Q1. Single Number Given a non-empty array of integers nums, every element appears twice except for one. Find that single one. You must implement a solution with a linear runtime complexity and use only constant extra space.

```
class Solution {
    public int singleNumber(int[] nums) {
        int result = 0;
        for (int num : nums) {
            result ^= num;
        }
        return result;
    }
}
```

Q2. Reverse Bits Reverse bits of a given 32 bits unsigned integer.

```
public class Solution {

  // you need treat n as an unsigned value
  public int reverseBits(int n) {
    int result = 0;
    for (int i = 0; i < 32; i++) {
      int lsb = n & 1;
      int reverselsb = lsb << (31 - i);
      result = result | reverselsb;
      n = n >> 1;
    }
    return result;
  }
}
```

Q3. Counting Bits Given an integer n, return an array ans of length n + 1 such that for each i (0 <= i <= n), ans[i] is the number of 1's in the binary representation of i.

```
public class Solution {
    public int[] countBits(int n) {
        int[] ans = new int[n + 1];
        for (int i = 1; i <= n; i++) {
            ans[i] = ans[i >> 1] + (i & 1);
        }
        return ans;
    }
}
```

Q4. Number Complement The complement of an integer is the integer you get when you flip all the 0's to 1's and all the 1's to 0's in its binary representation. For example, The integer 5 is "101" in binary and its complement is "010" which is the integer 2. Given an integer num, return its complement.

```
class Solution {
    public int findComplement(int num) {
        int tmp = (int) (Math.pow(2,digits(num))-1);
        return tmp - num;
    }

    static int digits(int n){
        int count = 0;
        while(n > 0){
            n = n/2;
            count++;
        }
        return count;
    }
}
```

Q5. Binary Gap Given a positive integer n, find and return the longest distance between any two adjacent 1's in the binary representation of n. If there are no two adjacent 1's, return 0. Two 1's are adjacent if there are only 0's separating them (possibly no 0's). The distance between two 1's is the absolute difference between their bit positions. For example, the two 1's in "1001" have a distance of 3.

```
class Solution {
    public int binaryGap(int n) {
        int count = 0;
        int max = 0;
        while(n > 0) {
            if ((n & 1) == 1) {
                max = Math.max(count, max);
                count = 1;
            } else if(count > 0) {
                count++;
            }
            n >>= 1;
        }
        return max;
    }
}
```

Q6. Maximum Product of Word Lengths Given a string array words, return the maximum value of length(word[i]) * length(word[j]) where the two words do not share common letters. If no such two words exist, return 0.

```
class Solution {
    public int maxProduct(String[] words) {
        int n = words.length;
        int[] masks = new int[n];

        for (int i=0; i<n; i++)
            for (char c: words[i].toCharArray())
                masks[i] |= (1 << (c - 'a'));

        int largest = 0;
        for (int i=0; i<n-1; i++)
            for (int j=i+1; j<n; j++)
                if ((masks[i] & masks[j]) == 0)
                    largest = Math.max(largest, words[i].length() *
words[j].length());

        return largest;
    }
}
```

Q7. Divide Two Integers Given two integers dividend and divisor, divide two integers without using multiplication, division, and mod operator. The integer division should truncate toward zero, which means losing its fractional part. For example, 8.345 would be truncated to 8, and -2.7335 would be truncated to -2. Return the quotient after dividing dividend by divisor.

```
class Solution {
    public int divide(int dividend, int divisor) {
        int x;
        if(dividend==divisor){
            return 1;
        }

        if(dividend==Integer.MIN_VALUE && divisor==-1){
            x= Integer.MAX_VALUE;
            return x;
        }

        x=dividend/divisor;
        return x;


    }
}
```

Q8. Can I Win In the "100 game" two players take turns adding, to a running total, any integer from 1 to 10. The player who first causes the running total to reach or exceed 100 wins. What if we change the game so that players cannot re-use integers? For example, two players might take turns drawing from a common pool of numbers from 1 to 15 without replacement until they reach a total >= 100.

```
class Solution {
    int numlimit, tgt;

    public boolean canIWin(int maxChoosableInteger, int desiredTotal) {
        numlimit = maxChoosableInteger;
        tgt = desiredTotal;

        int maxsum = (numlimit * (numlimit + 1)) / 2;
        if (maxsum < tgt) return false;

        int[] dp = new int[1 << numlimit];
        return solve(0, 0, dp);
    }

    public boolean solve(int mask, int lstsum, int dp[]) {
        if (dp[mask] != 0) return dp[mask] == 1;

        for (int i = 0; i < numlimit; i++) {
            if ((mask & (1 << i)) == 0 && (lstsum + (i + 1) >= tgt || !solve((mask
| (1 << i)), lstsum + (i + 1), dp))) {
                dp[mask] = 1;
                return true;
            }
        }

        dp[mask] = -1;
        return false;
    }
}
```