Ex.1

**package** MathForDSA;

**import** java.util.\*;

**public** **class** A111EvenNumUsingBitwise {

//Que.Even Number using Bitwise

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

System.***out***.println("Enter any No: ");

Scanner sc=**new** Scanner(System.***in***);

**int** no = sc.nextInt();

**if**((no&1)==1) { //Bitwise & operator

System.***out***.println(no+" is odd");

}**else** {

System.***out***.println(no+" is even");

}

}

/\*output:

\* 1) Enter any No: 12

\* 12 is even

\*

\* 2) Enter any No: 3

\* 3 is odd

\*

\*/

}

Ex.2  
**package** MathForDSA;

**import** java.util.\*;

**public** **class** A11UniqueNoInArray {

//Que. Find the Unique no from array other elements are double i.e ans should be 6

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

**int** arr[] = {1,5,4,6,4,1,5};

//We know XOR operator are 1 when one value is 0 and another 1 that is (0 XOR 1) = 1 , (1 XOR 0) = 1 otherwise 0

**int** ans=*findUnique*(arr);

System.***out***.println("Ans is "+ans);//Ans:6

}

**public** **static** **int** findUnique(**int** arr[]){

**int** unique=0;

**for**(**int** n:arr) {

unique ^=n; //We know a^a=0 therefore 4^4=0 here we do 1^5^4^6^4^1^5=6 like 1-2-1+2+3=3 Sequence No matter in XOR

}

**return** unique;

}

}

Ex.3  
**package** MathForDSA;

**public** **class** A1Find\_ith\_BitinNo {

// Que. Find the ith bit of a number.

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

**int** no=10; //10 means 1010

**int** ith=1; //3rd bit value of no 10 is 1 //position starts from 0 of right side

**int** ans=*findithBit*(no,ith);

System.***out***.println("Ans is :"+ans); // Ans:1

}

**public** **static** **int** findithBit(**int** no,**int** ith) {

**int** ithBit; //10 means 1010

**int** bitMask=1<<ith;// 1000 // 1 means 0001. For ith value i.e 3rd value we need to shift 1 to the left side

ithBit= no&bitMask; // Now finally 1010 & 1000= gives 1000 So ith i.e 3rd bit is 1.

**if**(ithBit>0) {

**return** 1;

}

**return** 0;

}

}

Ex.4

**package** MathForDSA;

**public** **class** A222Set\_ith\_Bit {

// Que. Set ith bit means set ith bit as a 1. no matter either ith bit is 0 or 1.

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

**int** no=10; //10 means 1010 //Kunal kushwaha says position starts from 1 of right side which is Wrong.

**int** ith=2; // here 2nd position bit is 0 .Remember position starts from 0 of right side. like ...,2,1,0

**int** ans=*setithBit*(no,ith);

System.***out***.println("Ans is :"+ans);//Ans:14 i.e 1110

}

**public** **static** **int** setithBit(**int** no,**int** ith) {

**int** ans;

**int** bitMask=1<<ith;//this gives 1<<2 means 0100

ans=no|bitMask; //1010 | 0100

**return** ans; //1110 means 14

}

}

Ex.5  
**package** MathForDSA;

**public** **class** A22Reset\_ith\_Bit {

//Que. Reset/Unset/Clear ith bit means to create ith bit as 0 either bit is already 0 or 1 no matter.

