{1}Add to Array-Form of Integer

The array-form of an integer num is an array representing its digits in left to right order.

For example, for num = 1321, the array form is [1,3,2,1].

Given num, the array-form of an integer, and an integer k, return the array-form of the integer num + k.

Example 1:

Input: num = [1,2,0,0], k = 34

Output: [1,2,3,4]

Explanation: 1200 + 34 = 1234

Example 2:

Input: num = [2,7,4], k = 181

Output: [4,5,5]

Explanation: 274 + 181 = 455

Example 3:

Input: num = [2,1,5], k = 806

Output: [1,0,2,1]

Explanation: 215 + 806 = 1021

```
class Solution {
  public List<Integer> addToArrayForm(int[] num, int k) {
  Stack<Integer> stack = new Stack<>();
    int carry = k;
// Iterate over the digits of the input array num from right to left, adding each digit to the carry and updating the carry accordingly.
```

```
        int i = num.length - 1;
        while (i >= 0 || carry > 0) {
            if (i >= 0) {
                carry += num[i];
            }
// Compute the least significant digit of the result by taking the remainder of the sum modulo 10, and push it onto the stack.
            stack.push(carry % 10);// Divide the sum by 10 to update the carry, and repeat the process for the next digit.
            carry /= 10;
            i--;
            }
// Create a new list to store the digits of the result in the correct order, and pop the     digits from the stack one by one, adding them to the list.
        List<Integer> result = new ArrayList<>();
        while (!stack.isEmpty()) {
            result.add(stack.pop());
        }// Return the list as the final result.
        return result;
    }
}
```

{2}Divisor Game

Alice and Bob take turns playing a game, with Alice starting first.

Initially, there is a number n on the chalkboard. On each player's turn, that player makes a move consisting of:

Choosing any x with 0 < x < n and n % x == 0.

Replacing the number n on the chalkboard with n - x.

Also, if a player cannot make a move, they lose the game.

Return true if and only if Alice wins the game, assuming both players play optimally.

Example 1:

Input: n = 2

Output: true

Explanation: Alice chooses 1, and Bob has no more moves.

Example 2:

Input: n = 3

Output: false

Explanation: Alice chooses 1, Bob chooses 1, and Alice has no more moves.

```
class Solution {
    public boolean divisorGame(int n) {
        return (n%2==0)?true:false;
    }
}
```

{3}Valid Boomerang

Given an array points where points[i] = [xi, yi] represents a point on the X-Y plane, return true if these points are a boomerang.

A boomerang is a set of three points that are all distinct and not in a straight line.

Example 1:

Input: points = [[1,1],[2,3],[3,2]]

Output: true

Example 2:

Input: points = [[1,1],[2,2],[3,3]]

Output: false

```
class Solution {
  public boolean isBoomerang(int[][] points) {
      if(Math.abs((points[0][0]-points[1][0])*(points[1][1]-points[2][1])-(points[1][0]-
points[2][0])*(points[0][1]-points[1][1]))!=0)
          return true;
       return false;
   }
}
```

{4}Calculate Money in Leetcode Bank

Hercy wants to save money for his first car. He puts money in the Leetcode bank every day.

He starts by putting in $1 on Monday, the first day. Every day from Tuesday to Sunday, he will put in $1 more than the day before. On every subsequent Monday, he will put in $1 more than the previous Monday.

Given n, return the total amount of money he will have in the Leetcode bank at the end of the nth day.

Example 1:

Input: n = 4

Output: 10

Explanation: After the 4th day, the total is 1 + 2 + 3 + 4 = 10.

Example 2:

Input: n = 10

Output: 37

Explanation: After the 10th day, the total is (1 + 2 + 3 + 4 + 5 + 6 + 7) + (2 + 3 + 4) = 37. Notice that on the 2nd Monday, Hercy only puts in $2.

