Recursividad I (2015-2016)

A - SuperPrimos

Problema A

```
using System;

namespace RecursividadI
{
    public class A
    {
        public static bool EsPrimo(long N)
        {
            if(N < 2) return false;
            if (N == 2 || N == 3) return true;
            if (N % 2 == 0 || N % 3 == 0) return false;
            for(long i = 5; i * i <= N; i += 2)
                if(N % i == 0) return false;

            return true;
        }

        public static bool EsSuperPrimo(long N)
        {
            if(N<10)
                return EsPrimo(N);
            return EsPrimo(N) && EsSuperPrimo(N/10);
        }

        public static void Main()
        {
            long N = long.Parse(Console.ReadLine());
            Console.WriteLine(EsSuperPrimo(N));
        }
    }
}
```

B - Buscando Números

Problema B

```
using System;

namespace RecursividadI
{
    public class B
    {
        static int BusquedaBinaria(int[] arr, int x, int l, int r)
        {
            if(l > r) return ~l;
            int m = l + (r - l)/2;
            if(arr[m] > x) return BusquedaBinaria(arr, x, l, m - 1);
            if(arr[m] < x) return BusquedaBinaria(arr, x, m + 1, r);
            return m;
        }

        public static void Main()
        {
            string[] line = Console.ReadLine().Split();
            int q = int.Parse(Console.ReadLine());
            int[] numbers = new int[line.Length];
            for(int i = 0; i < line.Length; i++)
                numbers[i] = int.Parse(line[i]);

            for(int i = 1; i <= q; i++)
            {
                int x = int.Parse(Console.ReadLine());

                int idx = BusquedaBinaria(numbers, x, 0, numbers.Length - 1);

                if(idx >= 0)
                    Console.WriteLine("Caso {0}: {1} esta en la posicion {2}",
                                      i, x, idx);
                else
                    Console.WriteLine("Caso {0}: {1} deberia estar en la posicion {2}",
                                      i , x, ~idx);
            }
        }
    }
}
```

Notar que una búsqueda secuencial daría TLE (*Time Limit Exceeded*) por lo que se usa Búsqueda Binaria que es mucho más eficiente

En esta implementación de Búsqueda Binaria si el elemento no está, la posición donde iría queda almacenada en la variable l. Para aprovecharse de esto, en esta implementación se buscó (hay otras maneras de hacerlo) una función que permita obtener l y al mismo tiempo diferenciar el caso en que está y no está el número buscado. La función seleccionada es la operación a nivel de bits (*bitwise operator*) **NOT**. Esta operación, representada en *C#* mediante el operador unario $\sim$, cumple que sea n una variable entera $\sim n = -n - 1$, o sea, siempre resulta un número negativo. De aquí que si el llamado a Búsqueda Binaria devuelve un número no negativo significa que el elemento está, por el contrario si devuelve un número negativo implica que el elemento no está y basta con aplicarle la función **NOT** a este resultado para obtener el índice donde iría.

C - MCM

$$MCM(a,b) = \frac{ab}{MCD(a,b)}$$

Problema C

```
using System;

namespace RecursividadI
{
    public class C
    {
        public static ulong MCD(ulong a, ulong b)
        {
            return b == 0? a: MCD(b, a % b);
        }
        public static void Main()
        {
            ulong A = ulong.Parse(Console.ReadLine());
            ulong B = ulong.Parse(Console.ReadLine());

            Console.WriteLine( A * ( B / MCD(A, B) ) );
        }
    }
}
```

D - Suma y Suma

Problema D

```
using System;

namespace RecursividadI
{
    public class D
    {
        static string S(string N)
        {
            long s = 0;
            for(int i = 0; i < N.Length; i++)
                s+= N[i] - '0';
            return s.ToString();
        }

        static string F(string N)
        {
            if(N.Length == 1) return N;
            return F(S(N));
        }


        public static void Main()
        {
            string N = Console.ReadLine();
            Console.WriteLine(F(N));
        }
    }
}
```

E - Ackermann

Problema E

```
using System;

namespace RecursividadI
{
    public class E
    {
        public static long Ackermann(long m, long n)
        {
            if(m == 0) return n + 1;
            if(n == 0) return Ackermann(m - 1, 1);
            return Ackermann(m - 1, Ackermann(m, n - 1));
        }
        public static void Main()
        {
            long m = long.Parse(Console.ReadLine());
            long n = long.Parse(Console.ReadLine());
            Console.WriteLine(Ackermann(m, n));
        }
    }
}
```