Task 4:Number Theory – Level 1 (Includes Factorial, Fibonacci Series, Odd or Even, Sum of Digits, …)

Given an integer **N**. You have to find the number of digits that appear in its factorial, where factorial is defined as, factorial(N) = 1*2*3*4… .... *N and factorial(0) = 1.

Aim: To write a program to find factorial of the numbers.

Algorithm:

1. Read the integer N.
2. Initialize a variable called factorial to 1.
3. Initialize an array called digits of size 10 to 0.
4. Loop from 1 to N and for each iteration, multiply the current factorial with the loop index.
5. Convert the factorial to a string.
6. Loop through each character of the string representation of the factorial and for each digit: a. Convert the digit from string to integer. b. Increment the corresponding element of the digits array.
7. Count the number of non-zero elements in the digits array.
8. Print the count of non-zero elements as the result.
9. End the program

Program:

```
#include <stdio.h>

int countDigitsInFactorial(int n);

int main() {

   int n;
   printf("Enter a positive integer: ");
   scanf("%d", &n);
   int count = countDigitsInFactorial(n);
   printf("Number of digits in %d! = %d\n", n, count);
   return 0;
}

int countDigitsInFactorial(int n) {
   if (n < 0) {
      return 0;
   }
   if (n <= 1) {
      return 1;
   }
   double digits = 0;

   for (int i = 2; i <= n; i++) {
      digits += log10(i);
   }
   return (int) floor(digits) + 1;
}
```

OUTPUT:

```
Enter a positive integer: 5
Number of digits in 5! = 3
```

b) Given a number positive number N, find value of $f_0 + f_1 + f_2 + \ldots. + f_N$ where f_i indicates i'th Fibonacci number.
Remember that $f_0 = 0$, $f_1 = 1$, $f_2 = 1$, $f_3 = 2$, $f_4 = 3$, $f_5 = 5$, …
Since the answer can be very large, **answer modulo 1000000007** should be returned.

Algorithm:

1. Read the positive integer N.
2. Initialize variables f0 and f1 to 0 and 1 respectively.
3. Initialize a variable sum to f0.
4. Loop from 1 to N and for each iteration: a. Calculate the current Fibonacci number fn as f0 + f1. b. Update the values of f0 and f1 as f1 and fn respectively. c. Add the current Fibonacci number fn to sum.
5. Return sum modulo 1000000007.
6. End the program.

Program:

```
#include <stdio.h>

#define MOD 1000000007

int fibonacciSum(int n) {
    int prev = 0, curr = 1, next, sum = 0, i;
    for (i = 0; i <= n; i++) {
        sum = (sum + curr) % MOD;
        next = (prev + curr) % MOD;
        prev = curr;
        curr = next;
    }
    return sum;
}

int main() {
    int N;
    scanf("%d", &N);
    printf("%d\n", fibonacciSum(N));
    return 0;
}
```

OUTPUT

Input:

N = 3

Output:

4

Result: Thus the program is exwcuted and verified successfully

Task 5: Basic Number Theory 1

TIC TAC TOE

King Tle4Ever of Time Limit Exceeded is really fascinated about Tic Tac Toe. He organizes a national level contest for Tic Tac Toe every year in Time Limit Exceeded. (Though I agree you need to be really stupid to loose a game of Tic Tac Toe but for the sake of assume playing Tic Tac Toe for them is same as playing Chess for us :P).

Every year the contest has lots of participants. This year there are n participants from all over the country. Seeing this huge participation he asks Moron a simple .

Suppose participant pi wins wi matches. The king wants to know the sum of wi2 from 1 to n. Now as you already know Moron is not good with maths, he asks you to help him.

Given the value of n find the minimum and maximum value of sum of wi2 from 1 to n. As values can be too large output the values mod 109+7.

Aim: To write and execute the program for given scenario based on Basic Number Theory-1

Algorithm:

1. Read the integer n.
2. Calculate the minimum value of the sum as (n*(n-1)*(2*n-1))/6 modulo 109+7.
3. Calculate the maximum value of the sum as ((n*(n-1))/2 * n*n* - *((n*(n-1))/2) * (2*n-1))/3 modulo 109+7.
4. Print the minimum and maximum values of the sum as output.
5. End the program.

Program:

```
#include <stdio.h>
#include <math.h>

#define MOD 1000000007
int main() {
   int t;
   scanf("%d", &t);
   while (t--) {
      long long n;
      scanf("%lld", &n);
      long long min_sum = ((n - 1) * (n - 1)*n /4) % MOD; // Minimum sum
      long long max_sum = ((n * (n + 1)) * (2 *( n + 1))/(12)) % MOD; // Maximum sum
      printf("%lld %lld\n", min_sum, max_sum);
   }
   return 0;
}
```

OUTPUT:

No of inputs: 2

5

20 30

Problem:

Given two integers, and , a recursive technique to find their GCD is the Euclidean Algorithm.

The algorithm states that, for computing the GCD of two positive integers and , if and are equal, . Otherwise if . There are a few optimizations that can be made to the above logic to arrive at a more efficient implementation.