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

**int** no=10; //1010

**int** ithPosition=1; //In 1 position of no 10 is 1 bit

**int** bitMask=1<<ithPosition; //0010

**int** notBitMask=~bitMask; //1101

**int** ans= no & notBitMask; //1010 & 1101

System.***out***.println("Ans is:"+ans);//1000 i.e 8

}

}

Ex.6  
**package** MathForDSA;

**public** **class** A2Find\_RightMost\_SetBt {

//Que.Find the position of the right most set bit. like 101100100. here 2nd is the right most set bit position

**public** **static** **void** main(String[] args) {

**int** no=14;//1110 here right most set bit position is 1. 1st bit i.e 0's position consider as 0.

**int** ans=*findRightMostSetBit*(no);

System.***out***.println("Right Most set bit is at position:"+ans);//ans:1

}

**public** **static** **int** findRightMostSetBit(**int** no) {

**int** i=0;

**while**(no>0) {

**if**((no&1)==1) { //no&1 gives ans as 0 or 1. if 1 means no is odd else even for 1st loop

**break**;

}**else** {

i++; //count the position of set bit

no>>=1; //Using right shift we remove the last bit of number

}

}

**return** i;

}

}

Ex.7  
**package** MathForDSA;

**public** **class** A33Magic\_Num {

//Que. Multiply each bit of num by 5^1,5^2,5^3.....and Give the ans of that number. //Question ask in Amazon

//ex. for 6 num (binary=110)=((5^3\*1)+(5^2\*1)+(5^1\*0))=150

**public** **static** **void** main(String[] args) {

**int** n=6;

**int** ans=*magicNumAns*(n);

System.***out***.println("Magic No Ans: "+ans);//Ans:150

}

//My Ans:

/\* public static int magicNumAns(int num) {

int NoToMultiply=5;

int magicNoAns=0;

int i=1;

while(num>0) {

int bitValue=0;

int totalValue=1;

bitValue=(num&1); //Here we check bit is 1 or 0

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

totalValue=totalValue\*5; //here we take power of 5 as per the i value.

}

magicNoAns+=bitValue\*totalValue; //multiply power of 5 with bitvalue and add in ans

i++;

num>>=1; //right shift is used for remove the last bit or divide by 2 to the num

}

return magicNoAns;

} \*/

//Kunal Kushwaha's ans:

**public** **static** **int** magicNumAns(**int** num) {

**int** baseValueToMultiply=5;

**int** ans=0;

**while**(num>0) {

**int** lastbit=num&1;

num>>=1;

ans+=(lastbit\*baseValueToMultiply);

baseValueToMultiply=baseValueToMultiply\*5;

}

**return** ans;

}

}

Ex.8

**package** MathForDSA;

**import** java.util.\*;

**public** **class** A3NoOfDigitsUsingFormula {

**public** **static** **void** main(String[] args) {

// int num=23415;

// int base=10; //base=10 means decimal

**int** num=11;

**int** base=2; //base=10 means decimal and base=2 means binary

**int** noOfDigits = (**int**)(Math.*log*(num)/Math.*log*(base))+1; //Formula for calculating digits

System.***out***.println("No of Digits:"+noOfDigits);//Ans:4

}

}

Ex.9

**package** MathForDSA;

**public** **class** A444PowerOfTwo {

//Que.Find the num is power of 2 or not //means 2,4,8,16,32,...

**public** **static** **void** main(String[] args) {

**int** num=8;

**boolean** ans=*isPowerOfTwoRightShift*(num);

System.***out***.println(ans);//Ans:true

**boolean** ans2=*isPowerOfTwoFormula*(num);

System.***out***.println(ans2);//Ans:true

}

//1st approach using right shift

//time complexity:O(log(Num))

**public** **static** **boolean** isPowerOfTwoRightShift(**int** Num) {

**int** count=0; //we count the no of bit 1 present in Num

**while**(Num>0) {

count+=Num&1;

Num>>=1; //we remove the last bit i.e devide the no by 2

}

**if**(count==1) { //Only 1 bit and other bits are 0 means Num is power of 2

**return** **true**;

}

**return** **false**;

}

//2nd approach using formula

**public** **static** **boolean** isPowerOfTwoFormula(**int** Num) {

**if**(Num==0) {

**return** **false**;

}

**if**((Num & (Num-1))==0){

**return** **true**;

}

**return** **false**;

}

}

Ex.10  
**package** MathForDSA;

