DSA (Number System)Solutions

Problem 1: given a number, print its binary representation. [easy]

Input 1: number = 5 Output 1: 101

Input 2: number = 10 Output 2: 1010

public class BinaryRepresentation {

public static void main(String[] args) {

// Test cases

int number1 = 5;

int number2 = 10;

// Print binary representation of the numbers

System.out.println("Binary representation of " + number1 + " is: " + toBinary(number1));

System.out.println("Binary representation of " + number2 + " is: " + toBinary(number2));

}

// Method to convert a number to its binary representation

public static String toBinary(int number) {

return Integer.toBinaryString(number);

}

}

Problem 2: given a number ‘n’, predict whether it is a power of two or not. [medium] Input 1: n = 15 Output 1: False Input 2: n = 32 Output 2: True

public class PowerOfTwo {

public static void main(String[] args) {

// Test cases

int n1 = 15;

int n2 = 32;

// Check if the numbers are powers of two and print results

System.out.println(n1 + " is a power of two: " + isPowerOfTwo(n1));

System.out.println(n2 + " is a power of two: " + isPowerOfTwo(n2));

}

// Method to check if a number is a power of two

public static boolean isPowerOfTwo(int n) {

return n > 0 && (n & (n - 1)) == 0;

}

}

Q3. Problem 1: Given a number. Using bit manipulation, check whether it is odd or even. Input 8, Even 3, False

public class OddEvenCheck {

public static void main(String[] args) {

// Test cases

int number1 = 8;

int number2 = 3;

// Check and print if the numbers are odd or even

System.out.println(number1 + " is even: " + isEven(number1));

System.out.println(number2 + " is even: " + isEven(number2));

}

// Method to check if a number is even using bit manipulation

public static boolean isEven(int number) {

return (number & 1) == 0;

}

}

Q4. Given a number, count the number of set bits in that number without using an extra space. Note : bit ‘1’ is also known as set bit. }

public class CountSetBits {

public static void main(String[] args) {

// Test cases

int number1 = 15; // Binary: 1111, Set bits: 4

int number2 = 9; // Binary: 1001, Set bits: 2

// Count and print the number of set bits for each number

System.out.println("Number of set bits in " + number1 + ": " + countSetBits(number1));

System.out.println("Number of set bits in " + number2 + ": " + countSetBits(number2));

}

// Method to count the number of set bits in a number using Brian Kernighan's algorithm

public static int countSetBits(int number) {

int count = 0;

while (number > 0) {

number = number & (number - 1); // Flip the least significant set bit

count++;

}

return count;

}

}

Q5. Given an integer array, duplicates are present in it in a way that all duplicates appear an even number of times except one which appears an odd number of times. Find that odd appearing element in linear time and without using any extra memory.

For example, Input : arr[] = [4, 3, 6, 2, 6, 4, 2, 3, 4, 3, 3] Output : The odd occurring element is 4.

public class OddOccurrenceFinder {

public static void main(String[] args) {

// Test case

int[] arr = {4, 3, 6, 2, 6, 4, 2, 3, 4, 3, 3};

// Find and print the element that appears an odd number of times

int oddOccurrenceElement = findOddOccurrence(arr);

System.out.println("The odd occurring element is: " + oddOccurrenceElement);

}

// Method to find the element that appears an odd number of times using XOR

public static int findOddOccurrence(int[] arr) {

int result = 0;

for (int num : arr) {

result ^= num; // XOR each element with result

}

return result;

}

}