Example 3:

Input: n = 20

Output: 96

Explanation: After the 20th day, the total is (1 + 2 + 3 + 4 + 5 + 6 + 7) + (2 + 3 + 4 + 5 + 6 + 7 + 8) + (3 + 4 + 5 + 6 + 7 + 8) = 96.

```
class Solution {

    public int totalMoney(int n) {

        int extra = n%7;

        int weeks = n/7;

        return 28 * weeks + 7 * (weeks)*(weeks-1)/2 + weeks*extra + (extra)*(extra+1)/2;

    }

}
```

{5}Matrix Cells in Distance Order

You are given four integers row, cols, rCenter, and cCenter. There is a rows x cols matrix and you are on the cell with the coordinates (rCenter, cCenter).

Return the coordinates of all cells in the matrix, sorted by their distance from (rCenter, cCenter) from the smallest distance to the largest distance. You may return the answer in any order that satisfies this condition.

The distance between two cells (r1, c1) and (r2, c2) is |r1 - r2| + |c1 - c2|.

Example 1:

Input: rows = 1, cols = 2, rCenter = 0, cCenter = 0

Output: [[0,0],[0,1]]

Explanation: The distances from (0, 0) to other cells are: [0,1]

Example 2:

Input: rows = 2, cols = 2, rCenter = 0, cCenter = 1

Output: [[0,1],[0,0],[1,1],[1,0]]

Explanation: The distances from (0, 1) to other cells are: [0,1,1,2]

The answer [[0,1],[1,1],[0,0],[1,0]] would also be accepted as correct.

Example 3:

Input: rows = 2, cols = 3, rCenter = 1, cCenter = 2

Output: [[1,2],[0,2],[1,1],[0,1],[1,0],[0,0]]

Explanation: The distances from (1, 2) to other cells are: [0,1,1,2,2,3]

There are other answers that would also be accepted as correct, such as [[1,2],[1,1],[0,2],[1,0],[0,1],[0,0]].

```
class Solution {

    public int[][] allCellsDistOrder(int rows, int cols, int rCenter, int cCenter) {

         int a[][]=new int[rows*cols][2];//for storing coordinates

        int k=0;

        for(int i=0;i<rows;i++)

        {

            for(int j=0;j<cols;j++)

            {

                a[k][0]=i;

                a[k][1]=j;

                k+=1;

            }
        }
        Arrays.sort(a,new Comparator<int[]>()
              {
                 @Override
                 public int compare(int a[],int b[])
                 {
                    int d1=(int)Math.abs(a[0]-rCenter)+(int)Math.abs(a[1]-cCenter);//distance of first point
                    int d2=(int)Math.abs(b[0]-rCenter)+(int)Math.abs(b[1]-cCenter);//distance of second point
                    if(d1>d2)return 1;
                    else if(d1<d2)return -1;
                    else return 0;
                 }
              }
            );
        return a;
    }
}
```

{6}Greatest Common Divisor of Strings

For two strings s and t, we say "t divides s" if and only if s = t + ... + t (i.e., t is concatenated with itself one or more times).

Given two strings str1 and str2, return the largest string x such that x divides both str1 and str2.

Example 1:

Input: str1 = "ABCABC", str2 = "ABC"

Output: "ABC"

Example 2:

Input: str1 = "ABABAB", str2 = "ABAB"

Output: "AB"

Example 3:

Input: str1 = "LEET", str2 = "CODE"

Output: ""

```
class Solution {
    public String gcdOfStrings(String str1, String str2)
    {
         if (!(str1 + str2).equals(str2 + str1))
        {
            return "";
        }

        int gcd = gcd(str1.length(), str2.length());
        return str2.substring(0, gcd);
    }

    public int gcd(int a, int b)
     {
        return (b == 0)? a : gcd(b, a % b);
    }
}
```

{7}Distribute Candies to People

We distribute some number of candies, to a row of n = num_people people in the following way:

We then give 1 candy to the first person, 2 candies to the second person, and so on until we give n candies to the last person.

Then, we go back to the start of the row, giving n + 1 candies to the first person, n + 2 candies to the second person, and so on until we give 2 * n candies to the last person.

This process repeats (with us giving one more candy each time, and moving to the start of the row after we reach the end) until we run out of candies. The last person will receive all of our remaining candies (not necessarily one more than the previous gift).

Return an array (of length num_people and sum candies) that represents the final distribution of candies.

Example 1:

Input: candies = 7, num_people = 4

Output: [1,2,3,1]

Explanation:

On the first turn, ans[0] += 1, and the array is [1,0,0,0].