**public** **class** A44PowerOfNum {

//Que.Find the value a^b in optimize time complexity

// 1 approach is (a\*a\*a\*\*...) b times but time complexity is O(b)

**public** **static** **void** main(String[] args) {

//second approach

**int** base =2;

**int** power=5; //3^6 =3^4\*3^2; power 6 means=110 in binary

**int** ans = *findPowerValue*(base,power);

System.***out***.println(ans);

}

//Time Complexity of this approach:O(log(b)) if a^b

**public** **static** **int** findPowerValue(**int** base,**int** power) {

**int** ans=1;

**while**(power>0) {

**int** bitValue=power&1;

**if**(bitValue==1) {

ans\*=base;

}

base\*=base; // value be 9, 81, ... 9 means 3^2 , 81 means 3^4, then 3^8.....

power>>=1; //using right shift we travel through binary bits of power

}

**return** ans;

}

}

Ex.11  
**package** MathForDSA;

**public** **class** A4CountSetBit {

//Que. Find the no of set bit in a given number like num=9 means(1001)= no of set bit=2

**public** **static** **void** main(String[] args) {

**int** num=98887;

System.***out***.println(Integer.*toBinaryString*(num)); //this gives binary value in string datatype //Ans:11000001001000111

**int** count=*noOfSetBit*(num);

System.***out***.println("No of set bits are: "+count); //Ans:No of set bits are: 7

}

**public** **static** **int** noOfSetBit(**int** No) {

**int** count=0;

//My Approach

/\* while(No>0) {

count+=(No&1); //count the 1

No>>=1; //remove the last bit through number i.e devide the no by 2

} \*/

//or Kunal Kushwaha approach

**while**(No>0) {

count++;

No = No & (No-1);

}

**return** count;

}

}

Ex.12  
**package** MathForDSA;

**public** **class** A5RangeForXOR {

//Que. Find the value of XOR from 0 to n

**public** **static** **void** main(String[] args) {

**int** n=10;

**int** ans=*xor*(n);

System.***out***.println("xor of Range from 0 to "+n+" is "+ans);//Ans:xor of Range from 0 to 10 is 11

//range of xor from a to b = xor[b]^xor[a-1]

**int** a=3;

**int** b=9;

**int** rangeAns=*xor*(b)^*xor*(a-1); //xor(a-1) part already comes in xor(b) so xor(b)^xor(a-1) substract a-1 from b

// which gives xor range between a and b

System.***out***.println("xor Range between "+a+" and "+b+" is "+rangeAns);//xor Range between 3 and 9 is 2

}

**public** **static** **int** xor(**int** num){

**if**(num%4==0) {

**return** num;

}**else** **if**(num%4==1) {

**return** 1;

}**else** **if**(num%4==2) {

**return** num+1;

}

**return** 0;

}

}

Ex.13

**package** MathForDSA;

**public** **class** A6FlippingImage {

//leetcode que. This que comes in Google

//Que. Given n\*n binary matrix image containing 0 and 1 i.e 2d array

// step 1: reverse the each row

// step 2: Replace 0 by 1 and 1 by 0

**public** **static** **void** main(String[] args) {

**int**[][] arr = {{1,1,0,0},{1,0,0,1},{0,1,1,1},{1,0,1,0}};

//Ans is : [[1,1,0,0],[0,1,1,0],[0,0,0,1],[1,0,1,0]]

**int**[][] newArray;

newArray=*flippingImage*(arr);

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

**for**(**int** j=0;j<newArray[i].length;j++) {

System.***out***.print(newArray[i][j]);

}

System.***out***.println();

}

}

//My Ans:

/\* public static int[][] flippingImage(int arr[][]){

int temp;

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

for(int j=0,k=arr[0].length-1;j<arr[0].length/2;j++,k--) {

temp=arr[i][j];

arr[i][j]=arr[i][k];

arr[i][k]=temp;

// System.out.println(arr[i][j]);

}

}

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

for(int j=0;j<arr[i].length;j++) {

arr[i][j]^=1;