Algorithm:

1. Read the two positive integers a and b.
2. If b is zero, return a as the GCD.
3. Otherwise, recursively call the function with arguments b and the remainder of a divided by b.
4. Return the result of the recursive call as the GCD.

Program:

```
#include <stdio.h>
int gcd(int a, int b) {
   if (b == 0) {
      return a;
   }
   return gcd(b, a % b);
}
int main() {
   int a, b;
   scanf("%d %d", &a, &b);
   int result = gcd(a, b);
   printf("%d\n", result);
   return 0;
}
```

OUTPUT:

Sample Input

1 5

Sample Output

1

Task 6-Basic Number Theory - 2

Euclidean

Given four integers x1, y1, x2 and y2, which represents two coordinates (x1, y1) and (x2, y2) of a two-dimensional graph. The task is to find the Euclidean distance between these two points.

Euclidean distance between two points is the length of a straight line drawn between those two given points.

Examples:

Input: x1, y1 = (3, 4)
x2, y2 = (7, 7)
Output: 5
Input: x1, y1 = (3, 4)
x2, y2 = (4, 3)
Output: 1.41421

Approach: Since the Euclidean distance is nothing but the straight line distance between two given points, therefore the distance formula derived from the Pythagorean theorem can be used.

Aim: To write and exexute the program based on basic Number Theory-2

Algorithm:

1. Read the values of x1, y1, x2 and y2.
2. Calculate the difference between x2 and x1 and store it in a variable dx.
3. Calculate the difference between y2 and y1 and store it in a variable dy.
4. Calculate the square of dx and store it in a variable dx2.

5. Calculate the square of dy and store it in a variable dy2.
6. Calculate the sum of dx2 and dy2 and store it in a variable d2.
7. Calculate the square root of d2 and store it in a variable distance.
8. Print the value of distance as output.

9. End the program.

Program:

```
#include <stdio.h>
#include <math.h>

int main() {
   int x1, y1, x2, y2;
   double distance;

   printf("Enter x1 and y1: ");
   scanf("%d %d", &x1, &y1);
   printf("Enter x2 and y2: ");
   scanf("%d %d", &x2, &y2);

   distance = sqrt(pow(x2 - x1, 2) + pow(y2 - y1, 2));

   printf("The Euclidean distance between (%d, %d) and (%d, %d) is %lf\n", x1, y1, x2, y2, distance);
   return 0;
}
```

OUTPUT:

Input**:** x1, y1 = (3, 4)
x2, y2 = (7, 7)
Output**:** 5

Input**:** x1, y1 = (3, 4)
x2, y2 = (4, 3)
Output**:** 1.41421

GCD

Given an array of size N of integers with each element denoted as array[i] . In this problem we are given a parameter K and we are supposed to find the size of the largest contiguous subarray whose GCD is atleast K If there is no subarray whose GCD is atleast K , print "0".

INPUT

The first line contains 2 integers N and K the size of the array and the parameter as described in the problem statement The second line containts N space separated integers denoting the array.

OUTPUT

Print a single integer in a single line denoting the maximum contiguous subarray size whose GCD is atleast K.

Contraints

1 <= N <= 500000
1 <= array[i] , K <= 1000000

Sample Input

5 9

10 20 5 15 45

Algorithm:

1. Read the value of N and the array of integers array[] of size N.
2. Read the value of K.
3. Initialize a variable max_len to 0, which will store the length of the largest contiguous subarray whose GCD is at least K.
4. Initialize two variables, left and right, to 0, which represent the left and right indices of the current subarray.
5. Initialize a variable current_gcd to array[0], which represents the GCD of the current subarray.
6. Start a loop with variable i from 1 to N-1: a. Update the current_gcd to be the GCD of the current_gcd and array[i]. b. While the current_gcd is greater than or equal to K: i. Update max_len to be the maximum of max_len and the difference between right and left plus 1. ii. Update current_gcd to be the GCD of current_gcd and array[left]. iii. Increment left by 1. c. Increment right by 1.
7. If max_len is still 0, print "0" as there is no subarray whose GCD is at least K.
8. Otherwise, print the value of max_len as the size of the largest contiguous subarray whose GCD is at least K.
9. End the program.

Program:

```
#include <stdio.h>
int gcd(int a, int b) {
   if (b == 0) {
      return a;
   } else {

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

int main() {
   int n, k;
   scanf("%d %d", &n, &k);

   int arr[n];
   for (int i = 0; i < n; i++) {
      scanf("%d", &arr[i]);
   }

   int maxLength = 0;
   int length = 0;
   for (int i = 0; i < n; i++) {
      int currentGcd = arr[i];
      if (currentGcd >= k) {
         length = 1;
      } else {
         continue;
      }

      for (int j = i + 1; j < n; j++) {
         currentGcd = gcd(currentGcd, arr[j]);
```

```
            if (currentGcd >= k) {
                length++;
            } else {
                break;
            }
        }

        if (length > maxLength) {
            maxLength = length;
        }
    }

    printf("%d\n", maxLength);
    return 0;
}
```

OUTPUT:

Sample Input

5 9
10 20 5 15 45

Sample Output

2

Result: Thus the program is executed and verified successfully.

xecuted and verified successfully.