```
public class Bit_Manipulation {
    // Introduction to Bit Manipulation
    // Check if the i-th bit is set or not
    class Solution {
        // Function to check if Kth bit is set or not.
        static boolean checkKthBit(int n, int k) {
            // Your code here
            int bit = 1 << k;
            int check = bit & n;
            if (check != 0) {
                return true;
            }
            return false;
        }

    }

    // Check if a number is odd or not
    class Solution {
        static String oddEven(int N) {
            // code here
            int check = N & 1;
            if (check != 0) {
                return "odd";
            }
            return "even";
        }
    }

    // Check if a number is power of 2 or not
    class Solution {
        public static boolean isPowerofTwo(long n) {
            if (n == 0) {
                return false;
            }
            long res = n & (n - 1);
            if (res == 0) {
                return true;
            } else
                return false;
        }
    }

    // Count the number of set bits
    class Solution {
        public static int countSetBits(int n) {
            int cnt = 0;
            for (int k = 1; k <= n; k++) {
                for (int i = 0; i < 32; i++) {
```

```
            int bit = 1 << i;
            int check = bit & k;
            if (check != 0) {
                cnt++;
            }
        }
    }
    return cnt;
}

class Solution {

    // Function to return sum of count of set bits in the integers from 1 to n.
    public static int countSetBits(int n) {

        // Your code here
        n += 1;
        int count = 0;
        for (int x = 2; x / 2 < n; x = x * 2) {
            int quotient = n / x;
            count += quotient * x / 2;
            int remainder = n % x;
            if (remainder > x / 2) {
                count += remainder - x / 2;
            }
        }
        return count;
    }
}

class Solution {

    // Function to return sum of count of set bits in the integers from 1 to n.
    public static int countSetBits(int n) {

        // Your code here
        int cnt = 0;
        for (int i = 1; i <= n; i++) {
            int x = i;
            while (x > 0) {
                if ((x & 1) == 1) {
                    cnt++;
                }
                x /= 2;
            }
        }
        return cnt;
    }
}
```

```
}

// Set/Unset the rightmost unset bit
class Solution {
    static int setBit(int N) {
        return setRightmostUnsetBit(N);
    }

    static int getPosOfRightmostSetBit(int n) {
        return (int) ((Math.log10(n & -n)) / (Math.log10(2))) + 1;
    }

    static int setRightmostUnsetBit(int n) {
        if (n == 0) {
            return 1;
        }
        if ((n & (n + 1)) == 0) {
            return n;
        }
        int pos = getPosOfRightmostSetBit(~n);
        return ((1 << (pos - 1)) | n);
    }
}

// Swap two numbers
class Solution {
    static List<Integer> get(int a, int b) {
        // code here
        a = a ^ b;
        b = a ^ b;

        a = a ^ b;

        List<Integer> ls = new ArrayList<>();
        ls.add(a);
        ls.add(b);
        return ls;
    }

}

// Divide two integers without using multiplication or division operator
class Solution {
    public static long divide(long dividend, long divisor) {
        // code here
        long sign = ((dividend < 0) ^ (divisor < 0)) ? -1 : 1;
        dividend = Math.abs(dividend);
        divisor = Math.abs(divisor);
```

```
        long quotient = 0, temp = 0;
        for (int i = 31; i >= 0; --i) {
            if (temp + (divisor << i) <= dividend) {
                temp += divisor << i;
                quotient += 1L << i;
            }
        }
        if (sign == -1) {
            quotient = -quotient;
        }
        return quotient;
    }
}

// Count number of bits to be flipped to convert
class Solution {
    public int minBitFlips(int start, int goal) {
        int mini = 0;
        int res = start ^ goal;
        for (int i = 0; i < 32; i++) {
            int x = res & (1 << i);
            if (x != 0) {
                mini++;
            }
        }
        return mini;
    }
}

// Find the number that appears odd number of times
class Solution {
    public int singleNumber(int[] nums) {
        int ans = 0;
        int n = nums.length;
        for (int i = 0; i < n; i++) {
            ans = ans ^ nums[i];
        }
        return ans;
    }
}

// Power Set
class Solution {
    public List<List<Integer>> subsets(int[] arr) {
        List<List<Integer>> ds = new ArrayList<>();
        int n = arr.length;
        int s = 1 << n;
        for (int i = 0; i < s; i++) {
            List<Integer> ans = new ArrayList<>();
```

```
            for (int j = 0; j < n; j++) {
                if (f(i, j)) {
                    ans.add(arr[j]);
                }
            }
            ds.add(ans);
        }
        return ds;
    }

    public boolean f(int num, int bitpos) {
        int x = num & (1 << bitpos);
        return x != 0;
    }
}

// Find xor of numbers from L to R
class Solution {
    public static int findXOR(int l, int r) {
        return computeXOR(r) ^ computeXOR(l - 1);
    }

    static int computeXOR(int n) {
        if (n % 4 == 0) {
            return n;
        }
        if (n % 4 == 1) {
            return 1;
        }
        if (n % 4 == 2) {
            return n + 1;
        }
        return 0;
    }
}

// Find the two numbers appearing odd times
class Solution {
    public int[] singleNumber(int[] nums) {
        int[] ans = new int[2];
        int n = nums.length;
        int xor = 0, firstbit = 0;
        for (int i = 0; i < n; i++) {
            xor = xor ^ nums[i];
        }
        firstbit = f(xor);
        int mask = 1 << firstbit;
        for (int i = 0; i < n; i++) {
            int check = nums[i] & mask;
```

```
            if (check != 0)
                ans[1] = ans[1] ^ nums[i];
            else
                ans[0] = ans[0] ^ nums[i];
        }
        return ans;

    }

    public int f(int xor) {
        int bitmask = 1;
        for (int i = 0; i < 32; i++) {
            bitmask = bitmask << i;
            int check = xor & bitmask;
            if (check != 0) {
                return i;
            }
        }
        return -1;
    }
  }
}
```