}

}

return arr;

} \*/

//Kunal Kushwaha optimized ans

**public** **static** **int**[][] flippingImage(**int** arr[][]){

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

**for**(**int** j=0;j<(arr[0].length+1)/2;j++) {

**int** temp=arr[i][arr[i].length-1-j]^1;

arr[i][arr[i].length-1-j]=arr[i][j]^1;

arr[i][j]=temp;

}

}

**return** arr;

}

//LeetCode Optimized ans:

/\* public int[][] flipAndInvertImage(int[][] A) {

int C = A[0].length;

for (int[] row: A)

for (int i = 0; i < (C + 1) / 2; ++i) {

int tmp = row[i] ^ 1;

row[i] = row[C - 1 - i] ^ 1;

row[C - 1 - i] = tmp;

}

return A;

} \*/

}

Ex.14  
**package** MathForDSA;

**public** **class** A7PrimeNum {

//Check the num's are prime or not

**public** **static** **void** main(String[] args) {

**int** n=20;

**for**(**int** i=1;i<=n;i++) {

System.***out***.println(i+" "+*isPrime*(i));

}

}

**public** **static** **boolean** isPrime(**int** num) {

**if**(num<=1) {

**return** **false**;

}

**int** c=2;

**while**(c\*c<=num) { //c\*c<=num means c<=sqrt(num)

**if**(num%c==0) {

**return** **false**;

}

c++;

}

**return** **true**;

}

}

/\* output:

\* 1 false

2 true

3 true

4 false

.

.

19 true

20 false

\* \*/

Ex.15  
**package** MathForDSA;

**public** **class** A8PrimeNumOptimizeWay {

//Que. find the prime num less than 40 in optimized way using Seive's algorithm

//Space Complexity:O(n);

//Time Complexity: O(nlog(logn))

**public** **static** **void** main(String[] args) {

**int** n=37;

**boolean**[] primenums = **new** **boolean**[n+1];//by default all indexes are false

*isPrimNumSeive*(n,primenums);

}

//boolean value false means Prime Num

**public** **static** **void** isPrimNumSeive(**int** num,**boolean**[] primes) {

**for**(**int** i=2;i\*i<=num;i++) { //loops work till the sqrt of num

**if**(!primes[i]) { // !false=true

**for**(**int** j=2\*i;j<=num;j+=i) {

primes[j]=**true**; //here the value can multiply which become true

} // here true means Not a prime number

}

}

//Print All prime num's

**for**(**int** i=2;i<=num;i++) {

**if**(!primes[i]) { //the value which are false in boolean array means Prime num

System.***out***.print(i+" "); //Ans:2 3 5 7 11 13 17 19 23 29 31 37

}

}

}

}

Ex.16

**package** MathForDSA;

**public** **class** A9sqrtUsingBinarySearchSeive {

//Que. Find the square root of num using binary search still 3 precision digits(decimal points)

//Time complexity: O(log(n))

**public** **static** **void** main(String[] args) {

**int** num=38;

**int** precision=3;

**double** ans=*sqrtUsingBinary*(num,precision);

System.***out***.println("Ans is "+ans);//Ans is 6.164

}

**public** **static** **double** sqrtUsingBinary(**int** num,**int** precision) {

**double** ans=0.0;

**int** start=0;

**int** end=num;

**double** root=0.0;

// int mid=(start+end)/2;

**while**(start<=end) { //imp

**int** mid=start+((end-start)/2);

**if**((mid\*mid)==num) {

ans=mid;

**return** ans;

}

**if**((mid\*mid)>num) {

end=mid-1;

root=end; //imp

}**else** {

start=mid+1;

root=start; //imp

}

}

//When num is not perfect square

**double** inc=0.1;

**for**(**int** i=0;i<precision;i++) {

**while**(root\*root<num) {

root+=inc;

}

root -= inc;

inc /= 10;