On the second turn, ans[1] += 2, and the array is [1,2,0,0].

On the third turn, ans[2] += 3, and the array is [1,2,3,0].

On the fourth turn, ans[3] += 1 (because there is only one candy left), and the final array is [1,2,3,1].

Example 2:

Input: candies = 10, num_people = 3

Output: [5,2,3]

Explanation:

On the first turn, ans[0] += 1, and the array is [1,0,0].

On the second turn, ans[1] += 2, and the array is [1,2,0].

On the third turn, ans[2] += 3, and the array is [1,2,3].

On the fourth turn, ans[0] += 4, and the final array is [5,2,3].

```
class Solution {
    public int[] distributeCandies(int candies, int num_people) {
         int[]ans=new int[num_people];
        Arrays.fill(ans,0);
        int cur=0;
        while(candies>0){
            for(int i=0;i<num_people;i++){
                cur++;
                if(candies>=cur){
                ans[i]+=cur;
                candies-=cur;
                }else{
                    ans[i]+=candies;
                    candies=0;
                }
            }
        }
        return ans;
    }
}
```

{8}N-th Tribonacci Number

The Tribonacci sequence Tn is defined as follows:

T0 = 0, T1 = 1, T2 = 1, and Tn+3 = Tn + Tn+1 + Tn+2 for n >= 0.

Given n, return the value of Tn.

Example 1:

Input: n = 4

Output: 4

Explanation:

T_3 = 0 + 1 + 1 = 2

T_4 = 1 + 1 + 2 = 4

Example 2:

Input: n = 25

Output: 1389537

```
class Solution
 {
public int tribonacci(int n)
{
   if (n < 2) return n;
   int a = 0, b = 1, c = 1, d;
   while (n-- > 2) {
   d = a + b + c;
   a = b;
   b = c;
   c = d;
}
return c;
}
 }
```

{9}Day of the Year

Given a string date representing a Gregorian calendar date formatted as YYYY-MM-DD, return the day number of the year.

Example 1:

Input: date = "2019-01-09"

Output: 9

Explanation: Given date is the 9th day of the year in 2019.

Example 2:

Input: date = "2019-02-10"

Output: 41

```
class Solution {
    public int dayOfYear(String date) {
        int[] days = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
        String[] dateParts = date.split("-");
        int year = Integer.parseInt(dateParts[0]);
        int month = Integer.parseInt(dateParts[1]);
        int day = Integer.parseInt(dateParts[2]);
        if (month > 2 && year % 4 == 0 && (year % 100 != 0 || year % 400 == 0)) {
            day++;
        }
        while(--month > 0) {
            day += days[month - 1];
        }
        return day;
    }
}
```

{10}Prime Arrangements

Return the number of permutations of 1 to n so that prime numbers are at prime indices (1-indexed.)

(Recall that an integer is prime if and only if it is greater than 1, and cannot be written as a product of two positive integers both smaller than it.)

Since the answer may be large, return the answer modulo 10^9 + 7.

Example 1:

Input: n = 5

Output: 12

Explanation: For example [1,2,5,4,3] is a valid permutation, but [5,2,3,4,1] is not because the prime number 5 is at index 1.

Example 2:

Input: n = 100

Output: 682289015

```
class Solution
{
    long mod=1000000007;
    public long factorial(long n)
    {
        if(n==0){return 1;}
        long result=1;
        for(long i=n;i>=1;i--)
        {
            result=(result*i)%mod;
            result=result%mod;
        }
        return result;
    }
    public long countPrimes(int n)
    {
        boolean[] primes=new boolean[n+1];
        long count=0;
        for(int i=2;i*i<=n;i++)
        {
        int j=i+i;
         while(j<=n)
         {
             primes[j]=true;
             j+=i;
         }
        }
        for(int i=2;i<=n;i++)
        {
            if(!primes[i])
            {
                count++;
            }
        }
        return count;
    }

    public int numPrimeArrangements(int n)

    {

        long tprimes=countPrimes(n);

        long pf=factorial(tprimes);

        long rem=factorial(n-tprimes);

        long r=(pf*rem)%mod;

        return (int)(r);

    }

}
```