**

**without Sorting: You can sort the array in ascending order and then retrieve the third largest element**

# find the three largest values in an array in Java without using sorting, we can follow a straightforward approach. Let’s assume the input array is called arr.

# Initialize three variables to keep track of the largest, second largest, and third largest values. Initially, set them to negative infinity (or any value that ensures they will be updated).

# Iterate through the array arr:

# If the current element is greater than the largest value, update the largest value and shift the previous largest value to the second largest.

# If the current element is greater than the second largest value but not equal to the largest value, update the second largest value and shift the previous second largest value to the third largest.

# After iterating through the entire array, the three variables will hold the three largest values.

**Question 32:- float find median (int z[])**

**Answer:-**  class Median {

public static void main(String[] args) {

int n = 5;

double[] a = new double[n];

a[0] = 10;

a[1] = 20;

a[2] = 30;

a[3] = 40;

a[4] = 50;

double median;

if (n % 2 == 1) {

median = a[(n + 1) / 2 - 1];

} else {

median = (a[n / 2 - 1] + a[n / 2]) / 2;

}

System.out.println("Median: " + median);

}

}

**Question 33:- float find mean (int z[]) in array**

# Answer:-  Here’s how the calculateMean method works:

# Initialize a sum variable to store the sum of all array elements.

# Iterate through each element in the array using an enhanced for loop.

# Add each element to the sum.

# Finally, divide the sum by the total number of elements in the array to find the mean.

**Question 34:- float find mode (int z[]) in array**

# Answer:-  **Using an Array to Count Occurrences**: We can create an array to keep track of how many times each value appears in the input array. Then, we find the value with the highest count.

findMode method in your main method:

**Question 35:- convert all array elements into zero (int z[])**

# Answer:-  **Pairwise Reduction to Zero**: Given an array arr[], we want to find the number of operations required to convert all array elements to zero by decrementing the value of array elements in pairs by any positive value. If it’s not possible to convert all elements to zero, we return