}

**return** root;

}

}

Ex.17  
**package** MathForDSA;

//import java.util.\*;

**public** **class** B1sqrtUsingNewtonRaphson {

//Que. Find the sqrt of the number using Newton Raphson method

// formula= root = 0.5(x + (n/x)) where n=number, x=assuming sqrt root=ans

**public** **static** **void** main(String[] args) {

**int** num=40;

System.***out***.println("Sqrt using Newton Raphson Method: "+*sqrt*(num));//Ans:6.325023209103984

}

**public** **static** **double** sqrt(**int** num) {

**double** x=num; //x means assume root

**double** root;

**while**(**true**) {

root = 0.5\*(x + (num/x)); //here for every iteration root becomes half of previous root

**if**(Math.*abs*(root-x)<0.5) { //Math.abs convert -ve or +ve num into +ve.

**break**; //when diff is less than 0.5 then we break the infinite while loop

}

x=root; //here x become root means every iteration root becomes half.

}

**return** root;

}

}

Ex.18  
**package** MathForDSA;

**import** java.util.ArrayList;

**public** **class** B2FactorsOfNumber {

//Find the factors of a number

// Brute force solution

//Time Complexity: O(n)

**public** **static** **void** main(String[] args) {

**int** num=36;

*findFactors*(num);

*findFactors2*(num);

*findFactors3*(num);

}

//Time Complexity: O(n)

**public** **static** **void** findFactors(**int** num) {

**for**(**int** i=1;i<=num;i++) {

**if**(num%i==0) {

System.***out***.print(i+" ");//Ans:1 2 3 4 6 9 12 18 36

}

}

}

//Optimized way time complexity:O(sqrt(n)) Unsorted order

**public** **static** **void** findFactors2(**int** num) { //output:1 36 2 18 3 12 4 9 6

**for**(**int** i=1; i<=Math.*sqrt*(num);i++) {

**if**(num%i==0) {

**if**(i==(num/i)) { //in case of num=36 it gives 2 times 6 therefore we write this condition

System.***out***.print(i+" ");

}**else** {

System.***out***.print(i+" "+num/i+" "); //Ans is Unsorted order

}

}

}

}

//Optimized way with Sorted order with time complexity:O(sqrt(n)) and Space complexity: O(sqrt(n))

//Here we use Arraylist So, Space complexity:O(sqrt(n));

**public** **static** **void** findFactors3(**int** num) { // Output: 1 2 3 4 6 9 12 18 36

ArrayList<Integer> list=**new** ArrayList<>();

**for**(**int** i=1;i<=Math.*sqrt*(num);i++) {

**if**(num%i==0) {

**if**(i==num/i) {

System.***out***.print(i+" ");

}**else** {

System.***out***.print(i+" ");

list.add(num/i); //here we add only the Descending Order elements in list

}

}

}

**for**(**int** j=list.size()-1;j>=0;j--) {//we want to display descending order elements into ascending order

System.***out***.print(list.get(j)+" "); //here we display list elements

}

}

}

Ex.19  
**package** MathForDSA;

**public** **class** B3GCD\_LCM {

//Que. Find the GCD and LCM of a number. Note: GCD means HCF

**public** **static** **void** main(String[] args) {

**int** a=105;

**int** b=224;

**int** gcdAns=*gcd*(a,b);

System.***out***.println("GCD/HCF is "+gcdAns);//Ans:GCD/HCF is 7

System.***out***.println("LCM is "+*lcm*(a,b));//Ans:LCM is 3360

}

//Here we use recursion because same step happened again and again

**public** **static** **int** gcd(**int** a,**int** b) {

//base case

**if**(a==0) {

**return** b; //a=0 and b=any num then gcd=b;

}

**return** *gcd*(b%a,a);

}

**public** **static** **int** lcm(**int** a, **int** b) {

**return** a\*b/*gcd*(a,b); //Formula: lcm\*gcd=a\*b So,lcm=a\*b/gcd;

}

}